Software Engineering GLOSSARY

Abstract Data Types (ADT)

A type whose internal form is hidden behind a set of access functions. Objects of the type are created and inspected only by calls to the access functions. This allows the implementation of the type to be changed without requiring any changes outside the module in which it is defined. Abstract data types are central to object-oriented programming where every class is an ADT. A classic example of an ADT is a stack data type for which functions might be provided to create an empty stack, to push values onto a stack and to pop values from a stack. [*foldoc*]

--

A class of data structures described by means of a set of operations rather than by physical representation, such as a class in object-oriented programming. [*sting*]

Abstract System

An abstract system is the system defined by a functional design. It is not a physical system to be found in the real world, but a conceptual system behaving as specified in the functional design. It is our understanding of the abstract system that enables us to reason about the specified behaviour and verify that it will satisfy the functional requirements. [*proteus*]

Abstraction

Generalisation, ignoring or hiding details. Examples are abstract data types (the representation details are hidden), abstract syntax (the details of the concrete syntax are ignored}, abstract interpretation (details are ignored to analyse specific properties). [*foldoc*]

--

Parameterisation, making something a function of something else. Examples are lambda abstractions (making a term into a function of some variable), higher-order functions (parameters are functions), bracket abstraction (making a term into a function of a variable). [*foldoc*]

Algorithm Animation

Algorithm animation is the process of abstracting the data, operations and semantics of computer programs and then creating animated graphical views of those abstractions. [*koschke95*]

American National Standards Institute (ANSI)

The United States government body responsible for approving US standards in many areas, including computers and communications. ANSI is a member of ISO. ANSI sells ANSI and ISO (international) standards. [*foldoc*]

--

An organization that reviews and approves product standards in the United States. In the electronics industry, its work enables designers and manufacturers to create and support products that are compatible with other hardware platforms in the industry. Examples are PHIGS and GKS. See also International Organization for Standardization (ISO). [*sun*]

--

American National Standards Institute, responsible for approving U.S. standards in many areas, including computers and communications. ANSI is a member of ISO [*sting*]

Application family

An application family is a generic representation of application systems. One purpose is to allow several abstract systems, possibly defined using different design languages, to be

composed in one application. A second purpose is to factor out the support systems which are generic and evolve independently from applications. The concept support heterogeneous applications, not easily covered by a single system description expressed in one of the design languages. [*proteus*]

Application Generator

An application generator takes as input a specification of the required product. This specification can be a 4GL program. The product of the generator is usually only modified by rerunning the generator with a changed specification. Building an application generator for some problem domains is difficult and requires much foresight. [*cleaveland88*]

Application System

The application part of a system instance implementation. An application system defines the application (the behaviour) a customer wants to buy in terms of implementation code. It is normally a partial implementation lacking the necessary support to execute. An application system is used to produce the concrete systems that actually execute and is expressed using some high-level programming language. [*proteus*]

Applications Programmer Interface (API)

The interface (calling conventions) by which an application program accesses operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code. An API can also provide an interface between a high level language and lower level utilities and services which were written without consideration for the calling conventions supported by compiled languages. In this case, the API's main task may be the translation of parameter lists from one format to another and the interpretation of call-by-value and call-by-reference arguments in one or both directions. [*foldoc*]

--

(1) The interface to a library of language-specific subroutines (such as a graphics library) that implement higher level functions.
(2) A set of calling conventions defining how a service is invoked through a software package. [*sun*]

Architectural Design

Large systems are divided into smaller subsystems and modules which import from each other. The subsystems and modules and their use relationship is called the architectural design. [*koschke95*]

--

The specific components of a computer system and the way they interact with one another. [*sun*]

--

Design, the way components fit together. The term is used particularly of processors, both individual and in general. It may also be used of any complex system, e.g. software architecture, network architecture. [*foldoc*]

Attachment

An encapsulated data object inside a document.[*sun*]

Automated Reverse Architectural Design

The architectural design of a system and its components can be recovered automatically using certain tools. Some approaches are able to subsume modules into an automatically derived subsystem structure. [*koschke95*]

CASE Based Reasoning

A technique for problem solving which looks for previous examples which are similar to the current problem. This is useful where heuristic knowledge is not available. Some key research areas are efficient indexing, how to define "similarity" between cases and how to use temporal information. [*foldoc*]

Class

A class defines an software object's interface and implementation. It specifies the object's internal representation and defines the operations that the object can be instructed to perform. [*james93*]

--

The prototype for an object in an object-oriented language; analogous to a derived type in a procedural language. A class may also be considered to be a set of objects which share a common structure and behaviour. The structure of a class is determined by the variables which represent the state of an object of that class and the behaviour is given by a set of methods associated with the class. [*foldoc*]

--

A grouping of data having similar characteristics. A class definition defines instance and class variables and methods, as well as specifying the interfaces the class implements and the immediate superclass of the class. [*sun*]

Client

A computer system or process that requests a service of another computer system or process (a server). For example, a workstation requesting the contents of a file from a file server is a client of the file server. [*foldoc*]

--

In the client-server model for communications, the client is a process that remotely accesses resources of a compute server, such as compute power and large memory capacity. [*sun*]

Client-Server

A common form of distributed system in which software is split between server tasks and client tasks. A client sends requests to a server, according to some protocol, asking for information or action, and the server responds. There may be either one centralised server or several distributed ones. This model allows clients and servers to be placed independently on nodes in a network, possibly on different hardware and operating systems appropriate to their function, e.g. fast server/cheap client. [*foldoc*]

Client-Server, Three-Tier

Application partitioning, or three-tier architectures are expected to be the next generation of client-server systems. A three-tier system adds a third component (the application server) in between the current client and server. The application server maintains some data and behaviour which reflects business rules. With a two-tier system if business rules change, then all client workstations have to be upgraded. In contrast a three-tier's application server provides a central location and transparent updating of the rules with respect to the clients. [*Hettler96*]

Cohesion

A measure of the level of functional integration within a module. High cohesion implies well defined modules serving one dedicated purpose, low cohesion implies ambiguity. See also coupling. [*guerre96*]

Common Object Request Broker Architecture (CORBA)

An Object Management Group specification which provides the standard interface definition between OMG-compliant objects. [*foldoc*]

Complexity

A code measure, which is a combination of code, data, data flow, structure and control flow metrics. [*koschke95*]

Component Integration Laboratories (CIL)

An effort to create a common framework for interoperability between application programs on desktop platforms, formed by Apple Computer Inc., IBM, Novell, Oracle, Taligent, WordPerfect and Xerox. [*foldoc*]

Component Software (Component Object Model, COM)

Compound documents and component software define object-based models that facilitate interactions between independent programs. These approaches aim to simplify the design and implementation of applications, and simplify human-computer interaction. Component software addresses the general problem of designing systems from application elements that were constructed independently by different vendors using different languages, tools, and computing platforms. The goal is to have end-users and developers enjoying the same level of plug-and-play application interoperability that are available to hardware manufacturers. Compound documents are one example of component interoperability. [*adler95*]

--

COM supports a client-server model where clients are service users, OLE objects are data, and OLE servers are the applications which implement OLE objects. [*adler95*]

Compound Document (Compound Object Model, COM)

Compound documents and component software define object-based models that facilitate interactions between independent programs. These approaches aim to simplify the design and implementation of applications, and simplify human-computer interaction. A compound document is a container for sharing heterogeneous data, which includes mechanisms which manage containment, association with an application, presentation of data/applications, user interaction with data/applications, provision of interfaces for data exchange, and more notably linking and embedding. Data can be incorporated into a document by a pointer (link) to the data contained elsewhere in the document, or in another document. Linking reduces storage requirements, and facilitates automatic transparent updates. Embedding is where the data is physically located within a compound document. [*adler95*]

--

Compound documents are containers and composed of (possibly nested) parts, organised by end-users. A part contains one or more types, e.g. sound, text, image. End users access and manipulate parts via component handlers. [*adler95*]

Computer Aided Software Engineering (CASE)

A technique for using computers to help with one or more phases of the software life-cycle, including the systematic analysis, design, implementation and maintenance of software. Adopting the CASE approach to building and maintaining systems involves software tools and training for the developers who will use them. [*foldoc*]

Computer Supported Cooperative/Collaborative Work (CSCW)

Software tools and technology to support groups of people working together on a project, often at different sites. [*foldoc*]

Conceptual Abstraction

Semi-formal, human-oriented and domain-specific abstractions play a critical role both in reverse and forward engineering, and therefore also in reengineering. Such conceptual abstractions are fundamental to the reengineering process whether it is a totally manual or partially automated process. [*biggerstaff90*]

--

A representation of the domain model (problem, program and application) knowledge in the form of informal and semi-formal information. [*koschke95*]

Concrete System

A concrete system implements the behaviour of one or more abstract systems. A typical concrete system will consist of hardware and executable code, and is what users actually use. Each concrete system will be distinct and will operate in a particular application context. [*proteus*]

Configuration Item

Hardware or software, or an aggregate of both, which is designated by the project configuration manager (or contracting agency) for configuration management. [*foldoc*]

Configuration Management

A discipline applying technical and administrative controls to:

- Identification and documentation of physical and functional characteristics of configuration items.
- Any changes to characteristics of those configuration items.
- Recording and reporting of change processing and implementation of the system. [*foldoc*]

--

The process of identifying, defining, recording and reporting the configuration items in a system and the change requests. Controlling the releases and change of the items throughout the life-cycle. [*sting*]

Configuration Programming

An approach that advocates the use of a separate configuration language to specify the coarse-grain structure of programs. Configuration programming is particularly attractive for concurrent, parallel and distributed systems that have inherently complex program structures. [*foldoc*]

Coupling

The degree to which components depend on one another. There are two types of coupling, "tight" and "loose". Loose coupling is desirable for good software engineering but tight coupling may be necessary for maximum performance. Coupling is increased when the data exchanged between components becomes larger or more complex. [*foldoc*]

Database

One or more large structured sets of persistent data, usually associated with software to update and query the data. A simple database might be a single file containing many records, each of which contains the same set of fields where each field is a certain fixed width. [*foldoc*]

--

Loosely, any aggregation of data; a file consisting of a number of records (or tables), each of which is constructed of fields (columns) of a particular type, together with a collection of operations that facilitate searching, sorting, recombining, or similar activities. [*sun*]

Database Management System

A suite of programs which typically manage large structured sets of persistent data, offering ad hoc query facilities to many users. A database management system is a complex set of software programs that controls the organisation, storage and retrieval of data (fields, records and files) in a database. It also controls the security and integrity of the database. The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data. When a DBMS is used, information systems can be changed much more easily as the organisation's information requirements change. New categories of data can be added to the database without disruption to the existing system. [*foldoc*]

--

A software system facilitating the creation and maintenance of a data base and the execution of programs using the data base. [*sun*]

Database, Object Oriented

A system offering DBMS facilities in an object-oriented programming environment. Data is stored as objects and can be interpreted only using the methods specified by its class. The relationship between similar objects is preserved (inheritance) as are references between objects. Queries can be faster because joins are often not needed (as in a relational database). This is because an object can be retrieved directly without a search, by following its object id. The same programming language can be used for both data definition and data manipulation. The full power of the database programming language's type system can be used to model data structures and the relationship between the different data items. [*foldoc*]

Database, Relational

A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations, and integrity constraints. In such a database, the data and relations between them are organised in tables. A table is a collection of records and each record in a table contains the same fields. Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up. Records in different tables may be linked if they have the same value in one particular field in each table. INGRES, Oracle and Microsoft Access are well-known examples. [*foldoc*]

Data Centered Program Understanding

Instead of focusing on the control structure of a program (such as call graphs, control-flow graphs and paths) data centered program understanding focuses on data and data-relationships. [*koschke95*]

Data Flow Analysis

A process to discover the dependencies between different data items manipulated by a program. The order of execution in a data driven language is determined solely by the data dependencies. [*foldoc*]

Data Flow Diagram (DFD)

A graphical notation used to describe how data flows between processes in a system. An important tool of most structured analysis techniques. [*foldoc*]

Data Name Rationalisation (DNR)

A special case of data re-engineering. DNR tools enforce uniform naming conventions across all software systems. [*jlccrm*]

Design

The phase of software development following analysis, and concerned with how the problem is to be solved. [foldoc]

Design Pattern

A design pattern systematically names, motivates and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution and it's consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customised and implemented to solve the problem in a particular context. [*james93*]

Design Recovery

A subset of reverse engineering in which 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. Design recovery recreated design abstractions from a combination of code, existing design documentation (if available), personal experience, and general knowledge about problem and application domains. [*foldoc*] and [*chikofsky90*]

--

Design recovery recreates design abstractions from a combination of code, existing design documentation, personal experience and general knowledge about problem and application domains. The recovered design abstractions must include conventional software engineering representation such as formal specifications, module breakdowns, data abstractions, data-flows, and program description languages. In addition they must include informal linguistic knowledge about problem domains, application idioms, and the world in general. [*biggerstaff89*]

--

The design recovery process consists of three steps:

(a) supporting program understanding for maintenance (what are the modules? What are the key data items? What are the software engineering artifacts? What are the other informal design abstractions?);
(b) supporting population of reuse and recovery libraries (The design abstractions of the former step are generalised and integrated into the reuse library and the recovery knowledge base);
(c) Applying the results of design recovery (The abstract design components stored in the domain model now become the starting point for discovering candidate concrete realisations of themselves in a new system's code). [*koschke95*]

Distributed Computing Environment (DCE)

An architecture consisting of standard programming interfaces, conventions and server functionalities (e.g. naming, distributed file system, remote procedure call) for distributing applications transparently across networks of heterogeneous computers. DCE is promoted and controlled by the Open Software Foundation. [*foldoc*]

Domain

A functional area covered by a family of systems. [*proteus*]

Domain Analysis

The set of activities aiming at identifying, collecting, organising, analysing and representing the relevant information in a domain, based on the study of existing systems and their development history, knowledge captured from domain experts, underlying theory, and emerging technologies within the domain. Domain analysis aims at producing domain models and analysing commonalities and variants among a family of products. [*proteus*]

Domain Architecture / Domain Architectural Model / Family Design

An architecture applicable to a family of applications belonging to the domain. Sometimes called the "generic architecture". [*proteus*]

Domain Engineering

An encompassing process which includes domain analysis and the subsequent methods and tools that address the problem of development through the application of domain analysis products (e.g., domain implementation). [*proteus*]

Domain Model

Domain models are the result of domain analysis. A domain model is a definition of domain abstractions (objects, relationships, functions, events, etc.) It consists off a concise and classified representation of the commonalities and variabilities of the problems in the domain and of

their solutions. It is a representation of a family. Domain models include domain requirements models (the problem) and domain architecture (the solution). [*proteus*]

--

The base of knowledge about the problem required for an informal reasoning approach to knowledge-based concept assignment. It is assumed that the domain model (problem, program and application) knowledge can be usefully represented as patterns of informal and semi-formal information, which are called conceptual abstractions. [*koschke95*]

Encapsulation

The result of hiding a representation and implementation in an object. The representation is not visible and cannot be accessed directly from outside the object. Operations are the only way to access and modify an object's state. [*james93*]

--

The ability to provide users with a well-defined interface to a set of functions in a way which hides their internal workings. In object-oriented programming, the technique of keeping together data structures and the methods (procedures) which act on them. [*foldoc*]

Entity

(1) International Organization for Standardization's open systems interconnection (OSI) terminology for a layer protocol machine. An entity within a layer performs the functions of the layer within a single computer system, accessing the layer entity below and providing services to the layer entity above at local service access points.

(2) In object-oriented programming, an entity is part of the definition of a class (group) of objects. In this instance, an entity might be an attribute of the class (as feathers are an attribute of birds), or it might be a variable or an argument in a routine associated with the class.

(3) In database design, an entity is an object of interest about which data can be collected. In a retail database application, customers, products, and suppliers might be entities. An entry can subsume a number of attributes: Product attributes might be color, size, and price; customer attributes might include name, address, and credit rating. [*sun*]

Family

A set of systems sharing some commonalities (equivalent to product line). [*proteus*]

Forward Engineering

The set of engineering activities, using the output of software reengineering, that consume the products and artifacts derived from legacy software and new requirements to produce a new target system. [*jlccrm*]

--

The traditional process of moving from high-level abstractions and logical implementation-independent designs to the physical implementation of a system. [*foldoc*] and [*chikofsky90*]

Framework

In object-oriented systems, a set of classes that embodies an abstract design for solutions to a number of related problems. [*foldoc*]

--

A set of cooperating classes that make up a reusable design for a specific class of software. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customises the framework to a particular application by sub-classing the composing instances of framework classes. [*james93*]

Groupware

See Computer-support collaborative work.

Heterogeneous Network

A network composed of systems of more than one architecture. Contrast with homogeneous network. [*sun*]

Homogeneous Network

A network composed of systems of only one architecture. Contrast with heterogeneous network. [*sun*]

Hypermedia

An extension of hypertext to include graphics, sound, video and other kinds of data. [*foldoc*]

Hypertext

A term coined by Ted Nelson around 1965 for a collection of documents (or nodes) containing cross-references or links which, with the aid of an interactive browser program, allow the reader to move easily from one document to another. [*foldoc*]

Implementation Design Description

An implementation design description is a complete description of all the information needed to produce a concrete system from one or more functional designs. It describes which components of a generic system family must be used, and which tools must be invoked with which parameters. The implementation design description acts as meta-description referring to other descriptions. [*proteus*]

Implementation Model

The implementation model consists of the code files and the used work structure. It includes the application software description as well as the support software description. While the design model is a more abstract view, the implementation model contains the full information necessary to build the system. [*proteus*]

Informal Reasoning

An approach to knowledge-based concept assignment. Informal reasoning is based on human oriented concepts, but takes knowledge like natural language comments, or grouping in account too. The needed base of knowledge about the problem for the informal reasoning is called the domain model. It is assumed that the domain model (problem, program and application) knowledge can be usefully represented as patterns of informal and semi-formal information, which are called conceptual abstractions. [*koschke95*]

Information Base

The main repository of information about the software. It can be created by decomposing any number of views of a system. [*koschke95*]

Interaction Diagram

A diagram that shows the flow of interaction between objects. [*james93*]

Interface

A boundary across which two systems communicate. An interface might be a hardware connector used to link to other devices, or it might be a convention used to allow communication between two software systems. Often there is some intermediate component between the two systems which connects their interfaces together. [*foldoc*]

--

(1) The point at which independent systems or diverse groups interact. The devices, rules, or conventions by which one component of a system communicates with another. Also, the point of communication between a person and a computer.

(2) The part of a program that defines constants, variables, and data structures, rather than procedures.

(3) The equipment that accepts electrical signals from one part of a computer system and renders them into a form that can be used by another part.
(4) Hardware or software that links the computer to a device.
(5) To convert signals from one form to another and pass them between two pieces of equipment. [*sun*]

International Organization for Standardization (ISO)

A voluntary, non-treaty organisation founded in 1946, responsible for creating international standards in many areas, including computers and communications. ISO produced the seven layer model for network architecture (Open Systems Interconnection). Its members are the national standards organisations of 89 countries, including the American National Standards Institute. [*foldoc*]

Internationalisation

The process of altering a program so that it is portable across several native languages. This portability may support both different character sets, such as the 8-bit ISO 8859/1 (ISO Latin 1) character set and the 7-bit ASCII character set, and different languages for documentation, help screens, and so on. [*sun*]

Language, 3rd Generation (3GL)

A language designed to be easier for a human to understand, including things like named variables. A fragment might be `let c = c + 2 * d`. FORTRAN, ALGOL and COBOL are early examples of this sort of language. Most "modern" languages (BASIC, C, C++) are third generation. Most 3GLs support structured programming. [*foldoc*]

Language, 4th Generation (4GL)

An application specific language. The term was invented to refer to non-procedural high level languages built around database systems. The first three generations were developed fairly quickly, but it was still frustrating, slow, and error prone to program computers, leading to the first "programming crisis", in which the amount of work that might be assigned to programmers greatly exceeded the amount of programmer time available to do it. Meanwhile, a lot of experience was gathered in certain areas, and it became clear that certain applications could be generalised by adding limited programming languages to them. Thus were born report-generator languages, which were fed a description of the data format and the report to generate and turned that into a COBOL (or other language) program which actually contained the commands to read and process the data and place the results on the page. Some other successful 4th-generation languages are: database query languages, e.g. SQL; Focus, Metafont, PostScript, RPG-II, S, IDL-PV/WAVE, Gauss, Mathematica and data-stream languages such as AVS, APE, Iris Explorer. [*foldoc*]

Language, Interface Definition (IDL)

To accomplish interoperability across languages and tools, an object model specifies standards for defining application interfaces in terms of a language independent - an interface definition language. Interface definitions are typically stored in a repository which clients can query at run-time. [*renaissance96*]

Language, Markup

Languages for annotation of source code to simply improve the source code's appearance with the means of bold-faced key words, slanted comments, etc. See also reformatting. [*koschke95*]
--
In computerised document preparation, a method of adding information to the text indicating the logical components of a document, or instructions for layout of the text on the page or other information which can be interpreted by some automatic system. [*foldoc*]

Language, Module Interconnection (MIL)

A module interconnection language is a language that is separate from and complementary to a program implementation language. MILs are concerned with the overall architecture of software systems. They deal with the composition of large systems out of modules, the interfaces between these modules & their specification, and the versioning of the resulting architecture over time. The purpose of MILs is to describe a system so that it can be constructed, unequivocally identified, and identically reproduced. A MIL is both a notation for design, documentation & communication, and a means of enforcing system architecture. [*renaissance96*]

Language, Object Oriented

A language for object oriented programming. The basic concept in this approach is that of an object which is a data structure (abstract data type) encapsulated with a set of routines, called methods which operate on the data. Operations on the data can only be performed via these methods, which are common to all objects which are instances of a particular class (see inheritance). Thus the interface to objects is well defined, and allows the code implementing the methods to be changed so long as the interface remains the same. Each class is a separate module and has a position in a class hierarchy. Methods or code in one class can be passed down the hierarchy to a subclass or inherited from a superclass. Procedure calls are described in term of message passing. A message names a method and may optionally include other arguments. When a message is sent to an object, the method is looked up in the object's class to find out how to perform that operation on the given object. If the method is not defined for the object's class, it is looked for in its superclass and so on up the class hierarchy until it is found or there is no higher superclass. Procedure calls always return a result object, which may be an error, as in the case where no superclass defines the requested method. [*foldoc*]

Language, Program Description/Design (PDL)

Any of a large class of formal and profoundly useless pseudo-languages in which management forces one to design programs. Too often, management expects PDL descriptions to be maintained in parallel with the code, imposing massive overhead of little or no benefit. [*foldoc*]

Language, Structured

A programming language where the program may be broken down into blocks or procedures which can be written without detailed knowledge of the inner workings of other blocks, thus allowing a top-down design approach. [*foldoc*]

Language, Specification and Description (SDL)

A language standardised by the ITU-T well suited to functional design of reactive systems comprising concurrent processes with state-transition behaviour. [*proteus*]

Language, Structured Query (SQL)

A language which provides a user interface to relational database management systems, developed by IBM in the 1970s. SQL is the de facto standard, as well as being an ISO and ANSI standard. It is often embedded in other programming languages. SQL provides provided basic language constructs for defining and manipulating tables of data, language extensions for referential integrity and generalised integrity constraints, facilities for schema manipulation and data administration, and capabilities for data definition and data manipulation. Development is currently underway to enhance SQL into a computationally complete language for the definition and management of persistent, complex objects. This includes: generalisation and specialisation hierarchies, multiple inheritance, user defined data types, triggers and assertions, support for knowledge based systems, recursive query expressions, and additional data administration tools. It also includes the specification of abstract data types (ADTs), object identifiers, methods, inheritance, polymorphism, encapsulation, and all of the other facilities normally associated with object data management. [*foldoc*]

Language, Visual Programming (VPL)

Any programming language that allows the user to specify a program in a two-(or more)-dimensionsional way. Conventional textual languages are not considered two-dimensional since the compiler or interpreter processes them as one-dimensional streams of characters. A VPL allows programming with visual expressions - spatial arrangements of textual and graphical symbols. VPLs may be further classified, according to the type and extent of visual expression used, into icon-based languages, form-based languages and diagram languages. Visual programming environments provide graphical or iconic elements which can be manipulated by the user in an interactive way according to some specific spatial grammar for program construction. A visually transformed language is a non-visual language with a superimposed visual representation. Naturally visual languages have an inherent visual expression for which there is no obvious textual equivalent. [*foldoc*]

Legacy System

A typical computer legacy system may be 10-25 years old, have been developed using archaic methods, have experienced several personnel changes, one for which current maintenance is very expensive, and one for which integration with current or modern technology or software systems is difficult or impossible. Legacy systems require reengineering to put them in a form where they may better suit modern requirements and may evolve more efficiently. [*sei95*]

--

A computer system or application program which continues to be used because of the prohibitive cost of replacing or redesigning it and despite its poor competitiveness and compatibility with modern equivalents. The implication is that the system is large, monolithic and difficult to modify. If the legacy software only runs on antiquated hardware the cost of maintaining this may eventually outweigh the cost of replacing both the software and hardware unless some form of emulation or backward compatibility allows the software to run on new hardware. [*foldoc*]

Maintenance / Maintainability

An important part of the software life-cycle. Maintenance is expensive in manpower and resources, and software engineering aims to reduce its cost. [*foldoc*]

--

Maintenance activities include:

- Perfective maintenance - Changes which improve the system in some way without changing its functionality;
- Adaptive maintenance - Maintenance which is required because of changes in the environment of a program;
- Corrective maintenance - The correction of previously undiscovered system errors. [*sommerville96*]

--

Maintainability is defined as the effort to perform maintenance tasks, the impact domain of the maintenance actions, and the error rate caused by those actions. [*sneed90*]

Metric

A measure of software quality which indicate the complexity, understandability, testability, description and intricacy of code. [*foldoc*]

Metric, Maintenance

Metrics that try to give a quantifying answer on how good a certain program is to maintain. [*koschke95*]

Middleware

Software that mediates between an application program and a network. It manages the interaction between disparate applications across the heterogeneous computing platforms. The Object Request Broker (ORB), software that manages communication between objects, is an example of a middleware program. [*foldoc*]

--

A middleware service is a general purpose service that sits between platforms and applications. It is defined by the APIs and protocols it supports. Middleware is generally not application-specific, not platform specific, distributed, and supports standard interfaces and protocols. [*bernstein96*]

Object

A run-time entity that packages both data and the procedures that operate on that data. [*james93*]

--

In object-oriented programming, a unique instance of a data structure defined according to the template provided by its class. Each object has its own values for the variables belonging to its class and can respond to the messages (methods) defined by its class. [*foldoc*]

Object Linking and Embedding (OLE)

A distributed object system and protocol from Microsoft, also used on the Acorn Archimedes. OLE allows an editor to "farm out" part of a document to another editor and them re-import it. For example, a desk-top publishing system might send some text to a word processor or a picture to a bitmap editor using OLE. [*foldoc*]

Object Management Group (OMG)

A consortium aimed at setting standards in object-oriented programming. The Common Object Request Broker Architecture (CORBA) specifies what it takes to be OMG-compliant. [*foldoc*]

Object Modelling Technique (OMT)

An object-oriented analysis and design method used for domain modelling. [*proteus*]

Object Oriented Design/Analysis (OOD/OOA)

A design method in which a system is modelled as a collection of cooperating objects and individual objects are treated as instances of a class within a class hierarchy. Four stages can be identified: identify the classes and objects, identify their semantics, identify their relationships and specify class and object interfaces and implementation. object-oriented design is one of the stages of object-oriented programming. [*foldoc*]

Object Oriented Programming (OOP)

The basic concept in this approach is that of an object which is a data structure (abstract data type) encapsulated with a set of routines, called methods which operate on the data. Operations on the data can only be performed via these methods, which are common to all objects which are instances of a particular class (see inheritance). Thus the interface to objects is well defined, and allows the code implementing the methods to be changed so long as the interface remains the same. Each class is a separate module and has a position in a class hierarchy. Methods or code in one class can be passed down the hierarchy to a subclass or inherited from a superclass. Procedure calls are described in term of message passing. A message names a method and may optionally include other arguments. When a message is sent to an object, the method is looked up in the object's class to find out how to perform that operation on the given object. If the method is not defined for the object's class, it is looked for in its superclass and so on up the class hierarchy until it is found or there is no higher superclass. Procedure calls always return a result object, which may be an error, as in the case where no superclass defines the requested method. [*foldoc*]

Object Request Broker (ORB)

ORBs are fundamental to CORBA. In a distributed environment they provide a common platform for client objects to request data and services from server objects, and for server objects to pass their responses back to clients. ORBs hide interoperability details from objects (i.e., programming language and operating system used, local or remote, etc.) [*halfhill96*]

OLE Custom Controls (OCX)

An Object Linking and Embedding (OLE) custom control allowing infinite extension of the Microsoft Access control set. OCX is similar in purpose to VBX used in Visual Basic. [*foldoc*]

OMT Model

A model built with the Object Modelling Technique. An OMT model may have three views (object, dynamic and functional). It consists of a set of diagrams and associated information necessary for characterising the domain (e.g., data dictionary, modelling rationale, costs, etc.) [*proteus*]

OpenDoc

A compound document architecture from CIL based on CORBA. It aims to enable embedding of features from different application programs into a single working document. [*foldoc*]

Open Scripting Architecture (OSA)

An automation technology that works with OpenDoc and lets parts of a component document be manipulated programmatically and coordinated to work together. [*adler95*]

--

A CIL approach to the coexistence of multiple scripting systems. [*foldoc*]

Open Software Foundation (OSF)

A foundation created by nine computer vendors, (Apollo, DEC, Hewlett-Packard, IBM, Bull, Nixdorf, Philips, Siemens and Hitachi) to promote open computing. It is planned that common operating systems and interfaces, based on developments of Unix and the X Window System will be forthcoming for a wide range of different hardware architectures. OSF announced the release of the industry's first open operating system - OSF/1 on 23 October 1990. [*foldoc*]

Plug-and-Play

Hardware or software that, after being installed (plugged in), can immediately be used (played with), as opposed to hardware or software which first requires configuration. [*foldoc*]

Portable Common Tool Environment (PCTE)

A European Computer Manufacturers Association standard framework for software tools developed in the Esprit programme. It is based on an entity-relationship Object Management System and defines the way in which tools access this. [*foldoc*]

Potpourri Module

A potpourri module is a module that provides more than one service to a program. This form of module violates the idea of a module being considered a *responsibility assignment*. The existence of this form of module increases considerably the effort that a programmer has to expend on a maintenance operation, and increases the likelihood of an error being introduced to a program as a result of maintenance work. [*calliss90*]

Process Model

A model for a set of partially ordered steps required to reach a goal. [*proteus*]

Product

A *product* is a broader than a *system*. It incorporates all components that the producer uses, and all the items delivered to customers. Documentation and confidential source code, for instance, is part of the *product* but not part of the *system*. [*proteus*]

Product Family

A product family is a collection of all the components used to produce concrete systems and any other items delivered to customers. It is generic in the sense that many distinct product instances can be produced from one generic product. [*proteus*]

Product Instance

A product instance is what a customer buys. A product instance consists of a copy of executable implementation code and any other items sold with it, typically documentation. [*proteus*]

Program Analysis

Program analysis tools are designed to aid the task of understanding existing source code by providing a large amount of detailed information about the program. Analysis tools help focus on the structure and attributes of the system. The relevant information can be extracted from a program by either analysing the program text (static analysis), or by observing it's behaviour (dynamic analysis). [*sei95*]

Program Heuristics

A general rule of programming concept which captures the conventions in programming and govern the composition of the program plans into programs. [*koschke95*]

Program Plan Recognition

The use of program plans to identify similar code fragments. Existing source code is often reused within a system via "cut and paste" text operations. Detection of cloned code fragments must be done using program heuristics since the decision whether two arbitrary programs perform the same function is undecidable. [*sei95*]

Program Plans/Concepts

An approach to knowledge-based concept assignment. They are schemes in which certain programming problems are usually solved. They are specified in terms of control and data flow and other structural information. A parsing approach (or sometime only pattern matching) will match plans stored in a plan base with the source code to assign concepts. [*koschke95*]

--

Generic program fragments that represent stereotypic action sequences in programming. [*soloway84*]

--

Program plans are abstract representations of source code fragments. Comparison methods are used to help recognise instances of programming plans in a subject system, the focus being to identify similar code fragments with pattern matching at the programming language semantic level. [*sei95*]

Program Slicing

A program slice is a fragment of a program in which some statements are omitted that are not necessary to understand a certain property of the program. For example if someone is interested in how the value for a certain returned value of a function is arrived at then only code that has a bearing, direct or indirect, on that value is relevant. [*koschke95*]

Program Understanding

A related term to reverse engineering. Program understanding implies always that understanding begins with the source code while reverse engineering can start at a binary and executable form of the system or at high level descriptions of the design. The science of program understanding includes the cognitive science of human mental processes in program understanding. Program understanding can be achieved in an ad hoc manner and no external representation has to arise. While reverse engineering is the systematic approach to develop an external representation of the subject system, program understanding is comparable with design recovery because both of them start at source code level. [*chikofsky90*]

--

Program comprehension is the process of acquiring knowledge about a computer program. Increased knowledge enables such activities as bug correction, enhancement, reuse and documentation. While efforts are underway to automate the understanding process, such significant amounts of knowledge and analytical power are required that today program understanding is largely a manual task. [*rugaber96*]

Program Visualisation

Program visualisation is defined as a mapping from programs to graphical representations. [*roman92*]

Protocol

A set of formal rules describing how to transmit data, especially across a network. Low level protocols define the electrical and physical standards to be observed, bit- and byte-ordering and the transmission and error detection and correction of the bit stream. High level protocols deal with the data formatting, including the syntax of messages, the terminal to computer dialogue, character sets, sequencing of messages etc. [*foldoc*]

Prototyping

The creation of a model and the simulation of all aspects of a product. CASE tools support different degrees of prototyping. Some offer the end-user the ability to review all aspects of the user interface and the structure of documentation and reports before code is generated. [*foldoc*]

Quality

The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs. Not to be mistaken for "degree of excellence" or "fitness for use" which meet only part of the definition. [*foldoc*]

Quality, Low

Low quality is defined as delivered software which does not work at all, or repeatedly fails in operation. A project where users report more than 0.5 bugs or defects per function point per calendar year are of low quality. [*jones94*]

Rapid Application Development (RAD)

A loose term for any software life-cycle designed to give faster development and better results and to take maximum advantage of recent advances in development software. RAD is associated with a wide range of approaches to software development: from hacking away in a GUI builder with little in the way of analysis and design to complete methodologies expanding on an information engineering framework. Some of the current RAD techniques are: CASE tools, iterative life-cycles, prototyping, workshops, SWAT teams, timebox development, and reuse of applications, templates and code. [*foldoc*]

Recode

Changes to implementation characteristics. Language translation and control-flow restructuring are source code level changes. Other possible changes include conforming to coding standards, improving source code readability, renaming programming items, etc. [*chikofsky90*]

Redesign

Changes to design characteristics. Possible changes include restructuring a design architecture, altering a system's data model as incorporated in data structures or in a database, improvements to an algorithm, etc. [*chikofsky90*]

Redocumentation

The process of analysing the system to produce new support documentation in various forms including user manuals and reformatting the system's source code listings. [*jlccrm*]

--

The creation or revision of a semantically equivalent representation within the same relative abstraction level. The resulting forms of representation are usually considered alternate views (for example, dataflow, data structures, and control flow) intended for a human audience. Redocumentation is the simplest and oldest form of reverse engineering, and can be considered to be an unintrusive, weak form of restructuring. [*foldoc*] and [*chikofsky90*]

Reengineering

The examination and modification of a system to reconstitute it in a new form and the subsequent re-implementation of the new form. [*foldoc*]

--

The examination and alteration of an existing subject system to reconstitute it in a new form. This process encompasses a combination of sub-processes such as reverse engineering, re-structuring, re-documentation, forward engineering and re-targeting. [*jlccrm*]

--

Also known as renovation and reclamation. It is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form. [*chikofsky90*]

--

Reengineering is the systematic transformation of an existing system into a new form to realise quality improvements in operation, system capability, functionality, performance or evolvability at a lower cost, schedule or risk to the customer. Reengineering also emphases the importance of a greater return on investment than could be achieved through a new development effort. [*sei95*]

--

Any activity that improves one's understanding of software and/or improves the software itself, "software" being taken to include source code, design records and other sources of documentation. The task is partitioned into two:

- Program understanding; activities such as browsing, measurement, and reverse engineering
- Software evolution; activities such as redocumentation, restructuring and remodularisation.

[*arnold93*].

--

The process of modifying the internal mechanisms of a system or program or the data structures of a system of program without changing its functionality. [*guide89*]

--

The examination and alteration of a subject system to reconstitute it in a new form and subsequent implementation of that form. [*chikofsky90*]

Reengineering, Business Process (BPR)

The fundamental rethinking and radical redesign of business procedures to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service and speed. An example BPR tool would be process modeling, to facilitate the experimentation with "what if?" scenarios on business processes. [*jlccrm*]

--

Business process reengineering aims to tackle the broad issue of how a business operates rather than merely the software which supports the business process. BPR involves the notion of discontinuous thinking - reorganising and breaking away from the outdated rules and assumptions that underlie operations. [*hammer90*]

Reengineering, Data

Tools that perform all the re-engineering functions associated with source code (reverse engineering, forward engineering, translation, redocumentation, restructuring / normalisation and retargeting), but act upon data files. [*jlccrm*]

Reformatting

The functional equivalent transformation of source code which changes only the structure to improve readability. Examples are pretty-printers and tools that replace GOTO loops with equivalent loops. [*koschke95*]

Remodularisation

Changing a module's structure in light of coupling analysis, in order to redefine the boundaries between modules and function of modules. [*koschke95*]

Remote Procedure Call (RPC)

A protocol which allows a program running on one host to cause code to be executed on another host without the programmer needing to explicitly code for this. RPC is an easy and popular paradigm for implementing the client-server model of distributed computing. An RPC is implemented by sending request message to a remote system (the server) to execute a designated procedure, using arguments supplied, and a result message returned to the caller (the client). There are many variations and subtleties in various implementations, resulting in a variety of different (incompatible) RPC protocols. [*foldoc*]

Representation Problem

Building models to understand software systems is an important part of reverse engineering. Formal and explicit model building is important because it focuses attention on modeling as an aid to understanding and results in artifacts that may be useful to others. The representation used to build models has great influence over the success and value of the result. Choosing the proper representation during reverse engineering is the representation problem. [*clayton93*]

Requirements Model

The requirements model describes the functionality and behaviour of a system. The requirements model is a functional design description of the system to be built. [*proteus*]

Respecify

Changes to requirements characteristics. This type of change can refer to changing only the form of existing requirements. For example, taking informal requirements expressed in English and generating a formal specification expressed in a formal language such as Z. This type of change can also refer to changing system requirements. Requirements changes include the addition of new requirements of the deletion or alteration of existing requirements. [*chikofsky90*]

Restructuring

The engineering process of transforming the system from one representational form to another at the same relative level of abstraction, whilst preserving the subject system's external functional behaviour. [*foldoc*] and [*jlccrm*]

--

The transformation from one representational form to another at the same relative abstraction level, while preserving the subject system's external behaviour (functionality and semantics). [*chikofsky90*]

Retargeting

The engineering process of transforming and hosting or porting the existing system in a new configuration. This could be a new hardware platform, new operating system or a new CASE platform. [*jlccrm*]

Reuse

Using code developed for one application program in another application. Traditionally achieved using program libraries. Object-oriented programming offers reusability of code via its techniques of inheritance and genericity. Class libraries with intelligent browsers and application generators are under development to help in this process. Polymorphic functional languages also support reusability while retaining the benefits of strong typing. [*foldoc*]

Reuse, Black Box

A style of reuse based on object composition. Composed objects reveal no internal details to each other and are thus analogous to 'black-boxes'. [*james93*]

Reuse Engineering

The modification of software to make it more reusable, usually rebuilding parts to be put into a library. [*olsen95*]

Reuse, White Box

A style of reuse based on class inheritance. A subclass reuses the interface and implementation of its parent class, but it may have access. [*james93*]

Reverse Engineering

The process of analysing an existing system to identify its components and their interrelationships and create representations of the system in another form or at a higher level of abstraction. Reverse engineering is usually undertaken in order to redesign the system for better maintainability or to produce a copy of a system without access to the design from which it was originally produced. [*foldoc*]

--

The engineering process of understanding, analysing and abstracting the system to a new form at a higher abstraction level. [*jlccrm*]

--

The process of analysing a subject system to

(1) Identify the systems components and their interrelationships, and

(2) Create representations of the system in another form or a higher level of abstraction. [*chikofsky90*]

Reverse Specification

A kind of reverse engineering where a specification is abstracted from the source code or design description. Specification in this context means an abstract description of what the software does. In forward engineering the specification tells us what the software has to do, but this information is not included in the source code. Only in rare cases can it be recovered from comments in the source code and from the people involved in the original forward engineering process. [*chikofsky90*]

--

Reverse specification is intended to extract a description of what the examined system does. The description is made in terms of the application domain. On one hand this process must be bottom-up since the only reliable description of the behaviour of software is its source code. On the other hand reverse specification must be top-down too. Trivially, knowledge of the application domain is necessary to describe in terms of the application domain. [*koschke95*]

Risk

Risk is defined as the possibility of loss or injury. Risk exposure is defined by the relationship

$$RE = P(UO) * L(UO)$$

Where *RE* is the risk exposure, *P(UO)* is the probability of an unsatisfactory outcome, and *P(LO)* is the loss to the parties affected by if the outcome is unsatifactory. Examples of unsatisfactory outcome include schedule slips, budget overruns, wrong functionality,

compromised non-functional requirements, user-interface shortfalls and poor quality. [*boehm91*]

Risk Analysis (and Prioritisation)

The assessment of the loss probability and loss magnitude for each identified risk item. Prioritisation involves producing a ranked and relative ordering of the risk items identified and analysed. [*boehm91*]

Risk Identification

The production of a list of project specific risk items that are likely to compromise a project's success. An example risk identification is the generation of checklists of likely risk factors. [*boehm91*]

Risk Management

Risk management is divided into the following tasks:

- Risk assessment
    - Risk identification
    - Risk analysis and prioritisation
- Risk control
    - Risk management planning
    - Risk resolution and monitoring [*boehm91*]

Risk Management Planning

Plans which lay out the activities necessary to bring the risk items under control. Activities include prototyping, simulation, modelling, tuning, etc. All management plans should be integrated to reuse parts of each where possible, and to be factored into the overall schedule. [*boehm91*]

Risk Resolution (and Monitoring)

Production of a situation in which the risk items are eliminated or resolved. Risk monitoring involves tracking the project's progress towards resolving it's risk items and taking corrective action where appropriate. [*boehm91*]

Semantics

The meaning of a string in some language, as opposed to syntax which describes how symbols may be combined independent of their meaning. The semantics of a programming language is a function from programs to answers. A program is a closed term and, in practical languages, an answer is a member of the syntactic category of values. The two main kinds are denotational semantics and operational semantics. [*foldoc*]

Server

A program which provides some service to other (client) programs. The connection between client and server is normally by means of message passing, often over a network, and uses some protocol to encode the client's requests and the server's responses. The server may run continuously (as a daemon), waiting for requests to arrive or it may be invoked by some higher level daemon which controls a number of specific servers. [*foldoc*]

Software Engineering

A systematic approach to the analysis, design, implementation and maintenance of software. It often involves the use of CASE tools. There are various models of the software life-cycle, and many methodologies for the different phases. [*foldoc*]

Software Evolution

The accommodation of perfective, corrective and adaptive maintenance, which may involve some reengineering activity. [*sommerville96*]

--

Those activities which are geared towards improving the software itself, rather than increasing one's understanding of the same. Examples include restructuring, redocumenting and data reengineering. All are meant to evolve the subject system from its current form to one that better meets the new requirements. [*sei95*]

Software Life-Cycle

The software life-cycle consists of: requirements analysis, design, construction, testing (validation) and maintenance. The development process tends to run iteratively through these phases rather than linearly; several models (spiral, waterfall etc.) have been proposed to describe this process. Other processes associated with a software product are: quality assurance, marketing, sales and support. [*foldoc*]

Software Methodology

The study of how to navigate through each phase of the software process model (determining data, control, or uses hierarchies, partitioning functions, and allocating requirements) and how to represent phase products (structure charts, stimulus-response threads, and state transition diagrams). [*foldoc*]

Software Psychology

Software psychology attempts to discover and describe human limitations in interacting with computers. These limitations can place restrictions on and form requirements for computing systems intended for human interaction. [*rugaber94*]

Structured System Analysis and Design Method (SSADM)

A software engineering method and toolset required by some UK government agencies. [*foldoc*]

Subsystem

An independent group of classes that collaborate to fulfill a set of responsibilities. [*james93*]

Subsystem Composition

The process of constructing composite software components out of building blocks such as variables, procedures, modules and subsystems. [*mueller90*]

Support Families

This is the collection of families used to compose support systems. The generic support is likely to be layer structured. Therefore several general support families are likely, e.g. operating system, I/O system, communication system, user interface, database management system. It is part of the implementation design to determine which support families to use, their composition and how instances are to be configured. [*proteus*]

Support System

The support system contains the support needed to actually execute an application system, e.g., the operating system, the user-interface library. It will normally consist of several layers of support where the lower layers provide services to the higher. [*proteus*]

Synchronised Refinement

Synchronised refinement is a systematic approach to detecting design decisions in source code and relating the detected decisions to the functionality of the system. [*koschke95*]

System Building

System building is the process of transforming descriptions using tools to create some less abstract description. This may involve converting designs to source programs to object code.

However, the building process may include other transformations such as the construction of system documentation from document fragments. [*proteus*]

System Modelling

System modelling is a technique to express, visualise, analyse and transform the architecture of a system. Here a system may consist of software components, hardware components, or both and the connections between these components. A system model is then a skeletal model of the system. It is intended to assist in developing and maintaining large systems with emphasis on the construction phase. [*renaissance96*]

System Object Model (SOM)

SOM is IBM's CORBA-compliant object request broker for a single address space architecture. A similar distributed system object model framework exists to allow objects to communicate across address spaces and networks. [*adler95*]

--

An implementation of CORBA by IBM. [*foldoc*]

Task Interaction Graph

Task interaction graphs divide a program into maximal sequential regions connected by edges representing task interactions. Task interaction graphs can be used to generate concurrency graph representations, and both facilitate analysis of concurrent programs. [*long89*]

Time-To-Market

The time between project start-up and delivery of the final concrete system. This duration is affected by organisation factors and non-software elements of the system, [*card95*]

Transaction

A unit of interaction with a DBMS or similar system. It must be treated in a coherent and reliable way independent of other transactions. [*foldoc*]

--

A transaction is the unit recovery, consistency and concurrency in a client-server system. [*orfali95*]

Transaction Processing (TP)

The exchange of transactions in a client-server system to achieve the same ends as would be performed by the equivalent single complex application. [*orfali95*]

Transaction Processing Monitor (TPM)

For mission-critical applications it is vital to manage the programs which operate on the data. TP monitors achieve this by breaking complex applications down into transactions. TPMs were invented for applications which serve thousands of clients. A TP monitor can manage transaction resources on a single server or across multiple servers. [*orfali95*]

Transformation/Translation, Program/Software

Transformation of source code from one language to another or from one version of a language to another version of the same language. For example, converting from COBOL-74 to COBOL-85. [*jlccrm*]

--

The systematic development of efficient programs from high-level specifications by meaning-preserving program manipulations. [*foldoc*]

--

Examples of reengineering transformations are the changes from unstructured code to structured code, updating design documents, or correcting specifications. It is assumed that the transformation improves the subject system according to some measurable criterion. [*sei95*]

Uniform Resource Locator (URL)

A draft standard for specifying an object on the Internet, such as a file or newsgroup. URLs are used extensively on the World-Wide Web. They are used in HTML documents to specify the target of a hyperlink. [*foldoc*]

User Interface (UI)

The aspects of a computer system or program which can be seen (or heard or otherwise perceived) by the human user, and the commands and mechanisms the user uses to control its operation and input data. A graphical user interface emphasises the use of pictures for output and a pointing device such as a mouse for input and control whereas a command line interface requires the user to type textual commands and input at a keyboard and produces a single stream of text as output. [*foldoc*]

User Interface, Graphical (GUI)

The use of pictures rather than just words to represent the input and output of a program. A program with a GUI runs under some windowing system (e.g. The X Window System, Microsoft Windows, Acorn RISC OS, NEXTSTEP). The program displays certain icons, buttons, dialogue boxes etc. in its windows on the screen and the user controls it mainly by moving a pointer on the screen (typically controlled by a mouse) and selecting certain objects by pressing buttons on the mouse while the pointer is pointing at them. [*foldoc*]

--

The graphical user interface, or GUI, provides the user with a method of interacting with the computer and its special applications, usually via a mouse or other selection device. The GUI usually includes such things as windows, an intuitive method of manipulating directories and files, and icons. [*sun*]

Version

(1) A version is a concrete instance of an object. There could exist multiple versions of one object.
(2) A concrete configuration with concrete versions of the different objects belonging to this configuration. Also known as a release. [*proteus*]

View

A view is a software representation or a document about software. Example views are requirements and specification documents, hierarchy charts, flowcharts, petri nets, test data, etc. Each view is classified according to a particular view type:

- Non-procedural - e.g. requirements documents
- Pseudo-procedural- e.g. software architecture documents
- Procedural - e.g. source code, data definition
- Analysis views which may accompany any other view. [*koschke95*]

Views, Code

Representations of the source code which cover the same information as the code (or parts of it) but in a manner that accelerates the comprehension process. Examples are program slices, call-graphs, data-flow, definition-use graphs, or control dependencies. [*koschke95*]

Visual Basic

An event-driven visual programming system for Microsoft Windows, in which fragments of BASIC code are invoked when the user performs certain operations on graphical objects on-screen. Widely used for in-house applications development by users and for prototyping. [*foldoc*]

Visual Programming Environment

Software which allows the use of visual expressions (such as graphics, drawings, animation or icons) in the process of programming. These visual expressions may be used as graphical interfaces for textual programming languages. They may be used to form the syntax of new visual programming languages leading to new paradigms such as programming by demonstration or they may be used in graphical presentations of the behaviour or structure of a program. [*foldoc*]

Visual Modelling

A class of RAD tool which allow for the construction and execution of models during design. [*snell95*]

Workflow

A workflow is composed of multiple tasks / steps / activities, of which there are two types:
(1) Simple, representing indivisible activities, and
(2) Compound, representing those which can be decomposed into sub-activities. An entire workflow can be regarded as a large compound task.

--

The set of relationships between all the activities in a project, from start to finish. Activities are related by different types of trigger relation. Activities may be triggered by external events or by other activities. Also the movement of documents around an organisation for purposes including sign-off, evaluation, performing activities in a process and co-writing. [*foldoc*]

Workflow Management (WFM)

Workflow management is a technology that supports the reengineering and automation of business and information processes. It involves:
(a) Defining workflows, i.e., those aspects of process that are relevant to control and coordinate the execution of its tasks, and
(b) Providing for fast (re)design and (re)implementation of the processes as business/information needs change. [*renaissance96*]

Workflow Management Coalition (WFMC)

A standards body formed in 1993 by a group of companies, intended to address the lack of standards in WFMSs. [*renaissance96*]

Workflow Management System (WFMS)

A workflow management system provides procedural automation of a business process by management of the sequence of work activities and the invocation of appropriate human/IT resources associated with the various activity steps. Workflow products are typically client-server software products in which the work is performed within defined time-scales. [*renaissance96*]

World Wide Web (WWW, W3, Web)

An Internet client-server hypertext distributed information retrieval system which originated from the CERN High-Energy Physics laboratories in Geneva, Switzerland. On the WWW everything (documents, menus, indices) is represented to the user as a hypertext object in HTML format. Hypertext links refer to other documents by their URLs. These can refer to local or remote resources accessible via FTP, Gopher, Telnet or news, as well as those available via the HTTP protocol used to transfer hypertext documents. The client program (known as a browser), e.g. Mosaic, Netscape, runs on the user's computer and provides two basic navigation operations: to follow a link or to send a query to a server. A variety of client and server software is freely available. [*foldoc*]

X/Open

An international consortium of vendors whose purpose is to define the X/Open Common Applications Environment to provide applications portability. They also produced the X/open Portability Guide (XPG). [*foldoc*]

REFERENCES

[*adler95*] R. M. Adler. Emerging standards for component software. *Computer*, March 68-77 1995

[*arnold93*] R. S. Arnold. Software reengineering. IEEE Computer Society Press, 1993.

[*bernstein96*] P. A. Bernstein. Middleware: a model for distributed system services. Communications of the ACM 39(2), 86-98, 1996.

[*biggerstaff89*] Ted J. Biggerstaff. Design recovery for maintenance and reuse, *IEEE Computer*, 22(7):36-49, July 1989

[*biggerstaff90*] Ted J Biggerstaff. Human-oriented conceptual abstractions in the reengineering of software. In *Proceedings of the 12th International Conference on Software Engineering*, page 120, March 1990

[*biggerstaff93*] Ted J Biggerstaff. The concept assignment problem in program understanding. In *Proceedings of the 15th International Conference on Software Engineering*, pages 482-498. IEEE Computer Society Press, April 1993

[*boehm91*] B. W. Boehm. Software risk management: principles and practices. IEEE software vol 8, no 1, pp 32-42, 1991.

[*calliss90*] Frank W. Calliss & Barry J. Cornelius. Potpourri module detection. In *Proceedings of the International Conference on Software Maintenance 1990*, pages 46-51, IEEE Computer Society Press, 1990

[*card95*] D. N. Card. The RAD Fad: Is timing really everything? IEEE Software 12(5), 19-22, 1995.

[*chikofsky90*] Eliot J Chikofsky & James H Cross. Reverse engineering and design recovery: A taxonomy. IEEE Software, 7(1):13-17, January 1990.

[*clayton93*] Richard Clayton & Spencer Rugaber. The representation problem in reverse engineering. In *Proceedings of the First Working Conference on Reverse Engineering*, Baltimore, Maryland, May 1993

[*cleaveland88*] J. C. Cleaveland. Building application generators. IEEE Software 5(4), 25-33. 1998.

[*foldoc*] The Free On-line Dictionary of Computing London UK, `http://wombat.doc.ic.ac.uk/cgi-bin/foldoc`

[*guerre96*] Jonathan Guerre. A primer on module design http://solo.hull.ac.uk/users/guerrej/primer.htm

[*guide89*] Application reengineering. Technical report GPP-208, Guide International Corp., 1989.

[*halfhill96*] T. R. Halfhill & S. Salamone. Components everywhere. Byte January 97-104, 1996.

[*hammer90*] Michael Hammer. Reengineering work: don't automate, obliterate! Havard Business Review Jul-Aug 1990.

[*hettler96*] M. Hettler. New leaders of the client-server migration. Byte, June 124-131, 1996.

[*james93*] James Martin. *Principles of object-oriented analysis and design*. Prentice-Hall, 1993

[*jlccrm*] Reengineering Definitions of the Joint Logistic Commanders Computer Resources Management group `http://www.stsc.hill.af.mil/~red/defin.html`

[*jones94*] C. Jones. Assessment and control of software risks. Yourdon Press, 1994.

[*koschke95*] A Bibliography on Reengineering `http://www.informatik.uni-stuttgart.de/ifi/ps/reengineering/reengineering.html`

[*long89*] D. L. Long & L. A. Clarke. Task interaction graphs for concurrency analysis. In *Proceedings of the 11th International Conference on Software Engineering*, pages 44-52, May 1989.

[*mueller90*] Hausi A. Mueller & James S. Uhl. Composing subsystem structures using (k,2)-partite graphs. In *Proceedings of the International Conference on Software Maintenance*

1990, pages 12-19. IEEE Computer Society Press, 1990.

[*olsen95*] N. C. Olsen. Survival of the fastest: improving service velocity. IEEE Software 12(5), 28-38, 1995.

[*orfali95*] R. Orfali, D. Harkey and J. Edwards. Intergalactic client-server computing. Byte, April, 108-122, 1995.

[*proteus*] Glossary to the PROTEUS project. `http://www.comp.lancs.ac.uk/computing/research/cseg/projects/PROTEUS/`

[*renaissance96*] Technology briefing report on Process Support.
Technology briefing report on System Modelling.

[*roman92*] G. C. Roman & K. C. Cox. Program visualisation: The art of mapping programs to pictures. In *Proceedings of the 14th International Conference on Software Engineering*, pages 412-420, May 1992

[*rugaber94*] Spencer Rugaber & Victoria Tisdale. Software psychology requirements for software maintenance activities. Technical report, *Software Engineering Centre Georgia Institute of Technology*, Atlanta GA, 1994.

[*rugaber96*] Spencer Rugaber. Program understanding. *Encyclopedia of Computer Science and Technology*, *1996*. To appear.

[*sei95*] Perspectives on legacy system reengineering. Reengineering centre, Software Engineering Institute, Carnegie Mellon University, 1995.

[*sneed90*] Harry M. Sneed & Agnes Kaposi. A study on the effect of reengineering on maintainability. In *Proceedings of the International Conference on Software Maintenance 1990*, pages 91-99. IEEE, Computer Society Press 1990.

[*snell95*] M. Snell. A new visual attitude, LAN Times Online, May 1995 `http://www.lantimes.com/lantimes/archive/505a047a.html`

[*soloway84*] Elliot Soloway & Kate Ehrlich. Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, SE-10(5):595-609, September 1984.

[*sommerville96*] Ian Sommerville, *Software Engineering*. Addison-Wesley, 1996.

[*sting*] Software Technology Interest Group On-line Glossary `http://dxsting.cern.ch/sting/glossary-intro.html`

[*sun*] Sun Microsystems On-line Glossary `http://www.sun.com/glossary/glossary.html`