MANAGER'S HANDBOOK FOR SOFTWARE DEVELOPMENT

APRIL 1984

National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES SEL-84-001

MANAGER'S HANDBOOK FOR SOFTWARE DEVELOPMENT

APRIL 1984

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; **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-83/6177.

The contributors to this document include

William Agresti, Computer Sciences Corporation
Frank McGarry, Goddard Space Flight Center
David Card, Computer Sciences Corporation
Jerry Page, Computer Sciences Corporation
Victor Church, Computer Sciences Corporation
Roger Werking, Goddard Space Flight Center

Single copies of this document can be obtained by writing to

Frank E. McGarry
Code 582
NASA/GSFC
Greenbelt, MD 20771

ABSTRACT

Methods and aids for the management of software development projects are presented. The recommendations are based on analyses and experiences of the Software Engineering Laboratory (SEL) with flight dynamics software development. The management aspects of the following subjects are described: organizing the project, producing a development plan, estimating costs, scheduling, staffing, preparing deliverable documents, using management tools, monitoring the project, conducting reviews, auditing, testing, and certifying.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

SECTION 1—INTRODUCTION

This handbook is intended to be a convenient reference on software management methods and aids. The approach is to offer concise information describing

- What the methods and aids can accomplish
- When they can be applied
- How they are applied
- Where the manager can find more background or explanatory material

The management methods and aids included here are those that have proved effective in the experiences of the Software Engineering Laboratory (SEL) (Reference 1). Table 1-1 lists the characteristics of the software projects that the SEL has analyzed in the flight dynamics environment. The applications include attitude determination and control, orbit adjustment, maneuver planning, and general mission analysis.

Table 1-1. Software Development Environment

PROCESS CHARACTERISTICS	AVG.	HIGH	LOW
Duration (months)	16	21	13
Effort (staff years)	8	11	2
Size (1000 source lines of code)			
Developed	57	111	21
Delivered	62	112	33
Staff (full-time equivalent)			
Average	5	6	2
Peak	10	14	2
Individuals	14	17	7
Application Experience (years)			
Managers	6	7	5
Technical Staff	4	5	3
Overall Experience (years)			
Managers	10	14	8
Technical Staff	9	11	7

NOTES: Type of Software: Scientific, ground-based, interactive graphic.
Languages: 85 percent FORTRAN, 15 percent assembler macros.
Machines: IBM S/360 and 4341; DEC PDP and VAX.

HANDBOOK OVERVIEW

This document consists of eight sections organized by specific management topics:

Section 1 presents the handbook's purpose, organization, and intended audience and summarizes the software life cycle.

Section 2 discusses the basic management concerns of organizing and planning in the context of software management. The production of the **software development/management plan** is covered in detail.

Section 3 describes **resource estimation and allocation.** Techniques are presented for estimating size, costs, and effort. Guidelines are given for project scheduling and for staff allocation and composition.

Section 4 outlines contents, timing, and evaluation of **key documents and deliverables** in a software project.

Section 5 discusses **management tools and aids** and briefly describes some automated management tools.

Section 6 covers **monitoring and controlling** a software project that is already underway. Key indicators of progress are listed along with warning signals and corresponding corrective measures.

Section 7 presents both the general function of **project reviews** and the specific implementation of the four major reviews. Guidelines for **auditing** a project are also introduced.

Section 8 discusses the management aspects of **testing and certification.**

A **glossary, references,** and a **bibliography** of SEL literature conclude the handbook.

INTENDED AUDIENCE

The intended audience of this document is the software manager, who, as defined in this handbook, serves as either an administrative or technical manager. The positions overlap somewhat in their information needs.

The **administrative manager** has overall responsibility for developing software that meets requirements and is delivered on time and within budget. In the SEL environment, a Government Technical Officer or Assistant Technical Representative (ATR) generally serves in this capacity. Typically, this manager is not involved with the day-to-day technical supervision of the programmers and analysts who are developing the software. The administrative manager will be involved in the activities listed below; the corresponding handbook sections are listed alongside.

• Organizing the project	Section 2
• Estimating resources required	Section 3
• Estimating costs	Section 3
• Evaluating documents and deliverables	Section 4
• Monitoring progress	Section 6
• Evaluating results of reviews and audits	Section 7
• Certifying the final product	Section 8

The **technical manager** is responsible for direct supervision of the developers. The position is frequently filled by a contractor manager in the SEL environment, although, on some projects, a Government manager will fulfill this role instead. This person shares some of the activities listed for the administrative manager, especially with regard to monitoring development progress. The technical manager's activities and the corresponding handbook references are presented below.

• Producing and executing the software development/management plan	Section 2
• Estimating costs	Section 3
• Scheduling the project	Section 3
• Staffing the project	Section 3
• Directing the production of documents and deliverables	Section 4
• Using automated management aids	Section 5
• Monitoring development progress	Section 6
• Supervising technical staff	Section 6
• Preparing for reviews	Section 7

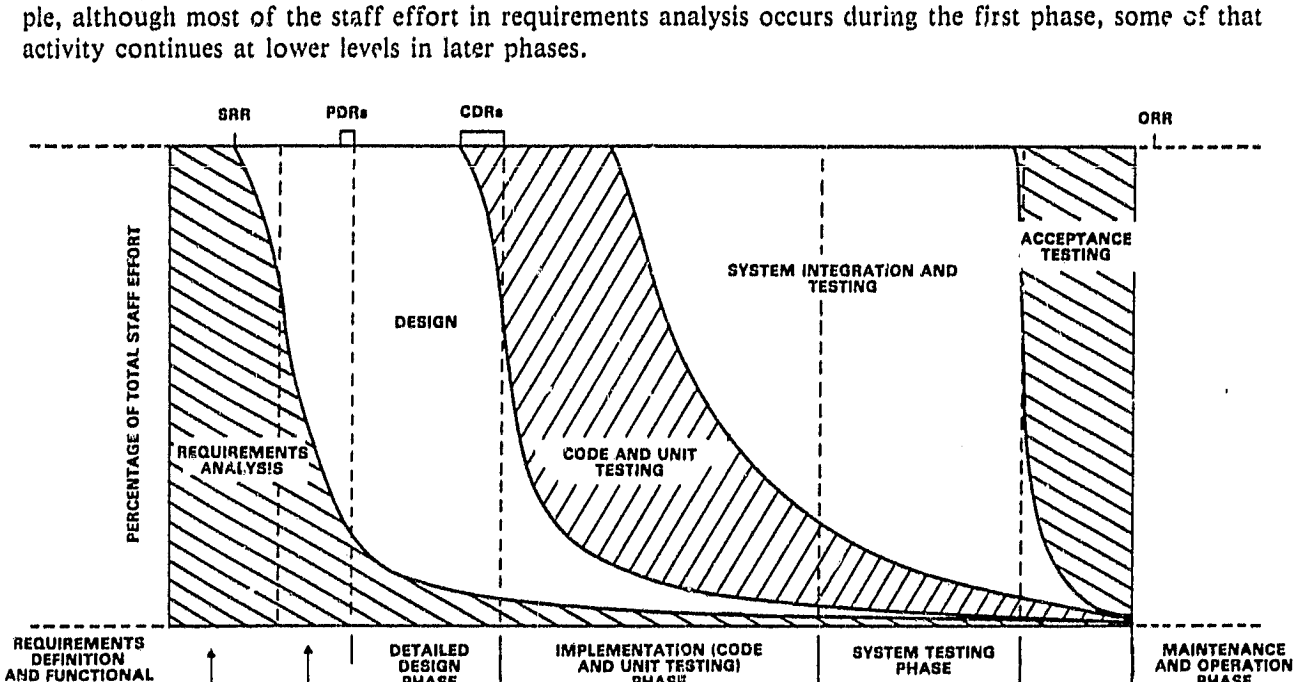
A secondary audience for the handbook consists of those who serve a particular peripheral function but do not act in either of the two managerial capacities. Two examples of such specific functions are participating as an external reviewer at a scheduled review and conducting an audit of the project.

SOFTWARE LIFE CYCLE

The process of software development involves much more than coding the problem in a programming language. It is often modeled as a series of stages that define the software life cycle. In the flight dynamics environment, the life cycle is preceded by a requirements definition and functional specification phase. The life cycle itself is defined by the following seven phases:

- Requirements analysis
- Preliminary design
- Detailed design
- Implementation (code and unit testing)
- System testing
- Acceptance testing
- Maintenance and operation

As shown in Figure 1-1, the phases divide the software life cycle into seven sequential time periods that do not overlap. However, the *activities* characteristic of one phase may be performed in other phases. For example, although most of the staff effort in requirements analysis occurs during the first phase, some of that activity continues at lower levels in later phases.



NOTE: FOR EXAMPLE, AT THE END OF THE IMPLEMENTATION PHASE (4TH DASHED LINE), APPROXIMATELY 78% OF THE STAFF ARE INVOLVED IN SYSTEM INTEGRATION AND TESTING; APPROXIMATELY 2% ARE ADDRESSING REQUIREMENTS CHANGES OR PROBLEMS; APPROXIMATELY 2% ARE DESIGNING MODIFICATIONS; AND APPROXIMATELY 17% ARE CODING AND UNIT TESTING CHANGES.

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

The life cycle phases are important reference points for the software manager. For example, in monitoring a project, the manager may find that the key indicators of project condition at one phase are not available at other phases. Milestones in the progress of a software project are keyed to the reviews, documents, and deliverables that mark the transitions between phases. Management aids and resource estimates can be applied only at certain phases because their use depends on the availability of specific information.

Although **requirements analysis** is the first phase in the SEL software development life cycle, it is preceded by the work of the requirements definition team, which completes the draft of the *functional specifications and requirements document.* During requirements analysis, the development team analyzes this document, assesses its completeness, identifies to-be-determined (TBD) requirements, specifies external interfaces, and decides on initial resource allocation. The results are given to the requirements definition team for incorporation into the final version of the functional specifications and requirements document. The conclusion of this phase is marked by the development team's preparation of the requirements analysis summary report and by the *system requirements review (SRR),* at which the completeness of the requirements is evaluated.

The functional specifications and requirements document is the starting point for the **preliminary design** activity. During this second phase, the development team defines the software system architecture and specifies the major functional subsystems, input-output interfaces, and processing modes. The *preliminary design review (PDR),* conducted at the end of this phase, provides an opportunity for evaluating the functional design presented by the development team.

In the third phase, **detailed design,** the system architecture defined during the previous phase is elaborated in successively greater detail, to the level of subroutines. The development team fully describes user input, system output, input-output files, and intermodule interfaces. An *implementation plan* is produced, describing a series of builds and releases that culminate with the delivered software system. The corresponding documentation, including complete baseline diagrams, makes up the *detailed design document.* At the *critical design review (CDR),* the detailed design is evaluated to determine if the levels of detail and completeness are sufficient for coding to begin.

During the **implementation** (code and unit testing) phase, the development team codes the required modules using the detailed design document. The system grows as new modules are coded, tested, and integrated. The developers also revise and test old code and integrate it into the evolving system. Implementation is complete when all code is integrated into the system and when supporting documents (*system test plan* and drafts of the *user's guide* and *system description*) are written.

The fifth phase, **system testing,** involves the functional testing of the end-to-end system capabilities according to the system test plan. The development team validates the completely integrated system. Successful completion of the tests required by the system test plan marks the end of this phase.

During the sixth phase, **acceptance testing,** an acceptance testing team (usually independent of the software development team) examines the completed system to determine if the original requirements have been met. Acceptance testing is complete when all tests specified in the acceptance test plan have been run successfully. An *operational readiness review (ORR)* is conducted to evaluate the system's readiness to begin operational support.

The last phase, **maintenance and operation,** begins when acceptance testing ends. The system becomes the responsibility of the maintenance and operation group. The nature and extent of activity during this phase depend on the type of software developed. For some support software, the maintenance and operation phase may be very active due to the changing needs of the users. Flight dynamics software, however, typically requires few changes during its lifetime. Because of its relatively minor role in the flight dynamics environment, the maintenance and operation phase is not covered further.

The management methods and aids in this handbook are associated with the first six phases, from requirements analysis through acceptance testing. Reference 2 contains a more detailed explanation of life cycle phases and activities.

SECTION 2—ORGANIZING AND PLANNING

The key to successful software management is to generate a realistic, usable plan and then follow it. The critical early stages of organizing and planning lay the foundation for effective project management and control.

ORGANIZING THE PROJECT

To get started, the manager must gain a clear understanding of the scope of the project and must establish the basis for control. The major initial concerns relate to clarifying the requirements, the deliverables, and the organizational framework. By addressing the four sets of questions below, the manager will be led to an understanding of the key elements that will affect project planning.

Identifying the Requirements

What functions must the system perform?
Are the boundaries of the system visible?
With what other systems will this system interact?
In what form does the job definition exist?
Is the current job definition understandable?
Does the project depend on external events or activities?

Identifying the Products and Deliverables

What documents, programs, and files are specified as deliverable products?
When must they be delivered?
In what form are the deliverables, e.g., draft copies or on tape?
Who will receive the deliverables and accept the final product?
What criteria will be used to judge the acceptability of the final product?

Preparing for Control

Is there a timetable for periodic reporting of project status?
What is the procedure for incorporating requirements changes that affect the scope of the work?
What reviews will be necessary to mark the transitions between phases?

Establishing an Organizational Identity

Who will be the key contact people from the customer, developer, and support groups?
Do the different groups understand their areas of project responsibility?
Where will the development work be done?
Which development computers will be used?
What level of access to the computers will be required?

PRODUCING THE SOFTWARE DEVELOPMENT/MANAGEMENT PLAN

In many environments, the software management plan and the software development plan are separate policy documents with different orientations. The management plan is directed toward the broader aspects of administration and control, e.g., project-level monitoring of resource expenditures and the functioning of the configuration control board. The development plan focuses more on methods and approaches to software production, e.g., testing strategies and programming methodologies. Although these differences exist between the two plans, there is generally some material in common. In fact, the situation parallels the discussion in Section 1 of the partially overlapping needs of the two intended audiences for this document, the administrative managers and technical managers.

In the flight dynamics environment of the SEL, the two plans are combined into a single document, the software development/management plan. Either approach is acceptable; organizations may find it more effective to produce two separate plans or to adopt the single-plan approach. Regardless of the choice, the critical issue is the awareness that, for a project to be successful, the items in this section must be addressed formally in a planning document or documents. Although the remainder of this section describes the contents of a single combined plan, the reader is encouraged to separate the contents into two separate plans if that is more appropriate to the needs of his/her environment.

The software development/management plan provides a disciplined approach to organizing and managing the software project. A successful plan serves as

- A structured checklist of important questions
- Consistent documentation for project organization
- A baseline reference with which to compare actual project performance and experiences
- A detailed clarification of the management approach to be used

By completing the plan early in the life cycle, the manager becomes familiar with the essential steps of organizing the development effort:

- Estimating resources
- Establishing schedules
- Assembling a staff
- Setting milestones

Writing the plan can begin as soon as any information about the project definition and scope becomes available. Producing the plan is made easier where some text (e.g., descriptions of methods, tools, and reviews) is not likely to change substantially from previous projects. The plan should be complete after the requirements analysis phase, except for information available only at later phases. If items in the software development/management plan are missing for any reason, the manager should indicate who will supply the information and when it will be supplied.

Figure 2-1 presents the suggested format and contents for the software development/management plan, including several references to sections of this handbook for detailed descriptions. The format is intended as a guide. Depending on the application environment, a different arrangement of items or the addition of new material may be appropriate.

EXECUTING THE SOFTWARE DEVELOPMENT/MANAGEMENT PLAN

The plan will be an effective management aid only to the extent that it is followed. The manager must direct and control the execution of the plan by

- Maintaining it
- Measuring progress and performance
- Recognizing danger signals
- Taking corrective action to solve problems

The last two items involve monitoring the project and are covered in Section 6.

SOFTWARE DEVELOPMENT/MANAGEMENT PLAN

TITLE PAGE—name of project, dates of this and last issue, and preparers.

LEAD SHEET—date of this issue, phase to which it is updated, list of all subsections and the pages within the subsections that have been updated since the last issue, and list of all incomplete subsections.

TABLE OF CONTENTS—list of subsection titles and page numbers

1. INTRODUCTION

1.1 Purpose—brief statement of the project's purpose.

1.2 Background—brief description that shows where the software products produced by the project fit in the overall system.

1.3 Organization and Responsibilities

1.3.1 Project Personnel—explanation and diagram of how the development team will organize activities and personnel to carry out the project: types of personnel assigned, reporting relationships, and team members' authorities and responsibilities *(see Section 3 of this handbook for guidelines on team composition).*

1.3.2 Interfacing Groups—list of interfacing groups, points of contact, and group responsibilities.

2. STATEMENT OF PROBLEM—brief elaboration of the work to be done, the steps (numbered) necessary to do it, and the relation (if any) to other projects.

3. DEVELOPMENT APPROACH—items in this section are produced for each development phase.

3.x (Each Phase)—requirements analysis, preliminary design, detailed design, implementation, system testing, and acceptance testing.

3.x.1 Technical Approach

3.x.1.1 Assumptions and Constraints—that govern the manner in which the work for this phase will be performed.

3.x.1.2 Anticipated and Unresolved Problems—that may affect the work in this and subsequent phases.

3.x.1.3 Major Activities—list of what will be done in this phase.

3.x.1.4 Methods and Tools—list of development practices and tools that the team will use and how they will use them *(see Reference 2).*

3.x.1.5 Products—list of products produced in this phase *(see Section 4).*

3.x.2 Management Approach

3.x.2.1 Concerns—that the managers feel may affect progress in this or subsequent phases.

3.x.2.2 Resource Estimates—tabular lists of estimated levels of resources required; include size estimate *(see Section 3).*

3.x.2.3 Support Needed—breakdown of type and amount of staff effort, computer time, and support effort needed for this phase; amounts expended in previous phases; and totals needed for project completion *(see Section 3).*

3.x.2.4 Assignments and Schedules—list of work to be done, who will do it, and when it will be completed.

3.x.2.5 Progress Accounting—methods used to measure and report development progress in this phase; development data that will be collected and monitored; measures that will be used to determine progress; and reports that will be produced *(see Section 6 and Reference 3).*

Figure 2-1. Software Development/Management Plan Contents (1 of 2)

3.x.2.6 **Quality Assurance**—methods used to ensure the quality of products from this phase.

3.x.2.7 **Reviews**—reviews (internal and external) conducted in this phase *(see Section 7)*.

3.x.3 **Configuration Management Procedures**—procedures used to ensure the integrity of the system configuration: when the area is under control; what documents or code are under control; how changes are requested; what forms are used; who makes the changes, etc.

• **Repeat Development Approach for Each Phase** •

4. **SUMMARY OF RESOURCES REQUIRED**—high-level summary of resources planned and expended by phase.
5. **MILESTONE CHARTS**—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. **ITEMS REQUIRED FROM CUSTOMER**—list of data, information, documents, software, hardware, and support to be supplied by the customer and delivery dates.
7. **ITEMS DELIVERED TO CUSTOMER**—list of data, information, documents, software, and support to be delivered to the customer and delivery dates.
8. **PLAN UPDATE HISTORY**—development plan lead sheets from each update indicating which sections were updated.

Figure 2-1. Software Development/Management Plan Contents (2 of 2)

One useful maintenance activity is to annotate the plan to reflect actual experiences in following the managerial and technical approaches. The information will help the organization determine which methods were effective and should be used again. As the project evolves, another maintenance function is to reestimate the size and effort measures as new indicators become available. Activities such as annotating and reestimating keep the plan current and provide information for use later in preparing the software development history (Section 4). When it is effectively maintained, the development plan documents the current state of the software development effort. By providing a uniform characterization of the project, the plan can be invaluable if changes occur in team leadership.

Significant revisions to the plan should not be considered routine maintenance. Effort should be invested when the plan is written to ensure that it is realistic, rather than continually modifying the plan so it agrees with actual decisions or experiences. Major shifts in technical approach or use of methodologies, for example, should occur only if necessary. Section 6 discusses development problems that may require revisions to the plan.

By measuring progress, the manager discovers whether the development/management plan is effective or not. The following performance data should be collected (Reference 3) to help determine whether progress is being made:

- System size, including number of subsystems, number of developed components, number of reused components, and developed lines of code
- Resources, including staff size, staff effort, computer usage, support staff effort
- Schedule, covering phase start and end dates, milestones and dates of deliverables, review dates
- Activity data, i.e., staff time spent on various activities
- Change data, including number of change report forms and number of requirements changes

The data alone are not sufficient for gauging the effectiveness of the plan, but by comparing the data to nominal values from related applications, some assessment is possible. Section 3 provides guidelines on resources and staffing that enable some comparison with the actual project data. The use of a project histories data base, as explained in Section 5, is another management aid for measuring progress.

SECTION 3—COST ESTIMATING, SCHEDULING, AND STAFFING

This section presents methods for managing and estimating the resources required for the software project. Two of the most critical resources are **development staff** and **time.** The software manager is concerned with how much time will be required to complete the project and what staffing level will be necessary over the development cycle. Both staff and time are estimated using the procedure discussed in this section. In addition, project staffing is discussed in greater detail than simply determining the total required effort. Issues of staff size and composition over the life cycle are considered. Guidelines are also given for estimating some additional important cost elements such as computer utilization and system documentation. Reference 4 provides the background and rationale for software cost estimation.

A cautionary note about the cost factors applies throughout this section. The values summarized in Table 1-1 reflect SEL experiences in the flight dynamics environment. **Readers of this handbook should assess how well that summary matches their own software development environment** as an indication of the degree of confidence to place in the particular cost values of this section. A prudent plan is to use the values here as a first approximation and begin collecting data (see Reference 3) to obtain cost factors that are representative of the reader's environment.

ESTIMATING DEVELOPMENT COST AND SCHEDULE

An understanding of the expected schedule consumption and effort expenditure in each phase of the life cycle is useful to managers. Figure 1-1 and Table 3-1 present these distributions as they reflect projects monitored by the SEL. Because the cost of developing software is often expressed in units of effort, e.g., staff-months, to avoid the effects of inflation and salary variation, cost and effort will be used interchangeably in this section when accounting for the expenditure of staff resources.

Table 3-1. Distribution of Time Schedule and Effort Over Phases

PHASE	PERCENT OF TIME SCHEDULE	PERCENT OF EFFORT
Requirements Analysis	5	6
Preliminary Design	10	8
Detailed Design	15	16
Implementation	40	45
System Testing	20	20
Acceptance Testing	10	5

Although it is the most uncertain, the initial estimate is, in many ways, the most important. It occurs at such an early stage (after the requirements definition activity) that the temptation is strong to ignore it; to do so is a mistake. Making the initial estimate has the welcome side effect of leading the manager to consider the various factors bearing on the size and complexity of the development task. The initial estimate seeds the estimation process, serving as a reference value with which to compare later estimates. In view of this singular role, the following steps are suggested for achieving an initial estimate:

- Decompose the requirements as far as possible. The decomposition unit at this point will probably be the system or subsystem.
- For each decomposition unit, identify similarities with functional units in previously developed systems and use any historical size data available from these completed systems.
- For decomposition units not strongly related to those of previous projects, use personal experience to estimate the size of units.
- Form the size estimate (in lines of code) for the entire project by adding the estimates for all the decomposition units.
- From historical data and personal experience, estimate the work rate (in lines of code per staff month).
- Divide the size estimate by the work rate to obtain an estimate of the effort in staff months.
- Apply the uncertainty proportion of 1.0 to the size and effort estimates to obtain a range of possible values.

After the initial estimate is made, five reestimates (numbered 2 through 6 in Figure 3-1) are prescribed. These reestimates exhibit a common pattern and are detailed in Table 3-2. They are based on the increasing granularity in the representation of the system during the life cycle. The uncertainties from Figure 3-1 are repeated in Table 3-2 because of their importance in transforming the individual estimates into ranges of estimated values.

The estimation factors in Table 3-2 represent average values over a wide range of development projects. The estimates should be adjusted (before the uncertainty proportion is applied) when the manager identifies certain aspects of the problem, process, or environment that vary significantly from customary development conditions. For example, any of the following conditions may strongly affect the effort necessary to complete the project: coding a new and dissimilar language, development by a completely inexperienced team, or the use of a new and dissimilar computer system.

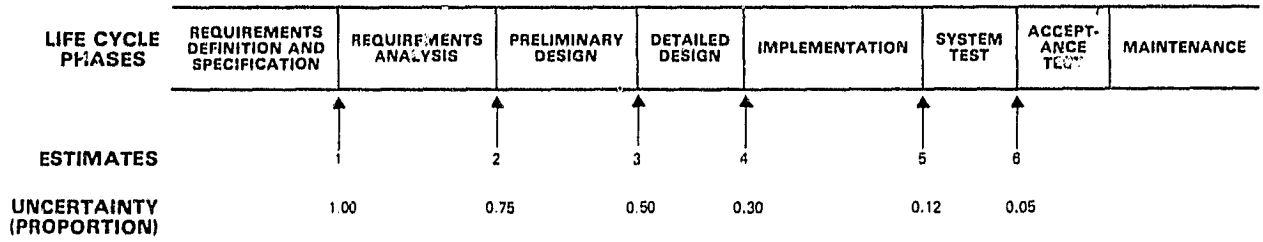


Figure 3-1. Cost Estimation Schedule

The effects of some of these conditions have been estimated by the SEL. Table 3-3 provides the recommended percentage adjustment to the effort estimate due to the complexity of the problem. Table 3-4 provides an adjustment to the effort estimate for the effect of different team experience levels.

Table 3-2. Procedures for Reestimating Size, Cost, and Schedule During Development

LIFE CYCLE PHASE[a]	DATA REQUIRED	SIZE ESTIMATE	COST (EFFORT) ESTIMATE	SCHEDULE ESTIMATE	UNCERTAINTY (PROPORTION)[b]
Requirements Analysis	Number of subsystems	Use 7500 LOC[c] per subsystem	Use 1850 hours per subsystem[e]	Use 45 weeks per subsystem per person[e]	0.75
Preliminary Design	Number of modules[d]	Use 125 LOC[c] per module	Use 30 hours per module[e]	Use 0.75 week per module per person[e]	0.50
Detailed Design	Number of new modules (N) Number of reused modules (R)	Compute: number of developed modules = N + 0.2 R Use: developed LOC = 125 x number of developed modules	Use 0.3 hour per developed LOC[e]	Use 1 week per developed module per person[e]	0.30
Implementation	Current size in LOC[c] Effort expended to date Time schedule expended to date	Add 10% to current size (for growth during testing)	Add 33% to effort already expended (for effort to complete)	Add 43% to time schedule expended (for time to complete)	0.12
System Testing	Effort expended to date	Final product size has been reached	Add 5% to effort already expended (for effort to complete)	Add 11% to time schedule expended (for time to complete)	0.05

NOTE: Parameter values are derived from experiences with the SEL software development environment summarized in Table 1-1 (an average of 85 percent FORTRAN and 15 percent assembler macros).

[a] At end of phase.

[b] Of size and effort estimates:
Upper limit = (size or effort estimate) x (1.0 + uncertainty).
Lower limit = (size or effort estimate) / (1.0 + uncertainty).

[c] Line of code: an 80-byte record that can be processed by a compiler or assembler.

[d] Module: a named subroutine unit that is independently compilable.

[e] Estimates of total effort (or time); subtract effort (or time) already expended to get effort (or time) to complete.

Table 3-3. Complexity Guideline

PROJECT TYPE[a]	ENVIRONMENT TYPE[b]	EFFORT MULTIPLIER
Old	Old	1.0
Old	New	1.4
New	Old	1.4
New	New	2.3

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

[b] Computing environment, e.g., IBM 4341, VAX-11/780. The environment type is old when the development team has more than 2 years of experience with it.

Table 3-4. Development Team Experience Guideline

TEAM YEARS OF APPLICABLE EXPERIENCE[a]	EFFORT MULTIPLIER
10	0.5
8	0.6
6	0.8
4	1.0
2	1.4
1	2.6

[a]Sum of team members' years of applicable experience weighted by a member's participation on the team.

PROJECT STAFFING

Although the level of staff is provided by the effort estimate, more specific guidelines are available for three aspects of staffing—**team size, staffing pattern,** and **team composition.** Table 3-5 presents guidelines for team size in terms of the experience of the team leaders. Table 3-6 provides guidelines for the staffing pattern: the rate at which team members can be added or released (phased in or out) without seriously affecting progress. Table 3-7 addresses team composition, listing recommended percentages of senior personnel and analysts. Reference 2 contains background information and additional guidelines on staffing decisions.

Table 3-5. 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 (requirements/specification definition, development, maintenance, and operation).
Org. = Experience with organization.
Leader = Experience as team leader or manager.

Table 3-6. 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
	5	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]Minimum interval between addition of team members to allow the project leader to maintain order.

[c]Minimum interval between departures of team members to allow assignments to be absorbed by the team and minimize callback.

Table 3-7. Team Composition 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 either the application (project type) or the computer (environment type).

OTHER SOFTWARE DEVELOPMENT COSTS

Estimates and guidelines are presented for other software cost elements: **computer utilization, system documentation, software rehosting, software reuse,** and **software maintenance.**

Cost of Computer Utilization

This cost is expressed in terms of developed lines of code, L (see Table 3-2). The estimate of total hours of CPU time, H, in an IBM 360/4341 environment is $H = 0.009L$. Figure 3-2 shows the expected computer utilization over the life cycle of the project.

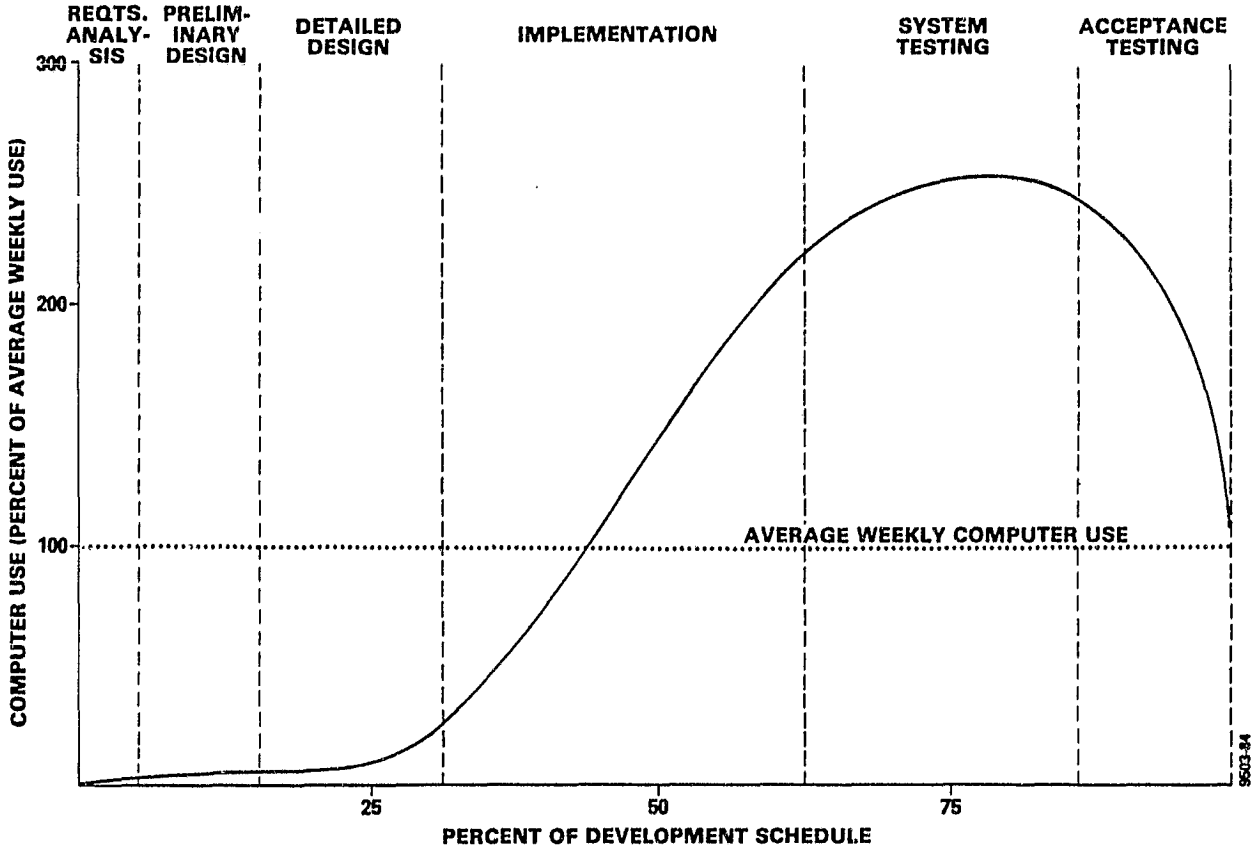


Figure 3-2. Typical Computer Utilization Profile

Cost of System Documentation

Documentation cost is included in the software development cost estimates of Table 3-2. For a separate documentation task, the expected size of the documentation is given by $P = 0.04L$, where P is pages of documentation and L is developed lines of code (Table 3-2). A cost rate of 4 staff hours per page is used to produce an estimate of the total cost of system documentation.

Cost of Rehosting Software

Rehosting means modifying existing software to operate on a new computer system. Testing will thus require a high percentage of effort. Table 3-8 provides the cost of rehosting high-level language software as a percentage of the original development cost in staff hours.

Table 3-8. Cost of Rehosting Software

SYSTEM'S RELATIONSHIP	RELATIVE COST[a]	TESTING EFFORTS[b]	NEW CODE[c]
Compatible[d]	15—21	67—70	0—3
Similar[e]	22—32	61—66	4—14
Dissimilar[f]	35—50	55—60	15—32

[a]Percent of original development cost.

[b]Percent of total rehosting cost.

[c]Percent of code that must be newly developed or extensively modified.

[d]Systems designed to be compatible (i.e., plug compatible), e.g., IBM S/360 and 4341.

[e]Similar: Some key architectural characteristics (e.g., word size) are shared and some are different (e.g., IBM S/360 and DEC VAX 11/780).

[f]Dissimilar: Differences in most characteristics of architecture and organization (e.g., IBM S/360 and PDP 11/70).

Cost of Reusing Software

Reusable modules should be identified during the design stage. As shown in Table 3-9, the estimated cost to reuse a module depends on the extent of the changes.

Table 3-9. Cost of Reusing Software

MODULE CLASSIFICATION	PERCENT OF MODULE'S CODE MODIFIED OR ADDED	RELATIVE COST[a]
New	100	100
Extensively Modified	>25	100
Slightly Modified	1—25	20
Old	0	20

[a]Cost as a percent of the cost to develop a new module.

Cost of Software Maintenance

Software maintenance refers to two types of activities occurring after the software is delivered—**correcting defects** detected during operational use and **modifying the behavior** of the software, e.g., by adding new capabilities

Expected maintenance costs vary widely, depending on the quality of the delivered software and the stability of the operational environment. From limited SEL experiences, the annual cost of error corrections and essential modifications ranges from 10 to 35 percent of the original development cost (in staff hours). This includes retesting, regenerating, and recertifying the software. New documentation is usually not produced. Little additional development (expansion of capabilities) is done with software maintained in the SEL environment.

SECTION 4—KEY DOCUMENTS AND DELIVERABLES

Documents and deliverables provide an ongoing system description and serve as key indicators of progress. They are a central concern of software managers because they mark the transitions between life cycle phases. The following documents and deliverables are of specific interest to the software manager:

- Software development/management plan
- Requirements analysis summary report
- Preliminary design report
- Detailed design document
- Test plans
- User's guide
- System description
- Software development history
- System delivery tape—Software product and supporting files plus any tools developed in connection with the project

The documents and deliverables associated with a software development project are keyed to life cycle phases. Figure 4-1 shows the phases when they should be completed. In some instances, preliminary versions are prepared, followed by updates. For any point in the life cycle, the software manager can determine what documents and deliverables should be in preparation. This section presents the recommended contents for the documents and deliverables as well as management guidelines for evaluating completed documents.

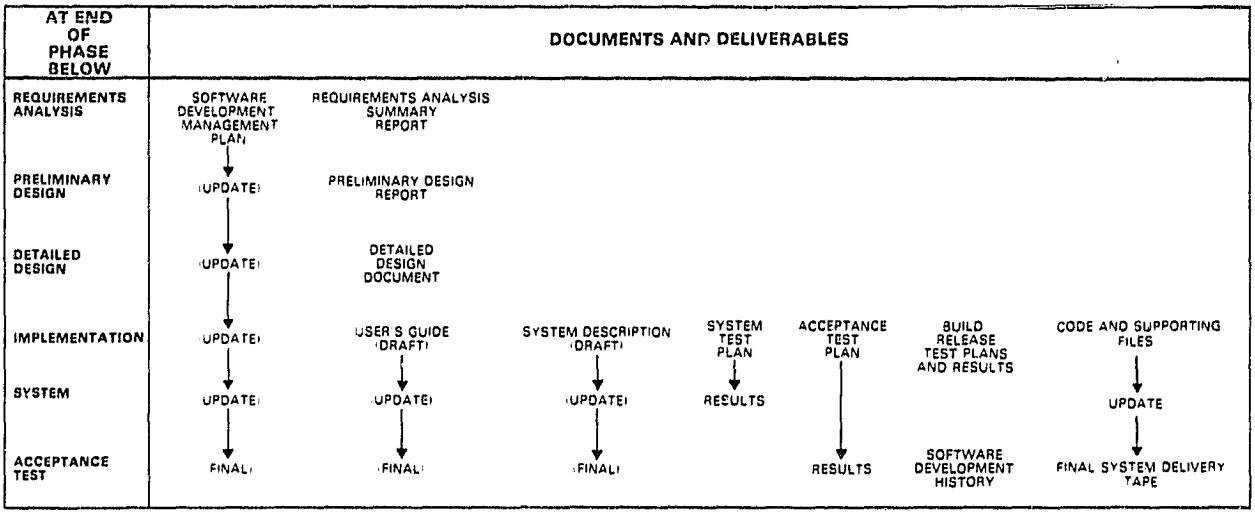


Figure 4-1. Key Documents and Deliverables by Phase

SUGGESTED CONTENTS OF DOCUMENTS AND DELIVERABLES

For each document and deliverable, a suggested format and contents are given (see Figures 4-2 though 4-9), with the exception of the software development/management plan, which was covered separately in Section 2. The actual contents of the documents may vary from the outlines presented here. Specific features of the application environment may lead the manager to exercise judgment in selecting the most appropriate material. The effectiveness of a particular document may be enhanced if some of the suggested contents are omitted and different material included. This allowance for flexibility should be understood when examining Figures 4-2 through 4-9.

REQUIREMENTS ANALYSIS SUMMARY REPORT

This report is the primary product of the requirements analysis phase. It summarizes the results of the requirements analysis and establishes a basis for beginning preliminary design. There is considerable overlap with the information required as part of the more comprehensive software development/management plan (see Section 2). The suggested contents are as follows:

1. **Introduction,** including purpose and background of project
   a. Overall system concepts
   b. Discussion and high-level diagrams of the system showing hardware interfaces, external data interfaces, and data flow
   c. Discussion and high-level diagrams of operating scenarios with interfaces and data flow
2. **System constraints**
   a. Hardware availability—execution, storage, 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 to progress
   b. List of TBD requirements and an assessment of their effect on system size, required effort, cost, and schedule
   c. List of priority areas
5. **Analysis of basic and derived requirements for the system,** including level of importance of key issues and completeness
   a. Stimulus for input—frequency, volume, coordinates, units, and timing
   b. Processing—functionality, accuracy, timing, and error handling
   c. Stimulus for output—frequency, volume, coordinates, units, and timing
6. **Analysis of basic and derived requirements for subsystems or major functions,** including level of importance of key issues and completeness. (Same as for item 5 except for subsystems or major functions.)
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**

Figure 4-2. Requirements Analysis Summary Report Contents

PRELIMINARY DESIGN REPORT

This report is the primary product of the preliminary design phase. It presents the functional description of the system and forms the basis for the detailed design document. Several items (1a, 1b, 1c, 4, 5) represent updates of similar material from the requirements analysis summary report. The suggested contents are as follows:

1. **Introduction**, including purpose and background of project
   a. Overall system concepts
   b. Discussion and high-level diagrams of system showing hardware interfaces, external data interfaces, and data flow
   c. Discussion and high-level 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 to progress
      (4) List of TBD requirements and an assessment of their effect on system size, required effort, cost, and schedule
      (5) List of priority areas
   e. Critique of alternative designs
2. **Design description** for each subsystem or major functional breakdown:
   a. Discussion and high-level 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 (tree charts) expanded to two levels below the subsystem driver
   e. Prologs (comments to describe the module's purpose, operation, calling sequence arguments, external references, etc.) and Program Design Language (PDL) for each module through the first level below subsystem driver
3. **Resource requirements**—discussion, high-level diagrams, and tables for system and subsystems
   a. Hardware
   b. Data definitions, i.e., data groupings and names
   c. Peripheral space considerations—data storage and printout
   d. Memory considerations—program storage, array storage, and 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
      (3) Layout of frames, samples, records, blocks, and/or transmissions
      (4) Storage requirements
5. **Summary of existing code that may be reused**

Figure 4-3. Preliminary Design Report Contents

DETAILED DESIGN DOCUMENT

This document is the primary product of the detailed design phase. It is most easily completed by updating similar material from the preliminary design report and adding greater detail in item 2. The suggested contents are as follows:

1. **Introduction,** including purpose and background of project
   a. Overall system concepts
   b. Discussion and high-level diagrams of system showing hardware interfaces, external data interfaces, and data flow
   c. Discussion and high-level 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 to progress
      (4) List of TBD requirements and an assessment of their effect on system size, required effort, cost, and schedule
      (5) List of priority areas

2. **Design description** for each subsystem or major functional breakdown:
   a. Overall subsystem capability
   b. Discussion and high-level diagrams of subsystem, including interfaces, data flow, and communications for each processing mode
   c. High-level description of input and output
   d. 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)
   e. Functional baseline diagrams (tree charts) 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 and PDL for each subroutine (normally kept in a separate document because of size)

3. **Resource requirements**
4. **Data interfaces**
5. **Summary of existing code**

} Updated from description in preliminary design report

Figure 4-4. Detailed Design Document Contents

TEST PLANS

BUILD/RELEASE TEST PLAN

- Prepared by the development team during the implementation phase
- Designed to test the functional capability of each build and release (functional subsets of the complete software system) identified during the detailed design phase
- Executed during the implementation phase as soon as coding and unit testing of each build and release are complete

SYSTEM TEST PLAN

- Prepared by the development team during implementation of the last release
- Designed to verify the system's end-to-end processing capability as specified in the requirements document
- Executed during the system testing phase

ACCEPTANCE TEST PLAN

- Prepared by the acceptance test team during the later stages of the implementation phase, based on information from the functional specifications and requirements document
- Designed to verify the system's acceptability to the analysts who will be its users
- Executed during the acceptance testing phase

TEST PLAN OUTLINE

1. **Introduction,** including purpose, type and level of testing, and schedule
2. **Test description** (normally the length need not exceed 1 to 2 pages) 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 the test results are acceptable
   g. Section for discussion of results, i.e., for explaining deviations from expected results and identifying the cause of the deviation or for justifying the deviation

Figure 4-5. Test Plan Contents

USER'S GUIDE

Formalization of the user's guide is started during the implementation phase, using the design document as a starting point. (Most modern approaches to software development suggest that the user's guide be completed by the end of the design phase, but experiments with this approach have yet to prove it advantageous in the flight dynamics environment.) Items 1, 2, and 4 represent updated material from the detailed design document, although some rewriting is expected to make it more accessible to the user. A typed draft is completed by the end of system testing. The user's guide is completed by the end of acceptance testing. The suggested contents are as follows:

1. **Introduction,** including purpose and background
   a. Overall system concepts
   b. Discussion and high-level diagrams of system showing hardware interfaces, external data interfaces, and data flow
   c. Discussion and high-level diagrams of operating scenarios with interfaces and data flow
2. **Description** for each subsystem or major functional breakdown:
   a. Overall subsystem capability
   b. Assumptions and restrictions to processing
   c. Discussion and high-level 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 diagrams, and tables for system and subsystems
      (1) Hardware
      (2) Data definitions, i.e., data groupings and names
      (3) Peripheral space considerations—data storage and printout
      (4) Memory considerations—program storage, array storage, and 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) Wall-clock 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

Figure 4-6. User's Guide Contents

SYSTEM DESCRIPTION

Formalization of the system description begins at system testing, using the design document as a starting point. It is completed by the end of acceptance testing. The suggested contents are as follows:

1. **Introduction,** including purpose and background of project
   a. Overall system capabilities
   b. Discussion and high-level diagrams of system showing hardware interfaces, external data interfaces, and data flow
   c. Discussion and high-level diagrams of operating scenarios with interfaces and data flow
2. **Description** for each subsystem or major functional breakdown:
   a. Overall subsystem capability
   b. Assumptions about and restrictions to processing
   c. Discussion and high-level diagrams 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 diagrams, and tables for system and subsystems
      (1) Hardware
      (2) Support data sets
      (3) Peripheral space considerations—source code storage, scratch space, and printout
      (4) Memory considerations—program generation storage and 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, for overlaying or loading
4. **Detailed description of input and output by step**—source code libraries for system and subsystems, object code libraries, execution code libraries, and auxiliary libraries, e.g., support tables
5. **Internal storage requirements**—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, computer type, length, and representation
   b. Format—organization, (e.g., indexed), transfer medium, (e.g., drum), layout of frames, (samples, records, blocks, and/or transmissions), and 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

Figure 4-7. System Description Contents

SOFTWARE DEVELOPMENT HISTORY

The software development history is completed within 3 months of software acceptance. Much of the material is available from the software development/management plan (see Section 2), which has been annotated throughout the life cycle. The suggested contents are as follows:

1. **Project description and background**
   a. Problem statement and list of key requirements—origin of requirements, purpose of system, customer of system, and development organization
   b. Key dates (actuals)—availability of functional specifications and requirements, development phase dates (start and finish), event dates, e.g., SRR, PDR, CDR, ORR
   c. Key products produced--all software and documents
   d. System characteristics
      (1) Total, new, and reused source lines of code (with and without comments) of product
      (2) Total, new, and reused number of modules of 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**—origin and timeliness; completeness and adequacy for design; change history and stability; and 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**—list of relevant background documents and reports

Figure 4-8. Software Development History Contents

SYSTEM DELIVERY TAPE

The system delivery tape contains the software product and certain supporting files. A desirable feature of the tape is that it be complete and self-documenting, so that no instruction sheets or documentation need to be consulted to generate the system. This objective can be met by a "bootstrapping" procedure: A label is attached to the tape itself, listing the job control language (JCL) commands necessary to obtain a printout of the first file on the tape. The first file then contains all of the information required to unload the remaining files on the tape and generate the system. The specific contents of a system delivery tape will vary among installations, and managers should consult standards or guidelines for their computer facility to determine the required contents. Typical contents are as follows:

FILE 1—TAPE DESCRIPTION AND INSTALLATION GUIDE

1. **Name of the system**
2. **Brief abstract of the system's function**
3. **List of files on the tape,** including
   a. File number
   b. File name
   c. Brief description of file contents
4. **Operating environment**
   a. Hardware requirements—specific models of processors and peripherals required
   b. Software requirements—specific versions of system software required
5. **System unloading procedure**—sequence of steps and specific commands required to unload the remaining files from the tape to prepare system for operational use
6. **References**—list of supporting documentation, such as system description and user's guide

FILES 2 THROUGH N—SOFTWARE SYSTEM AND SUPPORTING DATA SETS; ORGANIZED INTO THREE GROUPS:

1. **Files to generate an executable system load module**
   a. Primary source code (e.g., FORTRAN or assembler); code from more than one source language should not be in the same file
   b. JCL to compile and link-edit the source code
   c. Subroutine and macro libraries
2. **Files to execute the software system**
   a. Load module library
   b. Support libraries
   c. JCL to execute the system
   d. Permanent input and output data sets (e.g., NAMELIST data), including any initialization routines required
3. **Files to test the software system**
   a. JCL to execute the software system for each test
   b. Permanent input data sets (e.g., NAMELIST data) for each test
   c. Input data for each test
   d. Output data for each test; required if output is in the form of data sets instead of printouts

Figure 4-9. System Delivery Tape Contents

GUIDELINES FOR EVALUATING COMPLETED DOCUMENTS

The software manager will be critically reviewing completed documents. The general guidelines presented here involve checking the degree to which five basic attributes of a successful document are present in the document under review:

Accuracy—Is the document correct? Are there obvious mistakes? Are assumptions about resources and environment valid? Is there evidence of a lack of understanding of important aspects of the problem or process?

Clarity—Is the document expressed in a form that is accessible and understandable? Are tables and diagrams used where possible instead of text?

Completeness—Is the right information included, considering the purpose of the document? Have any necesary items been omitted? When the document reflects continuing development from a previous document, does it contain all the elements from the earlier document?

Consistency—Do passages in the document contradict other passages in the same document?

Level of detail—Do the contents reflect a level of detail appropriate to the purpose of the document? Is more elaboration needed in a specific area?

Using the suggested contents as a guide, the software manager can look for specific items in each key document. The following questions can be used to analyze the document for the existence and quality of essential features.

Requirements Analysis Summary Report

Are operating scenarios realistic?
Has the effect of TBD requirements been underestimated?
Is the assessment of hardware availability realistic?
Have recommended methods been used to estimate size, effort, cost, and schedule?
Are there additional sources of reused code?
Are resources sufficient?
Are all interfaces known?

Preliminary Design Report

Have all functional requirements been allocated to subsystems?
Are all interfaces understood?
Is there a rationale for the chosen design?
Is the subsystem partitioning sensible?
Will the nature of the remaining TBD requirements impede progress?

Detailed Design Document

Is the implementation plan reasonable in light of the resources?
Are baseline diagrams provided to the subroutine level?
Are all external files described in content and format (to the byte level)?
Are all TBD requirements resolved?
If the design is followed, will the system meet its requirements?
Is there evidence of information-hiding, i.e., localizing the data usage and access?
Is the coupling between modules low?
Are the modules cohesive?
Are build/release capabilities traceable to requirements?

Test Plans

Do the tests describe expected results?
Are the tests repeatable, i.e., do the test specifications adequately describe the setup and environment so that two different people would produce the same tests from the test descriptions?
How well do the tests cover the range of capabilities?
Are there explicit criteria for determining whether test results are acceptable?
Is the schedule reasonable in light of test resources?

User's Guide

Will it be understandable to the users?
Is it organized so that it can serve different user populations simultaneously?
Are examples provided?
Is input described in sufficient detail?
Are error messages and recovery explained?

System Description

Is the document structured to accommodate both those who want only a high-level view of the system and those who seek a detailed view?
Is the scope of the system clear?
Are the relationships to other systems explained?

Software Development History

Is there an update list that shows when estimates of system size, effort, schedule, and cost were made?
Have all of the problem areas been discussed?

SECTION 5—KEY MANAGEMENT AIDS

An aid or tool in the software environment is broadly considered to be any instrument that supports the software production effort. This section focuses on **management aids** in contrast to development aids such as code analyzers or preprocessors. Reference 2 describes such development aids.

Within the domain of management aids, further classification is possible. For example, the software development/management plan (Section 2), the cost estimation procedure (Section 3), and the project notebook (Reference 2) can be classified as management aids even though they are not automated. More prevalent is the position taken in this handbook—that aids or tools refer to **automated instruments.**

This section summarizes the basic capabilities of three software tools to support configuration management, project cost control, and comparison with past projects.

CONFIGURATION MANAGEMENT TOOLS

Configuration management refers to all the activities related to controlling the contents of a software system: monitoring the status of system components, preserving the integrity of released and developing versions of a system, and controlling the effects of changes throughout the system. Configuration management tools provide a central storage location for information about **milestones, documentation, changes, tests,** and **discrepancies.** Schedule data can be maintained at the detailed level of subsystems and modules. One example of such a tool is the Configuration Analysis Tool, described in detail in References 5 and 6.

PROJECT COST CONTROL

Automated tools are available to aid management by providing integrated support for project planning, scheduling, and cost control. A key feature of such tools is their ability to quantify the amount of work performed over time. Using various "earned-value" methods, equivalent units of work are awarded as progress is made on each work unit in a project. This measure of work accomplished is compared to actual costs and budgeted costs throughout the life cycle to provide a foundation for monitoring and controlling the project. The Performance Management System, described in Reference 7, is an example of this type of integrated management aid.

PROJECT HISTORIES DATA BASE

A data base of project histories is a valuable resource for the software manager. Although collecting and maintaining the data is a significant undertaking, there are benefits to be realized by creating an "organizational memory" that records key characteristics of software development in a particular environment. The costs and benefits are further discussed in References 3 and 8, respectively.

Flexible retrieval capability from such an automated data base can greatly assist the manager in monitoring the project. The guidelines in Section 6 concentrate on indicators that, by themselves, signal some project

characteristic. The added ability to refer to a baseline of data on related projects enhances the manager's capacity to assess performance and recognize problems.

Figure 5-1 suggests the possibilities for useful comparison when the project histories data base is available. Information on completed projects helps the manager initiate and revise the plans and estimates.

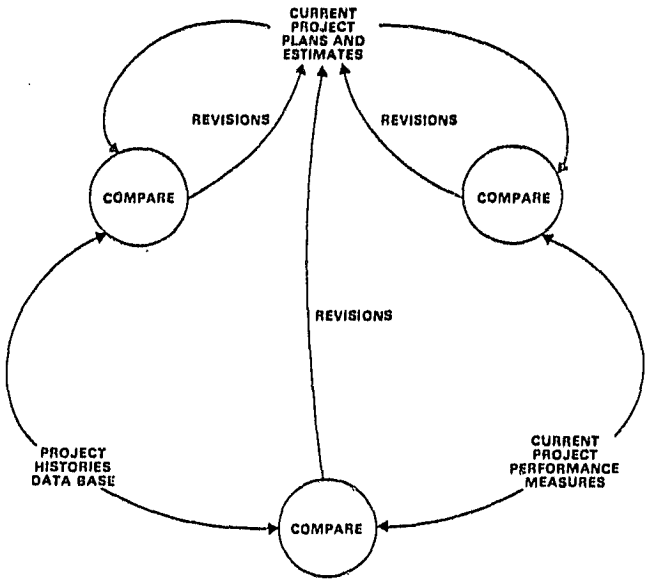


Figure 5-1. Role of Project Histories Data Base in Management Control

When actual performance measures become available on the current project, the manager can compare these values with those for related projects in the data base.

The three comparisons in Figure 5-1 can be viewed collectively as one component of a feedback and control system. The comparisons lead to revisions in development plans. To execute the revised plans, the manager makes changes in the development process, which result in adjusted measures for the next round of comparisons.

Some key items to extract from the data base are presented below. In each case, the data would correspond to the same point in the life cycle as the current project to facilitate comparison.

Key Project History Data

Resource expenditures

- Percentage of total effort expended thus far
- Distribution of effort among programmers, managers, and support staff
- Composition of team by staff level
- History of planned versus actual staffing
- Percentage of calendar time consumed thus far
- Level of computer utilization

System size

- Percentage of total software developed thus far
- Growth rate

Staff activity

- Distribution among design, code, and test activity

An example of a management aid in this area is the data base reporting software operating on the SEL data base. The software produces listings and summary reports of the contents of data base files. Reference 9 describes the use of this reporting software.

SECTION 6—KEY INDICATORS, WARNING SIGNALS, AND CORRECTIVE MEASURES

Because the active pace of the development process can easily give the appearance of progress, the project manager needs a list of characteristics that have been shown to be valuable as indicators of the true project condition. This section presents guidelines for effectively monitoring and controlling the software project. Lists of key indicators and probe points are presented. Some corrective measures for regaining control are suggested, should the indicators reveal problem areas.

KEY INDICATORS OF PROJECT STATUS

To assist the manager in monitoring the project, two groups of indicators are provided. The first consists of general features that are not associated with a particular life cycle phase. The second group is organized so that, at any phase in the life cycle, the software manager can use the list to find effective probe points.

General Indicators of Status Throughout the Project

Frequency of schedule/milestone changes

Frequency and magnitude of changes should be decreasing throughout the development process.

Consistency in organizational structure compared to original plans

Minor adjustments to the organization of the project team are expected, but major changes indicate problems.

Fluctuation in project staff level and system size estimates

Fluctuations should be within uncertainty limits that become narrower as project development evolves.

Ease of access to information on project status, schedules, and plans

Rapid responses to questions about project status and schedules reflect well on the quality of the software development plan.

Number of overtime hours required or planned to attain certain objectives

Relying on overtime hours may indicate problems with the staff's qualifications or the team leadership.

Level of detail understood and controlled by the project manager and project leader

Managers' responses to questions about development progress indicate the degree of control exercised by leadership.

Discrepancies in staff level and workload

Major differences between planned workload and actual workload may indicate lack of understanding.

Discrepancies in computer usage

A decrease or slow start in using the computer may indicate incomplete design.

Critical Items and Probe Points in Each Phase of the Life Cycle

Requirements Analysis Phase
- Number and effect of TBD requirements
- Number of requirements changes
- Number of requirements questions and answers
- Completeness of requirement summary report
- System requirements review

Preliminary Design Phase
- Number and effect of TBD requirements
- Number of requirements and design changes
- Number of interfaces per subsystem
- Completion of all preliminary design formalisms
- Preliminary design review

Detailed Design Phase
- Number and effect of TBD requirements
- Number of requirements and design changes
- Estimated number of lines of code per budgeted staff day
- Completion of all detailed design formalisms
- Build/release schedule and capabilities
- Critical design review

Implementation Phase
- Number and effect of TBD requirements
- Number of requirements and design changes
- Estimated and actual productivity rates
- Growth rate of number of lines of code and executable lines of code
- Error rate
- Number of changes to code
- Number of identified discrepancies and number of resolved discrepancies
- Detailed completion statistics for each module in system
- Review of test results

System Testing Phase
- Actual size of system versus planned size
- Actual productivity rates versus planned rates
- Error rate
- Number of changes to code
- Number of identified discrepancies and number of resolved discrepancies
- Review of test results

Acceptance Testing Phase
- Actual size of system versus planned size
- Actual productivity rates versus planned rates
- Error rate
- Number of changes to code
- Number of identified discrepancies and number of resolved discrepancies
- Review of test results

WARNING SIGNALS AND CORRECTIVE MEASURES

Two key aspects of monitoring and controlling a project are to
- Recognize projects that are in trouble (i.e., those in danger of not being completed on time)
- Take corrective action to move the project back on a successful course

The following lists of **warning signals** and **corrective measures** reflect many of the common problems identified by the SEL.

Problems With System Definition

Requirement or design is inconsistent or confusing

Difficulties become compounded if development is permitted to continue. Stop development activity and resolve inconsistency or confusion in consultation with the using organization. Negotiate a reduction in the scope of the system by defining an understandable subset of the original system.

Requirement or design is incomplete

Although the obvious correction is to resolve the incompleteness, this is not always possible. In such cases, development must continue despite the incompleteness. Assess the effect of missing requirements. Determine whether relatively safe assumptions about the missing requirements can be made. Before starting the next phase, prepare contingency plans to account for incorrect assumptions.

Problems With Development Plan

Capabilities originally planned for one time period are moved to a later time period

If a corresponding move of later capabilities to an earlier time period has not been made, the danger is that the development team will not be able to handle the additional work in the later period. Obtain justification for the change with detailed schedule information for the the new and old plans. If the shift of capabilities is extensive, stop development activity until the development/management plan can be revised, then proceed.

Change or decrease in planned use of methods or procedures occurs

The methods or procedures had some use or expected benefit or they would not have been included in the development plan. Obtain justification for the change, to include showing how the expected benefit from the planned use of the method will be realized in light of the change.

Requests are made to delete requirements

The motivation for the request must be determined: Are the resources inadequate? Are the requirements not understood? Based on the reasons for the deletion request, either revise the development plan to show new assignments, schedule, and effort or consult with customer to resolve a misunderstanding about requirements.

Problems With System Growth

Actual number of subroutines designed is fewer than estimated at a particular point in the detailed design phase

Lack of design growth may be due to poor direction from the team leader, inexperienced staff, or changing requirements. Determine the cause of the slow growth. Based on the cause, either replace junior personnel with senior personnel, decrease staff size to a manageable level, or decrease the scope of the system.

Actual number of subroutine units tested and integrated is fewer than those estimated at a particular point in the implementation phase

Lack of code growth may be due to poor direction from the team leader, inexperienced staff, changing requirements, or incomplete design. Determine the cause of the slow growth. Based on the cause, either replace junior personnel with senior personnel, stop staff growth, or hold changes and complete implementation of a build first.

Some aspect of a supposedly completed development is missing

The necessary level of completeness may not be understood, especially among junior staff. Revise status-reporting procedure to provide necessary level of detail and to accommodate accuracy.

Problems With Development Schedules

Continual schedule slippage

Staff ability may have been misjudged or the staff needs firmer direction. Replace junior-level personnel with senior-level personnel, but only if rapid phase-in is possible. Decrease the scope of the system.

"Miracle Finish": software completed on time but testing phase was significantly compressed

Testing may not have been as complete or as thorough as necessary. Review test plans and results closely; schedule additional testing if indicated.

Problems With System Configuration

More than one person controls the configuration

Sharing of configuration control responsibilities between the team leader and the manager can lead to wasted effort and the use of wrong versions for testing. Select one person who will control the configuration, review proposed changes, and issue documentation updates about the system.

"Corrected" errors reappear

The corrected version may not have been used because more than one person controlled the configuration, or the staff was not aware of the ripple effect of other changes that should have been made when the original error was corrected. Consolidate configuration control responsibility in one person. Assign more senior staff to analyze the effect of error corrections and other changes.

Basic Set of Corrective Measures

Some consistent themes appear in the lists of corrective measures, regardless of problem area. These recommendations constitute the SEL's basic approach to regaining control of the project when danger signals arise.

- 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

SECTION 7—REVIEWS AND AUDITS

Reviews and audits are methods for assessing the condition of the project. Although both techniques address quality assurance by examining the plans, methods, and intermediate products associated with the development process, they are conducted for different reasons. Reviews are routinely scheduled as part of the development process, and they mark key phase transitions in the software life cycle. Audits are generally not predetermined, but are conducted when needed to evaluate the project's status.

REVIEWS

Reviews are part of the development process, designed to provide regularly scheduled monitoring of project status. The following four questions can serve as general guidelines, regardless of the type of review:

Is the development plan being followed?
Is the project making satisfactory progress?
Are there indications of future problems?
Is the team prepared to proceed with the next phase of development?

Reviews may be characterized in various ways, such as formality or timing. An informal review may be held to brief higher level managers on the current state of the project. A formal review generally involves a more detailed presentation and discussion and follows a prescribed agenda. Some reviews may resemble progress reports delivered at fixed intervals, e.g., weekly or monthly.

In the SEL environment, four formal reviews, which are scheduled at key transition points between life cycle phases, are recommended (see Figure 7-1)—system requirements review (**SRR**),preliminary design review (**PDR**), critical design review (**CDR**), and, operational readiness review (**ORR**).

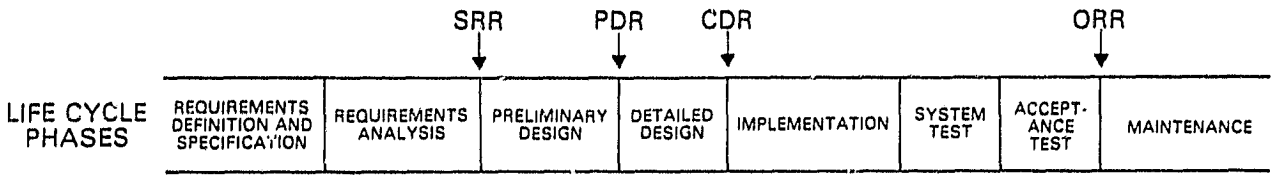


Figure 7-1. Scheduling of Formal Reviews

The remainder of this section examines the four reviews in order of occurrence and describes the **format of the review** (presenters, participants, and agenda), **key issues** to be addressed at the review (in addition to the general questions above), and **hardcopy material** (outline and suggested contents). The hardcopy material will contain some of the same information found in the documents described in Section 4. For example, when preparing the hardcopy material for the preliminary design review, some of the contents from the completed preliminary design report can be used. The manager should also keep in mind that, as with the documents in Section 4, there is some flexibility in selecting the most appropriate information to include in the hardcopy material. The contents suggested in this section are intended as a guideline.

SYSTEM REQUIREMENTS REVIEW

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

Agenda—selective presentation of information from the hardcopy material, omitting details (e.g., in items 9 and 11) that are more effectively communicated in hardcopy form and highlighting critical issues (e.g., item 16 on TBD items)

Hardcopy distribution—minimum of 5 days before SRR

Key Issues To Be Addressed

Have all elements of the hardcopy material been completed?
What is the effect of the TBD items?
What timetable has been established for resolving TBD items?
How satisfactory are the responses to questions raised at the SRR?
Is all external input and output identified?
Are constraints on time, memory, and accuracy understood?
Is the project feasible, given the constraints on and assumptions about available resources?
Is the foundation adequate to begin preliminary design?
Does the functional specifications and requirements document (FSRD) provide a basis for defining acceptance tests?

SRR Hardcopy Material

An outline and suggested contents of the SRR hardcopy material are presented in Figure 7-2. For items 13 through 17, the software development/management plan (Figure 2-1) may serve as the hardcopy material.

1. **Introduction**—purpose of system, background of project, and outline of review material
2. **Requirements summary**—review of top-level (basic) requirements developed to form the functional specifications
   a. Background of requirements—overview of project characteristics and major events
   b. Derivation of requirements—identification of input from project office, support organization, and system engineering organization used to formulate the requirements—support instrumentation requirements document (SIRD), memorandums of information (MOIs), and memorandums of understanding (MOUs)
   c. Relationship of requirements to level of support provided—typical support, critical support, and special or contingency support
   d. Organizations that provide system and support input and receive system output
   e. Data availability—frequency, volume, and format
   f. Facilities—target computing hardware and environment characteristics
   g. Requirements for computer storage, graphics, failure/recovery, operator interaction, system error recovery, and diagnostic output
   h. Requirements for support and test software—data simulators, test programs, and utilities
   i. Overview of FSRD—its evolution, including draft dates and reviews, and outline of contents
3. **Analysis overview**—analysis approach, degree of innovation required in analysis, special studies, and results
4. **Environmental considerations**—special computing capabilities, e.g., graphics; operating system limitations; computer facility operating procedures and policies; support software limitations; data base constraints; resource limitations; etc.
5. **Performance requirements**—system processing speed, system response time, system failure recovery time, and output data availability
6. **Interface requirements**—summary of human, special-purpose hardware, and automated system interfaces, including references to interface agreement documents (IADs) and interface control documents (ICDs)
7. **Derived system requirements**—list of those requirements **not explicity** called out in the requirements document but representing constraints, limitations, or implications that must be satisfied to achieve the explicitly stated requirements
8. **Operational requirements**—high-level diagrams of operating scenarios showing intended system behavior
9. **Utility, support, and test programs**—list of auxiliary software required to support development, e.g., data simulators, special test programs, software tools, etc.
10. **Reusable software summary**—identification of existing software components that satisfy specific system functional specifications exactly or that will satisfy them after specified modifications
11. **Data set definitions**—for interfaces to the system
12. **Functional specifications**—high-level data flow diagrams showing input, transforming processes, and output
13. **Requirements management approach**
    a. Personnel assignments
    b. Description of required documents
    c. Specifications/requirements change control procedures
    d. System enhancement/maintenance procedures
14. **Personnel organization and interfaces**
15. **Testing requirements**
16. **Issues, TBD items, and problems**—a characterization of all those items 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
17. **Milestones and suggested development schedule**

Figure 7-2. SRR Hardcopy Material

PRELIMINARY DESIGN REVIEW

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

Agenda—selective presentation of information from the hardcopy material, omitting details (e.g., in items 5 and 8) that are more effectively communicated in hardcopy form and highlighting critical issues (e.g., items 3, 4, 10, and 16)

Hardcopy distribution—minimum of 5 days before PDR

Key Issues To Be Addressed

Have alternative design approaches been examined?
Are all requirements traceable to subsystems in the functional design?
Is the subsystem partitioning sensible in view of the required processing?
Are all interface descriptions complete at both the system and subsystem level?
Are the layouts for all external data sets completely specified?
Is the error handling and recovery strategy comprehensive?
Is the estimate of resources realistic?
Is the schedule reasonable?
Has the effect of any remaining TBD requirements been assessed?
Has the design been elaborated in baseline diagrams to a sufficient level of detail? (Reference 2 presents information on level of detail.)
Does the design facilitate testing?

PDR Hardcopy Material

An outline and suggested contents of the PDR hardcopy material are presented in Figure 7-3. The information in items 13 through 17 is to be available to all reviewers but may be in the form of the software development/management plan. The information may be identical to that presented at the SRR, but modifications will most likely have occurred.

1. **Introduction**—purpose of the system and outline of review material
   a. Requirements summary—origin and format of requirements and a list of major system components, including the top-level (basic) requirements they satisfy
   b. Operating scenario requirements—data handling, execution frequency, etc.
   c. Environment considerations—target computing machine, operating system, etc.
   d. Software legacy (past experiences and history)—cost, schedule, and design experience
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, and interfaces to show how requirements are met
4. **High-level diagrams of system structure**—internal and external data, hardware interfaces, etc.
5. **Critique of alternative designs or approaches**
6. **Major software components**—each subsystem, major functional breakdown (in each processing mode)
   a. High-level diagrams of subsystems—internal and external data, hardware interfaces, etc.
   b. High-level input and output specifications, including frequency and volume
   c. Functional baseline diagrams (tree charts) expanded to two levels below the subsystem driver, showing interfaces, data flow, 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**—format, file structure, storage requirements
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, i.e., deterrents to progress
    d. List of TBD requirements and an assessment of their effect on system size, required effort, cost, and schedule
    e. List of priority areas
11. **Estimates of system size, required effort, cost, schedule, and staffing plan**
12. **Resource allocation and external support**
    a. Summary of how system functions will be performed—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 approach**
14. **Personnel organization and interfaces**
15. **Testing strategy**
16. **Issues, TBD items, and problems**
17. **Milestones and schedules**

Figure 7-3. PDR Hardcopy Material

CRITICAL DESIGN REVIEW

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

Agenda—selective presentation of information from the hardcopy material, omitting details (e.g., in items 5f, 5g, and 7) that are more effectively communicated in hardcopy form and highlighting critical issues (e.g., items 3, 4, 9, and 16)

Hardcopy distribution—minimum of 5 days before CDR

Key Issues To Be Addressed

Are all baseline diagrams complete to the subroutine level?
Are all interfaces—external and internal—completely specified at the subroutine level?
Is there PDL or equivalent representation for each subroutine?
Will an implementation of this design provide all the required functions?
Does the build/release schedule provide for early testing of end-to-end system capabilities?

CDR Hardcopy Material

An outline and suggested contents of the CDR hardcopy material are presented in Figure 7-4. The information in items 13 through 17 is to be available to all reviewers but may be in the form of the software development/management plan. The information may be identical to that presented at the PDR, but modifications will most likely have occurred.

1. **Introduction**—purpose of the system and outline of review material (updated from PDR material)
2. **Design overview** (updated from PDR material)
3. **High-level diagrams of operating scenarios** (updated from PDR material)
4. **High-level diagrams of system structure** (updated from PDR material)
5. **Major software components**—for each subsystem or major functional breakdown (in each processing mode):
   a— e. (Updated from corresponding PDR material)
   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** (updated from PDR material)
7. **Internal data sets** (updated from PDR material)
8. **Summary of existing code that may be reused** (updated from PDR material)
9. **Design team assessment** (updated from PDR material)
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** (updated from PDR material)
12. **Resource allocation and external support** (updated from PDR material)
13. **Development management plan**
14. **Personnel organization and interfaces**
15. **Testing strategy**
16. **Issues, TBD items, and problems**
17. **Milestones and schedules**

Figure 7-4. CDR Hardcopy Material

OPERATIONAL READINESS REVIEW

ORR Format

Presenters—operations and support team and development 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

Agenda—selective presentation of information from the hardcopy material, omitting details that are more effectively communicated in hardcopy form and highlighting critical issues (e.g., items 7 and 8)

Hardcopy distribution—minimum of 5 days before ORR

Key Issues To Be Addressed

What is the status of required system documentation?
What is the status of external interface agreements?
Were the methods used in acceptance testing adequate for verifying that all requirements have been met?
What is the status of the operations and support plan?
Is there sufficient access to necessary support hardware and software?
Are configuration control procedures established?
What contingency plans to provide operational support have been addressed?

ORR Hardcopy Material

An outline and suggested contents of the CDR hardcopy material are presented in Figure 7-5.

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. Derived requirements (updated from SRR)
   c. Relationship of requirements to support provided—typical, critical, and special or contingency support
   d. Operational support scenarios
   e. Relationship of requirements matrix, e.g., of top-level requirements to operational support scenarios
   f. Organizational interfaces, e.g., that provide system and support input and receive system output
   g. Data availability for the operating scenarios—frequency, volume, and format
   h. Facilities—computing hardware, environment characteristics, communications protocols, etc.
   i. General system considerations—high-level description of requirements for computer storage, graphics, and failure/recovery; operator interaction; system error recovery and diagnostic output, etc.
   j. Support and test software considerations—high-level description of requirements for data simulators, test programs, and support utilities
3. **Summary and status of IADs**—status of all interface documents with external organizations
4. **Support system overview**
   a. Major software components—purpose, general characteristics, and operating scenarios supported by programs and subsystems
   b. Testing philosophy
   c. Requirements verification philosophy—demonstration of methods used to ensure that the software satisfies all system requirements; matrix showing relation between requirements and tests
   d. Testing and performance evaluation results—summary of system integration and acceptance test results; evaluation of system performance measured against performance 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. **Status of operations and support plan**
   a. Organizational interfaces—diagrams and tables indicating organizational interfaces, points of contact, and responsibilities; data flow and medium (forms, tapes, voice, log)
   b. Data availability—nominal schedule of input and output data by type, format, frequency, volume, response time, turnaround time
   c. Facilities—nominal schedule of access to computers, support and special-purpose hardware, operating systems, and support software for normal, critical, emergency, and contingency operations
   d. Operating scenarios—top-level procedures, processing timelines, and estimated resources required
   e. Documentation of operations procedures—operations and support handbooks and update procedures
6. **System management plan**
   a. Configuration control procedures—explanation of step-by-step procedures for maintaining system integrity, recovering from loss, fixing faults, and enhancing system
   b. Enhancement/maintenance procedures—initiation, forms, reviews, approval, and authorization
   c. Reporting/testing evaluation procedures—forms, reviews, approval, authorization, distribution
   d. System performance evaluation procedures—for ongoing evaluation
7. **Issues, TBD items, and problems**—a characterization of all those items affecting 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
8. **Contingency plans**—a priority list of items 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
9. **Milestones and timeline of events**—diagrams, tables, and scripts of events; operating scenarios; maintenance; enhancement; reviews; training

Figure 7-5. ORR Hardcopy Material

AUDITS

The purpose of an audit is to provide an independent assessment of the software project—its condition, its problems, and its likelihood of reaching successful completion. The audit may be prompted by indications of problems or by lack of progress, or it may be fulfilling a routine contractual requirement.

An individual or, preferably, a group outside the development team is charged with conducting this examination. It is essential for the audit team to obtain a clear written statement of the specific objective of the audit at the start.

When preparing to conduct an audit, several key questions must be addressed:

What is the scope of the audit? Is the entire development effort being examined, or only some particular component of the project?

What is the final form the audit should take—a presentation, a written report, or both?

To whom will the results be presented?

What is the timetable for completing the audit?

What staff and support resources will be available for the audit work?

Have the development team and its management been informed that an audit is scheduled?

Have specific individuals on the development team been identified as principal contacts for the audit group?

What constraints exist on the work of the audit team regarding access to documents or individuals?

Where are the sources for documentation related to the project (requirements statements, plans, etc.)?

Are there specific auditing standards or guidelines that must be observed?

The answers to these questions will form the basis for planning the audit task. Sources of information include personal interviews with managers, team members, and individuals who interact with the development team. Documentation must be reviewed to understand the origin of the requirements and the plans and intermediate products of the development team.

Four steps are involved in conducting the audit of a software development project:

- Determine the current status of the project
- Determine whether the development process is under control
- Identify key items that are endangering successful completion
- Identify specific actions to put the project onto a successful course

AUDIT STEP NO. 1 — DETERMINE THE CURRENT STATUS OF THE PROJECT

Audit Question:	Audit Team's Checklist:
Given the size and nature of the problem, where should the project be?	• Consult Table 3-1 and project histories data base (see Section 5) for comparison data on similar projects: — Percentage of effort and calendar time consumed thus far — Percentage of effort by type of activity — Percentage of code developed thus far
According to the development/management plan, where should the project be?	• Refer to the software development/management plan for the project: — What activities should be current — How should it be staffed — What intermediate products should have been delivered — What reviews or milestones should have occurred
Where does the project actually stand now?	• From interviews and documentation, identify the following: — The current phase — Activity levels — Staff composition — Documents delivered — Milestones reached — Project budget — Actual expenditures — Results of reviews

AUDIT STEP NO. 2 — DETERMINE WHETHER THE DEVELOPMENT PROCESS IS UNDER CONTROL

Audit Question:	Audit Team's Checklist:
Is the problem well understood and stable?	• Refer to project documents for significance of TBD items — Requirements analysis summary report (see Section 4) — Hardcopy material from SRR (see Section 7) • Interview analysts who attended SRR to determine adequacy of responses to questions raised • From technical manager, identify the number and extent of specification modifications received by the development team
Is the development/management plan being followed?	• Examine the development/management plan • Compare the plan to actual development to determine whether — The schedule is being followed — Milestones have been met — The plan is being annotated (see Section 2) — Actual and planned staffing levels agree
Is adequate direction being provided?	• Interview team leaders, technical managers, and administrative managers to determine whether there is agreement on — Project scope and objectives — Expectations and responsibilities at each level — Methods for measuring progress

AUDIT STEP NO. 3 — IDENTIFY KEY ITEMS THAT ARE ENDANGERING SUCCESSFUL COMPLETION

Audit Question:	Audit Team's Checklist:
Are resources adequate?	• Time—determine if lack of schedule time (regardless of staff) is a concern by comparison to past projects (Section 5) and by time estimates (Section 3) • Staff—compare actual with desired staffing characteristics — Level of staff effort (Section 3) — Team size (Table 3-5) — Staffing pattern (Table 3-6) — Composition (Table 3-7) • Computer—compare actual with expected utilization (Figure 3-2); from interviews and computer facility schedules, determine degree of access to computer and level of service provided • Support—compare actual level of support services to typical levels
Is the development process adequate?	• Determine whether technical standards and guidelines are being followed for design, coding, and testing • Determine whether available tools and methodologies are being used • From interviews, determine the procedures for reporting and resolving problems
Are the organization and planning adequate?	• From interviews, assess the reporting relationships that exist, the team morale and turnover, and the pattern of delegating work • Assess the quality, completeness, and practicality of the software development/management plan (see Section 2) • From interviews and documentation, identify extent of contingency planning

AUDIT STEP NO. 4 — IDENTIFY SPECIFIC ACTIONS TO PUT THE PROJECT ON A SUCCESSFUL COURSE

- Recommended actions must be based on results of audit steps 1, 2, and 3
- For general problem of inadequate progress, some options are as follows
  - — Stop development; generate a realistic plan before resuming
  - — Replace junior personnel with senior staff
  - — Increase visibility by improving identification and review of intermediate products

SECTION 8—TESTING AND CERTIFICATION

Both testing and certification are approaches to ensuring quality in the delivered software. Testing identifies defects so the software can be repaired before it is released. Certification subjects the product and process to independent inspection and evaluation.

TESTING

A summary of essential management guidelines on software testing is presented below. The observations about the planning and control of testing are derived from SEL experience. For comprehensive presentations of the art and science of testing, References 10, 11, and 12 may be consulted.

Realize that testing is important—thirty percent of the life-cycle effort in the flight dynamics environment

Apply adequate resources

- Time—thirty percent of the time schedule
- Staff—experienced, well-trained in defect detection
- Computer—peak use in testing phases (see Figure 3-2)

Plan for it early—as part of the software development/management plan

Plan for it explicitly—using formatted test plans (see Section 4)

Test continually during the life cycle with four major types of testing (Reference 2)—**unit, build/release, system integration,** and **acceptance**

Prepare for testing—use testability as a criterion for evaluating requirements statements, designs, and build/release plans

Apply testing aids (Reference 2)

- Decision tables and test summary tables
- Test library

Monitor testing costs (Reference 3)—collect data on

- Cost of diagnosis—finding the defect
- Cost of repair—making all the necessary corrections to code and documentation

Measure testing progress

- Compare testing costs and number of defects with past projects
- Record defect detection rate as testing effort is applied

CERTIFICATION

Broadly defined, certification is a statement attesting to something. For example, an individual may be charged with certifying that

- Specific test cases were run
- Coding standards were followed
- All contractual items have been delivered
- Configuration management procedures have been followed
- Code agrees with PDL

Although there is considerable diversity in what is being certified, there are common aspects as well. Certification is a binary decision—either the materials or activities are certified or they are not. It is performed by individuals who can be objective about their certification assignment. This objectivity is a key reason for introducing certification into the software development process. Certification contributes to quality assurance by providing an independent check on development. Confidence in the final software product is enhanced if both the process and product are certified. Essential management guidelines for certification are summarized below.

Determine the objective of the certification, e.g., to ensure that

- Design, code, or documentation is correct
- Standards, procedures, or guidelines are being followed
- System performance meets operational requirements

Define entry criteria—what materials must be submitted for certification?

- Establish procedures for obtaining documents or code that will be required
- Introduce data collection forms, if needed, to capture required information about the development process

Define exit criteria—certification is a binary decision. How will submitted materials be evaluated to make the certification decision?

Specify the certification procedure, document it, and follow it

More detailed recommendations depend on the object of certification. For example, in certifying intermediate products like designs, test plans, or unit code, the materials submitted for certification and the evaluation criteria will vary. Figure 8-1 shows the general guidelines applied to an example unit code certification.

1. **Certification Objectives:**
   a. Inspect the code for consistency with its design, as specified in its prolog and PDL
   b. Check for conformance with standards and conventions regarding source code, prolog, and PDL
   c. Identify visible weaknesses in the code and unit testing strategy early so that defects can be repaired at the local unit level
   d. Encourage programmers to plan unit testing and to complete unit code by adding prolog and PDL
2. **Entry Criteria**—the following materials must be submitted:
   a. Source code listing, complete with prolog, PDL, and comments
   b. List of test cases or test summary table (Reference 12)
   c. Code certification form, containing author's name, date submitted for certification, module name, build/release/subsystem name, and test environment (names of calling units, test files, stub units—i.e., called units implemented as procedure stubs to facilitate testing)
3. **Exit Criteria**—some questions will be specific to the coding language used. The following are more general questions, typically used to evaluate submitted materials and decide on certification:
   a. Do the code, prolog, and PDL adhere to all prevailing standards and conventions?
   b. Are all necessary elements of the prolog complete, e.g., are all data elements described?
   c. Is the code consistent with its design, as presented in its prolog and PDL?
   d. Does the code appear to be correct for test cases that can be verified by inspection?
   e. Is all debug code clearly identified?
   f. Are all stubs and test files identified?
   g. Do test cases appear adequate, based on the PDL?
4. **Certification Procedure**—recommended steps for the certification:
   a. Meet with development team leader to establish position of unit code certification in development process; all code must pass through certification before the system test and integration phase
   b. Issue written descriptions of entry criteria, with examples of the required form for submitted materials
   c. Design a unit code certification checklist, based on exit criteria, to record evaluation outcome
   d. Document the procedure for obtaining materials, completing the certification checklist, and presenting results to the development team
   e. Implement the procedure; retain certification results for later sharing with development team to identify areas for improvement

Figure 8-1. Example of Unit Code Certification

GLOSSARY

ATR	Assistant Technical Representative
CDR	Critical Design Review
CPU	Central Processing Unit
FSRD	Functional Specifications and Requirements Document
IAD	Interface Agreement Document
ICD	Interface Control Document
I/O	input/output
LOC	lines of code
MOI	Memorandum of Information
MOU	Memorandum of Understanding
ORR	Operational Readiness Review
PDL	Program Design Language (pseudocode)
PDR	Preliminary Design Review
SEL	Software Engineering Laboratory
SIRD	Support Instrumentation Requirements Document
SRR	System Requirements Review
TBD	to be determined

REFERENCES

1. Software Engineering Laboratory, SEL-81-104, *The Software Engineering Laboratory,* D. N. Card et al., February 1982
2. —, SEL-81-205, *Recommended Approach to Software Development,* F. E. McGarry, G. Page, S. Eslinger, et al., April 1983
3. —, SEL-81-101, *Guide to Data Collection,* V. E. Church, D. N. Card, and F. E. McGarry, August 1982
4. —, SEL-83-001, *An Approach to Software Cost Estimation,* F. E. McGarry, G. Page, D. N. Card, et al., February 1984
5. —, SEL-81-007, *Software Engineering Laboratory (SEL) Compendium of Tools,* W. J. Decker et al., February 1981
6. —, SEL-80-104, *Configuration Analysis Tool (CAT) System Description and User's Guide (Revision 1),* W. J. Decker and W. Taylor, December 1982
7. Department of Defense, Instruction 7000.2: *Performance Management for Selected Acquisitions,* June 10, 1977
8. F. E. McGarry, "What Have We Learned in 6 Years?", *Proceedings of the Seventh Annual Software Engineering Workshop,* SEL-82-007, December 1982
9. Software Engineering Laboratory, SEL-82-003, *Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description,* P. Lo, September 1982
10. G. J. Myers, *The Art of Software Testing.* New York: John Wiley & Sons, 1979
11. T. J. McCabe, *Tutorial-Structured Testing.* New York: Computer Societies Press, 1983
12. W. C. Hetzel, ed., *Program Test Methods.* Englewood Cliffs, N.J.: Prentice-Hall, 1973

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 (Superseded)

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

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 (Superceded)

SEL-80-104, *Configuration Analysis Tool (CAT) System Description and User's Guide (Revision 1)*, W. Decker and W. Taylor, December 1982

SEL-80-005, *A Study of the Musa Reliability Model*, A. M. Miller, November 1980

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 (Superseded)

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 (Superseded)

SEL-81-102, *Software Engineering Laboratory (SEL) Data Base Organization and User's Guide Revision 1,* P. Lo and D. Wyckoff, July 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-81-103, *Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description,* P. Lo and D. Card, July 1983 (Superseded)

SEL-81-004, *The Software Engineering Laboratory,* D. N. Card, F. E. McGarry, G. Page, et al., September 1981 (Superseded)

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 (Superseded)

SEL-81-105, *Recommended Approach to Software Development,* S. Eslinger, F. E. McGarry, and G. Page, May 1982 (Superseded)

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

SEL-81-007, *Software Engineering Laboratory (SEL) Compendium of Tools,* W. J. Decker, E. J. Smith, A. L. Green, et al., February 1981 (Superseded)

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 (Superseded)

SEL-81-110, *Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics,* G. Page and F. McGarry, December 1983

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

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 I,* July 1982

SEL-82-005, *Glossary of Software Engineering Laboratory Terms,* M. G. Rohleder, December 1982 (Superseded)

SEL-82-105, *Glossary of Software Engineering Laboratory Terms,* T. A. Babst, F. E. McGarry, and M. G. Rohleder, October 1983

SEL-82-006, *Annotated Bibliography of Software Engineering Laboratory (SEL) Literature,* D. N. Card, November 1982 (Superseded)

SEL-82-106, *Annotated Bibliography of Software Engineering Laboratory Literature,* D. N. Card, T. A. Babst, and F. E. McGarry, November 1983

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-83-001, *An Approach to Software Cost Estimation,* F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, *Measures and Metrics for Software Development,* D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, *Collected Software Engineering Papers: Volume II,* November 1983

SEL-83-004, *SEL Data Base Retrieval System (DARES) User's Guide,* T. A. Babst and W. J. Decker, November 1983

SEL-83-005, *SEL Data Base Retrieval System (DARES) System Description,* P. Lo and W. J. Decker, November 1983

SEL-83-006, *Monitoring Software Development Through Dynamic Variables,* C. W. Doerflinger, November 1983

SEL-83-007, *Proceedings From the Eighth Annual Software Engineering Workshop,* November 1983

SEL-84-001, *Manager's Handbook for Software Development,* W. Agresti, F. E. McGarry, D. N. Card, et al., April 1984

SEL-RELATED LITERATURE

Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," *Program Transformation and Programming Environments.* New York: Springer-Verlag, 1984 (also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II,* November 1983)

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 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* July 1982)

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 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* July 1982)

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 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* 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 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* July 1982)

Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM, January 1984, vol. 27, no. 1 (also appears in SEL-83-003, Collected Software Engineering Papers: Volume II,* November 1983)

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 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* July 1982)

Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," *IEEE Transactions on Software Engineering,* November 1983 (also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II,* November 1983)

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 (also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II,* November 1983)

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 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* July 1982)

Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," *Computers and Structures,* August 1978, vol. 10 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* July 1982)

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

Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," *Proceedings of the Fifteenth Annual Conference on Computer Personnel Research,* August 1977 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* July 1982)

Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982 (also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II,* November 1983)

Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982 (also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II,* November 1983)

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., V. E. Church, W. W. Agresti, and Q. L. Jordan, "A Software Engineering View of the Flight Dynamics Analysis System," Parts I and II, Computer Sciences Corporation Technical Memorandum, February 1984

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 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* July 1982)

Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," *Proceedings of the Seventh International Computer Software and Applications Conference.* New York: Computer Societies Press, 1983 (also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II,* November 1983)

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)

McGarry, F. E., G. Page, and R. D. Werking, *Software Development History of the Dynamics Explorer (DE) Attitude Ground Support System (AGSS),* June 1983

Miller, A. M., "A Survey of Several Reliability Models" (paper prepared for the University of Maryland, December 1978)

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 (also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I,* July 1982)

Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," *Empirical Foundations for Computer and Information Science* (proceedings), November 1982 (also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II,* November 1983)

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," *Proceedings of the Software Life Cycle Management Workshop,* September 1977

Zelkowitz, M. V., and J. Sukri, "Evaluation of the FDAS Prototype as a Software Development System," (paper prepared for the University of Maryland, February 1984)