CS 422 SOFTWARE ENGINEERING PRINCIPLES
WASHINGTON STATE UNIVERSITY

GLOSSARY

In order to provide an optimum frame of reference for the class, this glossary establishes a set of consistent technical definitions. Definitions contained here in are based on such references as the IEEE Standard Dictionary of Electrical and Electronic Terms (e.g., ANSI/IEEE Std 729-1983), and the Rome Air Development Center (RADC-TR-90-239) Testability/Diagnostic Design Encyclopedia. Definitions may be considered to have come from these sources unless otherwise indicated.

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

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

Anomaly. An operational characteristic (or implementation) which is believed to require corrective action.

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.

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).

Corrective maintenance. Maintenance performed specifically to overcome existing faults. See also software maintenance.

Correctness. (1) The extent to which software is free from design defects and from coding defects; that is, fault free. (2) The extent to which software meets its specified requirements. (3) The extent to which software meets user expectations.

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

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

Defect Density. Defect density is a metric used after design and code inspections to judge the quality of the translation of requirements into design. It is defined as the cumulative (over time)
defects encountered divided by the total number of units in the CSCI and the cumulative (over time) defects corrected divided by the total number of units per CSCI.

**Design analysis.** (1) The evaluation of a design to determine correctness with respect to stated requirements, conformance to design standards, system efficiency, and other criteria. (2) The evaluation of alternative design approaches. See also preliminary design.

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

**Design.** (1) The process of defining the software architecture, components, modules, interfaces, test approach, and data for a software system to satisfy specified requirements. (2) The result of the design process.

**Diagnosis.** The functions performed and the techniques used in determining and isolating the cause of malfunctions.

**Diagnostic accuracy.** The degree of correctness with which the diagnostic output agrees with the true state of the item being diagnosed.

**Diagnostic capability.** All the diagnostic characteristics associated with the detection, isolation, and reporting of faults.

**Diagnostic element.** Any distinct, single part of the diagnostic capability, e.g., automatic and manual testing, training, maintenance aiding, and technical information.

**Diagnostic software.** Used to determine operational health of hardware and/or software and report diagnostic information (e.g., health status) according to the diagnostic requirements.

**Diagnostics.** Anything relating to or used in making a diagnosis.

**Documentation.** The documentation indicator identifies potential problems in the deliverable software documentation. This metric is the combined average of the weighted averages for the documentation and source listings in terms of a product's modularity, descriptiveness, consistency, simplicity, expandability, and testability or instrumentation characteristics [AFSCP 87].

**Embedded diagnostics.** That portion of the diagnostic capability that is an integral part of the prime item.

**Error analysis.** (1) The process of investigating an observed software fault with the purpose of tracing the fault to its source. (2) The process of investigating an observed software fault to identify such information as the cause of the fault, the phase of the development process during which the fault was introduced, methods by which the fault could have been prevented or detected earlier, and the method by which the fault was detected. (3) The process of investigating software errors, failures, and faults to determine quantitative rates and trends.

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

**Error data.** A term commonly (but not precisely) used to denote information describing software problems, faults, failures, and changes, their characteristics, and the conditions under which they are encountered.
Error model. A mathematical model used to predict or estimate the number of remaining faults, reliability, required test time, or similar characteristics of a software system. See also error prediction.

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

Error. (1) A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition (ANSI). (2) Human action that results in software containing a fault. Examples include omission or misinterpretation of user requirements in a software specification, incorrect translation or omission of a requirement in the design specification. This is not a preferred usage. See also failure, fault.

Failure rate. (1) The ratio of the number of failures is given unit of measure; for example, failures per unit of time, failures per number of transactions, failures per number of computer runs. (2) In reliability modeling, the ratio of the number of failures of a given category or severity to a given period of time; for example, failures per second of execution time, failures per month. Synonymous with failure ratio.

Failure. (1) The termination of the ability of a functional unit to perform its required function. (2) The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered. (3) A departure of program operation from program requirements.

Failure Density/Fault Intensity. Failure Density is a metric used to monitor faults during the maintenance period after software is released. This metric is a continuation of the fault density/intensity concept and is an estimate of the number of failures that may be discovered after release. The focus of this metric is on monitoring product quality from the customer's aspect.

A failure has been classified as "a manifestation of a fault or a departure, during execution, of the software program from its intended function" [STEP 92]. The inputs for this indicator are obtained during test activities. "The severity and class of failures as well as the software faults that caused the failure are documented and used as the basis for this indicator" [AFSCP 87].

Fault density. Fault density is a metric used after testing to judge how well the requirements have been implemented and to determine if sufficient software testing has been accomplished. It is defined as the cumulative (over time) faults (causes of the faults, not the faults themselves) divided by the total number of units in the CSCI and the cumulative (over time) faults corrected divided by the total number of units per CSCI; where, the average size of a unit is 100 lines of code.

Fault seeding. The process of intentionally adding a known number of faults to those already in a computer program for the purposes of estimating the number of indigenous faults in the program. Synonymous with the bug seeding (faults are typically called defects in the software).

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

Fault. (1) An accidental condition that causes a functional unit to fail to perform its required function. (ISO) (2) A manifestation of an error in software. Synonymous with bug.

Firmware. (1) Computer programs and data loaded in a class of memory that cannot be dynamically modified by the computer during processing (i.e., microcode, microprogram). (2) Hardware that contains a computer program and data that cannot be changed in its user environment. The computer programs and data contained in firmware are classified as software; the circuitry containing the computer program and data is classified as hardware.
Formal language. A language whose rules are explicitly established prior to its use. Synonymous with artificial language. Examples include programming languages, such as FORTRAN and Ada, and mathematical or logical languages, such as predicate calculus. Contrast with natural language.

Formal method. A mathematically sound approach to the specification and design of computer software.

Formal specification. (1) A specification written and approved in accordance with established standards. (2) In proof of correctness, a description in a formal language of the externally visible behavior of a system or system component.

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

Functional decomposition. A method of designing a system by breaking it down into its components in such a way that the components correspond directly to system functions and sub-functions.

Functional specification. A specification that defines the functions that a system or system components must perform. See also performance specification.

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

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

Implementation. (1) A realization of an abstraction in more concrete terms; in particular, in terms of software, or both. (2) A machine executable form of a program, or a form of a program that can be translated automatically to machine executable form. (3) The process of translating a design into code and debugging the code.

Independent verification and validation. (1) Verification and validation of a software product by an organization that is both technically and managerially separate from the organization responsible for developing the product. (2) Verification and validation of a software product by individuals or groups other than those who performed the original design, but, who may be from the same organization. The degree of independence must be a function of the importance of the software.

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

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

Inspection. (1) A formal evaluation technique in which software requirements, design, or code are examined in detail by a person or group other than the author to detect faults, violations of development standards, and other problems. Contrast with walk-through. (2) A phase of quality control that by means of examination, observation or measurement determines the conformance of materials. (3) A phase of quality control that by means of examination, observation or measurement
determines the conformance of materials, supplies, components, parts, appurtenances, systems, processes or structures to predetermined quality requirements.

**Integrated diagnostics.** A structured process that maximizes the effectiveness of diagnostics by integrating pertinent elements, such as testability, automatic and manual testing, training, maintenance aiding, and technical information, as a means for providing a cost effective capability to detect and isolate unambiguously all faults known or expected to occur in weapon systems and equipment in order to satisfy weapon system mission requirements.

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

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

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

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

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

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

**Interface.** (1) A shared boundary. An interface might be a hardware component to link two devices or it might be a portion of storage or registers accessed by two or more computer programs. (ANSI) (2) To interact or communicate with another system component.

**Maintainability.** (1) The ease with which software can be maintained. (2) The ease with which maintenance of a maintenance of a functional unit can be performed in accordance with prescribed requirements (ISO).

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

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

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

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

**Performance evaluation.** The technical assessment of a system or system component to determine how effectively operating objectives have been achieved.
Performance requirement. A requirement that specifies a performance characteristic that a system or system component, must possess; for example, speed, accuracy, frequency.

Performance specification. (1) A specification that sets forth the performance requirements for a system or system component. (2) Synonymous with requirements specification. (U.S. Navy usage) See also functional specification.

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

Process. (1) In a computer system, a unique, finite course of events defined by its purpose or by its effect, achieved under given conditions. (2) To perform operations on data in process. (ISO)

Product Metrics. Product metrics measure aspects relating to quality, customer satisfaction, and difficulty to produce, but "may not reveal anything about how the software has evolved into its current state" [Conte 86]. These indicators include: size - LOC, fault density/intensity, documentation, test sufficiency, prediction accuracy, and customer satisfaction.

Product specification. Synonymous with design specification. (DoD usage)

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

Program specification. Any specification for a computer program. See functional specification, performance specification, requirements specification.

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

Proof of correctness. (1) A formal technique used to prove mathematically that a program satisfies its specifications. See also total correctness. (2) A program proof that results from applying this technique.

Pseudo-code. A combination of programming language and natural language used for computer program design.

Qualification testing. Formal testing, usually conducted by the developer for the customer, to demonstrate that the software meets its specified requirements.

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

Real time. (1) Pertaining to the processing of data by a computer in connection with another process outside the computer according to time requirements imposed by the outside process. This term is also used to describe systems operating in conversational mode, and processes that can be influenced by human intervention while they are in progress. (ISO) (2) Pertaining to the actual time during which a physical process transpires: for example, the performance of a computation during the actual time that the related physical process transpires, in order that results of the computation can be used in guiding the physical process. (ANSI)

Redundancy. The inclusion of duplicate or alternate system elements to improve operational reliability by ensuring continued operation in the event that a primary element fails.
Regression testing. Selective re-testing to detect faults introduced during modification of a system or system component, to verify that modifications have not caused unintended adverse effects, or to verify that a modified system or system component still meets its specified requirements.

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

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

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

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

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

Reliability. The ability of an item to perform a required function under stated conditions for a stated period of time. (ANSI/ASQC A3-1978) (2) See software reliability.

Requirement. (1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. The set of all requirements forms the basis for subsequent development of the system or system component. See also requirements analysis, requirements phase, requirements specification.

Requirements analysis. (1) The process of studying user needs to arrive at a definition of system or software requirements. (2) The verification of system or software requirements.

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

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

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

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

Semantics. (1) The relationships of characters or groups of characters to their meanings, independent of the manner of their interpretation and use. (ISO) (2) The relationships between symbols and their meanings. (ANSI)

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

Size - LOC (Lines Of Code). Because there exists a relationship between the number of LOC and the amount of effort necessary to develop software products, an increase in the total number of LOC can lead to schedule slips, costs overruns, and staffing problems. The amount of estimated new, reused, modified and total CSCI LOC are monitored on a monthly basis. A linear relationship between LOC and cost does not exist since reused or modified code alters the total amount of time or effort necessary for production.

Software development cycle. (1) The period of time that begins with the decision to develop a software product and ends when the product is delivered. This cycle typically includes a requirements phase, design phase, implementation phase, test phase, and sometimes, installation and checkout phase. Contrast with software life-cycle. (2) The period of time that begins with the decision to develop a software product and ends when the product is no longer being enhanced by the developer. (3) Sometimes used as a synonym for software life-cycle.

Software development plan. A project plan for the development of a software product. synonymous with computer program development plan.

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

Software diagnostics. Methods, processes, and techniques applied to software for the development of high assurance programs, fault tolerance, including the support and maintenance of systems.

Software errors. (or document discrepancies) Functional deficiencies where the software operation or implementation (and/or document) does not meet requirements or standards.

Software life-cycle (definition I). The software life-cycle consists of a set of discrete activities occurring in a given order during the development and use of software and software systems. The time periods during which these activities occur are referred to as phase. At the current time a consensus has not developed as to which phases comprise the software life-cycle.

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

Software Maintenance. (1) Modification of a software product after delivery to correct faults. (2) Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment. See also adaptive maintenance, corrective maintenance, perfective maintenance.

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

Software reliability. (1) The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are countered. (2) The ability of a program to perform a required function under stated conditions for a stated period of time.
Software. (1) Computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system. See also application software, system software. (2) Programs, procedures, rules, and any associated documentation pertaining to the operation of a computer system. (ISO)

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

Stability. (1) The ability to continue unchanged despite disturbing or disruptive events. (2) The ability to return to an original state after disturbing or disruptive events.

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

Static analysis. The process of evaluating a program without executing the program. Similar to desk checking, code audit, inspection, static analyzer, walk-through. Contrasts with dynamic analysis.

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

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

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

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

Syntax. (1) The relationship among characters or groups of characters, independent of their meanings or the matter of their interpretation and use. (ISO) (2) The structure of expressions in a language. (ANSI)

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

System design. (1) The process of defining the hardware and software architectures, components, modules, interfaces, and data for a system to satisfy specified system requirements. (2) The result of the system design process.

System reliability. The probability that a system, including all hardware and software subsystems, will perform a required task or mission for a specified environment. See also operational reliability, software reliability.

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

Test bed. A test environment containing the hardware, instrumentation tools, simulators, and other support software necessary for testing a system or system component.
Test repeatability. An attribute of a test indicating whether the same results are produced each time the test is conducted.

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

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

Testability. (1) The extent to which software facilities both the establishment of test criteria and the evaluation of the software with the respect to those criteria. (2) The extent to which the definition of requirements facilitates analysis of the requirements to establish test criteria.

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

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

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

Validation involves checking that the program as implemented meets the expectations of the software customer in such a way to ensure compliance with software requirements. See also verification.

Verification. (1) The process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase. See also validation. (2) Formal proof of program correctness. See proof of correctness. (3) The act of reviewing, inspecting, testing, checking, auditing, or otherwise establishment and documenting whether or not items, processes, services, or documents conform to specified requirements. (ANSI/ASQC A3-1978).

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