AMERICAN NATIONAL STANDARD

IEEE Standard Glossary of Software Engineering Terminology

Circuits and Devices

Communications Technology

Computer

Sponsored by the
Software Engineering Technical Committee
of the IEEE Computer Society

Electromagnetics and Radiation

Energy and Power

Industrial Applications

Signals and Applications

Standards Coordinating Committees

ANSI/IEEE Std 729-1983

Published by the Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA.
February 18, 1983
SH08920

ANSI/IEEE
Std 729-1983

An American National Standard

IEEE Standard Glossary of Software Engineering Terminology

Sponsor

Software Engineering Technical Committee
of the
IEEE Computer Society

Approved September 23, 1982

IEEE Standards Board

Approved August 9, 1983

American National Standards Institute

The Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street, New York, NY 10017

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE which have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least once every five years for revision or reaffirmation. When a document is more than five years old, and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
345 East 47th Street
New York, NY 10017
USA

Foreword

(This Foreword is not a part of ANSI/IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology.)

Software engineering is an emerging field. New terms are continually being generated, and new meanings are being adopted for existing terms. The Glossary of Software Engineering Terminology was undertaken to document this vocabulary. Its purpose is to identify terms currently used in software engineering and to present the current meanings of these terms. It is intended to serve as a useful reference for software engineers and for those in related fields and to promote clarity and consistency in the vocabulary of software engineering. It is recognized that software engineering is a dynamic area; thus the standard will be subject to appropriate change as becomes necessary.

This glossary was prepared by the Terminology Task Group of the Software Engineering Standards subcommittee of the Software Engineering Technical Committee of the IEEE Computer Society.

Comments are welcome and should be directed to:

The Secretary
IEEE Standards Board
345 East 47th Street
New York, NY 10017

At the time this standard was approved, the Terminology Task Group had the following members:

Shirley A. Gloss-Soler, *Chairperson*

Steering Committee:

Russell J. Abbott	Glenn C. Hughes, II	Jane W. Radatz
Joan P. Bateman	John M. Ives	Marilyn J. Stewart
Stephen R. Beason	John J. McKissick, Jr	Alan N. Sukert
Milton E. Boyd, Jr	Albrecht J. Neumann	Donald A. Woodmancy
Kurt F. Fischer	John N. Postak	David Yablon

Other members:

Duane C. Abbey	Raymond Houghton	William E. Russell
Allen Troy Acree, Jr	William B. Humphrey	David J. Schultz
Lynn Beckwith	F.T. James	Lee Shaw
Peter C. Belford	Tom Kallai	Stanley Siegel
Edward H. Bersoff	Edwin Kammerer	Gene Sievert
Grady Booch	Kathleen A. Kenney	Ronald C. Sivertson
John B. Bowen	Brij M. Khandelwal	John Smart
Martha A. Branstad	Ronald Klobert	Mark K. Smith
John R. Brown	K. Peter Koschewa	I. Richard Smook
Fletcher J. Buckley	O.J. Krten	Joseph Spellman
E. Jane Cameron	James C. Landerkin	James D. Stringer
Joseph Cavano	John B. Lane	E. Burton Swanson
James V. Cellini, Jr	Myron Lipow	Douglas B. Tabor
Karl E. Christen	Benn Manny	Barbara J. Taute
Tom Clark	Harvey Marks	Linda T. Taylor
Michele J. DiFranza	Neldon Marshall	Alex Terentiev
Paul Doelger	James A. McCall	Richard H. Thayer
Edward H. Ely	Siba Mohanty	Rinaldo A. Vaccaro
B. Dale Farrar	Dennis B. Mulcare	Don F. Utter
Carolyn Gannon	John Nissen	Keith Waters
David Gelperin	John O'Rourke	Andrew H. Weigel
R.R. Gordon	R. Poston	Herb Weiner
Robert M. Gross	Stephen Pozgaj	Peter J. Weyman
David A. Gustafson	J.A. (Jock) Rader	Terry Wolf
Virl Haas	Geraldine (Burke) Rajcula	Yvonne Wong
George B. Hawthorne	Jean C. Rault	Ralph Worrest
Herbert Hecht	R. Ravichandran	Martin Young
Robert E. Hite	Donald J. Reifer	Peter F. Zoll
Daniel E. Hocking	Debra J. Richardson	Jean Cochrane Zolnowski

At the time it approved this glossary, the Software Engineering Standards Subcommittee had the following membership:

Fletcher J. Buckley, *Chairperson*

Russell J. Abbott
Peter G. Anderson
William L. Anderson
Joan P. Bateman
Leo Beltracchi
Moredechai Ben-Menachem
Richard L. Bernstein
Edward H. Bersoff
Barry W. Boehm
John B. Bowen
Martha A. Branstad
A. Winsor Brown, Jr
Fletcher J. Buckley
Brian H. Burger
James V. Cellini, Jr
John Center
Won L. Chung
Antonio M. Cicu
Bruce M. Clay
Marlene Conklin
Guy L. Copeland
Jack Cowan
Stewart G. Crawford
George Darling
James R. Dildine
James V. Dinkey
Walter Du Blanica
Lorraine Duvall
Robert E. Dwyer
Mary L. Eads
John D. Earls
Leo G. Egan, Jr
Richard E. Fairley
Chungpeng Fan
Dennis W. Fife
Kurt F. Fischer
Joel J. Forman
Andre Fortier
Heinz H. Frey
Morris Frimer
M. Galinier
G. W. Gates
David Gelperin
Edward L. Gibbs
Shirley A. Gloss-Soler
Amrit L. Goel
Jack A. Goetz
John J. Greene
Robert M. Gross
H. Mark Grove
Russell T. Gustin
Paul E. Haddon
Thomas L. Hannan
George B. Hawthorne
Joel J. Hebert
Herbert Hecht
Leslie R. Heselton, III
Charles P. Hollocker
Grace M. Hopper
Sam Horvitz
Glenn C. Hughes, II
Donald J. Humcke
Shuenn-Chang Hwang
Jim H. Ingram
John M. Ives
Dwayne L. Knirk
George Konomos
Richard W. Kubica
Thomas M. Kurihara
Dominic V. LaRosa
Gregory N. Larsen
Arvid G. Larson
Albert M. Lerner
Paul J. Levine, Jr
William M. Lively
Alan Cheuk-Wai Ma
G. H. MacEwen
Andy K. Mahindru
Ben Manny
Philip C. Marriott
W. R. Mattersdorff
Lawrence J. Mazlack
James A. McCall
John McIntosh
John McKissick, Jr
Bertrand Meyer
Edward F. Miller, Jr
Siba N. Mohanty
M. F. Moon
Gene T. Morun
David G. Mullens
Walter G. Murch
John D. Musa
Myron L. Nack
Saied Najafi
Geraldine Neidhart
John O. Neilson
Albrecht J. Neumann
Leon Osterweil
Donald J. Ostrom
Thomas Pittman
John N. Postak
Robert M. Poston
Patricia B. Powell
Jane W. Radatz
Steven R. Rakitin
Jean C. Rault
T.L. Regulinski
Hans Reiche
Donald J. Reifer
R. San Roman
R. Waldo Roth
Clarence W. Rutter, III
Burnett H. Sams
Norman F. Schneidewind
Harvey E. Shock, Jr
Roger W. Scholten
Leonard W. Seagren
I. Lee Shaw
Stanley Siegel
Roy L. Skelton
Orval N. Skousen
Marian P. Smith
I. Richard Smook
Harry M. Sneed
Al R. Sorkowitz
D. Steinberg
Edward A. Straker
Alan N. Sukert
Douglas B. Tabor
Barbara J. Taute
Daniel Thalmann
Rush Thompson
George D. Tice, Jr
Terrence L. Tillmanns
George W. Trever
William S. Turner, III
R.L. Van Tilburg
Robert D. Vavra
Udo Voges
Dale R. Webdale
Andrew H. Weigel
Peter J. Weyman
Farrell L. White
Paul A. Willis
Saul A. Zaveler
Jean C. Zolnowski

Special representatives to the Software Engineering Subcommittee were as follows:

American Society for Quality Control:	A. Ferlan Wayne Kost
ANSI Z1:	J. Milandin
EDP Auditors Association:	Roy Pritchett

The Group wishes to acknowledge the close coordination and assistance of the American National Standard Committee X3K5 Subcommittee.

The following organizations supported the development of this standard:

Advanced Data Management
The Aerospace Corporation
Air Force Weapons Laboratory
Airmics, U.S. Army
Amdahl Corporation
American Society for Quality Control
Auto-trol Technology Corporation
Bell-Northern Research
Bell Laboratories
Boeing Computer Services Co.
Boeing Commercial Airplane Company
CTEC, Inc.
California State University, Northridge
Cameron Programming Consultants
Computer Sciences Corporation
Computron Systems, Inc.
Coopers and Lybrand
Data Analysis Center for Software
Digital Equipment Corporation
Doty Associates, Inc.
EG&G, Idaho
Edinboro State College
Electronics Corporation of America
GEC Telecommunications, LTD
GTE Laboratories
General Electric Company
General Research Corporation
Georgia Institute of Technology
Greatermans Management Services, LTD
Honeywell, Inc.
Hughes Aircraft Company
Human Computing Resources Corporation
Identicon Corporation
IIT Research Institute
Informatics, Inc.
Institute DeRecherche D'Informatique
Et D'Automatique
Johnson Controls, Inc.
Kansas State University
Lockheed - Georgia Company
Logicon, Inc.
MRI Systems Corporation
Martin Marietta Labs
Microdate Corporation
NCR Corporation
National Bureau of Standards
Paul Revere Life Insurance Co.
Perkin Elmer Co.
Programming Environments, Inc.
QYX - Intelligent Typewriter Systems
RCA
Rome Air Development Center
Sacramento State University
Smart Communications, Inc.
SoHaR, Inc
Software Management Consultants
Sperry Systems Management
Super Valu Stores, Inc.
System Development Corporation
TRW
Tektronix, Inc.
Teledyne Brown Engineering
Time, Inc.
U.S. Air Force Academy
UCLA
United States Army Signal Corps
University of Massachusetts
Westinghouse Electric Company

When the IEEE Standards Board approved this standard on September 23, 1982, it had the following membership:

I. N. Howell, Jr, *Chairman*

Irving Kolodny, *Vice Chairman*

Sava I. Sherr, *Secretary*

G. Y. R. Allen
J. J. Archambault
James H. Beall
John T. Boettger
J. V. Bonucchi
Edward Chelotti
Edward J. Cohen
Len S. Corey
D. C. Fleckenstein
Jay Forster
Kurt Greene
Joseph L. Koepfinger
John E. May
Donald T. Michael*
A. R. Parsons
J. P. Riganati
Robert W. Seelbach
Jay A. Stewart
Clifford O. Swanson
Robert E. Weiler

*Member emeritus

An American National Standard

IEEE Standard Glossary of Software Engineering Terminology

1. Scope

This glossary defines terms in general use in the software engineering field.

A term was excluded from the glossary if it was considered to be:

(1) Parochial to one particular group or organization.

(2) A company proprietary term.

(3) A standard term in some other well defined discipline, although terms used in software engineering were repeated in this glossary to avoid conflicts.

(4) A multi-word term whose meaning could be inferred from definitions of the component terms.

(5) A term whose meaning for software engineering could be directly inferred from its standard English meaning.

2. Glossary Structure

The structure of the glossary is as follows.

(1) Entries in the glossary are arranged alphabetically.

(2) If a term has more than one definition, the definitions are listed with numerical prefixes.

(3) Where necessary, examples have been added to clarify the definitions.

(4) The following crossreferences have been used to show a term's relationship to other terms in the glossary.

(a) *Compare with* refers to a complementary term.

(b) *Contrast with* refers to a term with an opposite or substantially different meaning.

(c) *Synonymous with* refers to a synonymous term.

(d) *See* refers the reader to a preferred or highly related term.

(e) *See also* refers to a related term.

(f) Words which appear in bold face type within a definition are defined elsewhere in the glossary. When a phrase is in bold face type, the glossary may define either the entire phrase or its component words.

3. Sources

In order to avoid conflict with accepted standards, American National Standards Institute (ANSI) and International Organization for Standardization (ISO) definitions that have relevance to software engineering are included. In those cases where the ANSI or ISO definitions do not fully reflect current software engineering usage, additional definition(s) are provided.

Sources are identified as follows.

(1) Definitions extracted from the ANSI Technical Report, American National Dictionary for Information Processing, X3/TR-1-77, September 1977, are identified by (ANSI) following the definition.

(2) Definitions developed by Technical Committee 97, (Information Systems), Subcommittee 1, (Vocabulary) of The International Organization for Standardization are identified by (ISO) following the definition.

(3) Definitions appearing in other standards are identified by that standard number following the definition. See Appendix A for complete identification of these references.

4. Terms

abort. To terminate a **process** prior to completion.

absolute machine code. Machine language code that must be loaded into fixed storage locations at each use and may not be relocated. Contrast with **relocatable machine code.**

abstract machine. (1) A representation of the characteristics of a **process** or machine.

(2) A **module** that processes inputs as though it were a machine.

abstraction. (1) A view of a problem that extracts the essential information relevant to a par-ticular purpose and ignores the remainder of the information.

(2) The process of forming an abstraction.

acceptance criteria. The criteria a **software product** must meet to successfully complete a **test phase** or meet **delivery requirements.**

acceptance testing. Formal testing conducted to determine whether or not a **system** satisfies its **acceptance criteria** and to enable the customer to determine whether or not to accept the system. See also **qualification testing, system testing.**

accessibility. The extent to which **software** facilitates selective use or **maintenance** of its **components.**

access-control mechanism. Hardware or **software** features, operating **procedures,** or management procedures designed to permit authorized access and prevent unauthorized access to a **computer system.**

accuracy. (1) A quality of that which is free of **error**. (ISO)

(2) A qualitative assessment of freedom from **error,** a high assessment corresponding to a small error. (ISO)

(3) A quantitative measure of the magnitude of **error,** preferably expressed as a function of the relative error, a high value of this measure corresponding to a small error. (ISO)

(4) A quantitative assessment of freedom from **error.**

Contrast with **precision.**

actual parameter. An argument or expression used within a call to a **subprogram** to specify **data** or **program** elements to be transmitted to the subprogram. Contrast with **formal parameter.**

adaptability. The ease with which **software** allows differing **system** constraints and user needs to be satisfied.

adaptive maintenance. Maintenance performed to make a **software product** usable in a changed environment.

address. (1) A character or group of characters that identifies a register, a particular part of storage, or some other **data** source or destination. (ISO)

(2) To refer to a device or an item of **data.** (ISO)

address space. The range of **addresses** available to a **computer program.**

algorithm. (1) A finite set of well-defined rules for the solution of a problem in a finite number of steps; for example, a complete specification of a sequence of arithmetic operations for evaluating sin x to a given **precision.** (ISO)

(2) A finite set of well-defined rules that gives a sequence of operations for performing a specific task.

algorithm analysis. The examination of an **algorithm** to determine its **correctness** with respect to its intended use, to determine its operational characteristics, or to understand it more fully in order to modify, simplify, or improve it.

alias. (1) An additional name for an item.

(2) An alternate **label.** For example, a label and one or more aliases may be used to refer to the same **data** element or point in a **computer program.** (ANSI)

analysis phase. See **requirements phase.**

analytical model. A representation of a **process** or phenomenon by a set of solvable equations. Contrast with **simulation.**

application-oriented language. (1) A computer-oriented language with facilities or notations ap-

plicable primarily to a single application area; for example, a language for statistical analysis or machine design.

(2) A problem-oriented language whose statements contain or resemble the terminology of the occupation or profession of the user. (ANSI)

application software. Software specifically produced for the functional use of a **computer system;** for example, software for navigation, gun fire control, payroll, general ledger. Contrast with **system software.**

architecture. See **program architecture, system architecture.**

architectural design. (1) The **process** of defining a collection of **hardware** and **software components** and their **interfaces** to establish a framework for the development of a **computer system.**

(2) The result of the architectural design process.

artificial language. See **formal language.**

assemble. To translate a **program** expressed in an **assembly language** into a **machine language** and perhaps to link **subroutines.** Assembling is usually accomplished by substituting machine language operation codes for assembly language operation codes and by substituting absolute addresses, immediate addresses, relocatable addresses, or virtual addresses for symbolic addresses. Contrast with **compile, interpret.**

assembler. A **computer program** used to **assemble.** (ISO) Contrast with **compiler, interpreter.** Synonymous with assembly **program.**

assembly language. (1) A computer-oriented language whose **instructions** are usually in one-to-one correspondence with **computer** instructions and that may provide facilities such as the use of **macroinstructions.** (ISO) Contrast with **machine language, higher order language.** See also **assemble, assembler.**

(2) A machine-specific language whose **instructions** are usually in one-to-one correspondence with **computer** instructions.

assertion. A logical expression specifying a **program** state that must exist or a set of conditions that program **variables** must satisfy at a particular point during program **execution;** for example, *A is positive and A is greater than B.* See also **input assertion, output assertion.**

assignment statement. An **instruction** used to express a sequence of operations, or used to assign **operands** to specified **variables,** or symbols, or both. (ANSI)

audit. (1) An independent review for the purpose of assessing compliance with **software requirements, specifications, baselines,** standards, **procedures,** instructions, codes, and contractual and licensing requirements. See also **code audit.**

(2) An activity to determine through investigation the adequacy of, and adherence to, established **procedures, instructions, specifications,** codes, and standards or other applicable contractual and licensing requirements, and the effectiveness of **implementation.** (ANSI N45.2.10-1973)

automated design tool. A **software tool** that aids in the synthesis, analysis, modeling, or **documentation** of a software **design.** Examples include **simulators,** analytic aids, design representation processors, and documentation generators.

automated test case generator. See **automated test generator.**

automated test data generator. See **automated test generator.**

automated test generator. A **software tool** that accepts as input a **computer program** and test criteria, generates test input **data** that meet these criteria, and, sometimes, determines the expected results.

automated verification system. A **software tool** that accepts as input a **computer program** and a representation of its **specification,** and produces, possibly with human help, a **correctness proof** or disproof of the **program.** See also **automated verification tools.**

automated verification tools. A class of **software tools** used to evaluate products of the **software development process.** These **tools** aid in the **verification** of such characteristics as **cor-**

rectness, completeness, consistency, traceability, **testability,** and adherence to standards. Examples include **design analyzers, automated verification systems, static analyzers, dynamic analyzers,** and **standards enforcers.**

availability. (1) The probability that **software** will be able to perform its designated **system function** when required for use.

(2) The ratio of **system** up-time to total operating time.

(3) The ability of an item to perform its designated **function** when required for use. (ANSI/ ASQC A3-1978)

availability model. A **model** used for predicting, estimating, or assessing **availability.**

back-up. Provisions made for the recovery of **data files** or **software,** for restart of processing, or for use of alternative **computer** equipment after a **system failure** or a disaster.

backup programmer. The assistant leader of a **chief programmer team;** a senior-level programmer whose responsibilities include contributing significant portions of the **software** being developed by the team, aiding the **chief programmer** in reviewing the work of the other team members, substituting for the chief programmer when necessary, and having an overall technical understanding of the software being developed.

baseline. (1) A **specification** or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal **change control procedures.**

(2) A **configuration identification document** or a set of such documents formally designated and fixed at a specific time during a **configuration item's** life cycle. Baselines, plus approved changes from those baselines, constitute the current configuration identification. For **configuration management** there are three baselines, as follows:

(a) Functional baseline. The initial approved functional configuration.
(b) Allocated baseline. The initial approved allocated configuration.
(c) Product baseline. The initial approved or conditionally approved product configuration identification. (DoD-STD 480A)

begin-end block. A sequence of **design** or programming statements bracketed by *begin* and *end* delimiters and characterized by a single entrance and single exit.

binding. The assigning of a value or referent to an **identifier;** for example, the assigning of a value to a **parameter** or the assigning of an absolute address, virtual address, or device identifier to a symbolic address or **label** in a **computer program.** See also **dynamic binding, static binding.**

bit. A contraction of the term *binary digit*; a unit of information represented by either a zero or a one.

block. (1) A **string** of **records,** a string of **words,** or a character string, formed for technical or logic reasons to be treated as an entity. (ISO)

(2) A collection of contiguous **records** recorded as a unit. Blocks are separated by interblock gaps and each block may contain one or more records. (ANSI)

(3) A group of **bits** or **N-ary** digits, transmitted as a unit. An encoding procedure is generally applied to the group of bits or N-ary digits for **error-**control purposes. (ANSI)

(4) A set of things, such as **words,** characters, or digits, handled as a unit. (ANSI)

(5) See **program block.**

block diagram. A diagram of a **system,** a **computer,** or a device in which the principal parts are represented by suitably annotated geometrical figures to show both the basic **functions** of the parts and their functional relationships. (ISO) Contrast with **flowchart.**

block-structured language. A **design** or **programming language** in which sequences of statements are demarcated, usually with *begin* and *end* delimiters. See also **program block.**

bootstrap. (1) A short **computer program** that is permanently resident or easily loaded into a **computer,** whose **execution** brings another, larger **program,** such as an **operating system** or its **loader,** into memory.

(2) A set of **instructions** that causes additional instructions to be loaded until the complete **computer program** is in storage. (ISO)

(3) A technique or device designed to bring itself into a desired state by means of its own action; for

example, a machine **routine** whose first few **instructions** are sufficient to bring the rest of itself into the **computer** from an input device. (ANSI)

(4) That part of a **computer program** used to establish another version of the computer program. (ANSI)

(5) To use a bootstrap. (ISO)

bootstrap loader. An input **routine** in which preset **computer** operations are used to load a **bootstrap.** (ISO)

bottom-up. Pertaining to an approach that starts with the lowest level **software components** of a **hierarchy** and proceeds through progressively higher levels to the top level component; for example, **bottom-up design,** bottom-up programming, bottom-up **testing.** Contrast with **top-down.**

bottom-up design. The **design** of a **system** starting with the most basic or primitive **components** and proceeding to higher level components that use the lower level ones. Contrast with **top-down design.**

bug. See **fault.**

bug seeding. See **fault seeding.**

build. An operational version of a **software product** incorporating a specified subset of the capabilities that the final product will include.

building block. An individual unit or **module** that is utilized by higher-level **programs** or modules.

byte. (1) A binary character **string** operated upon as a unit and usually shorter than a **computer word** (ISO).

(2) A group of adjacent binary digits operated on as a unit and usually shorter than a computer word (frequently connotes a group of eight **bits**). (ANSI/IEEE Std 488-1978)

case. A multi-branch conditional statement that allows for selective **execution** of bounded groups of **program** statements depending upon the value of a control expression. See also **control structure.**

certification. (1) A written guarantee that a **system** or **computer program** complies with its specified **requirements.**

(2) A written authorization that states that a **computer system** is secure and is permitted to operate in a defined environment with or producing sensitive information.

(3) The formal demonstration of **system** acceptability to obtain authorization for its **operational** use.

(4) The process of confirming that a **system, software subsystem,** or **computer program** is capable of satisfying its specified **requirements** in an operational environment. Certification usually takes place in the field under actual conditions, and is utilized to evaluate not only the software itself, but also the **specifications** to which the software was constructed. Certification extends the process of **verification** and **validation** to an actual or simulated operational environment.

(5) The procedure and action by a duly authorized body of determining, verifying, and attesting in writing to the qualifications of personnel, **processes, procedures,** or items in accordance with applicable **requirements.** (ANSI/ASQC A3-1978)

chained list. A **list** in which the items may be dispersed but in which each item contains an **identifier** for locating the next item. (ISO) Synonymous with **linked list.**

change control. The process by which a change is proposed, evaluated, approved or rejected, scheduled, and tracked.

chief programmer. The leader of a **chief programmer team;** a senior-level programmer whose responsibilities include producing key portions of the **software** assigned to the team, coordinating the activities of the team, reviewing the work of the other team members, and having an overall technical understanding of the software being developed.

chief programmer team. A **software** development group that consists of a **chief programmer,** a **backup programmer,** a **secretary/librarian,** and additional programmers and specialists as needed, and that employs support **procedures** designed to enhance group communication and to make optimum use of each member's skills.

code. (1) A set of unambiguous rules specifying the manner in which **data** may be represented in a discrete form. (ISO)

(2) To represent **data** or a **computer program** in a symbolic form that can be accepted by a processor. (ISO)

(3) To write a **routine.** (ANSI)

(4) Loosely, one or more **computer programs,** or part of a computer program.

(5) An encryption of **data** for **security** purposes.

code audit. An independent review of source **code** by a person, team, or **tool** to verify compliance with **software design documentation** and programming standards. **Correctness** and **efficiency** may also be evaluated. See also **audit, static analysis, inspection, walk-through.**

code generator. A **program** or program **function,** often part of a **compiler,** that transforms a **computer program** from some intermediate level of representation (often the output of a parser) into a lower level representation such as assembly **code** or machine code.

code inspection. See **inspection.**

code walk-through. See **walk-through.**

cohesion. The degree to which the tasks performed by a single **program module** are functionally related. Contrast with **coupling.**

command language. A set of procedural **operators** with a related **syntax,** used to indicate the **functions** to be performed by an **operating system.** Synonymous with control language. (ISO)

comment. (1) Information embedded within a **computer program, command language,** or set of **data** that is intended to provide clarification to human readers and that does not effect machine interpretation.

(2) A description, reference, or explanation added to or interspersed among the statements of the **source language,** that has no effect in the **target language.** (ISO)

comparator. A **software tool** used to compare two **computer programs, files,** or sets of **data** to identify commonalities or differences. Typical objects of comparison are similar versions of source **code,** object code, **data base** files, or test results.

compatibility. The ability of two or more **systems** to exchange information. Compare with **interoperability.**

compile. To translate a **higher order language program** into its **relocatable** or **absolute machine code** equivalent. Contrast with **assemble, interpret.**

compiler. A **computer program** used to **compile.** (ISO) Contrast with **assembler, interpreter.**

compiler generator. A **translator** or an **interpreter** that is used to construct **compilers.** (ISO) Synonymous with **metacompiler.**

compiler compiler. See **compiler generator.**

complexity. The degree of complication of a **system** or system **component,** determined by such factors as the number and intricacy of **interfaces,** the number and intricacy of conditional branches, the degree of **nesting,** the types of **data structures,** and other system characteristics.

component. A basic part of a **system** or **program.**

computer. (1) A **functional unit** that can perform substantial computation, including numerous arithmetic operations or logic operations, without intervention by a human operator during a run. (ISO).

(2) A functional programmable unit that consists of one or more associated processing units and peripheral equipment, that is controlled by internally stored **programs,** and that can perform substantial computation, including numerous arithmetic operations or logic operations, without human intervention.

computer data. Data available for communication between or within **computer** equipment. Such data can be external (in computer-readable form) or resident within the computer equipment and can be in the form of analog or digital signals.

computer network. A complex consisting of two or more interconnected **computers.** (ANSI)

computer program. A sequence of **instructions** suitable for processing by a **computer.** Processing may include the use of an **assembler,** a **compiler,** an **interpreter,** or a **translator** to prepare the **program** for **execution** as well as to execute it. (ISO) See also **program.**

computer program abstract. A brief description of a **computer program,** providing sufficient information for potential users to determine the appropriateness of the computer program to their needs and resources.

computer program annotation. See **comment.**

computer program certification. See **certification.**

computer program configuration identification. See **configuration identification.**

computer program development plan. See **software development plan.**

computer program validation. See **validation.**

computer program verification. See **verification.**

computer system. A **functional unit,** consisting of one or more **computers** and associated **software,** that uses common storage for all or part of a **program** and also for all or part of the **data** necessary for the **execution** of the program; **executes** user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during their execution. A computer system may be a standalone unit or may consist of several interconnected units. Synonymous with ADP system, computing system. (ISO)

concurrent processes. Processes that may execute in parallel on multiple processors or asynchronously on a single processor. Concurrent processes may interact with each other, and one process may suspend **execution** pending receipt of information from another process or the occurrence of an external event. Contrast with **sequential processes.**

conditional control structure. A programming **control structure** that allows alternative **flow of control** in a **program** depending upon the fulfillment of specified conditions; for example, **case,** if...then... else...

configuration. (1) The arrangement of a **computer system** or **network** as defined by the nature, number, and the chief characteristics of its **functional units.** More specifically, the term configuration may refer to a **hardware** configuration or a **software** configuration. (ISO)

(2) The **requirements, design,** and **implementation** that define a particular version of a **system** or system **component.**

(3) The functional and/or physical characteristics of **hardware/software** as set forth in technical **documentation** and achieved in a product. (DoD-STD 480A)

configuration audit. The process of verifying that all required **configuration items** have been produced, that the current version agrees with specified **requirements,** that the technical **documentation** completely and accurately describes the configuration items, and that all change requests have been resolved.

configuration control. (1) The process of evaluating, approving or disapproving, and coordinating changes to **configuration items** after formal establishment of their **configuration identification.**

(2) The systematic evaluation, coordination, approval or disapproval, and **implementation** of all approved changes in the **configuration** of a **configuration item** after formal establishment of its **configuration identification.** (DoD-STD 480A)

configuration control board. The authority responsible for evaluating and approving or disapproving proposed engineering changes, and ensuring **implementation** of the approved changes.

configuration identification. (1) The process of designating the **configuration items** in a **system** and recording their characteristics.

(2) The approved **documentation** that defines a **configuration item.**

(3) The current approved or conditionally approved technical **documentation** for a **configuration item** as set forth in **specifications,** drawings and associated lists, and **documents** referenced therein. (DoD-STD 480A)

configuration item. (1) A collection of **hardware** or **software** elements treated as a unit for the purpose of **configuration management.**

(2) An aggregation of **hardware/software,** or any of its discrete portions, that satisfies an end use **function** and is designated for **configuration management.** Configuration items may vary widely in **complexity**, size, and type from an aircraft, electronic or ship **system** to a test meter or round of ammunition. During development and initial production, Configuration items are only those specification items that are referenced directly in a contract (or an equivalent in-house agreement). During the operation and maintenance period, any reparable item designated for separate procurement is a configuration item. (DoD-STD 480A)

configuration management. (1) The process of identifying and defining the **configuration items** in a **system,** controlling the release and change of these items throughout the system life cycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and **correctness** of configuration items. See also **change control, configuration identification, configuration control, configuration status accounting, configuration audit.**

(2) A discipline applying technical and administrative direction and surveillance to (a) identify and document the functional and physical characteristics of a **configuration item,** (b) control changes to those characteristics, and (c) record and report change processing and **implementation** status. (DoD-STD 480A)

configuration status accounting. The recording and reporting of the information that is needed to manage a **configuration** effectively, including a listing of the approved **configuration identification,** the status of proposed changes to the configuration, and the **implementation** status of approved changes. (DoD-Std 480A)

confinement. (1) Prevention of unauthorized alteration, use, destruction, or release of **data** during authorized access. See also **integrity.**

(2) Restriction on **programs** and **processes** so that they do not access or have influence on **data,** programs, or processes other than that allowed by specific authorization.

connection. (1) A reference in one part of a **program** to the **identifier** of another part (that is, something found elsewhere). See also **interface.**

(2) An association established between **functional units** for conveying information. (ISO)

control data. Data that selects an operating mode or submode in a **program,** directs the sequential flow, or otherwise directly influences the operation of **software.**

control statement. A **programming language** statement that affects the order in which operations are performed.

control structure. A construct that determines the **flow of control** through a **computer program.** See also **conditional control structure.**

conversational. Pertaining to an **interactive** system that provides for interaction between a user and a **system** similar to a human dialog.

conversion. Modification of existing **software** to enable it to operate with similar functional capability in a different environment; for example, converting a **program** from FORTRAN to Ada, converting a program that runs on one **computer** to run on another computer.

coroutines. Two or more **modules** that can call each other, but that are not in a superior to subordinate relationship.

corrective maintenance. Maintenance performed specifically to overcome existing **faults.** (ISO) See also **software maintenance.**

correctness. (1) The extent to which **software** is free from **design** defects and from coding defects; that is, **fault** free.

(2) The extent to which **software** meets its specified **requirements.**

(3) The extent to which **software** meets user expectations.

correctness proof. See **proof of correctness.**

coupling. A measure of the interdependence among **modules** in a **computer program.** Contrast with **cohesion.**

critical piece first. Pertaining to an approach to **software** development that focuses on implementing the most critical aspects of a software **system** first. The critical piece may be defined in terms of services provided, degree of risk, difficulty, or some other criterion.

critical section. A **segment** of **code** to be **executed** mutually exclusively with some other segment of code that is also called a critical section. Segments of code are required to be executed mutually exclusively if they make competing uses of a **computer** resource or **data** item.

criticality. A classification of a **software error** or **fault** based upon an evaluation of the degree of impact of that error or fault on the development or operation of a **system** (often used to determine whether or when a fault will be corrected).

cross-assembler. An **assembler** that executes on one **computer** but generates object **code** for a different computer.

cross-compiler. A **compiler** that executes on one **computer** but generates assembly **code** or object code for a different computer.

data. A representation of facts, concepts, or **instructions** in a formalized manner suitable for communication, interpretation, or processing by human or automatic means. (ISO) See also **computer data, control data, error data, reliability data, software experience data.**

data abstraction. The result of extracting and retaining only the essential characteristic properties of **data** by defining specific **data types** and their associated functional characteristics, thus separating and hiding the representation details. See also **information hiding.**

data base. (1) A set of **data,** part or the whole of another set of data, and consisting of at least one **file** that is sufficient for a given purpose or for a given data processing **system.** (ISO)

(2) A collection of **data** fundamental to a **system.** (ANSI)

(3) A collection of **data** fundamental to an enterprise. (ANSI)

data dictionary. (1) A collection of the names of all **data** items used in a **software system,** together with relevant properties of those items; for example, length of data item, representation, etc.

(2) A set of definitions of **data** flows, data elements, **files, data bases,** and **processes** referred to in a leveled **data flow diagram** set.

data flow chart. See **data flow diagram.**

data flow diagram. A graphic representation of a **system,** showing **data** sources, data sinks, storage, and **processes** performed on data as **nodes,** and logical flow of data as links between the nodes. Synonymous with **data flow graph, data flow chart.**

data flow graph. See **data flow diagram.**

data structure. A formalized representation of the ordering and accessibility relationships among **data** items without regard to their actual storage configuration.

data type. A class of **data** characterized by the members of the class and the operations that can be applied to them; for example, integer, real, logical.

debugging. The process of locating, analyzing, and correcting suspected **faults.** Compare with **testing.**

debugging model. See **error model.**

decision table. (1) A **table** of all contingencies that are to be considered in the description of a problem together with the actions to be taken for each set of contingencies. (ISO)

(2) A presentation in either matrix or tabular form of a set of conditions and their corresponding actions. (ANSI)

defect. See **fault.**

definition phase. See **requirements phase.**

delivery. (1) The point in the **software development cycle** at which a product is released to its intended user for **operational** use.

(2) The point in the **software development cycle** at which a product is accepted by its intended user.

design. (1) The process of defining the **software architecture, components, modules, interfaces,** test approach, and **data** for a software **system** to satisfy specified **requirements.**

(2) The result of the design process.

design analysis. (1) The evaluation of a **design** to determine **correctness** with respect to stated **requirements,** conformance to design standards, **system efficiency,** and other criteria.

(2) The evaluation of alternative **design** approaches. See also **preliminary design.**

design analyzer. An **automated design tool** that accepts information about a **program's design** and produces such outputs as **module hierarchy** diagrams, graphical representations of control and **data structure,** and **lists** of accessed **data blocks.**

design inspection. See **inspection.**

design language. A language with special constructs and, sometimes, **verification** protocols used to develop, analyze, and **document** a **design.**

design methodology. A systematic approach to creating a **design,** consisting of the ordered application of a specific collection of **tools,** techniques, and guidelines.

design phase. The period of time in the **software life cycle** during which the **designs** for **architecture,** software **components, interfaces,** and **data** are created, documented, and verified to satisfy **requirements.**

design requirement. Any **requirement** that impacts or constrains the **design** of a **software system** or software system **component;** for example, **functional requirements, physical requirements, performance requirements,** software development standards, **software quality assurance** standards. See also **requirements specification.**

design review. (1) A formal meeting at which the preliminary or **detailed design** of a **system** is presented to the user, customer, or other interested parties for comment and approval.

(2) The formal review of an existing or proposed **design** for the purpose of detection and remedy of design deficiencies that could affect fitness-for-use and environmental aspects of the product, process or service, and/or for identification of potential improvements of **performance,** safety and economic aspects. (ANSI/ASQC A3-1978)

design specification. A **specification** that **documents** the **design** of a **system** or system **component; for example, a software configuration item.** Typical contents include **system** or component **algorithms,** control logic, **data structures, data** set-use information, input/output formats, and **interface** descriptions. See also **requirements specification.**

design verification. See **verification.**

design walk-through. See **walk-through.**

desk checking. The manual **simulation** of **program execution** to detect **faults** through step-by-step examination of the source **code** for **errors** in logic or **syntax.** See also **static analysis.**

detailed design. (1) The process of refining and expanding the **preliminary design** to contain more detailed descriptions of the processing logic, **data structures,** and **data** definitions, to the extent that the **design** is sufficiently complete to be **implemented.**

(2) The result of the detailed design process.

development cycle. See **software development cycle.**

development life cycle. See **software development cycle.**

development methodology. A systematic approach to the creation of **software** that defines development phases and specifies the activities, products, **verification procedures,** and completion criteria for each phase.

development specification. Synonymous with **requirements specification.** (DOD usage).

diagnostic. (1) A message generated by a **computer program** indicating possible **faults** in another **system component;** for example, a **syntax** fault flagged by a **compiler.**

(2) Pertaining to the detection and isolation of **faults** or **failures.**

digraph. See **directed graph.**

directed graph. A **graph** whose edges are unidirectional.

document. (1) A **data** medium and the data recorded on it, that generally has permanence and that can be read by man or machine. (ISO) Often used to describe human readable items only, for example, technical documents, **design** documents, version description documents.

(2) To create a document.

documentation. (1) A collection of **documents** on a given subject. (ISO) See also **user documentation, software documentation, system documentation.**

(2) The management of **documents** which may include the actions of identifying, acquiring, processing, storing, and disseminating them. (ISO)

(3) The process of generating a **document.**

(4) Any written or pictorial information describing, defining, specifying, reporting or certifying activities, **requirements, procedures,** or results. (ANSI N45.2.10-1973)

documentation level. See **level of documentation.**

driver. A **program** that exercises a **system** or system **component** by simulating the activity of a higher level component. See also **test driver.**

dual coding. A development technique in which two functionally identical versions of a **program** are developed from the same **specification** by different programmers or different programming teams. The resulting source **code** may be in the same or different languages. The purpose of dual coding is to provide for **error** detection, increase **reliability,** provide additional **documentation,** or reduce the probability of systematic programming errors or **compiler** errors influencing the end result.

dummy parameter. See **formal parameter.**

dump. (1) **Data** that have been dumped. (ISO)

(2) To write the contents of a storage, or of part of a storage, usually from an internal storage to an external medium, for a specific purpose such as to allow other use of the storage, as a safeguard against **faults** or **errors,** or in connection with **debugging.** (ISO)

dynamic allocation. The allocation of addressable storage and other resources to a **program** while the program is executing.

dynamic analysis. The process of evaluating a **program** based on **execution** of the program. Contrast with **static analysis.**

dynamic analyzer. A **software tool** that aids in the evaluation of a **computer program** by monitoring **execution** of the **program.** Examples include **instrumentation tools, software monitors,** and **tracers.** Contrast with **static analyzer.**

dynamic binding. Binding performed during **execution** of a **program.** Contrast with **static binding.**

dynamic restructuring. (1) The process of changing **software components** or structure while a **system** is running.

(2) The process of restructuring a **data base** or **data structure** during **program execution.**

editor. A **computer program** that permits selective revision of **computer**-stored **data.**

efficiency. The extent to which **software** performs its intended **functions** with a minimum consumption of computing resources.

egoless programming. An approach to **software** development based upon the concept of team responsibility for **program** development. Its purpose is to prevent the programmer from identifying so closely with his or her output that objective evaluation is impaired.

embedded computer system. A **computer system** that is integral to a larger system whose primary purpose is not computational; for example, a computer system in a weapon, aircraft, command and control, or rapid transit system.

embedded software. Software for an **embedded computer system.**

emulation. The imitation of all or part of one **computer system** by another, primarily by **hardware,** so that the imitating computer system accepts the same **data,** executes the same **programs,** and achieves the same results as the imitated system. (ISO)

emulator. Hardware, software, or **firmware** that performs **emulation.**

encapsulation. The technique of isolating a **system function** within a **module** and providing a precise **specification** for the module. See also **information hiding.**

error. (1) A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. (ANSI)

(2) Human action that results in **software** containing a **fault.** Examples include omission or misinterpretation of user **requirements** in a software **specification,** incorrect translation or omission of a requirement in the **design specification.** This is not a preferred usage.
See also **failure, fault.**

error analysis. (1) The process of investigating an observed **software fault** with the purpose of tracing the fault to its source.

(2) The process of investigating an observed **software fault** to identify such information as the cause of the fault, the phase of the development process during which the fault was introduced, methods by which the fault could have been prevented or detected earlier, and the method by which the fault was detected.

(3) The process of investigating **software errors, failures,** and **faults** to determine quantitative rates and trends.

error category. One of a set of classes into which an **error, fault,** or **failure** might fall. Categories may be defined for the cause, **criticality,** effect, life cycle phase when introduced or detected, or other characteristics of the error, fault, or failure.

error data. A term commonly (but not precisely) used to denote information describing **software** problems, **faults, failures,** and changes, their characteristics, and the conditions under which they are encountered or corrected.

error model. A mathematical **model** used to predict or estimate the number of remaining **faults, reliability,** required test time, or similar characteristics of a **software system.** See also **error prediction.**

error prediction. A quantitative statement about the expected number or nature of **software** problems, **faults,** or **failures** in a software **system.** See also **error model.**

error prediction model. See **error model.**

error recovery. See **failure recovery.**

error seeding. See **fault seeding.**

exception. An event that causes suspension of normal **program execution.**

execution. The process of carrying out an **instruction** or the instructions of a **computer program** by a computer. (ISO)

execution time. (1) The amount of actual or central processor time used in executing a **program.**

(2) The period of time during which a **program** is executing.
See also **run time.**

execution time theory. A theory that uses cumulative **execution time** as the basis for estimating **software reliability.**

executive program. See **supervisory program.**

exit. (1) Any **instruction** in a **computer program,** in a **routine,** or in a **subroutine,** after the

execution of which control is no longer exercised by that computer program, that routine, or that subroutine. (ISO)

(2) The point beyond which control is no longer exercised by a **routine.**

failure. (1) The termination of the ability of a **functional unit** to perform its required **function.** (ISO)

(2) The inability of a **system** or system **component** to perform a required **function** within specified limits. A failure may be produced when a **fault** is encountered.

(3) A departure of **program** operation from program **requirements.**

failure category. See **error category.**

failure data. See **error data.**

failure rate. (1) The ratio of the number of **failures** to a given unit of measure; for example, failures per unit of time, failures per number of transactions, failures per number of **computer** runs.

(2) In **reliability** modeling, the ratio of the number of **failures** of a given category or severity to a given period of time; for example, failures per second of **execution time,** failures per month. Synonymous with **failure ratio.**

failure ratio. See **failure rate.**

failure recovery. The return of a **system** to a reliable operating state after **failure.**

fault. (1) An accidental condition that causes a **functional unit** to fail to perform its required **function.** (ISO)

(2) A manifestation of an **error(2)** in **software.** A fault, if encountered, may cause a **failure.** Synonymous with **bug.**

fault category. See **error category.**

fault insertion. See **fault seeding.**

fault seeding. The process of intentionally adding a known number of **faults** to those already in a **computer program** for the purpose of estimating the number of **indigenous faults** in the **program.** Synonymous with **bug seeding.**

fault tolerance. The built-in capability of a **system** to provide continued correct **execution** in the presence of a limited number of **hardware** or **software faults.**

file. A set of related **records** treated as a unit. (ISO) See also **logical file.**

finite state machine. A computational **model** consisting of a finite number of states, and transitions between these states.

firmware. (1) **Computer programs** and **data** loaded in a class of memory that cannot be dynamically modified by the **computer** during processing. See also **microcode, microprogram.**

(2) **Hardware** that contains a **computer program** and **data** that cannot be changed in its user environment. The computer programs and data contained in firmware are classified as **software;** the circuitry containing the computer program and data is classified as **hardware.**

(3) **Program instructions** stored in a read-only storage.

(4) An assembly composed of a **hardware** unit and a **computer program** integrated to form a functional entity whose **configuration** cannot be altered during normal operation. The computer program is stored in the hardware unit as an integrated circuit with a fixed logic configuration that will satisfy a specific application or operational **requirement.**

flag. (1) An indicator that signals the occurrence of an **error,** state, or other specified condition.

(2) Any of various types of indicators used for identification; for example, a **word** mark. (ANSI)

(3) A character that signals the occurrence of some condition, such as the end of a **word.** (ANSI)

(4) To indicate an **error,** state, or other specified condition in a **program.**

flow of control. The sequence of operations performed in the **execution** of an **algorithm.**

flowchart. A graphical representation of the definition, analysis, or solution of a problem in which symbols are used to represent operations, **data,** flow, and equipment. Contrast with **block diagram.** (ISO)

formal language. A language whose rules are explicitly established prior to its use. Synonymous with **artificial language.** Examples include **programming languages,** such as FORTRAN and Ada, and mathematical or logical languages, such as predicate calculus. Contrast with **natural language.**

formal parameter. A **variable** used in a **subprogram** to represent **data** or **program** elements to be transmitted to the subprogram by a calling **routine.** Synonymous with **dummy parameter.** Contrast with **actual parameter.**

formal specification. (1) A **specification** written and approved in accordance with established standards.

(2) In **proof of correctness,** a description in a **formal language** of the externally visible behavior of a **system** or system **component.**

formal testing. The process of conducting **testing** activities and reporting results in accordance with an approved **test plan.**

function. (1) A specific purpose of an entity or its characteristic action. (ANSI)

(2) A **subprogram** that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation. Contrast with **subroutine.**

functional decomposition. A method of designing a **system** by breaking it down into its **components** in such a way that the components correspond directly to system **functions** and subfunctions. See also **hierarchical decomposition.**

functional design. The **specification** of the working relationships among the parts of a **data processing system.** (ISO) See also **preliminary design.**

functional requirement. A **requirement** that specifies a **function** that a **system** or system **component** must be capable of performing.

functional specification. A **specification** that defines the **functions** that a **system** or system **component** must perform. See also **performance specification.**

functional unit. An entity of **hardware, software,** or both capable of accomplishing a specified purpose. (ISO)

graph. A **model** consisting of a finite set of **nodes** having connections called edges or arcs.

hardware. Physical equipment used in **data** processing, as opposed to **computer programs, procedures,** rules, and associated **documentation.** Contrast with **software.** (ISO)

hierarchical decomposition. A method of designing a **system** by breaking it down into its **components** through a series of **top-down** refinements. See also **functional decomposition, modular decomposition, stepwise refinement.**

hierarchy. A structure whose **components** are ranked into **levels** of subordination according to a specific set of rules.

high level language. Synonymous with **higher order language.**

higher order language. A **programming language** that usually includes features such as nested expressions, user defined **data types,** and **parameter** passing not normally found in lower order languages, that does not reflect the structure of any one given **computer** or class of computers, and that can be used to write machine independent **source programs.** A single, higher-order, language statement may represent multiple machine operations. Contrast with **machine language, assembly language.**

host machine. (1) The **computer** on which a **program** or **file** is installed.

(2) A **computer** used to develop **software** intended for another computer. Contrast with **target machine.**

(3) A **computer** used to **emulate** another computer. Contrast with **target machine.**

(4) In a **computer network,** a **computer** that provides processing capabilities to users of the **network.**

identifier. (1) A symbol used to name, indicate, or locate. Identifiers may be associated with such things as **data structures, data** items, or **program** locations.

(2) A character or group of characters used to identify or name an item of **data** and possibly to indicate certain properties of that data. (ISO)

imperfect debugging. In **reliability** modeling, the assumption that attempts to correct or remove a detected **fault** are not always successful.

implementation. (1) A realization of an **abstraction** in more concrete terms; in particular, in terms of **hardware, software,** or both.

(2) A machine executable form of a **program,** or a form of a program that can be translated automatically to machine executable form.

(3) The process of translating a **design** into **code** and **debugging** the code.

implementation phase. The period of time in the **software life cycle** during which a **software product** is created from **design documentation** and debugged. See also **installation and checkout phase, test phase.**

implementation requirement. Any **requirement** that impacts or constrains the **implementation** of a **software design;** for example, design descriptions, software development standards, **programming language** requirements, software **quality assurance** standards.

independent verification and validation. (1) **Verification** and **validation** of a **software product** by an organization that is both technically and managerially separate from the organization responsible for developing the product.

(2) **Verification** and **validation** of a **software product** by individuals or groups other than those who performed the original **design,** but who may be from the same organization. (10 CFR 50). The degree of independence must be a function of the importance of the **software.**

indigenous fault. A **fault** existing in a **computer program** that has not been inserted as part of a **fault seeding** process.

inductive assertion method. A **proof of correctness** technique in which **assertions** are written describing **program** inputs, outputs, and intermediate conditions, a set of theorems is developed relating satisfaction of the **input assertions** to satisfaction of the **output assertions,** and the theorems are proved to be true.

information hiding. The technique of **encapsulating software design** decisions in **modules** in such a way that the module's **interfaces** reveal as little as possible about the module's inner workings; thus, each module is a "black box" to the other modules in the **system.** The discipline of information hiding forbids the use of information about a module that is not in the module's **interface specification.** See also **encapsulation.**

input assertion. A logical expression specifying one or more conditions that **program** inputs must satisfy in order to be valid.

inspection. (1) A formal evaluation technique in which **software requirements, design,** or **code** are examined in detail by a person or group other than the author to detect **faults,** violations of development standards, and other problems. Contrast with **walk-through.** See also **code audit.**

(2) A phase of quality control that by means of examination, observation or measurement determines the conformance of materials, supplies, components, parts, appurtenances, **systems, processes** or structures to predetermined **quality requirements.** (ANSI N45.2.10-1973)

installation and checkout phase. The period of time in the **software life cycle** during which a **software product** is integrated into its **operational** environment and tested in this environment to ensure that it performs as required.

instruction. (1) A **program** statement that causes a **computer** to perform a particular operation or set of operations.

(2) In a **programming language,** a meaningful expression that specifies one operation and identifies its **operands,** if any. (ISO)

instruction set. The set of **instructions** of a **computer,** of a **programming language,** or of the programming languages in a programming **system.** (ISO)

instruction set architecture. An **abstract machine** characterized by an **instruction set.**

instruction trace. See **trace.**

instrumentation. See **program instrumentation.**

instrumentation tool. A **software tool** that generates and inserts counters or other probes at strategic points in another **program** to provide statistics about program **execution,** such as how thoroughly the program's **code** is exercised.

integration. The process of combining **software** elements, **hardware** elements, or both into an overall **system.**

integration testing. An orderly progression of **testing** in which **software** elements, **hardware** elements, or both are combined and tested until the entire **system** has been integrated. See also **system testing.**

integrity. The extent to which unauthorized access to or **modification** of **software** or **data** can be controlled in a **computer system.** See also **security.**

interactive. Pertaining to a **system** in which each user entry causes a response from the system. See also **conversational.**

interface. (1) A shared boundary. An interface might be a **hardware component** to link two devices or it might be a portion of storage or registers accessed by two or more **computer programs.** (ANSI)

(2) To interact or communicate with another **system component.**

interface requirement. A **requirement** that specifies a **hardware, software,** or **data base** element with which a **system** or system **component** must **interface,** or that sets forth constraints on formats, timing, or other factors caused by such an interface.

interface specification. A **specification** that sets forth the **interface requirements** for a **system** or system **component.**

interface testing. Testing conducted to ensure that **program** or **system components** pass information or control correctly to one another.

interoperability. The ability of two or more **systems** to exchange information and to mutually use the information that has been exchanged. Compare with **compatibility.**

interpret. To translate and to execute each **source language** statement of a **computer program** before translating and executing the next statement. (ISO) Contrast with **assemble, compile.**

interpreter. (1) **Software, hardware,** or **firmware** used to **interpret computer programs.** Contrast with **compiler, assembler.**

(2) A **computer program** used to **interpret.** (ISO)

interrupt. A suspension of a **process** such as the **execution** of a **computer program,** caused by an event external to that process, and performed in such a way that the process can be resumed. Synonymous with interruption. (ISO)

iteration. (1) The process of repeatedly executing a given sequence of **programming language** statements until a given condition is met or while a given condition is true.

(2) A single execution of a **loop.**

kernel. (1) A nucleus or core, as in the kernel of an **operating system.**

(2) An **encapsulation** of an elementary **function.** Kernels can be combined to form some or all of an **operating system** or set of **firmware.**

(3) A **model** used in **computer** selection studies to evaluate computer **performance.**

key. One or more characters, within a set of **data,** that contains information about the set, including its identification. (ISO)

label. (1) One or more characters, within or attached to a set of **data,** that contain information about the set, including its identification. (ISO)

(2) In **computer** programming, an **identifier** of an **instruction.** (ISO)

(3) An identification **record** for a tape or disk **file.**

language processor. (1) A **computer program** that performs such **functions** as translating, interpreting, and other tasks required for processing a specified **programming language;** for example, a FORTRAN processor, a COBOL processor. (ISO)

(2) A **software tool** that performs such **functions** as translating, **interpreting,** and other

tasks required for processing a specific language, such as a **requirements specification language,** a **design language,** or a **programming language.**

level. (1) The degree of subordination of an item in a hierarchical arrangement. (ISO)

(2) A rank within a **hierarchy.** An item is of the lowest level if it has no subordinates and of the highest level if it has no superiors.

level of documentation. A description of required **documentation** indicating its scope, content, format, and **quality.** Selection of the level may be based on project cost, intended usage, extent of effort, or other factors.

librarian. See **software librarian.**

library. See **software library, system library.**

life cycle. See **software life cycle.**

linkage editor. A **computer program** used to create one **load module** from one or more independently translated object modules or load modules by resolving cross-references among the object modules, and possibly by relocating elements. (ANSI) Note that not all object modules require linking prior to execution.

linked list. See **chained list.**

list. (1) An ordered set of items of **data.** (ISO)

(2) To print or otherwise display items of **data** that meet specified criteria. (ANSI)

(3) See **chained list.**

list processing. A method of processing **data** in the form of **lists.** Usually, **chained lists** are used so that the logical order of items can be changed without altering their physical locations. (ISO)

listing. (1) A **computer** output in the form of a human-readable **list.**

(2) A human-readable, textual **computer** output.

load map. A **computer** generated **list** that identifies the location or size of all or selected parts of a memory-resident **computer program** or of memory-resident **data.**

load module. A **program** unit that is suitable for loading into main storage for **execution;** it is usually the output of a **linkage editor.** (ISO)

loader. (1) A **routine** that reads an **object program** into main storage prior to its **execution.**

(2) A **routine,** commonly a **computer program,** that reads **data** into main storage. (ANSI)

logical file. A **file** independent of its physical environment. Portions of the same logical file may be located in different physical files, or several logical files or parts of logical files may be located in one physical file.

logical record. A **record** independent of its physical environment. Portions of the same logical record may be located in different physical records, or several logical records or parts of logical records may be located in one physical record. (ANSI)

loop. A set of **instructions** that may be executed repeatedly while a certain condition prevails. (ISO) See also **iteration.**

machine language. A representation of **instructions** and **data** that is directly executable by a **computer.** Contrast with **assembly language, higher order language.**

macro. (1) A predefined sequence of **instructions** that is inserted into a **program** during assembly or compilation at each place that its corresponding **macroinstruction** appears in the program.

(2) Synonymous with **macroinstruction.**

macroinstruction. An **instruction** in a **source language** that is to be replaced by a defined sequence of instructions in the same source language. The macroinstruction may also specify values for **parameters** in the instructions that are to replace it. (ISO)

macroprocessor. The portion of some **assemblers** and **compilers** that allows a programmer to define and use **macros.**

maintainability. (1) The ease with which **software** can be maintained.

(2) The ease with which **maintenance** of a **functional unit** can be performed in accordance with prescribed **requirements.** (ISO)

(3) Ability of an item under stated conditions of use to be retained in, or restored to, within a given period of time, a specified state in which it can perform its required **functions** when **maintenance** is performed under stated conditions and while using prescribed **procedures** and resources. (ANSI/ASQC A3-1978).

maintenance. See **software maintenance.**

maintenance phase. See **operation and maintenance phase.**

maintenance plan. A **document** that identifies the management and technical approach that will be used to maintain **software products.** Typically included are topics such as **tools,** resources, facilities, and schedules.

map program. A **compiler** or **assembler** feature that generates a **load map.**

master library. A **software library** containing formally released versions of **software** and **documentation.** Contrast with **production library.**

metacompiler. See **compiler generator.**

metalanguage. A language used to specify a language or languages.

microcode. (1) A symbolic representation of a **microprogram.**

(2) The internal representation of a **microprogram** in its storage medium. See also **firmware.**

microprogram. A sequence of elementary **instructions** that corresponds to a **computer** operation, that is maintained in special storage, and whose **execution** is initiated by the introduction of a computer instruction into an instruction register of a computer. (ISO) Microprograms are often used in place of hard-wired logic. See also **firmware.**

milestone. A scheduled event for which some project member or manager is held accountable and that is used to measure progress; for example, a formal review, issuance of a **specification,** product **delivery.**

mnemonic symbol. A symbol chosen to assist the human memory; for example, an abbreviation such as *"mpy"* for *"multiply"*. (ISO)

model. A representation of a real world **process,** device, or concept. See also **analytical model, availability model, debugging model, error model, reliability model, simulation, statistical test model.**

modification. (1) A change made to **software.**

(2) The process of changing **software.**

modular decomposition. A method of designing a **system** by breaking it down into **modules.** See also **hierarchical decomposition.**

modular programming. A technique for developing a **system** or **program** as a collection of **modules.**

modularity. The extent to which **software** is composed of discrete **components** such that a change to one component has minimal impact on other components.

module. (1) A **program** unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from, an **assembler, compiler, linkage editor,** or executive **routine.** (ANSI)

(2) A logically separable part of a **program.**

module strength. See **cohesion.**

multilevel security. A mode of operation permitting **data** at various **security levels** to be concurrently stored and processed in a **computer system,** when at least some users have neither the clearance nor the need-to-know for all data contained in the system.

multiprogramming. (1) A mode of operation that provides for the interleaved **execution** of two or more **computer programs** by a single processor. (ISO)

(2) Pertaining to the concurrent **execution** of two or more **computer programs** by a computer. (ANSI)

(3) The concurrent **execution** of two or more **functions** as though each function operates alone.

mutation. See **program mutation.**

N-ary. (1) Characterized by a selection, choice, or condition that has n possible different values or states. (ISO)

(2) Of a fixed radix numeration system, having a radix of n. (ISO)

natural language. A language whose rules are based on current usage without being explicitly prescribed. (ISO) Examples include English, Chinese, French, and Swahili. Contrast with **formal language.**

nest. (1) To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one **loop** (the nested loop) within another loop (the nesting loop); to nest one **subroutine** (the nested subroutine) within another subroutine (the nesting subroutine). (ISO)

(2) To place **subroutines** or **data** in other subroutines or data at a different hierarchical **level** so that subroutines can be executed as recursive subroutines or so that the data can be accessed recursively.

network. (1) An interconnected or interrelated group of **nodes.**

(2) In connection with a disciplinary or problem oriented qualifier, the combination of material, **documentation,** and human resources that are united by design to achieve certain objectives; for example, a social science network, a science information network.

node. (1) An end point of any branch of a **network** or **graph,** or a junction common to two or more branches.

(2) In a tree structure, a point at which subordinate items of **data** originate. (ANSI)

(3) In a **network,** a point where one or more **functional units** interconnect transmission lines. (ISO)

(4) The representation of a state or an event by means of a point on a diagram. (ANSI)

object program. A fully **compiled** or **assembled program** that is ready to be loaded into the **computer.** (ISO) Contrast with **source program.**

operand. (1) An entity on which an operation is performed. (ISO)

(2) That which is operated upon. An operand is usually identified by an **address** part of an **instruction.** (ANSI)
See also **operator.**

operating system. Software that controls the **execution** of **programs.** An operating system may provide services such as resource allocation scheduling, input/output control, and **data** management. Although operating systems are predominantly software, partial or complete **hardware implementations** are possible. (ISO) An operating system provides support in a single spot rather than forcing each program to be concerned with controlling hardware. See also **system software.**

operation and maintenance phase. The period of time in the **software life cycle** during which a **software product** is employed in its **operational** environment, monitored for satisfactory **performance,** and modified as necessary to correct problems or to respond to changing **requirements.**

operational. Pertaining to the status given a **software product** once it has entered the **operation and maintenance phase.**

operational reliability. The **reliability** of a **system** or **software subsystem** in its actual use environment. Operational reliability may differ considerably from reliability in the specified or test environment.

operational testing. Testing performed by the end user on **software** in its normal operating environment. (DoD usage)

operator. (1) In symbol manipulation, a symbol that represents the action to be performed in an operation. (ISO) Examples of operators are +,-,*,/.

(2) In the description of a **process,** that which indicates the action to be performed on **operands.** (ANSI)

(3) A person who operates a machine. (ANSI)
See also **operand.**

output assertion. A logical expression specifying one or more conditions that **program** outputs must satisfy in order for the program to be correct.

overlay. (1) In a **computer program,** a **segment** that is not permanently maintained in internal storage. (ISO)

(2) The technique of repeatedly using the same areas of internal storage during different stages of a **program.** (ANSI)

(3) In the **execution** of a **computer program,** to load a **segment** of the computer program in a storage area hitherto occupied by parts of the computer program that are not currently needed. (ISO)

parameter. (1) A **variable** that is given a constant value for a specified application and that may denote the application. (ISO)

(2) A **variable** that is used to pass values between **program routines.** See also **actual parameter, formal parameter.**

parse. To determine the syntactic structure of an **artificial** or **natural language** unit by decomposing the unit into more elementary subunits and establishing the relationships among the subunits; for example, **blocks,** statements, and expressions may be decomposed into statements, expressions, and **operators** and **operands.**

partial correctness. In **proof of correctness,** a designation indicating that a **program's output assertions** follow logically from its **input assertions** and processing steps. Contrast with **total correctness.**

patch. (1) A **modification** to an **object program** made by replacing a portion of existing machine **code** with modified machine code.

(2) To modify an **object program** without recompiling the **source program.**

path analysis. Program analysis performed to identify all possible paths through a program, to detect incomplete paths, or to discover portions of the program that are not on any path.

path condition. A set of conditions that must be met in order for a particular **program** path to be executed.

path expression. A logical expression indicating the input conditions that must be met in order for a particular **program** path to be executed.

perfective maintenance Maintenance performed to improve **performance, maintainability,** or other **software** attributes. See also **adaptive maintenance, corrective maintenance.**

performance. (1) The ability of a **computer system** or **subsystem** to perform its **functions.**

(2) A measure of the ability of a **computer system** or **subsystem** to perform its **functions;** for example, response time, **throughput,** number of transactions. See also **performance requirement.**

performance evaluation. The technical assessment of a **system** or system **component** to determine how effectively operating objectives have been achieved.

performance requirement. A **requirement** that specifies a **performance** characteristic that a **system** or system **component** must possess; for example, speed, **accuracy,** frequency.

performance specification. (1) A **specification** that sets forth the **performance requirements** for a **system** or system **component.**

(2) Synonymous with **requirements specification.** (U.S. Navy usage)
See also **functional specification.**

Petri net. An abstract, formal **model** of information flow, showing static and dynamic properties of a **system.** A Petri net is usually represented as a **graph** having two types of **nodes** (called places and transitions) connected by arcs, and markings (called tokens) indicating dynamic properties. See also **state diagram.**

physical requirement. A **requirement** that specifies a physical characteristic that a **system** or system **component** must possess; for example, material, shape, size, weight.

pointer. (1) An **identifier** that indicates the location of an item of **data.** (ANSI)

(2) A **data** item whose value is the location of another data item.

portability. The ease with which **software** can be transferred from one **computer system** or environment to another.

precision. (1) A measure of the ability to distinguish between nearly equal values; for example, four-place numerals are less precise than six-place numerals; nevertheless, a properly computed four-place numeral may be more accurate than an improperly computed six-place numeral. (ISO)

(2) The degree of discrimination with which a quantity is stated; for example, a three-digit numeral discriminates among 1000 possibilities. (ISO)
Contrast with **accuracy.**

precompiler. A **computer program** that preprocesses source **code,** part of which may be unacceptable to a **compiler,** to generate equivalent code acceptable to the compiler; for example, a **preprocessor** which converts structured FORTRAN to ANSI-standard FORTRAN.

preliminary design. (1) The process of analyzing **design** alternatives and defining the **software architecture.** Preliminary design typically includes definition and structuring of **computer program components** and **data,** definition of the **interfaces,** and preparation of timing and sizing estimates.

(2) The result of the preliminary design process. See also **design analysis, functional design.**

preprocessor. A **computer program** that effects some preliminary computation or organization. (ISO) See also **precompiler.**

privileged instruction. An **instruction** that may be used only by a **supervisory program.** (ISO)

procedure. (1) A portion of a **computer program** which is named and which performs a specific task. Compare with **subroutine, subprogram, function, module.**

(2) The course of action taken for the solution of a problem. (ISO)

(3) The description of the course of action taken for the solution of a problem. (ANSI)

(4) A set of manual steps to be followed to accomplish a task each time the task is to be done.

process. (1) In a **computer system,** a unique, finite course of events defined by its purpose or by its effect, achieved under given conditions. (ISO)

(2) To perform operations on **data** in process. (ISO)

product certification. See **certification.**

product specification. Synonymous with **design specification.** (DoD usage)

production library. A **software library** containing **software** approved for current **operational** use.

program. (1) A **computer program.**

(2) A schedule or plan that specifies actions to be taken.

(3) To **design,** write, and test **computer programs.** (ISO)

program architecture. The structure and relationships among the **components** of a **computer program.** The program architecture may also include the program's **interface** with its operational environment.

program block. In problem-oriented languages, a **computer program** subdivision that serves to group related statements, delimit **routines,** specify storage allocation, delineate the applicability of **labels,** or **segment** paths of the computer program for other purposes. (ANSI)

program correctness. See **correctness.**

program design language. See **design language.**

program extension. An **enhancement** made to existing **software** to increase the scope of its capabilities.

program instrumentation. (1) Probes, such as **instructions** or **assertions,** inserted into a **computer program** to facilitate **execution** monitoring, **proof of correctness,** resource monitoring, or other activities.

(2) The process of preparing and inserting probes into a **computer program.**

program library. An organized collection of **computer programs.** (ISO) See also **software library, system library.**

program mutation. (1) A **program** version purposely altered from the intended version to evaluate the ability of program **test cases** to detect the alteration. Synonymous with program mutant.

(2) The process of creating program mutations in order to evaluate the adequacy of **program test data.**

program protection. The application of internal or external controls to preclude any unauthorized access or **modification** to a **computer program.**

program specification. (1) Any **specification** for a **computer program.** See **design specification, functional specification, performance specification, requirements specification.**

(2) Synonymous with **design specification.**

program support library. See **software development library.**

program synthesis. The use of **software tools** to aid in the transformation of a **program specification** into a **program** that realizes that **specification.**

program validation. Synonymous with **computer program validation.** See **validation.**

programming language. An **artificial language** designed to generate or express **programs.** (ISO)

programming support environment. An integrated collection of **tools** accessed via a single **command language** to provide programming support capabilities throughout the **software life cycle.** The environment typically includes tools for designing, editing, compiling, loading, **testing, configuration management,** and project management.

project file. See **project notebook.**

project notebook. A central repository of written material such as memos, plans, technical reports, etc., pertaining to a project. Synonymous with **project file.** See also **software development notebook.**

project plan. A management **document** describing the approach that will be taken for a project. The plan typically describes the work to be done, the resources required, the methods to be used, the **configuration management** and **quality assurance** procedures to be followed, the schedules to be met, the project organization, etc.

prompt. (1) A message informing a user that a **system** is ready for the next command, message, or other user action.

(2) To inform a user that a **system** is ready for the next command, element, or other input.

proof of correctness. (1) A formal technique used to prove mathematically that a **program** satisfies its **specifications.** See also **partial correctness, total correctness.**

(2) A **program** proof that results from applying this technique.

protection. An arrangement for restricting access to or use of all, or part, of a **computer system.** (ISO)

protocol. (1) A set of conventions or rules that govern the interactions of **processes** or applications within a **computer system** or **network.**

(2) A set of rules that govern the operation of **functional units** to achieve communication. (ISO)

pseudo code. A combination of **programming language** and **natural language** used for **computer program design.**

pushdown storage. A storage device that handles **data** in such a way that the next item to be retrieved is the most recently stored item still in the storage device; i.e., last-in-first-out (LIFO). (ISO) See also **stack.**

qualification testing. Formal testing, usually conducted by the developer for the customer, to demonstrate that the **software** meets its specified **requirements.** See also **acceptance testing, system testing.**

quality. (1) The totality of features and characteristics of a product or service that bears on its ability to satisfy given needs. (ANSI/ASQC A3-1978)

(2) See **software quality.**

quality assurance. A planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical **requirements.** (ANSI/IEEE Std 730-1981)

quality metric. A quantitative measure of the degree to which **software** possesses a given attribute that affects its **quality.**

queue. A **list** that is accessed in a first-in, first-out manner. Contrast with **stack.**

real time. (1) Pertaining to the processing of **data** by a **computer** in connection with another **process** outside the computer according to time requirements imposed by the outside process. This term is also used to describe **systems** operating in **conversational** mode, and processes that can be influenced by human intervention while they are in progress. (ISO)

(2) Pertaining to the actual time during which a physical **process** transpires; for example, the performance of a computation during the actual time that the related physical process transpires, in order that results of the computation can be used in guiding the physical process. (ANSI)

record. A collection of related **data** or **words** treated as a unit. (ISO) See also **logical record.**

recursive routine. A **routine** that may be used as a **subroutine** of itself, calling itself directly or being called by another subroutine, one that it itself has called. The use of a recursive routine usually requires the keeping of records of the status of its unfinished uses in, for example, a pushdown **list.**

redundancy. The inclusion of duplicate or alternate **system** elements to improve **operational reliability** by ensuring continued operation in the event that a primary element fails.

regression testing. Selective retesting to detect **faults** introduced during **modification** of a **system** or system **component,** to verify that modifications have not caused unintended adverse effects, or to verify that a modified system or system component still meets its specified **requirements.**

reliability. (1) The ability of an item to perform a required **function** under stated conditions for a stated period of time. (ANSI/ASQC A3-1978)

(2) See **software reliability**.

reliability, numerical. The probability that an item will perform a required **function** under stated conditions for a stated period of time. (ANSI/ASQC A3-1978)

reliability assessment. The process of determining the achieved level of **reliability** of an existing **system** or system **component.**

reliability data. Information necessary to assess the **reliability** of **software** at selected points in the **software life cycle.** Examples include **error data** and time data for **reliability models, program** attributes such as **complexity,** and programming characteristics such as development techniques employed and programmer experience.

reliability evaluation. See **reliability assessment.**

reliability growth. The improvement in **software reliability** that results from correcting **faults** in the **software.**

reliability model. A **model** used for predicting, estimating, or assessing **reliability.** See also **reliability assessment.**

relocatable machine code. Machine language code that requires relative **addresses** to be translated into absolute addresses prior to **computer execution.** Contrast with **absolute machine code.**

rendezvous. The interaction that occurs between two parallel tasks when one task has called an entry of the other task, and a corresponding accept statement is being executed by the other task on behalf of the calling task.

repeatability. See **test repeatability.**

requirement. (1) A condition or capability needed by a user to solve a problem or achieve an objective.

(2) A condition or capability that must be met or possessed by a **system** or system **component** to satisfy a contract, standard, **specification,** or other formally imposed **document.** The set of all requirements forms the basis for subsequent development of the system or system component. See also **requirements analysis, requirements phase, requirements specification.**

requirements analysis. (1) The process of studying user needs to arrive at a definition of **system** or **software requirements.**

(2) The **verification** of **system** or **software requirements.**

requirements inspection. See **inspection.**

requirements phase. The period of time in the **software life cycle** during which the **requirements** for a **software product,** such as the functional and **performance** capabilities, are defined and documented.

requirements specification. A **specification** that sets forth the **requirements** for a **system** or system **component;** for example, a **software configuration item.** Typically included are **functional requirements, performance requirements, interface requirements, design requirements,** and development standards.

requirements specification language. A **formal language** with special constructs and **verification** protocols used to specify, verify, and **document requirements.**

requirements verification. See **verification.**

retirement phase. The period of time in the **software life cycle** during which support for a **software product** is terminated.

reusability. The extent to which a **module** can be used in multiple applications.

review. See **design review.**

robustness. The extent to which **software** can continue to operate correctly despite the introduction of invalid inputs.

root compiler. A **compiler** whose output is a machine independent, intermediate-level representation of a **program.** A root compiler, when combined with a machine-dependent **code generator,** comprises a full compiler.

routine. (1) A **computer program segment** that performs a specific task. See also **function, procedure, subroutine, subprogram.**

(2) A **program,** or a sequence of **instructions** called by a program, that may have some general or frequent use. (ISO)

run time. (1) A measure of the time expended to execute a **program.** While run time ordinarily reflects the expended central processor time, run time may also include peripheral processing and peripheral accessing time; for example, a run time of 5 hours.

(2) The instant at which a **program** begins to execute.
See also **execution time.**

secretary/librarian. The **software librarian** on a **chief programmer team.**

security. The **protection** of **computer hardware** and **software** from accidental or malicious access, use, **modification,** destruction, or disclosure. Security also pertains to personnel, **data,** communications, and the physical protection of computer installations.

security kernel. A small, self-contained collection of key **security**-related statements that works as a privileged part of an **operating system.** All criteria specified by the **kernel** must be met for a **program** or **data** to be accessed.

seeding. See **fault seeding.**

segment. (1) A self-contained portion of a **computer program** that may be executed without the entire **computer program** necessarily being maintained in internal storage at any one time. (ISO) See also **component, module, subprogram.**

(2) The sequence of **computer program** statements between two consecutive branch points. See also **path analysis.**

(3) To divide a **computer program** into segments. (ISO)

semantics. (1) The relationships of characters or groups of characters to their meanings, independent of the manner of their interpretation and use. (ISO)

(2) The relationships between symbols and their meanings. (ANSI)

(3) The discipline of expressing the meanings of **computer** language constructs in **metalanguages.**
See also **syntax.**

semaphore. A shared **variable** used to synchronize **concurrent processes** by indicating whether an action has been completed or an event has occurred.

sequential processes. Processes that execute in such a manner that one must finish before the next begins. Contrast with **concurrent processes.**

severity. See **criticality.**

side-effect. Processing or activities performed, or results obtained, secondary to the primary **function** of a **program, subprogram,** or operation.

simulation. The representation of selected characteristics of the behavior of one physical or abstract **system** by another system. In a digital **computer system,** simulation is done by **software;** for example, (a) the representation of physical phenomena by means of operations performed by a computer system, (b) the representation of operations of a computer system by those of another computer system. (ISO) Contrast with **analytical model.**

simulator. A device, **data** processing **system,** or **computer program** that represents certain features of the behavior of a physical or abstract system. (ANSI)

sizing. The process of estimating the amount of **computer** storage or the number of source lines that will be required for a **system** or system **component.**

software. (1) **Computer programs, procedures,** rules, and possibly associated **documentation** and **data** pertaining to the operation of a **computer system.** See also **application software, system software.**

(2) **Programs, procedures,** rules, and any associated **documentation** pertaining to the operation of a **computer system.** (ISO)
Contrast with **hardware.**

software configuration management. See **configuration management.**

software data base. A centralized **file** of **data** definitions and present values for data common to, and located internal to, an **operational software system.**

software development cycle. (1) The period of time that begins with the decision to develop a **software product** and ends when the product is delivered. This cycle typically includes a **requirements phase, design phase, implementation phase, test phase,** and sometimes, **installation** and **checkout phase.** Contrast with **software life cycle.**

(2) The period of time that begins with the decision to develop a **software product** and ends when the product is no longer being enhanced by the developer.

(3) Sometimes used as a synonym for **software life cycle.**

software development library. A **software library** containing **computer** readable and human readable information relevant to a **software** development effort.

software development notebook. A collection of material pertinent to the development of a given **software module.** Contents typically include the **requirements, design,** technical reports, **code listings, test plans,** test results, problem reports, schedules, notes, etc., for the module. See also **project notebook.**

software development plan. A **project plan** for the development of a **software product.** Synonymous with **computer program development plan.**

software development process. The process by which user needs are translated into **software requirements,** software requirements are transformed into **design,** the design is implemented in **code,** and the code is tested, documented, and certified for **operational** use.

software documentation. Technical **data** or information, including **computer listings** and printouts, in human-readable form, that describe or specify the **design** or details, explain the capabilities, or provide operating instructions for using the **software** to obtain desired results from a software **system.** See also **documentation, system documentation, user documentation.**

software engineering. The systematic approach to the development, operation, **maintenance,** and retirement of **software.**

software experience data. Data relating to the development or use of **software** that could be useful in developing **models, reliability** predictions, or other quantitative descriptions of software.

software librarian. The person responsible for establishing, controlling, and maintaining a **software library.**

software library. A controlled collection of **software** and related **documentation** designed to aid in software development, use, or **maintenance.** Types include **software development library, master library, production library, program library,** and **software repository.** See also **system library.**

software life cycle. The period of time that starts when a **software product** is conceived and ends when the product is no longer available for use. The software life cycle typically includes a **requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase,** and sometimes, **retirement phase.** Contrast with **software development cycle.**

software maintenance. (1) **Modification** of a **software product** after **delivery** to correct **faults.**

(2) **Modification** of a **software product** after **delivery** to correct **faults,** to improve **performance** or other attributes, or to adapt the product to a changed environment. See also **adaptive maintenance, corrective maintenance, perfective maintenance.**

software monitor. A **software tool** that executes concurrently with another **computer program** and that provides detailed information about the **execution** of the other **program.**

software product. A **software** entity designated for **delivery** to a user.

software quality. (1) The totality of features and characteristics of a **software product** that bear on its ability to satisfy given needs; for example, conform to **specifications.**

(2) The degree to which **software** possesses a desired combination of attributes.

(3) The degree to which a customer or user perceives that **software** meets his or her composite expectations.

(4) The composite characteristics of **software** that determine the degree to which the software in use will meet the expectations of the customer.

software quality assurance. See **quality assurance.**

software reliability. (1) The probability that **software** will not cause the **failure** of a **system** for a specified time under specified conditions. The probability is a function of the inputs to and use of the system as well as a function of the existence of **faults** in the software. The inputs to the system determine whether existing faults, if any, are encountered.

(2) The ability of a **program** to perform a required **function** under stated conditions for a stated period of time.

software repository. A **software library** providing permanent, archival storage for **software** and related **documentation.**

software sneak analysis. A technique applied to **software** to identify latent (sneak) logic control paths or conditions that could inhibit a desired operation or cause an unwanted operation to occur.

software tool. A **computer program** used to help develop, test, analyze, or maintain another computer program or its **documentation;** for example, **automated design tool, compiler,** test **tool, maintenance tool.**

source language. (1) A language used to write **source programs.**

(2) A language from which statements are translated. (ISO) Contrast with **target language.**

source program. (1) A **computer program** that must be **compiled, assembled,** or **interpreted** before being executed by a **computer.**

(2) A **computer program** expressed in a **source language.** Contrast with **object program.** (ISO)

specification. (1) A **document** that prescribes, in a complete, precise, verifiable manner, the **requirements, design,** behavior, or other characteristics of a **system** or system **component.** See also **design specification, formal specification, functional specification, interface specification, performance specification, requirements specification.**

(2) The process of developing a specification.

(3) A concise statement of a set of **requirements** to be satisfied by a product, a material or **process** indicating, whenever appropriate, the **procedure** by means of which it may be determined whether the requirements given are satisfied. (ANSI N45.2.10-1973)

specification language. A language, often a machine-processable combination of **natural** and **formal language,** used to specify the **requirements, design,** behavior, or other characteristics of a **system** or system **component.** See also **design language, requirements specification language.**

specification verification. See **verification.**

stability. (1) The ability to continue unchanged despite disturbing or disruptive events.

(2) The ability to return to an original state after disturbing or disruptive events.

stack. A **list** that is accessed in a last-in, first-out manner. Contrast with **queue.**

standards enforcer. A **software tool** that determines whether prescribed development standards have been followed. The standards may include **module** size, module structure, commenting conventions, use of certain statement forms, and **documentation** conventions.

state diagram. A **directed graph** in which **nodes** correspond to internal states of a **system,** and edges correspond to transitions; often used for describing a system in terms of state changes. See also **Petri net.**

static analysis. The process of evaluating a **program** without executing the program. See also **desk checking, code audit, inspection, static analyzer, walk-through.** Contrast with **dynamic analysis.**

static analyzer. A **software tool** that aids in the evaluation of a **computer program** without executing the **program.** Examples include **syntax** checkers, **compilers,** cross-reference generators, **standards enforcers,** and flowcharters. Contrast with **dynamic analyzer.**

static binding. Binding performed prior to **execution** of a **program** and not subject to change during execution. Contrast with **dynamic binding.**

statistical test model. A **model** that relates **program faults** to the input **data** set (or sets) which cause them to be encountered. The model also gives the probability that these faults will cause the program to fail.

stepwise refinement. A **system development methodology** in which **data** definitions and processing steps are defined broadly at first and then with increasing detail. See also **hierarchical decomposition, top-down, bottom-up.**

string. A linear sequence of entities such as characters or physical elements. (ISO)

strong typing. A **programming language** feature that requires the **data type** of each **data** object to be declared, and that precludes the application of **operators** to inappropriate data objects and, thereby, prevents the interaction of data objects of incompatible **types.**

structured design. A disciplined approach to **software design** that adheres to a specified set of rules based on principles such as **top-down design, stepwise refinement,** and **data** flow analysis.

structured program. A **program** constructed of a basic set of **control structures,** each one having one entry point and one exit. The set of control structures typically includes: sequence of two or more **instructions,** conditional selection of one of two or more instructions or sequences of instructions, and repetition of an instruction or a sequence of instructions.

structured programming. (1) A well-defined **software** development technique that incorpor-

ates **top-down design** and **implementation** and strict use of **structured program** control constructs.

(2) Loosely, any technique for organizing and coding **programs** that reduces **complexity,** improves clarity, and facilitates **debugging** and **modification.**

structured programming language. A **programming language** that provides the **structured program** constructs and that facilitates the development of structured programs.

stub. (1) A dummy **program module** used during the development and **testing** of a higher-level module.

(2) A **program** statement substituting for the body of a program unit and indicating that the unit is or will be defined elsewhere.

subprogram. A **program** unit that may be invoked by one or more other program units. Examples are **procedure, function, subroutine.**

subroutine. (1) A sequenced set of statements that may be used in one or more **computer programs** and at one or more points in a computer program. (ISO)

(2) A **routine** that can be part of another routine. (ANSI)

(3) A **subprogram** that is invoked by a calling statement, that may or may not receive input values, and that returns any output values through **parameter** names, **program variables,** or mechanisms other than the subroutine name itself. Contrast with **function.** See also **procedure.**

subsystem. A group of assemblies or **components** or both combined to perform a single **function.** (ANSI N45.2.10-1973)

supervisor. See **supervisory program**

supervisory program. A **computer program,** usually part of an **operating system,** that controls the **execution** of other computer programs and regulates the flow of work in a **data** processing **system.** Synonymous with **executive program, supervisor.** (ISO)

symbolic execution. A **verification** technique in which **program execution** is simulated using symbols rather than actual values for input **data,** and program outputs are expressed as logical or mathematical expressions involving these symbols.

syntax. (1) The relationship among characters or groups of characters, independent of their meanings or the manner of their interpretation and use. (ISO)

(2) The structure of expressions in a language. (ANSI)

(3) The rules governing the structure of a language. (ANSI)

See also **semantics.**

system. (1) A collection of people, machines, and methods organized to accomplish a set of specific **functions.**

(2) An integrated whole that is composed of diverse, interacting, specialized structures and subfunctions.

(3) A group or **subsystem** united by some interaction or interdependence, performing many duties but functioning as a single unit. (ANSI N45.2.10-1973)

system architecture. The structure and relationship among the **components** of a **system.** The system architecture may also include the system's **interface** with its operational environment.

system design. (1) The process of defining the **hardware** and **software** architectures, **components, modules, interfaces,** and **data** for a **system** to satisfy specified system **requirements.**

(2) The result of the system design process.

system documentation. Documentation conveying the **requirements, design** philosophy, design details, capabilities, limitations, and other characteristics of a **system.** Contrast with **user documentation.**

system library. A controlled collection of **system**-resident **software** that can be accessed for use or incorporated into other **programs** by reference, for example, a group of **routines** that a **linkage editor** can incorporate into a program as required. See also **software library.**

system reliability. The probability that a **system,** including all **hardware** and **software subsys-**

tems, will perform a required task or mission for a specified time in a specified environment. See also **operational reliability, software reliability.**

system software. Software designed for a specific **computer system** or family of computer systems to facilitate the operation and **maintenance** of the computer system and associated **programs,** for example, **operating systems, compilers,** utilities. Contrast with **application software.**

system testing. The process of **testing** an integrated **hardware** and **software system** to verify that the system meets its specified **requirements.** See also **acceptance testing, qualification testing.**

system validation. See **validation.**

system verification. See **verification.**

table. (1) An array of **data,** each item of which may be unambiguously identified by means of one or more arguments. (ISO)

(2) A collection of **data** in which each item is uniquely identified by a **label,** by its position relative to the other items, or by some other means. (ANSI)

target language. A language into which source statements are translated. (ANSI) Contrast with **source language.**

target machine. (1) The **computer** on which a **program** is intended to operate.

(2) The **computer** being emulated by another computer.
Contrast with **host machine.**

termination proof. In **proof of correctness,** the demonstration that a **program** will terminate under all specified input conditions.

test bed. (1) A test environment containing the **hardware, instrumentation tools, simulators,** and other support **software** necessary for **testing** a **system** or system **component.**

(2) The repertoire of **test cases** necessary for **testing** a **system** or system **component.**

test case. A specific set of **test data** and associated **procedures** developed for a particular objective, such as to exercise a particular **program** path or to verify compliance with a specific **requirement.** See also **testing.**

test case generator. See **automated test generator.**

test data. Data developed to test a **system** or system **component.** See also **test case.**

test data generator. See **automated test generator.**

test driver. A driver that invokes the item under test and that may provide test inputs and report test results.

test log. A chronological record of all relevant details of a **testing** activity.

test phase. The period of time in the **software life cycle** during which the **components** of a **software product** are evaluated and integrated, and the software product is evaluated to determine whether or not **requirements** have been satisfied.

test plan. A **document** prescribing the approach to be taken for intended **testing** activities. The plan typically identifies the items to be tested, the testing to be performed, test schedules, personnel requirements, reporting requirements, evaluation criteria, and any risks requiring contingency planning.

test procedure. Detailed instructions for the setup, operation, and evaluation of results for a given test. A set of associated **procedures** is often combined to form a test procedures **document.**

test repeatability. An attribute of a test indicating whether the same results are produced each time the test is conducted.

test report. A **document** describing the conduct and results of the **testing** carried out for a **system** or system **component.**

test validity. The degree to which a test accomplishes its specified goal.

testability. (1) The extent to which **software** facilitates both the establishment of test criteria and

the evaluation of the software with respect to those criteria.

(2) The extent to which the definition of **requirements** facilitates analysis of the requirements to establish test criteria.

testing. The process of exercising or evaluating a **system** or system **component** by manual or automated means to verify that it satisfies specified **requirements** or to identify differences between expected and actual results. Compare with **debugging.**

throughput. A measure of the amount of work performed by a **computer system** over a period of time; for example, number of jobs per day. (ISO)

time sharing. (1) An operating technique of a **computer system** that provides for the interleaving in time of two or more **processes** in one processor. (ISO)

(2) Pertaining to the interleaved use of time on a computing **system** that enables two or more users to execute **computer programs** concurrently. (ANSI)

timing analyzer. A **software tool** that estimates or measures the **execution time** of a **computer program** or portions of a computer program either by summing the execution times of the **instructions** in each path, or by inserting probes at specific points in the **program** and measuring the execution time between probes.

tolerance. The ability of a **system** to provide continuity of operation under various abnormal conditions.

tool. (1) See **software tool.**

(2) A **hardware** device used to analyze **software** or its **performance.**

top-down. Pertaining to an approach that starts with the highest level **component** of a **hierarchy** and proceeds through progressively lower **levels;** for example, **top-down design,** top-down programming, **top-down testing.** Contrast with **bottom-up.**

top-down design. The process of designing a **system** by identifying its major **components,** decomposing them into their lower **level** components, and iterating until the desired level of detail is achieved. Contrast with **bottom-up design.**

top-down testing. The process of checking out hierarchically organized **programs,** progressively, from top to bottom, using **simulation** of lower level **components.**

total correctness. In **proof of correctness,** a designation indicating that a **program's output assertions** follow logically from its **input assertions** and processing steps, and that, in addition, the program terminates under all specified input conditions. Contrast with **partial correctness.**

trace. (1) A record of the **execution** of a **computer program;** it exhibits the sequences in which the **instructions** were executed. (ANSI)

(2) A record of all or certain classes of **instructions** or **program** events occurring during **execution** of a **computer program.**

(3) To produce a **trace.**

tracer. A **software tool** used to **trace.**

translator. A **program** that transforms a sequence of statements in one language into an equivalent sequence of statements in another language. See also **assembler, compiler, interpreter.**

tree. An abstract **hierarchical** structure consisting of **nodes** connected by branches, in which: (a) each branch connects one node to a directly subsidiary node, and (b) there is a unique node called the root that is not subsidiary to any other node, and (c) every node besides the root is directly subsidiary to exactly one other node.

type. See **data type.**

user documentation. Documentation conveying to the end user of a **system** instructions for using the system to obtain desired results; for example, a user's manual. Contrast with **system documentation.**

utility software. Computer programs or **routines** designed to perform some general support **function** required by other **application soft-**

ware, by the **operating system,** or by **system** users.

validation. The process of evaluating **software** at the end of the **software development process** to ensure compliance with software **requirements.** See also **verification.**

variable. (1) A quantity that can assume any of a given set of values. (ISO)

(2) In programming, a character or group of characters that refers to a value and, in the **execution** of a **computer program,** corresponds to an **address.** (ANSI)

verification. (1) The process of determining whether or not the products of a given phase of the **software development cycle** fulfill the **requirements** established during the previous phase. See also **validation.**

(2) Formal proof of **program** correctness. See **proof of correctness.**

(3) The act of reviewing, inspecting, **testing,** checking, auditing, or otherwise establishing and documenting whether or not items, **processes,** services, or **documents** conform to specified **requirements.** (ANSI/ASQC A3-1978).

verification system. See **automated verification system.**

virtual machine. A functional **simulation** of a **computer** and its associated devices.

virtual memory. See **virtual storage.**

virtual storage. The storage space that may be regarded as addressable main storage by the user of a **computer system** in which virtual **addresses** are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available and not by the actual number of main storage locations. (ISO)

walk-through. A review process in which a designer or programmer leads one or more other members of the development team through a **segment** of **design** or **code** that he or she has written, while the other members ask questions and make comments about technique, style, possible **errors,** violation of development standards, and other problems. Contrast with **inspection.**

word. (1) An ordered set of **bits** or characters that is the normal unit in which information may be stored, transmitted, or operated upon within a given **computer.**

(2) A character **string** or **bit** string considered as an entity. (ANSI)

5. Software Life Cycle

The software life cycle consists of a set of discrete activities occurring in a given order during the development and use of software and software systems. The time periods during which these activities occur are referred to as phases. At the current time a consensus has not developed as to which phases comprise the software life cycle. The following diagram (Figure 1) provides an example illustration of the relationship among the life cycle phases. Each of the phases shown in the figure is defined in the glossary.

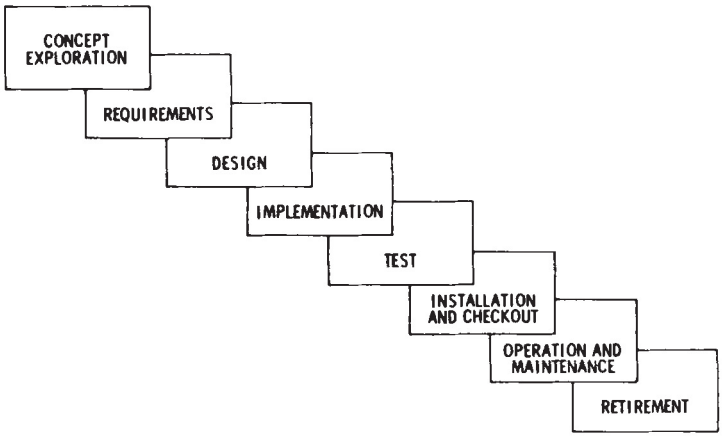

Figure 1. An Example Software Life Cycle

Appendix 1

Reference Identification

(This Appendix is not part of IEEE Std 729-1982, IEEE Standard Glossary of Software Engineering Terminology).

The following provides complete identification of the short references provided after specific terms.

(1) ANSI/ASQC A3-1978 Quality Systems Terminology.[1]

(2) ANSI N45.2.10-1973, Quality Assurance Terms and Definitions.

(3) ANSI/IEEE Std 488-1978, IEEE Standard Digital Interface for Programmable Instrumentation.[2]

(4) ANSI/IEEE Standard 730-1981, IEEE Standard for Software Quality Assurance Plans.

(5) DoD-STD 480A, Configuration Control -Engineering Changes, Deviation and Waivers.[3]

(6) Title 10, Code of Federal Regulations, Chapter 50.

[1]ANSI publications are available from Sales Department, Amercian National Standard Institute, 1430 Broadway, New York, NY 10018.

[2]IEEE publications are available from the IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854.

[3]DoD documents are available from the DoD Single Stock Point, Naval Publications and Forms Center, Philadelphia, PA 19120.