GLOSSARY OF SOFTWARE ENGINEERING LABORATORY TERMS

OCTOBER 1983

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

SOFTWARE ENGINEERING LABORATORY SERIES SEL-82-105

GLOSSARY OF SOFTWARE ENGINEERING LABORATORY TERMS

OCTOBER 1983

NASA
National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

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

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

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/TM-83/6168.

The contributors to this document include

Thomas Babst	(Computer Sciences Corporation)
Michael Rohleder	(Computer Sciences Corporation)
Frank McGarry	(Goddard Space Flight Center)

Single copies of this document can be obtained by writing to

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

ABSTRACT

This document is a glossary of terms used in the Software Engineering Laboratory (SEL). The terms are defined within the context of the software development environment for flight dynamics at Goddard Space Flight Center. The purposes of this document are to provide a concise reference for clarifying the language employed in SEL documents and data collection forms, establish standard definitions for use by SEL personel, and explain basic software engineering concepts.

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION

The glossary of Software Engineering Laboratory (SEL) terms presents a comprehensive collection of frequently used software engineering terms and expressions. Its objectives are to

- Provide a reference for clarifying the language of SEL documents and data collection forms
- Establish standard definitions for use by SEL personnel
- Explain basic software engineering concepts

The definitions provided in this document are consistent with the Institute of Electrical and Electronics Engineers (IEEE) publication Standard Glossary of Software Engineering Terminology (1983). However, some variations were needed to accommodate local (SEL) usages. Definitions were compiled from many sources: SEL personnel, data collection forms, and documents. The Data and Analysis Center for Software (DACS) document Glossary of Terms (1981) was also examined.

SECTION 2 - SOFTWARE ENGINEERING TERMS

acceptance testing	Independent testing conducted to verify that all functional requirements of a system have been satisfied. The results determine the acceptance or rejection of the software.
adaptability	A measure of the ease with which a program can be altered to fit differing user and system constraints.
adjusted lines of code	See lines of code.
algorithm	A prescribed set of well-defined rules or processes for the solution of a problem.
analyzer	Computer software used as a tool that is applied to a program to provide analytical information; it breaks the program into identifiable segments and reports statistical information. This information can include execution frequency statistics, program path analysis, and/or source code syntax analysis.
archive	Process involving the transfer of data or information from one source or volume to another to provide a backup or alternate copy of the information for future use.
argument	Variable or expression passed to an operation or function as input or output.
array	An ordered group or collection of variables, terms, or expressions. An array is usually dimensioned or indexed.
assemble	To translate a program written in assembly language into machine language. The assembly language operation codes are substituted with machine language operation codes, and symbolic addresses are substituted with absolute, immediate, relocatable, or virtual addresses.

assignment statement	An expression or instruction used to assign values to specified variables or symbols. Includes all statements that change the value of a variable as their main purpose; for example, READ statements. However, the assignment of the iteration counter in a DO statement is not included.
attribute list	A compiler-generated list of identifiers used by a program. The list includes type characteristics of identifiers, source statements that define or use the identifiers, and the relative storage location of the variables used in the program.
baseline	A tree chart or hierarchical graph of a software design containing all components in the system. A connection from a higher component to a lower one indicates that the higher component calls the lower one.
batch	Mode of operation of a computer in which the entire job is read into the machine before processing begins and in which there is no provision for interaction with the submitter during execution of the job.
block diagram	A diagram of a system or computer in which the principal parts are represented by geometrical figures that show both the basic functions and the functional relationships among the parts.
bug	See defect.
build	A functional subset of a more complex software development product. The "builds" approach to software development consists of developing a series of increasingly complete functional systems.
calibration error	An error in the gauge or tolerance of specifications.
certification test	A formal demonstration to the customer showing that requirements have been met.

change	A modification to requirements, design, code, or documentation made to correct an error, improve system performance, add capability, improve appearance, or implement a requirements change.
clerical error	An error made in the process of copying an item from one format to another or from one medium to another, which involves no interpretation or semantic translation.
code	A symbolic representation of a function composed of computer program statements.
code and unit test	See implementation.
code reading	Inspection of the source code by persons other than the creator of the code in an attempt to detect errors or to recommend coding improvements.
code walk-through	See walk-through.
coding	Representing a function in a form that is meaningful to a computer system.
cohesion	See module strength.
command/control	A class of software including programs used to generate satellite commands from the control center.
commission error	An error made by including an incorrect item that results in software containing a defect.
compile	To translate a computer program written in a high-level procedural language into a machine-language version.
complexity	A measure of the difficulty of implementing or understanding a component, independent of the implementor's experience; for example, the degree of interactions and number of dependencies among elements of a computer program.
component	A named piece of a system; for example, a separately compilable function, a functional subsystem, or a shared section of data such as a COMMON block.

computational error	Any error in which a value is computed by an incorrect mathematical expression.
computer architecture	The relationships between the parts of a computer system; the structural and functional definition of a computer as viewed in terms of its machine instruction set and input/output capabilities.
confidence level	The probability that a given statement is correct; 100 percent means that the statement is invariably true.
configuration control	A methodology for controlling the contents of a software system; a way of monitoring the status of system components, preserving the integrity of released and developing versions of a software system, and controlling the effects of changes throughout the system.
configuration item	A group or collection of computer hardware or software elements that are treated as a unit for the purpose of configuration management. Configuration items may vary widely in complexity and size.
control error	See logic error.
configuration management	All activities related to controlling the contents of a software system: monitoring the status of system components, preserving the integrity of released and developing versions of a system, and controlling the effects of changes throughout the system.
control statement	A statement that potentially alters the sequence of executed instructions; for example, GOTO, IF, RETURN, DO.
control structure	A recurrent pattern of control statements; for example, sequence, iteration, selection.
convention	An agreed-upon method, notation, or form of presentation.
correction	A change made to correct an error.

cosmetic change	A change in the source program made to improve clarity that has little effect on the performance of the program; for example, comment correction, movement of code that does not alter the implemented algorithm, or changing the name of a local variable.
cost estimation	Prediction made before and during a project's life cycle of the amount of labor necessary to complete a task, the amount and potential costs of computer time required, etc.
costing technique	A method for determining the cost of developing a system or any particular part of a system.
coupling	See module coupling.
criticality	A measure of the degree of dependence of the whole on a part of a system.
data	A series or collection of measurements.
data base	(1) A set of data files that are logically related. (2) An organized system of storing data.
data collection	The methods, forms, procedures, personnel, and activity used in measurement.
data definition language	A special-purpose language used to define data items in a data base and to create a data dictionary.
data dictionary	A file that describes the format of fields, values, and records in a data base.
data error	Any error in the use of a variable or data structure.
data set	A named collection of logically related data items, arranged in a prescribed manner residing in a physical storage location, usually magnetic tape or disk.
data structure	The logical relationship among the units of data in a data base.
data type	A set of attributes used to define a data item.
data validation	The process of verifying the completeness and accuracy of data.

data base management system	A software system for managing a data base, usually consisting of a data definition language and a data access language.
debugging	The process of locating and correcting software errors.
defect	An error in the design or implementation of a program. One or more software defects exist in a system if a software change is required to meet specified or implied system performance requirements. A defect may also be called a fault or bug.
design	(1) The process of defining how a system is to be constructed, its components, interfaces among those components, and interfaces with the external environment to satisfy specified requirements. (2) The results of the design process.
design language	A formal language for representing the logic, control, and data flow of a software system, usually input to an analyzer program.
design phase	The life cycle phase in which the structure of a system is planned and recorded.
- preliminary	The specification of major functional subsystems, input/output interfaces, processing modes, and implementation strategy. The software system architecture is defined, based on the requirements given in the functional specification and requirements document, and translated into software requirements in the requirements analysis summary report.
- detailed	The extension of the system architecture defined in the preliminary design phase to the subroutine level. The preliminary design is elaborated by successive refinement techniques to produce a "code-to" specification for the system.

design reading	Inspection of the design by persons other than the creator of the design for the purpose of detecting defects, development standard violations and other problems.
design review	A formal meeting between customer and developer to determine that a proposed software configuration will satisfy performance specifications.
design specification	A document describing the approved design for a program.
design verification	The formal examination or inspection of a software specification for the purpose of finding design errors and ambiguities.
design walk-through	See walk-through.
development methodology	A systematic approach to the creation of software that specifies the activities, products, verification, and completion criteria for each phase of development.
development phase	See life cycle.
developed lines of code	The total number of new lines of source code plus 20 percent of reused code.
discrepancy	The difference between the intention of a specification and its actual implementation.
documentation	Written material, other than source code statements, that describes a system or any of its components.
driver	A software component developed specifically to call other components; used in an informal testing technique during the implementation phase.
dynamic allocation	The allocation of memory required by an operating program during its execution phase rather than prior to execution.
efficiency	The ratio of useful work performed to the total energy expended. Code is efficient to the extent that it fulfills its purpose without wasting resources.
effort	The amount of resources, including staff and computer time, necessary to complete a particular project.

element	A basic segment of a named piece of a system (component).
embedded system	A dedicated computer system that is physically incorporated into a larger system whose primary function is not data processing; for example, an electromechanical system.
environment	The combination of hardware and software used to develop, maintain, and/or execute software, including the computer, operating system, support libraries, text editors, compiler, etc.
error	(1) An internal condition that prevents a software system from successfully performing its intended function. (2) Human action that results in software containing a defect. Also see failure, calibration error, clerical error, commission error, omission error, initialization error, logic error, interface error, data error, and computational error.
error analysis	The examination of errors with the purpose of tracing them to their sources and determining their effects.
error recovery	The ability of a system to resume processing rather than abort after an error.
estimation parameter	Any estimator or contributing factor to the process of estimation.
executable statement	Statement that changes the value of data or the state of a program.
execution	Performance by a computer of the instructions in a program.
execution time	The actual central processor time used in executing a program.
external reference	A call to a function or subroutine that is outside the calling program body.
failure rate	The number of failures occuring within a specified period of central processing unit time. Also see error rate.

failure, software	An unacceptable result produced during the operation of the computer program. Occurs when a fault is evoked by some input data. Also see error.
fault	See defect.
file	A set of related records treated as a unit.
flight dynamics software	Applications to support attitude determination and control, maneuver planning, orbit adjustment, and general mission analysis.
flow chart	A graphical representation of an algorithm in which symbols are used to represent operations, data, data flow, equipment, etc.
form	Questionnaire used to record information about the software development process and/or software product.
- change report	Records software change and error data during development.
- component status	Records time expended for activities.
- component summary	Records the status of system components.
- data base problem report	Used to identify and initiate action on data base problems.
- maintenance change report	Records software change and error data during maintenance.
- project summary	Used to classify the project and measure development progress.
- resource summary	Records expended resources.
- run analysis	Used to monitor activities for which the computer is used.
formal specification	A specification technique based on a strict set of rules for describing the specification and usually involving the use of an unambiguously defined notation; for example, mathematical functions or formal program design language.
formal testing	Testing performed in accordance with customer-approved test plans. Verifies that the software system is operating as specified in the requirements.

format statement	A source language statement that may accompany an input/output statement to specify the source or destination of the data and the arrangement of data items on the input or output record.
function	A mathematical subprogram used to specify an input set, an output set, and the relationship between the two.
functional specification	A specification of a software component as a set of functions defining the output for any input. Emphasizes what a program is to do rather than how to do it.
Halstead measures	Measures developed by M. Halstead in his theory of "software science," based on basic elements of programming languages: operator, operand, length, volume, and language level.
hardest first	The development approach of designing or implementing the most difficult aspects of a system first.
hardware	The physical and electronic components of a computer system including input/output devices, CPU, memory, etc.
hardware reliability	A measure of the probability of a hardware system operating without failure, usually measured as mean time to failure.
hierarchical input process output	A software design technique that defines each component in terms of a transformation from an input data set to an output data set, usually represented in graphic form.
hierarchy	A ranked series of elements, such as tasks, programs, people, functions, etc.
high-level language	A programming language that does not reflect the structure of any one given computer or that of any given class of computers.
historical	Of or pertaining to data archives on past experience with particular projects.
identifier	A symbol whose purpose is to identify, indicate, name, or locate a data structure or procedure entry point.

implementation	Life cycle phase in which code is developed or modified to meet design specifications. Each module (or unit) is integrated into the system and tested to ensure that the newly added capabilities function correctly.
informal testing	Testing involving no formal, written test plan.
initialization error	Any error resulting from an incorrectly initialized variable or failure to initialize a variable.
input/output	Usually refers to data or hardware processes involving the transfer of information to or from computer main memory.
instruction	See executable statement.
integration	The combination of subunits into an overall unit or system by means of interfacing.
integration test	A test of several modules to check that the interfaces are implemented correctly.
interactive	A mode of computer operation in which each line of input is immediately processed; allows communication with the program during its execution.
interface	The set of data and control information passed between two or more programs or segments of programs and the assumptions made by each program about how the others operate.
interface error	Any error of data exchange within a system (internal); any error of data exchange between some module and an entity outside the system (external).
interface testing	Validation that a module or set of modules operates within agreed interface specifications to ensure proper data and logical communications.
interpret	To translate and execute a high-level language program by translating each statement to a corresponding sequence of machine operations and executing them before proceeding to the next statement.

interrupt	Any stopping of a process by an external event in such a way that it can be resumed.
iteration	Repetition of a sequence of instructions until a specified set of conditions is satisfied.
iterative enhancement	The design or implementation of successive versions, each producing a usable subset of the final product, until the entire system is fully developed.
independent verification and validation	A software quality assurance technique in which an independent team reviews and tests the software while it is under development.
job	A unit of computer work consisting of one or more steps such as compilation, assembly, or utility runs.
job control language	A program language controlling the use of computer system resources.
librarian	Programming support person whose responsibilities include processing source statements but not writing them (for example, maintaining libraries, updating code, and producing tape backups).
life cycle	Sequence of phases during which the software product is developed from concept through delivery and operation. Also see individual phases: pretask planning, requirements analysis, preliminary design, detailed design, implementation, system testing, acceptance testing, and maintenance.
lines of code	Eighty-byte records that can be processed by a compiler or assembler.
- adjusted	An estimate of the number of executable lines of code developed. The sum of all new code plus 20 percent of the reused code, minus 50 percent of that total (estimated as the amount of comment lines), minus 10 percent of that result (estimated as the amount of nonexecutable statements).

- delivered	Total number of lines of source code generated as a deliverable item for a project. Includes all executable, nonexecutable, and comment statements whether newly coded or taken from existing programs and library routines.
- developed	Total number of new lines of source code plus 20 percent of reused code.
- executable	Code that changes the value or state of a program or data.
- modified	Previously developed code that has been changed for reuse in a new system.
- new	Total number of lines of source code written by programmers for a given task. Does not include any code that was taken from previously existing programs, but does include comments, executable, and nonexecutable statements.
- old	Total number of lines of source code taken from previously existing programs and reused without change.
- reused	See old lines of code.
load module	An executable program produced by translating and linking source code.
logic error	Any error resulting from an incorrectly formulated decision or transfer.
machine language	A system of numeric operation codes, values, and addresses, a sequence of which can be directly executed by a computer.
macro	A single instruction in a source language that represents a defined sequence of source instructions in the same language. A macro is replaced by the sequence it represents before program translation.
main program	A program unit containing at least one executable statement and having a starting address for program execution; normally, the set of instructions that determines the basic sequence of control.

maintenance	The process of modifying existing operational software to correct errors or enhance capabilities while leaving its primary function intact.
management, software	All the technical and management activities, decisions, and controls directly required to purchase, develop, or maintain software throughout its life cycle.
management, technical	Planning, organization, motivation (direction), and control of a technical project and technical personnel.
manpower	See staff-level and staff-unit.
measure	A count or numerical rating of the occurrence of some property. Examples include lines of code, number of computer runs, person-hours expended, and degree of use of top-down design methodology.
methodology	A prescribed set of principles and procedures for the development process. These principles may pertain to requirements, design, code, testing, or management. Examples include structured analysis, top-down design, information hiding, structured programming, formal test plans, and configuration management.
metric	See measure.
microcomputer	A class of computer based on a microprocessor.
microprocessor	A single integrated circuit (microprocessing unit) that performs the functions of a central processing unit.
mission date	The date that the system must be operational, usually 2 months before launch.
model	Equation relating two or more quantitative factors. A resource utilization model may provide an estimate of the cost of a project; a reliability model may indicate when sufficient testing has been done.
modification	The process of altering a program and its specification to perform either a new task or a different but similar task.

modified code	See lines of code.
module	A named subroutine unit that is independently compilable.
module coupling	A measure of the strength of the connections between two modules in a computer program. Module independence is a desirable software quality. The levels of module coupling from lowest (best) to highest (worst) are data, stamp, control, external, common, and content.
module strength	A measure of the unity of purpose or cooperation among the internal elements of a module. Module cohesion is a desirable software quality. The levels of module strength from highest (best) to lowest (worst) are functional, informational, communicational, procedural, classical, logical, and coincidental.
module test	See testing, unit.
new lines of code	See lines of code.
object module	A computer program expressed in machine language, usually the result of translating a source program by an assembler or compiler.
omission error	An error made by leaving out an item that results in software containing a defect.
online processing	Interactive processing, between humans and the computer.
operand	A symbol denoting a data item, indicator, or target of the action of an operator. Also see Halstead measures.
operator	A symbol denoting an operation, function, or action. Also see Halstead measures.
operating system	An integrated set of routines and services that monitor and manage system resources and the execution of application programs.
operation	A function that transforms data objects from input domain(s) into data objects in the operation's output domain(s).

optimization	A change in the source code to improve program performance, for example, to make it run faster or use less space. Optimization changes are not error corrections; however, if the change is made to conform to a specified requirement, the term "error" applies.
overlay	A hierarchical structure of program components that allows the program to be executed while only part of it resides in main memory at any given time.
parameter	A variable or measure that can take on more than one value, but only one at a time.
parse	To decompose a sequence of symbols (block, line, phrase) into a set of elementary subunits (words, commands, characters).
phase	See life cycle.
precompiler	A computer program used to add special-purpose capabilities to a language system. A precompiler translates special features implemented as macros into regular instruction sequences in a programming language.
preliminary design	See design phase.
pretask planning	Planning efforts prior to the start of requirements analysis; generation of software development plans and estimates.
preventive maintenance	Maintenance specifically intended to prevent faults from occurring.
procedure	(1) A sequence of steps that accomplishes some task. (2) A named subroutine.
procedural specification	A specification of a software component in an algorithmic manner, stating how the program is to work.
process design language	See program design language.
productivity	A measure of the rate of production per unit of effort expended. Typically, lines of code produced per staff hour.

program	A sequence of instructions that directs the computer to perform a task.
program complexity	A measure of the number of execution paths in the program and the difficulty of determining the path for an arbitrary set of input data. Also see complexity.
program design language	A language, often called pseudocode, used in the design and coding phases of a project, that contains a fixed set of control statements and a formal or informal way of defining and operating on data structures.
program listing	The sequence of instructions making up a computer program, usually in the form of a printout.
program validation	All techniques used to ensure correct programs, including system, and subsystem, and system integration testing.
programming language	A set of statements and instructions with a formal syntax and lexical rules; used in composing computer programs that require translation prior to machine execution.
project	A software development effort with set goals and defined objectives that uses the technical and managerial capabilities of personnel, has a life cycle with fixed endpoints, and produces a specified product.
proof technique	A method for formally demonstrating that a piece of software performs according to its specifications. Proof techniques usually use some form of mathematical notation to describe the result of executing a program.
prototype	A system developed with the intention of serving as a pattern for a future development effort.
quality	The degree to which software conforms to certain desirable characteristics. These may include, but are not limited to, correctness, reliability, usability, validity, efficiency, flexibility, and maintainability.

quality assurance	A planned and systematic procedure for ensuring that the product conforms to established technical requirements and quality standards.
read	The reading by peers of code and design materials to look for errors, invent tests, and so on.
real-time	A program that receives input from a process or activity and reacts in time to affect that process or activity.
reliability	The probability that software will function without failure under stated conditions for a stated period of time.
requirement	A system specification written by the user to define a system to a developer. The developer uses this specification in designing, implementing, and testing the system.
requirements analysis	An analysis of the contents of the functional specification and requirements document from a software system viewpoint, to recast the requirements in terms suitable for software design. The completeness and feasibility of the requirements are assessed; missing or to-be-determined requirements are identified; all external interfaces are specified; and the initial determination and allocation of resources are made.
requirements testing	The execution of a software product under controlled conditions to demonstrate that all stated or implied requirements and performance criteria have been met.
resource	Any person, equipment, or facility that may be allocated to the accomplishment of a task.
resource estimation model	A model that attempts to relate measures of staff and/or computer time to measures of the software problem, product, process, and environment. Models may range from simple, single variable equations to complex interactive software packages.
reused code	See lines of code.

review	A formal meeting of several individuals for the purpose of examining design, requirements, or code.
routine	A program or subprogram.
scheduling	The allocation of time and resources necessary to complete a given task or project.
segment	A contiguous piece of code that is unnamed and, hence, cannot be referred to as a single entity in a program statement. Could be one or several lines of a routine, subroutine, part of a data area, or an arbitrary contiguous section of memory.
shared items	Data and programs accessible by several components, such as COMMON blocks, external files, and library subroutines.
simulated constructs	Statements used to simulate structured control structures when the language to be used does not contain these structures.
software	Computer program code and its associated data, documentation, and operational procedures.
software class	The functional type of a software item. The principal types are scientific, data processing, and control.
software development life cycle	See life cycle.
software engineering	A scientific approach to software development integrating proven cost-effective methodologies, tools, and techniques into a comprehensive procedure.
software reliability	See reliability.
software testing	The process of exercising software in an attempt to detect errors that exist in the code. Also see formal testing.
source statements	All statements input to a compiler. Includes executable statements (assignment, IF, and GO TO); nonexecutable statements (DIMENSION, REAL, and END); and comments.

specification	A description of the input, output, and essential function(s) to be performed by a component of the system. Produced by the organization that is to develop the system; that is, it can be thought of as the contractor's interpretation of the requirements.
specification-driven	Uses the specifications of the program to determine test data; for example, generating test data by examining the input/output requirements and specifications.
staff-units	Units of measurement for human effort expended over time. Examples include staff-years, staff-months, and staff-hours.
standard	Any specification that refers to the method of development of the source program itself, and not to the problem to be implemented; for example, using structured code, limiting subroutines to 100 lines, or prefixing all module names with the subsystem name.
string processing	Operations performed on lists of characters.
structure-driven	Uses the structure of the program to determine test data; for example, generating data to ensure that each branch of a program is executed at least once.
structured code	Code that uses only a basic set of control structures: DO WHILE (iteration), IF-THEN-ELSE (selection), and BEGIN-END (sequence) or their derivatives (CASE, REPEAT UNTIL, etc.).
structured design	A set of techniques for reducing the complexity of large new programs by dividing them into independent modules. It produces a modular, hierarchical design consistent with structured coding practices.
structured programming	A set of techniques used to design, organize, and code programs that reduces complexity, improves clarity, facilitates debugging, and simplifies modification. The techniques include top-down development and structured coding.

stub	A "dummy" software element used in place of an expected functional element until that element becomes available.
subprogram	See subroutine.
subroutine	(1) A module that is separately compilable but not independently executable. (2) A collection of program elements that provides a function that is relatively independent of the whole program.
subsystem	A collection of subprograms that provides a major function and is independent of any other subsystem.
support software	All programs used in the development and maintenance of the delivered operational programs.
systems software	Software that is shared among application programs and facilitates or extends the use of system resources by the application programs.
system	A set or arrangement of software and/or hardware that together performs a common function.
system description	A document providing system baselines, data flows, and processing descriptions.
system integration	The process of combining system components to produce the total system.
system size	(1) The number of lines of code making up the software of a system. (2) The amount of memory, including instructions and data required to execute the system without overlays or paging.
system test	The process of trying to find discrepancies between the performance of a system and its original objectives.
table handler	A component that is specifically designed to generate or interpret information stored in a table format.

task	A set of defined objectives. Multiple tasks are initiated to complete a project. Also see project.
technical management	See management.
telemetry	Data transmitted at regular intervals from sensors.
test	A procedure designed to verify some aspect of the performance of a software system.
test plan	A description of test conditions that includes inputs, expected outputs, parameter values, etc.
test plan document	A management document that describes how and when specified test objectives will be met for the formal test plan.
testing	Software development activity in which a software system is subjected to specific conditions to show that it meets the intended design. Also see acceptance testing and system testing.
- functional	Testing designed to demonstrate a specific functional capability of a program or software system.
- structural	Testing designed to ensure that every path through the software is executed.
- unit	Test of a set of program statements treated logically as a whole. A unit is usually a component, subroutine, or module.
timesharing	A mode of operation that provides for the interleaving of two or more independent processes on one functional unit.
tool	A software aid used to facilitate the work of development team members; for example, text editors, precompilers, code auditors, and test generators.
top-down development	The design and implementation of the system by starting with the highest level component and developing the components on each successive level in turn.
top-down testing	Testing of modules in the top-down order in which they were produced.

tree chart	An acyclic connected graph, often representing a hierarchy in which the edges are directed to denote a subordinating relationship between the joined nodes.
uncertainty	The probability of error, or the probable magnitude of error.
unit	A set of computer program statements treated logically as a whole; usually a module or subroutine. Also see component, subroutine, and module.
unit test	See testing, unit.
user	The individual at the man/machine interface who is applying the software to the solution of a problem.
user-defined	A parameter determined by the user as input during program execution.
user's guide	A document designed to assist the user in operating the software product.
utility	Any component that is generated to satisfy some general support function required by other applications software.
validation	The process of determining whether a software product satifies its intended function regardless of whether or not it meets its requirements and specifications.
verification	The process of determining whether a software product meets its formal requirements and specifications.
walk-through	A formal meeting for the review of source code and/or design by project members for the purpose of error detection, not correction.
work-around	The process or result of counteracting the effects of an error in a program when the cause of the error and, consequently, the location of the statements containing the error is not known or is inaccessible; for example, a compiler error.

work unit	A measure of software size for which the effort required is known or can be approximated. A project is broken down into work units to facilitate cost estimation. Some common work units include the number of requirements, programs, subsystems, modules, pages of documentation, and lines of code.

SECTION 3 - ACRONYMS

ACC	Accounting Information File
ALC	Assembly Language Code
ATR	Assistant Technical Representative
BMDP	Biomedical Programs, P Series
CAREM	Cost and Reliability Estimating Models
CAT	Configuration Analysis Tool
CDR	Critical Design Review
CIF	Component Information File
CMT	Comment File
COCOMO	Constructive Cost Model
CRF	Change Report Form
CSC	Computer Sciences Corporation
CSF	Component Summary Form
CSR	Component Status Report
DAIO	Direct Access Input/Output Program
DARES	Data Base Retrieval System
DBA	Data Base Administrator
DBAM	Data Base Maintenance System
DLOC	Developed Lines of Code
FTIO	FORTRAN Input/Output Program
GESS	Graphic Executive Support System
GSFC	Goddard Space Flight Center
HDR	Project Header File
HIPO	Hierarchical Input Processing Output
HIS	Growth History File
IV&V	Independent Verification and Validation
JCL	Job Control Language
LOC	Lines of Code
MPP	Modern Programming Practices
MTTF	Mean Time to Failure
ORR	Operational Readiness Review
PANVALET	Computer Program Analysis and Security System

PDL	Program/Process Design Language
PDR	Preliminary Design Review
PRICE-S	Programmed Review of Information for Costing and Evaluation Software Model
RAF	Run Analysis Form
RSF	Resource Summary Form
SAP	FORTRAN Static Source Code Analyzer Program
SAS	Statistical Analysis System
SEF	Subjective Evaluations File
SEL	Software Engineering Laboratory
SFORT	Structured FORTRAN Preprocessor
SLIM	Software Life-Cycle Management Estimating Model
SRR	System Requirements Review
STL	Systems Technology Laboratory
TBD	To Be Determined
TSO	IBM Timesharing Option
UM	University of Maryland

BIBLIOGRAPHY OF SEL LITERATURE

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

SEL-Originated Documents

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

SEL-77-001, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

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

SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu and D. S. Wilson, September 1977

SEL-77-004, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

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

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

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

†This document superseded by revised document.

SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson and B. Chu, September 1978

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

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

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

SEL-79-001, SIMPL-D Data Base Reference Manual, M. V. Zelkowitz, July 1979

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

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

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

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

SEL-80-001, Functional Requirements/Specifications for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, A. L. Green, and C. E. Goorevich, February 1980

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

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

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

†This document superseded by revised document.

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

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

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

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

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

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

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

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

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

†SEL-81-103, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, P. Lo and D. Card, April 1983 (superseded by July 1983 version of SEL-81-103)

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

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

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

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

†This document superseded by revised document.

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

SEL-81-006, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, December 1981

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

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

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

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

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

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

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

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

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

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

SEL-82-002, FORTRAN Static Source Code Analyzer Program (SAP) System Description, W. A. Taylor and W. J. Decker, August 1982

†This document superseded by revised document.

SEL-82-003, Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description, P. Lo, September 1982

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

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

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

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

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

SEL-Related Literature

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

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

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

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

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

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.

††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

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

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

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

Basili, V. R., R. W. Selby, and T. Phillips, Metric Analysis and Data Validation Across FORTRAN Projects, University of Maryland, Technical Report, November 1982

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity and Cost, October 1979

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

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

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

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

---

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.

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

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

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

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

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

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

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

††Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Freburger, K., "A Model of the Software Life Cycle" (paper prepared for the University of Maryland, December 1978)

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

Hislop, G., "Some Tests of Halstead Measures" (paper prepared for the University of Maryland, December 1978)

---

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.

Lange, S. F., "A Child's Garden of Complexity Measures" (paper prepared for the University of Maryland, December 1978)

McGarry, F. E., G. Page, and R. Werking, Software Development History of the Dynamics Explorer (DE) Attitude Ground Support System, M&DO Development Report June 1983

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

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (proceedings), March 1980

Page, G., "Software Engineering Course Evaluation," Computer Sciences Corporation, Technical Memorandum, December 1977

Parr, F., and D. Weiss, "Concepts Used in the Change Report Form," NASA, Goddard Space Flight Center, Technical Memorandum, May 1978

Reiter, R. W., "The Nature, Organization, Measurement, and Management of Software Complexity" (paper prepared for the University of Maryland, December 1976)

Scheffer, P. A., and C. E. Velez, "GSFC NAVPAK Design Higher Order Languages Study: Addendum," Martin Marietta Corporation, Technical Memorandum, September 1977

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

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

Weiss, D. M., "Error and Change Analysis," Naval Research Laboratory, Technical Memorandum, December 1977

Williamson, I. M., "Resource Model Testing and Information," Naval Research Laboratory, Technical Memorandum, July 1979

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.

††Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: Computer Societies Press, 1979

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

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

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.