SOFTWARE ENGINEERING LABORATORY SERIES

SEL-92-002

DATA COLLECTION PROCEDURES FOR THE SOFTWARE ENGINEERING LABORATORY (SEL) DATABASE

MARCH 1992

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES SEL-92-002

DATA COLLECTION PROCEDURES FOR THE SOFTWARE ENGINEERING LABORATORY (SEL) DATABASE

MARCH 1992

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created to investigate the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1976 and has three primary organizational members:

NASA/GSFC, Systems Development Branch

University of Maryland, Department of Computer Science

Computer Sciences Corporation, Systems Development Operation

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

The major contributors to this document are

Gerard Heller (Computer Sciences Corporation)
Jon Valett (NASA/GSFC)
Mary Wild (Computer Sciences Corporation)

Single copies of this document can be obtained by writing to

Systems Development Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

PAGE ii INTENTIONALLY BLANK

ABSTRACT

This document is a guidebook to collecting software engineering data on software development and maintenance efforts, as practiced in the Software Engineering Laboratory (SEL). It supersedes the document entitled *Data Collection Procedures for the Rehosted SEL Database*, number SEL-87-008 in the SEL series, which was published in October 1987. It presents an overview of SEL data collection and the types of data the SEL collects. It then presents procedures to be followed on software development and maintenance projects in the Flight Dynamics Division (FDD) of Goddard Space Flight Center (GSFC) for collecting data in support of SEL software engineering research activities. These procedures include detailed instructions for the completion and submission of SEL data collection forms.

Table of Contents

Table of Contents (Continued)

List of Illustrations

List of Tables

SECTION 1—INTRODUCTION

The Software Engineering Laboratory (SEL) was established in 1976 to perform research in the measurement and evaluation of the software development process. Over the years, the SEL has collected and analyzed data from nearly 100 software development projects in the Goddard Space Flight Center (GSFC) Flight Dynamics Division (FDD). These data have been used to study both the processes and the products of such development efforts and to evaluate the impact of methodologies, tools, and technologies. In recent years, the SEL has expanded the scope of its activities to include the study of software maintenance and has begun collecting and analyzing data on this phase of the software life cycle.

Fundamental to the research activities of the SEL is the collection of data on development and maintenance projects in the FDD environment. A general introduction to the collection of software engineering data is provided in the SEL *Guide to Data Collection* (Reference 1). That document discusses the motivation for collecting data, the philosophy behind deciding what to collect, resources required, estimated costs, and data management issues. This document is intended to augment that earlier work with an updated overview of the types of data collected by the SEL and detailed procedures for collecting those data.

This document is intended to serve three audiences with overlapping needs. One audience encompasses those who want an overview of the types of data that the SEL collects and how it goes about collecting them. Another audience consists of software managers, developers, and maintainers working on projects being monitored by the SEL. In addition to an overview of the data collection process, members of this audience need specific information about their roles in that process. Thus, the document discusses how and when to complete and submit the various data collection forms used by the SEL; explains how final statistics are determined at project completion; and provides detailed instructions for completing each form. It also gives guidelines as to what communication must regularly take place between monitored projects and SEL data collection personnel. Finally, the document serves as a companion to SEL researchers who need to understand the origins of the data with which they work.

The remainder of the document is organized into three major sections:

- Section 2 provides an overview of the types of data collected from a SEL-monitored development or maintenance project and introduces the mechanisms by which they are collected.
- Section 3 details the data collection process for development projects from startup through system delivery. It also discusses when the data are needed (periodically or keyed to specific events) and whether they are submitted by developers or automatically monitored by the SEL. Samples of the 13 SEL forms associated with development and instructions for how and when to complete them are included.

- Section 4 discusses the transition from development to maintenance and data collection during maintenance. Again, when the data are needed and who provides them are discussed. Samples of the two SEL forms associated with maintenance and instructions for how and when to complete them are included.

- The Appendix contains samples of all SEL data collection forms discussed in this document.

SECTION 2—OVERVIEW OF SEL DATA COLLECTION

This section introduces software engineering data collection as practiced in the SEL. Section 2.1 presents a conceptual overview of the data the SEL collects. This overview is illustrated as a hierarchy of data types and subtypes. The data types in the hierarchy are mapped to the hardcopy forms the SEL uses to collect them.

Section 2.2 presents high-level SEL data collection concepts. It introduces the forms that are central to the SEL's data collection efforts. It categorizes the data collected on the forms with respect to their origin, collection mechanism, and collection frequency. It also summarizes the major data collection activities performed over the software life cycle.

Section 2.3 discusses managing the data collection process, including concepts for validation and feedback of the data.

Note that those discussions represent the practice of data collection as exercised at the time this document was published. The underlying principles guiding the data collection efforts will remain fairly constant. Detailed data collection models, however, are subject to change as life cycles and methodologies evolve, as new technologies replace existing ones, and as the focus of SEL software engineering research activities shifts. Section 2.3 concludes, therefore, with a brief discussion of adapting data collection procedures to meet changing needs.

2.1 DATA COLLECTED BY THE SEL

The basic entity about which the SEL collects data is the project. For each project the SEL monitors, it collects data that characterize various aspects of the project's software life cycle. These aspects include the problem the system is intended to solve, the process followed in solving it, the end product produced, the environment in which that product is developed, and the resources expended along the way. The SEL data collected on a given project can be grouped into the six high-level data types described in Table 2-1.

Tables 2-2 through 2-7 elaborate on the high-level data types introduced in Table 2-1 by breaking them down, via a hierarchy of subtypes, to the level of elemental data items or groups. Note that all SEL data, regardless of their source or collection mechanism, are recorded on hardcopy forms for entry into a central database. The acronyms for these forms appear in Tables 2-2 through 2-7 to provide a mapping from this section's conceptual descriptions of the data to the data collection mechanisms described in the following section. In these tables, when the acronym for a given form is listed next to a given data type, it means that the form is used to collect that data type (and any lower level data subtypes). Thus, for example, all the Size data elements in Table 2-3 are collected on a PCSF. In the same table, however, System Elements are collected on three different forms (PSF, SIF, COF). This is shown on the lower-level table containing the data subtypes that collectively make up System Elements.

Table 2-1. High-Level Classification of SEL Data Types

Type	Description
Estimate data	Data that capture the project leader's size, resource use, and schedule estimates at project start and periodically throughout development.
Product data	Data that characterize the final developed product in terms of size measures and system composition.
Resources data	Data that capture and characterize staff and computer resources expended during development and maintenance.
Process data	Data that characterize the development process: its schedule and profiles of development activities over time.
Change/error data	Data that characterize changes made and errors corrected during development and maintenance.
Annotation data	Data that capture subjective information about a project, the techniques and methodologies used on it, and the data collected on it.

6210G(39)-14

2.2 HOW THE SEL COLLECTS DATA

This section provides an overview of SEL data collection concepts and serves as a bridge from the conceptual view of the data the SEL collects (Section 2.1) to the details of how the SEL collects them (Sections 3 and 4).

As has been noted, the use of hardcopy forms is a central concept in the SEL approach to data collection. There are 15 types of SEL data collection forms, as listed in Table 2-8. Each form is referenced by a 3- to 5-letter acronym. The next four columns in this table describe the source of the data reported on each form, the mechanism by which the data are obtained, the party responsible for completing the form, and the frequency with which the form is submitted. The final column references the section of the document where the form is discussed in detail. Samples of the forms can be found in these reference sections, as well as in the Appendix.

As the table shows, the primary source of SEL data is the development and maintenance personnel working on the projects the SEL monitors. Most of the forms are filled out and submitted directly by developers, project leaders, and maintainers. These forms require the individual completing them to supply basic identification information, such as the team member's name, the project name, and the date. The remaining fields solicit both objective and subjective information, requiring either short answers or selection of options from a checklist. Two forms (the PMF and the

Table 2-2. Estimate Data

Type	Form	Description
Size	PEF	Estimated size of the completed system.
Resource	PEF	Estimated staff resources (in hours) that will be expended in developing the system: technical, management, and services.
Schedule	PEF	Estimated start and end dates for each phase in the development life cycle.

Subtype	Description
Subsystem count	Estimated number of logical subsystems that will be present in the design of the final system.
Component count	Estimated number of separately maintained components that will be present in the final system.
SLOC	Estimated volume of code (SLOC) in the final system, total and broken down by code origin: new, modified, old.

6210G(39)-15

PSF) are completed by SEL personnel based on interviews with development personnel (usually, the project leader).

A second source of SEL data is electronic, computer-based records that SEL personnel monitor on a regular basis. These records include computer resources accounting records (e.g., central processing unit (CPU) time used by user account identifier), project-specific libraries of configured source code, and organizational employee time accounting records. SEL personnel extract data from these sources and record them on SPFs. Although this type of data is not collected directly from the development or maintenance team, the project leader must communicate to SEL personnel exactly what electronic records (user accounts, library names, etc.) are to be monitored.

The third source of SEL data is the development products generated on a project. At the transition from development to maintenance and operations, SEL personnel analyze the source code and documentation being delivered. They also validate and sum the data collected over the development life cycle to compute final resource use statistics. SEL personnel record the data produced by these analyses on a PCSF and verify them in discussions with the project leader.

Table 2-3. Product Data

Type	Form	Description
Size	PCSF	Size measures of the developed product at delivery to operations and maintenance.
System elements	-----	Data that identify and characterize various components of the delivered product.

Subtype	Form	Description
Projects	PSF	Data that identify and categorize projects.
Subsystems	SIF	Data that identify subsystems, associate them with projects, and categorize their function.
Components	COF	Data that identify components, associate them with subsystems, rank their development difficulty, and categorize them by origin, type, and purpose.

Subtype	Description
Subsystem count	Actual count of logical subsystems present in the design of the delivered system.
Component count	Actual count of separately maintained components present in the delivered system.
Documentation pages	Volume of accompanying documentation developed for the delivered system.
SLOC	Count of SLOC in delivered system. Includes total SLOC, total comment lines, and SLOC breakdown by code origin: new, extensively modified, slightly modified, and old.
Executable modules	Count of high-order language components in delivered system that contain executable code. Includes total and breakdown by code origin: new, extensively modified, slightly modified, and old.
Statements	Count of high-order language statements in the delivered system to be processed by a language translator. Includes total and breakdown by code origin: new, extensively modified, slightly modified, and old.
Executable statements	Count of executable high-order language statements in the delivered system to be processed by a language translator. Includes total and breakdown by code origin: new, extensively modified, slightly modified, and old.

6210G(39)-16

The frequencies with which the various SEL forms are collected fall into four broad categories:

- Startup data: Data collected when a project initially comes under SEL monitoring
- Rate data: Data collected regularly with a predefined periodic frequency (weekly, biweekly, monthly, etc.)

Table 2-4. Resources Data

Type	Form	Description
Staff resources	-----	Personnel effort (in hours) expended in developing and maintaining the system.
Computer resources	-----	Computer resources expended in developing the system.

Subtype	Form	Description
Development effort	----	Personnel effort (in hours) expended in developing the system.
Maintenance effort	WMEF	Personnel effort (in hours) expended in maintaining the system - measured weekly, by maintainer, and broken down by class of maintenance performed and by type of maintenance activities performed.

Subtype	Form	Description
Weekly computer resources	SPF	CPU hours logged and number of runs made on development computers during development of the system, measured weekly.
Total computer resources	PCSF	Total CPU hours logged and number of computer runs made on development computers during development of the system.

Subtype	Form	Description
Activity hours	PRF CLPRF	Personnel effort (in hours) expended by developers and first line managers in developing the system - measured weekly, by developer, and broken down by type of development activity performed and by special activities in which the SEL has current research interests.
Services hours	SPF	Personnel effort (in hours) expended by support services personnel in developing the system - measured weekly, by type of support service (secretarial, publications, project management, other).
Total effort	PCSF	Total personnel effort expended in developing the system, broken down by combined total technical and management hours and total services hours.

6210G(39)-19

Table 2-5. Process Data

Type	Form	Description
Schedule	PCSF	Actual start and end dates for each phase in the development life cycle.
Status	DSF	Profiles of progress achieved toward development goals and of open item closure activity.
Growth	SPF	System growth measured by profiles over time of source code library statistics: number of components and total SLOC.

Subtype	Description
Requirements	Profiles over time of open items that reflect requirements stability: requirements questions asked vs. requirements questions answered and specification modifications received vs. specification modifications implemented.
Design	Profile over time of design progress: units planned vs. units designed.
Implementation	Profile over time of implementation progress: units planned vs. units implemented.
Testing	Profiles over time of system and acceptance testing progress: number of tests planned vs. number of tests run one time vs. number of tests passed.
Discrepancies	Profile over time of closure of reported discrepancies: number of discrepancies reported vs. number of discrepancies resolved.

6210G(39)-18

- Event data: Data collected when specific milestones or development events occur (e.g., start of project, phase transitions, configuration of a component, implementation of a change)
- Data at completion: Data collected at the transition to maintenance and operations and that summarize the development portion of the life cycle

These categorizations of data collection frequency are the basis for the organization of Sections 3 and 4.

The SEL views the software life cycle according to a traditional "waterfall" model of non-overlapping sequential phases. Although phases often overlap in practice, a sequential model is used to simplify classification and analysis of the data. The phases that make up this model, discussed in more detail in the *Manager's Handbook for*

Table 2-6. Change/Error Data

Type	Form	Description
Development changes/errors	-----	Data that characterize changes made to configured source code during development.
Maintenance changes/errors	MCRF	Data that characterize changes made to the system during operations and maintenance: includes classifications of the type of maintenance being performed, the cause of the change, the effort spent on the change, the objects changed, and the volume and characteristics of the changed code.

Subtype	Form	Description
Change information	CRF	Effects of the change (components changed), timeframe in which change was implemented, effort spent on change, categorization of change type, Ada-specific change information.
Error information	CRF	Categorization of error by source and class, characteristics of error, Ada-specific error information.
Change profile	SPF	Profile of development changes over time, measured by monitoring version numbers in the configured source library.

6210G(39)-17

Table 2-7. Annotation Data

Type	Form	Description
Data collection information	PSF	Information used by SEL data collection personnel to monitor data collection activities for a given project; collected at project startup and periodically thereafter as conditions change.
Messages	PMF	Free form messages for annotating information about a project, the methodologies or life cycle it follows, its relationship to other projects, or the data collected on it.
Subjective information	SEF	Subjective rankings that characterize the problem solved, the process followed, the development environment, the resources available, and the quality of the product.

6210G(39)-13

Table 2-8. SEL Data Collection Forms

Acronym	Form name	Source	Mechanism	Responsibility	Schedule	Reference Section
CCF	Component Change Form	Development personnel	Form completion	Project leader	Event	3.2.2.3
CLPRF	Cleanroom Personnel Resources Form	Development personnel	Form completion	Developers	Rate	3.2.1.1
COF	Component Origination Form	Development personnel	Form completion	Developers	Event	3.2.2.2
CRF	Change Report Form	Development personnel	Form completion	Developers	Event	3.2.2.4
DSF	Development Status Form	Development personnel	Form completion	Project leader	Rate	3.2.1.2
MCRF	Maintenance Change Report Form	Maintenance personnel	Form completion	Maintainers	Event	4.2.2.1
PCSF	Project Completion Statistics Form	System products/ development data	Analysis	SEL personnel	At completion	3.3.1
PEF	Project Estimates Form	Development personnel	Form completion	Project leader	Startup/ rate/event	3.1.2
PMF	Project Messages Form	Development personnel	Interview	SEL personnel	Event	3.2.2.5
PRF	Personnel Resources Form	Development personnel	Form completion	Developers	Rate	3.2.1.1
PSF	Project Startup Form	Development personnel	Interview	SEL personnel	Startup/ event	3.1.1
SEF	Subjective Evaluation Form	Development personnel	Form completion	Project leader	At completion	3.3.2
SIF	Subsystem Information Form	Development personnel	Form completion	Project leader	Event	3.2.2.1
SPF	Services/Products Form	Computer records	Monitoring	SEL personnel	Rate	3.2.1.3
WMEF	Weekly Maintenance Effort Form	Maintenance personnel	Form completion	Maintainers	Rate	4.2.1.1

6201G(39)-20

Software Development (Reference 2), are shown along the bottom of Figure 2-1. The bars and wedges on the figure illustrate the portions of the life cycle over which each SEL form is collected.

Thus. the primary data collection activities the SEL performs over the software development and maintenance life cycle are collecting and validating data collection forms, monitoring and recording data from electronic records, and analyzing development products at the transition from development to maintenance and operations. These activities depend on regular communication between data collectors and developers to ensure that the developers understand what they are expected to provide and that the correct computer records are being monitored.

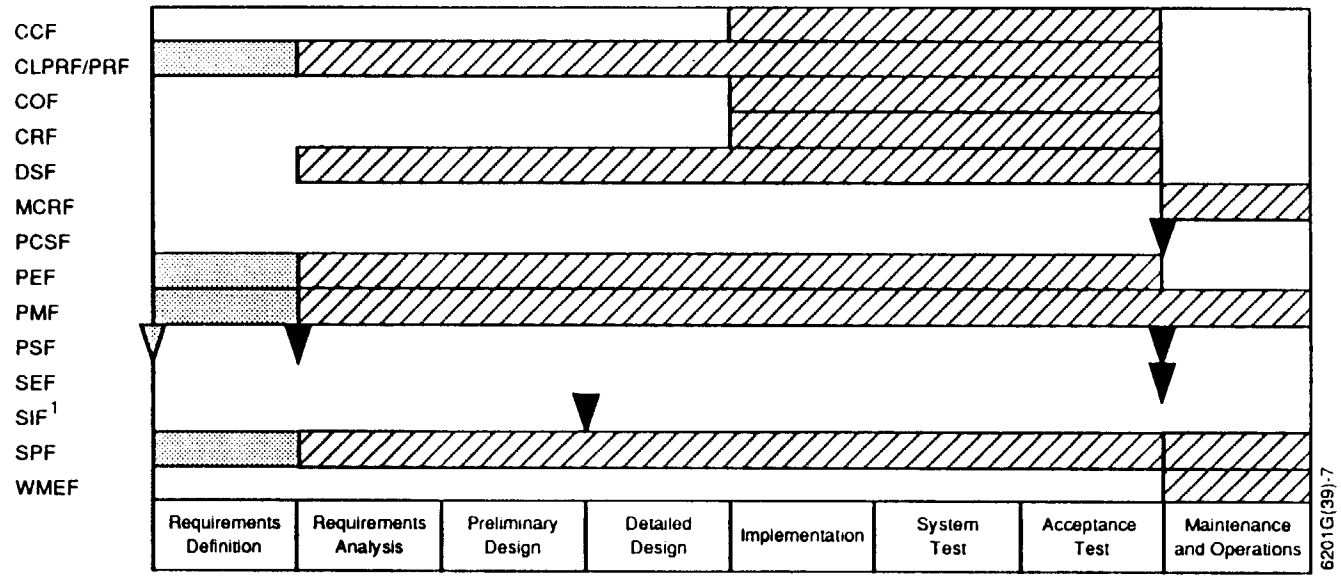
2.3 MANAGING THE DATA COLLECTION PROCESS

To be used effectively in analyzing and evaluating methodologies and technologies as well as in the day-to-day monitoring and controlling of active projects, the data collected must be available in an easily accessible electronic format and must be monitored for accuracy and completeness. Staff resources must be allocated to perform these database maintenance and data monitoring functions. The SEL has a dedicated team of database programmers, data collection experts, and data librarians whose sole task is to collect data, monitor the data collection process, and make the data available to researchers and managers.

SEL data are stored under a relational database management system (RDBMS) hosted on a Digital Equipment Corporation (DEC) VAX-series computer located in the Systems Technology Laboratory (STL) at GSFC. The data collection team is responsible for collecting SEL forms from development and maintenance projects, entering the data from the forms into the database, ensuring the quality of the data entered, and performing the automatic monitoring of computer records and development products mentioned earlier. It is also responsible for designing, implementing, and maintaining the database and its supporting application software (References 3 through 5).

One of the most critical functions of the data collection team is to monitor the data collection process to ensure that the data in the database are as accurate and complete as possible. This goes beyond verifying that what a developer enters onto a form is actually entered into the database correctly. It also involves ensuring that the developer has completed the form correctly, both from a mechanical standpoint and from the standpoint of correctly understanding and interpreting the questions on the form. This is a more complex task. Clearly, the data collection team cannot check every form submitted to make sure that the developer interpreted the questions correctly. Nor can it follow the day-to-day details of the more than 20 projects being monitored at any given time to judge the correctness of the data supplied.

Thus, it is crucial to establish regular two-way communication between developers and the data collection team. Figure 2-2 illustrates the communication paths used by the



[1] Required at Preliminary Design Review (PDR). Updates may be submitted thereafter.

Standard collection period

Typically, SEL data collection begins at requirements analysis. When requirements definition is included, some forms are collected earlier, as indicated.

Standard collection point

Figure 2-1. SEL Forms Collected by Phase

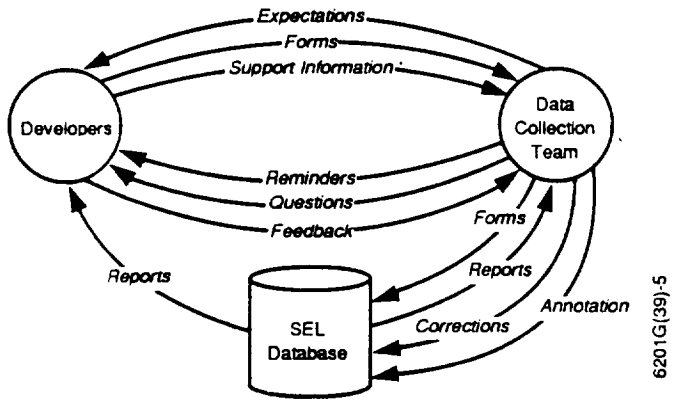


Figure 2-2. Communication Paths in Monitoring SEL Data Collection

SEL in monitoring data collection. The team must first make the developers aware of the SEL's expectations by distributing copies of this document to developers and by meeting with development teams at the start of a new project and at phase transitions when new forms are introduced. The developers then supply the data collection team with data—SEL forms as well as support information the team needs to monitor computer records and development products. The data collectors periodically send out reminders to the developers when data types that are not submitted on a routine basis are due or when data that were expected are not received.

In addition, once data are entered into the database, the data collection team generates periodic reports, both tabular and graphical, that summarize the data being collected. These reports are distributed to the leaders of development and maintenance projects for their analysis and to see whether the data being collected support their own intuitive understanding of what is happening on their projects.

The data collection team also performs its own analysis of database reports and questions the developers about unusual data points or trends identified in the data. Both of these analyses result in feedback to the data collection team, which allows it to correct problems in the database or annotate the data to indicate unusual circumstances.

Finally, the data collection process must be flexible enough to respond to changing needs. Often, a new technology or a new methodology will be introduced that requires changes in the data collected. This has happened many times in the SEL's history. To study the impact of Ada, for example, additional change characteristics specific to the use of Ada were added. More recently, the SEL has been investigating the effectiveness of the cleanroom methodology (Reference 6), which required different classifications of effort data. It is the job of the data collection team to respond to such changes in data collection needs by evaluating the effect of a proposed change on the design of the database structure and supporting application software, data collection procedures, and database documentation. It also involves ensuring that proposed changes do not

make data collected on earlier projects obsolete and invalidate comparisons among those projects and more recent projects. Once a change is planned, the team must communicate the nature of the change to both developers and database users and coordinate its implementation in a manner that minimizes its impact and inconvenience.

SECTION 3—DATA COLLECTION IN THE DEVELOPMENT LIFE CYCLE

This section presents detailed data collection procedures for the development phases of the software life cycle, i.e., from requirements definition through delivery of the system for maintenance and operations, following the life-cycle model referenced in Section 2. It is subdivided to follow the life cycle chronologically. It first discusses data collection activities at project startup, followed by data collection performed through all development phases, and concluding with the data collection activities at project completion. In each section, it provides detailed instructions for completing the corresponding SEL data collection forms. These instructions begin with a discussion of general background information needed to understand how the form is to be completed. This is followed by line-by-line instructions for completing the form. A set of helpful hints and caveats concludes the form instructions.

3.1 PROJECT STARTUP

When a project to be monitored by the SEL is initiated in the FDD, the first thing the SEL must do is establish the lines of communication with the development team and obtain some basic information about the project. To do this, a SEL data collection team member (usually the Database Administrator (DBA), who is the primary point of contact for SEL data collection issues) schedules a meeting with the project leader. This meeting is usually held at the beginning of the requirements analysis phase. If requirements are defined by the development team, however, it may be held at the beginning of requirements definition.

The purposes of this meeting are (1) to ensure that the project leader and developers understand their role in the data collection process, (2) to establish acronyms and naming conventions to be used in completing SEL forms, and (3) to give data collection personnel an understanding of the application and any unique characteristics of the development methodology being employed.

The first purpose is accomplished by distributing this document to the developers and reviewing with the project leader both when and where SEL forms are submitted and the instructions for the forms that are collected during the early life-cycle phases. The general project information and naming conventions collected at this meeting are recorded on a PSF. Descriptions of the application and unique characteristics of the methodologies being employed are also recorded on the PSF.

Topics that should be covered when discussing unique aspects of the project include any approach or methodology that is new to the environment. Recent examples include object-oriented design and cleanroom development. The discussion should also cover any nonstandard approaches to development, such as a prototype or spiral life cycle, or whether the project will be implemented in builds or releases. For SEL purposes, a

"build" refers to a multistage implementation phase. Each build may culminate in a distinct system test phase. A "release" refers to a life cycle where each successive iteration of the system requires separate design, implementation, system test, and, often, acceptance test phases. In other words, each release has a distinct life cycle and usually results in a delivered product. As a general rule, a project that is implemented in multiple builds should be treated as a single project in the SEL database. Projects that are developed in releases generally require each release to be treated as a separate project in the database.

In cases where the life cycle to be followed by the project does not conform to the SEL sequential phase model, the project leader and SEL data collection personnel should agree on how the actual life cycle will map to SEL phase definitions. This information as well should be documented at the startup meeting.

The final information to be obtained at project startup is an initial set of project estimates. These estimates are recorded on a PEF and include the following types of information: gross estimates of resources, software product sizes, dates on which the development life-cycle phases of the project are scheduled to start, and a projected project end date. The estimates provided reflect the project size and resource expenditure when the software is delivered for maintenance and operations.

3.1.1 Project Startup Form (PSF)

General Information

The PSF (Figure 3-1) is a template used by SEL data collection personnel for recording information collected at the project startup meeting. It is not filled out directly by developers. The information recorded on it allows the data collectors to initialize the project in the database so that the data librarians may begin collecting and entering the standard forms collected during development. In addition to being used at the startup of a development project, this form can be used to document similar information at a project's transition to maintenance and operations, a use of the form discussed further in Section 4. The instructions that follow focus on its use at the beginning of development.

The information recorded on the PSF may change during the project. It is not necessary to complete another PSF to document these changes. The data collection team distributes a monthly data collection status report for each active project being monitored. The report lists project personnel, computer accounts, configured source library names, computer systems, forms being collected, and task numbers. It is updated each month by the project leader so that the data collectors can keep this "data about the data collection" current on each project. A discussion of this report can be found in Section 3.2.1.3, along with a sample report page.

The initial general messages should be supplemented during development and at the transition to maintenance. This happens when a development team member (usually

PROJECT STARTUP FORM

Name: _____

Project: _____ **Date:** _____

PLEASE PROVIDE ALL AVAILABLE INFORMATION

Project Full Name: _____

Project Type: _____

Contacts: _____

Language: _____

Computer System: _____

Account: _____

Task Number: _____

Forms To Be Collected: (Circle forms that apply)

PEF PRF CLPRF DSF SPF SIF COF CCF CRF SEF PCSF WMEF MCRF

General Notes: _____

Personnel Names (indicate with * if not in database):

6201G(13) 36

NOVEMBER 1991

Figure 3-1. Project Startup Form

the project leader) communicates information to SEL personnel that explains any unique characteristics of the project, unusual data points or trends, or anything unusual about the data collection process. The SEL data collector records this information on a PMF (see Section 3.2.2.5) and enters it into the database.

Line-by-Line Instructions

Name: Enter the name of the data collection team member (usually the SEL DBA) conducting the startup meeting and completing the form.

Project: Enter the acronym (up to 8 characters) by which the project will be referred, as agreed to with the project leader at the startup meeting. This acronym will be used by all developers on all subsequent SEL forms to be submitted for the project.

Date: Enter the date on which the project startup meeting is held.

Project full name: Enter the complete project name, with spacecraft and application acronyms fully spelled out.

Project type: While discussing the nature of the application with the project leader, determine which of the following project categories best describes the application, and enter the project type code onto the form:

Type Code	Description
AGSS	Spacecraft Attitude Ground Support System
SIMULATOR	Spacecraft Dynamics or Telemetry Simulator
MP&A	Mission Planning and Analysis System
GRAPH/UI	Graphical display or user interface system
ATTITUDE	Attitude application (not an AGSS or simulator) – may be mission-specific or general use
ORBIT	Orbit application – may be mission-specific or general use
REALTIME	Real-time data processing or control application
DATABASE	Database support application – data entry, report generation, etc.
TOOL	Software development or management tool
OTHER	Application that does not fall into any of the above categories

Contacts: Enter the names of contractor and GSFC personnel responsible for managing the project. This includes the GSFC assistant technical representative (ATR), the contractor task leader, and the contractor section manager, one of whom is usually designated the key point of contact (project leader) for data collection purposes.

Language: Enter the primary language in which the application is being developed, along with any other languages used on the project.

Computer system: Enter the computer system on which the project is being developed, as well as that for which the system is targeted, if different. Enter all types of machines to be used if the system is distributed across multiple hardware platforms.

Account: Enter the names of the computer accounts to be monitored automatically by SEL personnel for CPU hours used and number of runs made. In the Flight Dynamics Facility (FDF) mainframe environment, this is the sponsor code. In the STL VAX environment, this is the group identifier common to all user identifiers for personnel working on the project. These accounts may not be known at project startup. If this is the case, the project leader should be reminded to communicate that information to the SEL as soon as it is available.

Task number: If the project is being developed by a contractor development team, enter the task number under which the contractor work is being performed. This is used by the SEL in automatically monitoring certain types of effort data extracted from personnel timekeeping records.

Forms to be collected: Circle the forms to be collected on the project. The standard set of forms collected on a typical project includes the PEF, either the PRF or the CLPRF (depending on the development methodology being followed), the COF, the CCF, the CRF, the SEF, and the PCSF. Usually, the DSF and the SPF are also collected, but there are cases where one or both may not be collected.

General notes: Enter free-format text that describes the application and any unique features about the development life cycle or methodology or about the data collection being performed on the project.

Personnel names: Enter the names of personnel who will be submitting forms on the project. If an individual has submitted forms on other projects in the past, use his/her name as it already appears in the database. For someone new to SEL data collection, record his/her full name and establish a database name, which is normally first initial followed by last name. In cases where this name conflicts with an existing name in the database, a variation may be used, such as adding the middle initial. New names should be identified by an asterisk, indicating that data collection personnel must enter them into the database. Be sure to leave a list of new names with the project leader and instruct him/her to have developers use the agreed-upon names when completing forms.

Helpful Hints

1. Determining a project type is sometimes a source of confusion. One variation arises when a subsystem of an Attitude Ground Support System (AGSS) is developed by a separate development team and tracked as a separate project by the SEL. This type of project should get a project type of AGSS,

and there should be a note in the general messages connecting it to the other projects being monitored that make up the AGSS.

One technique for deciding the project type is to go through the list of types in the order presented above and select the first category that applies.

2. There should be a single contact designated as project leader. This person will be responsible for all communication with the data collection team regarding data collection matters. He/she will also be the person who completes PEFs and DSFs, and to whom the SEL sends reminders when information is due.

3. The SEL does not monitor computer-use accounting information for work done on personal computers (PCs). If a major portion of actual development work (coding and testing) is being done on PCs, the SEL will not monitor any computer-use information for the project. This does not, however, include using PCs merely to generate documentation or design diagrams.

4. When determining what forms will be collected, the SPF (which is completed by data collection personnel with automatically monitored data) will not be submitted if none of the three types of data recorded on it can be monitored. For example, SPFs would not be submitted for an all-GSFC project (having no contractor timekeeping records to monitor) in which development was being performed on PCs (prohibiting the collection of both computer-use and growth data).

3.1.2 Project Estimates Form (PEF)

General Information

The PEF (Figure 3-2) is submitted by the project leader at startup and every 6 to 8 weeks thereafter throughout the development phases of the project life cycle. It is also submitted when new estimates are made at project milestones. This information provides a historical record of project size, schedule, and resource-use estimates that may be used in analyzing the project when completed. These estimates should be projections for the delivered system and should not include anticipated changes and expenditures in the maintenance and operations phase.

A project leader may submit a PEF at any time to inform the SEL of updated estimates. The SEL, however, sends out reminders that new estimates are due when 6 weeks have transpired since the last form was received. These reminders are in the form of a PEF completed by SEL personnel with the estimate values submitted on the most recently received form. Upon receiving this reminder, the project leader is to enter the form date and mark up the existing estimates on the form to indicate updates. If the project leader has not made a new estimate since the last PEF received, he/she may simply enter the form date and mark on the form that no updates need be made.

PROJECT ESTIMATES FORM

Name: _____

Project: _____ Date: _____

Phase Dates (Saturdays)	
Phase	Start Date
Requirements Definition	
Design	
Implementation	
System Test	
Acceptance Test	
Cleanup	
Project End	

Staff Resource Estimates	
Programmer Hours	
Management Hours	
Services Hours	

Project Size Estimates	
Number of subsystems	
Number of components	
Source Lines of Code	
Total	
New	
Modified	
Old	

Note: All of the values on this form are to be estimates of projected values at completion of the project. This form should be submitted with updated estimates every 6 to 8 weeks during the course of the project.

For Librarian's Use Only
Number: _____
Date: _____
Entered by: _____
Checked by: _____

6201G(13)-16

NOVEMBER 1991

Figure 3-2. Project Estimates Form

As the project proceeds through its life-cycle phases, the phase dates supplied for completed phases should represent the **actual** dates on which the phase transition occurred. The remaining estimate information is always a **projection** of resources used and project size at the end of the project. Resource estimates in terms of staff-hours are provided for technical, management, and services personnel. Project size estimates include the number of subsystems, components, and SLOC. The latter includes the total SLOC and a breakdown by new, modified, and old SLOC.

Line-by-Line Instructions

Name: Enter the SEL database name of the project leader completing the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored.

Date: Enter the date on which the form is completed.

Phase Dates

Enter the estimated start dates (or actual start dates for phase transitions that have already passed) for each of the listed phases that apply to the project. These phases map to the SEL life cycle introduced in Section 2 with two exceptions:

1. The design phase on the PEF encompasses the requirements analysis, preliminary design, and detailed design phases of the SEL model.
2. The PEF includes a cleanup phase that starts when the system is accepted and ends with the delivery of system products to a maintenance and operations team.

To supply accurate phase dates on the PEF, it is necessary to understand the definition of the phases in the SEL life-cycle model. The SEL uses the "waterfall" model to define this series of phases. The SEL realizes that, in practice, there is normally a period of overlap during the transition from one phase to the next. However, it is recommended that discrete events be used to signal the end of one phase and the beginning of the next phase. SEL phase definitions and guidelines for beginning new phases are summarized in Table 3-1. A more complete definition of project phases can be found in the *Recommended Approach to Software Development* (Reference 7).

If the project is following a nonstandard life cycle, the phases of that life cycle should have been mapped to the SEL phases in an agreement discussed at the project startup meeting. If, for example, each build in the implementation of a given development effort includes its own system test phase, it might be agreed that, for the SEL PEF, the system test phase begins when integration testing is complete for the first (or second, or last) build. The key is to have identified ahead of time a discrete event to be used to signal the phase transition.

Table 3-1. SEL Phase Definitions

SEL Phase	Activities	Start Date
Requirements Definition	Define requirements Write requirements and functional specifications	Start of project. Note: only applicable if developers are responsible for generating system requirements
Design	Requirements analysis Preliminary design Detailed design Design reviews and responses to questions	Delivery of requirements and functional specifications or start of software development task
Implementation	Code, read, and unit test modules Integration and integration testing	Critical Design Review (CDR)
System Test	End-to-end testing of integrated system	Successful completion of integration testing
Acceptance Test	Execution and evaluation of acceptance tests	Successful execution of all system tests
Cleanup	Generation of system tape Completion of system documentation	Acceptance of the system

6201G(13)-34

Phases must begin on a Saturday. Saturday dates are used to avoid any ambiguity as to which phase a given piece of data belongs, since much of the weekly rate data the SEL collects are tagged with a Friday date to represent the previous week's activity. If the event that signals the transition from one phase to the next occurs on a weekday (as it is most likely to do), the date of the nearest Saturday should be used. If a given phase is not included in the life cycle of a particular project, a start date for that phase need not be provided. For example, if the start date for acceptance test is omitted, it will be assumed that system test runs from the system test start date to the cleanup start date. By the same token, phases should not begin and end on the same date. At least one phase date and a project end date must be provided. If they are not, the PEF will be rejected at data entry.

Programmer hours: Enter the total technical hours projected to be expended by the delivery of the operational software product. (See hint 2.)

Management hours: Enter the total management hours projected to be expended by the delivery of the operational software product. (See hint 2.)

Services hours: Enter the total support services hours projected to be expended by the delivery of the operational software product. (Support services personnel include secretaries, librarians, technical publications personnel, couriers, etc.)

Number of subsystems: Enter the projected number of subsystems that will be included in the design of the finished product. Subsystems are defined as a logical partitioning of the system design. This should not be confused with the mutually exclusive partitioning of system components used to define subsystem prefixes (see Section 3.2.2.1). For example, FORTRAN COMMON block components may be referenced in more than one logical subsystem and may be grouped under a separate subsystem prefix. They would not be counted as a separate subsystem of the system design, however. For object-oriented designs, logical subsystems are sometimes nested within other subsystems. Each of these logical subsystems should be counted when providing the subsystem estimate. Also see hint 4 for a discussion of reuse and size estimation.

Number of components: Enter the projected number of components that will be delivered as part of the finished software product. A component is defined as the lowest level configuration item of the system or the smallest piece of the system maintained in its own file. Components include source code (FORTRAN subroutines, functions; Ada procedures, package specifications; display panels written in a graphics language; assembly language routines, etc.) as well as data files that are configured elements of the delivered system (definitions of screen displays, translation tables, etc.). Components do not include data files used to test the system or command procedures used to build the system, as these will vary in operational use and are usually not delivered as configured pieces of the system. Also see hint 4 for a discussion of reuse and size estimation.

Source lines of code—total: Enter the projected total SLOC for the delivered software product. A source line of code is defined as a carriage return or card image within a component. This includes blank lines, comments, code, and data. The total SLOC must equal the sum of the new, modified, and old code estimates on the following lines. Also see hint 4 for a discussion of reuse and size estimation.

Source lines of code—new: Enter the projected total new SLOC that are to be developed for the system. Also see hint 4 for a discussion of reuse and size estimation.

Source lines of code—modified: Enter the projected total SLOC that are to be reused from other sources with modifications to meet the requirements of the system. Also see hint 4 for a discussion of reuse and size estimation.

Source lines of code—old: Enter the projected total SLOC that are to be reused from other sources with no modification. Also see hint 4 for a discussion of reuse and size estimation.

Helpful Hints

1. One of the most common errors in completing the PEF is to provide a requirements definition date when the requirements definition phase is not monitored by the SEL. This results from confusing requirements definition with requirements analysis. Remember that, for purposes of SEL phase dates, requirements analysis is part of the design phase.

2. One area of potential confusion is the distinction between technical and management hours. Generally, estimation algorithms predict management effort as some percentage of total effort or some percentage of the technical effort that is added in to compute total effort. For early estimates, this is the number that should be used. The confusion arises when a project leader later tries to update the estimate based on actual and projected expenditures recorded as "Project Management" hours, which are automatically collected by SEL personnel from accounting records (see discussion under the SPF instructions, Section 3.2.1.3). Those hours do not include management hours charged by line managers, who record both technical and management hours on PRFs (Section 3.2.1.1). The proper way to update the estimate is to ignore how management charges are reported to the SEL (whether via SPF or PRF) and use actual and projected expenditures for all charges made to management accounts in the timecard accounting system being used to track these charges.

3. It is expected that early estimates will be coarse but will improve as more is learned about the system and updated estimates are submitted. Thus, some of the information requested on the PEF may not be known for the earliest estimates. The project leader is encouraged to think about all of the items on the form and try to come up with an estimate of each. If this is not possible, however, the minimum set of estimates required on the form includes a set of phase dates (including a project end date on which the system will be delivered), and estimates of technical hours, management hours, number of components, and total SLOC. More complete estimates will then be expected on subsequent forms.

4. Questions often arise as to how reuse should be treated in supplying size estimates. Obviously, reused source code that is copied into the project configured library, maintained there, and delivered as part of the system should be included in both the component count and the SLOC counts. It is a little less clear how to treat reused software that is maintained separately and linked in to the system. The guideline here is that if it is linked in from an institutionally maintained tool (such as a graphics display tool or vendor-supplied, language-specific library of mathematics routines), it **should not** be counted. If, however, the reused software is linked in from a separately maintained application or a generic set of application-specific software designed to be reused, this software **should** be included in the size estimates, whether it involves linking in single components or entire subsystems. Such reuse counts should be reflected in the subsystem, component, total SLOC, and old SLOC estimates. Examples of linked-in reuse that should be counted in the FDD environment are the Multimission Three-Axis Stabilized Spacecraft (MTASS) and Multimission Spin-Axis Stabilized Spacecraft (MSASS)

generic attitude ground support applications (References 8 and 9, respectively) and the Code 550 Reusable Software Library (RSL) (Reference 10).

3.2 DATA COLLECTION DURING DEVELOPMENT

Once startup information and an initial set of estimates have been collected, SEL data collection continues through the development life-cycle phases. As discussed in Section 3.1.2, the SEL continues to collect estimate data on PEFs at major project milestones and at 6- to 8-week intervals, sending reminders to project leaders when updates are due. In addition, the SEL begins to collect rate data. As can be seen by examining Table 2-8 and Figure 2-1, the rate data forms include the PRF (CLPRF), the DSF, and the SPF. Rate data and the forms used to collect them are discussed in Section 3.2.1. Figure 2-1 also shows the event data forms collected through development. These include the CCF, the COF, the CRF, the PMF, and the SIF. Event data and the corresponding forms are discussed in Section 3.2.2.

3.2.1 Rate Data

Rate data collected by the SEL originate from two sources. One source is the forms completed by the developers on the project. These forms include the PRF and its cleanroom variation (CLPRF), and the DSF. The other source of rate data is automatic monitoring performed by the SEL data collection team. The data types monitored automatically include growth, computer resources, and a subclass of effort data that is not recorded by developers on the PRF. SEL personnel record these automatically monitored rate data on an SPF. These four forms are discussed in the following sections.

3.2.1.1 PERSONNEL RESOURCES FORM (PRF) AND CLEANROOM PERSONNEL RESOURCES FORM (CLPRF)

General Information

Effort data are collected on either a PRF (Figure 3-3) or a CLPRF (Figure 3-4). Which resource form is used is determined by the methodology used to develop the software. Developers on projects following the cleanroom methodology complete the CLPRF; those on all other projects complete the regular PRF.

The PRF details the standard development activities performed during a given week and identifies how many hours were expended on each of them. The PRF also contains an area for recording hours spent on special activities. A special activity is any activity of current specific interest to SEL researchers, such as rework, documentation, or training in a new methodology. The CLPRF differs from the PRF in that the standard development activities and special activities are geared to accommodate the study of cleanroom techniques.

A PRF/CLPRF is submitted weekly by every member of the development team who performs technical work on the project. This includes managers who perform both

Personnel Resources Form

Name:_____

Project:_____ Date (Friday): _____

SECTION A: Total Hours Spent on Project for the Week:_____

SECTION B: Hours By Activity (Total of hours in Section B should equal total hours in Section A)

Activity	Activity Definitions	Hours
Predesign	Understanding the concepts of the system. Any work prior to the actual design (such as requirements analysis).	
Create Design	Development of the system, subsystem, or components design. Includes development of PDL, design diagrams, etc.	
Read/Review Design	Hours spent reading or reviewing design. Includes design meetings, formal and informal reviews, or walkthroughs.	
Write Code	Actually coding system components. Includes both desk and terminal code development.	
Read/Review Code	Code reading for any purpose other than isolation of errors.	
Test Code Units	Testing individual components of the system. Includes writing test drivers.	
Debugging	Hours spent finding a known error in the system and developing a solution. Includes generation and execution of tests associated with finding the error.	
Integration Test	Writing and executing tests that integrate system components, including system tests.	
Acceptance Test	Running/supporting acceptance testing.	
Other	Other hours spent on the project not covered above. Includes management, meetings, training hours, notebooks, system descriptions, user's guides, etc.	

SECTION C: Effort On Specific Activities (Need not add to A)
(Some hours may be counted in more than one area; view each activity separately)

Rework: Estimate of total hours spent that were caused by unplanned changes or errors. Includes effort caused by unplanned changes to specifications, erroneous or changed design, errors or unplanned changes to code, changes to documents. (This includes all hours spent debugging.) ☐

Enhancing/Refining/Optimizing: Estimate of total hours spent improving the efficiency or clarity of design, or code, or documentation. These are not caused by required changes or errors in the system. ☐

Documenting : Hours spent on any documentation of the system. Includes development of design documents, prologs, in-line commentary, test plans, system descriptions, user's guides, or any other system documentation. ☐

Reuse: Hours spent in an effort to reuse components of the system. Includes effort in looking at other system(s) design, code, or documentation. Count total hours in searching, applying, and testing. ☐

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

5150G(21) 38

NOVEMBER 1991

Figure 3-3. Personnel Resources Form

Personnel Resources Form

(CLEANROOM VERSION)

Name:_____

Project:_____ Date (Friday): _____

SECTION A: Total Hours Spent on Project for the Week:_____

SECTION B: Hours By Activity (Total of hours in Section B should equal total hours in Section A)

Activity	Activity Definitions	Hours
Predesign	Understanding the concepts of the system. Any work prior to the actual design (such as requirements analysis).	
Pretest	Developing a test plan and building the test environment. Includes generating test cases, generating JCL, compiling components, building libraries, and defining inputs and probabilities.	
Create Design	Development of the system, subsystem, or components design. Includes box structure decomposition, stepwise refinement, development of PDL, design diagrams, etc.	
Verify/Review Design	Includes design meetings, formal and informal reviews, and walkthroughs.	
Write Code	Actually coding system components. Includes both desk and terminal code development.	
Read/Review Code	Code reading for any purpose other than isolation of errors. Includes verifying and reviewing code for correctness.	
Independent Test	Executing and evaluating tests of system components.	
Response to SFR	Isolating a tester-reported problem and developing a solution. Includes writing and reviewing design or code to isolate and correct a tester-reported problem.	
Acceptance Test	Running/supporting acceptance testing.	
Other	Other hours spent on the project not covered above. Includes management, meetings, training hours, notebooks, system descriptions, user's guides, etc.	

SECTION C: Effort On Specific Activities

Methodology Understanding/Discussion: Estimate the total hours spent learning, discussing, reviewing or attempting to understand cleanroom-related methods and techniques. Includes all time spent in training. ☐

For Librarian's Use Only
Number: _____
Date: _____
Entered by: _____
Checked by: _____

6201G(13) 24

NOVEMBER 1991

Figure 3-4. Personnel Resources Form (Cleanroom Version)

technical and management work. A PRF/CLPRF is required from every team member for each week he/she is assigned to the project, even for weeks in which no hours are worked on the project (e.g., vacation or temporary assignment to another project). The "zero-hour" form is the mechanism by which the SEL data collectors ensure that the effort data collected for a given week are complete. Project leaders receive reminder notices for all team members from whom the SEL does not receive a form in a given week. The SEL maintains a list of developers currently assigned to each monitored project and uses it to generate these reminders. The list is given to project leaders to update each month as part of the data collection status report (see discussion and example in Section 3.2.1.3).

The SEL expects that the project leader will help to assure the quality of data submitted on PRFs and CLPRFs. He/she should spot check the PRFs submitted by team members to ensure that the hours recorded match those the team member charged to the project in the organization's timekeeping system and that the activities under which the team member recorded hours are appropriate for the types of activities being performed on the project.

Line-by-Line Instructions

Name: Enter the SEL database name of the developer completing the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored. Check with the project leader if unsure of the correct name.

Date (Friday): Enter the Friday date corresponding to the end of the week for which hours are being recorded. Data are to be reported on this form for all work performed on the project during the preceding Saturday-through-Friday period.

Section A

Total hours spent on project for the week: Enter the total hours actually worked on the project for the current week. This includes any overtime, whether paid or unpaid. It does not include paid hours not charged to the project, such as sick time, holidays, or vacations. Note that this number must equal the sum of the hours recorded for the development activities in Section B. If partial hours are recorded, enter them in decimal form to the nearest tenth of an hour. Do not enter fractions. This also applies to all of the activity hours in Sections B and C.

Section B for PRFs

Predesign: Enter the number of hours during the current week spent understanding the concepts of the system before any actual design work. This activity includes requirements definition and requirements analysis. It also includes the analysis of any

changes made to requirements or specifications, regardless of where in the life cycle they occur.

Create design: Enter the number of hours during the current week spent performing design activities, such as high-level partitioning of the problem, drawing design diagrams or structure charts, maintaining a data dictionary, specifying components, writing prologs and program design language (PDL), and compiling design notebooks or documents.

Read/review design: Enter the number of hours during the current week spent reviewing design materials. This includes formal design reviews, informal reviews or walkthroughs, and studying the current system design or that of other systems (such as those from which software is being reused).

Write code: Enter the number of hours during the current week spent actually writing code, whether modifying reused components, developing new components, implementing a change, or correcting an error. It includes both desk and terminal time. It also includes writing code when developing prototypes.

Read/review code: Enter the number of hours during the current week spent reading code. This includes desk checking, reviewing the code of other team members, studying old code for potential reusability, and preparing for and attending code inspections. **It does not include studying code to isolate an error.**

Test code units: Enter the number of hours during the current week spent unit testing individual system components. This includes time spent devising test cases, developing test matrices, and coding test drivers and program stubs, as well as time spent actually executing and evaluating tests. **It does not include time spent isolating and correcting errors encountered during the testing.**

Debugging: Enter the number of hours during the current week spent isolating errors in the system and developing a strategy for their solution. This includes time spent studying code, generating and executing special test cases, inserting debug code, and any other steps taken to **isolate** the error. Once the source of the error has been found, however, the time spent implementing the correction and performing regression testing should not be considered debugging time. Rather, it should be recorded under the appropriate designing, coding, and testing activities.

Integration test: Enter the number of hours during the current week spent integrating system components and testing integrated system components. This includes the generation of test plans; execution of build, integration, or release tests; and system testing. **It does not include, however, isolation and correction of errors that were uncovered as a result of such testing.**

Acceptance test: Enter the number of hours during the current week spent executing acceptance tests or supporting the acceptance test team in the execution of such tests. **Do not include time spent isolating and correcting errors that occur during acceptance testing.**

Other: Enter the number of hours during the current week that do not fall into any of the above categories. This category covers such activities as meetings, management, travel, training, configuration management, and documentation.

Section C for PRFs

Rework: Enter the number of hours during the current week spent reworking any portion of the system for any **unplanned** reason. This includes changes to the requirements, unforeseen hardware or software limitations, and correction of errors. These hours should include all hours recorded for the debugging activity in Section B. In addition, however, they should include the hours spent actually correcting errors and testing the corrections. Note that this category is not limited to the rewriting of code but includes redesigning, regression testing, and even updating documentation.

Enhancing/refining/optimizing: Enter the number of hours during the current week spent changing the system to improve the clarity or efficiency of the design or code or to improve system performance. This does not include changes made as a result of unforeseen requirements changes and error corrections.

Documenting: Enter the number of hours during the current week spent generating or updating system documentation. This includes development plans, design documents, in-line comments in code, prologs, test plans, system descriptions, user's guides, and project histories.

Reuse: Enter the number of hours during the current week spent attempting to reuse software from other systems. This includes reuse of design and documentation as well as of actual code. It includes the time spent searching for potential reusable components, evaluating them, modifying them to meet system requirements, if necessary, and testing them. It also includes evaluating the functionality of and interfaces with reused software that is linked to the system, rather than copied in as source code.

Section B for CLPRFs

Predesign: See PRF instructions. This activity is performed by both developers and testers.

Pretest: Enter the number of hours during the current week spent writing a statistical test model, developing a test plan, and building the test environment. This activity includes configuration management, creating job control language (JCL), compiling components, building libraries, and defining inputs and probabilities. This activity is performed by testers only.

Create design: Enter the number of hours during the current week spent performing design activities, such as developing a state machine representation, specifying module functionality, defining data, and writing PDL. This activity is performed by developers only.

Verify/review design: Enter the number of hours during the current week spent reviewing design materials. This includes formal design reviews, informal reviews or walkthroughs, and studying the current system design or that of other systems (such as those from which software is being reused). This also includes reviewing redesign work resulting from resolving software failure reports (SFRs), or from implementing specification modifications. This activity is performed by developers only.

Write code: See PRF instructions. This activity is performed by developers only.

Read/review code: See PRF instructions. This activity is performed by developers only.

Independent test: Enter the number of hours during the current week spent executing and evaluating tests of system components as an independent tester. This activity is performed by testers only.

Response to SFR: Enter the number of hours during the current week spent isolating a problem reported by a tester on an SFR, and developing a solution. This activity is performed by developers only.

Acceptance test: See PRF instructions. This activity is performed by both developers and testers.

Other: See PRF instructions. Note, however, that configuration management is a Pretest activity in the cleanroom methodology and should not be included in the Other category. This activity is performed by both developers and testers.

Section C for CLPRFs

Methodology understanding/discussion: Enter the number of hours during the current week spent learning, discussing, reviewing, or attempting to understand the cleanroom techniques and method. This also includes any training.

Helpful Hint

Perhaps the most common error made by developers in completing the PRF is to assume that the activities under which they record their hours must match the project's current life-cycle phase. They might assume, for example, that if the project is in the preliminary design phase, all of their hours should be recorded under the Create Design and Read/Review Design activities. In fact, they may be spending time reviewing requirements (Predesign), developing prototypes (Write Code, Debugging, etc.), or examining code for potential reuse (Read/Review Code). As another example, a developer resolving a problem during the system test phase may be tempted to charge all of his/her hours to Integration Test, when time spent isolating the problem should really be charged to Debugging, time spent creating a solution should really be charged to Write Code (and possibly to Create Design as well), and time spent retesting the corrected unit should really be charged to Test

Code Units. It is extremely important to remember that PRF activities do not map directly to calendar phases and to report the time spent performing each activity as accurately as possible.

3.2.1.2 DEVELOPMENT STATUS FORM (DSF)

General Information

The DSF (Figure 3-5) is used to record status data, requirements measures, and discrepancies. Status data are measured as progress toward a target goal. Requirements measures and discrepancies are measured as a number of reported events that require a response and the number of those events to which a response has been made.

As shown in Figure 2-1, DSF collection begins in requirements analysis and continues through system delivery. Requirements measures are recorded through that entire span. Status data are added in detailed design, and discrepancies are added in system test.

It is the responsibility of the project leader to complete the DSF biweekly (every other week). Since this deviates from the weekly norm for rate data, the SEL distributes to each project leader a DSF preprinted with the most recently submitted data for his/her project in weeks when a DSF is due. This way, the project leader need only mark up the form to indicate values that have changed since the last form submitted (Figure 3-6). There is a box on the form to check if no changes are necessary.

It is anticipated that project leaders are tracking the types of data collected on the DSF and have their own mechanisms and tools for doing so. There may, however, be cases in which not all of the data types are being tracked. A small project, for example, that developed its own requirements may not go through the formality of using requirements question-and-answer forms. Thus, the DSF is intended to capture whatever project leaders are measuring. They are not required to synthesize data that they would not normally track in the course of managing the project.

Status data are generally monitored only until the target value is reached. Usually, this corresponds to a calendar phase of the life cycle. There may, however, be periods of overlap during which more than one type of status data is measured. If the prolog and PDL for the units in each build are generated at the beginning of the build, for example, design status may be measured through the implementation phase at the same time that code status is being measured.

When a status measure reaches its target, measurement should stop on that activity. Rather than having data collectors make that decision and automatically stop monitoring on a given activity, the SEL relies on the project leader to indicate when a given activity is complete. This is done by marking the status data preprinted on the form with a slash through the values fields, as shown on the sample preprinted form (Figure 3-6). This will tell the data collectors not to enter the values in those fields and will stop those values from appearing on future preprinted forms distributed for updates.

DEVELOPMENT STATUS FORM

Name: _____

Project: _____ Date: _____

Please complete the section(s) that is appropriate for the current status of the project.

Design Status	
Planned total number of components to be designed (New, modified, and reused)	
Number of components designed (Prolog and PDL have been completed)	

Code Status	
Planned total number of components to be coded (New, modified, and reused)	
Number of components completed (Added to controlled library)	

Testing Status	System Test	Acceptance Test
Total number of separate tests planned		
Number of tests executed at least one time		
Number of tests passed		

Discrepancy Tracking Status (from beginning of system testing)	
Total number of discrepancies reported	
Total number of discrepancies resolved	

Specification Modification Status (from beginning of requirements analysis)	
Total number of specification modifications received	
Total number of specification modifications completed (implemented)	

Requirements Questions Status (from beginning of requirements analysis)	
Total number of questions submitted to analysts	
Total number of questions answered by analysts	

Check here if there are no changes ☐

For Librarian's Use Only
Number: _____
Date: _____
Entered by: _____
Checked by: _____

6201G(96) 8

NOVEMBER 1991

Figure 3-5. Development Status Form

DEVELOPMENT STATUS FORM

Name: PLEADER
Project: PROJECTX
Date: 19-JUL-91
Please complete the section(s) that is appropriate for the current status of the project.

Check here if there are no changes
☐

For Librarian's Use Only
Number:
Date:
Entered by:
Checked by:

*** This is the latest data as of 05-JUL-91

Design Status	
Planned total number of components to be designed (New, modified, and reused)	
Number of components designed (Prolog and PDL have been completed)	

Code Status	
Planned total number of components to be coded (New, modified, and reused)	395
Number of components completed (Added to controlled library)	395

Testing Status	**System**	**Acceptance**
Total number of separate tests planned	59	
Number of tests executed at least one time	6 ~~4~~	
Number of tests passed	3 ~~2~~	

Discrepancy Tracking Status (from beginning of system testing)	
Total number of discrepancies reported	24 ~~23~~
Total number of discrepancies resolved	19 ~~16~~

Specification Modification Status (throughout entire life cycle)	
Total number of specification modifications received	13
Total number of spec. mods. completed (implemented)	13

Questions to Analysts Status (throughout entire life cycle)	
Total number of questions submitted to analysts	67
Total number of questions answered by analysts	63 ~~52~~

Figure 3-6. Preprinted DSF for Update

Line-by-Line Instructions

Name: Enter the SEL database name of the project leader completing the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA. After the initial form, this name will be preprinted on the forms distributed for update. If the project leader changes, this name should be crossed out and the name of the new project leader written in.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored. After the initial form, this name will be preprinted on the forms distributed for update.

Date: Enter the Friday date corresponding to the week for which data are being reported. After the initial form, this date will be preprinted on the forms distributed for update. See hint 1 for a discussion of date tagging DSF data.

Design Status Section

Planned total number of components to be designed: Enter the total number of components to be included in the system. This number should include new, modified, and reused components to be developed on the project and maintained in the project's configured library. It does not include components that are reused by linking them to the system from a source external to the project. A component is a system element that is developed, maintained, and tracked as a separate configuration item (i.e., it is maintained as a distinct member of the project's configured library). This target number of units may fluctuate, but should reflect the entire system and not just the target number for the current build.

Number of components designed: Enter the number of components that have been designed. A component is considered designed when a prolog and PDL have been written, inspected, and certified. If the component does not contain executable code, the design may not include PDL. However, it is still considered complete when it has been inspected and certified.

Code Status Section

Planned total number of components to be coded: Enter the total number of components scheduled for implementation. Refer to the Design Status Section for the definition of a component and a discussion of what components are included in this target.

Number of components completed: Enter the number of components that have been coded, unit tested, certified, and added to the configured library. For cleanroom projects, components should be counted as complete when they have been certified and moved into the configured library for testing, since unit testing is not a part of that methodology.

Testing Status Section (System Test Column)

See hint 3 for a general discussion of test data tracking.

Total number of separate tests planned: Enter the number of tests to be conducted during system testing.

Number of tests executed at least one time: Enter the number of tests that have been executed for the first time, regardless of whether they passed, failed, or could not be evaluated.

Number of tests passed: Enter the number of tests that have been executed and evaluated as having passed successfully.

Testing Status Section (Acceptance Test Column)

See hint 3 for a general discussion of test data tracking.

Total number of separate tests planned: Enter the number of tests to be conducted during acceptance testing.

Number of tests executed at least one time: See instructions under Testing Status (System Test Column).

Number of tests passed: See instructions under Testing Status (System Test Column).

Discrepancy Tracking Status Section

Total number of discrepancies reported: Enter the cumulative number of software discrepancies reported since the start of system testing. A discrepancy is a reported occurrence of the software's performing incorrectly. Discrepancies are tracked internally on each task by such mechanisms as problem reports (PRs), software trouble reports (STRs), or SFRs. A discrepancy may or may not result in a change to the software, depending on its resolution.

Total number of discrepancies resolved: Enter the cumulative number of software discrepancies resolved since the start of system testing. A discrepancy is resolved when its cause has been isolated and, if necessary, corrected. Generally, a discrepancy is resolved when the PR, STR, or SFR on which it was reported has been closed out.

Specification Modification Status Section

Total number of specification modifications received: Enter the cumulative number of specification modifications that have been received from the analysts and approved by the ATR for implementation.

Total number of specification modifications completed: Enter the number of received specification modifications incorporated into the system.

Requirements Questions Status Section

Total number of questions submitted to analysts: Enter the cumulative number of requirements questions submitted to the analysts for clarification of requirements or specifications.

Total number of questions answered by analysts: Enter the cumulative number of submitted questions answered by the analysts.

No Changes Section

Check here if there are no changes: Check this box if the data supplied by the SEL on the current preprinted form have not changed.

Helpful Hints

1. DSFs are to be submitted every other Friday. SEL personnel distribute the preprinted forms with the most recently submitted data on the Wednesday preceding the Friday on which DSFs are due. The data entered on the form should reflect the most recent update the project leader has made to his/her internal records of project status. The Friday date on the form does not mean that a status measurement has to be taken on that date. For example, if the project leader routinely updates internal records on Monday mornings, the most recent Monday's update would be recorded on the DSF dated the following Friday. The project leader should not wait until the following Monday and submit the form late. The important thing is to be consistent so that the interval between reporting periods is uniform over the life cycle.

2. The preprinted DSF distributed by SEL personnel includes the date of the data that appear on the form (see Figure 3-6). In most cases, this will be the date of the Friday 2 weeks prior, when DSFs were last submitted. If, however, a form was submitted late, it will probably not have been processed by the time the preprinted forms for update are generated. In this case, the data on the form will be 4 weeks old and the data date on the form will so indicate. Thus, it is important to take note of this date. A common cause of errors is the project leader's thinking that no changes have occurred in a given measure over the preceding 2 weeks, but not realizing that the numbers printed on the form represent data that are 4 weeks old.

3. Test data should be measured at the lowest level of detail tracked. Test plans usually contain a series of individual tests, each of which may involve multiple runs. Each of these runs, in turn, may have multiple items to be evaluated. For best visibility into testing progress, the SEL recommends that testing be tracked on the test-item level, which, in the FDD environment, is generally the case for acceptance testing. If the system test plan does not call out individual items to be evaluated, testing should be tracked to the level of

individual tests or test runs. Tracking a small number of high-level tests or test series provides little visibility into the progress of testing.

4. In the FDD environment, more than one separately monitored development project may be generated from the same set of requirements and specifications. This most commonly occurs with the AGSS and telemetry simulator for a given spacecraft. In these cases, requirements questions and specification modifications may not be tracked separately for the two projects, since they are written against the same requirements and specifications documents. When this happens, the SEL encourages project leaders to identify the questions or specification modifications as to which systems they affect, so that they may be tracked separately for DSF data collection. If this is not possible, these data should be recorded on DSFs for one of the projects and not for the other. This should be discussed at project startup, and a general message should be entered to indicate that this combined tracking was performed for the two projects in question.

3.2.1.3 SERVICES/PRODUCTS FORM (SPF)

General Information

The SPF (Figure 3-7) is completed by SEL data collection personnel to capture the three types of weekly rate data that the SEL monitors automatically: computer resources, growth history, and services effort. Although development personnel are never required to complete or submit this form, it is crucial that project leaders understand the data recorded on it and their role in facilitating the collection of those data.

Computer resources data are collected and recorded by the SEL weekly. On most computers used by monitored projects, the SEL has access to accounting software that logs the number of runs and the CPU hours used. The SEL defines a run to be a logon session or a submitted batch job. On the FDF mainframe computers, the SEL tracks batch jobs and interactive sessions separately. On the STL VAX computers, interactive sessions and batch job submittals are combined to give a total number of runs.

Because projects often perform development activities on more than one computer, the SEL collects CPU hours that have been normalized to the relative speed of a given machine established to be representative of a particular class of machines. For example, the STL VAX environment is a cluster of different members of DEC's VAX family of computers, including a VAX 11/780. Since projects developed in this environment use more than one machine in the cluster, and since the different machines run at different speeds, the accounting data for CPU hours are normalized to report all hours in terms of VAX 11/780 equivalent hours. Similarly, in the FDF mainframe environment, CPU hours are normalized to NAS 8040 equivalent hours.

The SEL does not record computer resources data for all projects. If a substantial portion of the development work is performed on PCs or workstations to which the SEL

SERVICES/PRODUCTS FORM

Project:_____

Date (Friday):_____

COMPUTER RESOURCES

Computer	CPU Hours	No. of Runs

GROWTH HISTORY

Components	
Changes	
Lines of Code	

SERVICES EFFORT

Service	Hours
Tech Pubs	
Secretary	
Proj Mgmt	
Other	

For Librarian's Use Only

Number:_____

Date:_____

Entered by:_____

Checked by:_____

6201G(13)-08

NOVEMBER 1991

Figure 3-7. Services/Products Form

does not have access or for which accounting software is not available, computer resources data will not be collected. This should be discussed at the project startup meeting.

Growth history data are the second type of data collected on the SPF. Each week from the time the project establishes a library for placing developed code under configuration control, the SEL measures the number of components in the library, the number of SLOC in the library, and the number of changes that have been made to components since they were first entered into the library.

To collect the growth history data, the SEL maintains library monitoring tools in both the FDF and STL computing environments. The FDF tool computes statistics from one or more PANVALET libraries. The STL tool computes statistics from one or more DEC Code Management System (CMS) libraries. It can also monitor a subset of a library identified as belonging to a CMS "group." These tools count the number of library members to obtain a component count; they sum the number of records in all of the members to obtain a SLOC count; and they compute changes by summing the version (level, generation) numbers of each library member and subtracting from this sum the total number of library members. This gives the number of changes made to components after they initially were moved into the library at version (level, generation) 1. As with computer resources data, growth history data are not monitored for projects to whose libraries the SEL does not have access.

Services effort is the effort expended by all personnel who provide support services to a given project but do not submit their hours to the SEL on a PRF. Services effort hours are extracted from timecard accounting systems, where these records are available to the SEL from the organizations being monitored. On projects where such records are not available, services effort is not recorded. This should be established at the project startup meeting.

Services effort falls into four categories. Tech Pubs support includes hours spent by publications personnel involved in the production of project documentation. This includes editors, word processors, proofreaders, graphics professionals, and reproduction personnel. Secretary support includes hours spent by secretaries providing direct support services to the project. Proj Mgmt support includes all hours charged to the project by management personnel at levels above the first-line manager (who reports his/her management hours on PRFs). Other support includes hours charged to the project that do not fall into any of the other three support categories. This usually includes project control personnel, indirect secretarial support, and facilities personnel.

The project leader's participation is essential in the collection of all three of the data types collected on the SPF. In collecting computer resources data, the project leader must keep SEL data collectors informed as to what computers are being used and what accounts should be monitored. Similarly, he/she must tell the SEL what library or libraries need to be monitored to measure growth history data. If a partial CMS library

on the VAX is to be monitored, the CMS group must also be specified. The project leader must also ensure that the SEL has access to the libraries in read-only mode.

For services effort data, the project leader must provide the accounting cost collection numbers to be monitored. Usually, a SEL-monitored project corresponds to a single task number in the accounting system. There are, however, cases where part of a project (several subsystems, perhaps) is developed under a separate task number. In these cases, the accounting data from the two tasks must be combined to reflect the total services effort data for the project. The opposite case also occurs; that is, more than one separately monitored SEL project is developed under the same task number in the accounting system. When this happens, the project leader must meet with the SEL DBA to establish a proration algorithm for splitting the services effort hours among the projects.

The project leader must also specify the names of management personnel who report their management hours on PRFs, so that these hours are not double counted as Proj Mgmt hours.

In addition, secretaries who provide direct support to the project must be identified so that their hours, which should be recorded as Secretary support, may be distinguished from those of other secretaries providing indirect support to the project (whose hours should be recorded as Other support).

To help project leaders keep track of information being monitored automatically on their tasks, the SEL distributes a monthly report, called the Data Collection Status Report (Figure 3-8). Each project leader receives a page of the report for each project for which he/she is responsible. This report lists the computers and accounts being monitored for computer resources data, the libraries being monitored for growth history data, and the task numbers being monitored for services effort data. In addition, it lists the types of forms currently being submitted on the project and the programmers from whom the SEL expects to receive effort forms (PRFs, CLPRFs, or WMEFs) on a weekly basis.

Each month, it is the project leader's responsibility to review the data collection information for his/her projects and to return the report to the SEL DBA with corrections or an indication that the information is correct.

Line-by-Line Instructions

NOTE: These instructions are intended for SEL data collection personnel.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored.

Date (Friday): Enter the Friday date corresponding to the end of the week for which data are being reported. Data are to be reported on this form for all work performed on the project during the preceding Saturday-through-Friday period. Thus, the growth

November 1991

Please update any information that has changed. Place in the SEL forms bin or return to the SEL DBA, GreenTec II, Room (#) by **COB** (date). If you have any questions, contact me at 301-794-####.

Project: **PROJECTX**

Check here if there are no changes _____

Forms Being Collected: PRF, COF, CRF, DSF, PEF

Computer: FDF mainframes

Account(s) Being Monitored: FBACC

Configured Library(s) Being Monitored:

FBPRO.PROJX.PAN
FBPRO.INC.PAN
FBPRO.PROJX.PANELS

Personnel Submitting Effort Forms:

Programmer 1 CSC
Programmer 2 CSC
Programmer 3 CSC

Service Resources Being Monitored: 99-111, 99-222

Figure 3-8. Sample Page From Data Collection Status Report

history monitoring tools should be run on Fridays. Computer accounting reports that SEL personnel are not responsible for generating may cover periods other than Saturday through Friday. If this is the case, the first report run **following** the Friday date on the form should be used.

Computer Resources

Computer: Enter the name of the computer (as it appears in coded form in the COMPUTER table of the database) for which hours and runs are being recorded.

CPU hours: Enter the CPU hours used during the week (for the above computer) in decimal form to the nearest tenth of an hour.

No. of runs: Enter the number of runs (an integer) executed during the week for the above computer.

Growth History

Components: Enter the number of components in the system as reported by the SEL growth history monitoring software for the week being recorded.

Changes: Enter the cumulative number of changes to the system as reported by the SEL growth history monitoring software for the week being recorded.

Lines of code: Enter the number of lines of code in the system as reported by the SEL growth history monitoring software for the week being recorded.

Services Effort

Tech pubs: Enter the number of hours charged to the project by technical publications personnel during the week being recorded.

Secretary: Enter the number of hours charged to the project by secretarial personnel directly supporting the project during the week being recorded.

Proj mgmt: Enter the number of hours charged to the project by project management personnel during the week being recorded. Do not include hours for managers whose hours have already been reported on a PRF for the week in question.

Other: Enter the number of hours charged to the project by all other support personnel not included in the three previous categories during the week being recorded.

Helpful Hint

To obtain an accurate picture of system growth, the SEL requests that the following guidelines be adhered to when performing configuration management of source libraries.

- Move new components into the library at version or level 1.
- Move new components into the library when they have been coded and tested (and a COF has been completed), not when they have merely been designed.

In other words, do not use the configured source library for storing PDL or prologs. Instead, the use of a separate design library is recommended.

- Each time a component is updated in the library, increase the version number by one.
- Do not reset level numbers.
- Do not delete history records produced by the configuration management tool. This information is very important when the project is being closed out.
- Do not maintain more than one copy of a given component even if there is more than one configured library.

The last item is particularly important. Not only does it make good sense from a configuration management point of view to maintain a component in only one place, but if the same component appears in more than one library, it will be counted twice when monitoring growth history. In addition, if the version numbers are different in the two libraries, the SEL has no way of knowing which to use in counting changes. To prevent problems in collecting growth data, the project leader must regularly communicate changes in the names of configured source libraries that the SEL should be monitoring.

A common problem in counting changes arises when updated modules are copied from the developer's work library into the configured library at the version number of the component in the work library. If the developer updated the component several times in the work library before determining that the change had been correctly implemented, the new version number would show that more than one change had been made, which is clearly not the case. One way of avoiding the problem is to use a configuration management tool, such as the CMS on the VAX. If using PANVALET libraries on the IBM, the PANVALET Move/Copy function, available under the Software Development Environment (SDE) (Reference 11), will maintain level numbers correctly.

To provide a cross-check between developer-submitted data and SEL-monitored data, the SEL produces and distributes to project leaders monthly graphs that compare the number of COFs submitted and changed components appearing on CRFs with the number of components and changed components, respectively, measured by the SEL and recorded on SPFs.

A final note concerns implementation in builds where different configured libraries are used in different builds. When making the transition from one build to the next, the new configured library should initially contain components at the versions at which they existed in the old configured library. When this occurs, the SEL must be notified of the new library name and location. It will then be assumed that no more updates will be made to the old configured library, and the SEL will not continue to monitor it.

3.2.2 Event Data

In contrast to rate data, which are collected with a predetermined periodic frequency, event data are submitted to the SEL sporadically, when given events in the software development process occur. Referring to Table 2-8, event-driven forms collected during development include the CCF, COF, CRF, PMF, and SIF. The SIF, COF, and CCF are used to capture data on system elements. The CRF is used to capture change and error data that characterize modifications made to the software products after they are initially placed under configuration control. The PMF is used to record messages, which may be submitted any time during the life cycle to capture auxiliary information about a project. These five forms are discussed in the following sections.

3.2.2.1 SUBSYSTEM INFORMATION FORM (SIF)

General Information

The event that drives the completion of the initial SIF (Figure 3-9) is the Preliminary Design Review (PDR). According to the SEL methodology, that is when a high-level partitioning of the system into subsystems should have been accomplished. As the system is further decomposed into its lowest level elements, or components, in detailed design, it is essential to have a naming convention in place for referring to the components. This naming convention should associate each component with a subsystem in the design of the system. It is this aspect of the component-naming convention that the SIF is intended to record.

As mentioned in the discussion of the PEF (Section 3.1.2), the term "subsystem" has a slightly different interpretation in the context of the subsystem prefixes entered on the SIF. Rather than referring strictly to the logical partitioning of the system present in the high-level design, on the SIF a subsystem refers to a mutually exclusive partitioning of the low-level components that make up the system. This allows each component of the system to be a member of exactly one subsystem. The subsystem prefix is then used when completing COFs (Section 3.2.2.2) to establish that membership relationship.

The distinction is subtle. In general, every logical subsystem present in the system design should appear on the SIF. In addition, however, there may be classes of components that are used in more than one logical subsystem and should be assigned distinct subsystem prefixes. FORTRAN COMMON blocks, for example, are usually maintained as separate files in the FDD environment and are "included" into the appropriate routines at compile time. The SEL recommends that these components be grouped under a common subsystem prefix, such as CM. Even if a COMMON block is referenced exclusively by routines belonging to a single logical subsystem, it should be associated with the CM prefix. This simplifies the compilation of component- and system-level size statistics at the end of the project. Other classes of FORTRAN components that are usually grouped together are NAMELIST components, BLOCK DATA components, and commonly referenced utility routines.

The examples noted in the preceding paragraph do not apply to Ada projects. The object-oriented design approach used on Ada projects in the FDD environment does,

SUBSYSTEM INFORMATION FORM

Name:_____

Project: _____ Date: _____

Add New Subsystems

Subsystem Prefix	Subsystem Name	Subsystem Function

Change Existing Subsystems

Old Subsystem Prefix (Must exist in the database)	Action (R - Rename, D - Delete)	New Subsystem Prefix (Must not exist in the database)

This form is to be completed by the time of the Preliminary Design Review (PDR). An update must be submitted each time a new subsystem is defined thereafter. This form is also to be used when a subsystem is renamed or deleted.

Subsystem Prefix: A prefix of 2 to 5 characters used to identify the subsystem when naming components

Subsystem Name: A descriptive name of up to 40 characters

Subsystem Function: Enter the most appropriate function code from the list of functions below:

USERINT:	User Interface
DPDC:	Data Processing/Data Conversion
REALTIME:	Real-time Control
MATHCOMP:	Mathematical/Computational
GRAPH:	Graphics and Special Device Support
CPEXEC:	Control Processing/Executive
SYSSERV:	System Services

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

NOVEMBER 1991

6201G(13)-39

Figure 3-9. Subsystem Information Form

however, allow logical subsystems to be nested within other logical subsystems. Obviously, this type of situation would not provide a mutually exclusive partitioning of system components. The rule of thumb is that, for SEL purposes, an Ada component belongs to the lowest level subsystem of which it is a member.

It is important to note that, in addition to being the prefixes that will be used on COFs to establish subsystem membership, the prefixes entered on the SIF should be used in the names of components (files) in the project's configured library. At the end of development, SEL personnel must reconcile the components in the configured library with the names that have been submitted on COFs so that component-level statistics may be recorded. In addition, following this guideline will help developers to complete COFs correctly, since they will not have to remember different naming conventions for file names and SEL forms.

Completion of the SIF is the responsibility of the project leader. An initial SIF is expected at the time of PDR. Additional SIFs may be submitted any time thereafter when subsystems are added to the design. They may also be submitted to delete a subsystem that is no longer a part of the design or to rename a subsystem.

Line-by-Line Instructions

Name: Enter the SEL database name of the project leader completing the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored.

Date: Enter the date on which the form is completed.

Add New Subsystems

Subsystem prefix: Enter the 2- to 5-character subsystem prefix used to identify the subsystem in the SEL database. When naming system components, this prefix will become the first characters of the component name to show the subsystem to which the component belongs. When the file-naming convention involves separating the prefix and component name with an underscore, the underscore is not considered part of the prefix. Do not include the underscore when entering the prefix on SEL forms.

Subsystem name: Enter a descriptive name of up to 40 characters that specifies what role the subsystem plays in the overall system design, e.g., data adjuster, telemetry processor, truth model, NAMELIST components.

Subsystem function: Select the **one** subsystem function from the list provided on the form that best describes the type of processing performed by the subsystem. Sometimes a subsystem may provide more than one of the functions listed. In such cases, the predominant function of the subsystem should be chosen. As an example, a user interface

subsystem that interfaces with system services to put menus on a screen would have a function of USERINT rather than SYSSERV. In other situations, two functions may apply in which one function is more specific than the other. In this case, the more specific function should be chosen. For example, if a subsystem is implementing mathematical algorithms to provide real-time control functions, REALTIME should be chosen over MATHCOMP. See hint 3 for additional pointers.

Change Existing Subsystems

Old subsystem prefix: Enter the prefix of the subsystem to be renamed or deleted. This prefix must already exist in the database (i.e., it must previously have been submitted on an SIF).

Action: Enter "R" to rename the subsystem prefix or "D" to delete it from the database. (See hint 1 for restrictions on renaming and deleting subsystem prefixes.)

New subsystem prefix: Enter the new name of the prefix being renamed. This new prefix must not already exist in the database.

Helpful Hints

1. When deleting or renaming subsystems, be sure to consider what will happen to any components that may belong to the existing subsystem. The rename option cannot be used to move all of the components under one subsystem to another subsystem that already exists. This must be accomplished by renaming the individual components using the CCF (Section 3.2.2.3). Additionally, a subsystem cannot be deleted if there are components in the database that belong to it. Those components must be either deleted or renamed (via CCF) before the old subsystem can be deleted.

2. If individual components will be reused by linking them to the system in object form from RSL, MTASS, or MSASS, prefixes for these reuse sources must be provided. The RL prefix has been reserved to identify components reused from the RSL. The prefixes MTASS and MSASS are reserved for reuse from those two sources. COFs will be completed for individual components linked in from any of these sources. An MTASS or MSASS prefix need not be provided, however, if only complete subsystems are being reused, rather than individual components. This type of reuse does not require the completion of COFs. It should, however, be noted at the project startup meeting and confirmed with SEL personnel at project completion so that size statistics from those subsystems may be included in the final project size computations.

3. Questions sometimes arise about what subsystem function should be listed for a prefix that does not correspond to a functional subsystem in the high-level system design. The following guidelines for the examples of this

type of subsystem discussed under **General Information** will apply in most cases:

FORTRAN COMMON block components:	DPDC
FORTRAN NAMELIST components:	USERINT
FORTRAN BLOCK DATA components:	DPDC

3.2.2.2 COMPONENT ORIGINATION FORM (COF)

General Information

The COF (Figure 3-10) is used to record information that characterizes each component in the system at the time it initially becomes part of the system. It is completed by the developer responsible for coding and unit testing the component. The developer passes it on to the project leader or configuration manager so that configuration information may be recorded.

From a SEL data collection point of view, a software system at the lowest level is composed of elementary pieces called components. A component, as viewed by the SEL, is **any piece of the system that is maintained in a separate file.** Thus, a component does not necessarily have to correspond to an executable module in the system's implementation language. For example, a FORTRAN COMMON block is considered a component if it is maintained in its own file for the purpose of "including" it in other components at compile time. Also, if a single file contains more than one subroutine, procedure, or function, as in the case of nested procedures in Pascal and Ada, or multiple entry points in a FORTRAN subroutine, for example, the file itself is considered a component rather than each of the nested subroutines or procedures. This definition of a component is to be used when completing COFs.

The event that drives the completion of a COF is the origination of a component. A component is "originated" when it has been unit tested and is ready to be moved into the project's configured source library. A COF may be completed earlier than the implementation phase when the code is produced as part of the design effort (e.g., an Ada package specification). However, it should not be completed to record the origination of a component design (prolog, PDL) that is configured into a design library. The COF is designed to be used by the project configuration manager as well as by the SEL. When the component is physically transferred from the developer's library into the configured library, the configuration manager adds the configuration date to the form. After that, the component is considered to be under configuration control.

The conventions for identifying individual components in a system, as well as the subsystem to which they belong, should have been discussed in the project startup meeting. Before any COFs may be submitted, an SIF must be submitted to identify prefixes used in the component-naming convention. A SEL component name consists of two parts: a prefix of 2 to 5 characters that uniquely identifies the subsystem to which a component belongs, and a name of up to 40 characters that identifies the component within the

COMPONENT ORIGINATION FORM

Identification

Name:_____

Project: _____ Date:_____

Subsystem Prefix: _____

Component Name:_____

Configuration Management Information

Date entered into controlled library (supplied by configuration manager): _____

Library or directory containing developer's source file:_____

Member name:_____

Relative Difficulty of Developing Component

Please indicate your judgment by circling one of the numbers below.

Easy		Medium		Hard
1	2	3	4	5

Origin

If the component was modified or derived from a different project, please indicate the approximate amount of change and from where it was acquired; if it was coded new (from detailed design) indicate NEW.

_____ NEW
_____ Extensively modified (more than 25% of statements changed)
_____ Slightly modified
_____ Old (unchanged)

For Librarian's Use Only

Number: _____
Date:_____
Entered by:_____
Checked by:_____

If not new, what project or library is it from?_____
Component or member name: _____

Type of Component (Check one only)

_____ INCLUDE file (e.g., COMMON)
_____ Control language (e.g., JCL, DCL, CLIST)
_____ ALC (assembler code)
_____ FORTRAN source
_____ Pascal source
_____ C source
_____ NAMELIST or parameter list
_____ Display identification (e.g., GESS, FDAF)
_____ Menu definition or help
_____ Reference data files
_____ BLOCK DATA file
_____ Ada subprogram specification
_____ Ada subprogram body
_____ Ada package specification
_____ Ada package body
_____ Ada task body
_____ Ada generic instantiation
_____ Ada generic specification
_____ Ada generic body
_____ Other

Purpose of Executable Component

For executable code, please identify the major purpose or purposes of this component. (Check all that apply).

_____ I/O processing
_____ Algorithmic/computational
_____ Data transfer
_____ Logic/decision
_____ Control module
_____ Interface to operating system
_____ Process abstraction
_____ Data abstraction

6201G(13).38

NOVEMBER 1991

Figure 3-10. Component Origination Form

subsystem. The combination of subsystem prefix and component name must uniquely identify each component within the system. Thus, a system may have two components with the same 40-character name as long as they belong to different subsystems and have different subsystem prefixes. Ideally, the names chosen for the physical implementation of components (i.e., file names, library members) should be identical to the SEL component names (subsystem prefix concatenated with a component name). As discussed in the SIF instructions, at project completion the SEL reconciles the component names entered into the database (via COFs) with the names of components that appear in the project's configured source library. If these names are not the same, the project leader **must** provide a key showing the translation of SEL names to physical implementation names.

Line-by-Line Instructions

Identification

Name: Enter the SEL database name of the programmer completing the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored. Check with the project leader if unsure of the correct name.

Date: Enter the date on which the form is completed.

Subsystem prefix: Enter the prefix identifying the subsystem to which the component belongs. The prefix entered must have previously been entered into the database by means of an SIF.

Component name: Enter the name of the component **without** preceding it with its subsystem prefix. The component name may be up to 40 characters. The combination of subsystem prefix and component name must uniquely identify a single component in the system.

Configuration Management Information

Date entered into configured library: Enter the date on which the component is physically transferred from the developer's library to the project's configured source library. This date is usually entered by the configuration manager.

Library or directory containing developer's source files: Enter the name of the library (e.g., PANVALET library on FDF mainframes, CMS library on the STL VAX cluster) or directory (e.g., VAX/VMS directory) in which the source code for the unit-tested module resides. This field is to be completed by the developer for use by the project configuration manager.

Member name: Enter the name of the component as it appears in the above library or directory (e.g., PANVALET member name, CMS element name, VMS file name).

This field is also to be completed by the developer for use by the project configuration manager.

Relative Difficulty of Developing Component

Enter a subjective judgment of how difficult the component was **to develop**. This is not necessarily the same as the complexity of the component. For example, when modifying a reused component to meet system requirements, a component that performs a relatively simple function may be difficult to modify if it is not clearly written and well documented. The converse may also occur. A reused component may perform a very complicated function yet may be easy to modify if it is well written. For a verbatim reused component, the difficulty lies in understanding the component well enough to incorporate it into the system.

Origin

Check one of the listed options to describe the source of the component. If it was newly developed from the detailed design, check "new." If it was reused from another source, consider the amount of modification that was needed to make it meet system requirements and check one of the last three options. If the component is not new, be sure to complete the following two fields.

If not new, what project or library is it from?: Enter the name of either a SEL database project or a specific library from which the reused component was taken.

Component or member name: If a SEL database project was listed in response to the preceding question, enter the SEL component name (including the subsystem prefix) of that component in the source SEL project. If a library was listed, enter the member name of the component in the library.

Type of Component

Check the **one** type from the options listed that best describes the component. Follow the general rule of choosing the most specific type category that applies. For example, although a FORTRAN COMMON block is FORTRAN source code, "INCLUDE file" should be checked rather than "FORTRAN source." Control language components (JCL, DCL, CLISTs, etc.) that build the system (compile from source, link, etc.) are usually not stored in the project's configured source library, and COFs for these types of procedures are not submitted. If, however, a control language module is employed when the system is executing, that module is considered a configured part of the system, and a COF should be submitted for it. The "display identification" category is used for components written in a display language, such as the Graphics Executive Support System (GESS) (Reference 12) or the Flight Dynamics Application Framework (FDAF) (Reference 13), both of which are institutional software packages used in the FDF environment. Components written in vendor-supplied display languages, such as Interactive System Productivity Facility (ISPF) (Reference 14) panels also fall under this

category. The "menu definition or help files" category is used for ordinary data files that are used by the application software to define displays or provide online help. The "reference data files" are files used by the system that provide functionality and are not input files that will be varied from run to run when the system is used operationally. An example would be a file of information used by a simulator to translate and process encoded ground commands. Such a file would be considered part of the configured system (it could have been implemented in source code as an internal table) and a COF identifying it as a "reference data file" should be submitted. The remaining categories require no further clarification.

Purpose of Executable Component

Complete this portion of the form **only** if the type of component checked in the previous item indicates that the component contains executable code or if the component is written in Ada. This section should not be completed, for example, if the component is a COMMON, a NAMELIST, a BLOCK DATA subprogram, or a reference data file. If the component is executable, check **all** of the purposes that describe the functions performed by the component. Descriptions of the purposes follow.

- *I/O processing*: A major function of the component is to read from or write to disk files, tapes, display screens, or other peripheral devices. Examples include components that access an attitude history file and GESS or FDAF display screens.
- *Algorithmic/computational*: A major function of the component is to perform computations that implement a mathematical algorithm specified in the system requirements or functional specifications. Examples include components that model a spacecraft sensor or propagate spacecraft attitude.
- *Data transfer*: A major function of the component is to manipulate data, transferring them to and from internal data structures. Examples include components that pack or unpack telemetry records or transfer data from one data structure to another.
- *Logic/decision*: A major function of the component is to make decisions that affect the paths that are executed in the system. Examples include components that route messages to various destinations based on evaluating an address or that determine what type of orbit propagator to invoke, based on a user-supplied flag.
- *Control module*: A major function of the component is to control the overall process flow of the system or of a subsystem. Examples include driver components that control the order in which major functions are invoked or components that schedule the execution of discrete events in a spacecraft simulator.
- *Interface to operating system*: A major function of the component is to provide access to functionality supplied by the host operating system. Examples

include components that provide access to VAX/VMS system services, MVS direct access I/O functions, or the system clock.

- *Process abstraction*: A major function of the component is to provide a template for a given process, the detailed processing steps of which must be supplied for it to do useful work. The most common examples of process abstractions are implemented via Ada generics. A process abstraction for modeling ephemeris, for example, would be a template for computing ephemeris in which the object for which ephemerides are being computed and the method for computing them are supplied as parameters. An Ada generic that provides a template for a generalized process would be considered a "process abstraction." The instantiation of the generic to implement a specific process, however, would not be.

- *Data abstraction*: A major function of the component is to encapsulate a composite data type and the operations that may be performed on it. A typical example would be an Ada package used to define a data structure, such as a stack, and all the operations used to access it. In this case, the package specification and body would be considered to have a purpose of "data abstraction," but individual routines separate from the body would not.

Helpful Hints

1. One of the most common mistakes made in completing COFs arises from not understanding the distinction between the subsystem prefix and the component name. Because the name of the library member is usually a concatenation of these two identifiers and because the library name is often referred to as the component name by developers, there is a tendency to duplicate the prefix when writing the component name on the COF. The following examples should clarify this.

Example 1

Library name:	UTMATMPY
COF subsystem prefix:	UT
COF component name:	MATMPY (**not** UTMATMPY)

Example 2

Library name:	SHEM_SPACECRAFT_BODY.ADA
COF subsystem prefix:	SHEM
COF component name:	SPACECRAFT_BODY

The first example is a typical PANVALET library member name, where the prefix is simply the first two characters of the member name. The second example is a typical VAX CMS library element name. Note that, since longer names are permitted, the convention is to separate the prefix from the

component name with an underscore. The underscore, however, is not entered on the COF, either as part of the prefix or as part of the name. Note also that the file extension (.ADA) included in the library element name is not included when writing the component name on the COF.

2. The one exception to hint 1 comes when reusing individual components from either MTASS or MSASS. In this case, the reserved prefix MTASS or MSASS is entered as the subsystem prefix, and the **entire** library member name (including what would normally be considered the subsystem prefix) is entered in the component name field on the COF.

3.2.2.3 COMPONENT CHANGE FORM (CCF)

General Information

Occasionally a component is no longer needed and is deleted from a project's configured library. If this happens, the project leader submits a CCF (Figure 3-11) to have the component deleted from the database. A CCF can also be used to correct component names that have been entered into the database incorrectly. Similarly, when components in the project's configured library are renamed, the CCF is used to rename the components in the database.

Line-by-Line Instructions

Name: Enter the SEL database name of the project leader submitting the form.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored.

Date: Enter the date on which the form is submitted.

Old component: Enter the name of the component to be changed as it currently exists in the database. Include both the subsystem prefix and the 40-character component name. When the file-naming convention involves separating the prefix and component name with an underscore, the underscore is not considered part of the prefix. Do not include the underscore when writing the prefix on the SEL form.

Action: Enter "R" to rename the component or "D" to delete it from the database.

New component: Enter the new name of the renamed component. Include both the subsystem prefix and the 40-character component name. When the file-naming convention involves separating the prefix and component name with an underscore, the underscore is not considered part of the prefix. Do not include the underscore when writing the prefix on the SEL form. A name should be entered only if the action selected is "R."

COMPONENT CHANGE FORM

Name:_____

Project:_____ Date:_____

This form is to be used when components in the project controlled source library are deleted or renamed.

Old Component (Must exist in the database)		Action R – Rename D – Delete	New Component	
Prefix	Name		Prefix	Name

6201G(13) 37

NOVEMBER 1991

Figure 3-11. Component Change Form

Helpful Hints

1. In contrast to renaming a subsystem via an SIF, where the new subsystem name entered on the form must not already exist in the database, a component **may** be renamed to the name of an existing database component. This may be useful when two components are combined into a single component that retains the name of one of the original components. The COF data associated with the "new" component name will be retained, and the COF data associated with the component being renamed will be deleted. Nearly the same effect may be achieved by simply deleting the old component that is being subsumed into the new one. This is not recommended, however, since any changes made to the old component and reported on CRFs (Section 3.2.2.4) would no longer be associated with any valid component in the database, when the code that was changed now belongs to the new component and changes made to that code should reference that component.

2. The rename option of the CCF can be used when components are moved from one subsystem to another, simply by changing only the prefix when entering the new component name. The new subsystem prefix, however, must already exist in the database. The CCF cannot be used to create a new prefix.

3.2.2.4 CHANGE REPORT FORM (CRF)

General Information

Once system components have been placed under configuration control, i.e., moved into the project's configured source library after having been successfully unit tested and identified to the SEL on a COF, the SEL collects information on all subsequent changes to the system that cause those components to be modified and replaced in the configured source library. Each such change is documented by submitting a CRF (Figure 3-12).

The event triggering the submission of a CRF is the implementation of a **logical** change to the system. A logical change may be made for any number of reasons, ranging from requirements and specifications changes to error corrections. The implementation of a logical change may require any number of individual components to be modified, yet that single logical change is documented on a single CRF. Thus, the number of CRFs submitted will not correspond one-to-one with the number of changes made to the project's configured source library.

The key to understanding a logical change is that it has a single, well-defined purpose. For example, if the implementation of a requirements change requires 10 components to be modified, the changes to those 10 components constitute a single logical change and a single CRF is completed. If, however, while changing those 10 components, a developer makes additional changes in one of them to correct an outstanding error that

CHANGE REPORT FORM

Name:_____ Approved by:_____

Project:_____ Date:_____

Section A – Identification

Describe the change: (What, why, how) _____

Effect: What components are changed?

Prefix	Name	Version

(Attach list if more space is needed)

Effort: What additional components were examined in determining what change was needed?

Location of developer's source files_____

	month	day	year
Need for change determined on:			
Change completed (incorporated into system):			

☐ *Check here if change involves Ada components (If so, complete questions on reverse side)*

	1 hr/less	1 hr/1 day	1/3 days	>3 days
Effort in person time to isolate the change (or error):				
Effort in person time to implement the change (or correction):				

Section B – All Changes

Type of Change (Check one)

- ☐ Error correction
- ☐ Planned enhancement
- ☐ Implementation of requirements change
- ☐ Improvement of clarity, maintainability, or documentation
- ☐ Improvement of user services
- ☐ Insertion/deletion of debug code
- ☐ Optimization of time/space/ accuracy
- ☐ Adaptation to environment change
- ☐ Other (Describe below)

Effects of Change

Y	N	
☐	☐	Was the change or correction to one and only one component? (Must match Effect in Section A)
☐	☐	Did you look at any other component? (Must match Effort in Section A)
☐	☐	Did you have to be aware of parameters passed explicitly or implicitly (e.g., COMMON blocks) to or from the changed components?

Section C – For Error Corrections Only

Source of Error (Check one)

- ☐ Requirements
- ☐ Functional specifications
- ☐ Design
- ☐ Code
- ☐ Previous change

Class of Error (Check most applicable)*

- ☐ Initialization
- ☐ Logic/control structure (e.g., flow of control incorrect)
- ☐ Interface (internal) (module-to-module communication)
- ☐ Interface (external) (module to external communication)
- ☐ Data (value or structure) (e.g., wrong variable used)
- ☐ Computational (e.g., error in math expression)

*If two are equally applicable, check the one higher on the list.

Characteristics (Check Y or N for all)

Y	N	
☐	☐	Omission error (e.g., something was left out)
☐	☐	Commission error (e.g., something incorrect was included)
☐	☐	Error was created by transcription (clerical)

For Librarian's Use Only

Number:_____
Date:_____
Entered by:_____
Checked by:_____

6201G(13) 09

NOVEMBER 1991

Figure 3-12. Change Report Form (1 of 2)

CHANGE REPORT FORM

Ada Project Additional Information

1. Check which Ada feature(s) was involved in this change (Check all that apply)

- ☐ Data typing
- ☐ Subprograms
- ☐ Exceptions
- ☐ Generics
- ☐ Program structure and packaging
- ☐ Tasking
- ☐ System-dependent features
- ☐ Other, please specify _____ (e.g., I/O, Ada statements)

2. For an error involving Ada components:

a. Does the compiler documentation or the language reference manual explain the feature clearly? _____ (Y/N)

b. Which of the following is most true? (Check one)

- ☐ Understood features separately but not interaction
- ☐ Understood features, but did not apply correctly
- ☐ Did not understand features fully
- ☐ Confused feature with feature in another language

c. Which of the following resources provided the information needed to correct the error? (Check all that apply)

- ☐ Class notes
- ☐ Ada reference manual
- ☐ Own project team member
- ☐ Own memory
- ☐ Someone not on team
- ☐ Other

d. Which tools, if any, aided in the detection or correction of this error? (Check all that apply)

- ☐ Compiler
- ☐ Symbolic debugger
- ☐ Language-sensitive editor
- ☐ CMS
- ☐ Source Code Analyzer
- ☐ P&CA (Performance and Coverage Analyzer)
- ☐ DEC test manager
- ☐ Other, specify _____

3. Provide any other information about the interaction of Ada and this change that you feel might aid in evaluating the change and using Ada

6201G(13)-13

NOVEMBER 1991

Figure 3-12. Change Report Form (2 of 2)

had been discovered earlier during testing, that error correction constitutes a second logical change. A second CRF must be completed for that change, even though the individual component involved may have been removed from the library, modified, and replaced only one time.

The CRF is completed by the developer implementing the change once all affected components have been modified and retested. The form is then intended to be used as a configuration management tool for identifying to the project configuration manager the components that were updated, their version numbers, and their location. The configuration manager then makes the necessary library updates and notes on the form the date that the change is configured into the system. The form also contains a field for approval of the change. This is generally used by someone who reviews both the technical correctness of the change and the correctness of the information supplied on the form. This approver is often the project leader.

Line-by-Line Instructions

Name: Enter the SEL database name of the developer completing the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored. Check with the project leader if unsure of the correct name.

Approved by: This field is provided for use by a reviewer who has verified the technical correctness of the change and the accuracy of the information supplied on the form. This reviewer is often the project leader, but may be a team leader, a peer developer, or the configuration manager.

Date: Enter the date on which the form is completed.

Section A — Identification

Describe the change: Explain what change is being made, why the change is necessary, and how the change is being made. Provide sufficient detail so that people other than project staff can understand the change. The description should not be on the variable name level, but it should be sufficiently abstract so that the function of the changed code can be determined. For example, use "the input buffer was cleared," rather than "array BUFF was set to zero." Where applicable, reference underlying documentation that initiated the change, such as an STR number or the number of a specification modification.

Effect: Enter the names of all components that were modified and must be replaced in the configured library. The names listed must consist of the subsystem prefix followed by a component name of up to 40 characters. As on the COF, underscores connecting a prefix and a component name in the file-naming convention are not considered a part of either. All components listed must already have been entered into the database by

COFs. The version number, if available, should be supplied to assist the configuration manager. This should be the configured library version number of the component when it was checked out of the library for modifications, not the version number of the component in the developer's local directory or library, or the anticipated version number at which the component should be replaced in the library.

Although the form has room for only a limited number of components, all components affected by the change must be listed. This may be done by attaching a separate page to the CRF. The prefix, component name, and version number of each component must be clearly listed on the attachment.

Effort: Enter the names of all additional components that were examined, but not changed themselves, while determining the exact nature of the change. This list should not overlap the list of components actually changed. Version numbers need not be included in this list.

Location of developer's source files: Enter the name of the library (e.g., PANVALET library for FDF mainframes, directory or CMS library for STL VAX computers) in which the source code for the changed modules resides. This field is intended for use by the project configuration manager.

Need for change determined on: Enter the date on which the need for the change was first detected. For example, if the change involves an error correction, enter the date on which the symptoms of the error first appeared or were reported on an STR. For implementing a specifications change, enter the date on which the specifications modification was received. For a planned enhancement, the original configuration date of the affected components may be used.

Change completed: Enter the date on which the changed components are physically updated in the project's configured source library. This field is intended to be completed by the configuration manager.

Check here if change involves Ada components: Put a check in this box if any of the modified components listed is written in Ada. (If so, complete the questions on the reverse side of the form.) (See Figure 3-12 (2 of 2).)

Effort in person time to isolate the change: Put a check in the box that indicates how long it took to determine precisely what change was needed. This includes the effort required for understanding the change or finding the cause of the error, locating where the change is to be made, and determining that all effects of the change are accounted for. Note that this effort is to be reported in staff-days, not calendar days. If a team of five developers spent a full day isolating an error, the "> 3 days" box should be checked.

Effort in person time to implement the change: Put a check in the box that indicates how long it took to implement the change. This includes design changes, code modification, regression testing, and updates to documentation. Note again that effort is to be reported in staff-days.

Section B — All Changes

Type of change: Check the **one** change type from the list of change types that best describes the change. If more than one change type applies, that may be a clue that more than one logical change was made, in which case another CRF is required. There are, however, cases in which some confusion might arise. Generally, the first type listed that applies is the type that should be checked. Refer to the definitions below for clarification.

- *Error correction*: The change was made to correct an error that was introduced earlier in the life cycle. This includes errors in the requirements, the functional specifications, the design, or the code, as well as errors introduced as the result of previous changes. If this change type is checked, Section C of the CRF must also be completed.

- *Planned enhancement*: Code was inserted into a program stub that was initially created and configured as a dummy for testing purposes, or a planned capability was added to an already existing component. The key word is **planned**. The changed components must have initially been configured with the knowledge that they would later be modified. Another example is the addition of default values that were not defined when the component was originally developed.

- *Implementation of requirements change*: A requirement or functional specification was added, modified, or deleted. Usually, this type of change is the direct result of a specification modification. However, **if the specification modification was written to correct an error in the requirements or specifications, the change should be recorded as an "error correction."**

- *Improvement of clarity, maintainability, or documentation*: Changes were made to improve code quality, such as improving indentation or resequencing labels for readability, or adding or updating documentation or correcting grammatical errors in it. Nothing was technically wrong with the software, but changes were made to help future maintainers understand it better. Note, however, that improving the clarity of a display screen, help message, or any other end-user-oriented information should be classified as an "improvement of user services."

- *Improvement of user services*: This type of change is intended to improve the functionality, ease of use, or clarity of the system from the end-user's point of view. Do not check this category if the improvement was a "planned enhancement" or if it was required by a change in the requirements or specifications.

- *Insertion/deletion of debug code*: Changes were made to the program text specifically to provide additional information during test runs so that errors can

be isolated. Also check this category when such changes are removed from the program text.

- *Optimization of time/space/accuracy*: A localized adjustment was made to the program to reduce its execution time or memory or disk space requirements, or to obtain results of greater numerical accuracy by "tuning" the algorithms being used or converting to variables that allow greater precision. If the change resulted from a specification modification that introduced a new performance or accuracy requirement or made an existing one more stringent, the change is an "implementation of requirements change," and that category should be checked rather than this one.

- *Adaptation to environment change*: This category should be checked when implementing an unplanned change in response to a change that is outside of the system boundary. This includes a change in hardware, operating system, or compiler. This type should not be used if, for example, the changes were planned in order to move the system from its development environment to its target environment. It should also not be used if the change was initiated by a requirements change.

- *Other*: This category should be checked only if none of the preceding categories applies. Briefly describe the change type in the space below the checklist on the form.

Effects of change: For each of the three questions, check the appropriate answer ("yes" or "no"). Note that the answers to the first two questions must agree with the information supplied in the Effect and Effort items in Section A. Thus, if only one component is listed under Effect, the answer to the first question must be "yes." Similarly, if there are **any** components listed under Effort, the answer to the second question must be "yes." The third question should be answered "no" only if all of the changes made were localized to the components in which they were made. The intent is to record whether component interfaces were involved in the change or potentially affected by the change. Thus, for example, a change that affects values in a parameter list, FORTRAN COMMON block, or state data in an Ada package would require that this box be checked "yes," even if no other components had to be changed as a result.

Section C — For Error Corrections Only

This section must be completed if the type of change indicated in Section B is "error correction."

Source of error: Check the **one** box that best indicates in which phase of the development life cycle the error was introduced.

- Errors that originated in the *requirements* or *functional specifications* will normally be initiated by a specification modification. These would include such

errors as an error in one of the equations used to specify an algorithm to be implemented or the omission of a data item from a list of values required to be output on a display or report.

- *Design* errors are introduced in the process of transforming requirements and specifications into detailed (component-level) design. An example would be leaving a piece of required information out of a parameter list or omitting a step in a computation when generating PDL.
- *Code* errors are those errors that occur when transforming the detailed design to code, such as mistyping a variable name, incorrectly coding an assignment statement, or incorrectly coding the exit criteria for a loop.
- Finally, errors resulting from a *previous change* are those that were not in the system until some other change was implemented (in which case the implementer of the previous change did not consider all of its possible effects, or the change was simply implemented incorrectly).

Class of error: Check the **one** box that best classifies the error. If the error seems to fit into more than one class, check the first applicable class.

- *Initialization*: The error results from an incorrectly initialized variable, a failure to reinitialize a variable, or because a necessary initialization was missing. Failure to initialize or reinitialize a data structure properly upon a component's entry/exit would be considered an initialization error.
- *Logic/control structure* (e.g., flow of control incorrect): The error stems from an incorrect Boolean decision in a control structure. Errors causing an incorrect path to be taken in a component are considered logic/control structure errors.
- *Interface (internal)* (module-to-module communication): This is an error of data exchange within the system. Included in this category are parameter (calling sequence) errors, COMMON block errors, and errors in state data. An error in initializing COMMON block variables is considered an interface error and not an "initialization" error, because the COMMON block is used by the module but is not part of its local environment.
- *Interface (external)* (module to external communication): This is an error of data exchange between some module in the system and some external entity, such as system services, files, printers, or institutional software packages, e.g., GESS and FDAF.
- *Data (value or structure)* (e.g., wrong variable used): A data error is any error in the use of a variable or any error resulting from the incorrect use of a data structure. Examples of data errors are the use of incorrect subscripts for an array, the use of the wrong variable in an equation, the use of the wrong unit of measurement, or the inclusion of an incorrect declaration of a variable local to the component.

- *Computational* (e.g., error in math expression): This is an error in which an incorrect expression is computed, that is, a computation erroneously evaluates a variable's value. For example, a "+" was used where a "−" should have been used. This category does not include an error in which the wrong variable was used in the calculation; that is a "data" error.

Characteristics: All three of these questions must be answered. They are to be interpreted as follows:

- *Omission error* (e.g., something was left out): Check "yes" whenever the error was the result of missing code (even one statement or a part of one statement). The code may be missing because of an omission in a previous phase, such as a missing equation in the functional specification.
- *Commission error* (e.g., something incorrect was included): Check "yes" whenever the error resulted from wrong code as opposed to missing code. If the error included both incorrect code and missing code, check "yes" for both. There are **no** errors for which "no" is checked on both of these questions.

  NOTE: The above two questions are often misinterpreted. If something incorrect is replaced with something correct, this is an error of commission only. For example, if "X−Y" is replaced with "X+Y," the fact that the minus sign was incorrectly included and the plus sign was left out does **not** imply that the error was one of both commission **and** omission. Something incorrect was included and had to be **changed**, not added. Hence, it is an error of commission only. This is a subtle, but important, distinction.
- *Error was created by transcription* (clerical): Check "yes" only if the error was actually caused by a transcription mistake. This includes keying mistakes, spelling errors, etc.

Ada Project Additional Information (Reverse side of form)

This portion of the form must be completed if the box on the front side of the form is checked to indicate that the change affected components written in Ada.

Question 1: Should be answered for all changes involving Ada components. Select the category (or categories) that most closely characterize the change. A more detailed description of the categories follows:

- *Data typing:* Includes predefined and user-defined scalar types, variables, constants and actual parameters; subtypes and derived types; array types, variables, and constants; array slices; named and positional aggregates; string variables and constants; record types, variables, and constants; discriminated and variant records; and dynamic memory allocation, i.e., data structures using pointers.

- *Subprograms:* Includes function and procedure calls, "return" statements, named and default parameters, recursive subprogram calls, overloading of subprogram names, and user-defined operators (+,−,*,/).
- *Exceptions:* Includes predefined and user-defined exceptions, raising exceptions, "raise" statements, and handling exceptions.
- *Generics:* Includes declarations and instantiations of generic packages, type parameters, and subprogram parameters.
- *Program structure and packaging:* Includes package specifications; package bodies; package initialization; changes due to use of state information in package bodies; scope and visibility of subprogram formal parameters; local subprogram variables; variables declared in "block" statements; private and limited private types; generic and standard package implementations or instantiations of queues, linked lists, and stacks; changes, additions or deletions of "with" clauses; removal of inappropriate "use" statements; block statements; nesting of blocks, subprograms, packages, or tasks within other blocks, subprograms, packages, or tasks; and restructuring of software components (nesting vs. library units, subunits).
- *Tasking:* Includes entry calls, task buffers, task priority, task types and objects, task activation, task termination, family of entries, and selective wait.
- *System-dependent features:* Includes "delay" statements, objects of type TIME and DURATION, use of CALENDAR, exception "time_error," address clauses, length clauses, enumeration representation clauses, record representation clauses and alignment clauses, compiler directives (pragmas), importing of foreign code, predefined items and other environment-dependent features, and Ada low-level features.
- *Other:* Includes I/O features (Text_IO and instantiations of Integer_IO, Float_IO, Enumeration_IO, Sequential_IO, Direct_IO). If I/O was the cause of the change and the change is the result of an error, either "interface (internal)" or "interface (external)" should be checked under Class of Error on side one. *Other* also includes changes to Ada assignment and control statements and use of the "rename" statement. In this case, the class of error ("logic/control structure," "computational") should reflect this choice. If this category is selected, the feature involved must be supplied.

Question 2: Should be answered only if the type of change checked in Section B on the front of the form is "error correction."

- Question 2a should be answered "no" only if the compiler documentation or language reference manual was consulted when writing the code originally and led to a misinterpretation of the use of a particular Ada feature. It should be answered "yes" in all other cases, including those in which the documentation was not consulted.

- Question 2b should be answered regardless of whether the error was directly related to an Ada feature. Check the most applicable statement.

- Questions 2c and 2d are intended to capture the tools and resources used in isolating and correcting the error. Check as many items as apply.

Question 3: This question should be answered for all changes involving Ada components.

Helpful Hints

1. Perhaps the single most common error in completing the CRF occurs when identifying the source of an error in Section C. Logic says that because one is having to change code to correct the error, the "code" box is the one that should be checked. What is really intended is to determine **when in the life cycle the error was introduced**. The "code" box should only be checked if the error was made while transforming the detailed design (prolog and PDL) into code. If the PDL itself was wrong, the source of the error is "design." If the code correctly implements an incorrect requirement or specification, the appropriate one of those two boxes should be checked. Errors introduced by a previous change are a bit more difficult to identify, but revision histories in component prologs should provide enough information to allow that determination to be made.

2. When using the CRF as a configuration management tool, the configuration manager must be aware that components may appear on more than one CRF when being promoted from one version to the next, since they may be involved in more than one logical change being implemented at the same time. Those logical changes, however, should have been assigned to a single developer so that not more than one team member is working on a given component at the same time. It is suggested that, when possible, the team member submit the CRFs for multiple logical changes affecting the same component or components as a package, perhaps clipped together, to simplify the configuration manager's job of determining precisely what components need to be replaced in the configured source library.

3. A question that often arises is "What if the logical change involves adding a component to the system or deleting a component from the system?" New components resulting from the implementation of a change should be documented on COFs (Section 3.2.2.2) and should not be listed on the CRF. Similarly, deleted components should be identified to the SEL via a CCF (Section 3.2.2.3) and should not be listed on the CRF.

4. Another common question is "If the error involves Ada components, but was truly language independent (i.e., had nothing to do with the use of Ada), how should Question 2b on the back of the form be answered?" In this case, there

was no misunderstanding of Ada features, nor of their interactions, so the second response, "Understood features, but did not apply correctly," should be checked, even for something as simple as a misspelled variable name or incorrect operator in an assignment statement.

5. Changes made to components linked to the system from external reuse sources (RSL, MTASS, MSASS) should not be reported on CRFs by the reusing project.

3.2.2.5 PROJECT MESSAGES FORM (PMF)

General Information

The PMF (Figure 3-13) is completed by SEL data collection personnel to record general information about a given project or the data that have been collected for it. It is not filled out directly by developers. The developers may, if they wish, submit to SEL personnel specific messages that explain some aspect of the project or its data collection. More commonly, however, a data collection team member will draft and enter messages based on conversations and other interaction with project personnel.

Line-by-Line Instructions

Name: Enter the name of the data collection team member completing the form.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored.

Date: Enter the date on which the form is completed.

Messages: Enter free format text that describes the information being documented about the project.

Helpful Hint

To make these messages more readable when they are retrieved from the database or output in reports, they should be organized for readability. Devoting separate paragraphs to each major point is one technique. A bulleted list of major points is another.

3.3 PROJECT COMPLETION

When SEL-monitored development projects are complete, and the software has been delivered to maintenance and operations (end of cleanup phase), the project data undergo a process of validation and verification referred to as "closeout." This process involves a final quality assurance of all data collected on the project, reconciliation of the components stored in the SEL database against those in the delivered source library of the project, and computation of final size and resource use statistics for the project.. Once the SEL has determined the final statistics for a project, the project data are

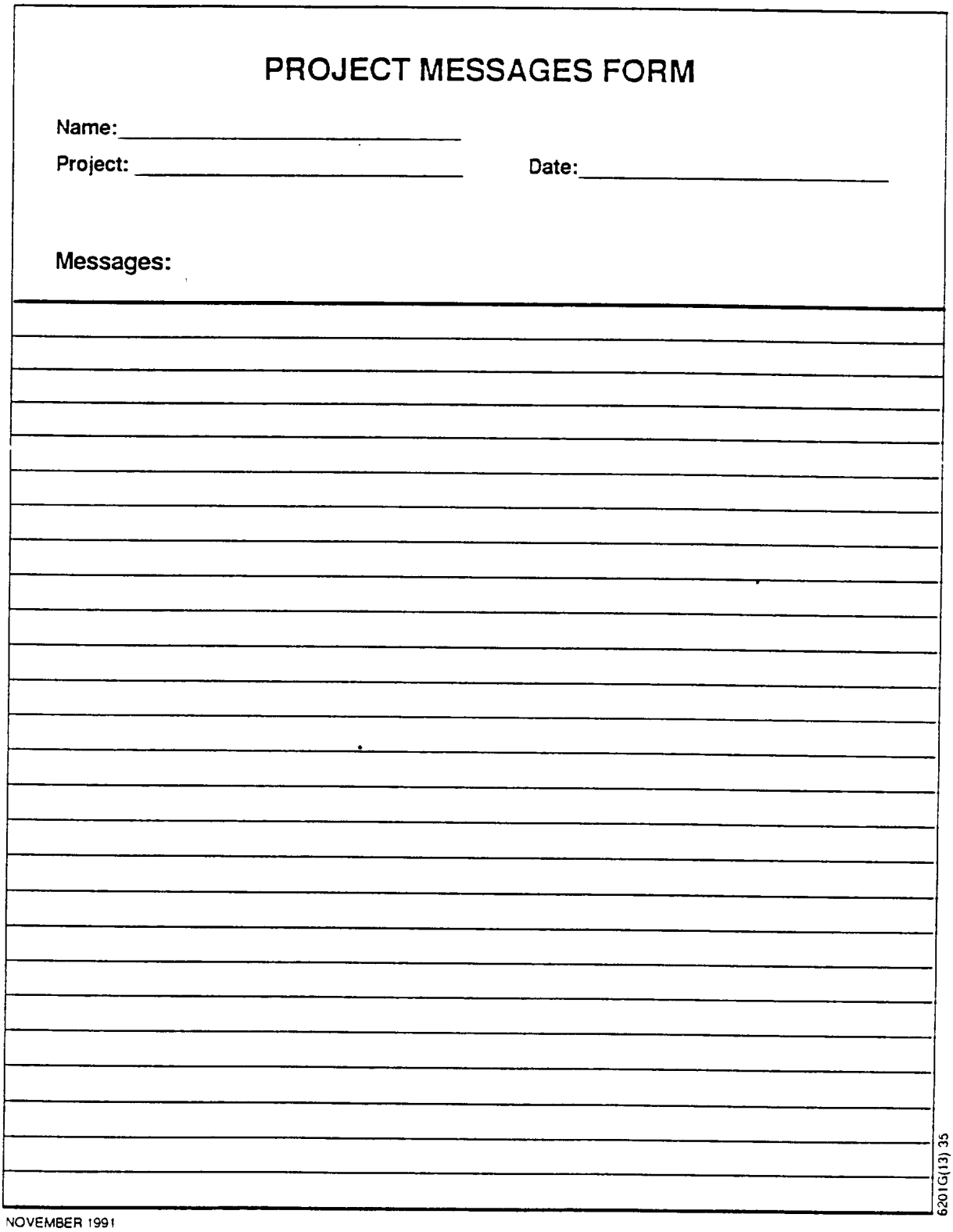


Figure 3-13. Project Messages Form

provided to the project leader for review. SEL personnel record the final statistics on a PCSF, which they work with the project leader to validate and complete. They also collect an SEF, which contains the consensus opinion of the managers involved in the project on a list of subjective project attributes. Finally, any last annotations about the methodologies employed on the project, unusual or unresolved anomalies in the project's data, and information about the data collection activities on the project that might help users of the data to interpret them more accurately are recorded on a PMF (Section 3.2.2.5).

One of the first steps listed above refers to reconciling components in the database with those in the project's configured source library. This is a necessary precursor to computing final project size statistics broken down by code origin, which is done from component-level size statistics since the component is the only system element for which origin information is recorded (on COFs). Thus, the goal of this step is to map every component for which the SEL received a COF to a source file in the delivered system and vice versa. This process often involves requesting COFs from the project leader for library components not found in the database, verifying and deleting COFs for components that are no longer part of the system, and mapping database (COF) names to file names in cases where naming conventions were not (or could not be) followed and the two names are different.

Individual components reused by linking in object code from separately maintained libraries (RSL, MTASS, MSASS) should also have database names reconciled with or mapped to the file names in the source libraries for these reuse sources.

Once SEL personnel have matched database names to file names and the locations of the files in source code libraries, they can then run the source code through line and statement counting tools to compute component-level size statistics. Line counts are computed for every component of the system. Comment counts are computed for every FORTRAN or Ada component. This includes INCLUDE files. Statements and executable statements are computed for every FORTRAN component that contains a complete, compilable FORTRAN element (subroutine, function, BLOCK DATA). Statements are also counted for every Ada component. The FORTRAN statements and executable statements are counted, with INCLUDE files expanded, by running them through the FORTRAN Static Source Code Analyzer Program (SAP) (Reference 15). SAP also produces a number of other metrics, and the reports it generates are stored in hard copy form by SEL personnel for future reference and access by researchers. Figure 3-14 is a sample SAP output report.

Once component-level statistics have been entered into the database, project completion statistics may be summed. These are transferred by SEL personnel to a PCSF. At this point, a member of the data collection team meets with the project leader and asks him/her to verify the project completion statistics and to supply any that might be missing. The data collector and the project leader then examine the project data by viewing them in various graphical representations. If any anomalies are observed, these are discussed, and possible causes are noted for documentation as project messages. At this

interview, the project leader supplies any information about the project, its life cycle, or the methodologies employed that may help researchers to understand the data collected. The data collector documents the results of this interview and enters them into the database by means of a PMF.

While the above steps are progressing, the data collection team also asks the GSFC contact for the project to complete an SEF. When this has been completed and entered and all of the above steps are also complete, the project is considered closed out, and the data may be used for research.

3.3.1 Project Completion Statistics Form (PCSF)

General Information

The PCSF (Figure 3-15) is used to record the actual project schedule and project-level size and resource use statistics at project completion. It is completed and entered one time only. To facilitate its collection, SEL personnel complete the form as far as possible by summing lower level data to obtain project-level totals and recording those totals on the form. They then ask the project leader to verify the data and provide missing information. Usually, the only field that the project leader must supply is the Pages of Documentation field, which cannot be summed up from lower level data collected during development.

The line-by-line instructions that follow outline how SEL personnel compute the values they enter in each field and what the project leader should consider in verifying or updating the information.

Line-by-Line Instructions

Name: Enter the SEL database name of the project leader who verifies and updates the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA.

SOURCE ANALYZER PROGRAM V3 GLOBAL STATISTICS SUMMARY FILE LUN 8 AHFCMP PAGE 1

GTL_DISK2:[SWDATA.SAP]SAMPLE_WT.SAP/-MO

MODULE DIRECTORY

MODULE NAME		SOURCE LINES	CMTS	EXEC STMT	N-EXC STMTS	STATMENT INFORMATION COUNT ASGN	I/O	CNTL	STRU	INCL	OTHER	HALSTEAD OPTR	OPND	****** COMPLEXITITES CYCLO	SEL	USER1	USER2	PAGE NO.	NUMBER OF ERRS	WARN
1 AHFCMP	SU	159	88	43	32	19	5	18	5	0	5	238	202	18	355.5	0.0	0.0	0	0	0
2 BLDOAT	SU	109	70	20	21	9	3	8	0	0	2	122	105	6	175.0	0.0	0.0	0	0	0
3 BLDHED	SU	812	196	426	247	210	2	214	0	0	0	2078	2254	73	4709.0	0.0	0.0	0	0	0
4 BLDNDX	SU	151	84	33	35	7	10	16	4	0	5	181	154	11	201.5	0.0	0.0	0	0	0
5 DMPNDX	SU	85	42	8	22	2	3	3	4	0	0	52	62	3	105.5	0.0	0.0	0	0	0
6 PRAHF	SU	847	184	212	385	0	141	71	0	0	1	1695	1940	72	1954.0	0.0	0.0	0	0	0
7 WTHIST	SU	763	311	161	296	68	15	67	30	0	6	1248	1713	69	2587.5	0.0	0.0	0	0	0
8 WTINIT	SU	95	61	15	21	5	4	6	2	0	2	80	30	6	134.5	0.0	0.0	0	0	0

Figure 3-14. Sample SAP Printout (1 of 3)

```
    SOURCE ANALYZER PROGRAM V3   GLOBAL STATISTICS SUMMARY FILE          LUN  8    WTINIT              PAGE    2

STL_DISK2:[SWDATA.SAP]SAMPLE_WT.SAP/-MO
                                        ***** GLOBAL SUMMARY *****

GLOBAL       TOTAL   NUMBER      NUMBER          NUMBER          NUMBER        SOURCE    CODE      COMMENT
MODULE       MODULES MAINS       SUBROUTINES     FUNCTIONS       BLOCKDATAS    LINES     LINES     LINES
               8       0           8               0               0           3021      1985      1036

MODULE                      LINES PER MODULE                    PROLOGUE        COMMENT LINES
COMMENTING       TOTAL          CODE            COMMENT         LENGTH      EMBEDDED          BLANK
STATISTICS    847 MAX.       663 MAX.        311 MAX.        190 MAX.      121 MAX.          38 MAX.
            377.6 AVG.     248.1 AVG.      129.5 AVG.       86.6 AVG.     42.9 AVG.         19.1 AVG.

              ENTRY POINTS   SUBR. CALLS   FUNCT. REF.   EXTERNAL   EXTERNALLY      ASF           ASF        ARG. LIST LNGTH
MODULE          0 TOTAL      168 TOTAL     2 TOTAL       NAMES      DEFINED         DEFINITIONS   REFERENCES IN REFERENCES
COMMUNICATION PER MODULE     PER MODULE    PER MODULE    DEFINED    REFERENCES      PER MODULE    PER MODULE TO SUBR/FUNCT.
STATISTICS      0 MAX        141 MAX       1 MAX         141 MAX      0 MAX           0 MAX         0 MAX        13 MAX
              0.0 AVG      21.0 AVG      0.3 AVG       21.5 AVG     0.0 AVG         0.0 AVG       0.0 AVG       2.7 AVG

                    EXECUTABLE                          NON-EXECUTABLE                      MISCELLANEOUS
                CNT.  PCT. STATEMENT          CNT.  PCT. STATEMENT          CNT.  PCT. STATEMENT
STATEMENT       918  46.4 EXECUTABLE         1059  53.6 NON-EXECUTABLE       0   0.0  NAMELIST
CLASS           320  16.2 ASSIGNMENT           16   0.8  SUBPROGRAM          11   0.6  DATA
COUNTERS        403  20.4 CONTROL             409  20.7  SPECIFICATION        0   0.0  ASF DEFINED    0   0.0  INCLUDE
                 45   2.3 STRUCTURED          401  20.3  TYPE SPECIF.       168   8.5  FORMAT        21   1.1  OTHER
                183   9.3 I/O

                  0  ASF DEF.    320  ASSIGNMENT     0  ACCEPT       0  ASSIGN       0  AT           0  BACKSPACE
                  0  BLOCKDATA     0  BYTE         168  CALL       112  CHARACTER    0  CLOSE        8  COMMON
                  0  COMPLEX      21  CONTINUE      11  DATA         0  DEBUG        0  DECODE       0  DEFINEFILE
                  0  DELETE        0  DIMENSION      0  DISPLAY      0  DOUBLECOMP   0  DOUBLEPREC   0  DOWHILE
                 12  DO            0  EJECT         12  ELSEIF       7  ELSE         0  ENCODE       0  ENDDO
                  0  ENDDEBUG      0  ENDFILE       26  ENDIF        8  END          0  ENTRY      400  EQUIVALENC
                  0  EXTERNAL      0  FIND         168  FORMAT       0  FUNCTION    88  GOTO         0  .IF
                127  IF            0  IMPLICIT       0  INCLUDE      0  INQUIRE     46  INTEGER      0  INTRINSIC
                  5  LOGICAL       0  NAMELIST       0  OPEN         1  PARAMETER    0  PAUSE        0  PRINT
                  0  PROGRAM       1  READ         238  REAL         8  RETURN       2  REWIND       0  REWRITE
                  0  SAVE          0  STOP           8  SUBROUTINE   0  THEN         0  TRACEOFF     0  TRACEON
                  0  TYPE          0  WAIT         180  WRITE        0  UNDECODED    0  UNLOCK       0  VIRTUAL

                IF STMTS          BLOCKIF          GOTO STMTS        DO STMTS          DO LOOP          STMTS PER
CONTROL         PER MODULE        NESTING          PER MODULE        PER MODULE        NESTING DEPTH    DO LOOP
STATEMENT       MAX.   AVG.       MAX.   AVG.      MAX.   AVG.       MAX.   AVG.       MAX.   AVG.      MAX.   AVE.
BREAKDOWN        72    15.9        3     1.9        70    11.0        4     1.5         2     1.2        15    5.9

ASSIGNMENT
STATEMENT       VARIABLES PER ASSIGNMENT        OPERATORS PER STATEMENT          SUBSCRIPT COMPLEXITY
BREAKDOWN       1.9 AVG.        7 MAX.          0.4 AVG.        4 MAX.           6 MAX.      1.1 AVG.

                VARIABLES NAMED     VARIABLES REFERENCED          EQUIVALENCED        DIMENSIONS       CHARACTERS
SPECIFICATION   PER MODULE          PER MODULE                    NAMES PER MODULE    PER ARRAY        PER VARIABLE
STATEMENT       MAX.    AVG.        EXEC. STMTS    COMMONS        MAX.     AVG.       MAX.    AVG.     MAX.    AVG.
BREAKDOWN       361     140.0       MAX.  AVG.     MAX. AVG.      324      125.3        2      1.0       6      5.6
                                    242   66.5      75   29.4
```

Figure 3-14. Sample SAP Printout (2 of 3)

SOURCE ANALYZER PROGRAM V3 GLOBAL STATISTICS SUMMARY FILE LUN 8 WTINIT PAGE 3

STL_DISK2:[SWDATA.SAP]SAMPLE_WT.SAP/-MO

COMPLEXITY ANALYSIS	HALSTEAD ANALYSIS OPERATORS TOTAL	OPERATORS UNIQUE	OPERANDS TOTAL	OPERANDS UNIQUE	LEVEL PROGRAM	LEVEL LANGUAGE	SEL COMPLEXITY	CYCLOMATIC COMPLEXITY	QUALITY INDEX	PREDICTED PROGRAM LENGTH	PREDICTED EFFORT REQUIRED
TOTAL	5694	178	6510	1331	0.131	11.059	10222.50	258	650.075	11681.0	129546224.0
MAX.	2078	41	2254	412	0.045	5.980	4709.00	73	86.647	3798.0	119550880.0
MEAN	711.8	22.3	813.8	166.4	0.016	1.382	1277.81	32.25	81.259	1460.125	16193278.000
STD. DEV.	775.3	9.3	906.0	172.5	0.014	1.907	1575.29	30.58	4.224	1554.032	39125912.000

Figure 3-14. Sample SAP Printout (3 of 3)

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored.

Date: Enter the date on which the form is completed by SEL personnel and submitted to the project leader for review and update.

Phase Dates

The phase dates entered by SEL personnel are those that appear on the most recent PEF submitted by the project leader. These dates must be Saturdays. (See the discussion of phase dates under the PEF instructions in Section 3.1.2.) The project end date should be the date on which all system products (source libraries and documents) are delivered to the organization that will be performing the maintenance and operations phase. Note that a maintenance phase start date should not be supplied at this time.

Staff Resource Statistics

The fields in this area of the form are computed from previously collected data as defined in the descriptions that follow. The project leader should check these totals against his/her most recent estimates for effort expenditures and against effort expenditures as recorded on organizational accounting records. Such comparisons will obviously not result in direct matches, but should be used to provide a "sanity check" on the final effort numbers being recorded. A large disparity might point to a problem in the data collected on the project. This should be investigated by SEL personnel and an annotation should be made via a PMF if a problem is found.

Technical and management hours: The total entered by SEL personnel is the sum of the activity hours recorded on all PRFs, plus the sum of all project management hours recorded weekly on SPFs.

Services hours: The total entered by SEL personnel is the sum of the support service personnel activity hours recorded on SPFs (support services personnel include secretaries, technical publications personnel, couriers, project control, etc.).

PROJECT COMPLETION STATISTICS FORM

Name: _____

Project: _____ Date: _____

Phase Dates (Saturdays)	
Phase	Start Date
Requirements Definition	
Design	
Implementation	
System Test	
Acceptance Test	
Cleanup	
Maintenance	
Project End	

Staff Resource Statistics	
Technical and Management Hours	
Services Hours	

Computer Resource Statistics		
Computer	CPU hours	No. of runs

Project Size Statistics			
General Parameters		Source Lines of Code	
Number of subsystems		Total	
Number of components		New	
Number of changes		Slightly Modified	
Pages of documentation		Extensively Modified	
		Old	
		Comments	

Executable Modules		Executable Statements		Statements	
Total		Total		Total	
New		New		New	
Slightly Modified		Slightly Modified		Slightly Modified	
Extensively Modified		Extensively Modified		Extensively Modified	
Old		Old		Old	

Note: All of the values on this form are to be actual values at the completion of the project. The values entered by hand by SEL personnel reflect the data collected by the SEL during the course of the project. Update these according to project records and supply values for all blank fields.

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

6201G(39)-11

NOVEMBER 1991

Figure 3-15. Project Completion Statistics Form

Computer Resource Statistics

The fields in this area of the form are computed from previously collected data as defined in the descriptions that follow. It is not expected that project leaders will have kept their own records of computer use. Thus, no validation of the data in these fields is necessary. The project leader should, however, verify that there is an entry for each computer used on the project for which he/she provided account identifiers to SEL personnel for monitoring.

Computer: The computer names entered by SEL personnel are the abbreviated SEL CPU names for each SEL-monitored computer used on the project.

CPU hours: For each CPU name listed, the total entered by SEL personnel is the sum of CPU hours used during the life cycle of the project as measured by system accounting software and recorded weekly by SEL personnel on SPFs.

Number of runs: For each CPU name listed, the total entered by SEL personnel is the sum of runs made on that computer as measured by system accounting software and recorded weekly by SEL personnel on SPFs. See the SPF instructions (Section 3.2.1.3) for the definition of a computer run.

Project Size Statistics

The remainder of the PEF is used to record measures that characterize the size of the final delivered product. Several of the subcategories under project size classify measures as new, slightly modified, extensively modified, or old. The Total field is always the sum of these four categories. SEL personnel compute these classifications by summing component-level size statistics with the components grouped by their "final" origin. The components that fall into each of these "final" origin categories are as follows:

- *New*: All components for which the Origin field on the COF was checked "new"
- *Slightly modified*: All components for which the Origin field on the COF was checked "slightly modified" **plus** all components for which the Origin field on the COF was checked "old unchanged," but which subsequently appeared as changed components on one or more CRFs
- *Extensively modified*: All components for which the Origin field on the COF was checked "extensively modified"
- *Old*: All components for which the Origin field on the COF was checked "old unchanged" **minus** those that subsequently appeared as changed components on one or more CRFs

Generally, the project leader does not keep records of size statistics broken down into these categories. It is not expected that he/she will be able to verify the breakdown

against project records. He/she should, however, have records of total counts for these size statistics and an idea of the level of reuse achieved. These should be used to perform a "sanity check" on the data, and any apparent discrepancies should be communicated to SEL personnel for investigation.

Project Size Statistics—General Parameters

The fields in this area of the form contain high-level project size statistics. All but the last field (Pages of Documentation) will have been completed by SEL personnel. A description of these four fields follows.

Number of subsystems: The number entered by SEL personnel is the one that appears on the most recent PEF submitted by the project leader. This number should be verified to represent the number of logical subsystems present in the design of the final delivered system. This will not necessarily match the number of subsystem prefixes that have been identified on SIFs. (See the discussion of subsystems under the PEF instructions in Section 3.1.2.)

Number of components: The number entered by SEL personnel is the total number of COFs received on the project (after they have been reconciled with the project's configured libraries) **plus** the sum of the number of components present in each of the subsystems reused in its entirety from a generic reuse source (such as MTASS or MSASS). Since these counts were to have been factored into the estimates supplied on PEFs, this number should be fairly close to the number of components estimated on the most recently submitted PEF.

Number of changes: The number entered by SEL personnel is the actual number of logical changes made to the system as reported to the SEL on CRFs. (See the discussion of the CRF in Section 3.2.2.4 for a definition of what constitutes a "change.") If the project leader has kept separate records of system changes, this number should be verified against those records.

Pages of documentation: Sum and enter the total page count for the following types of documents produced on the project. Note that this field is not completed by SEL personnel. The project leader must supply this information.

- Software development/management plans (including any separate quality assurance (QA) or configuration management (CM) plans)
- User's guides (finals only)
- System descriptions (finals only)
- Design book/detailed design notebook or document produced at Critical Design Review (CDR)
- Test plans (integration, build, and system test—**not** acceptance test)
- Prologs and PDL (count 1 page per system component)

Project Size Statistics—Source Lines of Code

As defined in the instructions for the PEF, SLOC is a count of carriage returns, or card images. It includes code, comments, blank lines, and data. SEL personnel count SLOC for every component for which a COF has been submitted, regardless of component type. These counts are made without expanding INCLUDE files. Refer to general instructions for project size statistics for a description of how size statistics are classified by origin.

Comments: This number is a count of the source lines that begin with a comment identifier. Only FORTRAN and Ada source code components are included in this count. As with SLOC, INCLUDE files are not expanded when counting comments. Blank lines are **not** counted as comments.

Project Size Statistics—Executable Modules

These measures are computed by counting the number of COFs on which a Component Purpose was recorded. (See Section 3.2.2.2.) Recall that a purpose is to be supplied for all Ada components and for components of any other type that contain executable code. Refer to general instructions for project size statistics for a description of how size statistics are classified by origin.

Project Size Statistics—Executable Statements

These measures are computed for FORTRAN source code components only. They are generated by running the source code through the SAP program with INCLUDE files expanded. The FORTRAN statements that are classified as executable are identified in the KEYWORDS.SAP file, an input file to the SAP program. (See Reference 15 for more detailed information on SAP and the classification of FORTRAN executable statements.) Refer to general instructions for project size statistics for a description of how size statistics are classified by origin.

Project Size Statistics—Statements

These measures are computed for FORTRAN and Ada source code components only. The FORTRAN statements are counted by running the source code through the SAP program with INCLUDE files expanded. The Ada statements are counted by running the source code through a statement counting tool maintained by SEL personnel that counts terminating semicolons (i.e., excluding those occurring in parameter lists). Refer to general instructions for project size statistics for a description of how size statistics are classified by origin.

Helpful Hints

1. Note that the final actual schedule, resource use statistics, and number of changes characterize the process and the resources used to produce the

portion of the final product actually **developed** by the development team. The size statistics, however, characterize the entire final **delivered** product, including certain application software reused by linking it in from external sources. This distinction causes some project leaders to question the data recorded on the form when projects achieve a high level of reuse. A relatively low expenditure of staff resources, for example, to develop a very large system might give the impression that a much higher productivity was achieved than that recorded by the project leader in his/her own monitoring of the project. These are, however, the data that the SEL would like to record. Size statistics must reflect the entire application so that comparisons with similar applications will be valid.

2. Remember that the number of subsystems recorded on this form should include any subsystems reused in their entirety from application-specific reuse sources, such as MTASS and MSASS. It is important to make sure that SEL personnel are aware of these subsystems so that they may factor them into total size statistics and record project messages documenting the fact that they were reused.

3. Although the executable statements and statements measures may seem intuitively to be a subset of the SLOC measures, they differ, as has been pointed out, in that FORTRAN executable statements and statements are counted with INCLUDE files expanded. Thus, some of the developed source code is counted multiple times in taking these measures. The rationale behind this is that the SLOC measure is intended to capture raw system size, whereas the statement counts are intended to capture the volume of system code that must be processed by a language processor or translator. Traditionally in the SEL, executable statements was the measure that captured this. With the introduction of Ada in the FDD environment, a definition of Ada executable statements that would yield a size measure suitable for the comparison of FORTRAN and Ada projects was not readily apparent. Thus, the measure of total statements was introduced for both FORTRAN and Ada projects. Executable statements are still counted for FORTRAN projects so that they may be compared with older FORTRAN projects in the data base.

3.3.2 Subjective Evaluation Form (SEF)

General Information

To complement the objective statistics generated in the closeout process, the SEL requests that the project's GSFC contact, or ATR, complete an SEF (Figure 3-16). The ATR is asked to solicit input from all of the project leaders (GSFC and contractor technical leads and line managers) and determine a composite answer for each of the questions on the SEF. This information provides overall subjective opinions that characterize the problem, process, environment, resources, and product.

SUBJECTIVE EVALUATION FORM

Name: _____

Project: _____ Date: _____

Indicate response by circling the corresponding numeric ranking.

I. PROBLEM CHARACTERISTICS

1. Assess the intrinsic difficulty or complexity of the problem that was addressed by the software development.

1	2	3	4	5
Easy		Average		Difficult

2. How tight were schedule constraints on project?

1	2	3	4	5
Loose		Average		Tight

3. How stable were requirements over development period?

1	2	3	4	5
Loose		Average		High

4. Assess the overall quality of the requirements specification documents, including their clarity, accuracy, consistency, and completeness.

1	2	3	4	5
Low		Average		High

5. How extensive were documentation requirements?

1	2	3	4	5
Low		Average		High

6. How rigorous were formal review requirements?

1	2	3	4	5
Low		Average		High

II. PERSONNEL CHARACTERISTICS: TECHNICAL STAFF

7. Assess overall quality and ability of development team.

1	2	3	4	5
Low		Average		High

8. How would you characterize the development team's experience and familiarity with the application area of the project?

1	2	3	4	5
Low		Average		High

9. Assess the development team's experience and familiarity with the development environment (hardware and support software).

1	2	3	4	5
Low		Average		High

10. How stable was the composition of the development team over the duration of the project?

1	2	3	4	5
Loose		Average		High

FOR LIBRARIAN'S USE ONLY

Number: _____ Entered by: _____

Date: _____ Checked by: _____

6201G(13) 29

NOVEMBER 1991

Figure 3-16. Subjective Evaluation Form (1 of 3)

SUBJECTIVE EVALUATION FORM

III. PERSONNEL CHARACTERISTICS: TECHNICAL MANAGEMENT

11. Assess the overall performance of project management.

1	2	3	4	5
Low		Average		High

12. Assess project management's experience and familiarity with the application.

1	2	3	4	5
Low		Average		High

13. How stable was project management during the project?

1	2	3	4	5
Low		Average		High

14. What degree of disciplined project planning was used?

1	2	3	4	5
Low		Average		High

15. To what degree were project plans followed?

1	2	3	4	5
Low		Average		High

IV. PROCESS CHARACTERISTICS

16. To what extent did the development team use modern programming practices (PDL, top-down development, structured programming, and code reading)?

1	2	3	4	5
Low		Average		High

17. To what extent did the development team use well-defined or disciplined procedures to record specification modifications, requirements questions and answers, and interface agreements?

1	2	3	4	5
Low		Average		High

18. To what extent did the development team use a well-defined or disciplined requirements analysis methodology?

1	2	3	4	5
Low		Average		High

19. To what extent did the development team use a well-defined or disciplined design methodology?

1	2	3	4	5
Low		Average		High

20. To what extent did the development team use a well-defined or disciplined testing methodology?

1	2	3	4	5
Low		Average		High

IV. PROCESS CHARACTERISTICS

21. What software tools were used by the development team? Check all that apply from the list that follows and identify any other tools that were used but are not listed.

- ☐ Compiler
- ☐ Linker
- ☐ Editor
- ☐ Graphic display builder
- ☐ Requirements language processor
- ☐ Structured analysis support tool
- ☐ PDL processor
- ☐ ISPF
- ☐ SAP
- ☐ CAT
- ☐ PANVALET
- ☐ Test coverage tool
- ☐ Interface checker (RXVP80, etc.)
- ☐ Language-sensitive editor
- ☐ Symbolic debugger
- ☐ Configuration Management Tool (CMS, etc.)
- ☐ Others (identify by name and function)

22. To what extent did the development team prepare and follow test plans?

1	2	3	4	5
Low		Average		High

6201(13)-30

Figure 3-16. Subjective Evaluation Form (2 of 3)

SUBJECTIVE EVALUATION FORM

IV. PROCESS CHARACTERISTICS (CONT'D)

23. To what extent did the development team use well-defined and disciplined quality assurance procedures (reviews, inspections, and walkthroughs)?

1	2	3	4	5
Low		Average		High

24. To what extent did development team use well-defined or disciplined configuration management procedures?

1	2	3	4	5
Low		Average		High

V. ENVIRONMENT CHARACTERISTICS

25. How would you characterize the development team's degree of access to the development system?

1	2	3	4	5
Low		Average		High

26. What was the ratio of programmers to terminals?

1	2	3	4	5
8:1	4:1	2:1	1:1	1:2

27. To what degree was the development team constrained by the size of main memory or direct-access storage available on the development system?

1	2	3	4	5
Low		Average		High

28. Assess the system response time: were the turnaround times experienced by the team satisfactory in light of the size and nature of the jobs?

1	2	3	4	5
Poor		Average		Very Good

29. How stable was the hardware and system support software (including language processors) during the project?

1	2	3	4	5
Low		Average		High

30. Assess the effectiveness of the software tools.

1	2	3	4	5
Low		Average		High

VI. PRODUCT CHARACTERISTICS

31. To what degree does the delivered software provide the capabilities specified in the requirements?

1	2	3	4	5
Low		Average		High

32. Assess the quality of the delivered software product.

1	2	3	4	5
Low		Average		High

33. Assess the quality of the design that is present in the software product.

1	2	3	4	5
Low		Average		High

34. Assess the quality and completeness of the delivered system documentation.

1	2	3	4	5
Low		Average		High

35. To what degree were software products delivered on time?

1	2	3	4	5
Low		Average		High

36. Assess smoothness or relative ease of acceptance testing.

1	2	3	4	5
Low		Average		High

6201G(13) 31

Figure 3-16. Subjective Evaluation Form (3 of 3)

Line-by-Line Instructions

Name: Enter the SEL database name of the ATR responsible for completing the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA.

Project: Enter the acronym selected at project startup that uniquely identifies the project being monitored.

Date: Enter the date on which the form is completed.

General Instructions

The questions on this form are self-explanatory. Therefore, no further line-by-line instructions are provided. The form asks for retrospective subjective opinions in six major areas of the software development effort. These areas are problem characteristics, personnel characteristics (technical staff), personnel characteristics (technical management), process characteristics, environment characteristics, and product characteristics. The 36 questions are grouped into these six categories. With the exception of question 21, all questions require the circling of a single numeric response indicating a subjective opinion. Question 21 requires an "X" or check mark in the box by each tool listed that was used on the project.

Helpful Hint

The technique for collecting a composite opinion is left to the discretion of the ATR. One suggestion is to have each technical lead or line manager complete a copy of the form, compile the results, and try to resolve outlying rankings by discussing them with the individuals who gave them. Another option is to hold a meeting of the individuals involved and come to a consensus in that setting.

SECTION 4—DATA COLLECTION IN MAINTENANCE

This section presents detailed procedures for collecting SEL data during the maintenance and operations phase of the software life cycle. It begins with a discussion of data collection activities during the transition to maintenance, which may involve a change in the organization responsible for the system. This is followed by an overview of data collection activities during maintenance, which includes an outline of the maintenance procedures followed on projects in the GSFC FDD environment. Lastly, detailed instructions for completing the SEL data collection forms submitted during maintenance are presented in the same format as those presented in Section 3 (i.e., background information, line-by-line instructions, and helpful hints).

4.1 TRANSITION TO MAINTENANCE

When the products of a SEL-monitored development project are delivered for operational use, the SEL initiates the collection of maintenance data. Often, the organization responsible for maintenance is different from the organization responsible for development. Thus, as in project startup, the first thing the SEL must do is establish lines of communication with the maintenance team and obtain some basic information about the project.

SEL data collection personnel schedule a meeting with the maintenance team to discuss maintenance data collection activities. This meeting should be scheduled as soon as maintenance work begins on the system. This generally does not start until the system has been accepted. There may, however, be some overlap with the collection of development data, especially if the development team continues to work on final system documentation and the project history report after the actual software is delivered.

The main purposes of this meeting are (1) to acquaint the maintainers with the SEL data collection process and their role in that process, (2) to establish naming conventions and identify team members who have not previously submitted SEL forms and whose names will have to be entered into the database, and (3) to give the data collectors an understanding of the maintenance process being followed and any peculiarities that may have an impact on the accuracy or completeness of the data collected. An example of such a peculiarity would be a maintenance effort that was split among two or more organizations, with only one of whom the SEL has made arrangements to collect data. In such a case, the data collected will be incomplete, and SEL personnel will note this in the form of project messages so that researchers will understand the limitations of the data.

SEL personnel use the PSF (Section 3.1.1) to document the information gathered at the maintenance startup meeting. They complete the header information, contacts, forms to be collected, general notes, and personnel names portions of that form. As in development, it is very important to establish a single point of contact, or project leader, with

whom SEL personnel will communicate on data collection issues. In addition, if the project was not monitored by the SEL during development or is being maintained under a different name from that used during development, the project full name, language, and computer system fields are completed. Filling in the computer account and task number fields is not necessary, since the SEL does not collect computer resources data or services effort data during maintenance.

Two additional pieces of information the SEL must obtain at the startup meeting are the names of the configured libraries to be monitored for growth and changes and the scheduled duration of the maintenance activity. This last item, the schedule, is collected in lieu of collecting estimates of the scope of the maintenance activity and the effort that will be involved (as is done via the PEF during development).

The maintenance start date recorded should be the date on which the maintainers assume responsibility for the system, regardless of whether the developers have completed and delivered all final documentation. The maintenance end date is the date on which SEL monitoring of the maintenance activity is expected to cease. Generally, this is either the point at which the system is transferred to an organization from which the SEL does not collect data, or it is the estimated end of the operational life of the system.

4.2 DATA COLLECTION DURING MAINTENANCE

Once maintenance startup information and start and stop dates for the maintenance phase have been collected, the SEL collects several types of data on a regular basis throughout the maintenance phase. Data collected during maintenance and operations include maintenance effort and growth, which are rate data, and maintenance changes/errors and messages, which are event data.

The maintenance life cycle, as performed in the GSFC FDD environment and documented in the *Operational Software Modification Procedures (OSMP)* (Reference 16), centers on the concept of a logical change (as defined in Section 3.2.2.4). The cycle begins with identifying the need for a change, be it an error correction, an enhancement, or an adaptation to changes in the environment, and documenting it on an Operational Software Modification Report (OSMR) form.

Sometimes changes are initiated by changes to the system requirements or specifications. The SEL does **not** collect data on the work performed to update these documents, but rather collects data on the implementation of approved OSMRs.

Once the OSMR is approved, work on implementing the change begins. This includes designing, coding, and testing the change. Once the maintainer has completed and tested the change in his/her local library, there are two additional levels of testing that are performed at two different levels of configured source code libraries: integration testing and acceptance testing. Changes that successfully pass acceptance testing are promoted to the operational library.

The SEL collects data through all of the above levels of maintenance activities. In some cases, the different levels of testing are performed by different organizations. For

example, a maintenance organization might perform the implementation of the change and integration testing, after which an operations organization takes over for acceptance testing. If the SEL has not established data collection agreements with one or the other organization, the data collected will be incomplete. The segments of the Operational Software Modification (OSM) life cycle for which the SEL does not receive data should be noted as project messages, so that researchers looking at the data will understand that there are pieces missing.

4.2.1 Maintenance Rate Data

Maintenance rate data collected by the SEL originate from two sources. One source is the effort data supplied on WMEFs completed and submitted by maintainers on the project. The other source of rate data is growth data automatically monitored by the SEL data collection team. The WMEF and maintenance growth data are discussed in the following sections.

4.2.1.1 WEEKLY MAINTENANCE EFFORT FORM (WMEF)

During maintenance, effort data are collected on a WMEF (Figure 4-1). This form is analogous to the PRF and CLPRF (Figures 3-3 and 3-4, respectively) submitted during development. It categorizes the hours spent by a given maintainer on the project along two dimensions. The first is the class of maintenance change (or changes) being worked on. The second is the breakdown of activities performed in implementing a change or changes.

The WMEF is submitted weekly by every member of the maintenance team who performs technical work on the maintenance effort. Recall that this does not include work performed to update requirements or specifications. A WMEF is required from every team member for each week he/she is assigned to the project, even for weeks in which no hours are worked on the project (e.g., vacation or temporary assignment to another project). The "zero-hour" form is the mechanism by which the SEL data collectors ensure that the effort data collected for a given week are complete. Project leaders receive reminder notices for all team members from whom the SEL does not receive a form in a given week. The SEL keeps a list of maintainers assigned to each monitored maintenance project and uses it to generate these reminders. The list is distributed to project leaders to update each month as part of the data collection status report (see Section 3.2.1.3).

Maintenance activity is often performed sporadically; i.e., there may be periods of heavy activity alternating with periods in which little or no maintenance is performed. It is important nonetheless that the zero-hour forms be submitted, because the SEL has no other mechanism for determining the level of maintenance activity on the projects it monitors. If, however, there are personnel who perform maintenance activities on the project infrequently, it is not reasonable for them to receive regular reminder notices. Thus, it is up to the project leader to decide whether individual maintainers should be included on the list of maintainers assigned to the project and, for those who are not

WEEKLY MAINTENANCE EFFORT FORM

Name: _____

Project: _____ Date (Friday): _____

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

Section A – Total Hours Spent on Maintenance

(Includes time spent on all maintenance activities for the project excluding writing specification modifications)

Section B – Hours By Class of Maintenance

(Total of hours in Section B should equal total hours in Section A)

Class	Definition	Hours
Correction	Hours spent on all maintenance associated with a system failure.	
Enhancement	Hours spent on all maintenance associated with modifying the system due to a requirements change. Includes adding, deleting, or modifying system features as a result of a requirements change.	
Adaptation	Hours spent on all maintenance associated with modifying a system to adapt to a change in hardware, system software, or environmental characteristics.	
Other	Other hours spent on the project (related to maintenance) not covered above. Includes management, meetings, etc.	

Section C – Hours By Maintenance Activity

(Total of hours in Section C should equal total hours in Section A)

Activity	Activity Definitions	Hours
Isolation	Hours spent understanding the failure or request for enhancement or adaptation.	
Change Design	Hours spent actually redesigning the system based on an understanding of the necessary change.	
Implementation	Hours spent changing the system to complete the necessary change. This includes changing not only the code, but the associated documentation.	
Unit Test/ System Test	Hours spent testing the changed or added components. Includes hours spent testing the integration of the components.	
Acceptance/ Benchmark Test	Hours spent acceptance testing or benchmark testing the modified system.	
Other	Other hours spent on the project (related to maintenance) not covered above. Includes management, meetings, etc.	

6201G(39):10

NOVEMBER 1991

Figure 4-1. Weekly Maintenance Effort Form

listed, to ensure that they submit WMEFs when they actually do perform maintenance work on the project.

The SEL also expects that the project leader will help to assure the quality of data submitted on WMEFs by periodically scanning the forms submitted by team members to ensure that the hours recorded match those the team member charged to the maintenance cost collector in the organization's timekeeping system and that the classification of hours is appropriate for the types of activities being performed on the project.

Line-by-Line Instructions

Name: Enter the SEL database name of the maintainer completing the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA.

Project: Enter the acronym selected at maintenance startup that uniquely identifies the project being monitored. Check with the project leader if unsure of the correct name.

Date (Friday): Enter the Friday date corresponding to the end of the week for which data are being reported. Data are to be reported on this form for all work performed on the project during the preceding Saturday-through-Friday period.

Section A

Total hours spent on maintenance: Enter the total hours actually worked on maintenance for the project for the current week. This includes any overtime, whether paid or unpaid. It does not include paid hours not charged to the project, such as sick time, holidays, or vacations. Note that this number must equal the sum of the hours recorded under the maintenance classes in Section B, as well as the sum of the hours recorded under the maintenance activities in Section C. If partial hours are recorded, enter them in decimal form to the nearest tenth of an hour. Do not enter fractions. This also applies to all hours entered in Sections B and C.

Section B

Correction: Enter the number of hours during the current week spent working on OSMRs to correct system errors. This includes those that originated from errors in the requirements or specifications.

Enhancement: Enter the number of hours during the current week spent working on OSMRs to modify the system due to a requirements or specifications change. This activity includes adding, deleting, or modifying system capabilities. It does not include correcting errors in the requirements and specifications themselves.

Adaptation: Enter the number of hours during the current week spent working on OSMRs to adapt the system for a change in hardware, system software, or environ-

mental characteristics. This includes changes necessitated by an upgrade to the compiler or operating system, changes to support a new or upgraded I/O device, or changes needed to port the system to a different hardware platform.

Other: Enter the number of hours during the current week that were not spent working on a particular OSMR. This category includes such activities as meetings, management, and training, provided they are related to the maintenance of the system. Configuration management and documentation hours should be recorded here only if they cannot be associated with specific OSMRs.

Section C

Isolation: Enter the number of hours during the current week spent isolating an error or understanding a request for enhancement or adaptation. This includes running tests to isolate the source of an error, analyzing specification modifications, or simply studying system code to become familiar with the areas affected by a change.

Change design: Enter the number of hours during the current week spent redesigning portions of the system based on an understanding of the necessary change. This includes generating new design (diagrams, prologs, PDL) for implementing system enhancements. It also includes time spent inspecting and certifying new and modified design products.

Implementation: Enter the number of hours during the current week spent updating code and documentation or generating new code and documentation to complete the necessary change. This activity includes time spent inspecting and certifying new and modified code.

Unit test/system test: Enter the number of hours during the current week spent testing and integrating the changed or added components and testing them in the context of the end-to-end system. This activity includes designing and executing tests and writing test drivers and program stubs. In the maintenance life cycle, this activity covers testing at both the programmer and integration levels.

Acceptance/benchmark test: Enter the number of hours during the current week spent acceptance testing or benchmark testing. This activity involves verifying that the software meets requirements and performs correctly with existing operational software. In the maintenance life cycle, this activity covers testing at the acceptance level.

Other: Enter the number of hours during the current week spent on any miscellaneous activities involving maintenance not categorized in any of the above activities. This includes management, meetings, training, configuration management, and system build activities. Note that updating documentation falls under the Implementation activity.

Helpful Hint

As with the PRF in development, there is a tendency on Section C of the WMEF to record activities by "phases," rather than truly reflecting the type of work being

performed. For example, if a change has been promoted to the acceptance test level, it is common to see all hours associated with it reported under the Acceptance/Benchmark Test activity in Section C. If some hours were actually spent correcting errors reported during acceptance test on trouble reports, however, those hours should be recorded under the appropriate categories (i.e., Isolation, Change Design, etc.) based on the actual **activities** performed. The Acceptance/Benchmark Test activity should be reserved for actually executing and evaluating tests.

4.2.1.2 GROWTH DATA

The second type of rate data collected during maintenance is growth data. The SEL collects these data automatically by monitoring the configured source code library or libraries. In development, libraries are monitored weekly to provide a profile of source code growth through the life-cycle phases. In maintenance, however, libraries are monitored monthly, since changes are not expected to occur as rapidly in maintenance as they do in development. In addition, the measures taken in maintenance represent a profile of maintenance activity more than they do a profile of system growth, since growth in maintenance is generally a slower, long-term phenomenon characterized by short-term increases and decreases.

Since the OSMP calls for multiple levels of libraries to be used in maintaining an operational system, it should be clarified that the **operational libraries** are those that the SEL monitors. The maintenance project leader must communicate the names and locations of these libraries to SEL data collection personnel. The libraries the SEL is monitoring at any given time are listed on the data collection status report (see Section 3.2.1.3), which is distributed monthly to project leaders for validation and update.

The SEL uses the SPF to record monthly growth data. This is the same form used in development, only the computer resources and services effort portions are not used, and it is completed monthly rather than weekly. (SEL personnel run growth history tools for maintenance projects on the last Friday of each calendar month.) Refer to Section 3.2.1.3 for a complete description of the SPF and a discussion of the growth data collected on it. The "Helpful Hint" included in that section contains library management guidelines that are applicable to both maintenance and development.

4.2.2 Maintenance Event Data

As in development, event data in maintenance are submitted to the SEL sporadically, when given events in the maintenance life cycle occur, as opposed to being submitted on a regular, periodic basis. Two types of event data are collected in maintenance: maintenance changes/errors and messages. Changes data are collected on MCRFs and characterize the OSMRs that are implemented in the operational system. Messages data may be submitted at any time to capture auxiliary information about the maintenance effort or the data being collected on it. The MCRF and maintenance messages data are discussed in the following sections.

4.2.2.1 MAINTENANCE CHANGE REPORT FORM (MCRF)

General Information

For every OSMR implemented by the maintenance team, the SEL receives a corresponding MCRF (Figure 4-2), which provides data that characterize the change. A copy of the OSMR should be attached to the MCRF when it is submitted.

The MCRF is completed by the maintainer responsible for implementing and testing the OSMR. It should be submitted after the change has been tested at the integration level and promoted to the acceptance test library for acceptance/benchmark testing. Although the implementing maintainer may not be responsible for integration testing, he/she should be aware of the progress of the change so that the form may be submitted when the change is promoted for acceptance level testing.

Line-by-line Instructions

Name: Enter the SEL database name of the maintainer completing the form. Usually, the database name consists of a first initial followed by a last name. Questions about database names should be referred to the SEL DBA.

OSMR number: Enter the tracking number of the OSMR that authorized the change being characterized on the MCRF.

Project: Enter the acronym selected at maintenance startup that uniquely identifies the project being monitored. Check with the project leader if unsure of the correct name.

Date: Enter the date on which the form is completed.

Section A

Functional description of change: Explain what change is being made, why the change is necessary, and how the change is being made. Provide sufficient detail so that people other than project staff can understand the change. The description should not be on the variable name level, but should be sufficiently abstract so that the function of the changed code can be determined.

What was the type of modification? Check the **one** option that best classifies the OSMR according to the following definitions:

- *Correction:* A change made to correct an error in the system; usually arises from a system failure reported on some type of trouble report or failure report form; may originate in requirements, specifications, design, code, or documentation.
- *Enhancement:* A change made to improve the functionality or performance of the system; usually originates from a change in requirements or specifications, but **not** a requirements or specifications change that simply corrects an error.

MAINTENANCE CHANGE REPORT FORM

Name: _____ OSMR Number: _____

Project: _____ Date: _____

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

SECTION A: Change Request Information

Functional Description of Change: _____

_____

_____

_____

What was the type of modification?

- ____ Correction
- ____ Enhancement
- ____ Adaptation

What caused the change?

- ____ Requirements/specifications
- ____ Software design
- ____ Code
- ____ Previous change
- ____ Other

SECTION B: Change Implementation Information

Components Added/Changed/Deleted: _____

_____

_____

	< 1hr	1 hr to 1 day	1 day to 1 week	1 week to 1 month	> 1 month
Estimate effort spent isolating/determining the change:					
Estimate effort to design, implement, and test the change:					

Check all changed objects:

- ____ Requirements/Specifications Document
- ____ Design Document
- ____ Code
- ____ System Description
- ____ User's Guide
- ____ Other

If code changed, characterize the change (check most applicable):

- ____ Initialization
- ____ Logic/control structure (e.g., changed flow of control)
- ____ Interface (internal) (module-to-module communication)
- ____ Interface (external) (module to external communication)
- ____ Data (value or structure) (e.g., variable or value changed)
- ____ Computational (e.g., change of math expression)
- ____ Other (none of the above apply)

Estimate the number of lines of code (including comments): _____ added _____ changed _____ deleted

Enter the number of components: _____ added _____ changed _____ deleted

Enter the number of the added components that are: _____ totally new _____ totally reused _____ reused with modifications

6210G(39):11

NOVEMBER 1991

Figure 4-2. Maintenance Change Report Form

- *Adaptation:* A change made to adapt the system to a change in the environment; may originate as a requirements change (e.g., a requirement is added that the system must execute on hardware platform Y as well as on platform X, for which it was originally designed). Such changes include adapting to a new hardware platform, upgrading to run under a new version of the compiler or operating system, and supporting new or upgraded hardware devices.

Cause of change: Check the **one** option that best indicates where the change originated, or to what part of the development life cycle represented in the final product the source of the change can be traced. The options are as follows:

- *Requirements/specifications:* Any change requiring updates to the requirements or specifications that **is not** the direct result of a previous change. Any of the three types of change may originate in requirements or specifications: corrections may derive from correcting errors in the requirements; enhancements may derive from additional requirements to add new functionality or requirements changes to improve performance; and adaptations may derive from new requirements to run on different hardware or support different devices.

- *Software design:* Any change requiring updates to the software design that **does not** originate in requirements or specifications and **is not** the result of a previous change; includes changes to calling sequences, PDL, structure charts, etc. Any of the three types of change may originate in design: corrections may be made to fix design errors; enhancements to improve performance (not specifically called out in a requirements or specifications change) may involve modifications to the design; adaptations to a new version of the compiler or operating system (usually versions of system software and tools are not specified in requirements or specifications) may also involve modifications to the design.

- *Code:* Any change requiring updates to the code that **does not** originate in requirements, specifications, or design, and **is not** the result of a previous change. Any of the three types of change may originate in code: corrections may be made to fix coding errors that result from the incorrect implementation of a correct design; enhancements to improve performance may be made that involve the way in which a given design is implemented; and adaptations to changes in the environment may be made that involve only changes at the code level and leave the design intact.

- *Previous change:* Any change that is the direct result of implementing a previous OSMR (changes made during development are not considered when determining if a change falls into this category). Any of the three types of changes may be the result of a previous change. Changes in this category may involve changes to requirements, specifications, design, and code, but if their

cause can be traced directly to a previous OSMR, they should be classified in this category.

- *Other*: Any change that does not result from a previous change and does not affect requirements, specifications, design, or code. Changes to improve the clarity of stand-alone or inline documentation would fall into this category.

Section B

Components added/changed/deleted: Supply three lists (attaching a separate sheet if necessary) that identify new components that were added to the system (including any that were totally reused from another source), existing components that were modified, and existing components that were deleted from the system in implementing the OSMR.

Estimate the effort spent isolating/determining the change: Put a check mark in the box that indicates the approximate effort spent understanding the change (or finding the cause of the error), locating where the change is to be made, and determining that all effects of the change are accounted for. Note that this effort is to be reported in staff-days, not calendar days. Note also that this does not include effort spent making modifications to requirements or specifications, but begins with the effort spent by the maintainer understanding the system modifications necessitated by such changes.

Estimate the effort to design, implement, and test the change: Put a check mark in the box that indicates the approximate effort spent implementing the change. This includes all effort spent by the maintainer modifying design, code, and documentation, and testing the change at the local level. It also includes as much of the effort spent at higher levels of testing as is possible to obtain. There are two factors that determine the testing level at which effort should no longer be included. One is the level at which the change is delivered to an organization from which the SEL does not collect data. The other is the level at which multiple changes are being integrated and tested simultaneously, such as may happen when several OSMRs are combined to constitute a new release of the software. In this case, effort associated with testing a particular OSMR cannot be reasonably distinguished from that associated with other OSMRs in the release.

Check all changed objects: Put a check mark by all of the objects listed that were modified as a result of implementing the OSMR. The items listed are standard development products and require no clarification. The Other item should be checked if any other documents, software, or procedures directly related to the system were changed. This includes, among other things, JCL, build procedures, interface control documents (ICDs), development tools, and test procedures.

If code changed, characterize the change: Put a check mark by the **one** classification that best describes the majority of the code changes made. The options are as follows:

- *Initialization:* The largest proportion of the code changes involved adding or changing code that initializes data structures at the beginning of a run or upon entry to a subroutine or procedure. This includes modifying DATA statements and BLOCK DATA subprograms.

- *Logic/control structure* (e.g., changed flow of control): The largest proportion of the code changes involved modifying Boolean decision points that control program flow. This includes correcting or changing condition expressions on IF and CASE statements and changing loop entry or exit criteria.

- *Interface (internal)* (module-to-module communication): The largest proportion of the code changes involved modifying the way data move through the system internally. This includes changes in calling sequences (parameter and argument lists), the specification and use of COMMON blocks, and the use of state data.

- *Interface (external)* (module to external communication): The largest proportion of the code changes involved modifying the way the system communicates with the external world. This includes, among other things, changes to the format or access method used with external files, the contents or format of reports, the data presented on display screens, and the mechanism by which the user supplies input to the system.

- *Data (value or structure)* (e.g., variable or value changed): The largest proportion of the code changes involved modifying the specification of and access to data structures. This includes, among other things, changes to variables, variable names, array indexes, dynamic data structures, and the use of pointers.

- *Computational* (e.g., change of math expression): The largest proportion of the code changes involved adding or modifying code that computes mathematical expressions to evaluate or assign the value of a variable.

- *Other* (none of the above apply): Since virtually all code changes fall into one of the preceding categories (even deletions involve removing code that falls into the categories listed), this option should be checked only if the proportions of code changes falling into two or more of the categories are close enough that a single category with the largest proportion is not discernible.

Estimate the number of lines of code (including comments):

NOTE: The lines of code counted for the following three fields are SLOC, which is defined as a count of carriage returns, card images, or file records. In counting SLOC, each line of the system stored in a source code file is counted only one time. Thus, for example, the addition of an INCLUDE statement to a component would be counted as one additional source line of code, rather than counting the size of the file being included. (See also the discussion of line counting in hint 3 of PCSF instructions—Section 3.3.1).

- *Added:* Enter the number of newly created lines of code added to the system. This includes all lines in new components that are added to the system, new segments of code that are added to existing system components, and net increases in segments of code that are modified in existing components.
- *Changed:* Enter the number of existing lines of code that were modified.
- *Deleted:* Enter the number of lines of code deleted from the system. This includes all lines in components that are deleted from the system, entire segments of code that are deleted from existing system components, and net decreases in segments of code that are modified in existing components.

Enter the number of components:

NOTE: The component counts entered in the following three fields must match the number of components in the lists provided in the Components Added/ Changed/Deleted field.

- *Added:* Enter the number of components that were added to the system. This is not limited to newly developed components, since components may have been added by reusing them from other sources. It is also not a net increase in components from before the change to after it.
- *Changed:* Enter the number of existing system components in which source code was added, modified, or deleted.
- *Deleted:* Enter the number of components that were deleted from the system. This may include components that were linked into the system from another source but are no longer linked in as a result of the change (see hint 4). It is not a net decrease in components from before the change to after it.

Enter the number of added components that are:

NOTE: The sum of the component counts entered in the following three fields must equal the total number of components added to the system as recorded in the preceding set of fields.

- *Totally new:* Enter the number of added components that were designed and implemented from scratch.

- *Totally reused:* Enter the number of added components that were reused without modification from other sources. This includes all totally reused components that are copied into the project's operational library, as well as certain classes of reused components that are linked in from external reuse libraries (see hint 4).

- *Reused with modifications:* Enter the number of added components that were reused from other sources but were modified during the implementation of the change.

Helpful Hints

1. Estimating the effort spent implementing a change can be imprecise, especially if there are several levels of testing covered by the estimate. The maintainer may not have information about how much time is spent on a particular change by testers at the integration level or at higher testing levels. One suggestion is to have a key point of contact responsible for coordinating a given level of testing. This individual would be in the best position to estimate the amount of effort spent testing a given change at that particular level. The responsible maintainer should consult these key points of contact for each of the testing levels included and combine the estimates when filling out this part of the form.

2. Characterizing the predominant type of code changes made is really more of a judgment call by the maintainer than a precise measurement. However, the maintainer is strongly urged to think through the types of modifications performed (including thinking about what type of code was deleted) and select a predominant type of change when completing this part of the form, rather than resorting to the Other category to indicate that no single type of change was predominant.

3. Questions often arise about how to determine the number of lines added, changed, and deleted. If, for example, a maintainer is modifying a five-line segment of code and finds it easier to delete the five lines and retype them with the modifications, are those counted as five lines deleted and five added, or simply as five lines changed? The answer is not clear-cut. If the change involved simply changing a variable name in each of the five lines, those five lines would definitely be considered changed lines. If, however, the five lines deleted are replaced by five entirely different lines that implement the same function via an entirely different algorithm, they should be counted as five lines deleted and five lines added. Perhaps the best advice for supplying these counts for modified components (it is obvious for added and deleted components) is to use a comparison tool to produce difference listings between the old versions of the changed components and the new versions. Such tools are available in the FDF mainframe environment, the STL VAX environment, and on PCs, and many will produce summary counts of added, changed, and

deleted lines. It is recommended that such a tool be selected and used consistently for computing these line counts.

4. Another point of clarification concerns defining the "boundaries" of the system with respect to reused software linked in from other sources for the purpose of counting added, changed, and deleted lines and components. As discussed in hint 4 of the PEF instructions (Section 3.1.2), software that is linked in from institutionally maintained tools (such as a graphics display tool or vendor-supplied, language-specific library of mathematics routines), **is not** considered within the system boundaries for purposes of measuring the effects of a change. If, however, the reused software is linked in from a separately maintained application or a generic library of application-specific software (e.g., RSL, MTASS, or MSASS), then it **is** considered inside the system boundaries for purposes of measuring the effects of a change.

4.2.2.2 PROJECT MESSAGES

The second type of event data collected during maintenance is project messages. These are submitted whenever a maintainer or SEL data collector wants to provide additional information to annotate the data being collected. Messages are submitted on a PMF, described in Section 3.2.2.5.

Typical message information that should be submitted during maintenance includes the testing level through which effort data are collected, the testing level through which the effort spent on a given OSMR is being tracked, the organizational responsibilities for maintenance of the system, periods during which data collected should be considered incomplete for one reason or another, any deviations from the OSMP being adopted by the project that would be important to users of the data, and any additional information that describes the nature of the data, their accuracy, or how they should be interpreted.

Although there is no formal closeout process defined for the end of maintenance data monitoring, SEL data collection personnel meet with the maintenance project leader to discuss the data collected and identify any final areas where clarification or annotation is necessary. The results of this final, informal meeting are documented in the database via a PMF completed by a SEL data collector.

APPENDIX—SEL FORMS

The SEL forms appear on the following pages.

CHANGE REPORT FORM

Name:_____ Approved by:_____

Project:_____ Date:_____

Section A – Identification

Describe the change: (What, why, how) _____

Effect: What components are changed?

Prefix	Name	Version

(Attach list if more space is needed)

Effort: What additional components were examined in determining what change was needed?

Location of developer's source files_____

	month	day	year
Need for change determined on:			
Change completed (incorporated into system):			

Check here if change involves Ada components (If so, complete questions on reverse side) ☐

	1 hr/less	1 hr/1 day	1/3 days	>3 days
Effort in person time to isolate the change (or error):				
Effort in person time to implement the change (or correction):				

Section B – All Changes

Type of Change (Check one)

- ☐ Error correction
- ☐ Planned enhancement
- ☐ Implementation of requirements change
- ☐ Improvement of clarity, maintainability, or documentation
- ☐ Improvement of user services
- ☐ Insertion/deletion of debug code
- ☐ Optimization of time/space/ accuracy
- ☐ Adaptation to environment change
- ☐ Other (Describe below)

Effects of Change

Y	N	
☐	☐	Was the change or correction to one and only one component? (Must match Effect in Section A)
☐	☐	Did you look at any other component? (Must match Effort in Section A)
☐	☐	Did you have to be aware of parameters passed explicitly or implicitly (e.g., COMMON blocks) to or from the changed components?

Section C – For Error Corrections Only

Source of Error (Check one)

- ☐ Requirements
- ☐ Functional specifications
- ☐ Design
- ☐ Code
- ☐ Previous change

Class of Error (Check most applicable)*

- ☐ Initialization
- ☐ Logic/control structure (e.g., flow of control incorrect)
- ☐ Interface (internal) (module-to-module communication)
- ☐ Interface (external) (module to external communication)
- ☐ Data (value or structure) (e.g., wrong variable used)
- ☐ Computational (e.g., error in math expression)

*If two are equally applicable, check the one higher on the list.

Characteristics (Check Y or N for all)

Y	N	
☐	☐	Omission error (e.g., something was left out)
☐	☐	Commission error (e.g., something incorrect was included)
☐	☐	Error was created by transcription (clerical)

For Librarian's Use Only

Number:_____

Date:_____

Entered by:_____

Checked by:_____

6201G(13) 09

NOVEMBER 1991

CHANGE REPORT FORM

Ada Project Additional Information

1. Check which Ada feature(s) was involved in this change (Check all that apply)

- ☐ Data typing
- ☐ Subprograms
- ☐ Exceptions
- ☐ Generics
- ☐ Program structure and packaging
- ☐ Tasking
- ☐ System-dependent features
- ☐ Other, please specify _____ (e.g., I/O, Ada statements)

2. For an error involving Ada components:

a. Does the compiler documentation or the language reference manual explain the feature clearly? _____ (Y/N)

b. Which of the following is most true? (Check one)

- ☐ Understood features separately but not interaction
- ☐ Understood features, but did not apply correctly
- ☐ Did not understand features fully
- ☐ Confused feature with feature in another language

c. Which of the following resources provided the information needed to correct the error? (Check all that apply)

- ☐ Class notes
- ☐ Ada reference manual
- ☐ Own project team member
- ☐ Own memory
- ☐ Someone not on team
- ☐ Other

d. Which tools, if any, aided in the detection or correction of this error? (Check all that apply)

- ☐ Compiler
- ☐ Symbolic debugger
- ☐ Language-sensitive editor
- ☐ CMS
- ☐ Source Code Analyzer
- ☐ P&CA (Performance and Coverage Analyzer)
- ☐ DEC test manager
- ☐ Other, specify _____

3. Provide any other information about the interaction of Ada and this change that you feel might aid in evaluating the change and using Ada

6201G(13)-13

NOVEMBER 1991

COMPONENT CHANGE FORM

Name:_____

Project:_____ Date:_____

This form is to be used when components in the project controlled source library are deleted or renamed.

Old Component (Must exist in the database)		Action R – Rename D – Delete	New Component	
Prefix	Name		Prefix	Name

6201G(13) 37

NOVEMBER 1991

COMPONENT ORIGINATION FORM

Identification

Name:_____

Project: _____ Date:_____

Subsystem Prefix: _____

Component Name:_____

Configuration Management Information

Date entered into controlled library (supplied by configuration manager): _____

Library or directory containing developer's source file:_____

Member name:_____

Relative Difficulty of Developing Component

Please indicate your judgment by circling one of the numbers below.

Easy		Medium		Hard
1	2	3	4	5

Origin

If the component was modified or derived from a different project, please indicate the approximate amount of change and from where it was acquired; if it was coded new (from detailed design) indicate NEW.

_____ NEW
_____ Extensively modified (more than 25% of statements changed)
_____ Slightly modified
_____ Old (unchanged)

For Librarian's Use Only

Number: _____
Date:_____
Entered by:_____
Checked by:_____

If not new, what project or library is it from?_____
Component or member name: _____

Type of Component (Check one only)

_____ INCLUDE file (e.g., COMMON)
_____ Control language (e.g., JCL, DCL, CLIST)
_____ ALC (assembler code)
_____ FORTRAN source
_____ Pascal source
_____ C source
_____ NAMELIST or parameter list
_____ Display identification (e.g., GESS, FDAF)
_____ Menu definition or help
_____ Reference data files
_____ BLOCK DATA file
_____ Ada subprogram specification
_____ Ada subprogram body
_____ Ada package specification
_____ Ada package body
_____ Ada task body
_____ Ada generic instantiation
_____ Ada generic specification
_____ Ada generic body
_____ Other

Purpose of Executable Component

For executable code, please identify the major purpose or purposes of this component. (Check all that apply).

_____ I/O processing
_____ Algorithmic/computational
_____ Data transfer
_____ Logic/decision
_____ Control module
_____ Interface to operating system
_____ Process abstraction
_____ Data abstraction

6201G(13)-38

NOVEMBER 1991

DEVELOPMENT STATUS FORM

Name: _____

Project: _____ Date: _____

Please complete the section(s) that is appropriate for the current status of the project.

Design Status	
Planned total number of components to be designed (New, modified, and reused)	
Number of components designed (Prolog and PDL have been completed)	

Code Status	
Planned total number of components to be coded (New, modified, and reused)	
Number of components completed (Added to controlled library)	

Testing Status	System Test	Acceptance Test
Total number of separate tests planned		
Number of tests executed at least one time		
Number of tests passed		

Discrepancy Tracking Status (from beginning of system testing)	
Total number of discrepancies reported	
Total number of discrepancies resolved	

Specification Modification Status (from beginning of requirements analysis)	
Total number of specification modifications received	
Total number of specification modifications completed (implemented)	

Requirements Questions Status (from beginning of requirements analysis)	
Total number of questions submitted to analysts	
Total number of questions answered by analysts	

Check here if there are no changes ☐

For Librarian's Use Only
Number: _____
Date: _____
Entered by: _____
Checked by: _____

6201G(39) 8

NOVEMBER 1991

MAINTENANCE CHANGE REPORT FORM

Name: _____ OSMR Number: _____

Project: _____ Date: _____

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

SECTION A: Change Request Information

Functional Description of Change: _____

_____

_____

_____

What was the type of modification?

- ___ Correction
- ___ Enhancement
- ___ Adaptation

What caused the change?

- ___ Requirements/specifications
- ___ Software design
- ___ Code
- ___ Previous change
- ___ Other

SECTION B: Change Implementation Information

Components Added/Changed/Deleted: _____

_____

_____

	< 1hr	1 hr to 1 day	1 day to 1 week	1 week to 1 month	> 1 month
Estimate effort spent isolating/determining the change:					
Estimate effort to design, implement, and test the change:					

Check all changed objects:

- ___ Requirements/Specifications Document
- ___ Design Document
- ___ Code
- ___ System Description
- ___ User's Guide
- ___ Other

If code changed, characterize the change (check most applicable):

- ___ Initialization
- ___ Logic/control structure (e.g., changed flow of control)
- ___ Interface (internal) (module-to-module communication)
- ___ Interface (external) (module to external communication)
- ___ Data (value or structure) (e.g., variable or value changed)
- ___ Computational (e.g., change of math expression)
- ___ Other (none of the above apply)

Estimate the number of lines of code (including comments): _____ added _____ changed _____ deleted

Enter the number of components: _____ added _____ changed _____ deleted

Enter the number of the added components that are: _____ totally new _____ totally reused _____ reused with modifications

6210G(39)-11

NOVEMBER 1991

Personnel Resources Form

Name:_____

Project:_____ Date (Friday): _____

SECTION A: Total Hours Spent on Project for the Week:_____

SECTION B: Hours By Activity (Total of hours in Section B should equal total hours in Section A)

Activity	Activity Definitions	Hours
Predesign	Understanding the concepts of the system. Any work prior to the actual design (such as requirements analysis).	
Create Design	Development of the system, subsystem, or components design. Includes development of PDL, design diagrams, etc.	
Read/Review Design	Hours spent reading or reviewing design. Includes design meetings, formal and informal reviews, or walkthroughs.	
Write Code	Actually coding system components. Includes both desk and terminal code development.	
Read/Review Code	Code reading for any purpose other than isolation of errors.	
Test Code Units	Testing individual components of the system. Includes writing test drivers.	
Debugging	Hours spent finding a known error in the system and developing a solution. Includes generation and execution of tests associated with finding the error.	
Integration Test	Writing and executing tests that integrate system components, including system tests.	
Acceptance Test	Running/supporting acceptance testing.	
Other	Other hours spent on the project not covered above. Includes management, meetings, training hours, notebooks, system descriptions, user's guides, etc.	

SECTION C: Effort On Specific Activities (Need not add to A)
(Some hours may be counted in more than one area; view each activity separately)

Rework: Estimate of total hours spent that were caused by unplanned changes or errors. Includes effort caused by unplanned changes to specifications, erroneous or changed design, errors or unplanned changes to code, changes to documents. (This includes all hours spent debugging.) ☐

Enhancing/Refining/Optimizing: Estimate of total hours spent improving the efficiency or clarity of design, or code, or documentation. These are not caused by required changes or errors in the system. ☐

Documenting : Hours spent on any documentation of the system. Includes development of design documents, prologs, in-line commentary, test plans, system descriptions, user's guides, or any other system documentation. ☐

Reuse: Hours spent in an effort to reuse components of the system. Includes effort in looking at other system(s) design, code, or documentation. Count total hours in searching, applying, and testing. ☐

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

5150G(21):38

NOVEMBER 1991

Personnel Resources Form

(CLEANROOM VERSION)

Name:_____

Project:_____ Date (Friday): _____

SECTION A: Total Hours Spent on Project for the Week:_____

SECTION B: Hours By Activity (Total of hours in Section B should equal total hours in Section A)

Activity	Activity Definitions	Hours
Predesign	Understanding the concepts of the system. Any work prior to the actual design (such as requirements analysis).	
Pretest	Developing a test plan and building the test environment. Includes generating test cases, generating JCL, compiling components, building libraries, and defining inputs and probabilities.	
Create Design	Development of the system, subsystem, or components design. Includes box structure decomposition, stepwise refinement, development of PDL, design diagrams, etc.	
Verify/Review Design	Includes design meetings, formal and informal reviews, and walkthrougns.	
Write Code	Actually coding system components. Includes both desk and terminal code development.	
Read/Review Code	Code reading for any purpose other than isolation of errors. Includes verifying and reviewing code for correctness.	
Independent Test	Executing and evaluating tests of system components.	
Response to SFR	Isolating a tester-reported problem and developing a solution. Includes writing and reviewing design or code to isolate and correct a tester-reported problem.	
Acceptance Test	Running/supporting acceptance testing.	
Other	Other hours spent on the project not covered above. Includes management, meetings, training hours, notebooks, system descriptions, user's guides, etc.	

SECTION C: Effort On Specific Activities

Methodology Understanding/Discussion: Estimate the total hours spent learning, discussing, reviewing or attempting to understand cleanroom-related methods and techniques. Includes all time spent in training. ☐

For Librarian's Use Only
Number: _____
Date: _____
Entered by: _____
Checked by: _____

6201G(13):24

NOVEMBER 1991

PROJECT COMPLETION STATISTICS FORM

Name: _____

Project: _____ Date: _____

Phase Dates (Saturdays)	
Phase	Start Date
Requirements Definition	
Design	
Implementation	
System Test	
Acceptance Test	
Cleanup	
Maintenance	
Project End	

Staff Resource Statistics	
Technical and Management Hours	
Services Hours	

Computer Resource Statistics		
Computer	CPU hours	No. of runs

Project Size Statistics			
General Parameters		Source Lines of Code	
Number of subsystems		Total	
Number of components		New	
Number of changes		Slightly Modified	
Pages of documentation		Extensively Modified	
		Old	
		Comments	

Executable Modules		Executable Statements		Statements	
Total		Total		Total	
New		New		New	
Slightly Modified		Slightly Modified		Slightly Modified	
Extensively Modified		Extensively Modified		Extensively Modified	
Old		Old		Old	

Note: All of the values on this form are to be actual values at the completion of the project. The values entered by hand by SEL personnel reflect the data collected by the SEL during the course of the project. Update these according to project records and supply values for all blank fields.

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

6201(5)(69) 11

NOVEMBER 1991

PROJECT ESTIMATES FORM

Name: _____

Project: _____ Date: _____

Phase Dates (Saturdays)	
Phase	Start Date
Requirements Definition	
Design	
Implementation	
System Test	
Acceptance Test	
Cleanup	
Project End	

Staff Resource Estimates	
Programmer Hours	
Management Hours	
Services Hours	

Project Size Estimates	
Number of subsystems	
Number of components	
Source Lines of Code	
Total	
New	
Modified	
Old	

Note: All of the values on this form are to be estimates of projected values at completion of the project. This form should be submitted with updated estimates every 6 to 8 weeks during the course of the project.

For Librarian's Use Only
Number: _____
Date: _____
Entered by: _____
Checked by: _____

6201G(13):16

NOVEMBER 1991

PROJECT MESSAGES FORM

Name:_____

Project: _____ Date:_____

Messages:

6201G(13) 35

NOVEMBER 1991

PROJECT STARTUP FORM

Name: _____

Project: _____ **Date:** _____

PLEASE PROVIDE ALL AVAILABLE INFORMATION

Project Full Name: _____

Project Type: _____

Contacts: _____

Language: _____

Computer System: _____

Account: _____

Task Number: _____

Forms To Be Collected: (Circle forms that apply)

PEF PRF CLPRF DSF SPF SIF COF CCF CRF SEF PCSF WMEF MCRF

General Notes: _____

Personnel Names (indicate with * if not in database):

_____ _____

_____ _____

_____ _____

_____ _____

_____ _____

6201G(13) 36

NOVEMBER 1991

SERVICES/PRODUCTS FORM

Project:_____

Date (Friday):_____

COMPUTER RESOURCES

Computer	CPU Hours	No. of Runs

GROWTH HISTORY

Components	
Changes	
Lines of Code	

SERVICES EFFORT

Service	Hours
Tech Pubs	
Secretary	
Proj Mgmt	
Other	

For Librarian's Use Only
Number:_____
Date:_____
Entered by:_____
Checked by:_____

6201G(13)-08

NOVEMBER 1991

SUBJECTIVE EVALUATION FORM

Name: _____

Project: _____ Date: _____

Indicate response by circling the corresponding numeric ranking.

I. PROBLEM CHARACTERISTICS

1. Assess the intrinsic difficulty or complexity of the problem that was addressed by the software development.

1	2	3	4	5
Easy		Average		Difficult

2. How tight were schedule constraints on project?

1	2	3	4	5
Loose		Average		Tight

3. How stable were requirements over development period?

1	2	3	4	5
Loose		Average		High

4. Assess the overall quality of the requirements specification documents, including their clarity, accuracy, consistency, and completeness.

1	2	3	4	5
Low		Average		High

5. How extensive were documentation requirements?

1	2	3	4	5
Low		Average		High

6. How rigorous were formal review requirements?

1	2	3	4	5
Low		Average		High

II. PERSONNEL CHARACTERISTICS: TECHNICAL STAFF

7. Assess overall quality and ability of development team.

1	2	3	4	5
Low		Average		High

8. How would you characterize the development team's experience and familiarity with the application area of the project?

1	2	3	4	5
Low		Average		High

9. Assess the development team's experience and familiarity with the development environment (hardware and support software).

1	2	3	4	5
Low		Average		High

10. How stable was the composition of the development team over the duration of the project?

1	2	3	4	5
Loose		Average		High

FOR LIBRARIAN'S USE ONLY

Number: _____ Entered by: _____

Date: _____ Checked by: _____

6201G(13) 29

NOVEMBER 1991

SUBJECTIVE EVALUATION FORM

III. PERSONNEL CHARACTERISTICS: TECHNICAL MANAGEMENT

11. Assess the overall performance of project management.

1	2	3	4	5
Low		Average		High

12. Assess project management's experience and familiarity with the application.

1	2	3	4	5
Low		Average		High

13. How stable was project management during the project?

1	2	3	4	5
Low		Average		High

14. What degree of disciplined project planning was used?

1	2	3	4	5
Low		Average		High

15. To what degree were project plans followed?

1	2	3	4	5
Low		Average		High

IV. PROCESS CHARACTERISTICS

16. To what extent did the development team use modern programming practices (PDL, top-down development, structured programming, and code reading)?

1	2	3	4	5
Low		Average		High

17. To what extent did the development team use well-defined or disciplined procedures to record specification modifications, requirements questions and answers, and interface agreements?

1	2	3	4	5
Low		Average		High

18. To what extent did the development team use a well-defined or disciplined requirements analysis methodology?

1	2	3	4	5
Low		Average		High

19. To what extent did the development team use a well-defined or disciplined design methodology?

1	2	3	4	5
Low		Average		High

20. To what extent did the development team use a well-defined or disciplined testing methodology?

1	2	3	4	5
Low		Average		High

IV. PROCESS CHARACTERISTICS

21. What software tools were used by the development team? Check all that apply from the list that follows and identify any other tools that were used but are not listed.

- ☐ Compiler
- ☐ Linker
- ☐ Editor
- ☐ Graphic display builder
- ☐ Requirements language processor
- ☐ Structured analysis support tool
- ☐ PDL processor
- ☐ ISPF
- ☐ SAP
- ☐ CAT
- ☐ PANVALET
- ☐ Test coverage tool
- ☐ Interface checker (RXVP80, etc.)
- ☐ Language-sensitive editor
- ☐ Symbolic debugger
- ☐ Configuration Management Tool (CMS, etc.)
- ☐ Others (identify by name and function)

22. To what extent did the development team prepare and follow test plans?

1	2	3	4	5
Low		Average		High

6201G(13) 30

SUBJECTIVE EVALUATION FORM

IV. PROCESS CHARACTERISTICS (CONT'D)

23. To what extent did the development team use well-defined and disciplined quality assurance procedures (reviews, inspections, and walkthroughs)?

1	2	3	4	5
Low		Average		High

24. To what extent did development team use well-defined or disciplined configuration management procedures?

1	2	3	4	5
Low		Average		High

V. ENVIRONMENT CHARACTERISTICS

25. How would you characterize the development team's degree of access to the development system?

1	2	3	4	5
Low		Average		High

26. What was the ratio of programmers to terminals?

1	2	3	4	5
8:1	4:1	2:1	1:1	1:2

27. To what degree was the development team constrained by the size of main memory or direct-access storage available on the development system?

1	2	3	4	5
Low		Average		High

28. Assess the system response time: were the turnaround times experienced by the team satisfactory in light of the size and nature of the jobs?

1	2	3	4	5
Poor		Average		Very Good

29. How stable was the hardware and system support software (including language processors) during the project?

1	2	3	4	5
Low		Average		High

30. Assess the effectiveness of the software tools.

1	2	3	4	5
Low		Average		High

VI. PRODUCT CHARACTERISTICS

31. To what degree does the delivered software provide the capabilities specified in the requirements?

1	2	3	4	5
Low		Average		High

32. Assess the quality of the delivered software product.

1	2	3	4	5
Low		Average		High

33. Assess the quality of the design that is present in the software product.

1	2	3	4	5
Low		Average		High

34. Assess the quality and completeness of the delivered system documentation.

1	2	3	4	5
Low		Average		High

35. To what degree were software products delivered on time?

1	2	3	4	5
Low		Average		High

36. Assess smoothness or relative ease of acceptance testing.

1	2	3	4	5
Low		Average		High

6201G(13)-31

SUBSYSTEM INFORMATION FORM

Name:_____

Project: _____ Date: _____

Add New Subsystems

Subsystem Prefix	Subsystem Name	Subsystem Function

Change Existing Subsystems

Old Subsystem Prefix (Must exist in the database)	Action (R - Rename, D - Delete)	New Subsystem Prefix (Must not exist in the database)

This form is to be completed by the time of the Preliminary Design Review (PDR). An update must be submitted each time a new subsystem is defined thereafter. This form is also to be used when a subsystem is renamed or deleted.

Subsystem Prefix: A prefix of 2 to 5 characters used to identify the subsystem when naming components

Subsystem Name: A descriptive name of up to 40 characters

Subsystem Function: Enter the most appropriate function code from the list of functions below:

USERINT:	User Interface
DPDC:	Data Processing/Data Conversion
REALTIME:	Real-time Control
MATHCOMP:	Mathematical/Computational
GRAPH:	Graphics and Special Device Support
CPEXEC:	Control Processing/Executive
SYSSERV:	System Services

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

NOVEMBER 1991

6201G(13):39

WEEKLY MAINTENANCE EFFORT FORM

Name: _____

Project: _____ Date (Friday): _____

For Librarian's Use Only

Number: _____

Date: _____

Entered by: _____

Checked by: _____

Section A – Total Hours Spent on Maintenance

(Includes time spent on all maintenance activities for the project excluding writing specification modifications)

Section B – Hours By Class of Maintenance

(Total of hours in Section B should equal total hours in Section A)

Class	Definition	Hours
Correction	Hours spent on all maintenance associated with a system failure.	
Enhancement	Hours spent on all maintenance associated with modifying the system due to a requirements change. Includes adding, deleting, or modifying system features as a result of a requirements change.	
Adaptation	Hours spent on all maintenance associated with modifying a system to adapt to a change in hardware, system software, or environmental characteristics.	
Other	Other hours spent on the project (related to maintenance) not covered above. Includes management, meetings, etc.	

Section C – Hours By Maintenance Activity

(Total of hours in Section C should equal total hours in Section A)

Activity	Activity Definitions	Hours
Isolation	Hours spent understanding the failure or request for enhancement or adaptation.	
Change Design	Hours spent actually redesigning the system based on an understanding of the necessary change.	
Implementation	Hours spent changing the system to complete the necessary change. This includes changing not only the code, but the associated documentation.	
Unit Test/ System Test	Hours spent testing the changed or added components. Includes hours spent testing the integration of the components.	
Acceptance/ Benchmark Test	Hours spent acceptance testing or benchmark testing the modified system.	
Other	Other hours spent on the project (related to maintenance) not covered above. Includes management, meetings, etc.	

6201G(39)-10

NOVEMBER 1991

GLOSSARY

AGSS	Attitude Ground Support System
ATR	assistant technical representative
CCF	Component Change Form
CDR	Critical Design Review
CLPRF	Cleanroom Personnel Resources Form
CM	configuration management
CMS	Code Management System
COF	Component Origination Form
CPU	central processing unit
CRF	Change Report Form
DBA	Database Administrator
DEC	Digital Equipment Corporation
DSF	Development Status Form
FDAF	Flight Dynamics Application Framework
FDD	Flight Dynamics Division
FDF	Flight Dynamics Facility
GESS	Graphics Executive Support System
GSFC	Goddard Space Flight Center
ICD	interface control document
ISPF	Interactive System Productivity Facility
JCL	job control language
MCRF	Maintenance Change Report Form
MSASS	Multimission Spin-Axis Stabilized Spacecraft
MTASS	Multimission Three-Axis Stabilized Spacecraft
NASA	National Aeronautics and Space Administration
OSM	Operational Software Modification
OSMP	Operational Software Modification Procedures

OSMR	Operational Software Modification Report
PC	personal computer
PCSF	Project Completion Statistics Form
PDL	program design language
PDR	Preliminary Design Review
PEF	Project Estimates Form
PMF	Project Messages Form
PR	problem report
PRF	Personnel Resources Form
PSF	Project Startup Form
QA	quality assurance
RDBMS	Relational Database Management System
SAP	FORTRAN Static Source Code Analyzer Program
SDE	Software Development Environment
SEF	Subjective Evaluation Form
SEL	Software Engineering Laboratory
SFR	software failure report
SIF	Subsystem Information Form
SLOC	source lines of code
SPF	Services/Products Form
STL	Systems Technology Laboratory
STR	software trouble report
WMEF	Weekly Maintenance Effort Form

REFERENCES

1. Software Engineering Laboratory, SEL-81-101, *Guide to Data Collection*, V. Church et al., August 1982

2. --, SEL-84-101, *Manager's Handbook for Software Development (Revision 1)*, L. Landis et al., November 1990

3. --, SEL-89-101, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 1)*, M. So et al., February 1990

4. --, SEL-90-001, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) User's Guide*, M. Buhler et al., March 1990

5. Goddard Space Flight Center, FDD/552-90/008, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) System Description (Revision 1)*, M. So et al., prepared by Computer Sciences Corporation, April 1990

6. Software Engineering Laboratory, SEL-91-004, *Cleanroom Process Model*, S. Green, November 1991

7. --, SEL-81-205, *Recommended Approach to Software Development*, F. E. McGarry et al., April 1983

8. Goddard Space Flight Center, FDD/552-90/053, *Multimission Three-Axis Stabilized Spacecraft (MTASS) Flight Dynamics Support System (FDSS) Attitude Determination System (ADS) User's Guide*, R. Coon et al., prepared by Computer Sciences Corporation, September 1990

9. --, 552-FDD-91/019, *Multimission Spin-Axis Stabilized Spacecraft (MSASS) Flight Dynamics Support System (FDSS) User's Guide*, C. Crognale et al., prepared by Computer Sciences Corporation, October 1991 (Draft)

10. --, 552-FDD-91/048, *Reusable Software Library (RSL) User's Reference, Revision 1*, M. Woolsey et al., prepared by Computer Sciences Corporation, July 1991

11. Software Engineering Laboratory, SEL-86-003, *Flight Dynamics System Software Development Environment (FDS/SDE) Tutorial*, J. Buell et al., July 1986

12. Computer Sciences Corporation, CSC/SD-75/6057UD1, *Graphic Executive Support System (GESS) User's Guide, Update 1*, D. Green, September 1989

13. Goddard Space Flight Center, 552-FDD-89/011UD1, *Flight Dynamics Application Framework (FDAF) User's Guide, Revision 1, Update 2*, D. Green, prepared by Computer Sciences Corporation, May 1991

14. International Business Machines Corporation, *Interactive System Productivity Facility (ISPF) and ISPF/Program Development Facility (ISPF/PDF) Version 3 Release 2 General Information*, GC34-4250, March 1990

15. Software Engineering Laboratory, SEL-78-302, *FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3)*, W. Decker and W. Taylor, July 1986

16. Goddard Space Flight Center, 553-FDD-91/023, *Flight Dynamics Facility (FDF), Operational Software Modification Procedures (OSMP) Revision 1*, R. Jenkins et al., prepared by Computer Sciences Corporation, July 1991 (Draft)

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, *Proceedings From the First Summer Software Engineering Workshop*, August 1976

SEL-77-002, *Proceedings From the Second Summer Software Engineering Workshop*, September 1977

SEL-77-004, *A Demonstration of AXES for NAVPAK*, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, *GSFC NAVPAK Design Specifications Languages Study*, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-005, *Proceedings From the Third Summer Software Engineering Workshop*, September 1978

SEL-78-006, *GSFC Software Engineering Research Requirements Analysis Study*, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, *Applicability of the Rayleigh Curve to the SEL Environment*, T. E. Mapp, December 1978

SEL-78-302, *FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3)*, W. J. Decker, W. A. Taylor, et al., July 1986

SEL-79-002, *The Software Engineering Laboratory: Relationship Equations*, K. Freburger and V. R. Basili, May 1979

SEL-79-003, *Common Software Module Repository (CSMR) System Description and User's Guide*, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, *Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment*, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

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

10000229
0328/1300

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

SEL-80-003, *Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study*, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-005, *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-80-008, *Tutorial on Models and Metrics for Software Management and Engineering*, V. R. Basili, 1980

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

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

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

SEL-81-012, *The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems*, G. O. Picasso, December 1981

SEL-81-013, *Proceedings of the Sixth Annual Software Engineering Workshop*, December 1981

SEL-81-014, *Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL)*, A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

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

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

SEL-81-107, *Software Engineering Laboratory (SEL) Compendium of Tools (Revision 1)*, W. J. Decker, W. A. Taylor, E. J. Smith, et al., February 1982

SEL-81-110, *Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics*, G. Page, F. E. McGarry, and D. N. Card, June 1985

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

SEL-82-001, *Evaluation of Management Measures of Software Development*, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-004, *Collected Software Engineering Papers: Volume 1*, July 1982

SEL-82-007, *Proceedings of the Seventh Annual Software Engineering Workshop*, December 1982

SEL-82-008, *Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory*, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, *FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1)*, W. A. Taylor and W. J. Decker, April 1985

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

SEL-82-1006, *Annotated Bibliography of Software Engineering Laboratory Literature*, L. Morusiewicz and J. Valett, November 1991

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

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

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

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

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

SEL-83-106, *Monitoring Software Development Through Dynamic Variables (Revision 1)*, C. W. Doerflinger, November 1989

SEL-84-003, *Investigation of Specification Measures for the Software Engineering Laboratory (SEL)*, W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, *Proceedings of the Ninth Annual Software Engineering Workshop*, November 1984

SEL-84-101, *Manager's Handbook for Software Development (Revision 1)*, L. Landis, F. E. McGarry, S. Waligora, et al., November 1990

SEL-85-001, *A Comparison of Software Verification Techniques*, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

10000229
0328/1300

SEL-85-002, *Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team*, R. Murphy and M. Stark, October 1985

SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985

SEL-85-004, *Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics*, R. W. Selby, Jr., and V. R. Basili, May 1985

SEL-85-005, *Software Verification and Testing*, D. N. Card, E. Edwards, F. McGarry, and C. Antle, December 1985

SEL-85-006, *Proceedings of the Tenth Annual Software Engineering Workshop*, December 1985

SEL-86-001, *Programmer's Handbook for Flight Dynamics Software Development*, R. Wood and E. Edwards, March 1986

SEL-86-002, *General Object-Oriented Software Development*, E. Seidewitz and M. Stark, August 1986

SEL-86-003, *Flight Dynamics System Software Development Environment (FDS/SDE) Tutorial*, J. Buell and P. Myers, July 1986

SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986

SEL-86-005, *Measuring Software Design*, D. N. Card et al., November 1986

SEL-86-006, *Proceedings of the Eleventh Annual Software Engineering Workshop*, December 1986

SEL-87-001, *Product Assurance Policies and Procedures for Flight Dynamics Software Development*, S. Perry et al., March 1987

SEL-87-002, *Ada® Style Guide (Version 1.1)*, E. Seidewitz et al., May 1987

SEL-87-003, *Guidelines for Applying the Composite Specification Model (CSM)*, W. W. Agresti, June 1987

SEL-87-004, *Assessing the Ada® Design Process and Its Implications: A Case Study*, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987

SEL-87-010, *Proceedings of the Twelfth Annual Software Engineering Workshop*, December 1987

SEL-88-001, *System Testing of a Production Ada Project: The GRODY Study*, J. Seigle, L. Esker, and Y. Shi, November 1988

SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988

10000229
0328/1300

SEL-88-003, *Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis*, K. Quimby and L. Esker, December 1988

SEL-88-004, *Proceedings of the Thirteenth Annual Software Engineering Workshop*, November 1988

SEL-88-005, *Proceedings of the First NASA Ada User's Symposium*, December 1988

SEL-89-002, *Implementation of a Production Ada Project: The GRODY Study*, S. Godfrey and C. Brophy, September 1989

SEL-89-003, *Software Management Environment (SME) Concepts and Architecture*, W. Decker and J. Valett, August 1989

SEL-89-004, *Evolution of Ada Technology in the Flight Dynamics Area: Implementation/ Testing Phase Analysis*, K. Quimby, L. Esker, L. Smith, M. Stark, and F. McGarry, November 1989

SEL-89-005, *Lessons Learned in the Transition to Ada From FORTRAN at NASA/ Goddard*, C. Brophy, November 1989

SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989

SEL-89-007, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, November 1989

SEL-89-008, *Proceedings of the Second NASA Ada Users' Symposium*, November 1989

SEL-89-101, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 1)*, M. So, G. Heller, S. Steinberg, K. Pumphrey, and D. Spiegel, February 1990

SEL-90-001, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) User's Guide*, M. Buhler, K. Pumphrey, and D. Spiegel, March 1990

SEL-90-002, *The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis*, S. Green et al., March 1990

SEL-90-003, *A Study of the Portability of an Ada System in the Software Engineering Laboratory (SEL)*, L. O. Jun and S. R. Valett, June 1990

SEL-90-004, *Gamma Ray Observatory Dynamics Simulator in Ada (GRODY) Experiment Summary*, T. McDermott and M. Stark, September 1990

SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990

SEL-90-006, *Proceedings of the Fifteenth Annual Software Engineering Workshop*, November 1990

SEL-91-001, *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules*, W. Decker, R. Hendrick, and J. Valett, February 1991

SEL-91-003, *Software Engineering Laboratory (SEL) Ada Performance Study Report*, E. W. Booth and M. E. Stark, July 1991

SEL-91-004, *Software Engineering Laboratory (SEL) Cleanroom Process Model*, S. Green, November 1991

SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991

SEL-91-006, *Proceedings of the Sixteenth Annual Software Engineering Workshop*, December 1991

SEL-91-102, *Software Engineering Laboratory (SEL) Data and Information Policy (Revision 1)*, F. McGarry, August 1991

SEL-92-001, *Software Management Environment (SME) Installation Guide*, D. Kistler, January 1992

SEL-92-002, *Data Collection Procedures for the Software Engineering Laboratory (SEL) Database*, G. Heller, March 1992

SEL-RELATED LITERATURE

[4]Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

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

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

[8]Bailey, J. W., and V. R. Basili, "Software Reclamation: Improving Post-Development Reusability," *Proceedings of the Eighth Annual National Conference on Ada Technology*, March 1990

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

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

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," *Proceedings of the First Pan-Pacific Computer Conference*, September 1985

[7]Basili, V. R., *Maintenance = Reuse-Oriented Software Development*, University of Maryland, Technical Report TR-2244, May 1989

[7]Basili, V. R., *Software Development: A Paradigm for the Future*, University of Maryland, Technical Report TR-2263, June 1989

[8]Basili, V. R., "Viewing Maintenance of Reuse-Oriented Software Development," *IEEE Software*, January 1990

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

[9]Basili, V. R., and G. Caldiera, *A Reference Architecture for the Component Factory*, University of Maryland, Technical Report TR-2607, March 1991

[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

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," *Proceedings of the International Computer Software and Applications Conference*, October 1985

[4]Basili, V. R., and D. Patnaik, *A Study on Fault Prediction and Reliability Assessment in the SEL Environment*, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM*, January 1984, vol. 27, no. 1

[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

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P—A Prototype Expert System for Software Engineering Management," *Proceedings of the IEEE/MITRE Expert Systems in Government Symposium*, October 1985

Basili, V. R., and J. Ramsey, *Structural Coverage of Functional Testing*, University of Maryland, Technical Report TR-1442, September 1984

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*. New York: IEEE Computer Society Press, 1979

[5]Basili, V. R., and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," *Proceedings of the 9th International Conference on Software Engineering*, March 1987

[5]Basili, V. R., and H. D. Rombach, "T A M E: Tailoring an Ada Measurement Environment," *Proceedings of the Joint Ada Conference*, March 1987

[5]Basili, V. R., and H. D. Rombach, "T A M E: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

[6]Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, June 1988

[7]Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment*, University of Maryland, Technical Report TR-2158, December 1988

[8]Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: Model-Based Reuse Characterization Schemes*, University of Maryland, Technical Report TR-2446, April 1990

[9]Basili, V. R., and H. D. Rombach, *Support for Comprehensive Reuse*, University of Maryland, Technical Report TR-2606, February 1991

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environment's Characteristic Software Metric Set," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

Basili, V. R., and R. W. Selby, Jr., *Comparing the Effectiveness of Software Testing Strategies*, University of Maryland, Technical Report TR-1501, May 1985

[3]Basili, V. R., and R. W. Selby, Jr., "Four Applications of a Software Data Collection and Analysis Methodology," *Proceedings of the NATO Advanced Study Institute*, August 1985

[5]Basili, V. R., and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering*, December 1987

[9]Basili, V. R., and R. W. Selby, "Paradigms for Experimentation and Empirical Studies in Software Engineering," *Reliability Engineering and System Safety*, January 1991

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, July 1986

[2]Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," *IEEE Transactions on Software Engineering*, November 1983

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

[3]Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, November 1984

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

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

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

[1]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: IEEE Computer Society Press, 1978

[9]Booth, E. W., and M. E. Stark, "Designing Configurable Software: COMPASS Implementation Concepts," *Proceedings of Tri-Ada 1991*, October 1991

[9]Briand, L. C., V. R. Basili, and W. M. Thomas, *A Pattern Recognition Approach for Software Engineering Data Analysis*, University of Maryland, Technical Report TR-2672, May 1991

[5]Brophy, C. E., W. W. Agresti, and V. R. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," *Proceedings of the Joint Ada Conference*, March 1987

[6]Brophy, C. E., S. Godfrey, W. W. Agresti, and V. R. Basili, "Lessons Learned in the Implementation Phase of a Large Ada Project," *Proceedings of the Washington Ada Technical Conference*, March 1988

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

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

[3]Card, D. N., "A Software Technology Evaluation Program," *Annais do XVIII Congresso Nacional de Informatica*, October 1985

[5]Card, D. N., and W. W. Agresti, "Resolving the Software Science Anomaly," *The Journal of Systems and Software*, 1987

[6]Card, D. N., and W. W. Agresti, "Measuring Software Design Complexity," *The Journal of Systems and Software*, June 1988

[4]Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," *IEEE Transactions on Software Engineering*, February 1986

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

Card, D. N., Q. L. Jordan, and V. E. Church, "Characteristics of FORTRAN Modules," Computer Sciences Corporation, Technical Memorandum, June 1984

[5]Card, D. N., F. E. McGarry, and G. T. Page, "Evaluating Software Engineering Technologies," *IEEE Transactions on Software Engineering*, July 1987

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[1]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: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," *ACM Software Engineering Notes*, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," *Proceedings of the Seventh International Computer Software and Applications Conference*. New York: IEEE Computer Society Press, 1983

Doubleday, D., *ASAP: An Ada Static Source Code Analyzer Program*, University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

[6]Godfrey, S., and C. Brophy, "Experiences in the Implementation of a Large Ada Project," *Proceedings of the 1988 Washington Ada Symposium*, June 1988

Hamilton, M., and S. Zeldin, *A Demonstration of AXES for NAVPAK*, Higher Order Software, Inc., TR-9, September 1977 (also designated SEL-77-005)

[5]Jeffery, D. R., and V. Basili, *Characterizing Resource Data: A Model for Logical Association of Software Data*, University of Maryland, Technical Report TR-1848, May 1987

[6]Jeffery, D. R., and V. R. Basili, "Validating the TAME Resource Data Model," *Proceedings of the Tenth International Conference on Software Engineering*, April 1988

[5]Mark, L., and H. D. Rombach, *A Meta Information Base for Software Engineering*, University of Maryland, Technical Report TR-1765, July 1987

[6]Mark, L., and H. D. Rombach, "Generating Customized Software Engineering Information Bases From Software Process and Product Specifications," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[5]McGarry, F. E., and W. W. Agresti, "Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[7]McGarry, F., L. Esker, and K. Quimby, "Evolution of Ada Technology in a Production Software Environment," *Proceedings of the Sixth Washington Ada Symposium (WADAS)*, June 1989

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," *Proceedings of the Hawaiian International Conference on System Sciences*, January 1985

National Aeronautics and Space Administration (NASA), *NASA Software Research Technology Workshop* (Proceedings), March 1980

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," *Proceedings of the Eighth International Computer Software and Applications Conference*, November 1984

[5]Ramsey, C. L., and V. R. Basili, *An Evaluation of Expert Systems for Software Engineering Management*, University of Maryland, Technical Report TR-1708, September 1986

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[5]Rombach, H. D., "A Controlled Experiment on the Impact of Software Structure on Maintainability," *IEEE Transactions on Software Engineering*, March 1987

[8]Rombach, H. D., "Design Measurement: Some Lessons Learned," *IEEE Software*, March 1990

[9]Rombach, H. D., "Software Reuse: A Key to the Maintenance Problem," *Butterworth Journal of Information and Software Technology*, January/February 1991

[6]Rombach, H. D., and V. R. Basili, "Quantitative Assessment of Maintenance: An Industrial Case Study," *Proceedings From the Conference on Software Maintenance*, September 1987

[6]Rombach, H. D., and L. Mark, "Software Process and Product Specifications: A Basis for Generating Customized SE Information Bases," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[7]Rombach, H. D., and B. T. Ulery, *Establishing a Measurement Based Maintenance Improvement Program: Lessons Learned in the SEL*, University of Maryland, Technical Report TR-2252, May 1989

[6]Seidewitz, E., "Object-Oriented Programming in Smalltalk and Ada," *Proceedings of the 1987 Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 1987

[5]Seidewitz, E., "General Object-Oriented Software Development: Background and Experience," *Proceedings of the 21st Hawaii International Conference on System Sciences*, January 1988

[6]Seidewitz, E., "General Object-Oriented Software Development with Ada: A Life Cycle Approach," *Proceedings of the CASE Technology Conference*, April 1988

[9]Seidewitz, E., "Object-Oriented Programming Through Type Extension in Ada 9X," *Ada Letters*, March/April 1991

[4]Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

[9]Seidewitz, E., and M. Stark, "An Object-Oriented Approach to Parameterized Software in Ada," *Proceedings of the Eighth Washington Ada Symposium*, June 1991

[8]Stark, M., "On Designing Parametrized Systems Using Ada," *Proceedings of the Seventh Washington Ada Symposium*, June 1990

[7]Stark, M. E. and E. W. Booth, "Using Ada to Maximize Verbatim Software Reuse," *Proceedings of TRI-Ada 1989*, October 1989

[5]Stark, M., and E. Seidewitz, "Towards a General Object-Oriented Ada Lifecycle," *Proceedings of the Joint Ada Conference*, March 1987

[8]Straub, P. A., and M. V. Zelkowitz, "PUC: A Functional Specification Language for Ada," *Proceedings of the Tenth International Conference of the Chilean Computer Science Society*, July 1990

[7]Sunazuka, T., and V. R. Basili, *Integrating Automated Support for a Software Management Cycle Into the TAME System*, University of Maryland, Technical Report TR-2289, July 1989

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

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

[5]Valett, J. D., and F. E. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," *IEEE Transactions on Software Engineering*, February 1985

[5]Wu, L., V. R. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," *Proceedings of the Joint Ada Conference*, March 1987

[1]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: IEEE Computer Society Press, 1979

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

[6]Zelkowitz, M. V., "The Effectiveness of Software Prototyping: A Case Study," *Proceedings of the 26th Annual Technical Symposium of the Washington, D. C., Chapter of the ACM*, June 1987

[6]Zelkowitz, M. V., "Resource Utilization During Software Development," *Journal of Systems and Software*, 1988

[8]Zelkowitz, M. V., "Evolution Towards Specifications Environment: Experiences With Syntax Editors," *Information and Software Technology*, April 1990

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

NOTES:

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

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

[3]This article also appears in SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985.

[4]This article also appears in SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986.

[5]This article also appears in SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987.

[6]This article also appears in SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988.

[7]This article also appears in SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989.

[8]This article also appears in SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990.

[9]This article also appears in SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1992	CR189295

4. TITLE AND SUBTITLE
DATA COLLECTION PROCEDURES FOR THE SOFTWARE ENGINEERING LABORATORY (SEL) DATABASE

5. FUNDING NUMBERS
SEL 91 002

6. AUTHOR(S)
NASA, UNIV. OF MD, COMPUTER SCIENCES CORP.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES)
SAME AS #6

8. PEFORMING ORGANIZATION REPORT NUMBER
CR189295

9. SPONSORING / MONITORING ADGENCY NAME(S) AND ADDRESS (ES)
National Aeronautics and Space Administration
Goddard Space Flight Center, Greenbelt, MD 20771

10. SPONSORING / MONITORING ADGENCY REPORT NUMBER
CR189295

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATMENT
SINGLE COPIES CAN BE OBTAINED FROM CODE 552/GSFC

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document is a guidebook to collecting software engineering data on software development and maintenance efforts, as practiced in the Software Engineering Laboratory (SEL). It supersedes the document entitled Data Collection Procedures for the Rehosted SEL Database, number SEL 87 008 in the SEL series, which was published in October 1987. It presents an overview of SEL data collection and the types of data the SEL collects. It then presents procedures to be followed on software development and maintenance projects in the Flight Dynamics Division (FDD) of Goddard Space Flight Center (GSFC) for collecting data in support of SEL software engineering research activities. These procedures include detailed instructions for the completion and submission of SEL data collection forms.

14. SUBJECT TERMS
Development Life Cycle-Startup Form/ Estimates Form; Personnel Resources and Cleanroom Resources; Event Data; Project Completion; Data Collection-Maintenance

15. NUMBER OF PAGES
App. 100

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL