# GLOSSARY

A man is born alone and dies alone; and he experiences the good and bad consequences of his karma alone; and he goes alone to hell or the supreme abode.

—Chanakya

**Abstract syntax tree (AST)** An abstract syntax tree is a tree representation of the syntax of some source code that has been written in a programming language. Each node of the tree denotes a construct occurring in the source code. The tree is abstract in the sense that it may not represent some constructs that appear in the original source. An AST is often built by a parser as part of the processing of compiling source code.

**Accidental reuse** This informal practice, in which components are selected from general libraries, is usually called opportunistic reuse or accidental reuse.

**Activity** An action of one of the following types: an investigation activity, a modification activity, a management activity, or a quality assurance activity. An activity may be made up of a number of sub-activities. Usually, it takes as input one or more existing artifacts and outputs zero, one, or many new or modified artifacts.

**Actual impacted set (AIS)** The set of components actually modified as the results of performing a change request.

**Adaptive maintenance** The process that modifies the software to properly interface with a changing or changed environment. Adaptive maintenance includes

*Software Evolution and Maintenance: A Practitioner's Approach*, First Edition.
Priyadarshi Tripathy and Kshirasagar Naik.

system changes, additions, deletions, modifications, extensions, and enhancements to meet the evolving needs of the environment in which the system must operate.

**Adequacy** It is the capability of an impact analysis approach to produce a set of potentially affected elements that includes all the objects to be modified.

**Adherence** It is the ability of the approach to produce a CIS which is as adherent as possible to the AIS. The smaller the difference between these two sets, the smaller the number of candidate objects that will fail to be included in the actual modification, and the smaller the effort required to specify the change. Adherence is expressed by means of the following ratio called *S-Ratio*:

$$S\text{-}Ratio = \frac{|\,AIS\,|}{|\,CIS\,|}.$$

If the approach is adequate, AIS is included in CIS, so that $|$ AIS $| \leq |$ CIS $|$ and *S-Ratio* varies in the range from 0 to 1.

**Agile software development** It is an iterative and incremental (revolutionary) approach to software development which is performed in a highly collaborative manner with "just enough" ceremony that produces high quality software. Agile methods refer to a collection of "lightweight" software development methodologies that are basically aimed at minimizing risk and achieving customer satisfaction through a short feedback loop.

**Annotation of the program** If one views the development process of the program as a layered system where the top layer is the set of program goals and the bottom layer is source code, then the annotation can be viewed as the abstraction of the middle layer connecting the top and the bottom layers. Basically, a program annotation is used to bridge the abstraction gap between the top and the bottom layers.

**Antiregression work** A term introduced by Lehman and Belady to describe the work done to decrease the complexity of a program without altering the functionality of the system as perceived by the users. Anti-regressive work includes activities such as code rewriting, refactoring, reengineering, restructuring, and redocumenting.

**Application engineering** The process of constructing or refining application systems by reusing assets.

**Application gateway** An *application gateway* is used for a semidecomposable LIS. This gateway is positioned between separable user and system interfaces and the legacy application.

**Artifact** Artifacts that together correspond to a software product can be of the following types: document that can be subdivided into textual and graphical documents, component off-the-shelf products, and object code components. Textual documents include source code listings, plans, design, and requirements specifications.

**Aspect** A modular unit designed to implement a (crosscutting) concern. In other words, an aspect provides a solution for abstracting code that would otherwise be spread throughout, that is, crosscut the entire program.

**Aspect-oriented software development** An approach to software development that addresses limitation inherent in other approaches, including object-oriented programming. The approach aims to address crosscutting concerns by providing means for systematic identification, separation, representation, and composition. Crosscutting concerns are encapsulated in separate modules, known as aspects, so that localization can be promoted. This results in better support for modularization, thereby reducing development, maintenance, and evolution costs.

**Assembler** Refers to a computer program to translate between lower level representations of computer programs. The original sequence is usually called the assembly language and the output called object code.

**Assertion** An assertion is a Boolean statement which should never be false or can be false only if an error has occurred. In other words, an assertion is a check on a condition which is assumed to be true, but it can cause a problem if it is not true.

**Asset** An item, such as design, specifications, source code, documentation, test suites, manual procedures, that has been designed for use in multiple contexts.

**Backward wrappers** Backward wrappers emulate the legacy technology on top of the new database.

**Bad smell** A bad smell is a structure in the code that suggests— and sometimes even scream for—opportunities for refactoring.

**Baseline** Specification or product that has been formally reviewed and agreed upon that thereafter serves as the basis for further development, and can be changed only through a change control procedure.

**Beacon** A beacon is a code text that gives a clue to the computation being performed in a code block.

**Big bang** It is a reengineering approach to replace an entire system at one. It is also known as Cold turkey strategy.

**Black-box testing** It is a method of software testing that examines the functionality of an application without peering into its internal structures or workings.

**Butterfly** A methodology to migrate a mission-critical LIS to a target system in a simple, fast, and safe way. The methodology eliminates, during the migration, the need of simultaneously accessing both the legacy and target systems, and, therefore, avoids the complexity of maintaining the consistency between these two information systems.

**Call graph** It is a directed graph in which nodes represent functions (components or methods) and an edge from node $A$ to node $B$ means that $A$ may call $B$.

**Candidate impact set (CIS)** The set of components estimated to be affected according to a particular impact analysis approach. It is also known as estimated impacted set.

**Change control** Change control is the process responsible for evaluating the results of maintenance event investigations and deciding whether or not to approve a product modification.

**Change coupling** It is the implicit dependency between two or more software entities that have been observed to frequently change together during the evolution of a system. This co-change information can either be present in the versioning system, or must be inferred by analysis.

**Change management** The process of making changes to software and controlling their effects during the entire life cycle of the software.

**Change propagation activity** It ensures that a change made in one component is propagated properly throughout the entire system.

**Chicken little** This migration methodology is a refinement of the composite database migration approach.

**Chunk** A chunk is a block of related code segment. The concept of program chunks enable programmers to create higher levels of abstractions from the lower level of abstractions.

**Chunking** A programmer creates higher level abstraction structures by combining lower level chunks. The process of creating higher level of chunks is called chunking.

**Cliché** A cliché is a pattern that appears frequently in programs (e.g., algorithms, data structures, domain-specific patterns).

**Closed source software (CSS)** Closed source software is developed by a single person or company. Only executable code is made available, while the all important source code is kept secret. The software is normally copyrighted or patented and is legally protected as intellectual property. The owner of the software distributes the software directly or via vendors to the end user.

**Cold turkey** It is a migration strategy that rewrites LIS from scratch using modern architecture, tools, and databases, running on a new platform.

**Code decay** Code decay is antithesis of evolution. While the evolution process involves progressive changes, the changes are degenerative in the case of decay.

**Code churn** It is the amount of code change taking place within a software unit over time.

**Code smell** A code smell is a symptom in the source code of a software that possibly indicates a deeper problem.

**Commercial off-the-shelf (COTS) components** Software components produced by third-party vendor organizations, that can be reused in a system, are known as commercial off-the-shelf components. Often, these types of components are delivered without their source code.

**Compilation** It refers to the translation of source code into object code by a compiler. A compiler is a computer program that translates text written in one computer language (the source language) into another computer language (the target language). The original sequence is usually called the source code and the output is called object code.

**Composite database** It is a migration methodology in which the LIS and its target IS operated in parallel throughout the migration project. The target applications are gradually rebuilt on the target platform using software tools and technology.

**Concepts** Concepts are units of human knowledge that can be processed by the human mind in one instance.

**Conceptual schema** A conceptual schema of a database is an abstract, computer-independent description of the information that the data implement. A conceptual schema expressed into the entity-relationship model comprises entity types, relationship types, attributes and various properties and constraints that translate the concepts and structures of the application domain.

**Configuration** A configuration is a set of related items (a.k.a. configuration items) satisfying three criteria: (i) the configuration is uniquely identifiable; (ii) the items are consistent, that is, the items work with one another in a way that is well understood; and (iii) the set of items is recreatable as a unit.

**Configuration item** It is an elementary part (usually a file) of the configuration that must be: (i) identified or versioned; (ii) tracked; and (iii) controlled.

**Configuration management** The set of activities that is developed to manage changes to the software's documentation and code throughout the software life cycle.

**Construction** The process of writing, assembling, or generating assets.

**Control flow analysis** It refers to the order in which the individual statements, instructions, or function calls of an imperative or functional program are executed or evaluated.

**Corrective maintenance** A process that includes isolation and correction of defects in the software. The correction repairs the software product to satisfy requirements.

**Crosscutting concerns** Concerns that do not fit within the dominant decomposition of a given software system, and as such have an implementation that cuts across that decomposition. Aspect-oriented programming is intended to be a solution to modularize such crosscutting concerns.

**Cross-referencing** It means being able to link elements of different abstraction levels. This helps in building a mental model of the program under study.

**Customer** The person or persons for whom the product is intended, and usually, but not necessarily, who decides the requirements.

**Cut over** It is the last step in the migration process from the LIS to the target system.

**Data administration** Data administration or data resource management is an organizational function working in the areas of information systems that plans, organizes, describes, and controls data resources.

**Data conversion** Data conversion is the migration of the data instance from the old database to the new one.

**Database gateway** Both the *forward gateway* and *reverse gateway* are also known as *database gateways*, since it encapsulates the entire database service and database from the perspective of application modules.

**Database reverse engineering** It is a software engineering process through which one tries to understand and redocument the files and/or the database of an application. In the process, conceptual schema is recovered from the application.

**Data flow analysis** Data flow analysis is a technique for gathering information about the possible set of values calculated at various points in a computer program. A program's control flow graph (CFG) is used to determine those parts of a program to which a particular value assigned to a variable might propagate. The information gathered is often used by compilers when optimizing a program.

**Data reverse engineering (DRE)** The use of structured techniques to reconstitute the data assets of an existing system. The two vital aspects of the DRE process are to: (i) recover *data assets* that are useful or valuable; and (ii) *reconstitute* the recovered data assets to make them more useful. Thus, DRE can be regarded as adding value to the existing data assets, making it easier for organizations to conduct business efficiently and effectively.

**Decompilation** It is a tool by which a high level source code of an executable program is discovered. A decompiler is the name given to a computer program that performs the reverse operation to that of a compiler.

**Design recovery** It aims at recreating design abstractions from the source code, existing documentation, experts' knowledge, and any other source of information. In design recovery the domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system to identify meaningful higher level abstractions beyond those obtained directly by examining the system itself.

**Decay** Decay is the antithesis of evolution. While the evolution process involves progressive changes, the changes are degenerative in the case of decay.

**Development model** A framework used to guide the set of activities performed to translate user needs into software product.

**Development process** One of the primary processes of the ISO/IEC life cycle model. It provides guidance for the development of software.

**Development technology** The technology used when the product and its constituent artifacts were originally constructed. The original development technology constraints the possible maintenance procedures.

**Disassembler** A disassembler is a computer program that translates machine language into assembly language—the inverse operation to that of an assembler. Disassembly, the output of a disassembler, is often formatted for human readability rather than suitability for input to an assembler, making it principally a reverse engineering tool.

**Discovered impact set (DIS)** New impact components that are discovered during the implementation of the change request which are not included in the candidate impact set (CIS).

**Domain** A problem space. A sphere of activity, concern, or function.

**Domain analysis** Domain analysis is the process of analyzing related software systems in a domain to find their common and variable parts. Domain analysis is the first phase of domain engineering. It is a key method for realizing systematic software reuse.

**Domain design** It covers the development of a common architecture for all the members of the system family and a plan of how individual systems will be created based on the reusable assets.

**Domain engineering** A reuse-based approach to defining the scope, specifying the structure, and building the assets for a class of systems, subsystems, or applications. Domain engineering may include the following activities: domain definition, domain analysis, domain design, and domain implementation.

**Domain implementation a.k.a. product line** The process of creating adaptable assets that can be reused in the development of software systems within a domain. Domain implementation may also include the specification of a software development process that describes how software systems in the domain are developed through reuse of assets.

**Domain model** A product of domain analysis that provides a representation of the requirements of the domain. The domain model identifies and describes the structure of data, flow of information, functions, constraints, and controls within the domain that are included in software systems in the domain. The domain model describes the commonalities and variabilities among requirements for software systems in the domain.

**Domain-specific language** A problem- or task-oriented modeling language used to simplify the process of assembling, customizing, generating, or configuring a system or component.

**Effectiveness** It is the ability of the impact analysis approach to produce results that actually benefit the maintainer's work. The abstract definition of *effectiveness* is refined by means of three characteristics: *Ripple-sensitivity*, *Sharpness*, and *Adherence*.

**Equivalence tests** The equivalence test aims to ensure that the behavior of the software system after a maintenance activity of one or more of its components is exactly the same as before the change.

**E-type program** The distinctive properties of E-type programs are: (i) the problem that they address cannot be formally and completely specified; (ii) the program has an imperfect model of the operational domain embedded in it; (iii) the program reflects an unbounded number of assumptions about the real world; (iv) the installation of the program changes the operation domain; and (v) the process of developing and evolving E-type software is driven by feedback.

**Event management** Event management is the process responsible for handling the stream of events received by the maintenance organization.

**Evolution process** A software process model that explicitly takes into account the iterative and incremental nature of the software development.

**External reuse (a.k.a. public reuse)** The portion of a product which was constructed externally.

**Extreme programming (XP)** This is a software development methodology which is self-adaptive and people oriented. It is a specific instance of agile software development that aims to simplify and expedite the development of new software in a volatile environment of rapidly changing requirements. XP begins with five values: communication, feedback, simplicity, courage, and respect. It then builds up 12 rules/recommendations, which XP projects should follow.

**False positive impact set (FPIS)** The set of components in the candidate impact set (CIS) that are not impacted by the implementation of a change request.

**Feedback** In engineering, feedback refers to the case when at least some part of the output(s) of the system are fed back to the input, normally for control purposes. In systems thinking and related disciplines, that is system dynamics, feedback describes a property of many complex systems in which the outputs determine the inputs.

**Forward engineering** Forward engineering is the traditional process of moving from high level abstractions and logical, implementation-independent designs to the physical implementation of a system.

**Forward gateway** The forward gateway translates and redirects LIS application calls to the new database service. The LIS interoperates with its target system through a *forward gateway* while legacy applications and interfaces are being reengineered.

**Forward migration** It is a type of migration methodology which involves the initial migration of database including the data to a modern DBMS then gradually migrating the legacy application program and interfaces. It is also known as database first approach.

**Forward wrappers** The forward wrappers translate data and queries from the legacy data model and interface to those expected by the new components.

**Framework** A set of cooperating classes or frames that makes up a reusable design for a specific class of software.

**Free and open source software (FOSS)** Software of which the source code is available for users and third parties to be inspected and used. It is made available to the general public with either relaxed or non-existent intellectual property restrictions. It is generally used as a synonym of free software even though the two terms have different connotations. *Free* emphasizes the freedom to modify and redistribute under the terms of the original license while open emphasizes the accessibility to the source code.

**Fully decomposable information system** A fully decomposable information system is one where applications, interfaces and databases are considered to be distinct components with clearly defined interfaces. Applications must be independent of each other and interact only with the database service.

**Gateway** A software module introduced between operational software components to mediate between them.

**Glue** A glue is a piece of code that one builds to combine different components.

**Graph exchange language (GXL)** It is an XML-based standard exchange format for sharing data between tools. Formally, GXL represents typed, attributed, directed, ordered graphs which are extended to represent hypergraphs and hierarchical graphs. The GXL has been ratified by reengineering and graph transformation research communities and is being considered for adoption by other communities.

**Graph transformation (a.k.a. graph rewriting)** A theory and a set of associated tools that allows to modify graph-based structures by means of transformation rules, and to reason about the formal properties of these rules.

**Hazard** State of a system or a physical situation which, when combined with certain environmental conditions, could lead to an accident or mishap. A hazard is a prerequisite for an accident.

**Horizontal reuse** A type of reuse where assets are used across domains.

**Horizontal (a.k.a. external) traceability** It refers to the ability to trace artifacts between different models.

**Horseshoe** It is a visual metaphor to describe a three-step architectural reengineering process. The first step—represented on the left side of the horseshoe—aims at extracting the architecture from the source code (abstraction principle). The second part of the process—represented in the upper part of the horseshoe—is related to architecture transformation toward the target architecture (alteration principle). The last part—on the right side of the horseshoe—represents the instantiation of the new architecture (refinement).

**Hypothesis** Programmers test the results of their program understanding as a conjecture.

**Impact analysis (IA)** Task of identifying potential consequences of a change, or estimating what needs to be modified to accomplish the change.

**Incremental development** It is a staging and scheduling strategy in which the various parts of the system are developed at different times or rates, and integrated as they are completed.

**Information system (IS)** It is often used to designate large-scale business applications.

**Information system gateway** It is a kind of gateway for nondecomposable systems which are positioned between end-user and other information systems and LIS. An information system gateway encapsulates the whole functionality of the legacy system. It is a primary means for dealing with the migration of user interface.

**Internal reuse (a.k.a. private)** The extent to which modules within a product are reused within the same product.

**Investigation activity** An activity that assesses the impact of undertaking a modification arising from a change request.

**Iterative development** It is a rework scheduling strategy in which time is set aside to revise and improve parts of the system.

**Knowledge base** A knowledge base is a body of information about the program.

**Knowledge elements** A knowledge element is anything that is useful in understanding a program.

**Knowledge structure** It specifies principal elements in a domain and includes mechanisms which facilitate the comprehension and generation process. It is used to organize complex entities or concepts into constituents.

**Legacy system a.k.a. Legacy information system (LIS)** A legacy system is any system that significantly resists modifications and change. It may have been developed using an outdated programming language or an obsolete development method. Most likely it has changed hands several times and shows signs of many modifications and adaptations.

**Lexical analysis** It is the process of converting a sequence of characters into a sequence of tokens. Programs performing lexical analysis are called lexical analyzers.

**Logical schema** The description of the data structures as they are implemented by the data manager, and they are seen by the application programmer. For example, the logical schema of a relational database describes its tables, columns, and primary and foreign keys as well as all the explicit and implicit constraints to which the data are submitted.

**Maintainer** An organization or a personal who performs maintenance activities.

**Maintenance event** A problem report or change request originating from a customer or user of the maintained product or a member of the maintenance organization.

**Maintenance management** The process used to manage the maintenance service as opposed to the procedure used to manage individual maintenance requests. The organization process is established and maintained by senior maintenance managers. It is responsible for defining the structure of the maintenance organization such that it can fulfill its service level agreement. Maintenance management has three main concerns other than the normal concerns of quality assurance and project management: event management, configuration control, and change control.

**Maintenance model** A framework used to guide the set of activities to perform maintenance.

**Maintenance organization structure** The roles undertaken by maintenance human resources in a maintenance organization in order to perform the required administrative procedures.

**Maintenance process** One of the primary processes of the ISO/IEC life cycle model. It provides guidance for the maintenance of software.

**Management activity** An activity related to the management of the maintenance process or to the configuration control of the maintained product.

**Mental model** It describes a programmer's mental representation of the program to be comprehended.

**Method** A systematic procedure defining steps and heuristics to permit the accomplishment of one or more activities.

**Metric** A metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.

**Migration** A variant of reengineering in which the transformation is driven by a major technology change is called *migration*.

**Migration of information legacy system** LIS migration basically moves an existing, operational system to a new platform, retaining the legacy system's functionality and causing minimal disruption to the existing operational business environment.

**Mishap** Also called an accident, an unintended event that results in death, injury, illness, damage or loss of property, or harm to the environment.

**Model** A model is a simplified representation of a system on a higher level of abstraction. It is an abstract view on the actual system emphasizing those aspects that are of interest to someone. Depending on the system under consideration, we talk about, for example, *software models* (for software systems) and *database model* (for database systems).

**Model-driven engineering** A software engineering approach that promotes the use of models and transformations as primary artifacts throughout the software development process. Its goal is to tackle the problem of developing, maintaining and evolving complex software systems by raising the level of abstraction from source code to models. As such, model-driven engineering promises reuse at the domain level, increasing the overall software quality.

**Modification activity** An activity that takes one or more input artifacts and produces one or more output artifacts that, when incorporated into an existing system, change its behavior or implementation.

**Modification request** The means by which problems are reported and enhancements are requested. It is also known as change request.

**Nondecomposable information system** A nondecomposable information system is one in which no functional components are separable.

**Non-functional requirements** Non-functional requirements are requirements which specify criteria to be used to judge the operation of a system, rather than specific behavior. Non-functional requirements are often called qualities of a system.

**Normalization** A normalization reduces a program to another program in a sublanguage, with the purpose of decreasing its syntactic complexity. Elimination of GOTO and module flattening in a program are examples of program normalization.

**Open source software (OSS)** Software of which the source code is available for users and third parties to be inspected and used. It is made available to the general public with either relaxed or non-existent intellectual property restrictions. It is generally used as a synonym of free software even though the two terms have different connotations. *Open* emphasizes the accessibility to the source code, while *free* emphasizes the freedom to modify and redistribute under the terms of original license.

**Obfuscation** It is a transformation that makes a program harder to understand. Programs known as obfuscators operate on source code, object code, or both, mainly for the purpose of deterring reverse engineering, disassembly, or decompilation.

**Optimization** An optimization is a transformation that improves the run-time and/or space performance of a program.

**Paradigm** The philosophy adopted during the original construction of the maintained product, for example, the object-oriented paradigm or procedural paradigm. The original paradigm constrains the possible maintenance procedures.

**Perfective maintenance** The process that improves the software in terms of performance, processing efficiency, or maintainability. These activities may include restructuring the code, creating and updating documentations, or tuning the system to improve performance.

**Physical schema** A physical schema of a database is the implementation of its logical schema. It is the technical descriptions of a database where all the physical constructs, such as indices, and parameters, such as page size or buffer management policy, are specified.

**Plan** A plan is an abstract representation of a cliché.

**Portfolio analysis** The portfolio analysis establishes measures of technical quality and business value for a set of software systems and evaluate this against the measure on a chi-square chart.

**Precision** In data mining or information retrieval, precision is defined as the proportion of retrieved and relevant data or documents to all the data or documents retrieved:

$$precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|}$$

Precision is a measure of how well the technique performs in not returning non-relevant items. Precision is 100% when every document returned to the user is relevant to the query. Being very precise usually comes at the risk of missing documents that are relevant, hence precision should be combined with recall.

**Preventive maintenance** Software maintenance performed for the purpose of preventing problems before they occur. It is an activity during which one attempts to prevent an unnecessary change in the future.

**Principle of abstraction** The gradual increase in the abstraction level of a system representation is created by the successive replacement of existing detailed information with information that is more abstract. Abstraction produces a representation that emphasizes certain system characteristics by suppressing information about others.

**Principle of alteration** Alteration is the making of one or more changes to a system representation without changing the degree of abstraction, including addition, deletion, and modification of information.

**Principle of refinement** The gradual decrease in the abstraction level of a system representation is caused by the successive replacement of existing systems with more detailed information.

**Procedure** The conduct followed to perform an activity. A procedure may be classified as a method, technique, or script. A procedure may be adopted to perform a specific activity from a set of possible procedures.

**Process** A process is a collection of related activities to accomplish some task. A process can be described by creating a process model.

**Process model** A process can be described by creating a process model. A process model represents key relationships among a variety of objects, such as activities, data objects, tools, and human roles within a process. In other words, it is a framework to identify, define, and organize data, strategies, rules, and processes needed to support the way an organization wants to do business. Process models are important vehicles for understanding, evaluating, reasoning, and improving process.

**Product** A product is a software application, product, or package that is undergoing modification. A product is a conglomerate of a number of different artifacts.

**Product upgrade** A change to the baseline product that implements or documents a maintenance activity. An upgrade may be a new version of the product, an object code patch, or a restriction notice.

**Program analysis** Program analysis reduces a program to one aspect, such as its control flow or data flow. Analysis can thus be considered a transformation to a sublanguage or an aspect language.

**Program comprehension (a.k.a. program understanding)** It is a process of knowledge acquisition about the program. It is the task of building mental models of an underlying software system at various abstraction levels, ranging from models of the code itself to ones of the underlying application domain, for software maintenance, evolution, and reengineering purposes.

**Program dependency graph (PDG)** The program dependency graph (PDG) of a program has one node for each simple statement and one node for each control predicate expression. It has two types of directed edges—data dependency edges and control dependency edges. A data dependency edge from vertex $v_i$ to vertex $v_j$ implies that the computation performed at vertex $v_i$ directly depends on the value computed at vertex $v_j$. A control dependency edge from $v_i$ to $v_j$ means that node $v_i$ may or may not be executed depending on the Boolean outcome of the predicate expression at node $v_j$.

**Program families** A set of programs whose common properties are so extensive that it becomes advantageous to study the common properties of these programs before analyzing individual differences.

**Program migration** In migration, a program is transformed to another language at the same level of abstraction. This can be a translation between dialects, or a translation from one language to another.

**Program rephrasing** Program rephrasing is source-code-level changes. Rephrasings are transformations that transform a program into a different program in the same language.

**Program slicing** Program slicing is a method used by experienced computer programmers for abstracting from programs. Starting from a subset of a program's behavior, slicing reduces that program to a minimal form which still produces that behavior. The reduced program, called a "slice," is an independent program guaranteed to faithfully represent the original program within the domain of the specified subset of behavior.

**Program translation** Program translation is a source-code-level change. In program translation a program is transformed from a source language into a program in a different target language.

**Protocol analysis** It is a methodology for eliciting verbal reports from participants about their thought sequences as a valid source of data on thinking.

**Protocol data** The thinking-aloud action of the programmer can be recorded and transcribed. The transcribed verbal report is called protocol data.

**P-type program** It is based on a practical abstraction of the problem rather than on a completely defined specification. Even though the exact solution may exist, the solution produced by a P-type program is tempered by the environment in which it must be produced. The solution of a P-type program is acceptable if the results make sense to the stakeholder(s) in the world in which the problem is embedded.

**Quality assurance activity** An activity aimed at ensuring that a modification activity does not damage the integrity of the product being maintained. Quality assurance activities may be classified as testing.

**Recall** In data mining or information retrieval, recall is defined as the proportion of relevant data or documents retrieved, out of all relevant data or documents known or available:

$$precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|}$$

Recall is 100% when every relevant item is retrieved. In theory, it is easy to achieve good recall: simply return every item in the collection, thus recall by itself is not a good measure and should be combined with precision.

**Reclamation** Reclamation is the process of reclaiming something from loss or from a less useful condition. Some technical articles use this term as a synonym of reengineering.

**Re-code** In the context of software engineering, re-code involves changing the implementation characteristics of the source code.

**Re-design** In the context of software engineering, re-design involves changing the design characteristics. Possible changes include restructuring a design architecture, altering a system's data model as incorporated in data structures or in a database, and improving an algorithm.

**Redocumentation** Redocumentation aims at producing/revising alternate views of a given artifact, at the same level of abstraction, for example, pretty printing source code or visualizing control flow graph. It is a weak form of *restructuring*.

**Reengineering** Software reengineering is an activity that: (1) improves one's understanding of software, or (2) prepares or improves the software itself, usually for increased maintainability, reusability, or evolvability.

**Refactoring** Refactoring is the object-oriented equivalent of restructuring. Refactoring is a change to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

**Rehosting** It means reengineering of source code without addition or reduction of features in the transformed targeted source code. It includes porting, migration, or conversion of the existing code and data from one computing platform to another, from one operating system to another, from one language to another, or all of the above, but no new features are added to the code.

**Rejuvenation** It is a periodic preemptive rollback of continuously running applications to prevent failures in the future. Rejuvenation may sometimes increase the downtime of the application, however it prevents the occurrence of more severe and costly failures.

**Release** A release is a version of a software system that has been approved and distributed to users outside the development team.

**Replace** It is a type of software reengineering strategy that incorporates the principles of abstraction and refinement. To change an existing system characteristic, abstraction is used to reconstruct a system representation at a level of abstraction that contains no information about the characteristic. Refinement is then used to create a suitable target system representation at a lower level of abstraction.

**Repository** A kind of database, or file system in which the version history of a software system are stored. The repository may be used to store source code, executable code, documentation or any other type of software artifacts of which different versions may exist over time or even at the same time.

**Resource** Everything that is used to perform an activity. Resources may be hardware, software, and human resources.

**Respecify** In the context of software engineering, respecify involves changing the requirement characteristics. This type of changes can refer to changing only the form of existing requirements that is taking informal requirements expressed in English and generating a formal specification expressed in a formal language such as SDL or UML. This type of change can also refer to changing system requirements, such as the addition of new requirements, or the deletion or alteration of existing requirements.

**Restructuring** Restructuring is the transformation from one representation form to another at the same relative abstraction level, while preserving the system's external behavior.

**Rethink** In the context of software engineering, rethink involves changing the conceptual characteristics. This type of change can result in drastic changes to a system. Rethinking a system means manipulating the concepts embodied in an existing system to create a system that operates in a different problem domain.

**Reusability** The degree to which an asset can be used in more than one software system, or in building other assets.

**Reuse** The use of an asset in the solution of a different problem.

**Reuse capability** The range of expected results in reuse proficiency, efficiency, and effectiveness that can be achieved by an organization's process.

**Reuse effectiveness** It is the ratio of the reuse benefit to the reuse cost.

**Reuse efficiency** It is the ratio of the actual reuse opportunities exploited to the organizations targeted reuse opportunity.

**Reuse maturity model** It is a self-assessment and planning aid for improving an organization's reuse capability. A reuse maturity model is at the core of planned reuse, helping organizations understand their past, current, and future goals for reuse activities.

**Reuse opportunity** It is an occasion where an asset (existing or to be developed for reuse) may satisfy a need (current or anticipated). Within the set of reuse opportunities there are potential reuse opportunities and targeted reuse opportunities. Potential reuse opportunities are the set of reuse opportunities that will result in actual reuse when exploited. Targeted reuse opportunities are the set of reuse opportunities toward which an organization directs its effort implicitly and explicitly. A targeted reuse opportunity may not always be a potential reuse opportunity.

**Reuse proficiency** It is the ratio of the value of actual reuse opportunities exploited to the value of potential reuse opportunities.

**Reuse library** A classified collection of assets that allows searching, browsing, and extracting.

**Reverse engineering** In the context of software, reverse engineering is defined as the process of analyzing a subject system to: (i) identify the system's components and their interrelationships; and (ii) create representations of the system in another form or at a higher level of abstraction.

**Reverse gateway** A *reverse gateway* enables target applications to access the LIS database. It is employed to convert calls from the newly created applications and redirect them to the legacy database service.

**Reverse migration** It is a type of migration methodology in which the LIS applications are gradually migrated to the target platform while the legacy database remains on the original platform. The LIS database migration is the final step of the migration process. It is also known as database last approach.

**Rework** This is a type of software reengineering strategy that incorporates the principles of abstraction, alteration, and refinement. To change an existing system's characteristics, abstraction is used to reconstruct a system representation at the appropriate abstraction level. Alteration is then used to transform the reconstructed system representation into the target system representation at the same level of abstraction. Finally, refinement is used to create a suitable target system representation at a lower level of abstraction.

**Rewrite** It is a type of software reengineering strategy that incorporates only the principle of alteration. To change an existing system, alteration is used to transform the existing system, represented at some level of abstraction, into the target system, represented at the same abstraction level.

**Ripple effect** The effect caused by making a small change to a system which affects many other systems.

**Ripple-sensitivity** It is the property of the approach to produce results that are affected by the *ripple effect*. From the point of view of an impact analysis approach, an indicator of the ripple effect may be the cardinality of the set of objects that are indirectly impacted by the change (IIS) (a.k.a. secondary impacted set), given the set of objects that are directly affected by the change (DIS) (a.k.a. primary impacted set). The software maintenance engineer expects the cardinality of IIS to be not very far from the cardinality of the direct impact set; in other words,

$$Amplification = \frac{|\, IIS \,|}{|\, DIS \,|} \longrightarrow 1$$

where | IIS | is the cardinality of the indirectly impact set and | DIS | is the cardinality of the direct impact set. Ripple-sensitivity is expressed by the *Amplification* ratio. If *Amplification* is much greater than 1, this means that the approach produces a much larger indirect impact set than the direct impact set.

**Rules of programming discourse** It specifies the conventions, also called "rules," that programmers follow while writing code.

**Scattering and tangling** Occur when the code needed to implement a given concern is spread out (scattered) over and clutters (is tangled with) the code needed to satisfy one or more other concern. Scattering or tangling are typically the results of a program's inability to handle what is called a crosscutting concern.

**Schema** It is a knowledge structure for describing program behavior in a specific domain of activity.

**Script** A guideline for constructing and amending a specific type of document.

**Screen scrapping** It is a wrapping technique that replaces old, text-based interfaces with new graphical interfaces.

**Semidecomposable information system** A semidecomposable information system is one where only the user and the system interfaces are separate components, but applications and database service are not separable.

**Service-level agreement (SLA)** An agreement between the providers of a maintenance service and the customers of a maintenance service that specifies the performance targets for the maintenance service.

**Service-oriented Architecture (SOA)** In software engineering, an SOA is a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined business functionalities that are built as software components that can be reused for different purposes.

**Sharpness** It is the property of the impact analysis approach to avoid including all the objects or components belonging to the software system in the candidate impact set, unless they really should be included. Sharpness is expressed by means of the following ratio called *Change Rate*:

$$Change\ Rate = \frac{|\ CIS\ |}{|\ System\ |}$$

where $|\ System\ |$ is the cardinality of the set of the software system objects; $|\ CIS\ |$ is the cardinality of the candidate impact set. CIS is of course included in “System,” so that | CIS | ≤ | System | and *Change Rate* falls in the range from 0 to 1.

**Side effect** It is an error or other undesirable behavior that occurs as a result of a modification.

**Soft-goal graph** A soft-goal graph for a quality attribute is a hierarchical graph rooted at the desired change in the attribute. Internal nodes represent successive refinements of the attribute and are the soft goals. The leaf nodes represent refactoring transformations which fulfill or contribute positively/negatively to soft goals which appear above them in the hierarchy.

**Software aging** Software product aging is degradation in software code and documentation quality by frequent maintenance, whereas software process execution aging manifests as degradation in performance or transient failures in a continuously running software system.

**Software clone** Copying code fragments and then reuse by pasting with or without modifications is called software cloning and pasted code fragment (with or without modifications) is called a clone of the original.

**Software clone detection** The activity of locating duplicates or fragments of code with a high degree of similarity and redundancy.

**Software configuration management (SCM)** It is configuration management applied to software systems.

**Software entropy** The combination of defect repairs and enhancements tends to gradually degrade the structure and increase the complexity of the software application. The increase in software complexity (e.g., Cyclomatic complexity) over time is called software *entropy*.

**Software evolution** It is defined as “the applications of software maintenance activities and processes that generate a new operational software version with a changed customer-experienced functionality or properties from a prior operational version, where the time period between versions may last from less than a minute to decades, together with the associated quality assurance activities and processes, and with the management of the activities and processes.”

**Software life cycle objects (SLOs)** SLOs are work products representing documents of one kind or another containing varied levels of software engineering information.

**Software maintenance** The totality of activities required to provide cost-effective support to a software system. Activities are performed during pre-delivery stage as well as the post-delivery stage. Note that pre-delivery activities include planning for post-delivery operations, supportability, and logistics determination. Post-delivery activities include software modification, training, and operating help desk.

**Software migration** An activity of the ISO/IEC 14764 maintenance process. It includes the activity of moving an old system to a new operational environment.

**Software reclamation** A technique to convert the existing software into standard building blocks (objects) to form new programs.

**Software retirement** An activity of the ISO/IEC 14764 maintenance process. It involves removing a product from service in an orderly manner once it has outlived its usefulness.

**Stability measures** The resistance to the potential ripple effect that a program would have when it is modified.

**Stakeholder** Person or organization that influences a system's behavior or is impacted by the system.

**Starting impact set (SIS)** The initial set of components thought to be affected by a software change request.

**Strategy** A strategy is a planned sequence of actions to reach a specific goal. A strategy is formulated by identifying actions to achieve a goal.

**S-type program** It implements solution to the problems that can be completely and unambiguously specified, for which, in theory at least, a program implementation can be proven to be correct with respect to the specification. The definition of the S-type requires that the program is correct in the full mathematical sense relative to the specification.

**Syntactic analysis** It is the process of analyzing a sequence of tokens to determine their grammatical structure with respect to a given (more or less) formal grammar.

**Systematic reuse** The practice of reuse according to a well-defined repeatable process.

**Tailoring** Tailoring is a piece of code to enhance the functionality of a component. Tailoring is done by adding some elements to a component to enrich it with a new functionality not provided by the original component.

**Technique** A procedure used to accomplish an activity that is less rigorously defined than a method.

**Text-structure** Code text and its structure are known as text-structure. Understanding text-structure is the beginning of program comprehension.

**Thinking aloud** It reflects the knowledge currently active in the programmer's working memory.

**Usability** A measure of an executable software unit's or system's functionality, case of use, and efficiency.

**User** The person or persons operating or interacting directly with the system. The user often states requirements to the customer, supplier, or maintainer.

**Version** A version is a snapshot of a certain software system at a certain point in time. Whenever a change is made to the software system, a new version is created. The version history is the collection of all versions and their relationships.

**Version control** It is a software tool to retrace and restore programming past, proceed along two or more development lines in a single project, and coordinate the work with other people in the same project.

**Vertical reuse** A type of reuse in which assets are reused within an application domain.

**Vertical (a.k.a. internal) traceability** It refers to the ability to trace dependent artifacts within a model.

**Wrapping** It is a black-box modernization technique—surrounds the LIS with a software layer that hides the unwanted complexity of the existing data, individual programs, application systems and interfaces with the new interfaces. Essentially, this gives old components new operation or a "modern and improved" look.

**White-box reuse** Reuse of a component by applying major changes to the component.

**White-box testing** It is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality that is black-box testing.