Copy #2

CSC/TM-82/6033

THE SOFTWARE ENGINEERING LABORATORY

Prepared For
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Goddard Space Flight Center
Greenbelt, Maryland

CONTRACT NAS 5-24300
Task Assignment 936

FEBRUARY 1982

(NASA-CR-183441) THE SOFTWARE ENGINEERING LABORATORY Final Report (Computer Sciences Corp.) 122 p
N89-71121
Unclas
00/61 0224550

CSC

COMPUTER SCIENCES CORPORATION

THE SOFTWARE ENGINEERING LABORATORY

Prepared for

GODDARD SPACE FLIGHT CENTER

By

COMPUTER SCIENCES CORPORATION

Under

Contract NAS 5-24300
Task Assignment 936

Prepared by:

David N. Card 2/10/82
D. N. Card Date

F. E. McGarry (GSFC)

Reviewed by:

2/10/82
W. J. Decker Date

Approved by:

S. Eslinger 2/10/82
S. Eslinger Date
Section Manager

Gerald Page 2-10-82
G. Page Date
Department Manager

2/10/82
S. E. Cheuvront Date
Operation Manager

PREFACE

This document is the final version of the document that was originally prepared as a preliminary draft (CSC/TM-81/6104). It incorporates the results of an extensive review by GSFC and CSC personnel. This document is also being issued as a volume in the Software Engineering Laboratory Series (SEL-81-104).

ABSTRACT

This document describes the history, organization, operation, and research results of the Software Engineering Laboratory (SEL). The SEL is a joint effort of the Goddard Space Flight Center (GSFC), Computer Sciences Corporation (CSC), and the University of Maryland to study and improve the software development process. The document discusses specific data collection and analysis activities and general considerations of motivation and approach.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF TABLES (Cont'd)

SECTION 1 - INTRODUCTION

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

To accomplish these goals, the SEL studies software for satellite mission support during its development life cycle. This software is developed by the Systems Development Section at NASA/GSFC, which is responsible for generating flight dynamics support software for GSFC-supported missions. The software includes attitude determination, attitude control, maneuver planning, orbit adjustment, and general mission analysis support systems.

The SEL continually monitors and studies all Systems Development Section software, which includes software developed both by GSFC employees and by contractor personnel.[1] This data covers software development projects that started as early as 1976 and as late as 1981; and the SEL anticipates that data will continue to be collected and studied in the future. Approximately 40 projects, which range in size from 1500 lines of source code to over 110,000 lines, have been involved to date.

---

[1] The primary On/Off-Site contractor supporting the flight dynamics area has been Computer Sciences Corporation (CSC).

All the projects being studied supported the flight dynamics area of GSFC's Mission Support Computing and Analysis Division. Much of the data is collected from a series of forms used by all projects. Data is also collected through computer accounting monitoring, personal interviews, automated tools, and summary management reviews (see Sections 2.2.1 through 2.2.5).

While investigating projects totaling more than 1 million lines of code, SEL personnel gained insight into the software development process and began to discern trends in the relative effects of various techniques applied to the software projects. This document

- Describes the motivation and background of the SEL
- Relates the concepts and activities of the SEL
- Summarizes the results of SEL research
- Reports the status, conclusions, and recommendations of the SEL

This document is not a general survey of software engineering literature. Rather, it is a survey of SEL research that only outlines the relationship of that work to the activities of other members of the software engineering community.

The document does not describe in detail all of the results thus far produced by the SEL. Other documents provide additional information about SEL activities. A previous SEL description was generated in 1977 (Reference 1), and numerous papers explain SEL research experiences with methodologies, models, and measures. A complete list of related documentation is given in the bibliography. These documents span the 5 years during which the laboratory has been in operation and provide useful reference material about the studies carried out by the SEL.

This document consists of the following sections:

- Section 1--A general overview of the SEL. Includes the motivation for the creation of the SEL, the areas of concern for the software developers at GSFC, and a description of the relevant environment.
- Section 2--A description of the overall operations of the SEL. Includes descriptions of the functional organization of the SEL, the data collection and validation process, the SEL data management approach, and the types of data analyses that are being performed with the existing data base and software.
- Section 3--A discussion of the experimental results. Includes the SEL's general findings to date. Organized into five topics: profiles, methodologies, models, tools, and measures.
- Section 4--A summary of the status of the SEL's activities. Includes conclusions and recommendations.
- Appendix A--Detailed tabulations of SEL resource, change/error, component, and computer utilization data.
- Appendix B--Summaries of software development projects studied by the SEL. Includes resources, software, and environmental characteristics. Experimental objectives are also identified.

In addition to these six main sections, the document also contains a glossary of acronyms and abbreviations used in the document, references, and a bibliography.

1.1 THE SEL APPROACH

Extensive efforts have been made during recent years to devise improved software development techniques. This work generated numerous tools (e.g., precompilers and programmer workbenches), cost and reliability models, and methodologies (e.g., structured programming and top-down design); all were supposed to improve the development process. Early evaluations of the effectiveness of these techniques were incomplete and/or inconclusive. This may have been due, in part, to an unrealistic assumption that the software development process could be isolated from the environment in which it occurs. However, no element of the development process can be understood outside the context of related factors.

For example, productivity in some environments may be constrained by staffing patterns. Thus, the possible beneficial effect of a productivity-enhancing methodology may remain unrealized and unrecognized because of an inappropriate allocation of manpower.

The SEL approach to software engineering research is holistic. Figure 1-1 shows the SEL high-level software development model. Its four components are a problem statement, an environment, a process or activity, and a product (software). The development process is subdivided into seven sequential phases of activity. This model is elaborated upon elsewhere (Reference 2). A goal of the SEL, then, is to refine the definitions of the model elements and to define their relationships.

The first step toward this goal is to understand the software development process currently in operation and its environment. Important attributes of the software problem and products must also be investigated. Such an understanding provides a baseline from which the effects of attempted improvements can be measured and allows the identification

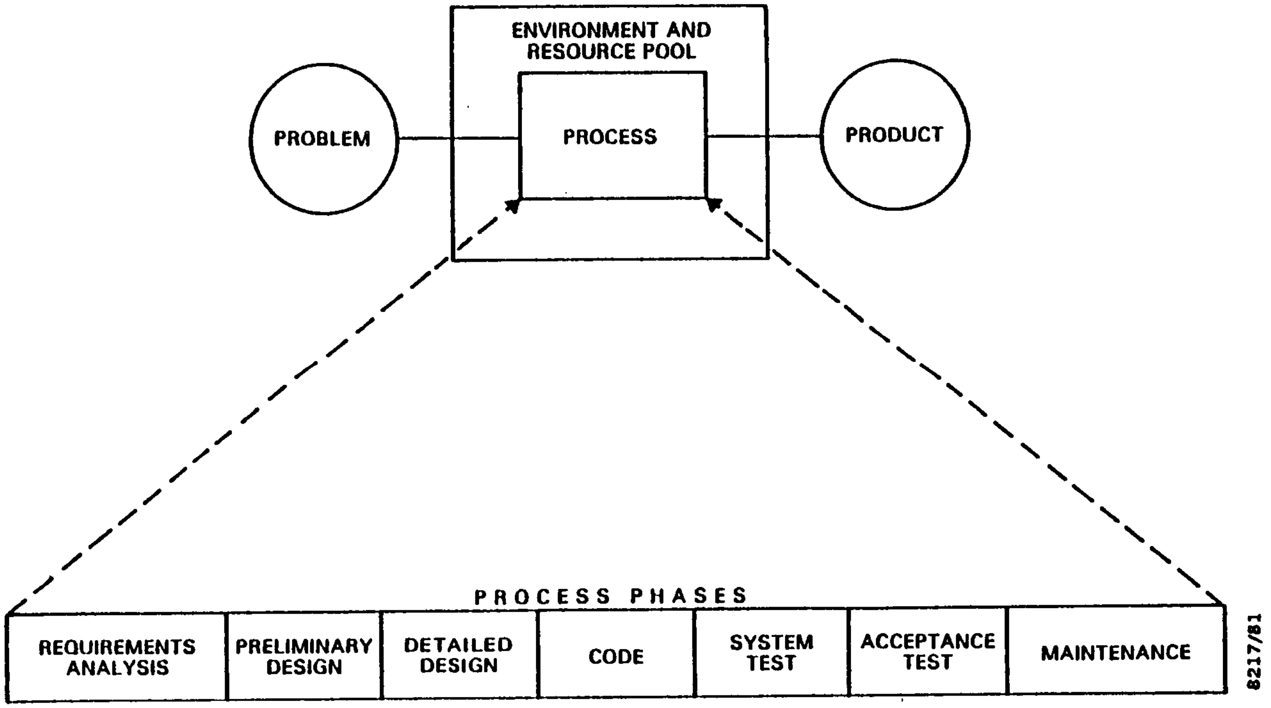


Figure 1-1. Software Development Model

of strengths and weaknesses so that efforts can be focused on the areas of greatest need.

Beyond understanding the current process and environment, the SEL is interested in improving that process and environment. The SEL recognizes a four-step procedure leading to more effective software development. The steps are to

- Become aware of the development techniques available
- Evaluate the available techniques to determine those most effective
- Engineer (customize) those "best" techniques to perform optimally in the user's environment
- Apply the customized techniques

This procedure can become the basis of a regular system of self-evaluation and improvement, whereby as new techniques become available, they are tested and incorporated in the software development process.

The SEL maintains contact with other software engineering research efforts through its sponsorship of annual workshops and its association with the Department of Computer Science at the University of Maryland. New ideas and techniques are constantly being introduced for consideration.

1.2 AREAS OF CONCERN

The current store of knowledge about software development that can be called scientific is still relatively limited. However, a multitude of software development technologies have been established on this small foundation. For the SEL's purposes, technologies are classified into three major areas of concern: methodologies, models, and tools.

Methodologies are systematic applications of prescribed principles to the development process. These principles may pertain to requirements, design, code, test, or management. Examples include structured analysis, top-down design, information hiding, structured programming, formal test plans, and configuration management.

Models attempt to explain and/or predict some aspect of the behavior of the development process. They are usually formulated as mathematical equations (or sets thereof) that relate two or more quantitative factors. Models are frequently useful to management. A resource utilization model may provide an estimate of the cost of a project; a reliability model may indicate when sufficient testing has been done.

Tools are software aids utilized during the development process to facilitate the work of development team members. Some examples are requirements language processors, precompilers, code auditors, and test generators. These are often packaged into a programmer workbench system (see Section 3.4).

The maximum benefit may be derived by applying several of these techniques to a software development project. The rational application of these modern programming and management practices has become known as "software engineering." It is a scientific approach to software development that attempts to incorporate the structure and discipline that underlie more traditional engineering activities. The expected result of such an approach is the production of high-quality software with fewer errors at a lower cost. However, a prerequisite to the application of software engineering techniques is the determination of the effectiveness of the available technologies within the target environment.

Section 3 contains detailed evaluations of the methodologies, models, and tools in the software engineer's repertoire as they have been employed in the GSFC environment. The next subsection discusses the specific objectives of the SEL.

1.3 OBJECTIVES OF THE SEL

The overall objective of the SEL is to understand the software development process and the ways in which it can be altered to improve the quality and to reduce the cost of the product. However, the SEL has defined some intermediate objectives within the previously defined areas of concern that will help meet that general goal. These objectives fall into two classes: experimentation and communication.

Experimentation involves evaluating existing software development technologies (previously defined in Section 1.2) and developing new technologies. Specific objectives of the SEL are to

- Conduct controlled experiments
- Evaluate software development methodologies
- Evaluate software development tools
- Analyze cost estimation models
- Analyze software reliability models
- Develop a set of software quality metrics

The results of experimentation must be incorporated in the software development process to improve it. This process requires communication between researchers and developers. Specific communications objectives of the SEL are to

- Devise software development standards
- Develop software management guidelines
- Provide real-time feedback to development teams being monitored

- Maintain contact with the software engineering research community

Clearly, the objectives of the SEL reflect its multistep approach to software engineering, as described in the previous sections. Section 1.4 describes the environment in which the SEL works to achieve those objectives.

1.4 FLIGHT DYNAMICS ENVIRONMENT

The development environment must be clearly understood to evaluate any software development approach effectively. This subsection describes the development environment of the projects studied by the SEL. The discussion is divided into three sections: the development organization, hardware resources, and characteristics of the software developed.

1.4.1 FLIGHT DYNAMICS ORGANIZATION

The data used in the analyses described in this document was collected from software development projects within the flight dynamics area of NASA/GSFC, under the supervision of the Systems Development Section. Most of the software development effort was provided by an independent contractor, although at times GSFC personnel participated in development. This subsection outlines the organization of a development team composed of GSFC and contractor personnel.

The members of a team supporting a typical software development project and their duties are identified in Table 1-1. This team includes managers, programmers, and librarians.

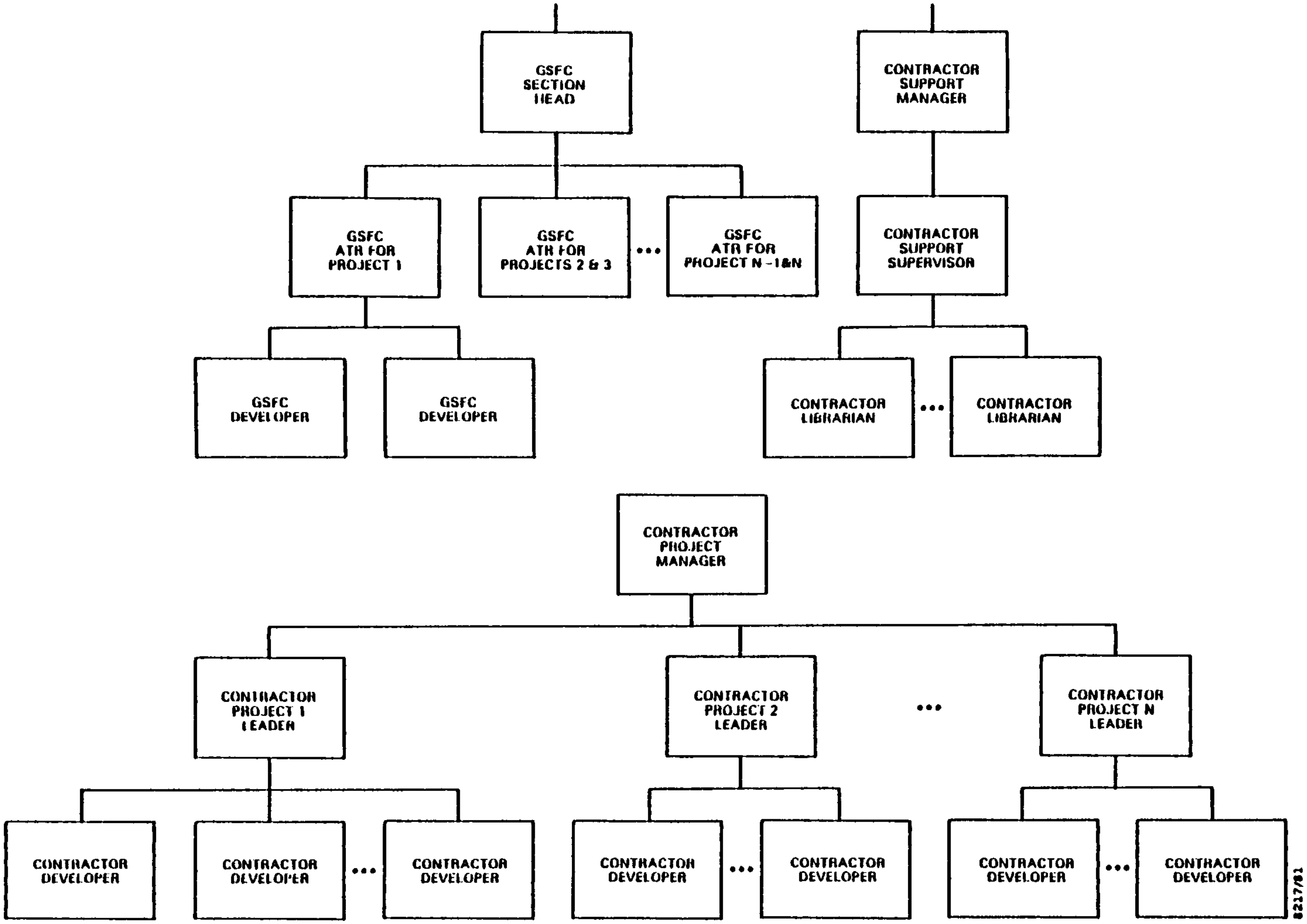
Figure 1-2 illustrates the organization of a development team. The interactions of the members of a development team with the SEL are shown in Figure 1-3. The organization of the SEL is explained in Section 2.1.

Table 1-1. Flight Dynamics Development Team

PERSONNEL	ORGANIZATION	INVOLVEMENT (%)	YEARS OF EXPERIENCE	FUNCTION
ASSISTANT TECHNICAL REPRESENTATIVE	GSFC	15-100	3-12	MONITORS CONTRACTED PROJECT; SUPERVISES GSFC DEVELOPERS
PROJECT MANAGER	CONTRACTOR	20-100	8-16	MANAGES PROJECT RESOURCES; PROVIDES TECHNICAL CONSULTATION
PROJECT LEADER	CONTRACTOR	100	4-12	SUPERVISES CONTRACTOR DEVELOPERS; PARTICIPATES IN DEVELOPMENT
CONTRACTOR DEVELOPER	CONTRACTOR	100	2-12	DESIGNS, IMPLEMENTS, TESTS, AND DOCUMENTS SOFTWARE
GSFC DEVELOPER	GSFC	100	1-10	DESIGNS, IMPLEMENTS, TESTS, AND DOCUMENTS SOFTWARE
LIBRARIAN	CONTRACTOR[1]	10	1-5	MAINTAINS SOFTWARE LIBRARIES; ENTERS PROGRAM SOURCE CODE

8217/82

[1] NOT NECESSARILY THE SAME CONTRACTOR AS FOR OTHER PERSONNEL.



NOTE: N = 2, 3, ..., 7 FOR ONE CONTRACTOR MANAGER. NUMBER OF PERSONNEL UNDER CONTRACTOR MANAGER RANGES FROM 10 TO 25.

Figure 1-2. Software Development Team Organization

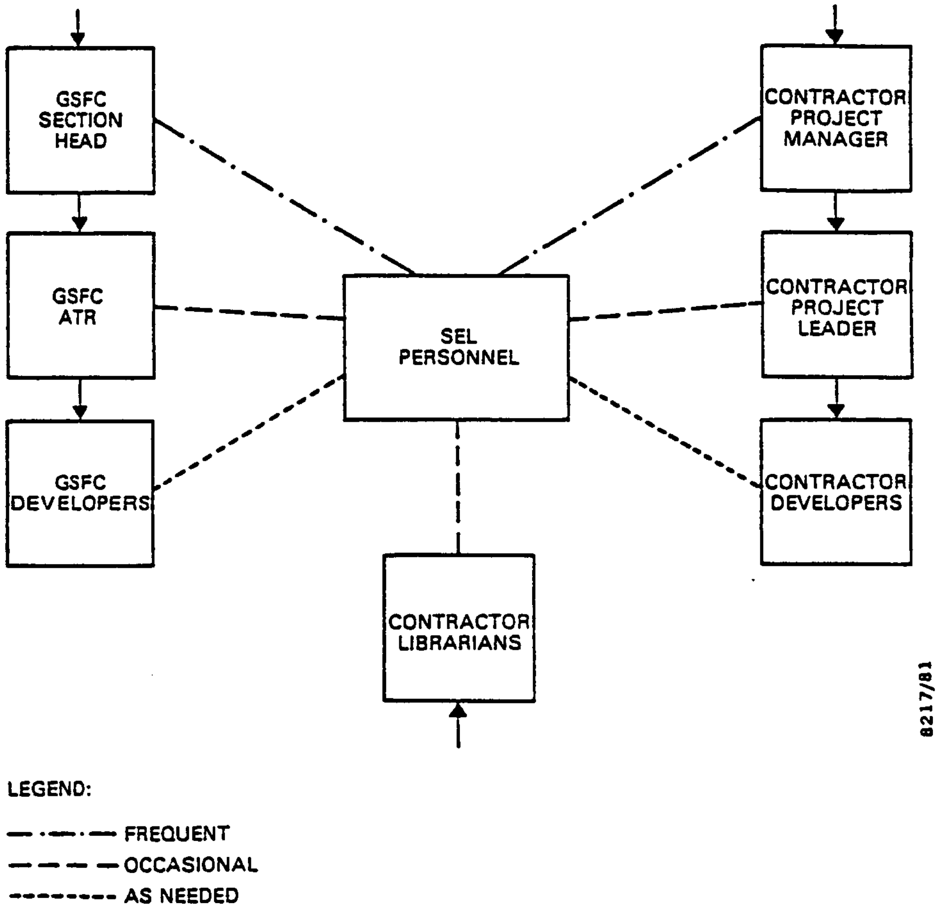


Figure 1-3. Development Team Interactions With SEL Personnel

1.4.2 HARDWARE RESOURCES

The development hardware has remained fairly constant from project to project during the past 5 years of SEL activity. These computers are listed in Table 1-2. The primary development and operations equipment is a group of IBM S/360 computers. The machine that supports most development activity is the S/360-95. However, development projects also use the S/360-75, primarily for graphics system testing. In addition to the IBM S/360s, a DEC PDP-11/70 and a DEC VAX-11/780 are occasionally used to develop utilities and support systems for the flight dynamics area.

Table 1-2. Flight Dynamics Computers

Computer	Operation	Memory (Bytes)
IBM S/360-95	Batch	5000K
IBM S/360-75	Batch	3000K
DEC VAX-11/780	Interactive	1000K[a]
DEC PDP-11/70	Interactive	756K

---

[a] Virtual memory space of 4 gigabytes.

Both the S/360-95 and -75 are primarily batch loaded. However, the S/360-75 is card deck oriented, whereas the S/360-95 receives a large part of its work via timesharing option (TSO) submittal. The primary language used by the local software community is FORTRAN or a locally developed structured variant of FORTRAN called SFORT. Usage of Assembly Language Code (ALC) and other languages is limited to special applications.

Various devices are available for user storage of software libraries. Mountable disk and magnetic tape can be used to store source code, load modules, and data in general. Online disk space for general users is very limited.

Although the S/360-95 has 5 million bytes of main memory, special requirements and daily operational support activities reduce the memory available to the general user to about 2 million bytes. This machine has nearly 1000 registered users contending for service.

Because machines are shared among the analysis, software development, and operations areas, software development schedules are affected when simulations, launches, and maneuvers occur. During these times, the operations and analysis areas often require all available resources.

For all system testing and diagnostic testing involving graphics capabilities, the developer must schedule time on one of the computers in order to gain access to one of the graphic devices (such as an IBM 2250). Although cathode ray tubes (CRTs) are available continuously for editing or job submittal, only a few true graphic devices (i.e., those having vector generation capabilities) are available for system development.

1.4.3 SOFTWARE CHARACTERISTICS

The general category of flight dynamics software includes applications to support attitude determination, attitude control, maneuver planning, orbit adjustment, and general mission analysis. Most of these programs are scientific and mathematical in nature. The attitude systems, in particular, are 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.

Typically, attitude systems read sensor measurements from a telemetry stream and determine the attitude of the spacecraft from this data. Depending on the types of data available and the accuracies required, the size of these systems may range from 30,000 lines of code to about 120,000 lines of code. All these systems are designed to run in batch and/or interactive graphic mode. Some existing software can be reused for each new system, since there are always some similarities to past systems, especially in the high-level design. The percentage of reused code ranges from 10 percent to an upper limit of nearly 70 percent.

All applications developed in the flight dynamics area of GSFC have development time constraints corresponding to launch dates. Most of the software discussed in this document must be completed (implying completion of acceptance testing) 60 days before the scheduled launch. If the software is not completed, required capabilities must be deleted or redefined, and an alternate version of the intended system must be defined to ensure that the mission can be supported in some limited fashion.

The development process normally begins approximately 16 to 24 months before a scheduled launch in order to be completed two months in advance of launch. This development period is divided into phases as shown in Table 1-3.

Table 1-3. The Development Cycle

Development Phase	Time (Months)
Design	4-8
Requirements Analysis	1-3
Preliminary Design	1-2
Detailed Design	2-3
Code and Unit Test	6-8
Testing	4-6
System Testing	2-3
Acceptance Testing	2-3

SECTION 2 - SEL OPERATIONS

The SEL is involved in many aspects of software engineering research. However, the ultimate goal of the SEL is the actual application of improved techniques to the software development process. The prerequisites for achieving this goal (as described in Section 1.2) are the evaluation of available software development techniques and the customization of them to fit the GSFC environment. Evaluation and customization are analytical activities requiring the collection, validation, and management of data. On the other hand, application is promoted by management and training. The following subsections describe the SEL organization and its relationship to software development management, as well as the data collection and analysis activities of the SEL.

2.1 SEL ORGANIZATION

This subsection describes the general organization of the SEL and its relationship to software development management. Participants in the SEL include the following types of personnel:

- Managers
- Programmers
- Data base administrator
- Data technicians
- Researchers

The organizational structure of the SEL is illustrated in Figure 2-1. The activities corresponding to the roles defined in that figure are described in Table 2-1. The interaction of the SEL with members of software development teams is shown in Figure 1-3.

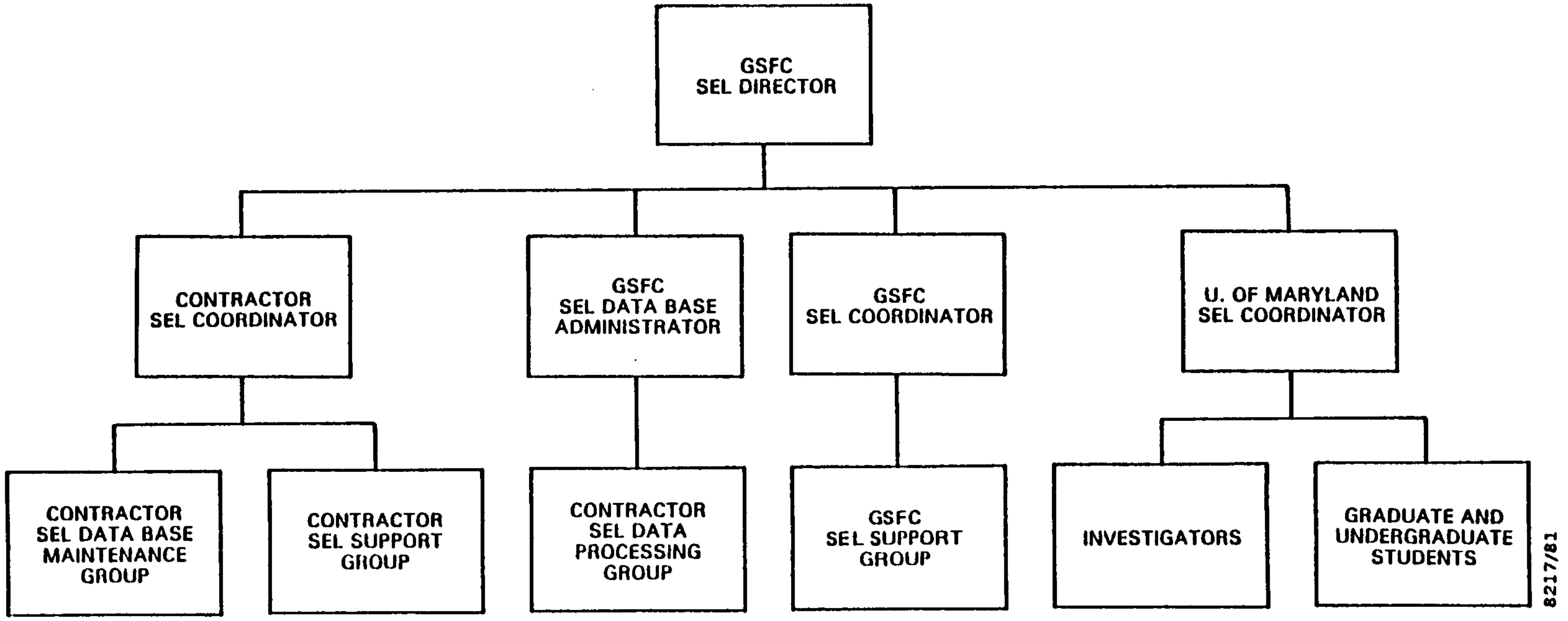


Figure 2-1. SEL Organization

Table 2-1. SEL Personnel

PERSONNEL	ORGANIZATION	FUNCTION
SEL DIRECTOR	GSFC	DETERMINES DIRECTION AND SETS PRIORITIES FOR THE ENTIRE SOFTWARE ENGINEERING RESEARCH EFFORT
CONTRACTOR COORDINATOR	CONTRACTOR	DIRECTS THE EFFORTS OF THE DATA BASE MAINTENANCE AND ANALYTICAL SUPPORT GROUPS; MANAGES THE INTERFACE WITH SOFTWARE DEVELOPERS
GSFC COORDINATOR	GSFC	DIRECTS THE EFFORTS OF GSFC PERSONNEL ASSOCIATED WITH THE SEL
UM COORDINATOR	UNIV. OF MD.	COORDINATES THE SEL-RELATED ACTIVITIES OF STUDENTS AND INVESTIGATORS AT THE UNIVERSITY OF MARYLAND
DATA BASE ADMINISTRATOR	GSFC	SUPERVISES DATA PROCESSING PERSONNEL; COORDINATES ACTIVITIES AFFECTING THE ORGANIZATION OF THE SEL DATA BASE
DATA PROCESSING GROUP	CONTRACTOR[1]	PERFORMS DATA ENTRY AND EDITING FUNCTIONS; LOGS, MICROFICHES, AND FILES PAPER RECORDS AND DATA FORMS
DATA BASE MAINTENANCE GROUP	CONTRACTOR	MAINTAINS DATA BASE SUPPORT SOFTWARE; DEVELOPS SPECIALIZED REPORTING AND VALIDATION SOFTWARE
ANALYTICAL SUPPORT GROUP	CONTRACTOR	PERFORMS DATA QUALITY ASSURANCE FUNCTIONS; PROVIDES PROGRAMMING, DOCUMENTATION, AND STATISTICAL SUPPORT TO ANALYSTS
COOPERATING RESEARCHERS	UNIV. OF MD.	ANALYZES SEL DATA AND REPORTS FINDINGS

8217/82

[1] NOT NECESSARILY THE SAME CONTRACTOR AS FOR OTHER PERSONNEL.

The SEL director and contractor coordinator are also involved in the management of the software development projects under study, frequently as the Assistant Technical Representative (ATR) and project manager, respectively (see Section 1.4.1). Combining the management of development projects and research activities into the joint roles of the SEL director and contractor coordinator facilitates the work of the SEL. Political and organizational conflicts between the two activities are avoided. Moreover, projects can be closely monitored to ensure that the appropriate experimental design is followed. Finally, techniques that have been proved effective can then be directly implemented without additional administrative complications.

2.2 DATA COLLECTION

The basis of software development research is the collection of experimental data. Data collection is a coordinated effort of applications programmers, associated management personnel, and library personnel. The responsibilities of each group are defined at the beginning of a project to ensure accurate, complete, and timely collection of information. Collected data is recorded on the SEL data base and in the SEL Central Library. The automated data base organization is discussed in Section 2.4. The central library contains the following items:

- All original software engineering forms
- Microfiche copies of forms
- Computer accounting sheets
- Copies of coded and validated forms
- Master resource summary, including plots and tables produced from forms and computer accounting sheets
- Weekly SEL data base activity summaries

- Documentation for all projects (including functional specifications and design documents)
- All paper records of data base transactions

Data collected by the SEL comes from five major sources: software engineering forms, computer accounting, personal interviews, automated tools, and management summaries. A general discussion of data collection procedures may be found in Guide to Data Collection (Reference 2). Each of the sources cited and the manner in which that data is collected are outlined in one of the following subsections.

2.2.1 SOFTWARE ENGINEERING FORMS

The primary medium for collecting pertinent information on software development is a series of data collection forms that are filled out by development team members. Forms are submitted by the developers, who provide detailed information; the managers, who provide summary information; and SEL personnel, who obtain accounting and source-code activity information. The SEL data collection forms were designed to allow the collection of data with the minimum impact on developers.

Seven basic types of form have evolved for use with the SEL. These forms are listed and described in Table 2-2. More detailed information about the forms, including facsimiles, can be obtained from the SEL Data Base Organization and User's Guide (Reference 3).

All forms are reviewed for completeness and consistency by each project leader before the forms are submitted to the SEL. Once the forms are determined to be complete and accurate, they are sent to the SEL data processing group. The forms are then logged in the library and prepared for entry into the SEL data base. Forms processing is examined in

Table 2-2. SEL Data Collection Forms

Form	Description of Content
General Project Summary	Computer resources used, starting and ending dates of each phase, cost information, size of product, methodologies and tools used in each phase of development, personnel involved, standards used, documentation produced, problems anticipated, and quality assurance information
Change Report	Change description, components changed, effort to change, type of change or error, and activities used to validate changes, to detect errors, and to find their cause
Resource Summary	Number of hours of worktime per week per staff member spent on a particular project, computer usage, and other charges
Component Status	Time spent during the week in a certain activity of component development (e.g., design, testing, or documentation)
Component Summary	Interfaces, programming language, complexity, resources required for each phase of development, relation to other components, and code specifications
Run Analysis	Computer used, purpose of the run, type of run, run results, and comments
Maintenance Report	Subset of change report with some maintenance-specific questions

detail in the SEL Data Base Maintenance System (DBAM) User's Guide and System Description (Reference 4). DBAM is the interactive data entry system.

2.2.2 COMPUTER ACCOUNTING INFORMATION

Computer accounting statistics for projects using the IBM S/360-95 and IBM S/360-75 computers are automatically collected for each job by the S/360 operating systems at execution time. Central processing unit (CPU) time, input/ output (I/O) time, job type, and job termination status (error code) are recorded. This data is made available to the SEL on a computer tape and/or as a printed biweekly accounting summary. Data on the computer tapes is condensed into totals for 4-hour blocks and saved on the data base. Accounting sheets are sent to the librarians by the Data Base Administrator (DBA). This information can be used to cross-check the data reported in the resource summary, component summary, and computer program run analysis report forms.

2.2.3 PERSONAL INTERVIEWS

Interviews are used to validate the accuracy of the data collected on the forms and to supplement that information in areas of uncertainty and probable error. Basically, two different types of interviews are conducted: spot-check interviews and management in-depth interviews.

Spot-check interviews are conducted by an analyst with the project personnel who fill out the forms. A check is made to determine that they have given correct and complete information as interpreted by an independent observer. Agreement is looked for in such areas as the cause of an error or the point in the development process at which the error was caused or detected. If necessary, the form is modified; the corrected form is then processed like any other form.

In-depth interviews are held to gather information on management decisions (e.g., why a particular personnel organization was chosen). These interviews cover the kinds of points that often require discussion rather than a simple answer on a form.

2.2.4 AUTOMATED DATA COLLECTION TOOLS

Two types of automated tools are used by the SEL for data collection: a FORTRAN source code analyzer and a library monitor. The data from these sources is one of the most objective and reliable data types available to the SEL.

The FORTRAN Static Source Code Analyzer Program (SAP) is a single-pass FORTRAN interpreter (with no execution phase) that produces statistics on occurrences of statements and structures within a FORTRAN source program. The program accepts, as input, syntactically correct FORTRAN source code written for the DEC PDP-11/70 FORTRAN IV PLUS compiler or the IBM S/360 FORTRAN IV Level H compiler. Component-level and summary statistics are calculated. The statistics include "Halstead Measures" (counts of the number of operators and operands, Reference 5) and "McCabe's Measure" (a count of the number of decisions in a component, Reference 6), as well as traditional measures such as the number of executable statements. Source code from the IBM S/360 is copied to tape by the librarians at the completion of a project; then the tape is processed on the DEC PDP-11/70 with SAP. SAP produces an output file that is processed by a special program to check for duplicate names and to incorporate all pertinent information in the SEL data base.

The PANVALET Program Management and Security System is used to establish, maintain, and control a central library of source programs and card image data files (data sets). Directory reports can be generated that show the status, number of statements, version number, date of last access, and several other statistics for each data set. When generated on a regular basis, these reports show the growth history of source programs in the PANVALET library. A library analysis report is also available that contains the averages, percentages, and totals of the number of statements, blocks, and data sets in the PANVALET library. These statistics can be broken down by data set name prefixes or as other subsets of the library.

PANVALET output is examined every 2 weeks. The growth and change history of the code is recorded and entered into the SEL data base.

2.2.5 SUMMARY MANAGEMENT INFORMATION

Two types of summary management information are collected. First, subjective evaluation data is generated during a review of a project by key SEL and development members familiar with the project. The quality of the delivered product and its development history are considered. Second, summary statistics are also collected; these include lines of code, resources used, and number of components. Together, this data fully describes the developed product, process, and environment at the project level.

Before the start of development, an experimental design (see Section 2.5) and development techniques are chosen. During development, an effort is made to ensure that these techniques and methodologies are used. SEL and development management decide upon the techniques to be used; the project manager and leader enforce the use of these techniques.

Formal and/or informal training may be required to familiarize project members with the techniques selected.

The subjective evaluation made at the conclusion of the project includes an estimate of the extent to which these target techniques were utilized during development. This evaluation is based on observations made by the evaluators during development.

The summary management information thus obtained is intended to be an independent evaluation of the quality of the product and the effectiveness of the techniques employed. This data is sent to the librarians for inclusion in the data base.

2.3 DATA VALIDATION

Data validation is the process by which information from all identified sources is checked by various means for correctness, completeness, consistency, and relevance. Depending on the source and type of information that is provided, different levels and types of validation can occur. In general, the types of validation that may be used are as follows:

- Spot checking
- Reviews (by project members, SEL coordinator, and librarian)
- Validation by data base software
- Generation of summaries
- Cross-checking form data with other data
- Comparisons among projects
- Statistical evaluations

A general discussion of these techniques may be found in Guide to Data Collection (Reference 2). However, some of these classes of validation, as used by the SEL, deserve elaboration here.

Data is reviewed at three levels: by the project members generating the data, by the contractor coordinator, and by the SEL director. Spot checks are also made by the librarians.

Another type of validation is performed by the data base software. It checks the information on the forms for completeness, consistency, and, in particular, for valid component and project names as well as other mandatory information items. The SEL Data Base Maintenance System (DBAM) User's Guide and System Description (Reference 4) describes these checks in detail.

In addition, cross-checks are made between groups of forms by taking advantage of the redundancy designed into the forms. This process ensures the quality and validity of the data for an entire project.

2.4 DATA BASE ORGANIZATION

The SEL data base is organized as a set of disk-resident files grouped by record format. Each is created as an indexed file and consists of a set of fixed-length records. The files are located on a disk device that is peripheral to the DEC PDP-11/70 computer of the Systems Technology Laboratory (STL) at GSFC.

A file type is a set of files with the same record definition (format) and index structure. File types may be grouped into three classes: (1) project summary file types that consist of a single file containing information about all projects in the data base; (2) project detail file types that consist of several files, one per project for which data has been collected; and (3) a directory file. Each form has a corresponding file type; for example, Run Analysis Form data from project 1 is stored in the Project 1 Run Analysis Form File. File types also exist for data from other sources. The Encoding Dictionary, a separate class of file, is a directory containing definitions of coded fields used in other files. A complete description of the data base is included in the SEL Data Base Organization and User's Guide (Reference 3). Table 2-3 outlines the file types.

An indexed organization was chosen to speed access by allowing the user to select records randomly for processing. The record selected is identified by key, which is a portion of the record defined as such when the file is created. A file may have several keys, allowing the user to select records in several ways. Additionally, a particular key defines an ordering of the records within a file. The data entry and reporting software that have been developed for the data base use these indexed features.

Table 2-3. SEL Data Base File Types

Class	File Type	Record Length (Bytes)
Directory	Encoding Dictionary	60
Summary	Phase Dates	112
	File Name and Status	52
	Subjective Evaluations	109
	Estimated Statistics	95
Detail	Component Information	67
	Component Summary Form	250
	Change Report Form	101
	Comment	104
	Attitude Maintenance	77
	Resource Summary Form	115
	Run Analysis Form	53
	Component Status Report	79
	Cumulative History	23
	Accounting Information	67

2.5 DATA ANALYSIS

The primary objective of the SEL is to improve the software development process by identifying the effects of methodological and environmental factors on that process. The specific analyses that have been attempted are discussed in Section 3. This subsection illustrates the analytical techniques and resources employed by the SEL in that research. Specifically, experimental design and analytical software are considered.

Three types of experiments have been performed: screening, semicontrolled, and controlled. The data collected from all of these experiments, with the exception of some controlled experiments performed by the University of Maryland, has been assembled in the SEL data base.

Screening experiments provide detailed information about how software is currently developed in the environment under study. Projects of all sizes and types have been monitored. In the experiments performed, the only impact on the tasks was the necessity of providing data via the data collection forms developed by the SEL. No attempt was made to impose new or different methodologies on these tasks.

Semicontrolled experiments provide information on the effects of various software development techniques. Specific methodologies were designated for application to each software development project, and an effort was made to ensure that these methodologies were followed by training the personnel and reviewing their efforts. It was anticipated that by comparing similar projects (i.e., similar in size, complexity, environment, and type of software) trends might be isolated that would characterize the effects of the various tools and techniques applied.

Controlled experiments are the most rigorous type of experiment. These may be implemented in either of two ways. Two carefully matched development teams may be assigned the same task but required to use different methodologies. Alternatively, two or more teams (not matched by experience or environment) may be assigned two consecutive tasks, with some additional methodology applied to the second task. These experimental designs of matched samples and repeated measures are very powerful, but they are also very costly to implement. Thus, they are not often employed.

One of the greatest concerns in designing experiments is the added impact or effect that the monitoring process itself may have on the performance of project members. This phenomenon, sometimes called the Hawthorne effect, must certainly be considered in any evaluation of experiment design. One way of eliminating any possibility of biased information is to make certain that the software development teams are unaware that the project is being monitored. However, this solution is impractical in the SEL environment, since the design of the experiment requires active participation (i.e., filling out forms and training) by all members of the project. Considering the large number of projects studied, the duration of projects (typically 15 months), and the professionalism of development personnel, the SEL principals have concluded that the Hawthorne effect is minimal or nonexistent.

The SEL has several software tools available for analyzing the data collected from these various types of experiments. They include data profile and graphical display programs. These displays and tabulations are employed to monitor the progress of ongoing projects. This information is also provided to project members to apprise them of project status.

In addition, more sophisticated analyses are possible with the approximately 40 statistical procedures of the Biomedical Programs, P-Series (BMDP, Reference 7) available on the STL PDP-11/70. These include multivariate analyses and hypothesis tests. Section 3 reviews some of the specific research and analysis efforts undertaken by the SEL.

SECTION 3 - SURVEY OF SEL RESEARCH

The preceding sections of this document describe the background of the SEL organization and its operations. This section outlines some of the specific results of SEL investigations of software development technology.

The data provided by the SEL has formed the basis of numerous software engineering studies. The specific software development tasks from which data was collected for the SEL data base are summarized in Appendix B. All data collected by the SEL is assembled in a computer data base to facilitate its access by researchers and managers. This data base is described in Sections 2.2 through 2.4.

The studies discussed in the following subsections touch every aspect of the software development process. Five classes of analyses are described

- Profile analysis--Section 3.1
- Methodology evaluation--Section 3.2
- Models--Section 3.3
- Tool evaluation--Section 3.4
- Measures and metrics--Section 3.5

Two very strong effects were identified early in the SEL investigations and have been confirmed in the literature (Reference 8). That is, variation in programmer abilities appears to be the most powerful influence on the productivity and quality of software development. In addition, the nature of the local computing and work environments seems to be a significant determining force on the process. Any valid experimental design must account for or eliminate these effects.

Consequently, the SEL has emphasized the goal of understanding the current software development process and environment as a prerequisite to more advanced analyses. Section 3.1 explains the efforts toward that goal in greater detail.

3.1 PROFILE ANALYSIS

Profile data reports the history or result of a software development effort; it is often presented in graphical or tabular form. A profile characterizes a specific software development project. The goal of such profiling is to define the software development process, environment, and product as a baseline for later comparisons. These elements are discussed as components of the SEL software development model in Section 2.

Profile analysis attempts to answer basic questions such as

- What rates of productivity were obtained?
- What kinds of and how many errors were discovered?
- What are the typical characteristics (e.g., size and complexity) of a component?
- How is the development effort distributed over the software life cycle?

Profile data is accumulated as part of the data collection process outlined in Section 2. The role of profile analysis in the SEL approach to software engineering research is primarily descriptive and comparative. The flight dynamics profile developed by the SEL and the comparisons of it with profiles of other development organizations are presented in Sections 3.1.1 through 3.1.4.

The SEL data discussed in this section is taken from seven similar large projects that have been studied extensively. Table 3-1 illustrates some important attributes of these software development efforts. (Data for a larger group of projects is described in Appendixes A and B.) Since the problem component (see Figure 1-1) is similar for all seven projects, it is not specifically considered here, although it is referred to obliquely in the following sections.

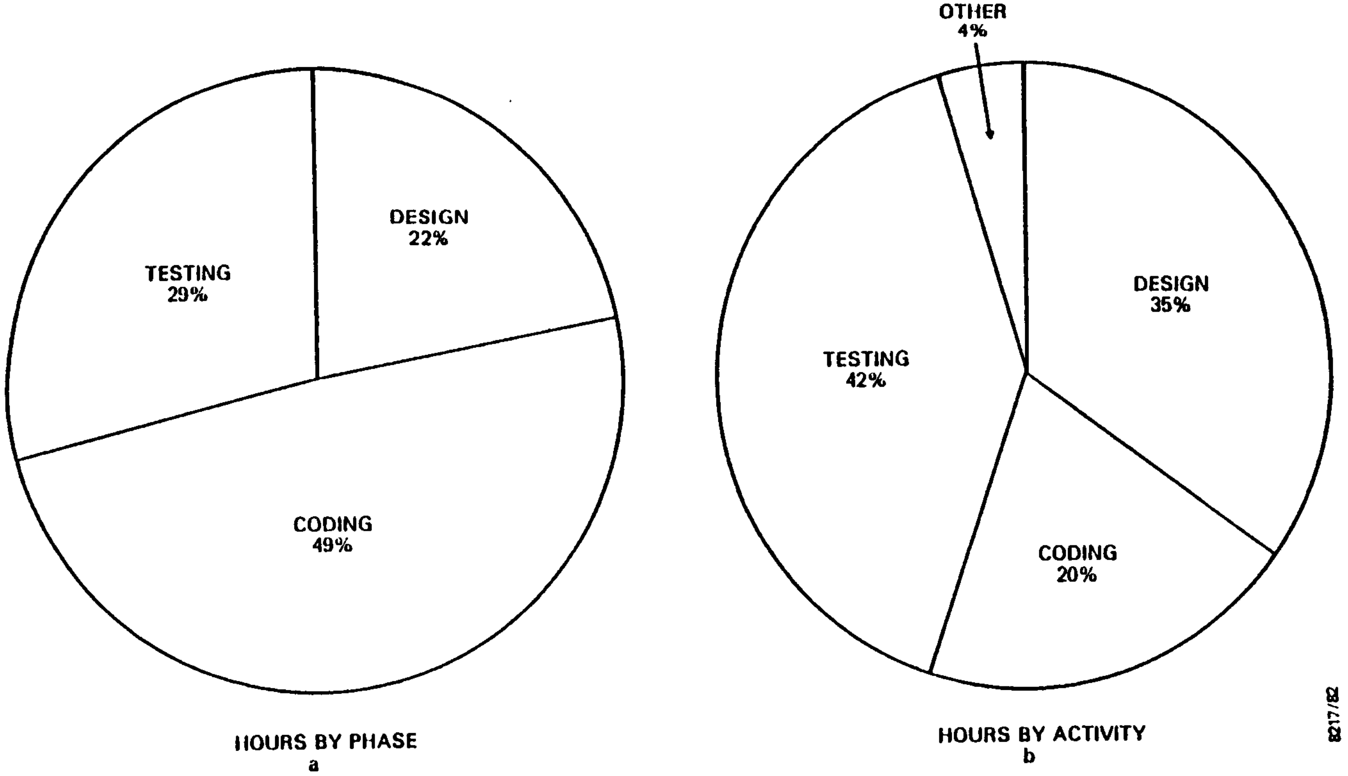
Table 3-1. Sources of Profile Data

Project	Size (Lines of Code)	Modules	Person-Months	Computer Runs	Software Changes
1	89,513	604	98	7,379	2761
2	50,911	201	78	4,604	1255
3	111,868	510	115	11,976	3228
4	75,393	535	90	7,500	2107
5	85,369	519	98	7,527	2710
6	75,420	374	39	3,033	858
7	55,237	320	95	6,871	1649

3.1.1 THE DEVELOPMENT PROCESS

Efforts to profile the flight dynamics software development process have focused on three areas: manpower utilization, computer utilization, and change/error characterization.

Data collected from the various projects can be presented in a manner that clearly illustrates the application of manpower to a software development task. Figure 3-1a shows the distribution of staff-hours worked by development phase. This chart is based on Resource Summary Form data submitted by managers (see Section 2.2.1). This chronological distribution of effort can be compared with the distribution of effort by activity reported by programmers in Figure 3-1b for a typical flight dynamics project. An additional category, "other," is present in the latter chart. This category includes such activities as system documentation,



HOURS BY PHASE
a

HOURS BY ACTIVITY
b

Figure 3-1. Manpower Utilization by Phase/Activity progress reports, and meetings that can not be defined as related to design, coding, or testing.

The difference in the two distributions of effort can be explained by the overlap of activities among chronological phases. For example, detailed design activity continues into the coding phase while testing normally commences during coding. Thus, the amount of effort assigned to the coding phase is greater than the amount of actual coding effort expended. This effect will be especially pronounced in software development operations that follow a top-down implementation technique or other methodology that advocates development by parts.

Figure 3-2 shows the types of computer runs made. The major roles of the computer in the development process are highlighted in this chart. These data are obtained from the Run Analysis Form. More data on computer usage is presented in Appendix A.

Change and error data track two important elements of the software development process: reliability and efficiency. Significant insight into the development process can be gained by examining profiles of the instances of change and/or error. Such data is especially valuable in detecting weaknesses in testing, programmer training, and development practices.

Figure 3-3 shows the distribution of the types of changes that occurred. Nonerror corrections are widely dispersed among several categories. A large number of code changes associated with requirements changes may indicate requirements instability. That appears to be only a small problem in the GSFC environment. More data on changes and errors is presented in Appendix A.

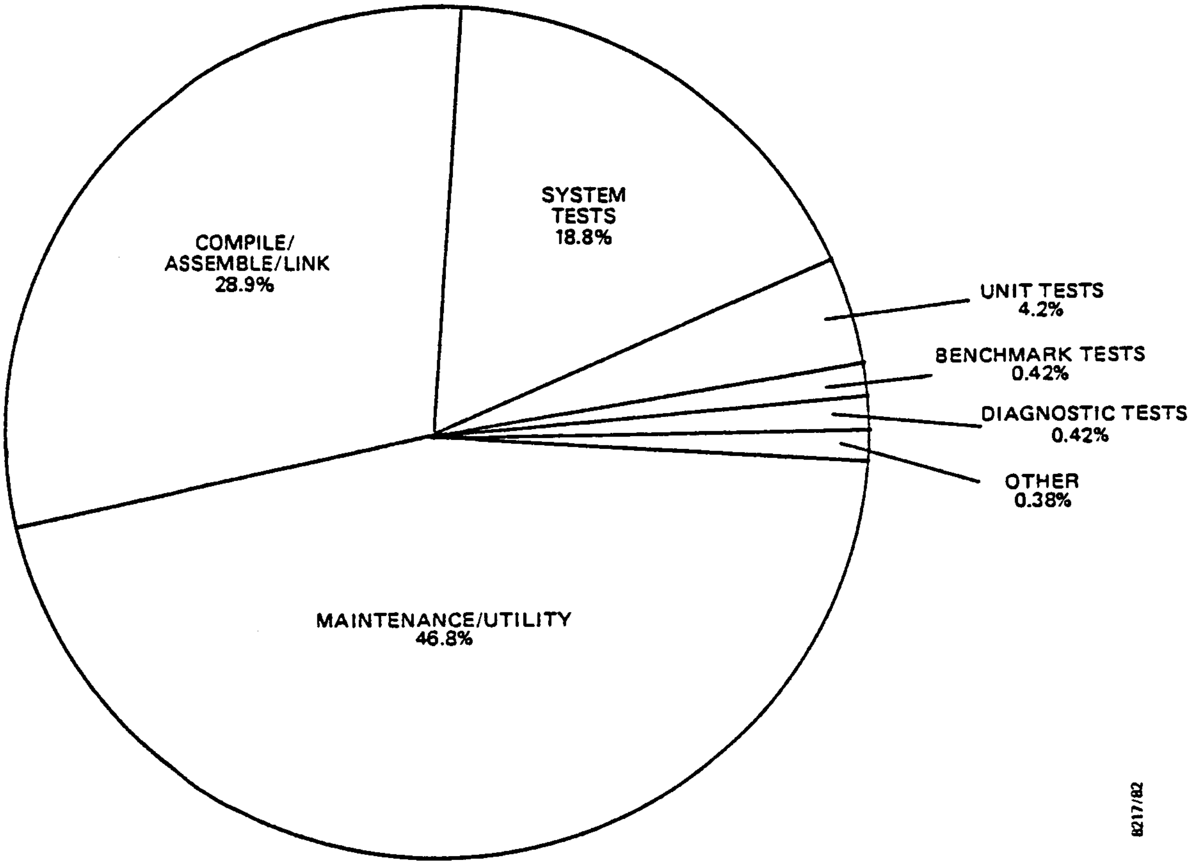


Figure 3-2. Types of Runs Made During Development

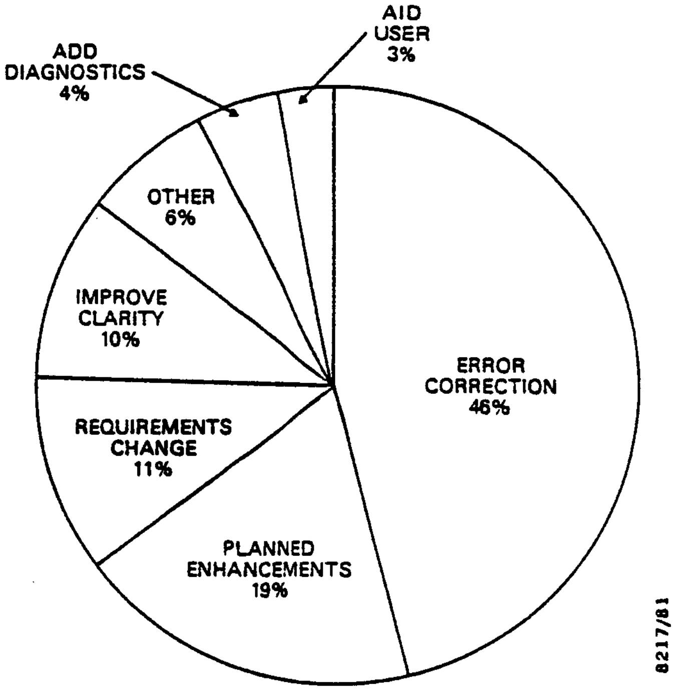


Figure 3-3. Types of Code Changes

3.1.2 THE DEVELOPMENT ENVIRONMENT

The environment is frequently a constraint rather than a controllable factor in the development process. Studying the other components of the software development model usually identifies these environmental contraints. The GSFC software development environment is described in detail in Section 1.3.

The principal physical element of the development environment is the computer system. Figure 3-4 shows the availability and reliability of the two principal flight dynamics development computers for a typical interval of time. The low reliability of the S/360-95 system compared with more modern equipment must be considered when evaluating the effectiveness of development techniques. One related finding of the SEL is that this hardware unreliability makes batch development more productive than interactive development. Interactive programmers are unable to work when the system is unavailable and are affected more (e.g., loss of data sets) by sudden system failure than batch developers are.

3.1.3 THE DEVELOPMENT PRODUCT

Most SEL data is collected on the project or component (subroutine) level. A measure of the total sizes of the seven projects from which the data studied here was obtained is shown in Table 3-1. A measure of the sizes of the FORTRAN subroutines in those projects is illustrated in Figure 3-5a. The distribution of McCabe's complexity measure is also displayed (Figure 3-5b). Note that 70 percent of the subroutines have complexities less than or equal to 10, the maximum recommended by McCabe (Reference 6). The McCabe measure is discussed in more detail in Section 3.5.

The distributions of these measures as represented by the histograms in Figure 3-5 are clearly not normal (see Section 3.5). Thus, the application of statistical tests and

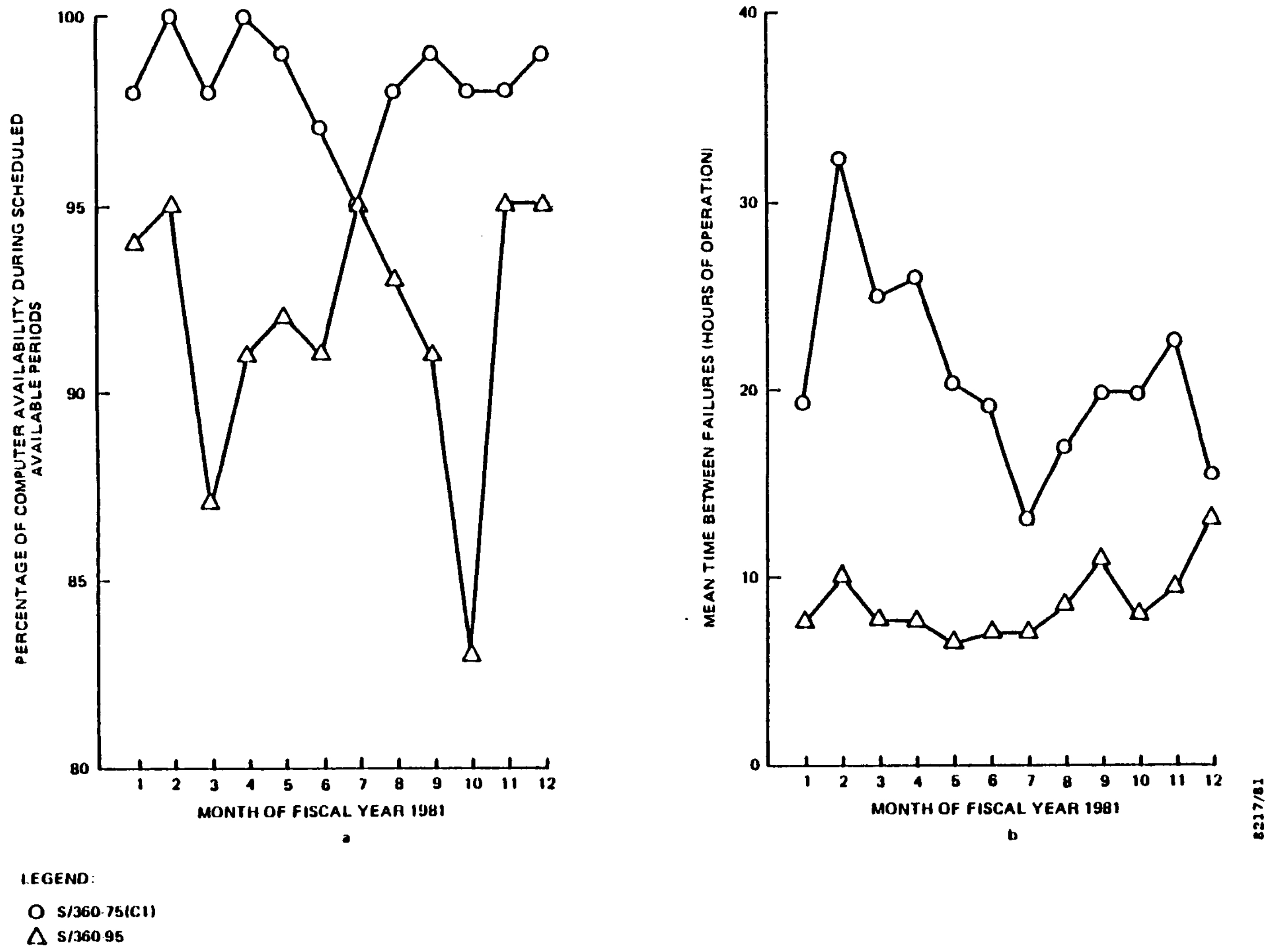


Figure 3-4. Measures of Reliability of the Computing Environment

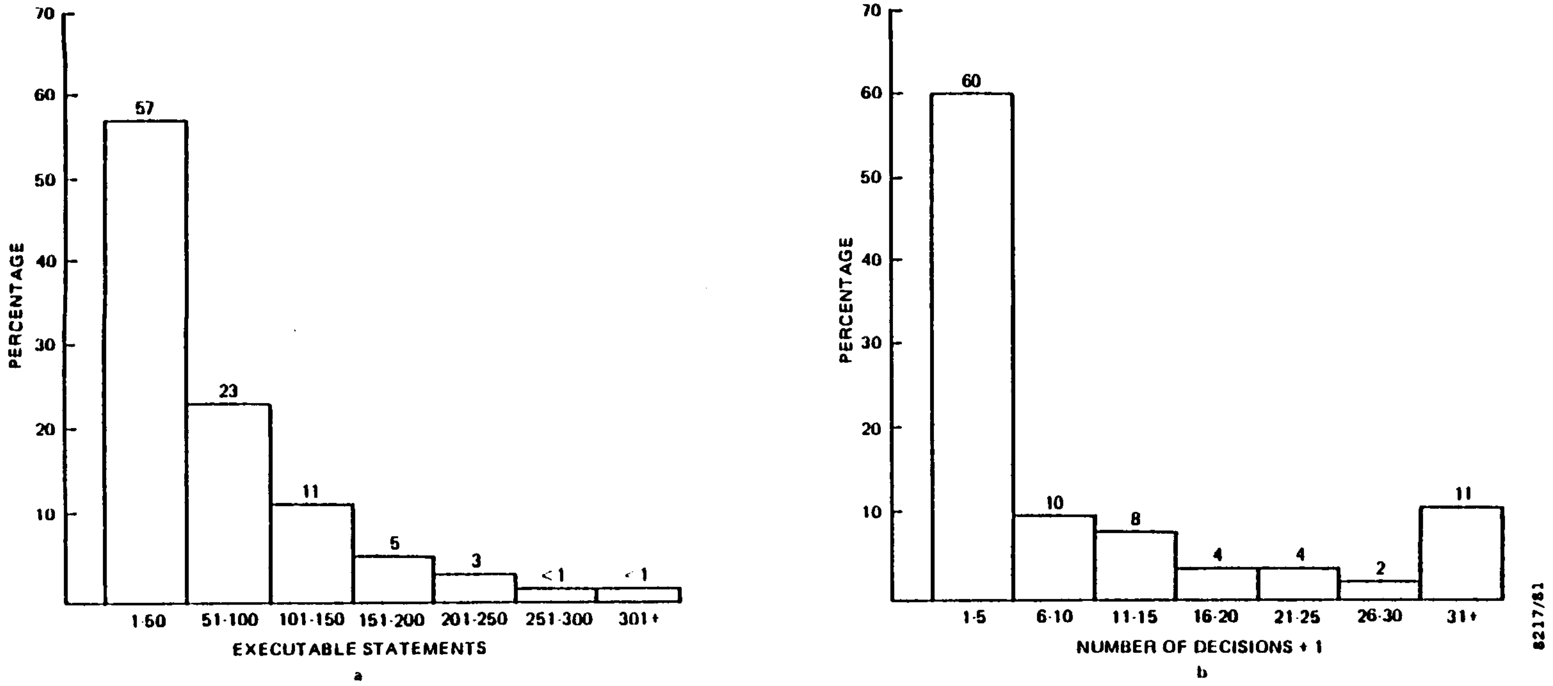


Figure 3-5. Characteristics of FORTRAN Modules Developed

regression procedures based on the assumption of normality cannot be expected to give good results with these measures. Nonparametric statistics for some FORTRAN subroutine measures based on a larger group of data are presented in Table A-2 of Appendix A.

Product profiles such as these are useful in determining the nature of the software developed and in identifying strategies for improvement. For example, a tendency to code lengthy or complex subroutines might be corrected by stressing strength and coupling, data abstraction, and structured techniques during programmer training.

3.1.4 PROFILE COMPARISONS

Comparing experimental results derived from different software development environments is difficult unless the relationships (similarities and differences) among the software development processes are well understood. When they are, valid extrapolations of experience from one organization to another can be made.

Table 3-2 shows the distribution of effort by activity for three software development organizations. The marked difference in resource utilization patterns suggests caution in making any inferences from one environment to another. The reason for these differences is discussed in Section 3.1.1. The data for this table is drawn from References 9 and 10.

Another brief comparison of several other measures for two of these organizations is presented in Table A-9 of Appendix A. A detailed comparison of SEL and Rome Air Development Center data in terms of size, effort, productivity, and error rate was made by Turner and Caron (Reference 11). Although that study showed consistency between the data bases, some significant differences were also noted.

Table 3-2. Comparison of Effort by Development Activity

Development Phase	Percentage of Effort		
	TRW	IBM[1]	NASA/GSFC SEL (Component Status)
Code	20	33	20
Design	40	39	35
Checkout and Test	40	22	42
Other	-	6	4

---

[1] Rescaled to sum to 100 percent.

3.2 METHODOLOGY EVALUATION

A software development methodology is the regular application of a set of specified techniques to part or all of the software development process. The methodologies and techniques studied by the SEL can be classified into five groups. The groups and some examples of each are listed below:

- Design Techniques
  - Top-down structured design
  - Tree charts
  - Data flow diagrams
  - HIPO charts
  - Process design languages
- Design Evaluation Techniques
  - Strength and coupling analysis
  - Connection matrices
  - Program correctness proofs
- Structured Implementation Techniques
  - Top-down structured programming
  - Structured languages

  - Code reading
  - Walkthroughs

- Management Techniques
  - Chief programmer teams
  - Design reviews
  - Librarian functions
  - Independent test teams

- Documentation Techniques
  - Automated documentation systems
  - Structured code

The SEL's approach to evaluating methodologies has been to collect cost and quality data from similar projects that employed different development methodologies (semicontrolled experiments). The relative effects of the methodologies on the product can then be observed and the useful techniques identified. Controlled experiments (as described in Section 2.5) would be the ideal means of collecting data for these analyses. However, the cost of duplicating any large development effort precludes that strategy.

The inability to make complete comparisons of the projects studied has delayed the derivation of definitive conclusions from the data. However, some effects are apparent. A summary of the early results of methodology evaluations is presented in Table 3-3. A superficial examination of this table suggests the reasonable conclusion that most techniques that do not significantly increase the programmer's and/or designer's workload but that provide a higher level of organization to his/her activities have a positive impact on the development process.

Table 3-3. SEL Metholodogy Evaluation: Some Early Conclusions

Results of Evaluations		
Cost Effective	Cost Unclear	Not Cost Effective
Formal Test Plan	Code Walkthroughs	Simulated Constructs
Process Design Language (PDL)	Top-Down Design	Axiomatic Design
Code Reading	Top-Down Code	Code Analyzers
Formal Training	Chief Programmer Team	Large Problem Statement Languages
Librarian	Code Auditors	Independent Verification and Integration
Configuration Management	Structured Analysis	Reliability Models
Design Formalisms	Requirements Languages	Automated Flowcharters
Formal Design Reviews	Automated PDL	
Structured Code (Precompilers)	Unit Development Folders	
Iterative Refinement	Resource Estimation Models	

More rigorous techniques have been applied to the analysis of some subsets of the SEL data on methodologies. Table 3-4 shows the results of a study of the effects of methodology on productivity (Reference 12). Essentially, it confirms the SEL's earlier conclusions.

Table 3-4. Relationship Between Productivity and Various Factors

Factors	Correlation
PDL	0.26
Formal Design Review	0.62**
Design Formalism	0.38
Design Decision Notes	0.62**
Design Walkthrough	0.28
Code Walkthrough	0.19
Code Reading	0.58**
Top-Down Design	-0.19
Structured Code	0.02
Librarian Use	0.52*
Chief Programmer Team	0.62**
Formal Test Plans	0.51*
Heavy Management Involvement	-0.09
Formal Training	0.58**
Top-Down Code	0.29

---

*SIG.<0.05
**SIG.<0.01

In addition, two commercially available axiomatic design methodologies were investigated by applying them to a demonstration project. The products of the design process included graphic representations of the functionally decomposed process and detailed component descriptions. The conclusion of the SEL was that the additional effort required by these methodologies was not justified by any improvement in design (References 13 and 14).

3.3 MODELS

Models have two important applications in the context of software engineering: explanation and estimation. The models considered by the SEL are mathematical abstractions of the software development process relating two or more fundamental characteristics. The characteristics of widest general interest and on which SEL efforts have been focused are resource utilization and software reliability.

A model isolates specific determining properties of the software development process. For example, the level of programmer experience might be included in a model relating staff-hours of effort to lines of developed code. This would reflect the analyst's understanding and explanation of the important factors in that relationship. The model thus developed can then be used to estimate the value of one factor from the known or assumed values of the other factors.

The development of valid models as explanatory and estimating tools is highly desirable. SEL efforts in the investigation of resource utilization and reliability models are described in Sections 3.3.1 through 3.3.3.

3.3.1 RESOURCE UTILIZATION MODELS

Resource utilization models relate measures of manpower and/or computer time to other aspects of the software development process. Many such models have been proposed. Reference 15 describes the SEL investigations of some of them; these include the Doty, Walston-Felix, Tecolote, GRC, SLIM, and PRICE S models. Only those that have been most influential on the SEL are discussed in detail in this document.

The resource utilization modeling problem has two parts: defining the total resources required and identifying the optimum distribution of those resources over the development cycle. Both parts of the modeling problem have been studied by the SEL.

The Putnam model of staffing (Reference 16) was among the earliest considered by the SEL. Putnam studied the distribution of manpower expenditures over time for several hundred medium to large software development projects of different classes. These projects exhibited similar development staffing patterns--a rise in manpower followed by a slower tailing off of effort. Putnam associated this curve with an optimum staffing level dependent on the rate at which work could be done at any phase of development (see Figure 3-6). The shape of the distribution of effort derived by Putnam is that of a Rayleigh curve.

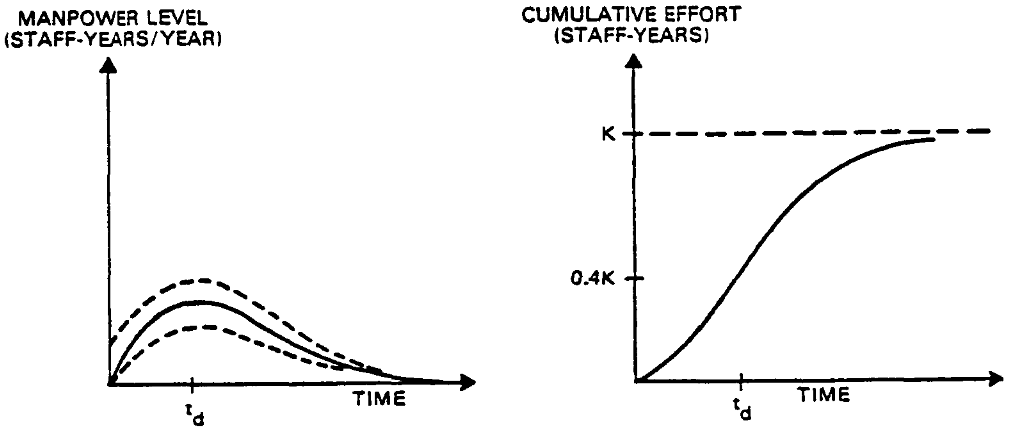
The form of the equation describing the curve is as follows:

$$Y = K/t_d^2 \cdot t \cdot e^{-t^2/2t_d^2}$$

where Y = the manpower at any time t
K = the area under the curve and corresponds to the total life cycle effort in man-years
t = the development time
t_d = time of peak manpower

This equation can be used to estimate the appropriate staffing level at any time and the total development time required. However, the accuracy of estimates is affected by variations in the development process and by the difficulty of exactly maintaining the optimum staffing level.

The correspondence between the Putnam model and the SEL data was not especially good (Reference 17). Several other



NOTE: THE CURVES DEFINED BY THIS FIGURE WERE ORIGINALLY APPLIED BY LORD RAYLEIGH TO DESCRIBE OTHER SCIENTIFIC PHENOMENA.

8217/82

Figure 3-6. Rayleigh Curve

curves were also fit to SEL data. A trapezoid and a parabola fit approximately as well as the Rayleigh curve. This may be explained in part by considering the effect of a fixed deadline for delivery. A project that is not begun early enough or that experiences unexpected difficulties will demonstrate a second peak of activity near the deadline. This phenomenon can be observed in Figure 3-7 where actual data is compared with two estimates of resource expenditures. The irregularity of the plotted data may be attributable to the relatively small size of the project being studied. The staffing level for such projects may be a step function rather than continuous.

The SEL also examined several models of the relationship between size of the developed system and the total effort required for development (Reference 18). Specifically, the Walston-Felix model (Reference 9) and the Boehm model (Reference 19) were evaluated. SEL experience with those analytic techniques contributed to the construction of the SEL "Meta-model." The next subsection discusses the derivation and formulation of that model.

3.3.2 THE SEL META-MODEL

The derivation of the SEL Meta-model is described in detail in Reference 20. However, it will also be outlined here.

Both the Walston-Felix and Boehm models propose a relationship among effort, lines of code, and an index of local conditions. The Walston-Felix index includes 29 factors; the Boehm index is a multiplicative combination of 16 factors. Although the SEL data seemed consistent with these models, a closer fit was desired.

The general equation was modified by devising a new measure of system size and by refining the selection of factors included in the index. The lines-of-code factor was replaced by the factor of new lines plus 20 percent of reused lines.

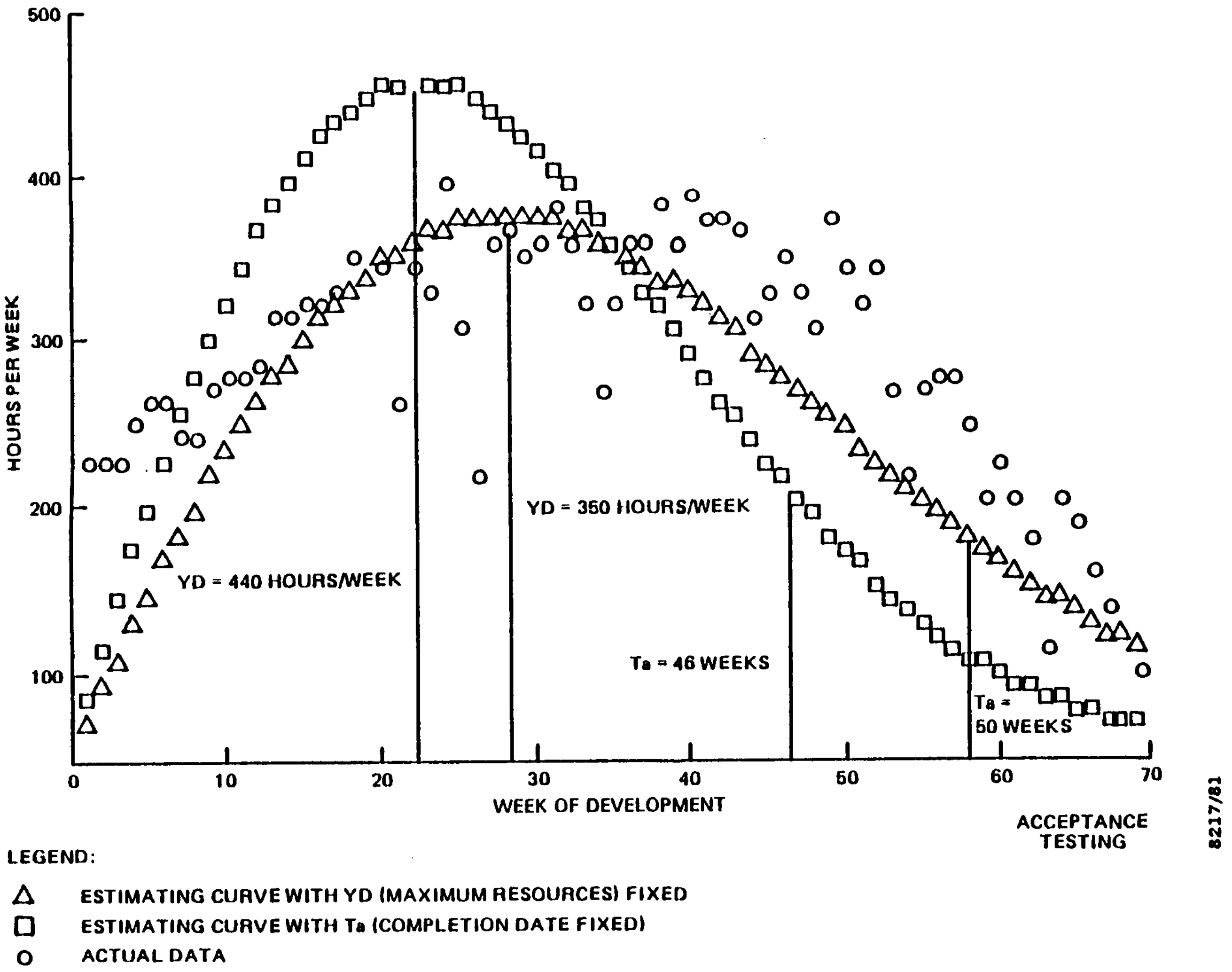


Figure 3-7. Estimated Resource Expenditures Curves

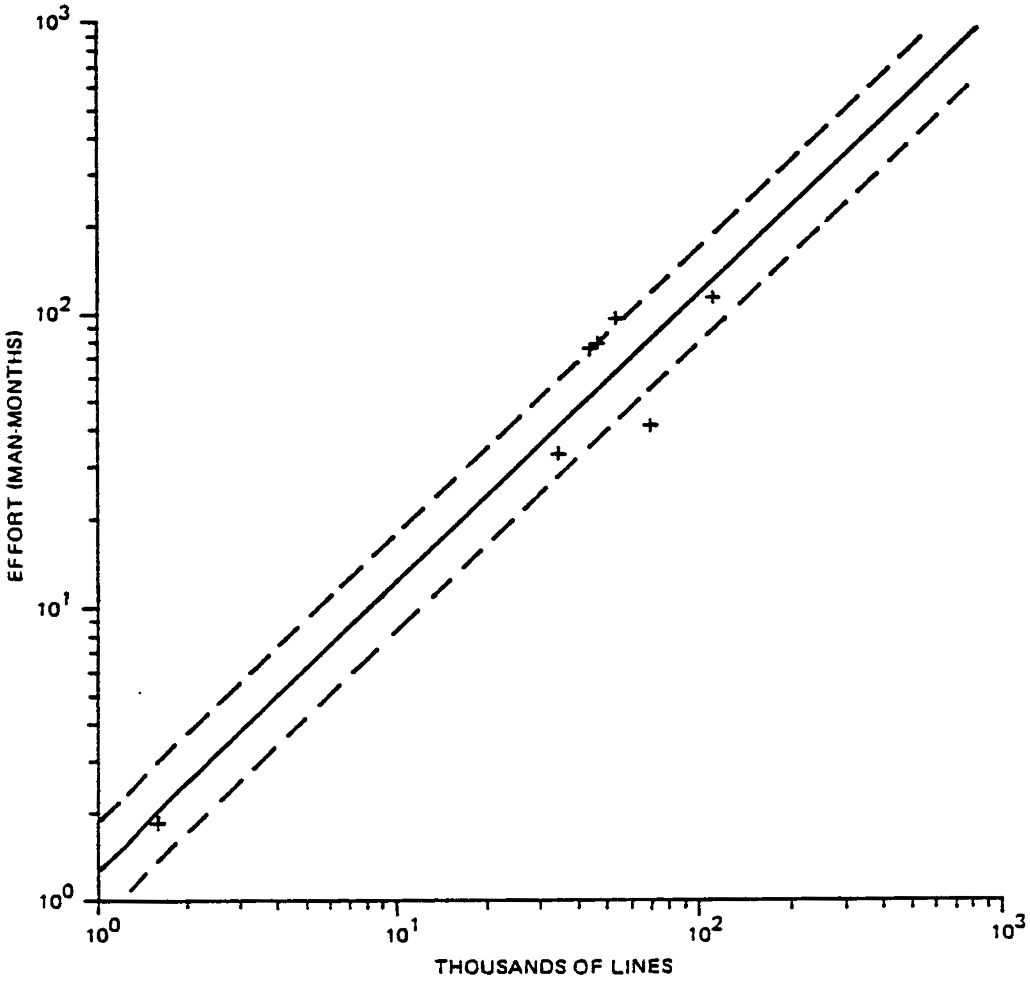
This measure is referred to as "developed" lines of code. It compensates for the bloating of size statistics that occurs when a substantial amount of previously developed code is reused. The correlation of this measure (developed lines) with effort is demonstrated in Figure 3-8. The relationship (base equation) established between effort and size is as follows:

$$E = .73 * DL^{1.16} + 3.5$$

where E = effort (staff-months)
DL = developed lines (thousands)

An attribute index refines the estimate of effort provided by this base equation by accounting for the variation due to such factors as problem complexity, programmer experience, and development techniques. The selection of significant attributes (factors) was accomplished by employing factor analysis as a data screening and reduction tool. Nearly 100 attributes were examined and 21 were selected for inclusion. They are grouped into three classes as follows:

- Total Methodology
  - Tree charts
  - Top-down design
  - Design formalisms
  - Formal documentation
  - Code reading
  - Chief programmer teams
  - Formal test plans
  - Unit development folders
  - Formal training
- Cumulative Complexity
  - Customer interface complexity
  - Customer-initiated program design changes



LOG-LOG PLOT SHOWING ONE STANDARD ERROR CONFIDENCE BAND.

NOTE: STANDARD ERROR = 1.456
CORRELATION = 0.958

8217/82

Figure 3-8. Effort Versus Developed Lines of Code

    - Application process complexity
    - Program flow complexity
    - Internal communication complexity
    - External communication complexity
    - Data base complexity

- Cumulative Experience
    - Programmer qualifications
    - Programmer experience with machine
    - Programmer experience with language
    - Programmer experience with application
    - Team previously worked together on same type problem

Each attribute for each project was rated on a scale from 0 to 5. Then, a sum was calculated for each of the three classes of attributes indicated in the list. These sums are the indices used to adjust the initial estimate of effort based on delivered lines of code. The final equation used includes the two major indices. That equation is as follows:

$$E_f = E_i * (-0.036 * M + 0.009 * C + 0.86)$$

where E_f = final estimate of effort
E_i = initial estimate of effort
M = sum of methodology ratings (index)
C = sum of complexity ratings (index)

The resulting adjusted estimator is the best predictor of the effort required for development of those estimators examined thus far by the SEL.

3.3.3 RELIABILITY MODELS

Software reliability can be defined as the length of time that a program will operate without a software failure. Ideally, developers would like to produce error-free software that operates indefinitely without failure. The cost of ensuring absolute freedom from error, however, is so great that most software developers accept less than that. Thus, they speak of developing software with the longest possible mean time to failure (MTTF).

Numerous models have been proposed that relate MTTF to the number of errors in a software system (Reference 21). An effective model of this relationship would have several uses. It could provide estimates of the number of errors present at the beginning and end of testing, as well as estimates of the time until the next software failure.

The only reliability model that has been carefully examined by the SEL is that of Musa (Reference 22). The mathematical representation of this model is a sequence of Poisson functions of the form

$$T_t = \frac{1}{fkN_o} e^{fkt}$$

where t = elapsed (CPU) testing time

T_t = MTTF at t

N_o = initial number of errors present

f = average execution rate

k = proportionality constant

Unfortunately, the SEL evaluation (Reference 23) of the Musa model had several weaknesses. Assumptions were made in the model that could not be experimentally validated; and data was not collected in a form convenient for these analyses.

As a result, the projects studied did not correspond very well to the Musa model.

3.4 TOOL EVALUATION

The SEL has attempted to evaluate the effectiveness of several software development tools in the GSFC environment. The evaluation process is similar to that used for methodologies. A tool is applied and its effect on software development is observed. The types of tools that have been examined by the SEL include requirements languages, design languages, programming languages (and preprocessors), code analyzers, and management tools. The most important tool evaluation efforts of the SEL are outlined below.

- URL/URA Requirements Language--This is an extensive and powerful requirements language that was acquired by GSFC. However, the complexity and overhead associated with its operation make it unsuitable for application in this environment.

- MEDL-R Requirements Language--MEDL-R is a small requirements language processor (Reference 24). Although it is still under review, the preliminary indications are favorable. However, it is also expensive (in systems and clerical costs) to use.

- Process Design Language (PDL) Processor (Caine, Farber, and Gordon)--This tool appears to promote a beneficial formalization of the detailed design process and to aid in the identification of design errors (Reference 25).

- Automated Flowcharters--Several automated flowcharters have been examined by the SEL. These are marginally useful in documentation but do not have any significantly favorable impact on other software development activities.

- Source Analyzer Program (SAP)--The SAP extracts measures of size, complexity, and function from software on a module-by-module basis. This tool has proved more useful for analysis than for development monitoring.

- Configuration Analysis Tool (CAT)--The CAT is an automated configuration management recordkeeping system. Its effectiveness is still under review by the SEL.

- Structured FORTRAN Preprocessor (SFORT)--This tool, developed in-house, is a structured FORTRAN preprocessor that extends the standard FORTRAN language to enable a user to write structured, top-down, label-free, FORTRAN-like code. The impact of this tool on software development was found to be very favorable. It is now routinely applied to applications projects.

The next step after identifying useful software development tools is to combine them into a comprehensive development system. The term "programmer workbench" is used to describe such a collection of tools implemented on an interactive computer system. The programmer workbench is an attempt to maximize the effectiveness of interactive programming by providing powerful, easily used software support for design, coding, testing, and documentation. Thus, the range of functions normally supported spans all phases of software development.

The programmer workbench includes many capabilities that may not be required for the operational (target) computer system for which the developed software is intended. As a result, the workbench is often implemented on a separate development system. This kind of implementation has several advantages in the GSFC environment. First, the workload of the S/360s is reduced, allowing a faster response to other activities.

Second, the progress of the development tasks becomes independent of the irregular availability and reliability of the S/360s (as indicated in Section 3.1.2).

The SEL is developing a programmer workbench for the DEC PDP-11/70. That effort is still at an early stage. Thus, this concept has not yet been tested. The SEL has, however, defined the general requirements of a GSFC Programmer Workbench (Reference 26).

3.5 MEASURES AND METRICS

The role of software measures and metrics is to define, explain, and predict important software qualities and quantities. The study of software measures and metrics overlaps the analyses described in the previous sections. For example, no consistent evaluation of the effectiveness of software development methodologies and tools is possible without having previously defined a standard of measurement. The profiling process is one of accumulating measures of the activities and conditions associated with software development. Successful modeling also depends on identifying meaningful and reliable metrics. Consequently, research on software measures and metrics has in the past, been driven by the need for measurement by those analyses. Recently, however, the attention of the software engineering research community has focused on testing the validity of commonly accepted metrics and on developing new, more powerful metrics.

Through the careful examination of graphs and histograms such as those in Figure 3-9, the SEL has discovered that the distributions of many software measures do not conform to the normal model. A numerical test of normality can also be made to detect this condition. The most commonly used statistical techniques are based on an assumption of normality that does not seem to be justifiable in these cases. Thus,

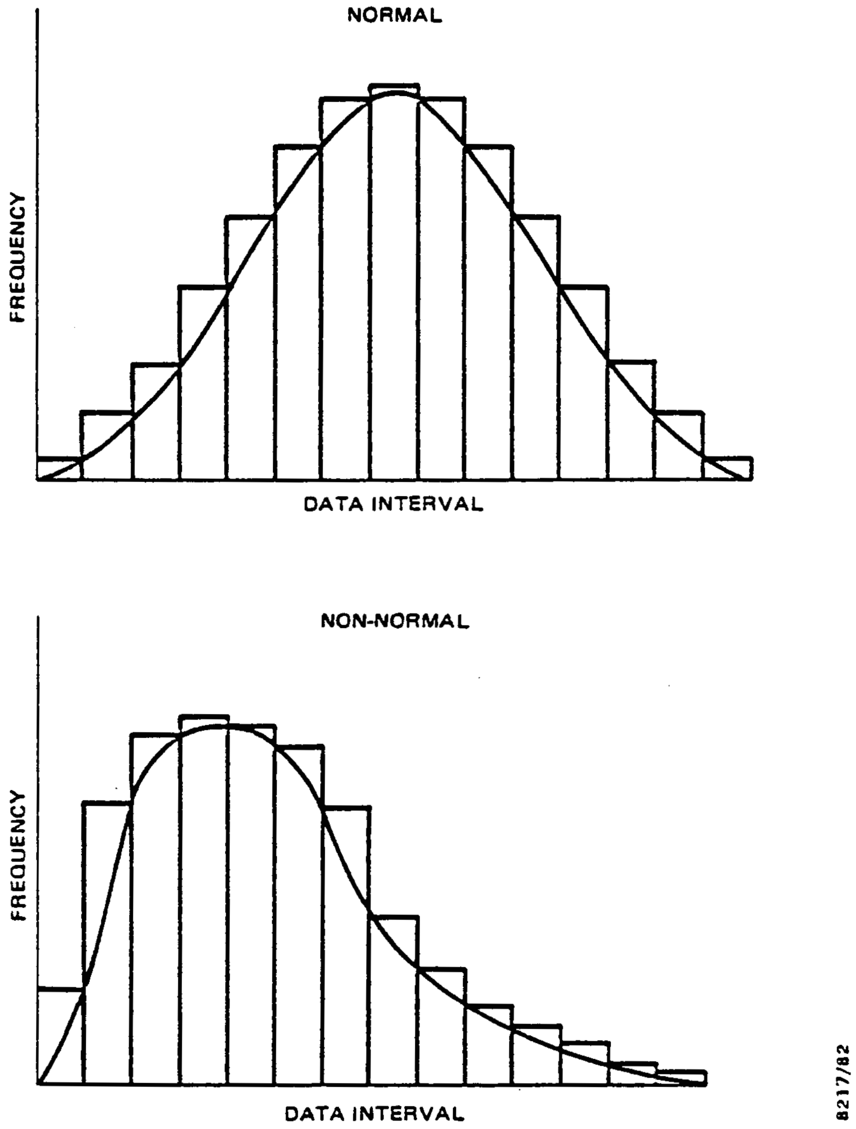


Figure 3-9. Hypothetical Data Distributions

future analyses will be planned with more consideration for the nature of the data involved.

The various software development measures with which the SEL has concerned itself may be grouped into three classes: static, derived, and subjective. Static measures are simple counts of significant features of the developed product and events in the development process. Derived measures are computed or are derived by analysis of source code or documentation. Subjective measures are qualitative determinations of attributes. These classes are explained in the following subsections.

3.5.1 STATIC MEASURES

Static measures include those collected as profile data. This is the most commonly employed class of software measures. Lines of code, number of errors, and staff-hours worked are examples of static measures. They describe the software development process in simple numerical terms. Unfortunately, static measures do not provide any estimate of the quality of software.

This type of metric is frequently standardized to facilitate comparisons. Thus, lines of code per day, number of errors per thousand lines of code, and staff-hours per component are used. SEL experience suggests that the standard form, "per thousand lines of code," is most useful for most static (profile) measures. Some of these measures are discussed in Section 3.1 under the heading of "Profile Analysis." A number of such measures are tabulated in Appendix A.

3.5.2 DERIVED MEASURES

Some software attributes of interest to the researcher cannot be quantified as easily as those just described. These qualities require the derivation of more sophisticated measures. Effort and complexity are examples of such attributes.

The SEL has studied the software science metrics of Halstead (Reference 5) and the cyclomatic complexity metric of McCabe (Reference 6). Attempts were made to validate the utility of these metrics and to compare them with standard (static) size measures.

Halstead's "length" and McCabe's "complexity" measures showed good agreement with the number of executable statements and related measures (Reference 27). However, some of the other Halstead measures, such as "language level," did not show the type of behavior predicted (Reference 28). A very high correlation of these derived measures with static measures (such as lines of code) is a negative result because it indicates that the simple measures provide just as much information as the more sophisticated measures. Statistics for these measures derived from the SEL data base are shown in Appendix A.

3.5.3 SUBJECTIVE MEASURES

Comparison of the results of applying different software development methodologies and/or tools must include an evaluation of the relative "quality" of the developed products. As previously suggested, such quality attributes cannot usually be measured objectively. They depend on the requirements of the specific system being developed. Thus, they must be estimated subjectively by persons familiar both with the requirements and with the implementations of the systems under study. However, some quality characteristics may be measured indirectly.

Attempts have been made to provide standard procedures for estimating quality measures. McCall (Reference 29) has identified important quality attributes and schemes for producing numerical values for them. These metrics are currently being studied as a possible method of defining software acceptability for the U.S. Air Force. The SEL has also assembled a group of subjective measures (see Section 2.2.5) that it is attempting to validate. The results of this evaluation effort are, as yet, incomplete.

SECTION 4 - SUMMARY

Preceding sections of this document attempt to answer several historical questions about the SEL: what is the SEL, how does it operate, and what has it done? This section recapitulates and explains some important points made earlier and suggests the future direction of SEL activities.

The discussion is in three parts. The status of the SEL relative to its objectives is reviewed in Section 4.1. Next, the general conclusions derived by the SEL from this research are outlined in Section 4.2. Finally, Section 4.3 presents some recommendations, based on the SEL experience, for conducting similar studies.

4.1 STATUS OF SEL

The objectives of the SEL are identified in Section 1.3. The SEL has met with varying degrees of success in achieving these objectives. Some have been satisfied, others were determined to be impossible (or nearly so), and more are still being worked toward.

The objectives that follow have been achieved, although work in these areas has not stopped. All of this effort has contributed toward a clearer understanding of the software development process.

- A number of software development methodologies have been evaluated. However, this activity has proceeded much more slowly than originally planned due to the myriad details and interrelationships that must be considered.
- A wide range of software development tools was evaluated. New tools will be tested as they become available.

- Many of the available cost estimation models were analyzed. A model was developed for use in the SEL environment (see the Meta-model, Section 3.3.2).
- A recommended approach to software development was arrived at and formalized in a document (Reference 30). This set of standards is expected to grow and change as knowledge about the software development process increases.
- Contact with the software engineering community has been maintained through the sponsorship of annual workshops.

Two of the SEL's original objectives appear to be impossible to achieve. These are as follows:

- The application of controlled experiments has proved to be too expensive and difficult to manage.
- Data processing constraints have prevented true real-time feedback to development teams. However, it may still be possible to make some information available on a timely basis during the development effort.

The course of future SEL research will be guided by its past experience. Many areas of research explored by the SEL did not produce conclusive results. Some analyses were adversely affected by a lack of reliable data. Others used approaches that were ultimately discovered to be inappropriate or ineffective. Some very promising studies have yet to be completed. The objectives toward which progress is, as yet, incomplete include the following:

- The analysis of software reliability models is a much more complex problem than originally envisioned. The available models do not agree very well with the collected data.

- The development of a set of software quality metrics is still in an early stage of activity.
- Although a number of important parameters have been identified, the development of software management guidelines also remains in its first stages.

The conclusions derived from the SEL's efforts to satisfy these objectives, as described in Section 3, are presented in the next subsection.

4.2 CONCLUSIONS

Several points stand out among the results of the research documented in Section 3. These conclusions are as follows:

- The software development process can be improved through the application of selected methodologies.

This general conclusion was derived from observations made during the past several years. Productivity rates have steadily increased through the years with the application of more refined methodologies. Even with the additional overhead of data collection and special training, a steady improvement in the development process is evident.

The amount of improvement attributable to any given methodology is very difficult to quantify, but the history of the SEL indicates that almost any of the disciplined methodologies available will favorably affect the process by about 5 to 10 percent over the absence of any such approach. A methodology that is particularly well suited to a specific environment could enhance productivity by as much as 20 percent. Optimizing the organizational structure of the people supporting the project can produce an additional improvement of 10 percent.

- The application of software development tools has not fully matured.

Although numerous software tools are now available and the use of tools is ever more popular, they are still not being applied effectively. Too many tools are adopted that are not cost effective given the software development environment. More emphasis must be placed on making tools user-friendly, rather than making users tool-friendly.

The SEL has found the supply of tools that do the "easily managed" tasks, such as flowcharting, code auditing, and language preprocessing, to be more than adequate. Additional effort should be expended on building and studying tools that facilitate difficult tasks, such as requirements analysis, project management, structured analysis, and design verification.

- Software cost models are useful but inadequate by themselves.

The SEL has reviewed and tested numerous software cost estimation models during the past several years but has obtained only mixed results. No cost model can replace "smart" engineers and historical cost data. However, cost models can supplement the cost estimation process when used properly. The larger, more sophisticated models (PRICE-S and SLIM) provide useful management planning statistics but must be delicately tuned and retuned.

The greatest danger in the application of current software cost models is that of placing an unjustifiably high degree of confidence in the results of models alone.

- Software reliability models are not useful in their present state of development.

The SEL has not yet extensively evaluated software reliability models; those that have been examined do not seem to be useful to software developers. The results of these models are difficult to interpret and apply in practice. However, the potential applications of reliability models to software development are significant.

- The greatest need is for the rational application of the available technologies, not for the creation of new technologies.

During the past several years, the SEL has learned that there are no shortages of well-defined methodologies and tools. The deficiency of current practice is in the utilization of the available software technology. Software implementers have been slow to evaluate and adapt these approaches to their particular environments.

Software technologies should not be accepted without critically examining their effects and without understanding the environment in which they operate. However, the evidence is conclusive that the software development process can be substantially improved through the application of appropriate technology.

4.3 RECOMMENDATIONS

The SEL's experience in software engineering research is a basis on which recommendations can be made for conducting similar studies. Because these suggestions are related to the specific goals pursued by the SEL, they may not be of great value to someone of different interests. However, the lessons are of a general enough nature to be important.

- Understand the current software development process. Evaluating potential improvements is impossible without first establishing a baseline for comparison. Moreover, a careful analysis of current practices may indicate those areas having the most potential for improvement. For example, an organization that expends most of its effort in testing and little effort in coding could maximize its return on effort by concentrating on improving testing techniques.

- Gather high-level data first. Management summaries and subjective evaluations on a project basis are the easiest to collect and analyze. Furthermore, they will introduce relevant questions that can be answered only at a more detailed level of analysis, in a realistic context. Thus, the researcher is in the position of identifying data to be collected to solve a specific problem, rather than of identifying a problem to solve with the data he/she has already collected.

- Control the effects of variations in the developers' skills. One of the most powerful effects on software development is the ability of the developers involved. This quantity must be measured if its effect is to be considered or controlled.

- Maintain close cooperation between research and development personnel. Collecting reliable data requires the active participation of the development groups being monitored. This may best be ensured by maintaining links at the management level.

- Two classes of productivity gains seem to be possible in software development. Immediate gains of 20 to 30 percent can be made by selecting and optimizing development methodologies, tools, and management practices. More extensive long term improvements of 200 to 500 percent would require the radical alteration of the development process and environment in such ways as employing very high-level languages, automating design activities, training programmers in specific software technologies, and increasing the reusability of code. This magnitude of improvement is the final goal of software engineering research.

APPENDIX A - SEL DATA TABULATIONS

The tables in this appendix display some data items from the SEL data base. Specifically, the data used to prepare these tables was drawn from 17 of the projects described in Appendix B. Graphs and tables presented elsewhere in this document were prepared from subsets of this data. This appendix includes nine tables. The first two (Tables A-1 and A-2) describe FORTRAN modules. Table A-3 shows computer runs. Changes and errors are displayed in Tables A-4 and A-5. Table A-6 shows total development effort. Summary statistics for each project are presented in Table A-7. Life cycle phase dates for these projects are reported in Table A-8. A comparison of Walston-Felix data (Reference 9) with SEL data is presented in Table A-9.

Tables A-3 through A-6 are organized similarly. Three major headings appear in each table. The first heading contains a list of data classes. The middle heading labels a breakdown, by percentages, of each class with respect to another classification criterion. The third heading labels a column showing the percentage of the data that each class represents.

Table A-1. Origins of FORTRAN Modules

ORIGIN	PERCENTAGE OF TOTAL
NEWLY DEVELOPED	57.2
EXTENSIVE CHANGES	7.4
SLIGHT CHANGES	13.6
REUSED UNCHANGED	21.8

8217/81

NOTE: TOTAL NUMBER OF FORTRAN MODULES = 2877.

Table A-2. FORTRAN Module Statistics

MEASURE	ORIGIN	MEDIAN	INTERQUARTILE RANGE[1]	HIGHEST VALUE
NUMBER OF EXECUTABLE STATEMENTS	ALL MODULES	38	30.5	
	NEW MODULES	41	30.0	1874
	EXT CHANGES	61	55.5	801
	SLT CHANGES	37	27.0	409
	OLD MODULES	26	26.5	819
NUMBER OF LINES OF CODE[2] (INCLUDING COMMENTS)	ALL MODULES	156	89.0	
	NEW MODULES	172	89.5	2093
	EXT CHANGES	218	134.8	1242
	SLT CHANGES	145	75.5	777
	OLD MODULES	92	69.3	1071
HALSTEAD LENGTH	ALL MODULES	210	201.5	
	NEW MODULES	215	201.0	3638
	EXT CHANGES	345	359.3	2786
	SLT CHANGES	211	183.0	3213
	OLD MODULES	146	181.3	4910
NUMBER OF DECISIONS (McCABE −1)	ALL MODULES	9	9.0	
	NEW MODULES	10	9.5	205
	EXT CHANGES	14	16.0	151
	SLT CHANGES	9	8.5	147
	OLD MODULES	5	9.0	161
NUMBER OF FUNCTION REFERENCES	ALL MODULES	0	1.5	
	NEW MODULES	0	1.5	89
	EXT CHANGES	1	2.5	54
	SLT CHANGES	0	1.5	42
	OLD MODULES	1	2.0	41
NUMBER OF EXTERNAL CALLS	ALL MODULES	3	4.5	
	NEW MODULES	4	5.0	137
	EXT CHANGES	7	6.0	75
	SLT CHANGES	2	3.5	102
	OLD MODULES	1	2.0	61
NUMBER OF I/O STATEMENTS[3]	ALL MODULES	1	2.0	
	NEW MODULES	2	2.5	910
	EXT CHANGES	3	2.5	470
	SLT CHANGES	1	2.0	230
	OLD MODULES	0	5.0	301

8217/81

[1] VALUE IS ½ OF RANGE BETWEEN THE FIRST AND THIRD QUARTILES.

[2] CONTENTS OF "INCLUDE" STATEMENTS ARE INCLUDED.

[3] USE OF I/O PACKAGES IS NOT INCLUDED.

NOTES: TOTAL NUMBER OF FORTRAN MODULES = 2877.

EXT = EXTENSIVE

SLT = SLIGHT

Table A-3. Distribution of Results of Computer Runs by Purpose of Run

PURPOSE OF RUN	RESULT OF RUN											PURPOSE PERCENTAGE OF TOTAL
	GOOD RUN	SUBMIT ERROR	JCL ERROR	SETUP ERROR	HARDWARE ERROR	SOFTWARE ERROR	COMPILE ERROR	LINK ERROR	EXECUTION ERROR	USER MESSAGE	RUN TO COMPLETION	
UNIT TEST	40.8	3.4	4.9	9.1	2.0	3.5	6.4	14.9	21.7	1.8	4.9	7.4
SYSTEM TEST	46.5	3.5	4.2	13.4	2.2	0.8	2.5	1.2	18.0	2.0	4.5	18.5
BENCHMARK	57.7	5.7	4.5	10.9	1.2	0.9	1.8	0.6	10.6	1.5	4.5	3.3
UTILITY	78.9	2.1	2.2	5.8	1.4	0.6	7.2	0.4	0.6	0.1	0.3	60.0
COMPILE/LINK	62.5	2.7	2.7	6.8	1.0	1.0	21.0	1.2	0.4	0.0	0.7	8.2
DIAGNOSTICS	40.7	2.2	5.5	12.1	1.1	1.1	3.3	1.1	24.2	4.4	5.5	0.9
OTHER	73.5	4.1	4.1	11.2	1.2	1.2	0.6	0.0	2.4	0.0	1.8	1.7
RESULT PERCENTAGE OF TOTAL	67.6	2.6	3.0	7.9	1.6	0.9	7.3	0.8	6.0	0.6	1.7	100.0

8217/81

NOTE: VALUES ARE PERCENTAGES; TOTAL NUMBER OF RUNS = 23,626.

Table A-4. Distribution of Effort To Change by Type of Change

TYPE OF CHANGE	EFFORT TO CHANGE				CHANGE TYPE PERCENTAGE OF TOTAL
	LESS THAN 1 HOUR	1 HOUR TO 1 DAY	1 DAY TO 3 DAYS	MORE THAN 3 DAYS	
ERROR CORRECTION	52.7	35.8	6.6	4.9	52.2
PLANNED ENHANCEMENT	25.3	36.8	16.5	21.4	15.4
REQUIREMENTS CHANGE	34.2	34.2	19.6	12.0	9.9
IMPROVE CLARITY	56.4	31.9	7.8	3.9	11.8
IMPROVE USER SERVICE	46.7	41.1	10.3	1.9	3.1
IMPROVE UTILITY	56.2	36.5	5.8	1.5	4.0
OPTIMIZATION	60.0	32.5	7.5	0.0	2.3
ENVIRONMENT CHANGE	33.3	66.7	0.0	0.0	0.4
OTHER	27.5	47.5	17.5	7.5	1.2
EFFORT TO CHANGE PERCENTAGE OF TOTAL	46.9	35.7	9.8	7.7	100.0

NOTE: VALUES ARE PERCENTAGES; TOTAL NUMBER OF CHANGES = 3470.

8217/81

Table A-5. Distribution of Effort To Correct by Type of Error

TYPE OF ERROR	EFFORT TO CORRECT				ERROR TYPE PERCENTAGE OF TOTAL
	LESS THAN 1 HOUR	1 HOUR TO 1 DAY	1 DAY TO 3 DAYS	MORE THAN 3 DAYS	
REQUIREMENTS	28.8	28.8	11.9	30.5	3.3
FUNCTIONAL SPECIFICATIONS	35.3	38.2	20.0	6.6	7.5
DESIGN OF MULTIPLE COMPONENTS	31.8	49.3	11.0	8.0	15.1
DESIGN OF ONE COMPONENT	53.4	39.3	4.7	2.7	50.1
ENVIRONMENT INTERFACE	30.0	60.0	0.0	10.0	0.6
LANGUAGE USE	64.6	32.3	1.5	1.5	3.6
CLERICAL	78.2	17.5	2.5	1.8	18.0
OTHER	62.9	20.0	5.7	11.4	1.9
EFFORT TO CORRECT PERCENTAGE OF TOTAL	52.9	36.0	6.5	4.7	100.0

NOTE: VALUES ARE PERCENTAGES; TOTAL NUMBER OF ERRORS = 1812.

8217/81

Table A-6. Distribution of Type of Development Effort by Phase

TYPE OF EFFORT	DEVELOPMENT PHASE[1]				EFFORT TYPE PERCENTAGE OF TOTAL
	DESIGN	CODE	SYSTEM TEST	ACCEPTANCE TEST	
MANAGER	26.5	45.7	12.5	15.3	20.1
PROGRAMMER/ANALYST	20.8	47.9	15.8	15.5	68.4
OTHER SERVICES	15.3	43.5	13.9	27.4	11.5
PHASE PERCENTAGE OF TOTAL	21.3	47.0	14.9	16.9	100.0

8217/81

[1] DETERMINED BY CALENDAR TIME OF OCCURRENCE.

NOTE: VALUES ARE PERCENTAGES; TOTAL NUMBER OF PROJECTS = 17.

Table A-7. Project Summary Statistics

PROJECT NUMBER	TOTAL COMPONENTS	TOTAL MODULES	NEW MODULES	MODIFIED MODULES	TOTAL LINES	NEW LINES	MODIFIED LINES	COMPUTER RUNS	CODE CHANGES	PAGES OF DOCUMENTATION	PROGRAMMER HOURS	MANAGER HOURS	SERVICES HOURS	S/360-95 HOURS	S/360-75 HOURS	OTHER COMPUTER HOURS
1[a]	88	64	–	–	7,033	–	–	–	–	–	–	–	–	–	–	–
2[a]	242	186	–	–	10,600	–	–	–	–	–	–	–	–	–	–	–
3	292	201	172	18	50,911	46,346	4,073	4,604	1266	1613	8,911.6	3876.6	1109.0	222.0	160.5	0
4[a]	267	210	–	–	–	–	–	–	–	–	–	–	–	–	–	–
5[a]	82	62	–	–	–	–	–	–	–	–	–	–	–	–	–	–
6	432	373	182	70	67,326	46,004	9,706	12,796	2077	2107	14,947.6	4627.3	2846.2	626.8	223.0	0
7	444	301	216	86	66,266	44,644	8,606	14,468	1676	2360	13,483.8	4632.8	3268.9	360.8	440.0	0
8	283	263	216	17	20,840	17,909	1,374	270	1274	140	3,793.4	190.6	63.9	262.7	119.6	80.0
9	161	134	74	22	17,271	10,822	2,331	2,187	641	760	3,463.2	1180.0	686.0	770.0	100.0	0
10	73	73	51	1	8,004	4,868	130	580	423	246	1,168.0	72.0	36.0	15.6	0	20.0
11	113	102	83	0	16,258	14,873	0	–	266	763	3,163.0	1302.2	1184.2	–	0	0
12	66	66	30	10	6,338	3,806	676	647	811	306	320.0	32.0	3.0	8.1	8.1	0
13	27	18	16	1	2,672	1,628	143	303	–	106	636.0	32.0	0	8.6	0.7	–
14	42	41	38	0	5,838	5,640	0	736	256	68	1,018.0	76.8	3.4	21.2	0.7	106.0
15	101	74	44	14	9,128	5,364	1,323	548	275	300	1,820.6	600.2	627.6	22.7	56.6	0
16	306	303	303	0	35,000[b]	–		–[c]	2176	721	3,211.1	840.9	648.0	0	0	–
17	98	98	65	0	80,702	56,601	4,181	[c]	–	318	3,200.0	660.0	325.0	0	0	2906.0
18	617	561	441	76	71,800	58,868	8,141	12,726	1740	3300	11,000.0[d]	–	–	362.0	140.6	0
19	74	66	46	0	10,172	9,627	527	1,476	218	265	1,667.6	498.0	543.6	64.3	14.0	0
20	366	283	200	21	65,237	43,966	3,608	6,071	1648	1104	12,829.0	2331.6	1378.0	163.0	166.3	0
21	423	374	82	30	76,420	20,076	6,727	3,033	868	1120	4,170.6	1620.9	1079.0	93.0	76.3	3.0
22	861	804	408	84	88,613	61,960	14,297	7,379	2761	2686	12,314.3	2807.0	1828.5	127.0	182.5	0
23	687	610	346	122	111,868	84,728	20,041	11,078	3228	2473	12,862.2	2907.8	4316.0	311.3	163.7	0
24	638	536	337	31	76,383	49,318	4,252	7,600	2107	1793	10,868.5	3651.0	1231.0	208.0	193.0	0
25	638	518	418	58	86,368	78,683	6,662	7,627	2710	2468	11,668.8	2711.9	2744.4	312.0	185.2	0

8217/82

[a] DEVELOPMENT IN PROGRESS

[b] CODE TRANSLATION

[c] INTERACTIVELY DEVELOPED

[d] INCLUDES MANAGER AND SERVICES HOURS

Table A-8. Life Cycle Phase Dates

PROJECT NUMBER	DESIGN START	CODE START	SYSTEM TEST START	ACCEPTANCE TEST START	ACCEPTANCE TEST END
1	10/04/80	10/31/81	—[a]	—[a]	—[a]
2	06/30/81	09/01/81	—[a]	—[a]	—[a]
3	02/13/77	06/04/77	12/03/77	02/04/78	03/18/78
4	05/01/80	12/13/80	10/03/81	—[a]	—[a]
5	02/03/81	05/02/81	09/05/81	—[a]	—[a]
6	10/01/79	05/10/80	02/28/81	03/28/81	06/13/81
7	10/01/79	05/10/80	12/12/80	02/21/81	05/02/81
8	02/01/80	06/15/80	11/15/80	02/15/81	05/15/81
9	12/01/79	05/17/80	01/17/81	02/14/81	04/11/81
10	01/01/81	09/12/80	10/10/80	02/02/81	06/01/81
11	10/01/79	04/12/80	08/30/80	09/27/80	10/25/80
12	07/01/80	09/12/80	01/01/81	01/26/81	02/13/81
13	09/01/78	10/01/78	01/01/79	03/01/79	05/30/79
14	02/03/79	06/21/79	08/18/79	09/01/79	10/13/79
15	02/03/79	05/26/79	08/04/79	09/01/79	10/13/79
16	04/01/76	07/03/76	09/24/77	03/01/78[b]	—[b]
17	02/03/81	03/28/81	08/01/81	09/11/81	09/30/81
18	03/01/75	07/05/75	01/01/77	05/28/77	07/30/77
19	05/01/78	02/03/79	05/19/79	07/14/79	08/18/79
20	10/01/76	02/26/77	07/23/77	08/20/77	09/17/77
21	08/15/77	12/03/77	03/11/78	04/08/78	05/06/78
22	06/01/78	10/14/78	03/31/79	06/02/79	08/11/79
23	06/01/76	10/09/76	05/21/77	07/23/77	09/24/77
24	04/01/77	07/30/77	01/14/78	02/18/78	04/15/78
25	05/01/78	10/14/78	03/31/79	06/02/79	10/13/79

8217/82

[a] NOT AVAILABLE AT THIS TIME; DEVELOPMENT IN PROGRESS.

[b] END OF SYSTEM TEST; ACCEPTANCE TEST NOT PERFORMED.

Table A-9. Comparison of Walston-Felix Data With SEL Data

MEASURES	W-F MEDIAN[a]	SEL MEDIAN[b]
TOTAL SOURCE LINES (THOUSANDS)	20	49[c]
PERCENT OF LINES NOT DELIVERED	5	0
SOURCE LINES PER STAFF-MONTH	274	601[c]
DOCUMENTATION (PAGES) PER THOUSAND LINES	69	26
TOTAL EFFORT (STAFF-MONTHS)	67	96
AVERAGE STAFFING LEVEL	6	5
DURATION (MONTHS)	11	15
DISTRIBUTION OF EFFORT		
MANAGER	22[d]	19
PROGRAMMER	73[d]	68
OTHER	5[d]	13
ERRORS PER THOUSAND LINES	1.4	0.8

8217/82

[a] DATA FROM TABLE 3 OF REFERENCE 9.

[b] DATA FROM 11 SIMILAR PROJECTS (SEE APPENDIX B).

[c] LINES ARE DEVELOPED LINES OF CODE.

[d] RESCALED TO SUM TO 100 PERCENT.

APPENDIX B - SEL PROJECT SUMMARIES

The following pages describe the major projects studied by the SEL, the types of data collected, and the experimental objectives toward which the data applies.

Notes to the Data Summaries

1. Developed lines of code (program size) is computed as total new lines of code plus 20 percent of reused lines of code. The use of this measure is justified in Section 3.3.2.
2. The data types shown in the summaries correspond to the data files identified below. These files are described in Section 2.4.
    a. Manpower Utilization--Component Status Report, Resource Summary
    b. Computer Utilization--Run Analysis, Computer Accounting
    c. Product Measures--Component Summary, Component Information
    d. Change/Error Characteristics--Change Report
    e. Project Summary Statistics--Subjective Evaluations, Estimated Statistics

PROJECT 1

SOFTWARE CHARACTERISTICS	Scientific High reliability requirement Batch Nongraphics Real-time
ENVIRONMENT	
Target Computer	8086
Development Computer	VAX-11/780, 8086
Language	FORTRAN
RESOURCES	
Level of Effort	63.5 staff-months[a]
Project Duration	29 months[a]
Peak Staff Level	
Full-Time Equivalent	3.0
Individual Members	5
Average Staff Level	
Full-Time Equivalent	2.2
PROGRAM SIZE	
Modules	120[a]
Delivered Lines of Code	15,000[a]
Developed Lines of Code	15,000[a]
EXPERIMENTAL OBJECTIVES	Study software transportability Methodology evaluation Resource and cost estimation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Product measures Change/error characteristics Project summary statistics

---

[a] Estimate based on incomplete data.

PROJECT 2

SOFTWARE CHARACTERISTICS	Scientific Batch Nongraphics Real-time
ENVIRONMENT	
Target Computer	VAX-11/780
Development Computer	VAX-11/780
Language	FORTRAN
RESOURCES	
Level of Effort	38.1 staff-months[a]
Project Duration	21 months[a]
Peak Staff Level	
Full-Time Equivalent	3.0[a]
Individual Members	5[a]
Average Staff Level	
Full-Time Equivalent	2.0[a]
PROGRAM SIZE	
Modules	200
Delivered Lines of Code	15,000[a]
Developed Lines of Code	11,000[a]
EXPERIMENTAL OBJECTIVES	Profile small task Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Project summary statistics

---

[a] Estimate based on incomplete data.

PROJECT 3

SOFTWARE CHARACTERISTICS	Scientific/data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	79.0 staff-months
Project Duration	13 months
Peak Staff Level	
Full-Time Equivalent	7.7
Individual Members	11
Average Staff Level	
Full-Time Equivalent	5.3
PROGRAM SIZE	
Modules	201
Delivered Lines of Code	50,911
Developed Lines of Code	46,458
EXPERIMENTAL OBJECTIVES	Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Project summary statistics

PROJECT 4

SOFTWARE CHARACTERISTICS	Scientific High reliability requirement Batch Nongraphics Real-time
ENVIRONMENT	
Target Computer	PDP-11/23
Development Computer	PDP-11/70, PDP-11/23
Language	FORTRAN
RESOURCES	
Level of Effort	75.0 staff-months[a]
Project Duration	23 months[a]
Peak Staff Level	
Full-Time Equivalent	4[a]
Individual Members	8[a]
Average Staff Level	
Full-Time Equivalent	2.2[a]
PROGRAM SIZE	
Modules	240[a]
Delivered Lines of Code	20,000[a]
Developed Lines of Code	16,800[a]
EXPERIMENTAL OBJECTIVES	Study software transportability Methodology evaluation Study effect of time/memory constraints Resource and cost estimation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Product measures Change/error characteristics Project summary statistics

---

[a] Estimate based on incomplete data.

PROJECT 5

SOFTWARE CHARACTERISTICS	Scientific High reliability requirement Batch Nongraphics Real-time
ENVIRONMENT	
Target Computer	PDP-11/23
Development Computer	PDP-11/70, PDP-11/23
Language	FORTRAN
RESOURCES	
Level of Effort	19.0 staff-months[a]
Project Duration	13 months[a]
Peak Staff Level	
Full-Time Equivalent	2.5[a]
Individual Members	6[a]
Average Staff Level	
Full-Time Equivalent	1.4[a]
PROGRAM SIZE	
Modules	50[a]
Delivered Lines of Code	3000[a]
Developed Lines of Code	2520[a]
EXPERIMENTAL OBJECTIVES	Study software transportability Study effect of memory constraints Methodology evaluation Resource and cost estimation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Product measures Change/error characteristics Project summary statistics

---

[a] Estimate based on incomplete data.

PROJECT 6

SOFTWARE CHARACTERISTICS	Scientific/data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	128.8 staff-months
Project Duration	20.5 months
Peak Staff Level	
Full-Time Equivalent	13.9
Individual Members	7
Average Staff Level	
Full-Time Equivalent	5.2
PROGRAM SIZE	
Modules	373
Delivered Lines of Code	67,325
Developed Lines of Code	49,468
EXPERIMENTAL OBJECTIVES	Study effect of independent verification and validation Methodology evaluation Use of configuration management tool Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Change/error characteristics Project summary statistics

PROJECT 7

SOFTWARE CHARACTERISTICS	Scientific/data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	122.7 staff-months
Project Duration	19 months
Peak Staff Level	
Full-Time Equivalent	9.7
Individual Members	17
Average Staff Level	
Full-Time Equivalent	5.1
PROGRAM SIZE	
Modules	391
Delivered Lines of Code	66,266
Developed Lines of Code	48,968
EXPERIMENTAL OBJECTIVES	Study effect of independent verification and validation Methodology evaluation Use of configuration management tool Use of requirements language tool Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Project measures Change/error characteristics Project summary statistics

PROJECT 8

SOFTWARE CHARACTERISTICS	Scientific Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	23.3 staff-months
Project Duration	15.5 months
Peak Staff Level	
Full-Time Equivalent	2.9
Individual Members	6
Average Staff Level	
Full-Time Equivalent	1.6
PROGRAM SIZE	
Modules	263
Delivered Lines of Code	20,648
Developed Lines of Code	18,529
EXPERIMENTAL OBJECTIVES	Profile small task Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Project summary statistics

PROJECT 9

SOFTWARE CHARACTERISTICS	Scientific/data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	30.7 staff-months
Project Duration	16.5 months
Peak Staff Level	
Full-Time Equivalent	3.0
Individual Members	10
Average Staff Level	
Full-Time Equivalent	1.7
PROGRAM SIZE	
Modules	134
Delivered Lines of Code	17,271
Developed Lines of Code	12,112
EXPERIMENTAL OBJECTIVES	Study effect of independent verification and validation Methodology evaluation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Computer utilization Product measures Change/error characteristics Project summary statistics

PROJECT 10

SOFTWARE CHARACTERISTICS	Scientific Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	VAX-11/780
Language	FORTRAN
RESOURCES	
Level of Effort	7.1 staff-months
Project Duration	5 months
Peak Staff Level	
Full-Time Equivalent	1.1
Individual Members	3
Average Staff Level	
Full-Time Equivalent	0.6
PROGRAM SIZE	
Modules	73
Delivered Lines of Code	9004
Developed Lines of Code	5768
EXPERIMENTAL OBJECTIVES	Study software transportability Evaluate programmer workbench environment Use of requirements language tool Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Product measures Project summary statistics

PROJECT 11

SOFTWARE CHARACTERISTICS	Scientific/data processing Batch Nongraphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	32.7 staff-months
Project Duration	13 months
Peak Staff Level	
Full-Time Equivalent	3.4
Individual Members	8
Average Staff Level	
Full-Time Equivalent	1.8
PROGRAM SIZE	
Modules	102
Delivered Lines of Code	15,258
Developed Lines of Code	14,950
EXPERIMENTAL OBJECTIVES	Methodology evaluation Resource and cost estimation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Computer utilization Product measures Change/error characteristics Project summary statistics

PROJECT 12

SOFTWARE CHARACTERISTICS	Data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	2.1 staff-months
Project Duration	7.5 months
Peak Staff Level	
Full-Time Equivalent	0.6
Individual Members	3
Average Staff Level	
Full-Time Equivalent	0.3
PROGRAM SIZE	
Modules	55
Delivered Lines of Code	5336
Developed Lines of Code	4111
EXPERIMENTAL OBJECTIVES	Profile small task Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Project summary statistics

PROJECT 13

SOFTWARE CHARACTERISTICS	Scientific Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	PDP-11/70
Language	FORTRAN
RESOURCES	
Level of Effort	4.0 staff-months
Project Duration	9 months
Peak Staff Level	
Full-Time Equivalent	0.7
Individual Members	3
Average Staff Level	
Full-Time Equivalent	0.4
PROGRAM SIZE	
Modules	18
Delivered Lines of Code	2572
Developed Lines of Code	1817
EXPERIMENTAL OBJECTIVES	Profile small task Evaluate programmer workbench environment Study of software transportability Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Project summary statistics

PROJECT 14

SOFTWARE CHARACTERISTICS	Data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	PDP-11/70, S/360
Language	FORTRAN
RESOURCES	
Level of Effort	6.3 staff-months
Project Duration	8.5 months
Peak Staff Level	
Full-Time Equivalent	2.0
Individual Members	5
Average Staff Level	
Full-Time Equivalent	1.0
PROGRAM SIZE	
Modules	41
Delivered Lines of Code	5639
Developed Lines of Code	5560
EXPERIMENTAL OBJECTIVES	Profile small task Evaluate programmer workbench environment Study software transportability Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Project summary statistics

PROJECT 15

SOFTWARE CHARACTERISTICS	Scientific Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	17.6 staff-months
Project Duration	8.5 months
Peak Staff Level	
Full-Time Equivalent	4.5
Individual Members	9
Average Staff Level	
Full-Time Equivalent	1.6
PROGRAM SIZE	
Modules	74
Delivered Lines of Code	9126
Developed Lines of Code	6108
EXPERIMENTAL OBJECTIVES	Evaluate formal training Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Project summary statistics

PROJECT 16

SOFTWARE CHARACTERISTICS	System executive Graphics Not real-time
ENVIRONMENT	
Target Computer	PDP-11/70
Development Computer	PDP-11/70
Language	MACRO-11, FORTRAN
RESOURCES	
Level of Effort	27.7 staff-months
Project Duration	23 months
Peak Staff Level	
Full-Time Equivalent	2.2
Individual Members	4
Average Staff Level	
Full-Time Equivalent	1.2
PROGRAM SIZE	
Modules	393
Delivered Lines of Code	35,000[a]
EXPERIMENTAL OBJECTIVES	Study assembly language software conversion Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Product measures Project summary statistics

---

[a] Estimate includes assembler statements and macros.

PROJECT 17

SOFTWARE CHARACTERISTICS	Scientific/data processing Batch Nongraphics Not real-time
ENVIRONMENT	
Target Computer	VAX-11/780
Development Computer	VAX-11/780
Language	FORTRAN
RESOURCES	
Level of Effort	23.5 staff-months
Project Duration	10 months
Peak Staff Level	
Full-Time Equivalent	5.0
Individual Members	7
Average Staff Level	
Full-Time Equivalent	4.1
PROGRAM SIZE	
Modules	99
Delivered Lines of Code	60,762
Developed Lines of Code	57,433
EXPERIMENTAL OBJECTIVES	Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Product measures Project summary statistics

PROJECT 18

SOFTWARE CHARACTERISTICS	Scientific Batch Nongraphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	63.5 staff-months
Project Duration	29 months
Peak Staff Level	
Full-Time Equivalent	6.6
Individual Members	7
Average Staff Level	
Full-Time Equivalent	3.3
PROGRAM SIZE	
Modules	551
Delivered Lines of Code	71,800
Developed Lines of Code	62,087
EXPERIMENTAL OBJECTIVES	Resource and cost estimation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Computer utilization Product measures Change/error characteristics Project summary statistics

PROJECT 19

SOFTWARE CHARACTERISTICS	Scientific Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	15.6 staff-months
Project Duration	15.5 months
Peak Staff Level	
Full-Time Equivalent	2.2
Individual Members	9
Average Staff Level	
Full-Time Equivalent	0.8
PROGRAM SIZE	
Modules	55
Delivered Lines of Code	10,172
Developed Lines of Code	9,736
EXPERIMENTAL OBJECTIVES	Evaluate formal training Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Project summary statistics

PROJECT 20

SOFTWARE CHARACTERISTICS

Scientific/data processing
Interactive
Graphics
Not real-time

ENVIRONMENT

Target Computer	S/360
Development Computer	S/360
Language	FORTRAN

RESOURCES

Level of Effort	96.0 staff-months
Project Duration	11.5 months
Peak Staff Level	
Full-Time Equivalent	11.6
Individual Members	12
Average Staff Level	
Full-Time Equivalent	6.0

PROGRAM SIZE

Modules	283
Delivered Lines of Code	55,237
Developed Lines of Code	46,211

EXPERIMENTAL OBJECTIVES

Methodology evaluation
Resource and cost estimation
Software measures and metrics
Reliability and error modeling

DATA COLLECTED

Manpower utilization
Computer utilization
Change/error characteristics
Project summary statistics

PROJECT 21

SOFTWARE CHARACTERISTICS	Scientific/data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	39.6 staff-months
Project Duration	9 months
Peak Staff Level	
Full-Time Equivalent	7.9
Individual Members	7
Average Staff Level	
Full-Time Equivalent	4.4
PROGRAM SIZE	
Modules	374
Delivered Lines of Code	75,420
Developed Lines of Code	31,144
EXPERIMENTAL OBJECTIVES	Evaluate formal training Study extensive reuse of code Methodology evaluation Resource and cost estimation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Computer utilization Product measures Change/error characteristics Project summary statistics

PROJECT 22

SOFTWARE CHARACTERISTICS	Scientific/data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	98.4 staff-months
Project Duration	14.5 months
Peak Staff Level	
Full-Time Equivalent	9.5
Individual Members	14
Average Staff Level	
Full-Time Equivalent	5.6
PROGRAM SIZE	
Modules	604
Delivered Lines of Code	89,513
Developed Lines of Code	67,463
EXPERIMENTAL OBJECTIVES	Methodology evaluation Resource and cost estimation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Computer utilization Product measures Change/error characteristics Project summary statistics

PROJECT 23

SOFTWARE CHARACTERISTICS	Scientific/data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	115.8 staff-months
Project Duration	16 months
Peak Staff Level	
Full-Time Equivalent	8.9
Individual Members	12
Average Staff Level	
Full-Time Equivalent	5.9
PROGRAM SIZE	
Modules	510
Delivered Lines of Code	111,868
Developed Lines of Code	90,157
EXPERIMENTAL OBJECTIVES	Methodology evaluation Resource and cost estimation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Computer utilization Product measures Change/error characteristics Project summary statistics

PROJECT 24

SOFTWARE CHARACTERISTICS	Scientific/data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	90.8 staff-months
Project Duration	12.5 months
Peak Staff Level	
Full-Time Equivalent	10.0
Individual Members	11
Average Staff Level	
Full-Time Equivalent	5.8
PROGRAM SIZE	
Modules	535
Delivered Lines of Code	75,393
Developed Lines of Code	54,531
EXPERIMENTAL OBJECTIVES	Methodology evaluation Resource and cost estimation Software measures and metrics
DATA COLLECTED	Manpower utilization Computer utilization Product measures Project summary statistics

PROJECT 25

SOFTWARE CHARACTERISTICS	Scientific/data processing Interactive Graphics Not real-time
ENVIRONMENT	
Target Computer	S/360
Development Computer	S/360
Language	FORTRAN
RESOURCES	
Level of Effort	98.7 staff-months
Project Duration	17.5 months
Peak Staff Level	
Full-Time Equivalent	8.9
Individual Members	13
Average Staff Level	
Full-Time Equivalent	4.8
PROGRAM SIZE	
Modules	519
Delivered Lines of Code	85,369
Developed Lines of Code	78,580
EXPERIMENTAL OBJECTIVES	Evaluate formal training Use of requirements language tool Methodology evaluation Resource and cost estimation Software measures and metrics Reliability and error modeling
DATA COLLECTED	Manpower utilization Computer utilization Product measures Change/error characteristics Project summary statistics

GLOSSARY

ALC	Assembly Language Code
ATR	Assistant Technical Representative
BMDP	Biomedical Programs, P Series
CAT	Configuration Analysis Tool
CSC	Computer Sciences Corporation
DBA	Data Base Administrator
GESS	Graphic Executive Support System
GSFC	Goddard Space Flight Center
HIPO	Hierarchical Input Processing Output
MPP	Modern Programming Practices
MTTF	Mean Time to Failure
PANVALET	Computer Program Analysis and Security System
PDL	Program/Process Design Language
SAP	FORTRAN Static Source Code Analyzer Program
SEL	Software Engineering Laboratory
SFORT	Structured FORTRAN Preprocessor
STL	Systems Technology Laboratory
TSO	IBM Timesharing Option

REFERENCES

1. University of Maryland, TR-535, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

2. Computer Sciences Corporation, CSC/TM-81/6102, Guide to Data Collection, V. E. Church, F. E. McGarry, and D. N. Card, September 1981

3. --, CSC/SD-81/6011UD1, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, D. C. Wyckoff, September 1981

4. --, CSC/SD-81/6079, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, D. N. Card, September 1981

5. M. Halstead, Elements of Software Science. New York: Elsevier Publishing Co., 1977

6. T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, December 1976, vol. 2, no. 4, pp. 308-320

7. W. J. Dixon and M. B. Brown, BMDP Biomedical Computer Programs. Los Angeles: University of California Press, 1979

8. B. A. Sheil, "The Psychological Study of Programming," Computing Surveys, March 1981, vol. 13, no. 1, pp. 101-120

9. C. E. Walston and C. P. Felix, "A Method of Programming Measurement and Estimation," IBM Systems Journal, January 1977, vol. 16, no. 1

10. R. W. Wolverton, "The Cost of Developing Large Scale Software," IEEE Transactions on Computers, June 1974, pp. 615-636

11. C. Turner and G. Caron, "A Comparison of RADC and NASA/SEL Software Development Data," Data and Analysis Center for Software, Special Publication, May 1981

12. V. R. Basili, "Measuring the Effects of Specific Software Methodologies Within the SEL," Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

13. K. Tasaki, "Evaluation of Draper NAVPAK Software Design," SEL Internal Report, May 1977

14. Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977

15. National Aeronautics and Space Administration/Goddard Space Flight Center, X-582-81-1, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook, December 1980

16. L. H. Putman, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, July 1978, pp. 345-361

17. T. E. Mapp, "Applicability of the Rayleigh Curve to the SEL Environment" (paper prepared for the University of Maryland, May 1978)

18. G. O. Picasso, "Software Engineering Laboratory" (paper prepared for the University of Maryland, June 1979)

19. B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality," Proceedings of the Second International Conference on Software Engineering, 1976

20. V. R. Basili and N. Bailey, "A Meta-Model of Software Development Resource Expenditures," SEL Internal Report, August 1980

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

22. J. D. Musa, "A Theory of Software Reliability and Its Application," IEEE Transactions on Software Engineering, September 1975, vol. 1, no. 3

23. A. B. Miller, "A Study of the Musa Reliability Model" (Master's Thesis, University of Maryland, December 1980)

24. Computer Sciences Corporation, CSC/TM-78/6093, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker and C. E. Goorevich, September 1978

25. --, CSC/TM-79/6263, Evaluation of the Caine, Farber, and Gordon Program Design Language in the GSFC Environment, C. E. Goorevich, September 1979

26. --, CSC/TM-81/6091, SEL Programmer Workbench Phase 1 Evaluation, W. J. Decker, March 1981

27. V. R. Basili and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," SEL Internal Report, December 1980

28. G. Hislop, "Some Tests of Halstead Metrics," SEL Internal Report, December 1978

29. Rome Air Development Center, RADC-TR-77-369, Factors in Software Quality, J. A. McCall, P. K. Richards, and G. F. Walters, November 1977

30. Computer Sciences Corporation, CSC/TM-81/6103, Standard Approach to Software Development, V. E. Church, September 1981

BIBLIOGRAPHY OF SEL LITERATURE

Anderson, L., "SEL Library Software User's Guide," Computer Sciences-Technicolor Associates, Technical Memorandum, June 1980

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

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

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

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 1980

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 the Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

Basili, V. R., and T. Phillips, "Validating Metrics on Project Data," University of Maryland, Technical Memorandum, December 1981

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 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, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

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

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

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

Church, V. E., "User's Guides for SEL PDP-11/70 Programs," Computer Sciences Corporation, Technical Memorandum, March 1980

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)

Mapp, T. E., "Applicability of the Rayleigh Curve to the SEL Environment" (paper prepared for the University of Maryland, December 1978)

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

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

Perricone, B. T., "Relationships Between Computer Software and Associated Errors: Empirical Investigation" (paper prepared for the University of Maryland, December 1981)

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

Software Engineering Laboratory, 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, D. S. Wilson, and R. Beard, 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, January 1978

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

--, SEL-78-003, 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, B. Chu, and G. Page, 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, November 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, S. R. Waligora, and A. L. Green, 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 F. E. McGarry, September 1979

--, SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

--, SEL-80-001, Configuration Analysis Tool (CAT) Functional Requirements/Specifications, F. K. Banks, C. E. Goorevich, and A. L. Green, February 1980

--, SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker, C. E. Goorevich, and A. L. Green, May 1980

--, SEL-80-003, Multimission Modular Spacecraft Ground Support System (MSS/GSSS) State-of-the-Art Computer System/Compatibility Study, T. Weldon, M. McClellan, P. Liebertz, et al., May 1980

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

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

--, SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

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

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

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

--, SEL-81-003, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, D. N. Card, D. C. Wyckoff, G. Page, et al., September 1981

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

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

--, 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

--, SEL-81-008, Cost and Reliability Estimating 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, A. L. Green, and F. E. McGarry, March 1981

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

--, SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

--, SEL-81-012, Software Engineering Laboratory, 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

Turner, C., G. Caron, and G. Brement, "NASA/SEL Data Compendium," Data and Analysis Center for Software, Special Publication, April 1981

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

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

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