Glossary

Abstract class: a class that has no instances; a superclass that acts only as a generalized template for its instantiated subclasses.

Abstract operation: an operation that is not implemented in the class in which it appears (usually an abstract superclass), but will be implemented in a subclass.

Abstraction: a simplified representation that contains only those features that are relevant for a particular task; the act of separating out the general or reusable parts of an element of a system from its particular implementation.

Activation: the execution of an operation, represented in interaction sequence diagrams as a long thin rectangle.

Activity: an activity is some behaviour that may persist for the duration of a state.

Activity diagram: a variation of a statechart diagram that focuses on a flow of activity driven by internal processing within an object rather than by events that are external to it. In an activity diagram most (or all) states are action states (also called *activities*), each of which represents the execution of an operation.

Actor: an actor is an external entity of any form that interacts with the system. Actors may be physical devices, humans or information systems.

Adornment: an element attached to another model element, for example a stereotype icon or a constraint.

Aggregation: a whole–part association between two or more objects, where one represents the whole and the others parts of that whole.

Algorithm: a description of the internal logic of a process or decision in terms of a sequence of smaller steps.

Analysis class stereotype: one of three specialised kinds of class (boundary, control and entity classes (*q.v.*)) that feature in analysis class diagrams. The separation of concerns that these represent forms the basis of the architecture recommended for most models developed following USDP guidelines (*cf.* stereotype).

Antipattern: documents unsuccessful attempts at providing solutions to certain recurring problems but includes reworked solutions that are effective.

Association: a logical connection, usually between different classes although in some circumstances a class can have an association with itself. An association describes possible links between objects, and may correspond either to logical relationships in the application domain or to message paths in software.

Association class: a class that is modelled in order to provide a location for attributes or operations that properly belong to an association between other classes.

Association instance: another name for a link (*q.v.*).

Attribute: an element of the data structure that, together with operations, defines a class. Describes some property of instances of the class.

Boundary class: a stereotyped class that provides an interface to users or other systems.

Business rule: see *enterprise rule*.

Capta: data that has been selected for processing due to its relevance to a particular purpose.

Cardinality: the number of elements in a set; contrast with *multiplicity* (*q.v.*).

Class: a descriptor for a collection of objects that are logically similar in terms of their behaviour and the structure of their data.

Class diagram: a UML diagram that shows classes with their attributes and operations, together with the associations between classes.

Class Responsibility Collaboration (CRC): CRC cards provide a technique for exploring the possible ways of allocating responsibilities to classes and the collaborations that are necessary to fulfil the responsibilities

Class-scope: a class-scope attribute occurs only once and is attached to the class (not to any individual object. A class-scope operation is accessed through the class (i.e. prefixed with the class name) not through an object. Model elements that are of class scope are underlined in class diagrams.

Cohesion: a measure of the degree to which an element of a model contributes to a single purpose.

Collaboration: the structure and links between a group of instances that participate in a behaviour. The behaviour can be that of an operation or a use case (or any other behavioural classifier in UML).

Collaboration diagram: a collaboration diagram shows an interaction between objects and the context of the interaction in terms of the links between the objects.

Collection class: provides collection-specific behaviour to maintain a collection. Used when designing associations with a many multiplicity to hold collections of object identifiers.

Common Object Request Broker Architecture (CORBA): a mechanism to support the construction of systems in which objects, possibly written in different languages, reside on different machines and are able to interact by message passing.

Component: an executable software module with a well-defined interface and identity.

Component diagram: a diagram that shows the organization of and dependencies among components. One of two UML implementation diagrams (*q.v.*).

Composition: a strong form of aggregation with a lifetime dependency between each part and the whole. No part can belong to more than one composition at a time, and if the composite whole is deleted its parts are deleted with it.

Concrete class: a class that may have instances.

Concurrent states: if an object may be in two or more states at the same time, then these states are concurrent states.

Constructor operation: an operation that creates new instances of a class.

Context (in OCL—q.v.): the domain within which an OCL expression is valid, for example, a class.

Context (of a pattern): the circumstances in which a particular problem occurs.

Contract: a black box description of a service (of a class or sub-system) that specifies the results of the service and the conditions under which it will be provided.

Control class: a stereotyped class that controls the interaction between boundary classes and entity classes.

Coupling: relates to the degree of interconnectedness between design components and is reflected by the number of links and the degree of interaction an object has with other objects.

Critical Path Analysis (CPA): a diagrammatic technique for analysing the dependencies between project tasks and determining those tasks that must be completed

on time if the project itself is to be completed on time.

Data: raw facts, not yet identified as relevant to any particular purpose.

Dependency: a relationship between two model elements, such that a change in one element may require a change in the dependent element.

Deployment diagram: A diagram that shows the run-time configuration of processing nodes (*q.v.*) and the components, processes and objects that are located on them. One of two UML implementation diagrams (*q.v.*).

Design constraint: a constraint that limits the design options that may be used. Common design constraints include cost and data storage requirements.

Destructor operation: an operation that destroys instances of a class.

Domain model: an analysis class model that is independent of any particular use cases or applications, and that typically contains only entity objects. A domain model may serve as a basis for the analysis and design of components that can be reused in more than one software system.

Encapsulation: hiding internal details of a sub-system (typically a class) from the view of other sub-systems, so that each can be maintained or modified without affecting the operation of other parts of the system.

Enterprise (or business) rule: a statement that expresses business constraints on the multiplicity of an association; for example, an order is placed by exactly one customer.

Entity class: a stereotyped class that represents objects in the business domain model.

Event: an occurrence that is of significance to the information system.

Exception: a mechanism for handling errors in object-oriented languages.

Extend relationship: a relationship between use cases where one use case extends or adds new actions to another. Written as a stereotype: «extend».

Extreme Programming (XP): an approach to systems development that focuses on producing the simplest coding solution for application requirements. It uses pair programming, where program code is always written by two developers working at the same workstation.

Forces (of a pattern): the particular issues that must be addressed in resolving a problem.

Functional requirement: a requirement that specifies a part of the functionality required by the user.

Generalization: the abstraction of common features among elements (for example, classes) by the creation of a hierarchy of more general elements (for example, superclasses) that encapsulate the common features.

Guard condition: a Boolean expression associated with a transition that is evaluated at the time the event fires. The transition only takes place if the condition is true. A guard condition is a function that may involve parameters of the triggering event and also attributes and links of the object that owns the statechart.

Implementation diagram: a generic term for the two UML diagrams used in modelling the implementation of a system.

Include relationship: a relationship between use cases where one use case includes the actions described in another use case. Written as a stereotype: «include».

Incremental development: involves some initial analysis to scope the problem and identify the major requirements. The requirements are then reviewed and those that deliver most benefit to the client become the focus of the first increment of development and delivery. The installation of the first increment provides valuable feedback to the development team and informs the development of the second increment and so on.

Information: facts that have been selected as relevant to a purpose and then organized or processed in such a way that they have meaning for that purpose.

Inheritance: the mechanism by which object-oriented programming languages implement a relationship of generalization and specialization between classes. A subclass automatically acquires features of its superclasses.

Instance: a single object, usually called an instance in the context of its membership of a particular class or type (also object instance).

Instance diagram: a UML diagram similar in form to a class diagram, but that contains object instances instead of classes, links instead of asociations and may show attribute values (also known as an object diagram).

Instance value (of an attribute): the value of an attribute that is taken by a particular object at a particular time.

Integrity constraint: a constraint that has to be enforced to ensure that the information system holds data that is mutually consistent and is manipulated correctly. Referential integrity ensures that an object identifier in one object actually refers to an object that exists. Dependency constraints ensure that attribute dependencies, values are maintained consistently, where the value of one attribute is calculated from other attributes, are maintained consistently. Domain integrity ensures that attributes only hold permissible values.

Interaction: defines the message passing between objects within the context of a collaboration to achieve a particular behaviour.

Interaction diagram: an umbrella term for sequence diagrams and collaboration diagrams.

Interface: that part of the boundary between two interacting systems across which they communicate; the set of all signatures for the public operations of a class or package.

Interface class: a system interacts with its actors via its interface or boundary classes

Invariant: an aspect of a UML model expressed as a formal statement that must always remain true. For example, the value of a derived attribute `totalCost` may need always to be equal to the total of all `cost` attribute values.

Knowledge: a complex structure of information, usually one that allows its possessor to decide how to behave in particular situations.

Legacy system: any computerized information system, that has probably been used for some time, that was built with older technologies (maybe using different development approaches at different times) and that, most importantly, continues to deliver benefit to the organization.

Life cycle (of a project): the phases through which a development project passes from the inception of the idea to completion of the product and its eventual decommissioning.

Lifeline: a lifeline is a vertical dashed line that represents the existence of an object on an interaction sequence diagrams. An object symbol containing the object's name is placed at the top of a lifeline.

Link: a connection between objects; an instance of an association.

Message: a request to an object that it provide some specified service, either an action that it can carry out or some information that it can provide.

Message passing: a metaphor for the way that objects interact in an object-oriented system by sending each other messages that request services, or request or supply information. Since objects interact only through the messages they exchange, their internal details can remain hidden from each other.

Method: the implementation of an operation.

Methodology: comprises an approach to software development (e.g. object-orientation), a series of techniques and notations (e.g. the Unified Modelling Language—UML) that support the approach, a life cycle model (e.g. spiral incremental) to structure the development process, and a unifying set of procedures and philosophy.

Modular construction: an approach that aims to build systems that are easy to maintain, modify or extend. Modular construction relies on modules that are essentially decoupled sub-systems, with their internal details encapsulated.

Multiplicity: a constraint that specifies the range of permitted *cardinalities* (*q.v.*), for example in an association role or in a composite class. An association may have a multiplicity of 1..5; a particular instance of that association may have a cardinality of 3.

Node: A physical computational resource used by a system at run-time, typically having processing capability and memory.

Non-functional requirement: a requirement that relates to system features such as performance, maintainability and portability.

Normalization: a technique that groups attributes based upon functional dependencies according to several rules to produce normalized data structures that are largely redundancy free.

Object: a single thing or concept, either in a model of an application domain or in a software system, that can be represented as an encapsulation of state, behaviour and identity; a member of a class that defines a set of similar objects.

Object constraint language (OCL): a formal language that supplements the graphical notations of UML. OCL is generally used to give precise definitions for operation logic, or for properties such as invariants (q.v.).

Object diagram: see *instance diagram*.

Operation: an aspect of the behaviour that defines a class; an element of the services that are provided by a class; a specification of an element of system functionality that will be implemented as a method of an object.

Operation (in OCL—q.v.): usually an arithmetic, set or type operator, such as '+', 'size' or 'isEmpty', that is applied to the property (q.v.) in an OCL expression.

Operation signature: determined by the operation's name, the number and type of its parameters and the type of the return value if any. Polymorphically redefined operations have the same signature.

Package: a mechanism for grouping UML elements, usually classes, into groups. Packages can be nested within other packages.

Pattern: a pattern is an abstract solution to a commonly occurring problem in a given context.

Polymorphism: the ability of different methods to implement the same operation, and thus to respond to the same message in different ways that are appropriate to their class. For example, objects of different subclasses in an inheritance hierarchy may respond differently to the same message, yet with a common meaning to their responses.

Post-condition: part of an operation specification; those conditions that must be true before the operation can execute.

Pre-condition: part of an operation specification; those conditions that must be true after the operation has executed—in other words the valid results of the operation.

Primary operation: an operation to create or destroy an instance of a class, or to get or set the value of an attribute.

Property: a feature or characteristic of a UML element, usually one for which there is

no specific UML notation.

Property (in OCL—q.v.): that specific element of the context (q.v.) to which an OCL expression applies, for example an attribute of a class.

Prototype: a prototype is a system or partially complete system that is built quickly to explore some aspect of the system requirements. It is not intended as the final working system.

Query operation: an operation that returns information but causes no change of state within a model or a software system.

Realize relationship: a relationship between two classes where one class offers the interface of the other but does not necessarily have the same structure of the other. Commonly used to show that a class supports an interface. Written as a stereotype: «realize».

Refactoring: restructuring and simplifying programme code so that duplication is removed and flexibility is enhanced.

Relation: a group of related data items organized in columns and rows, also known as a table.

Repository: the part of a CASE tool environment that handles the storage of models, including diagrams, specifications and definitions.

Responsibility: a high level description of behaviour a class exhibits. It reflects the knowledge or information that is available to that class, either stored within its own attributes or requested via collaboration with other classes, and also the services that it can offer to other objects.

Sequence diagram: or interaction sequence diagram, shows an interaction between objects arranged in a time sequence. Sequence diagrams can be drawn at different levels of detail and also to meet different purposes at several stages in the development life cycle.

Service: a useful function (or set of functionality) that is carried out by an object (or a sub-system) when requested to do so by another object.

Signal: an asynchronous communication between objects that may have parameters.

Software architecture: describes the sub-systems and components of a software system and the relationships between the components.

Specialization: the other face of generalization; an element (for example, a class) is said to be specialized when it has a set of characteristics that uniquely distinguish it from other elements. Distinguishes subclasses from their superclass.

Stakeholders: anyone who is affected by the information system. Stakeholders not only include users and development team members, but also resource managers and the quality assurance team, for example.

State: the state of an object is determined by values of some of its attributes and the presence or absence of certain links with other objects. It reflects a particular condition for the object and normally persists for a period of time until a transition to another state is triggered by an event. Instantaneous 'flow-through' states are allowed in UML 1.4.

Stereotype: a specialized UML modelling element. The stereotype name is contained within matched guillemets «...». For example, an interface package is a stereotype of a package.

Stimulus: an interaction between two objects that conveys information with an expectation of some action.

Subclass: a specialized class that acquires general features from its ancestor superclasses in a generalization hierarchy, but that also adds one or more specialized characteristics of its own.

Sub-system: a part of a system that can be regarded as a system in its own right.

Superclass: a generalized class that is an abstraction of the common characteristics of its subclasses in a generalization hierarchy.

Synchronizing operation: an operation that ensures that those attribute values which are dependent upon each other (e.g. may be calculated from each other) have consistent values.

System: an abstraction of a complex interacting set of elements, for which it is possible to identify a boundary, an environment, inputs and outputs, a control mechanism and some process or transformation that the system achieves.

Table: group of related data items organized in columns and rows. Used to store data in relational databases.

Task: a specific activity or step in a project.

Technique: a method for carrying out a project task.

Transaction: an elementary exchange, say of an item of capta (*q.v.*) or of a unit of value.

Transition: the movement from one state or activity to another, triggered by an event. A transition may start and end at the same state.

Type: a stereotype of class that is distinct from an implementation class; a type is defined by its attributes and operations but, unlike an implementation class, may not contain any methods. Classes that represent the concepts of the application domain are in fact types. An object may change its type dynamically during system execution, and may thus appear at different times to belong to different classes.

Usability requirement: user requirement that describes criteria by which the ease of use of the system can be judged.

Use case: describes, from a user's perspective, a behaviourally related set of transactions that are normally performed together to produce some value for the user. Use cases can be represented graphically in a use case diagram, each use case being described in the data dictionary. Use cases may be modelled at varying degrees of abstraction, essential use cases, the most abstract, are technologically and implementation independent whereas real use cases describe how the use case actually operates in a particular environment.

Use case realization: a set of model elements that show the internal behaviour of the software that corresponds to the use case—usually a collaboration.

User requirement: something that users require a software system to do (functional requirement); alternatively, a standard for the performance of a system (non-functional requirement).

User story: in Extreme Programming requirements are captured as user stories. A user story is very similar to a use case.

Visibility: UML modelling elements (e.g. attributes or operations) may be designated with different levels of accessibility or visibility. Public visibility means that the element is directly accessible by any class; private visibility means that the element may only be used by the class that it belongs to; protected visibility means that the element may only be used by either the class that includes it or a subclass of that class; and package visibility means that an element is visible to objects in the package.

Wrapper: or object wrapper, used to integrate object-oriented and non-object-oriented systems by encapsulating the non-object-oriented system with an object-oriented style of interface.