

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 10017-2199
February 18, 1983

Documents
SH08920
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25, 2026 at 23:06:29 UTC from IEEE Xplore. Restrictions apply.

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

© Copyright 1983 by

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

*No part of this publication may be reproduced in any form,
in an electronic retrieval system or otherwise,
without the prior written permission of the publisher.*

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
Joan P. Bateman
Stephen R. Beason
Milton E. Boyd, Jr
Kurt F. Fischer

Glenn C. Hughes, II
John M. Ives
John J. McKissick, Jr
Albrecht J. Neumann
John N. Postak

Jane W. Radatz
Marilyn J. Stewart
Alan N. Sukert
Donald A. Woodmancy
David Yablon

Other members:

Duane C. Abbey
Allen Troy Acree, Jr
Lynn Beckwith
Peter C. Belford
Edward H. Bersoff
Grady Booch
John B. Bowen
Martha A. Branstad
John R. Brown
Fletcher J. Buckley
E. Jane Cameron
Joseph Cavano
James V. Cellini, Jr
Karl E. Christen
Tom Clark
Michele J. DiFranza
Paul Doelger
Edward H. Ely
B. Dale Farrar
Carolyn Gannon
David Gelperin
R.R. Gordon
Robert M. Gross
David A. Gustafson
Virl Haas
George B. Hawthorne
Herbert Hecht
Robert E. Hite
Daniel E. Hocking

Raymond Houghton
William B. Humphrey
F.T. James
Tom Kallai
Edwin Kammerer
Kathleen A. Kenney
Brij M. Khandelwal
Ronald Klobert
K. Peter Koschewa
O.J. Krten
James C. Landerkin
John B. Lane
Myron Lipow
Benn Manny
Harvey Marks
Neldon Marshall
James A. McCall
Siba Mohanty
Dennis B. Mulcare
John Nissen
John O'Rourke
R. Poston
Stephen Pozgaj
J.A. (Jock) Rader
Geraldine (Burke) Rajcula
Jean C. Rault
R. Ravichandran
Donald J. Reifer
Debra J. Richardson

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

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

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 particular 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 upon 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 ob-

jects 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)