GB

NATIONAL STANDARD OF THE

PEOPLE'S REPUBLIC OF CHINA

ICS 35.080

L 77

GB/T 11457-2006

Replacing GB/T 11457-1995

Information technology - Software engineering terminology

信息技术 软件工程术语

Issued on: March 14, 2006 **Implemented on: July 01, 2006**

Issued by: General Administration of Quality Supervision, Inspection and Quarantine;

Standardization Administration of the People's Republic of China.

Table of Contents

Foreword 3

1 Scope 5

2 Terms, definitions and abbreviations 5

English index 274

Information technology - Software engineering terminology

1 Scope

This Standard defines general terms in the field of software engineering. It is applicable to software development, use and maintenance, research, teaching and publishing.

2 Terms, definitions and abbreviations

2.1 1GL

1GL is acronym for first generation language. See: machine language (2.890).

2.2 2GL

2GL is acronym for second generation language. See: assembly language (2.86).

2.3 3GL

3GL is acronym for third generation language. See: high order language (2.702).

2.4 4GL

4GL is acronym for fourth generation language. See: 2.654.

2.5 5GL

5GL is acronym for fifth generation language. See: 2.623.

2.6 abend

Abbreviation for abnormal end. A process is terminated before it is completed. See also: abort (2.8) and exception (2.575).

2.7 abnormal end

A process is terminated before it is completed. See also: abort (2.8) and exception (2.575).

2.8 abort

2.15 abstract data type

A data type for which only the properties of the data and the operations to be performed on the data are specified, without concern for how the data will be represented or how the operations will be implemented.

2.16 abstract machine

a) A representation of a process or machine;

b) A module that handles input like a machine.

2.17 abstraction

a) A view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information;

b) The process of formulating a view as in a).

See also: data abstraction (2.388).

2.18 acceptance criteria; acceptance criterion

The criteria that a system or component must satisfy in order to be accepted by a user, customer, or other authorized entity.

See also: requirement (2.1361) and test criteria (2.1699).

2.19 acceptance testing

a) 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 (2.1291), system testing (2.1669).

b) Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component.

See also: operational testing (2.1065) and qualification testing (2.1291).

Contrast with: development testing (2.468).

2.20 accessibility

The degree of ease with which software elements are used or maintained.

2.21 access-control mechanism

A characteristic operating process or management process of hardware or

See also: software process improvement proposal (2.1517).

2.28 action sequence

A representation of the sequence that determines the action.

2.29 action state

A state to indicate that an atomic action is performed (typically an operation is enabled).

2.30 activation

Perform an action.

2.31 active class

A class that its example is active object.

See also: active object (2.33).

2.32 active file

A file that has not exceeded the end time.

2.33 active object

An object that has a thread and can initiate control activities. An example of active class.

See also: active class (2.31) and thread (2.1732).

2.34 active redundancy

In fault tolerance, the use of redundant elements operating simultaneously to prevent, or permit recovery from failures.

Contrast with: standby redundancy (2.1565).

2.35 activity

a) A constituent element of a process;

NOTE: Changes to the baseline are subject to formal approval by the relevant agency.

b) Any of the steps taken or any of the functions performed for a purpose, mental or physical. Activity includes all the work that managers and technicians do to complete the tasks of a project and an organization.

2.36 activity graph

2.44 address

a) A number, character, or group of characters that identifies a register, device, memory specific part or some other data source or destination;

b) Used to specify a device or a data item;

c) To refer to a device or storage location by an identifying number, character, or group of characters.

See also: absolute address (2.9), effective address (2.531), implied address (2.731), indirect address (2.744), relative address (2.1331), relocatable address (2.1342), symbolic address (2.1636) and virtual address (2.1836).

2.45 address field

Any of the fields of a computer instruction that contain addresses, information necessary to derive addresses, or values of operands.

Contrast with: operation field (2.1060).

2.46 address format

a) The number and arrangement of address fields in a computer instruction.

See also: n-address instruction (2.1002) and n-plus-one address instruction (2.1008).

b) The number and arrangement of elements within an address, such as the elements needed to identify a particular channel, device, disk sector, and record in magnetic disk storage.

2.47 address modification

Any arithmetic, logical, or syntactic operation performed on an address.

See also: effective address (2.531), indexed address (2.740), relative address (2.1331) and relocatable address (2.1342).

2.48 address part

See: address field (2.45).

2.49 address space

a) The addresses that a computer program can access.

NOTE: In some systems, this may be the set of physical storage locations that a program can access, disjoint from other programs, together with the set of virtual

understand an algorithm so as to modify, simplify or improve it.

2.56 algorithmic language

A programming language designed for expressing algorithms; for example, ALGOL.

See also: algebraic language (2.53), list processing language (2.860) and logic programming language (2.873).

2.57 allocated baseline

In configuration management, the initial approved specifications governing the development of configuration items that are part of a higher-level configuration item.

See also: allocated configuration identification (2.58).

Contrast with: developmental configuration (2.470), functional baseline (2.659) and product baseline (2.1213).

2.58 allocated configuration identification

In configuration management, the current approved specifications governing the development of configuration items that are part of a higher-level configuration item. Each specification defines the functional characteristics that are al located from those of the higher-level configuration item, establishes the tests required to demonstrate achievement of its allocated functional characteristics, delineates necessary inter - face requirements with other associated configuration items, and establishes design constraints (if any).

See also: allocated baseline (2.57).

Contrast with: functional configuration identification (2.663) and product configuration identification (2.1215).

2.59 alias

a) Another name for an item;

b) A replacement label; for example, a label and one or more aliases can be used to indicate the same element or point in a computer program.

2.60 allocation

a) The process of distributing requirements, resources, or other entities among the components of a system or program;

b) The result of the distribution in a).

2.69 application-oriented language

a) A computer language with facilities or notations applicable primarily to a single application area; for example, a language for computer assisted instruction or hardware design;

b) A problem-oriented language whose statements contain or incorporate terminology for the user's profession.

See also: authoring language (2.106), specification language (2.1550) and query language (2.1303).

2.70 application problem

A problem that is proposed by the end user and requires information processing to solve it.

2.71 application software

Software designed to fulfill specific needs of a user; for example, software for navigation (browsing), payroll, or process control.

Contrast with: support software (2.1632) and system software (2.1667).

2.72 architectural design

a) The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system;

See also: functional design (2.665).

b) The result of architectural design process.

2.73 architecture

The organizational structure of a system or component.

See also: component (2.261), module (2.977), subprogram (2.1618), routine (2.1402), program architecture (2.1226) and system architecture (2.1647).

2.74 argument

a) An independent variable; for example, the variable m in the equation $E = mc^2$;

b) A specific value of an independent variable; for example, the value m = 24kg;

2.81 assembled origin

The address of the initial storage location assigned to a computer program by an assembler, a compiler, or a linkage editor.

See also: offset (2.1043) and starting address (2.1567).

Contrast with: loaded origin (2.867).

2.82 assembler

A computer program that translates programs expressed in assembly language into their machine language equivalents.

See also: absolute assembler (2.10), cross-assembler (2.373) and relocating assembler (2.1346).

Contrast with: compiler (2.254) and interpreter (2.809).

2.83 assembler code

See: assembly code (2.85).

2.84 assembler language

See: assembly language (2.86).

2.85 assembly code

Computer instruction and data definition (assembler code) represented by a form that the assembler can recognize and process.

Contrast with: compiler code (2.255), interpretive code (2.810) and machine code (2.887).

2.86 assembly language

a) A programming language that corresponds closely to the instruction set of given computer, allows symbolic naming of operations and addresses, and usually results in a one-to-one translation of program instructions into machine instructions;

b) A specific machine language whose instructions are usually one-to-one with computer instructions.

Contrast with: fifth generation language (2.623), fourth generation language (2.654), high order language (2.702) and machine language (2.890).

2.87 assertion

2.96 assessment record

An orderly, documented collection of that information which is pertinent to the assessment and adds to the understanding and verification of the process profiles generated by the assessment.

2.97 assessment scope

A definition of the boundaries of the assessment, provided as part of the assessment input, encompassing the organizational limits of the assessment, the processes to be included, and the context within which the processes operate (See: process context (2.1194)).

2.98 assessment sponsor

The individual, internal or external to the organization being assessed, who requires the assessment to be performed, and provides financial or other resources to carry it out.

2.99 assignment statement

A computer program statement that is used to express a series of operations, or to assign an operand to a specified variable, or a symbol, or both a variable and a symbol; for example, Y = X + 5.

See also: clear (2.213), initialize (2.756) and reset (2.1372).

Contrast with: control statement (2.343) and declaration (2.416).

2.100 association

To specify semantic association between multiple categories connected between instances.

2.101 association class

A model element that combines the dual nature of association and class. An association class can be thought of as having class nature or as a class with associative nature.

2.102 association end

For an association, the end point that connects it to classifier.

2.103 atomic type

A data type, each of whose members consists of a single, non-decomposable data item.

A software tool which uses computer programs and guidelines as input to generate test input data that meets the requirements of these guidelines, and sometimes determines the expected results.

2.112 automated verification system

a) A software tool that accepts as input a computer program and a representation of its specification and produces, possibly with human help, a proof or disproof of the correctness of the program;

See also: automated verification tools (2.113).

b) Any software tool that automates part or all of the verification process.

2.113 automated verification tools

a) A class of software tools used to evaluate products in the software development process. These tools help verify correctness, completeness, consistency, traceability, testability, and check compliance with standards. Software verification tools include design analyzers, automatic verification system;

b) Static analyzer, dynamic analyzer, and standard implementer.

2.114 auxiliary class

A derivation of a class that is to achieve a two-level logic or control flow as a typical way so as to support another, more central or more basic class. The use of auxiliary class with focus class is representative and is particularly useful during the design process for specifying secondary business logic or control flow for each component.

See also: focus class (2.639).

2.115 availability

a) The degree to which a system or component is operational and accessible when required for use; often expressed as a probability;

See also: error tolerance (2.570), fault tolerance (2.617) and robustness (2.1397).

b) The ratio of the normal working hours of system to the total running time;

c) The ability of a configuration item to implement a specified function at runtime.

2.116 availability model

2.122 backward execution

See: reversible execution (2.1391).

2.123 backward recovery

a) The reconstruct ion of a file to a given state by reversing all changes made to the file since i t was in that state;

b) A type of recovery in which a system, program, database, or other system resource is restored to a previous state in which it can perform required functions.

Contrast with: forward recovery (2.650).

2.124 base address

An address used as a reference point to which a relative address is added to determine the address of the storage location to be accessed.

See also: indexed address (2.740), relative address (2.1331) and self-relative address (2.1429).

2.125 baseline

a) 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;

b) A document or a set of such documents formally designated and fixed at a specific time during the life cycle of a configuration item. Baselines, plus approved changes from those baselines, constitute the current configuration identification.

See also: allocated baseline (2.57), developmental configuration (2.470), functional baseline (2.659) and product baseline (2.1213).

c) Any agreement or result designated and fixed at a given time, from which changes require justification and approval.

For configuration management, there are three baselines:

Functional baseline: initially passed feature configuration;

Allocated baseline: initially passed allocation configuration;

Product baseline: initially passed or conditionally passed product configuration.

2.126 baseline configuration management

system (including method, collaboration, and state).

2.134 benchmark

a) A standard against which measurements or comparisons can be made.

b) A procedure, problem, or test that can be used to compare systems or components to each other or to a standard as in a).

c) A recovery file.

2.135 bidder

An individual, partnership, company or association that has submitted a proposal and been accepted as a candidate of one or more product design, development and/or manufacturing contract.

2.136 big-bang testing

A type of integration testing in which software elements, hardware elements, or both are combined all at once into an overall system, rather than in stages.

2.137 binary association

An association between two classes. A special case of N-ary associations.

2.138 binary digit [bit]

a) A unit of information that can be represented by either a 0 or a 1.

b) An element of computer storage that can hold a unit of information as in a).

c) A numeral used to represent one of the two digits in the binary numeration system; either 0 or 1.

See also: byte (2.170) and word (2.1850).

2.139 bind

To assign a value to an identifier. For example, to assign a value to a parameter or to assign an absolute address to a symbolic address in a computer program.

See also: dynamic binding (2.515) and static binding (2.1579).

2.140 binding

To assign a value or a specified object (referent) to an identifier. For example, to assign a value to a parameter or assign an absolute address, virtual address,

2.151 bootstrap

a) A short computer program that is permanently resident or easily loaded into a computer and whose execution brings a larger program, such as an operating system or its loader, into memory;

b) A set of instructions that enable other instructions to be loaded until the computer program is fully loaded into memory;

c) A technique or device that achieves a desired state by its own actions. For example, a machine subroutine whose first few instructions are sufficient to cause the rest of the instructions to be input from the input device to the computer;

d) A portion of a computer program used to establish another version of a computer program;

e) Use a bootstrap;

f) A program that can play the role specified in a).

2.152 bootstrap loader

A short computer program used to load a bootstrap.

2.153 bottom-up

Pertaining to an activity that starts with the lowest-level components of hierarchy and proceeds through progressively higher levels; for example, bottom-up design; bottom-up program design; bottom-up testing.

See also: critical piece first (2.368).

Contrast with: top-down (2.1746).

2.154 bottom-up design

A systematic design that starts with the most basic or original parts then moves to a higher level.

Contrast with: top-down design (2.1747).

2.155 boundary value

A data value that corresponds to a minimum or maximum input, internal, or output value specified for a system or component.

2.156 box diagram

Figure 3 -- Bubble chart

2.162 buffer

a) A device or storage area used to store data temporarily to compensate for differences in rates of data flow, time of occurrence of events, or amounts of data that can be handled by the devices or processes involved in the transfer or use of the data.

b) A routine that accomplishes the objectives in a).

c) To allocate, schedule, or use devices or storage areas as in a).

See also: anticipatory buffering (2.65), dynamic buffering (2.517) and simple buffering (2.1451).

2.163 bug

See: error (2.561) and fault (2.609).

2.164 bug seeding

See also: error seeding (2.569) and fault seeding (2.615).

2.165 build

An operational version of a system or component that incorporates a specified subset of the capabilities that the final product will provide.

2.166 building block

A unit or module used by a higher-level program or module.

2.167 burn-in period

See: early-failure period (2.526).

2.168 busy

Pertaining to a system or component that is operational, in service, and in use.

See also: down (2.502), idle (2.716) and up (2.1797).

2.169 busy time

In computer performance engineering, the period of time during which a system or component is operational, in service, and in use.

See also: down time (2.503), idle time (2.717), set-up time (2.1444) and up time (2.1798).

Defect analysis to determine the source of the defect.

2.189 causal analysis meeting

A meeting held to analyze defects that are exposed during task completion after a specific task is completed.

2.190 CCB

Acronym for Configuration Control Board or acronym for Change Control Board.

2.191 CDR

Acronym for Critical Design Review.

2.192 certification

a) A written guarantee that a system or component complies with its specified requirements and is acceptable for operational use. For example, a written authorization that a computer system is secure and is permitted to operate in a defined environment;

b) A formal demonstration that a system or component complies with its specified requirements and is acceptable for operational use;

c) The process of confirming that a system or component complies with its specified requirements and is acceptable for operational use.

2.193 chained list

A list in which items can be decentralized, but each item contains a pointer or identifier that indicates the next item's location.

2.194 change control

The process of making a change and estimating, agreeing or rejecting, scheduling, and tracking it.

See: configuration control (2.307).

2.195 change control board

See: configuration control board (2.308).

2.196 change dump

A selective dump of those storage locations whose contents have changed since some specified time or event.

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.

See also: backup programmer (2.121) and chief programmer team (2.206).

2.206 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 procedures designed to enhance group communication and to make optimum use of each member's skills.

See also: backup programmer (2.121), chief programmer (2.205) and egoless programming (2.536).

2.207 child

In a generalized connection, for another element, it is the parent's generalization.

See also: subclass (2.1613) and (2.1623 subtype).

Contrast with: parent (2.1107).

2.208 CI

Acronym for configuration item.

2.209 class

A description of the collection of objects that share the same attributes, methods of operation, associations, and semantics. Class can use a set of interfaces to specify a collection of operations for its environment.

See also: interface (2.796).

2.210 classifier

A mechanism describing structural and behavioral characteristics. Classifier is a generic term for modeling elements such as class, interface data type, and component.

2.211 classification

A mechanism assigning object to a purpose.

See also: dynamic classification (2.518), multiple classification (2.991) and

c) A character or bit pattern that is assigned a particular meaning; for example, a status code.

2.219 code audit

An independent review of the source code by an individual, a group, or with some tool. Its purpose is to verify that it meets software design documents and programming standards. It is also possible to estimate the correctness and validity.

See also: audit (2.105), static analysis (2.1577), inspection (2.764) and walk-through (2.1843).

2.220 code breakpoint

A breakpoint that is initiated upon execution of a given computer instruction.

See also: dynamic breakpoint (2.516), epilog breakpoint (2.559), programmable breakpoint (2.1249), prolog breakpoint (2.1268) and static breakpoint (2.1580).

Contrast with: data breakpoint (2.391).

2.221 code generator

a) A routine, often part of a compiler, that transforms a computer program from some intermediate level of representation (often the output of a root compiler or parser) into a form that is closer to the language of the machine on which the program will execute.

b) A software tool that accepts as input the requirements or design for a computer program and produces source code that implements the requirements or design.

See also: application generator (2.68).

2.222 code inspection

See also: inspection (2.764).

2.223 code of ethics standard

A standard that describes the characteristics of a set of moral principles dealing with accepted standards of conduct by, within, and among professionals.

2.224 code review

A meeting at which software code is presented to project personnel, managers, users, customers, or other interested parties for comment or approval.

instances and links organized around the structure of a model. Unlike the sequence diagram, the collaboration diagram shows the connections between the instances. Sequence diagram and collaboration diagram express the similar information but in different way.

See also: sequence diagram (2.1437).

2.232 command

An expression that can be input to a computer system to initiate an action or affect the execution of a computer program; for example, the “log on” command to initiate a computer session.

2.233 command-driven

Pertaining to a system or mode of operation in which the user direct s the system through commands.

Contrast with: menu-driven (2.931).

2.234 command language

A set of procedural operators and their associated grammar (a language) used to express commands to a computer system (indicating the function assigned to the operating system).

See also: command-driven (2.233).

2.235 comment

a) Information embedded within a computer program, command language, job control statements, or a set of data, that provides clarification to human readers but does not affect machine interpretation;

b) A description, note, or explanation added to or scattered in the source language, which are invalid in target language.

2.236 commitment

A visual agreement that it wants all parties to abide by.

2.237 common

See: common storage (2.245).

2.238 common area

See: common storage (2.245).

d) Measurement analysis: necessary description of the process measurement and analysis of the measurement results. Measurements and analysis generally include examples of measurements that may be taken to determine the status and effectiveness of an activity performed;

e) Verification implementation: measures to ensure that activities are carried out in accordance with established processes. Verification generally includes reviews and examinations by managers and software quality assurance groups.

2.245 common storage

A portion of main storage that can be accessed by two or more modules in a software system.

See also: global data (2.680).

2.246 communicational cohesion

A type of cohesion in which the tasks performed by a software module use the same input data or contribute to producing the same output data.

Contrast with: coincidental cohesion (2.229), functional cohesion (2.661), l logical cohesion (2.874), procedural cohesion (2.1179) and sequential cohesion (2.1439).

2.247 compaction

In microprogramming, the process of converting a microprogram into a functionally equivalent microprogram that is faster or shorter than the original.

See also: local compaction (2.869) and global compaction (2.679).

2.248 comparator

A software tool that compares 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.

2.249 compatibility

a) The ability of two or more systems or component s to perform their required functions while sharing the same hardware or software environment;

b) The ability of two or more systems or components to exchange information.

See also: interoperability (2.807).

2.258 compile time

Something that occurs during the compilation of a software module.

See also: modeling time (2.970) and running time (2.1408).

2.259 completion code

A code communicated to a job stream processor by a batch program to influence the execution of succeeding steps in the input stream.

2.260 complexity

The degree to which a system or component has a design or implementation that is difficult to understand and verify. It is determined by the following factors, for example, the number and intricacy of interfaces, the number and inconsistency of conditional transitions, the depth of nesting, the type of data structure, and other system characteristics.

Contrast with: simplicity (2.1452).

2.261 component

a) One of the parts (basic part of system or program) that make up a system. A component may be hardware or software and may be subdivided into other components;

Note: The terms “module”, “component” and “unit” are often used interchangeably or defined to be sub-elements of one another in different ways depending upon the context. The relationship of these terms is not yet standardized.

b) Mature, reusable parts.

2.262 component diagram

A diagram illustrating the organization and dependency between components.

2.263 component standard

A standard that describes the characteristics of data or program components.

2.264 component testing

Testing of individual hardware or software components or groups of related components.

See also: integration testing (2.788), interface testing (2.800), system testing (2.1669) and unit testing (2.1792).

Data used for communication between a computer device and a computer device or within a computer device. Such data may be external (computer readable form) or resident within a computer device, either an analog signal or a digital signal.

2.273 computer language

A language designed to enable humans to communicate with computers.

See also: design language (2.447), query language (2.1303) and programming language (2.1252).

2.274 computer network

A complex consisting of two or more computers interconnected by a certain protocol.

2.275 computer performance evaluation

An engineering discipline that measures the performance of computer systems and investigates methods by which that performance can be improved.

See also: system profile (2.1661), throughput (2.1735), utilization (2.1817) and workload model (2.1857).

2.276 computer program

A combination of computer instructions and data definitions that enable computer hardware to perform computational or control functions.

See also: software (2.1469).

2.277 computer program abstract

A brief description of a computer program. It provides the user with enough information to determine whether the computer program is suitable for the needs and to determine the resources it has.

2.278 computer program annotation

See also: comment (2.235).

2.279 computer program certification

See also: certification (2.192).

2.280 computer program component

See: computer software component (2.289).

See also: configuration item (2.311).

Contrast with: hardware configuration item (2.691).

2.291 computer system

A system containing one or more computers and associated software.

2.292 computer word

See: word (2.1850).

2.293 computer center

A facility designed to provide computer services to a variety of users through the operation of computers and auxiliary hardware and through services provided by the facility's staff.

2.294 concept phase

a) The period of time in the software development cycle during which the user needs are described and evaluated through documentation (for example, statement of needs, advance planning report, project initiation memo, feasibility studies, system definition, documentation, regulations, procedures, or policies relevant to the project);

b) The initial phase of a software development project, in which the user needs are described and evaluated through documentation (for example, statement of needs, advance planning report, project initiation memo, feasibility studies, system definition, documentation, regulations, procedures, or policies relevant to the project).

2.295 conceptual system design

A system design activity involving the logical aspects, processes, and system-wide information flows that define the organization of the system.

2.296 concrete class

A class that can be used directly to create an instance.

2.297 concurrency

The emergence of multiple activities in the same time interval. Concurrency can be obtained by interleaving or simultaneous execution of multiple threads.

See also: thread (2.1732).

2.298 concurrent

b) In configuration management, the functional and physical characteristics of hardware or software as set forth in technical documentation or achieved in a product. See also: configuration item (2.311), form, fit and function (2.643) and version (2.1829).

2.306 configuration audit

Proof that all required configuration items have been generated, and the current configuration is consistent with the specified requirements. It is a process that the technical documentation describes the various configuration items completely and accurately, and all the change requests that have been made have been resolved.

See: functional configuration audit (2.662) and physical configuration audit (2.1139).

2.307 configuration control

An element of configuration management, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification.

See also: configuration control board (2.308), deviation (2.473), engineering change (2.546), interface control (2.796), notice of revision (2.1028), specification change notice (2.1549) and waiver (2.1842).

Contrast with: configuration identification (2.309) and configuration status accounting (2.315).

2.308 configuration control board

A group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes.

See also: configuration control (2.307).

2.309 configuration identification

a) An element of configuration management, consisting of selecting the configuration items for a system and recording their functional and physical characteristics in technical documentation;

Contrast with: configuration control (2.307) and configuration status accounting (2.315).

b) The current approved technical documentation for a configuration item as

2.315 configuration status accounting

An element of configuration management, consisting of the recording and reporting of information needed to manage a configuration effectively. This information includes a listing of the approved configuration identification, the status of proposed changes to the configuration, and the implementation status of approved changes.

See also: configuration index (2.310) and configuration item development record (2.312).

Contrast with: configuration control (2.307) and configuration identification (2.309).

2.316 configuration unit

The lowest level entity that can be placed into the configuration management library system and a configuration item or component retrieved from the library.

2.317 confinement

a) The measures to prevent unauthorized changes, use, destruction, and discarding of data during an approved visit;

See also: integrity (2.789).

b) The restrictions imposed on procedures and processes are designed to prevent them from interfering with or affecting unauthorized data, procedures or processes.

2.318 connection

a) A reference to an identifier of one part of a program to another part (i.e., an identifier found elsewhere);

See also: interface (2.795).

b) The relationship established between functional components in order to convey information.

2.319 consecutive

Pertaining to the occurrence of two sequential events or items without the intervention of any other event or item; that is, one immediately after the other.

2.320 consistency

The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component.

A type of coupling in which some or all of the contents of one software module are included in the contents of another module.

Contrast with: common environment coupling (2.243), control coupling (2.336), data coupling (2.393), hybrid coupling (2.714) and pathological coupling (2.1123).

2.328 context

a) The process environment that needs to be saved (when the process exits the run) and resumed (when the process enters the run) during the stop-and-go operation;

b) A view of a collection of modeling elements for a specific purpose (for example, specifying an operation).

2.329 contiguous allocation

A storage allocation technique in which programs or data to be stored are allocated a block of storage of equal or greater size, so that logically contiguous programs and data are assigned physically contiguous storage locations.

Contrast with: paging (2.1101).

2.330 contingency factor

An adjustment (increase) to size, cost, or schedule, in case these parameters are underestimated due to incomplete specifications and lack of estimation experience in the application field.

2.331 continuous iteration

A loop that has no exit.

2.332 contract

An agreement to bind the parties through the law, or an internal agreement within an organization to provide services. The services provided by this agreement apply to the supply, development, production, operation or maintenance of a system or part of a system.

2.333 contract terms and conditions

Contents of legal, financial and management aspects of the contract.

2.334 contractually required audit

Review required by contract. It is usually conducted by the acquirer or by an independent agency. This process provides an independent evaluation of a

2.342 control program

See: supervisory program (2.1628).

2.343 control statement

A program statement that selects among alternative sets of program statements or affects the order in which operations are performed. For example, if-then-else, case.

Contrast with: assignment statement (2.99), declaration statement (2.416).

2.344 control store

In a microprogrammed computer, the computer memory in which microprograms reside.

See also: microword (2.953) and nanostore (2.1011).

2.345 control structure

A construction that the control flow is determined by a computer program.

See also: conditional control structure (2.303).

2.346 control variable

See: loop control variable (2.883).

2.347 conventions

Requirements employed to prescribe a disciplined uniform approach to providing consistency in a software product, that is, uniform patterns or forms for arranging data.

See also: practices (2.1156) and standards (2.1562).

2.348 conversational

Pertaining to an interactive system or mode of operation in which the interaction between the user and the system resembles a human dialog.

See also: interactive (2.792), on-line (2.1045) and real time (2.1313).

Contrast with: batch (2.128).

2.349 conversational compiler

See incremental compiler (2.737).

2.356 correctness proof

See: proof of correctness (2.1270).

2.357 counter

A variable used to record the number of occurrences of a given event during the execution of a computer program; for example, a variable that records the number of times a loop is executed.

2.358 coupling

The manner and degree of interdependence between software modules. Types include common-environment coupling, content coupling, control coupling, data coupling, hybrid coupling, and pathological coupling.

Contrast with: cohesion (2.228).

2.359 courseware

A computer software. It is used for course teaching or self-study and is an integral part of computer-aided teaching software.

2.360 CPC

Acronym for computer program component.

See: computer software component (2.289).

2.361 CPCI

Acronym for computer program configuration item.

See: computer software configuration item (2.290).

2.362 crash

The sudden and complete failure of a computer system or component.

2.363 critical

Referring to:

a) Serious consequences that due to improper design, certain parts or parts of a system or a software are out of critical range at runtime, or there are potential, undetected errors that can cause crashes, personal injury, mission failure, data loss, and financial loss or catastrophic equipment damage. Or:

2.370 critical section

A piece of code that shall be executed. Its execution is mutually exclusive with the execution of code of another critical segment. If some code segments are competing to use a computer resource and data item, they are required to execute mutually exclusive.

2.371 critical software

Software whose failure could have an impact on safety or could cause large financial or social loss.

2.372 criticality

The degree of impact that a requirement, module, error, fault, failure, or other item has on the development or operation of a system.

2.373 cross-assembler

An assembler that executes on one computer but generates machine code for a different computer.

2.374 cross-compiler

A compiler that executes on one computer but generates machine code for a different computer.

2.375 cross-reference generator

A software tool that accepts as input the source code of a computer program and produces as output a listing that identifies each of the program's variables, labels, and other identifiers and indicates which statements in the program define, set, or use each one.

2.376 cross-reference list

A list that identifies each of the variables, labels, and other identifiers in a computer program and indicates which statements in the program define, set, or use each one.

2.377 cross-referencer

See: cross-reference generator (2.375).

2.378 CSC

Acronym for computer software component.

2.379 CSCI

storage begins with the block following the one last allocated.

2.387 data

a) A representation of facts, concepts, or instructions in a manner suitable for communication, interpretation, or processing by humans or by automatic means.

See also: data type (2.408), computer data (2.272), control data (2.337), error data (2.564), reliability evaluation (2.1337) and software experience data (2.1500).

b) Sometimes used as a synonym for documentation (2.496).

2.388 data abstraction

a) The process of extracting the essential characteristics of data by defining data types and their associated functional characteristics and disregarding representation details;

See also: encapsulation (2.542) and information hiding (2.751).

b) The result of the process in a).

2.389 data analysis

A systematic survey of data and its processes in actual or planned system.

2.390 database

a) A data set, or part or all of a data set. It shall at least contain one file that is enough to be used for a given purpose or given data processing system;

b) Basic data collection for a system.

2.391 data breakpoint

A break point that is initiated when a specified data item is accessed.

See also: dynamic breakpoint (2.516), epilog breakpoint (2.559), programmable breakpoint (2.1249), prolog breakpoint (2.1268) and static breakpoint (2.1580).

Contrast with: code breakpoint (2.220).

2.392 data characteristic

An inherent, possibly accidental, trait, quality, or property of data (for example, arrival rates, formats, value ranges, or relationships between field values).

A situation in which computer processing is suspended because two or more devices or processes are each awaiting resources assigned to the others.

Refer to: lockout (2.872).

2.411 deassembler

See: disassembler (2.488).

2.412 deblock

To separate the parts of a block.

Contrast with: block (2.143).

2.413 debugging

To detect, locate, and correct faults in a computer program. Techniques include use of breakpoint, desk checking, dump, inspection, reversible execution, single-step operation, and trace.

2.414 debugging model

See also: error model (2.565).

2.415 decision table

a) A table of all possible situations to be considered in describing a problem and the actions to be taken for each group of possible situations;

b) A table used to show sets of conditions and the actions resulting from them.

2.416 declaration

A non-executable program statement that affects the assembler or compiler's interpretation of other statements in the program. For example, a statement that identifies a name, specifies what the name represents, and, possibly, assigns it an initial value.

See also: pseudo instruction (2.1280).

Contrast with: assignment statement (2.99) and control statement (2.343).

2.417 declarative language

A nonprocedural language that permits the user to declare a set of facts and to express queries or problems that use these facts.

See also: interactive language (2.793) and rule-based language (2.1403).

2.427 definition phase

See also: requirement phase (2.1365).

2.428 delegation

The ability of an object to post a message to another object in response to a message. Delegation can be used as an alternative to inheritance.

Contrast with: inheritance (2.752).

2.429 delimiter

A character or set of characters used to denote the beginning or end of a group of related bits, characters, words, or statements.

2.430 delivery

a) A stage in the software development cycle. At this stage, the product is submitted to its customers or users in the program to use;

b) A stage in the software development cycle. At this stage, the product is accepted by its intended users.

2.431 demand paging

A storage allocation technique in which pages are transferred from auxiliary storage to main storage only when those pages are needed.

Contrast with: anticipatory paging (2.66).

2.432 demodularization

In software design, the process of combining related software modules, usually to optimize system performance.

See also: downward compression (2.505), lateral compression (2.841) and upward compression (2.1800).

2.433 demonstration

A dynamic analysis technique that relies on observation of system or component behavior during execution, without need for post -execution analysis, to detect errors, violations of development standards, and other problems.

See also: testing (2.1726).

2.434 dependency

generates the following outputs, for example, graphical representation of module hierarchy diagrams, controls and data structures, and a table of accessed data blocks.

2.442 design description

A document that describes the design of a system or component. Typical contents include system or component architecture, control logic, data structures, input/ output formats, interface descript ions, and algorithms.

See also: product specification (2.1219).

Contrast with: requirement specification (2.1367).

2.443 design document

See: design description (2.442).

2.444 design element

A basic component or building block in a design.

2.445 design entity

An element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced.

2.446 design inspection

See: inspection (2.764).

2.447 design language

A specification language with special constructs and, sometimes, verification protocols, used to develop, analyze, and document a hardware or software design. Types include hardware design language, program design language.

See also: requirement specification language (2.1368).

2.448 design level

The design decomposition of the software item (for example, system, subsystem, program, or module).

2.449 design methodology

Systematic approach to design, consisting of an orderly application of specially selected tools, techniques, and guidelines.

A logically related col lection of design elements. In an Ada PDL, a design unit is represented by an Ada compilation unit.

2.456 design view

A subset of design entity attribute information that is specifically suited to the needs of a software project activity.

2.457 design verification

See: verification (2.1826).

2.458 design walk-through

See: walk-through (2.1843).

2.459 desk checking

A static analysis technique in which code listings, test results, or other documentation are visually examined, usually by the person who generated them, to identify errors, violations of development standards, or other problems.

See: inspection (2.764) and walk-through (2.1843).

2.460 destination address

The address of the device or storage location to which data is to be transferred.

Contrast with: source address (2.1540).

2.461 destructive read

A read operation that alters the data in the accessed location.

Contrast with: nondestructive read (2.1024).

2.462 detailed design

a) The process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented;

See also: software development process (2.1491).

b) The result of detailed design process.

2.463 developer

An organization that performs development activities (including requirements analysis, design, and acceptance) during the software life cycle.

control of the developer. Configuration items placed under development configuration management are not baselines, although at some point in the development process they may be baselined and placed under baseline configuration management.

2.472 development process

A set of partially ordered steps performed during a software development for a given purpose, such as constructing a model or implementing a model.

2.473 deviation

a) A departure from a specified requirement.

b) A written authorization, granted prior to the manufacture of an item, to depart from a particular performance or design requirement for a specific number of units or a specific period of time.

NOTE: Unlike an engineering change, a deviation does not require revision of the documentation defining the affected item.

See also: configuration control (2.307).

Contrast with: engineering change (2.546) and waiver (2.1842).

2.474 device

A mechanism or piece of equipment designed to serve a purpose or perform a function.

2.475 diagnostic

Pertaining to the detection and isolation of faults or failures; for example, a diagnostic message, a diagnostic manual.

2.476 diagnostic manual

A document that presents the information necessary to execute diagnostic procedures for a system or component, identify mal functions, and remedy those malfunctions. Typically described are the diagnostic features of the system or component and the diagnostic tools available for its support.

See also: installation manual (2.766), operator manual (2.1068), programmer manual (2.1250), support manual (2.1631) and user manual (2.1813).

2.477 diagonal microinstruction

A microinstruction capable of specifying a limited number of simultaneous operations needed to carry out a machine language instruction.

2.492 diverse redundancy

See: diversity (2.493).

2.493 diversity

In fault tolerance, realization of the same function by different means. For example, use of different processors, storage media, programming languages, algorithms, or development teams.

See also: software diversity (2.1492).

2.494 do-nothing operation

See: no-operation (2.1019).

2.495 document

a) A medium, and the information recorded on it, that generally has permanence and can be read by a person or a machine. Examples in software engineering include project plans, specifications, test plans, user manuals;

b) To create a document;

c) To add comments to a computer program.

2.496 documentation

a) A collection of documents on a given subject;

See also: user documentation (2.1809), software documentation (2.1493) and system documentation (2.1653);

b) Any written or pictorial information describing, defining, specifying, reporting, or certifying activities, requirements, procedures, or results;

c) The process of generating or revising a document;

d) The management of documents, including identification, acquisition, processing, storage, and dissemination.

2.497 documentation level

See: level of documentation (2.846).

2.498 documentation tree

A diagram that depicts all of the documents for a given system and shows their

See also: busy time (2.169), idle time (2.717), mean time to repair (2.921) and set-up time (2.1444).

Contrast with: up time (2.1798).

2.504 downward compatible

Pertaining to hardware or software that is compatible with an earlier or less complex version of itself; for example, a program that handles files created by an earlier version of itself.

Contrast with: upward compatible (2.1799).

2.505 downward compression

In software design, a form of demodularization in which a superordinate module is copied into the body of a subordinate module.

Contrast with: lateral compression (2.841) and upward compression (2.1800).

2.506 driver

a) A software module that invokes and, perhaps, controls and monitors the execution of one or more other software modules;

See also: test driver (2.1705).

b) A computer program that controls a peripheral device and, sometimes, reformats data for transfer to and from the device.

2.507 [program] dual coding

A development technology. Two versions of functionally identical programs are developed by different programmers or different programming groups based on the same specification. The source code obtained can be in the same language or in a different language. The purpose of dual coding is to provide error checking, improve reliability, provide additional documentation, or reduce the probability of system programming errors or compiler errors affecting the final result.

See also: software diversity (2.1492).

2.508 dummy parameter

See: formal parameter (2.645).

2.509 dump

a) A display of some aspect of a computer program's execution state, usually

Contrast with: static analyzer (2.1578).

2.515 dynamic binding

Binding made during the execution of a computer program.

Contrast with: static binding (2.1579).

2.516 dynamic breakpoint

A breakpoint whose predefined initiation event is a runtime characteristic of the program, such as the execution of any twenty source statements.

See also: code breakpoint (2.220), data breakpoint (2.391), epilog breakpoint (2.559), programmable breakpoint (2.1249) and prolog breakpoint (2.1268).

Contrast with: static breakpoint (2.1580).

2.517 dynamic buffering

A buffering technique in which the buffer allocated to a computer program varies during program execution, based on current need.

Contrast with: simple buffering (2.1451).

2.518 dynamic classification

One of these objects can change a semantic class of its class's generalization.

Contrast with: static classification (2.1581).

2.519 dynamic dump

A dump that is produced during the execution of a computer program.

See also: change dump (2.196), memory dump (2.928), postmortem dump (2.1153), selective dump (2.1425) and snapshot dump (2.1466).

Contrast with: static dump (2.1582).

2.520 dynamic error

An error that is dependent on the time-varying nature of an input.

Contrast with: static error (2.1583).

2.521 dynamic relocation

Relocation of a computer program during its execution.

a) See: text editor (2.1728);

b) See: linkage editor (2.854).

2.531 effective address

The address that results from performing any required indexing, indirect addressing, or other address modification on a specified address. NOTE: If the specified address requires no modification, it is also the effective address.

See also: generated address (2.675), indirect address (2.744) and relative address (2.1331).

2.532 effective instruction

The computer instruction that results from per forming any required indexing, indirect addressing, or other modification on the addresses in a specified computer instruction.

NOTE: If the specified instruction requires no modification, it is also the effective instruction.

See also: absolute instruction (2.12), direct instruction (2.483), immediate instruction (2.722) and indirect instruction (2.745).

2.533 effective process

Process that has been practiced, documented, mandatory, trained, measured, and improved.

See also: well-defined process (2.1486).

2.534 efferent

Pertaining to a flow of data or control from a superordinate module to a subordinate module in a software system.

Contrast with: afferent (2.51).

2.535 efficiency

The degree to which a system or component performs its designated functions with minimum consumption of resources.

See also: execution efficiency (2.578) and storage efficiency (2.1594).

2.536 egoless programming

A software development technique based on the concept of team, rather than individual, responsibility for program development. Its purpose is to prevent

2.543 end user

Individual or group that uses system to achieve its intended use when system is deployed in its operating environment.

2.544 end user representatives

Selected individuals or groups that represent end users.

2.545 engineering

The application of a systematic, disciplined, quantifiable approach to structures, machines, products, systems, or processes.

2.546 engineering change

In configuration management, an alteration in the configuration of a configuration item or other designated item after formal establishment of its configuration identification.

See also: configuration control (2.307) and engineering change proposal (2.547).

Contrast with: deviation (2.473) and waiver (2.1842).

2.547 engineering change proposal

In configuration management, a proposed engineering change and the documentation by which the change is described and suggested.

See also: configuration control (2.307).

2.548 engineering group

A collection of people (including responsible persons, managers, and technicians) who work in engineering disciplines. Examples of engineering disciplines include system engineering, hardware engineering, system testing, software engineering, software configuration management, and software quality assurance.

2.549 enhanced capability

A capability that can be demonstrated by a trusted process improvement program that is more powerful than currently evaluated.

2.550 entity

In computer programming, any item that can be named or denoted in a program. For example, a data item, program statement, or subprogram.

See also: code breakpoint (2.220), data breakpoint (2.391), dynamic breakpoint (2.516), programmable breakpoint (2.1249) and static breakpoint (2.1580).

Contrast with: 2.1268 prolog breakpoint.

2.560 equivalent faults

Two or more faults that result in the same failure mode.

2.561 error

a) The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. For example, a difference of 30 meters between a computed result and the correct result;

b) An incorrect step, process, or data definition. For example, an incorrect instruction in a computer program;

c) An incorrect result. For example, a computed result of 12 when the correct result is 10;

d) A human action that produces an incorrect result. For example, an incorrect action on the part of a programmer or operator;

NOTE: While all four definitions are commonly used, one distinct ion assigns definition a) to the word “error”, definition b) to the word “fault,” definition c) to the word “failure”, and definition d) to the word “mistake”.

See also: dynamic error (2.520), fatal error (2.608), indigenous error (2.742), semantic error (2.1430), static error (2.1583), syntactic error (2.1641) and transient error (2.1767).

2.562 error analysis

a) The process of investigation on observed software failures. The purpose of the investigation is to track that fault to find the source of the fault;

b) Investigate observed software failures so as to find out some of the following information, for example, the cause of the failure. The method is to find out at which stage of the development process the fault occurred, and prevent or detect software failures earlier;

c) The process of investigating software errors, malfunctions, failures to determine quantitative rates and trends.

2.563 error category

This is an excerpt of the PDF (Some pages are marked off intentionally)

------ The End ------