SOFTWARE ENGINEERING LABORATORY SERIES **SEL-83-001**

(NASA-TM-85607) AN APPROACH TO SOFTWARE COST ESTIMATION (NASA) 74 p HC A04/MF A01
CSCL 09B

N84-23136

Unclas
G3/61 19074

SOFTWARE ENGINEERING LABORATORY SERIES SEL-83-001

AN APPROACH TO SOFTWARE COST ESTIMATION

FEBRUARY 1984

NASA
National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

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

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

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/TM-83/6076.

The contributors to this document include

Frank McGarry	(Goddard Space Flight Center)
Jerry Page	(Computer Sciences Corporation)
David Card	(Computer Sciences Corporation)
Michael Rohleder	(Computer Sciences Corporation)
Victor Church	(Computer Sciences Corporation)

Single copies of this document can be obtained by writing to

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

ABSTRACT

This document outlines a general procedure for software cost estimation in any environment. The basic concepts of work and effort estimation are explained, some popular resource estimation models are reviewed, and the accuracy of resource estimates is discussed. A software cost prediction procedure based on the experiences of the Software Engineering Laboratory in the flight dynamics area and incorporating management expertise, cost models, and historical data is described. The sources of information and relevant parameters available during each phase of the software life cycle are identified. The methodology suggested incorporates these elements into a customized management tool for software cost prediction. Detailed guidelines for estimation in the flight dynamics environment developed using this methodology are presented.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF EQUATIONS

SECTION 1 - INTRODUCTION

This document presents an approach to software cost prediction that is based on the experience of the Software Engineering Laboratory (SEL) in the flight dynamics environment. This procedure, which produces relatively reliable software cost estimates throughout the software life cycle, consists of the following components:

- Management expertise
- Historical data
- Resource models

Management expertise is fundamental to developing effective software cost estimates. The factors affecting resource expenditures are not all easily quantifiable and may vary in importance from environment to environment. Historical data ranges from personal recollections to corporate data bases. The manager must extrapolate from this data to the estimation task at hand. Prior management experience and careful judgment provide an extra margin of accuracy.

Software resource models are a sometimes useful formalism for developing cost estimates. A number of models are available, but each must be calibrated to the specific development environment in which it is used. Historical data provides the essential reference for the calibration of resource models and for making comparisons.

The process of software cost estimation must be examined in the context of each phase of the development life cycle. Reestimation is performed throughout the changing development effort, not just in the beginning where uncertainty is especially high. The accuracy of each estimate may be improved by awareness and understanding of the steps and processes involved in software development. As the life cycle progresses, more information becomes available about the size and complexity of the system, and the resources already expended are known. This additional information can be used to improve the estimates. Thus, the estimates become more accurate each time they are updated.

This document presents an approach to software cost estimation that can serve as a model for technical managers and others concerned with estimation. Although the recommendation is based on the experiences of the SEL in a specific environment, its applicability is not restricted to that environment. Its specific numerical parameters must, however, be verified or redefined for each new environment. This procedure produces a management tool customized to the user's environment.

1.1 DOCUMENT ORGANIZATION

This document is divided into five major sections and an appendix. Section 1 describes the purpose and scope of the document, presents summaries of the SEL and the flight dynamics software development environment, and examines the software development life cycle.

Section 2 describes the basic concepts of software cost estimation. Common approaches to estimating software work and development effort are discussed, and some comprehensive resource estimation models are reviewed. The reader familiar with these topics may want to skim (or skip) this section.

Section 3 outlines a general software cost estimation procedure. The sources of information and relevant parameters available during each phase of the life cycle are discussed, and their roles are defined.

Section 4 provides detailed guidelines for the application of the general procedure in the flight dynamics environment.

With appropriate modifications, these guidelines are applicable in any similar environment.

Section 5 summarizes the major points made in Sections 3 and 4. Some important cost estimation considerations and general recommendations are reemphasized.

The appendix provides additional details of the SEL software cost experiences. Tables summarize the type of software developed and the basic estimation relationships derived from its study.

1.2 SOFTWARE ENGINEERING LABORATORY

This document is based on the practical and analytical experience of the SEL (Reference 1). The SEL monitors and studies all software developed by the Systems Development Section at the National Aeronautics and Space Administration/ Goddard Space Flight Center (NASA/GSFC). This section is responsible for producing flight dynamics support software for GSFC-supported space missions. Nearly 50 projects developed by both GSFC and contractor employees were studied through 1983. Much of the data is collected on a series of forms completed by project personnel throughout the development effort. Data is also collected through computer accounting monitoring, personal interviews, automated tools, and summary management reviews.

Most flight dynamics projects are developed on a group of IBM mainframe computers using FORTRAN and assembly language. The specific software applications include attitude determination, attitude control, maneuver planning, orbit adjustment, and general mission analysis. Project sizes range from 1500 to 110,000 lines of source code. Project schedules range from 12 to 21 months. The typical technical staff member has about 4 years of experience developing flight dynamics applications.

The initial goal of the SEL was to understand the flight dynamics software development process and its environment. This understanding provides a baseline for measuring the effects of attempted improvements. Currently, the SEL is trying to improve the process and environment to produce high-quality software with fewer errors at a lower cost. To achieve these goals, the SEL must identify the development techniques available, evaluate these techniques to determine the most effective ones, adapt the "best" techniques for optimal performance, and apply the customized techniques to the software development process.

The software cost estimation procedure presented in this document is based on more than 7 years of software development experience and detailed analysis of projects developed in the flight dynamics environment. Reference 1 examines the SEL and the flight dynamics environment in more detail.

1.3 SOFTWARE LIFE CYCLE

The SEL uses a conceptual model of software development that includes four components and defines a software life cycle (see Figure 1-1). The components of the model are as follows:

- Problem (software requirement)
- Environment (in which development takes place)
- Process (divided into phases)
- Product (software solution to the problem)

The parameters available for estimating resource utilization are derived from measures of these components. The rate at which the software development process uses resources (especially human and computer) from the environment is related to the current activity (or phase) of the process. Thus, any effective resource estimation procedure must be phase dependent.

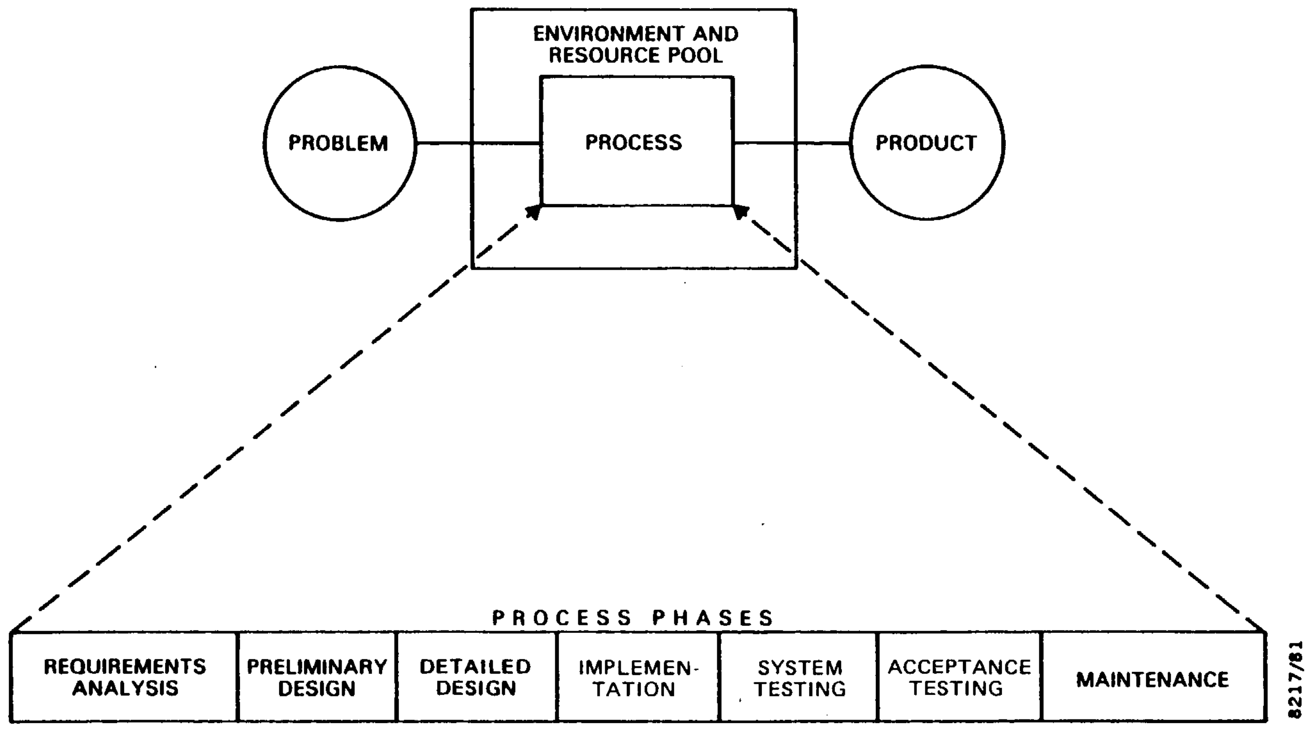


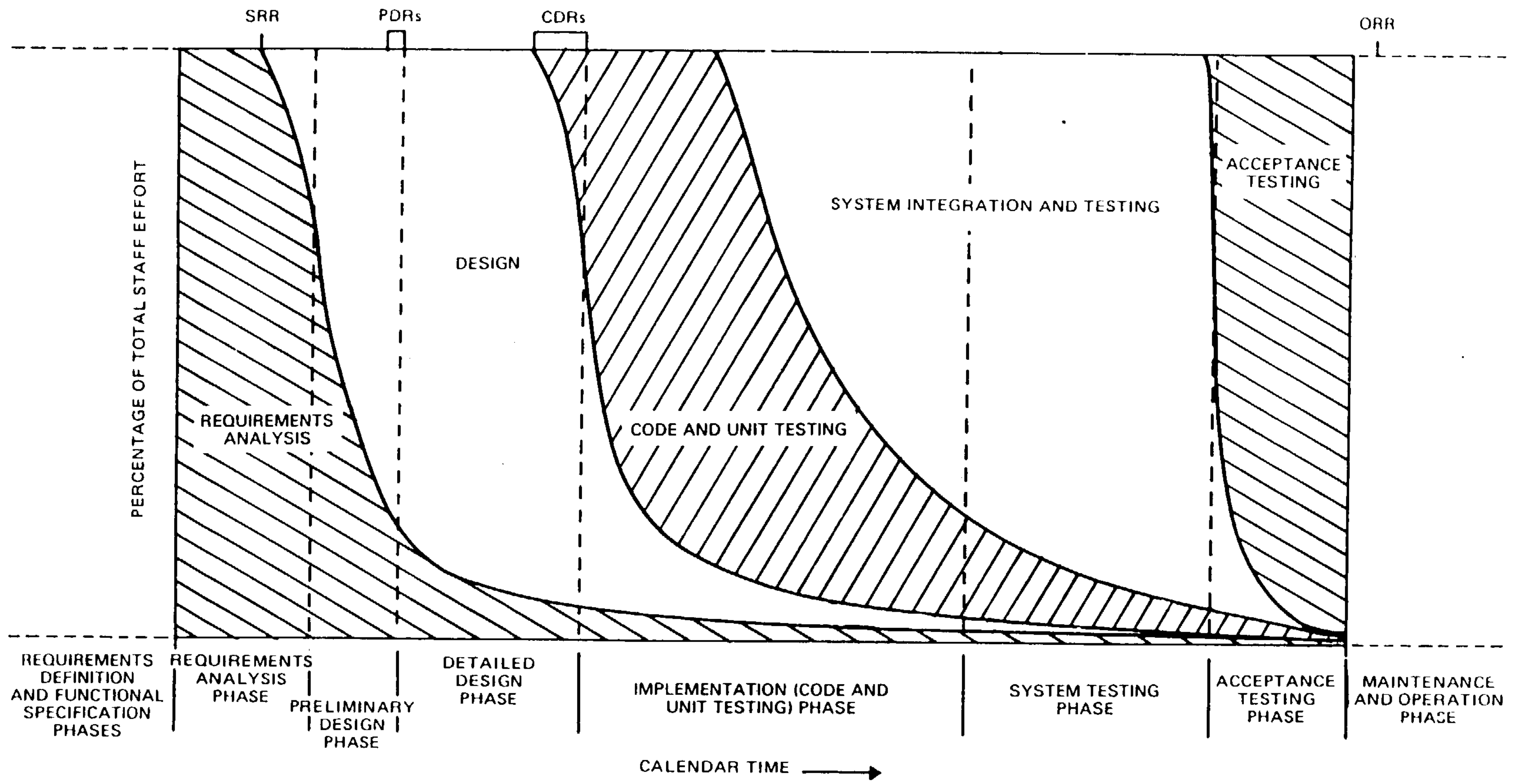
Figure 1-1. Software Development Model

The process component of the model is divided into the following sequential phases (see Figure 1-1), referred to as the software development life cycle:

- Requirements analysis
- Preliminary design
- Detailed design
- Implementation
- System integration and testing
- Acceptance testing
- Maintenance and operation

These seven life cycle phases divide the software development effort into sequential, nonoverlapping periods of time. In addition, prior to the start of development, there is a problem definition phase referred to as "requirements definition and functional specification." This phase is not, however, an important consideration of this document.

Each calendar phase of the software development life cycle is characterized by specific activities and the products generated by those activities. Reference 2 presents an in-depth discussion of the activities occurring during each life cycle phase. Activities that are characteristic of one calendar phase, may, however, be performed in other phases. For instance, the activity of analyzing requirements, which makes up the bulk of effort during the requirements analysis phase, continues at a lower level throughout the software development life cycle as further understanding of the requirements is obtained and as changes to the requirements are made. Changes to the requirements after the requirements analysis phase may necessitate additional activity from all earlier life cycle phases. Figure 1-2 identifies the activities performed during each calendar life cycle phase as a percentage of the total staff effort.



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

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

The life cycle is simplified for some discussions in this document. Three phases are used in these cases: design (consisting of requirements analysis, preliminary design, and detailed design), implementation, and testing (consisting of system and acceptance testing).

SECTION 2 - CONCEPTS OF SOFTWARE COST ESTIMATION

Cost estimation is an essential function of software development management. As the range of computer applications has expanded and the complexity of tasks increased, the cost of software development has multiplied until software is now the largest component of total system cost. Accurate prediction of resource requirements and schedules is a prerequisite for effective management. Reliable estimates are especially critical to the planning of large projects.

This section discusses the basic concepts and relationships underlying any software cost estimation procedure. Although Section 2.5 reviews some comprehensive cost models, no specific recommendations are made about them. Instead, Section 3 proposes a more general procedure incorporating the concepts introduced here. This section simply identifies some widely accepted ideas and techniques.

The estimation process consists of two discrete steps that are, however, often combined into a single computational formula: (1) estimating the amount of work to be done and (2) estimating the amount of effort needed to do the work. After these quantities have been estimated, a schedule must be developed. However, the schedule selected may also affect the cost (see Section 2.3). The following output is desired from the estimation process:

- Size of product upon completion
- Effort to complete product
- Development schedule
- Uncertainty of the estimates

Although these quantities should be reestimated periodically, that is not commonly done. Usually, a single estimate is made prior to the start of a software development project.

A common practice used to develop initial estimates is to divide the project into subsystems, multiply by the approximate lines of code needed to implement each subsystem, divide this result by a nominal productivity rate, and then distribute the estimated effort over the life cycle according to some rule. The accuracy of this process can be improved by regularly collecting and reviewing relevant historical data such as subsystem size, productivity rate, and effort distribution.

The following sections discuss some common approaches to estimating work and effort, developing schedules, and defining estimate accuracy. In addition, Section 2.5 describes some formal resource estimation models that have been used in the flight dynamics environment.

2.1 ESTIMATING SOFTWARE WORK

The first step in producing a cost estimate for a software development project is to estimate the amount of work required. This step relies heavily on management expertise, because to some extent, every project is unique. Important parameters to consider when estimating software work are as follows:

- Number of functions
- Number of subsystems
- Number of programs
- Number of requirements
- Number of interfaces
- Number of modules

The estimator usually divides a project into elements whose cost can be estimated separately, and then combines the estimates for the individual elements into an estimate for the total project. These elements are referred to as "work units." The more detailed the work units, the more accurate the resulting cost estimate will be, when the more detailed work units are accurately known (Reference 3).

Lines of source code is the most widely accepted work unit. Most comprehensive resource models (see Section 2.5) expect as input an estimate of size in terms of lines of code. It is difficult, however, to do more than guess at the number of lines of code that will compose a system without first decomposing it in some manner. The usual approach is to hierarchically decompose the software system into intermediate work units from which an estimate of lines of code can be made. Many organizations employ the "work breakdown structure" formalism, but that technique is not discussed in this document.

Decomposition is a continuous process that gradually refines the system definition throughout the software life cycle. Estimates during any phase are based on the structures and functions defined thus far. Decomposition consists of first dividing a system into subsystems, then dividing the subsystems into modules. Historical data can be used to estimate the size, in lines of code, of each module. This decomposition cannot, however, be completed until the preliminary design is completed. Section 3.1 provides some guidelines for performing the decomposition.

Other approaches to work estimation are possible. For example, Albrecht (Reference 4) proposed a cost estimation relationship employing "function points" as the work units. The number of function points in a system is a weighted sum of the number of data items, transaction types, and interfaces. This technique does not require that the number of lines of code be estimated. It has been shown to give good results in some data processing applications (primarily COBOL and PLI programs).

Although other work units have been employed in cost estimation, lines of code has proven to be the most durable. A line of code is an 80-byte record that can be processed by an assembler or compiler. A FORTRAN line of code can contain one or part of one statement or comment.

Several variations of this basic measure are in use, including

- Delivered lines of code
- Developed lines of code
- Executable lines of code
- Non-comment lines of code

The best measure for a specific environment can only be determined through experimentation with historical data. However, consistent use of the same measure preserves the comparability of data collected. Reference 5 defines a number of variations of the lines of code measure.

2.2 ESTIMATING DEVELOPMENT EFFORT

The second step in producing a cost estimate for a software development project is to estimate the amount of effort required. This basically consists of dividing the estimate of work, usually lines of code, by a nominal or historical productivity rate. Many other factors may, however, affect the actual productivity rate of a project, including:

- Methodologies applied
- Staff experience
- Schedule
- Number and severity of requirements changes
- Clarity and completeness of requirements
- Complexity of software application
- Environmental constraints
- Documentation to be produced
- Reliability required

The manager must rely on his/her accumulated experience to adjust the initial cost estimate (based on historical data) to account for the effects of these factors. Section 3.1 provides some guidelines for making this adjustment. Cost estimates are usually made in terms of staff-hours. This makes them resistant to inflation and salary variations.

2.3 PROJECTING DEVELOPMENT SCHEDULES

After developing an estimate of the amount of effort required to complete a software project, a schedule for software production must be defined. This requires the determination of the optimum distribution of effort and the total development time. Generally, a longer development time and smaller staff are preferred. In practice, the total development time is usually fixed by a deadline for software delivery.

The task of developing a schedule is then reduced to optimizing resource expenditures within the fixed development time. However, as Brooks (Reference 6) pointed out, time and effort are not freely interchangeable. A shorter schedule reduces productivity, although the magnitude of this effect is not clear. Tausworthe (Reference 7) suggested that it is very costly to shorten a schedule by more than one-half from the time that it would take a single individual to do the work.

Two concepts are widely used in projecting phase transition dates and staffing levels once a target completion date has been selected. Wolverton (Reference 8) proposed that software development effort should be apportioned as follows: design, 40 percent; implementation, 20 percent; and testing, 40 percent. This is referred to as the 40-20-40 rule. The manager may vary this rule to account for specific features of a project or an environment. For example, a software rehosting project will require less design and more testing effort than developing a new project of comparable size.

Putnam (Reference 9) suggested that the optimum level of effort during the life of a software development project followed a Rayleigh curve. This curve has been successfully applied to many other time-dependent processes. Manipulation of its equation produces values for the maximum staffing level and staffing levels throughout the project life cycle. Equation (2-1) defines the Rayleigh curve:

$$Y = 2Kate^{-at^2} \qquad (2\text{-}1)$$

where K = total development effort
t = time elapsed since start
t_m = time of maximum effort
$a = t_m^2/2$
Y = effort level at time t

Figure 2-1 shows the effort profile of a hypothetical project. Combining Rayleigh curves for each of the three activities according to the 40-20-40 rule produces a graph (also a Rayleigh curve) of the level of effort through development. The area under the curve represents the estimate of total development effort to complete the project. The time from start to completion is the total development time. The maximum level of effort (or staffing level) occurs when 40 percent of the effort has been expended, usually near the time of the critical design review (CDR). Testing falls off gradually until project completion.

2.4 DEFINING ESTIMATE ACCURACY

Reliance on any software cost estimation procedure must be tempered by knowledge of its expected accuracy. Underestimation is the most common error. There are a number of reasons why managers and developers tend to underestimate software development costs. They may vary from project to

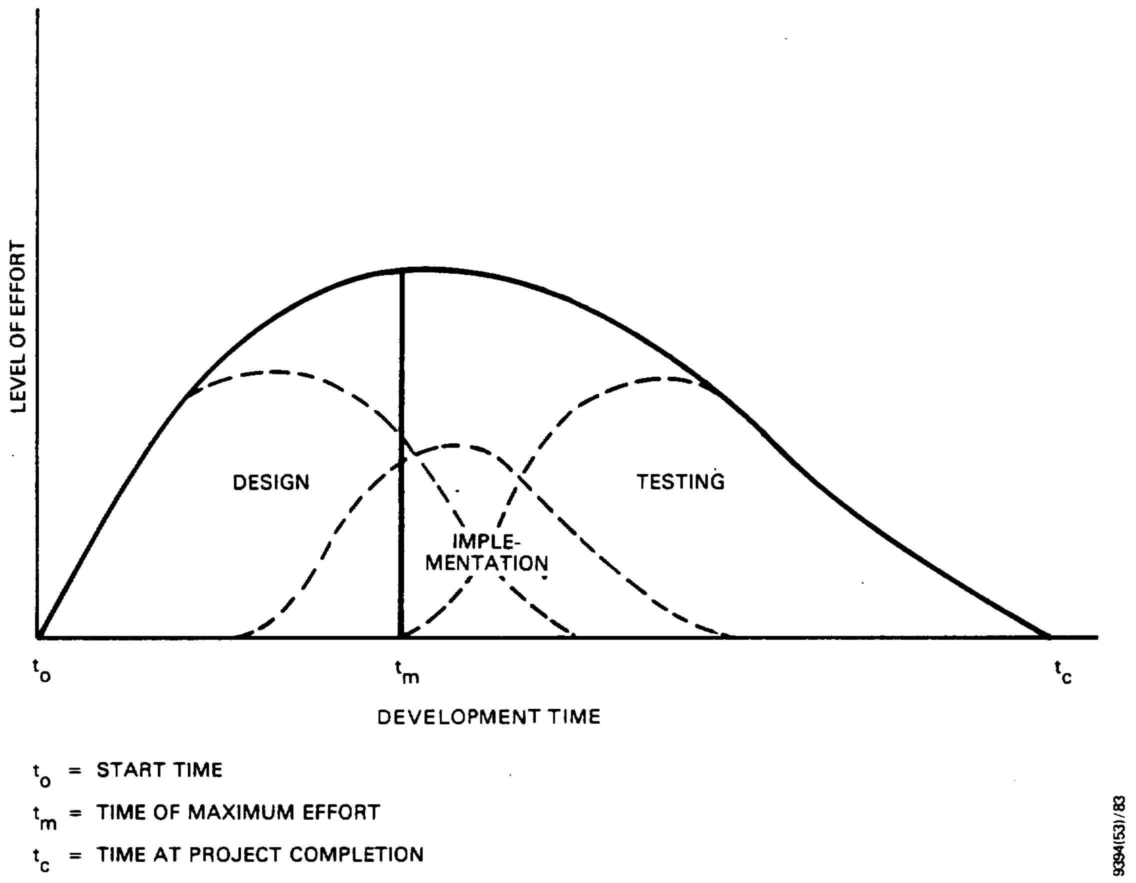


Figure 2-1. Rayleigh Curve Defined Schedule

project and from environment to environment, but include the following:

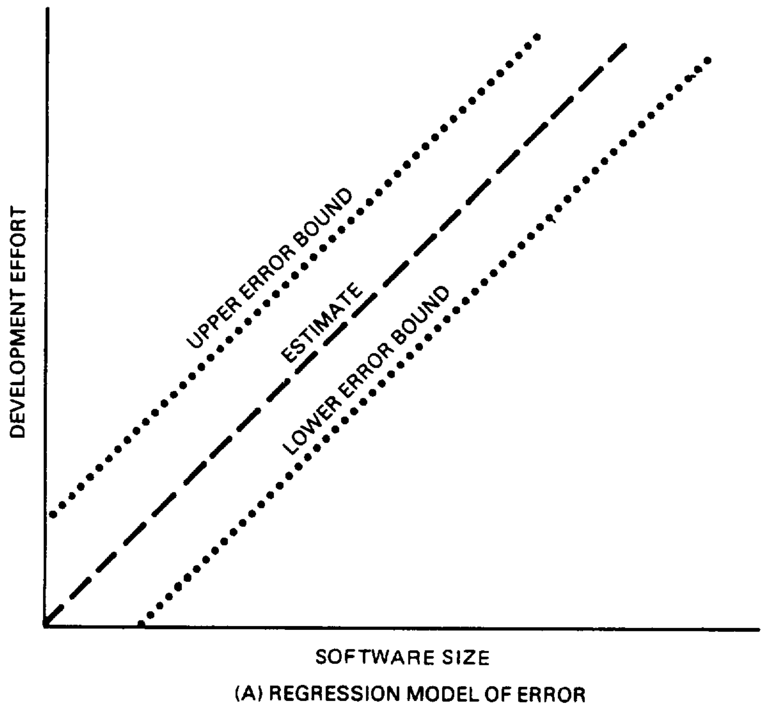
- Full scope of work not known
- Lack of adequate history and experience
- Management pressure to maintain original cost and schedule
- Tendency of human nature to be optimistic

Because these causes are due to errors of subjective judgment, they cannot be compensated for by a simple bias in the estimation procedure. However, keeping alert for these problems can minimize their impact.

Potential estimation errors are usually represented as ranges. An error range is a probabilistic statement. That is, a probability (confidence) of the actual value occurring within a specified range can be associated with that range. A larger error range has a larger probability associated with it. Error ranges are selected to be large enough that the confidence in them is great, typically 95 or 99 percent.

Appropriate error ranges and corresponding probabilities can be determined by regression analysis when reliable data are available. Figure 2-2(A) illustrates the results of this type of analysis. Adequate data are not, however, always available for this procedure. Furthermore, the results are not fully consistent with SEL software cost experience. As mentioned earlier, the SEL has observed that the error distribution is skewed; overestimates (actual cost less than estimated cost) are much less likely than underestimates (actual cost greater than estimated cost). Also, larger projects often exhibit significantly larger estimation errors than small projects.

The SEL has developed an alternate model of estimation error that conforms more closely to observed error behavior.



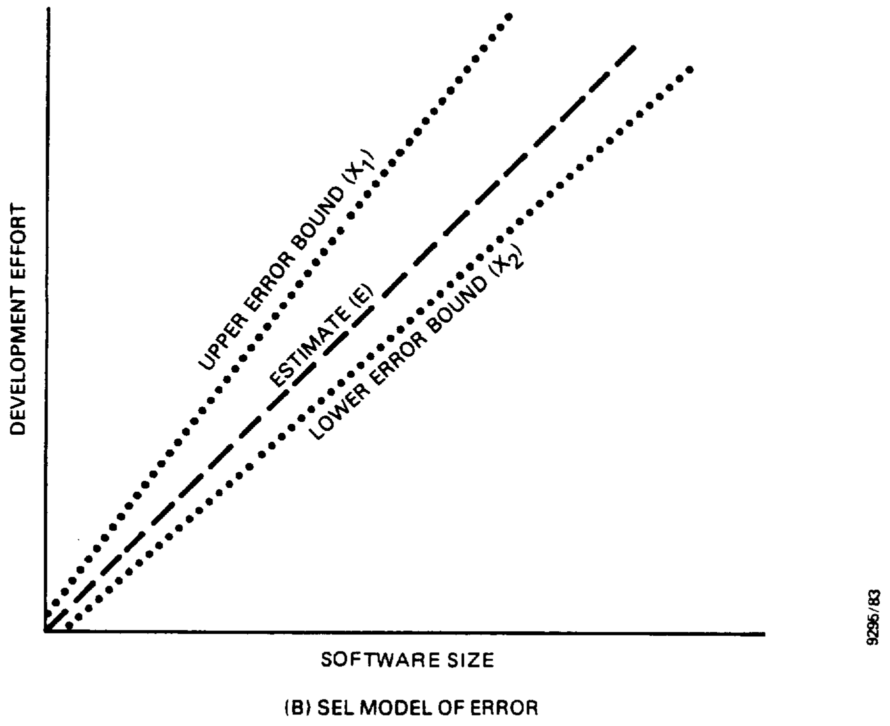


Figure 2-2. Alternative Models of Estimation Error

Figure 2-2(B) illustrates the properties of this model. It incorporates a higher probability for underestimation. The equations used to define error bounds are as follows:

$$X_1 = E(1 + X) \qquad (2\text{-}2)$$

$$X_2 = E/(1 + X) \qquad (2\text{-}3)$$

where X = error magnitude (proportion)
X_1 = upper error bound
X_2 = lower error bound
E = estimate

The error magnitude corresponding to an error range is the size of the greatest deviation (from the estimate) possible within the error range. Error magnitudes are usually determined to provide a nominal level of confidence of 95 percent. The notation used in this document to provide error ranges is $A \pm X$, where A is the estimated cost and X is the error magnitude as previously defined. One useful property of this model is that it precludes error ranges that include zero effort. Table 2-1 gives the approximate uncertainty for estimates made in each life cycle phase based on SEL experience. For example, an estimate made at the end of requirements definition (start of requirements analysis) could deviate by a factor of two, from twice to one-half of the actual cost.

Table 2-1. Accuracy of Resource Estimation by Life Cycle Phase

Life Cycle Phase	Estimate Uncertainty at End of Phase (Proportion)
Requirements Definition	1.00
Requirements Analysis	0.75
Preliminary Design	0.50
Detailed Design	0.30
Implementation	0.12
System Testing	0.05

2.5 RESOURCE ESTIMATION MODELS

Resource estimation models are rigorous formulations of the estimation process. Most of these models are equations of the form

$$E = AW^B + C \tag{2-4}$$

where E = measure of effort expended (usually hours)
A = constant (or index of local conditions)
B = constant (usually specific to each environment)
C = constant (or index of local conditions)
W = measure of work required (usually lines of code)

Models differ as to the measures of work and effort used. Some derive values for A, B, and C by regression with historical data. Others substitute indexes (measures) of local conditions and development practices for A and/or C.

The SEL has reviewed and tested many resource estimation models over the past years (Reference 10). The following appear to be useful in the SEL environment:

- Meta-Model (SEL, Reference 11)
- COCOMO (TRW, Reference 12)

- SLIM (QSM, Reference 13)
- PRICE S (RCA, Reference 14)

Although these models are discussed briefly in the following sections, they are not presented as recommendations but rather as examples of the ways in which models can be implemented.

2.5.1 META-MODEL

A set of relationship equations was developed by the SEL to define an environment-specific model. The relationship (base equation) established between effort and size is as follows:

$$E_i = 0.73L^{1.16} + 3.5 \qquad (2\text{-}5)$$

where E_i = initial estimate of effort (staff-months)
L = developed lines (thousands, see Section 2.1.1)

The next step was to collect data about the programming environment to determine why some projects took more effort and consumed more resources than others when normalized for size. Attribute indices were constructed to account for the variation due to such factors as problem complexity, programmer experience, and development techniques. Each attribute for each project was rated on a scale from 0 to 5. A sum was then calculated for each class of attributes. These sums are the indices used to adjust the initial estimate of effort based on delivered lines of code. The final equation used, which includes two such indices, is as follows:

$$E_f = E_i\ (-0.036\ M + 0.009\ C + 0.86) \qquad (2\text{-}6)$$

where E_f = final estimate of effort (staff-months)
E_i = initial estimate of effort (see Equation (2-5))
M = sum of methodology ratings
C = sum of complexity ratings

The resulting adjusted estimator is the best predictor of development effort (in this environment) of those estimators examined thus far by the SEL. A full treatment of the derivation of the model is given in Reference 11.

2.5.2 COCOMO

The Constructive Cost Model (COCOMO) exists as a hierarchy of increasingly detailed and accurate forms. The basic equation for estimating the number of staff-months required to develop software in terms of the number of thousands of delivered source instructions in the software product is as follows:

$$M = 2.4\ I^{1.05} \tag{2-7}$$

where M = staff-months of effort
I = thousands of delivered source instructions

An equation for estimating the optimum development time is also provided:

$$T = 2.5\ M^{0.38} \tag{2-8}$$

where T = months of development time
M = staff-months of effort (see Equation (2-7))

This basic model is good for quick order-of-magnitude estimates of software costs. A more detailed presentation of the COCOMO model is provided in Reference 12, which discusses the effects of hardware constraints, personnel quality and experience, use of modern programming tools and techniques, and other project attributes assumed to have a significant influence on software costs.

2.5.3 SLIM

The Software Life-Cycle Management (SLIM) model is a proprietary software package available through a timesharing network (Reference 13). Based on data collected from past software projects from a user's own organization as well as the type and size of a proposed new project, the user can obtain manpower, cost, and schedule estimates for the new project. Cash flow over the life cycle can be projected. In addition, SLIM will identify limiting constraints on manpower and schedule, find the trade-off between cost and development time, and adjust for requirements changes to give new cost, manpower, and schedule estimates. A key parameter used in SLIM is called the technology factor. It is described as an indicator of the state of technology that a particular organization applies to a software development project. It is based on data supplied by the user from past projects.

SLIM produces a primary estimate of the development cost at the system level. It provides an optional "front-end" estimate that includes the analysis and design phases. Factors such as operation and support cost can be obtained as another option. Estimates of computer hours and documentation are also available. The life cycle phases are overlapping but fixed in relative size. Milestone events describe the beginnings and ends of phases.

2.5.4 PRICE S

The Programmed Review of Information for Costing and Evaluation - Software (PRICE S) model is another proprietary software package available through a timesharing network (Reference 14). The model computes the projected costs and manpower required for each of three overlapping development phases: engineering design, implementation, and test and integration. The model also computes typical schedules appropriate for the size, type, and difficulty of the proposed project. Alternative scheduling may be specified by the user.

PRICE S is unique in that it allows subsystem-level definitions to be explicitly stated for all of its life cycle and activity elements. Adjustments to account for the additional effort needed to integrate each subsystem into the system may be specified by the user.

PRICE S supports two alternative modes of operation. One allows the model to run in "reverse" to calculate empirical factors from historical costs. Another uses specified costs to compute typical program sizes and project schedules.

SECTION 3 - SOFTWARE COST ESTIMATION GUIDELINES

This section provides an outline of the procedures and guidelines recommended by the SEL for estimating the resource requirements of software development projects. These recommendations are based on experiences with the development of medium-scale (up to 150K lines of code) scientific software from relatively good and complete specifications in a reasonably stable (i.e., hardware interfaces and support software) environment (see Appendix). However, these recommendations are, to some extent, also appropriate to larger systems.

Some potentially important factors, such as special hardware configurations, personnel capabilities, and unusual software constraints, are not discussed in this document. Although these factors, if applicable, can invalidate any estimate, the purpose of this discussion is to explain a general procedure rather than to attempt to account for all the exceptional situations. Boehm's comprehensive text (Reference 12) deals with many of these possibly critical factors.

Software development costs include a wide range of human and material resources. For the purposes of exposition, these costs are grouped into three classes:

- Basic cost of software development--Staff resources (programmer, manager, and support) required to develop a new software system
- Additional software development costs--Other resources (computers, documentation, test team, analysis) needed to support the development team
- Other software development activities--Staff resources required in situations other than the development of a new system (rehosting, maintenance, and reuse)

The following subsections describe the steps of the general estimation process, explain the sources of information for each cost type, and identify the key cost estimation parameters. There are, however, two prerequisites for effective cost estimation:

- Defining the software life cycle and the products associated with costs
- Determining nominal values for the basic cost parameters (e.g., productivity and module size)

The life cycle and products defined need not correspond to a work breakdown structure, although this is a good way to proceed. The nominal parameter values may be based on management experience or historical data. The values reported in this document are based on SEL data, except where otherwise specified.

3.1 BASIC COST OF SOFTWARE DEVELOPMENT

This section describes a general procedure for estimating the staff resources required for the development of a new software system (not a modification or extension of an existing system). The procedure is based on the life cycle and products defined by the SEL for the flight dynamics environment at GSFC. Figure 3-1 shows the life cycle and the points at which estimates are recommended. Table 3-1 lists the products associated with each life cycle phase. The estimation process consists of the development of initial estimates of both software size and cost as well as its periodic refinement as additional information becomes available throughout the software life cycle.

3.1.1 INITIAL ESTIMATE

The initial estimate must be made prior to the start of development, during the requirements definition and specification phase. This estimate is based largely on the previous

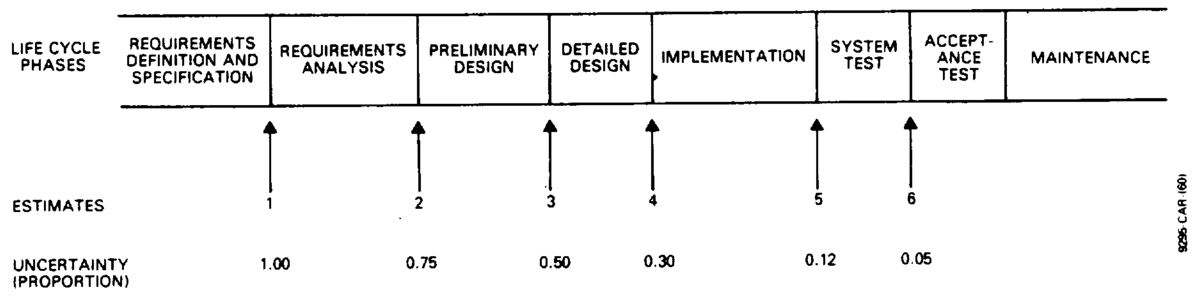


Figure 3-1. Cost Estimation Schedule

Table 3-1. Software Life Cycle Summary

Phase	Products	Percent of Schedule[1]
Requirements Analysis	Requirements Summary Implementation Plan	5
Preliminary Design	High-level Design	10
Detailed Design	"Code-to" Specification	15
Implementation	System Code System Documentation	40
System Testing	Verified Code	20
Acceptance Testing	Reverified Code and Documentation	10

[1]Percent of total development schedule (calendar time) spent in each phase.

experience of the manager/estimator with similar projects. However, following a general procedure provides a formal mechanism for incorporating this experience in the software cost estimate. These four steps are recommended:

1. Decompose the problem and products as far as possible.
2. Relate the elements defined in the decomposition to functions in previously developed systems.
3. Identify special capabilities and considerations unique to this system.
4. Select a probable rate of work and other parameters appropriate to this system.

The first step is to decompose the information supplied by the requirements specifications to the deepest level of detail possible (e.g., system, subsystem, module). All major system functions must be identified. Although decomposing the requirements to the "line of code" level is desirable, it is not usually possible. A decomposition to the subsystem level is usually adequate for an initial estimate.

The second step is to associate the elements defined in the decomposition with similar functional units (e.g., telemetry processing, data base management, input/output processing) in previously developed software systems. The key to a successful initial estimate is to identify as many similarities as possible. Such prior experience is the best guide to estimating the size and cost required to develop that function or capability. Ideally, numerical data will be available from several similar systems. In the absence of historical data and personal experience, cost estimation is largely guesswork.

The third step is to identify those functional capabilities and system characteristics that are special or unique to the system to be developed. A list of these features should be prepared to ensure that their effects are reconsidered at each subsequent reestimation. Features to be alert for include the following:

- Graphic versus nongraphic
- Real-time versus non-real-time
- Interactive versus noninteractive
- Unusual hardware interfaces

The inclusion of any of these features can substantially affect software cost; the manager must rely on his/her personal expertise to estimate the magnitude of these effects.

The fourth and last step is to select estimation parameters appropriate to the level of detail of the functional decomposition. For example, if the software specification has been decomposed to the module level, the parameters needed are lines of code per module, weeks per module per person, and hours per module. Appropriate parameter values may be suggested by the comparison with similar systems in step 2. Simple arithmetic calculations using these parameters generate the desired estimates as follows:

Size = Lines of code per module x Estimated modules

Cost = Hours per module x Estimated modules

Time = Weeks per module per person x Estimated modules ÷ Estimated staff

These estimates must be adjusted to account for any of the special or unique factors identified in step 3. The depth of decomposition increases throughout the software life cycle (see Table 3-2). Because the decomposition only achieves the subsystem level during requirements definition and specification, subsequent reestimation is necessary.

Table 3-2. Basic Cost Estimation Parameters[a]

Requirements Analysis[b]		Nominal Value[c]
Size:	Lines of code per subsystem	7500
Cost:	Hours per subsystem	1850
Schedule:	Weeks per subsystem per person	45
Preliminary Design[b]		
Size:	Lines of code per module	125
Cost:	Hours per module	30
Schedule:	Weeks per module per person	0.75
Detailed Design[b]		
Size:	Relative weight of reused[d] code	0.2
Cost:	Hours per developed line of code	0.3
Schedule:	Weeks per developed module per person	1.0
Implementation		
Size:	Percent growth during testing	10
Cost:	Testing percent of total effort	25
Schedule:	Testing percent of total schedule	30
System Testing		
Cost:	Acceptance testing percent of total effort	5
Schedule:	Acceptance testing percent of total schedule	10

[a]At end of each phase.

[b]Estimates of total cost, size, and schedule; not required to complete.

[c]Based on data collected in the flight dynamics environment (see appendix).

[d]Does not include extensively modified reused modules.

3.1.2 PERIODIC REESTIMATION

Subsequent reestimates follow the same four steps as the initial cost estimate, but make use of the most detailed decomposition to date and other information that becomes available. Table 3-2 lists the principal cost estimation parameters by the life cycle phase in which they are expected to be used. The nominal values for the parameters reported in the table are examples of how historical data can be used to define cost estimation relationships for a specific environment. The accuracy of estimates increases as more detailed information becomes available during the development process.

During requirements analysis, the nature of the software development task becomes clearer. Although the system decomposition may not advance beyond the subsystem level, the scope of subsystems, complexity of functions, and special requirements are better understood. Estimates made at the end of requirements analysis should fall within 75 percent of the actual size and effort.

The system decomposition is refined during the design process. At the end of preliminary design, the total number of modules will be known. There may also be some indication of the development team's productivity. Any special factors identified during the initial estimate must be reevaluated. Estimates made with the additional information available at this time should fall within 50 percent of the actual size and effort.

During detailed design, those modules that can be adapted from existing software are identified. The amount of new and reused code can be determined by multiplying the nominal module size by the number of modules of each type. <u>Reused modules cost only about 20 percent as much as new modules</u>.

Equation (3-1) produces a weighted measure of software size (develcped lines of code) incorporating this cost difference.

$$L = N + 0.2R \qquad (3\text{-}1)$$

where L = developed lines of code

N = new and extensively modified lines of code

R = slightly modified and unchanged lines of code

This additional information permits a further refinement of the estimates. Estimates made at the end of detailed design should fall within 30 percent of the actual size and effort.

By the end of implementation, a system will reach nearly full size, and most of the development effort will have been expended. Estimates of the amount and cost of testing must be based on the size and perceived quality of the software. Estimates made at this time should fall within 12 percent of the actual size and effort.

Although some system growth may occur during system testing, the software should stabilize by the end of this phase. Estimates made at this time should fall within 5 percent of the actual size and effort. Unless unusual problems are encountered, subsequent acceptance testing should require only a small expenditure of time and effort.

Development normally ends with the completion of system and acceptance testing. However, some subsequent cleanup and delivery effort may be required. This includes generating tapes, compiling reports, and finalizing documentation. No more than 3 percent of the development effort should be required for this activity.

3.2 OTHER SOFTWARE DEVELOPMENT COSTS

Several other costs and quantities, in addition to the basic cost of software production, should be estimated by the software development manager. These include the following:

- Computer utilization
- System documentation
- Independent test team
- Analysis support

Computer use and documentation are important considerations for any software development project. However, the costs of an independent test team or analysis support are not applicable to all.

Sections 3.2.1 through 3.2.4 provide guidelines for estimating these costs and quantities based on SEL experience. The costs described in these sections are incurred in addition to the development costs described in Section 3.1.

3.2.1 COMPUTER UTILIZATION

A lack of computer resources is one of the most commonly cited reasons for software development delay. However, little attention is usually given to ensuring adequate computer support until a bottleneck is encountered. Accurate forecasting and effective scheduling can minimize delays from this source. The demand for computer time depends on the following factors:

- Life cycle phase (see Figure 3-2)
- Type of software developed
- Development environment

Equation (3-2) estimates the total computer resources required for a flight dynamics project:

$$H = 0.009L \qquad (3\text{-}2)$$

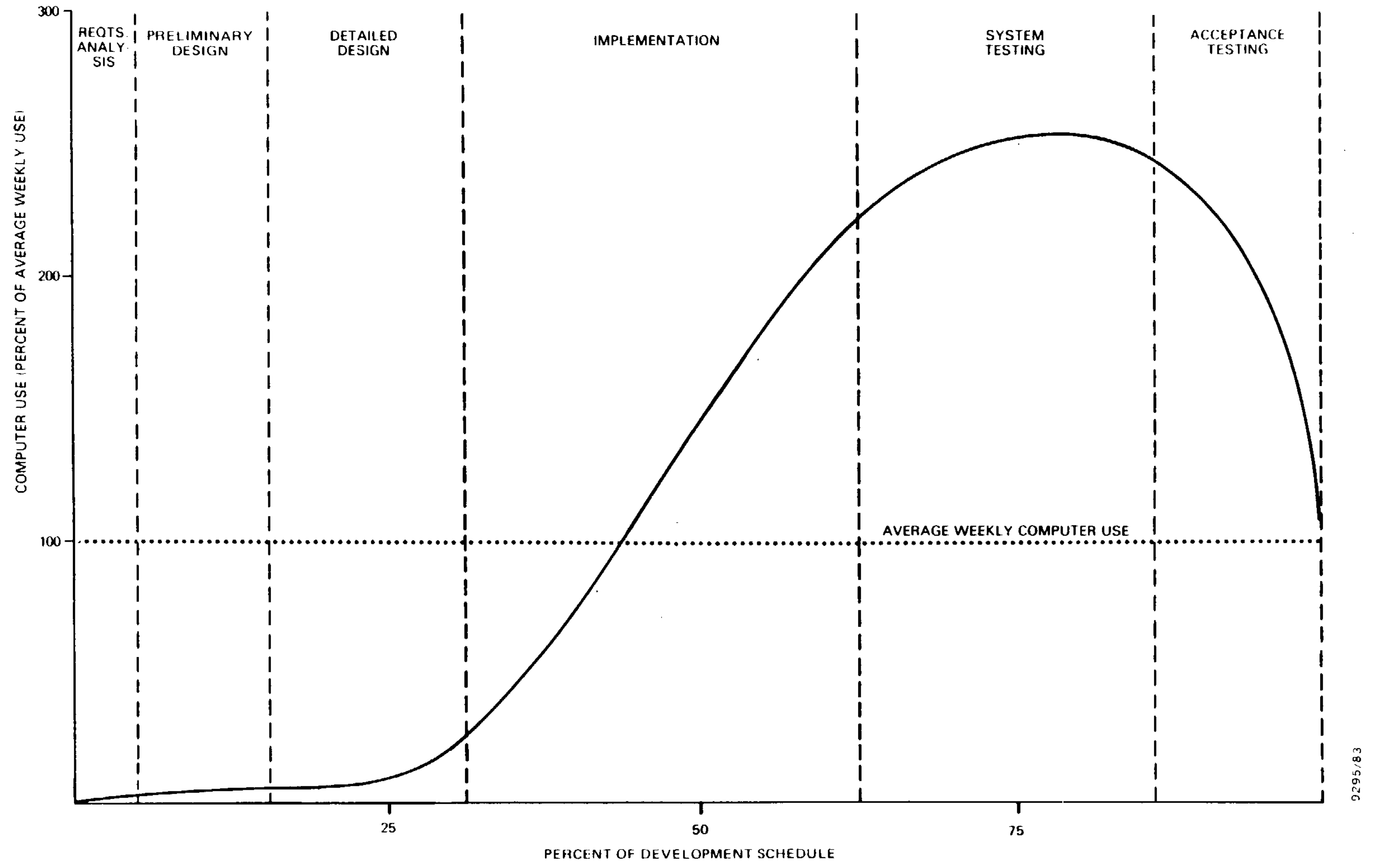


Figure 3-2. Typical Computer Utilization Profile

where H = CPU hours of computer time
L = developed lines of code (see Equation (3-1))

Some types of projects (e.g., rehosting and real-time software) will require more computer resources. Online development also encourages computer use. Figure 3-2 shows the level of computer use during the life cycle of a typical flight dynamics project. The maximum rate of computer use is attained during system testing and should not exceed three times the average weekly rate.

3.2.2 SYSTEM DOCUMENTATION

Although often regarded as an expendable item, effective system documentation has been proven to promote software quality (Reference 15). Cost and schedule overruns elsewhere should not be made up by skipping essential documentation. However, the amount and types of documentation appropriate to a system depend on its planned lifetime and user needs. Equation (3-3) provides an approximation of system documentation size:

$$P = 0.04L \tag{3-3}$$

where P = pages of documentation
L = developed lines of code (see Equation (3-1))

This system documentation includes the design description, test plans, user documents, component prologs, and the development/management plan. Although some organizations produce separate development and management plans, in the flight dynamics environment these are combined into a single document. Table 3-3 shows the relative proportions of these document types.

Table 3-3. Composition of System Documentation

Document Type	Percent of Total Pages
Design Description	33
Test Plans	7
User Documents	41
Component Prologs	16
Development/Management Plan	3[a]

[a]Usually does not exceed 100 pages.

The development team must generate most of the materials contained in these documents as an intrinsic part of the software development process. However, the user documents (user's guide and system description) are optional. The cost of informal user documents includes the effort spent organizing materials (baseline diagrams, data set descriptions, etc.) and writing explanatory text. Formal documents also require typing, editing, review, and graphics production. Table 3-4 shows the cost of these documentation alternatives.

Table 3-4. Cost of User Documents

Document Level	Additional Cost[a]
No User Documents	0
Informal User Documents	5
Formal User Documents	16

[a]Percent of basic development cost.

3.2.3 INDEPENDENT TEST TEAM

One technique employed to increase software quality is to supplement the development team with an independent test and verification team. This team participates in requirements and design reviews, inspects code, and conducts tests. The independent test team does not, however, replace any of the functions of the development team. The purpose of such a team is to discover errors and discrepancies, but not to fix them. Problems are reported to the development team for resolution.

Operating an independent test team in conjunction with a development project adds approximately 10 to 20 percent to the cost of the development project, based on SEL experience (Reference 16). The staff level of the independent test team usually remains constant throughout the development effort. Because the additional cost associated with an independent test team is significant, its use is recommended only for very large projects or projects with high reliability requirements. For some projects with extreme reliability requirements, the effort expended by the independent test team may equal that of the development team.

3.2.4 ANALYSIS SUPPORT

The development team may require support from other groups during software development, especially from analysts with a clear understanding of the problem. This support includes requirements analysis, functional specifications development, acceptance testing, data simulation, and requirements clarification (during design and implementation). Table 3-5 shows the approximate cost of analysis support activities as a percentage of the basic software development cost.

Table 3-5. Cost of Analysis Support

Support Type	Cost[a]
Requirements Specification[b]	25
Data Simulation	5
Acceptance Testing	5
Requirements Clarification[c]	10

[a]Percent of basic development cost.

[b]Prior to start of requirements analysis.

[c]During development.

These are activities that ensure that the developers understand the problem and obtain a correct solution. Requirements specification includes those analyst activities that produce a definition of the problem prior to the start of development. During the software development process, questions about the requirements may arise that result in requirements clarification. Analysts may also be required to produce simulated data for software testing by the developers. Finally, the analysts participate in the acceptance testing and evaluate its results.

3.3 OTHER SOFTWARE ACTIVITIES

A substantial amount of effort goes into software activities other than developing new systems (Sections 3.1 and 3.2). These activities include modifying existing software systems to operate on new hardware systems and maintaining existing software systems after development is complete. A related issue is the effect of reusing existing code in a new system. These topics are discussed in the following sections.

3.3.1 SOFTWARE REHOSTING

The cost of modifying existing software to operate on a new computer system depends on the nature of the software and differences between the computer systems. Software developed in a high-level language is more easily transported than assembler programs (which may have to be completely rewritten). Table 3-6 identifies the approximate cost of rehosting a high-level language in several situations. The cost is expressed as a percentage of the original development cost in staff-hours.

Table 3-6. Cost of Rehosting FORTRAN Software

System's Relationship	Relative Cost[a]	Testing Effort[b]	New Code[c]
Compatible[d]	15-21	67-70	0-3
Similar[e]	22-32	61-66	4-14
Dissimilar[f]	33-50	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]Basic architectural similarities (e.g., same word size and instruction set).

[f]Basic architectural differences (e.g., differing word sizes and instruction sets).

The table also shows that the percentage of development effort spent in testing is substantially greater for rehosting projects than the 30 percent spent in new development projects. The "new code" column in Table 3-6 lists the percentage of new and extensively modified code expected to be produced. The number of new and old lines can be combined to determine the number of "developed" lines of code (see Equation (3-1)). This work measure can be converted into a cost estimate via the procedure discussed in Section 3.1.

3.3.2 SOFTWARE MAINTENANCE

Software maintenance includes all programming occurring after acceptance testing. It consists of two different activities. First, errors that were not detected during development must be corrected as they are exposed during operation. Second, capabilities may be added and enhancements made to satisfy new or newly recognized needs. The cost of each of these software maintenance activities can be estimated separately.

Estimates of the cost of error correction must be based on an estimate of the reliability of the delivered software. Estimates of additional development costs depend on the expected length of the software's operational life and the importance of the system to the user organization.

Limited SEL experience indicates that 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.

3.3.3 SOFTWARE REUSE

An effective way to reduce the cost of software development is to reuse appropriate software components previously developed for other systems. Rehosting (Section 3.3.1) is an example of software reuse in which almost all the code is reused.

A careful review of available software during the design phase of a project often identifies reusable modules. Mathematical procedures are especially easy to reuse. Input and output procedures are usually idiosyncratic. However, applying the principles of modularity and structure during the initial development of a module facilitates its later reuse (Reference 17). Table 3-7 shows the relationship between the amount of code modified and the cost to reuse that module.

A significant saving can also be realized by reusing parts of a system's design. Effort and time can be reduced during design by adapting a similar and already proven design from a previous system.

Table 3-7. Cost of Reusing Software

Module Classification[a]	Percent of Code Modified or Added[b]	Relative Cost[c]
New	100	100
Extensively Modified	>25	100
Slightly Modified	1-25	20
Old	0	20

[a]Names used in Sections 3 and 4.

[b]Percent of module's code.

[c]Cost as a percent of the cost of developing a new module.

SECTION 4 - A COMPREHENSIVE SOFTWARE COST ESTIMATION PROCEDURE

This section describes the application in the flight dynamics environment (see Appendix) of the software cost estimation principles and guidelines presented previously. It assigns numerical values to some of the factors described in Section 3.1 that affect the basic cost of software development. It also shows how these estimates are used to develop schedules and staffing plans.

A series of tables condense the relevant historical data into a form useful to the experienced manager. The numerical quantities employed are specific to the flight dynamics environment. Although this procedure is generally applicable, appropriate values for all table entries must be rederived for each new environment to which the procedure is adapted.

The procedure presented in this section can be used throughout the development cycle of new software. However, the number and detail of estimators available varies from phase to phase (Table 4-1). Initial estimates are made as described in Section 3.1.1. Subsequent reestimates incorporate the most detailed information available at the time (see Section 3.1.2).

There are four steps in the estimation and planning of software development:

- Estimating the size of the software product
- Converting that estimate to an estimate of development effort
- Defining a feasible development schedule
- Determining the development staff required to complete the project on schedule

Table 4-1. Effort Estimators and Uncertainty Limits by Phase

End of Phase	Effort Estimators	Estimate Uncertainty[a] (Proportion)
Requirements Definition	Similar projects, general subsystems	1.00
Requirements Analysis	General subsystems, special capabilities	0.75
Preliminary Design	Specific subsystems, expenditures to date	0.50
Detailed Design	Actual subsystems, modules, code, documentation, expenditures to date	0.30
Implementation	Modules, code, tests, documentation, expenditures to date	0.12
System Testing	Code, tests, documentation, expenditures to date	0.05

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

The following sections describe these four steps as they are applied in the SEL environment. The development organization produces software systems for a broad application of related scientific, data base, and data support systems. New applications (project types) and computing facilities (environment types) are introduced every few years. The development organization develops systems for both the old and new computing facilities and for the same and different, but usually related, applications. Therefore, in general, a moderate amount of code is reused from project to project.

4.1 SOFTWARE SIZE

At each phase of the development life cycle, the development organization extracts essential parameters and combines them with archived information (see Table 3-2) to produce an estimate of the system's size as described in Section 3.1. The work unit used in this estimation procedure is lines of executable code. Executable code makes up about 40 percent of total lines of code. Usually, a substantial amount of software is adapted from previous projects. The proportion of reused code should be estimated as part of the software size estimation step. Equation (4-1) determines the developed lines of executable code (L), a weighted measure of system size:

$$L = N + 0.2R \qquad (4\text{-}1)$$

where N is the amount of new and extensively modified code (in thousands of lines), and R is the amount of code reused with little or no change (in thousands of lines). This is the estimate of work upon which subsequent estimation steps are based.

4.2 DEVELOPMENT EFFORT

The next step after estimating size is to convert this into an estimate of development effort. Equation (4-2) predicts the effort required for complete development:

$$E = 8.45\ f_1 f_2 f_3 \ldots f_k\ L^{1.05} \tag{4-2}$$

where E is effort in staff-months (156 staff-hours per staff-month), L is thousands of developed lines of executable code (see Equation (4-1)), and the f_i's are project-specific factors that increase or decrease required effort. Sections 2.2 and 3.1 identified many such factors. This section assigns numerical values to the most important of them. (The constants 8.45 and 1.05 are empirical values derived from SEL data.)

The manager needs, as a minimum, to consider the complexity of the problem, the development team's experience, and the schedule. As the development organization becomes more knowledgeable about itself, other factors (such as methodology usage) can be added to fine-tune the estimation process.

Table 4-2 provides the development organization with a simple guideline for adjusting effort with a complexity factor. This table defines four complexity classes based on the development team's familiarity with the application and environment. Table 4-3 provides a simple guideline for adjusting effort with a team experience factor. This factor is measured in terms of years of applicable experience. Table 4-4 provides a simple guideline for adjusting effort with a schedule factor. This table identifies three schedule types; Section 4.3 explains how schedule type is determined.

Table 4-2. Complexity Guideline

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

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

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

Table 4-3. Development Team Experience Guideline

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

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

Table 4-4. Schedule Guideline

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

As an illustration, assume that the development organization must develop a system estimated at 25,000 executable LOC. It is similar to ones developed before (old project type), but it is being developed for a new computing facility (new environment type) (f_1 = 0.65). Fifty percent of the code can be reused without modification (Equation (4-1), N = R = 12.5). The development organization has a team in mind whose weighted applicable experience is 6 years (f_2 = 0.8) and the luxury of a slow schedule (f_3 = 0.85). Then, assuming that all other factors are normal (f_i = 1, for i = 4 to k), the estimated effort is 64.1 staff-months. This includes the cost of developing informal user documentation (as defined in Section 3.3.2).

Since it is the beginning of the project, there is an uncertainty limit of ±100 percent in the size estimate. Therefore, project cost will range between 32.0 and 128.2 staff-months. (See Section 2.4 for a detailed discussion of estimate accuracy and ranges of uncertainty.)

4.3 DEVELOPMENT SCHEDULE

The time required to complete a software project depends on the staff and software sizes. To some extent, a larger staff can be used to shorten development time. Table 4-4, however, shows that a faster schedule is more costly. The development time is usually fixed by a specified delivery date for the software. The tightness of a schedule can be determined by calculating the work rate as shown in Equation (4-3).

$$W = L/T \tag{4-3}$$

where W = work rate

L = thousands of developed lines of executable code (see Equation (4-1))

T = development time (duration in weeks)

Table 4-5 presents a classification of schedule difficulty based on work rate. The schedule type of a project is incorporated in the effort estimation step described in Section 4.2. Once the total schedule length has been determined, phase transition dates must be defined. Table 4-6 gives the distributions of time and effort for planning the life cycle for flight dynamics projects.

Table 4-5. Determination of Schedule Type From Work Rate[a]

Complexity	Schedule Type		
Class[b]	Fast	Average	Slow
Old/Old	>0.24	0.24-0.16	<0.16
Old/New	>0.17	0.17-0.10	<0.10
New/Old	>0.17	0.17-0.10	<0.10
New/New	>0.11	0.11-0.07	<0.07

[a]Thousands of developed lines of executable code per week as defined in Equation (4-3).

[b]Project/environment pairs defined in Table 4-2.

Table 4-6. Distribution of Development Effort

Phase	Percent of 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

4.4 PROJECT STAFFING

Once the effort for each phase is known, the development manager must determine how fast the development team can be staffed, how large it can get, and how soon team members can be released without disturbing discipline and order. This determination has to be made based on the project leader's experience level. In this example, Table 4-7 provides the development organization with a simple guideline for determining team size in terms of the experience of the team leaders. Table 4-8 provides a simple guideline for planning changes in staffing level.

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

Table 4-7. Team Size Guideline

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

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

Org. = Experience with organization.

Leader = Experience as team leader or manager.

Table 4-8. 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

8849/82

[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 ADDITIONS 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.

because each smaller project will have a project leader and its own reporting and support requirements.

In this environment, Table 4-9 provides the development organization with a simple guideline for determining the type and experience level of technical personnel needed in terms of the complexity of the problem. Other personnel types must also be considered. Management charges usually amount to 10 percent of development charges. Administrative and publications support usually amounts to 6 percent of development charges.

Table 4-9. Development Team Staffing Guideline

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

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

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

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

SECTION 5 - SUMMARY AND RECOMMENDATIONS

The preceding sections reviewed the state of the art and described a general procedure for software cost estimation. Accurate estimates are the result of careful evaluation and planning. The following points are especially important to bear in mind:

- A regular program of data collection is necessary to supply the historical data essential to software cost estimation. Reference 18 describes the comprehensive program of data collection employed by the SEL.

- A software life cycle must be defined at the start of a project. Reestimation must be performed throughout the life cycle to incorporate the additional information that becomes available.

- A software development plan is essential to effective software cost estimation. This plan should contain the schedule and staffing details needed to measure the progress of the development effort. Estimates must be updated as the schedule and staffing assumptions on which they are based change.

- An early understanding of the size and complexity of a software development problem is essential to effective planning and estimation.

- The basic prerequisites for estimate accuracy are a detailed decomposition, careful analysis, and good historical data.

The procedure explained in this document provides the software development manager with an effective tool for cost estimation. Careful attention to the points made above will maximize the accuracy and information obtained from this estimation procedure.

APPENDIX - SEL SOFTWARE COST ESTIMATION EXPERIENCE

This appendix describes the nature of the software applications and environment (flight dynamics) that is the basis of SEL software cost estimation experience. The numerical values and relationships reported in this document were derived from data collected by the SEL from flight dynamics projects at GSFC.

Most flight dynamics projects are developed on a group of IBM mainframe computers using FORTRAN and assembly programming languages. Table A-1 summarizes the important characteristics of the flight dynamics environment. Specific software applications include attitude determination, attitude control, maneuver planning, orbit adjustment, and general mission analysis. The attitude systems, in particular, form a large and homogeneous group of software that has been studied extensively.

The attitude determination and control systems are designed similarly for each mission using a standard executive support package, the Graphic Executive Support System (GESS), as the controlling system. All of these systems are designed to run in batch and/or interactive graphic mode. Depending on mission characteristics, the size of the systems may range from approximately 30,000 to 120,000 lines of code (LOC). Some existing software can be reused in new systems when the new system is similar to previous systems. The percentage of reused code ranges from 10 percent to nearly 70 percent, with most systems reusing about 30 percent.

The applications developed in the flight dynamics environment are largely scientific and mathematical in nature, with moderate reliability requirements. Severe development time constraints are imposed by the fixed spacecraft launch date.

Table A-1. Flight Dynamics Environment

Type of Software: Scientific, ground-based, interactive graphic with moderate reliability and response requirements

Languages: 85 percent FORTRAN, 15 percent assembler macros

Machines: Primarily IBM (S/360 and 4341) and DEC (PDP-11 and VAX-11)

Project Characteristics	Average	High	Low
Duration (months)	15.6	20.5	12.9
Effort (staff years)	8.0	11.5	2.4
Size (1000 LOC)			
Developed	57.0	111.3	21.5
Delivered	62.0	112.0	32.8
Staff (full-time equivalent)			
Average	5.4	6.0	1.9
Peak	10.0	13.9	3.8
Individuals	14	17	7
Application Experience (years)			
Managers	5.8	6.5	5.0
Technical Staff	4.0	5.0	2.9
Overall Experience (years)			
Managers	10.0	14.0	8.4
Technical Staff	8.5	11.0	7.0

The development process normally begins approximately 12 to 24 months before a scheduled launch. The software must be completed (through acceptance testing) 90 days before the scheduled launch.

The SEL has monitored the development of more than 45 such flight dynamics projects during the past 7 years. Table A-2 reports the nominal values of some software cost estimation parameters extracted from SEL historical data. Most of these were discussed in detail in Sections 3 and 4. The SEL is described in more detail in Reference 1.

Table A-2. Summary of Cost Estimation Parameters

Parameter	Nominal Value	Discussion[a]
Lines of code per subsystem	7500	3.1
Staff hours to develop a subsystem	1850	3.1
Lines of code per module	125	3.1
Staff hours to develop a module	30	3.1
Staff hours to develop a line of code	0.3	3.1
Computer hours per line of code	0.009	3.2.1
Pages of documentation per line of code	0.04	3.2.2
Cost of independent test team	15[b]	3.2.3
Cost of analysis support	45[b]	3.2.4
Cost to rehost software to compatible computer	18[b]	3.3.1
Cost to rehost software to similar computer	27[b]	3.3.1
Cost to rehost software to dissimilar computer	42[b]	3.3.1
Annual cost of maintenance	20[b]	3.3.2
Percent of code reused	30	3.3.3
Relative cost of reusing software	20	3.3.3
Executable lines as a percent of total source lines	40	4.1
Percent of effort by programmers	84	4.4
Percent of effort by managers	10	4.4
Percent of effort by support staff	6	4.4

[a] Section of this document where discussed.

[b] Percent of basic software development cost.

REFERENCES

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

2. --, SEL-81-205, Recommended Approach to Software Development, F. E. McGarry, G. Page, S. Eslinger, et al., March 1983

3. Computer Sciences Corporation, "Early Estimation of Resource Expenditures and Program Size," D. N. Card, Technical Memorandum, June 1982

4. A. J. Albrecht and J. E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Transactions on Software Engineering, November 1983

5. Software Engineering Laboratory, SEL-81-105, Glossary of Software Engineering Laboratory Terms, T. A. Babst, M. G. Rohleder, and F. E. McGarry, October 1983

6. F. B. Brooks, The Mythical Man-Month. Reading, Massachusetts: Addison-Wesley Publishing Co., 1975

7. R. C. Tausworthe, "Staffing Implications of Software Productivity Models," Proceedings of the Seventh Annual Software Engineering Workshop, December 1982

8. R. W. Wolverton, "The Cost of Developing Large Scale Software," IEEE Transaction on Computers, June 1974

9. L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, July 1978

10. Software Engineering Laboratory, SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

11. J. W. Bailey and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceeding of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

12. B. Boehm, Software Engineering Economics. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981

13. Quantitative Software Management, Inc., SLIM, 1979

14. Radio Corporation of America, PRICE S, 1979

15. D. N. Card, F. E. McGarry, and G. Page, "Evaluating Software Engineering Technologies in the SEL," Proceedings of the Eighth Annual Software Engineering Workshop, November 1983

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

17. B. Meyer, "Principles of Package Design," ACM Communications, vol. 25, no. 7, July 1982

18. Software Engineering Laboratory, SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

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

[1]SEL-78-002, FORTRAN Static Source Code Analyzer (SAP) User's Guide, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

SEL-78-102, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 1), W. J. Decker and W. A. Taylor, September 1982

SEL-78-003, Evaluation of Draper NAVPAK Software Design, K. Tasaki and F. E. McGarry, June 1978

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

[1]SEL-80-004, System Description and User's Guide for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, W. J. Decker, J. G. Garrahan, et al., October 1980

SEL-80-104, 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

[1]SEL-81-001, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., September 1981

SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

[1]SEL-81-002, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, D. C. Wyckoff, G. Page, and F. E. McGarry, September 1981

SEL-81-102, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide Revision 1, P. Lo and D. Wyckoff, July 1983

[1]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

[1]SEL-81-004, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., September 1981

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

[1]SEL-81-005, Standard Approach to Software Development, V. E. Church, F. E. McGarry, G. Page, et al., September 1981

[1]SEL-81-105, Recommended Approach to Software Development, S. Eslinger, F. E. McGarry, and G. Page, May 1982

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

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

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker and F. E. McGarry, March 1981

[1]SEL-81-010, Performance and Evaluation of an Independent Software Verification and Integration Process, G. Page and F. E. McGarry, May 1981

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 1, July 1982

[1]SEL-82-005, Glossary of Software Engineering Laboratory Terms, M. G. Rohleder, December 1982

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

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

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-RELATED LITERATURE

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," Program Transformation and Programming Environments. New York: Springer-Verlag, 1984

[3]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Banks, F. K., "Configuration Analysis Tool (CAT) Design," Computer Sciences Corporation, Technical Memorandum, March 1980

[3]Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., "SEL Relationships for Programming Measurement and Estimation," University of Maryland, Technical Memorandum, October 1979

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: Computer Societies Press, 1980 (also designated SEL-80-008)

[3]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

[3]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

[2]Basili, V. R., and B. T. Perricone, Software Errors and Complexity: An Empirical Investigation, University of Maryland, Technical Report TR-1195, August 1982

[3]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

[2]Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," IEEE Transactions on Software Engineering, 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

[2]Basili, V.R., and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, University of Maryland, Technical Report TR-1235, December 1982

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

[3]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

[3]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: Computer Societies Press, 1978

[3]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

[2]Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

[2]Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

Card, D. N., and V. E. Church, "Analysis Software Requirements for the Data Retrieval System," Computer Sciences Corporation Technical Memorandum, March 1983

Card, D. N., and V. E. Church, "A Plan of Analysis for Software Engineering Laboratory Data," Computer Sciences Corporation Technical Memorandum, March 1983

Card, D. N., and M. G. Rohleder, "Report of Data Expansion Efforts," Computer Sciences Corporation, Technical Memorandum, September 1982

[3]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

[2]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

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

[3]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

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," Empirical Foundations for Computer and Information Science (proceedings), November 1982

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

[1]This document superseded by revised document.

[2]This article also appears in SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983.

[3]This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume I, July 1982.