**Glossary**

**accuracy** – a quantitative measure of the magnitude of error

**add user** – initiated by an Elite, a new employee is entered into the system and assigned a username, password, and employee classification.

**analysis** – an activity during which developers ensure that the system requirements are *correct*, *complete*, *consistent*, and unambiguous. Analysis produces the *analysis model*.

**analysis model** – a model of the system that aims to be *correct*, *complete*, *consistent*, and unambiguous. The analysis model consists of the *functional model*, *the analysis object model*, and the *dynamic model*.

**analysis object model** – the object produced during analysis. The analysis object model describes the application domain concepts that the system manipulates and the user-visible interfaces of the system.

**approve timesheet** – timesheet is reviewed and approved by an Elite

**associations** – a relationship between two or more classes denoting the possible links between instances of the classes. An association has a name and can have multiplicity and role information attached to each of its ends.

**attributes** – a named property of a class defining a range of the values an object can contain.

**availability** – the degree to which a system or component is operational and accessible when required for use.

**Chief Financial Officer (CFO)** – an employee who ultimately receives all of the timesheets for analysis and payment of the employees.

**completeness** – the property of a model describing whether or not all relevant phenomena are modeled. A model is incomplete if one or more relevant phenomena do not have a corresponding concept in the model.

**consistency** – the property of a model that indicates whether or not it contradicts itself. A model is inconsistent if it provides several incompatible views of the system.

**constraint** – a rule attached to a UML element restricting its semantics.

Constraints can be depicted by a note containing natural language text or an expression in a formal language (e.g., OCL).

**contracts** – a set of *constraints* on a class or component allowing the caller and callee to share the same assumptions about the class or component.

**correctness** – the property of a model that indicates if it accurately represents the system that the client needs and that the developers intend to build.

**current system** – current state of the proposed system.

**deny timesheet** – timesheet is reviewed and denied by an Elite. A notification is then sent to the employee stating that he/she needs to meet with his/her supervisor to fix the discrepancies.

**dependability** – the property of a computer system that describes how reliably it performs.

**design goals** – a quality that the system should optimize. Design goals are often inferred from *non-functional requirements* and are used to guide design decisions.

**dynamic model** – describes the components of the system that have interesting behavior.

**Elite** – a salaried employee (usually a manager of Grunt/+) who approves or denies the timesheets of the employees in his department.

**entry condition** – a condition that needs to be satisfied before a use case is initiated.

**exit condition** – a condition that is satisfied after the completion of a use case.

**export** – all employee hourly info will be exported into a Microsoft Excel file. This action is only available to the CFO.

**functional model** – describes the functionality of the system from the user’s point of view.

**functional requirements** – an area of functionality the system must support. The functional requirements describe the interactions between the actors and the system independent of the realization of the system.

**global hours** -- hours administered and approved by a CFO that are automatically given to the employee. These hours override any entered by an employee during that given time slow.

**Grunt** – an hourly employee who simply clocks in and out in real time when at work. Sick and vacation days do not apply to this type of employee. His hours are logged by clicking a button when he logs into the timesheet application at work.

**Grunt+** - an hourly employee who must submit a timesheet at the end of every week. This type of employee logs his hours (time in, lunch out, lunch in, time out) for every day of the week by entering it on the timesheet page after he logs into the system. Sick and vacation time apply for this type of employee.

**hog hours** – available to Grunt+, the employee will enter his time in, time out, lunch in, and lunch out into a table. This table will then be submitted as the employee’s timesheet.

**implementation** – an activity during which developers translate the object model into code.

**interface** – a point of interaction between components

**legal** – Constraint concerned with licensing, regulation, and certification issues.

**modify user** – initiated by and Elite, an employees information can be adjusted in the event of change. Elite can change all info except for an employees password, which can only be changed by the CFO.

**non-functional requirements** – a user-visible *constraint*on the system. Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functionality of the system.

**object design model** – a detailed model representing the application and solution objects that make up the system. The object design model includes detailed class specifications, *contracts*, *types*, *signatures*, and *visibilities* for public operations.

**object model** – describes the structure of a system in terms of *objects*, *attributes, associations*, and *operations*. During *requirements* and *analysis*, the object model starts as the *analysis object model* and describes the application concepts relevant to the system. During system design, the object model is refined into the *system design object model* and includes descriptions of the subsystem interfaces. During object design, the object model is refined into the *object design model* and includes detailed descriptions of solution objects.

**objects** – an instance of a class. An object has an identity and stores attribute values.

**operations** – an atomic piece of behavior that is provided by a class. A calling object triggers the execution of an operation by sending a message to the object on which the operation should be executed.

**packaging** – constraint on the actual delivery of the system.

**performance** – any quantifiable attribute of a system, such as *response time*, *throughput*, *availability*, and *accuracy*.

**Prophet** – a system administrator of the system. A Prophet has almost universal access rights and can change many properties of the system.

**reliability** – ability of a system or component to perform its required functions under stated conditions for a specified period of time. Reliability requirements include, for example, an acceptable mean time to failure, the ability to detect specified faults, or to withstand specified security attacks. Reliability includes *dependability*, *robustness*, and *safety*.

**remove user** - initiated by an Elite, an employee is taken out of the system and prevented from entering.

**requirements** – a function that the system must have. or a user-viable constraint on the system.

**response time** – an attribute of the system denoting how quickly it can react to a user input.

**robustness** – The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions.

**safety** – a measure of the absence of catastrophic consequences to the environment

**scenario** – instance of a use case. A scenario represents a concrete sequence of interaction between one or more actors of the system.

**signatures** – given an operation, the *tuple* made up of the types of its parameters and the type of the return values. Operation signatures are specified during object design.

**subsystem decomposition** – the division of the system into subsystems. Each subsystem is described in terms of its services during system design and its API during object design.

**supportability** – the ease of changing the system after deployment.

**system design object model** – a high level description of the system, including *design goals*, *subsystem decomposition*, hardware/software platform, persistent storage strategy, global control flow, access control policy, and boundary condition strategies. The system design model represents the strategic decisions made by the architecture team that allow subsystem teams to work concurrently and cooperate effectively.

**throughput** – an attribute of the system denoting how much work it can accomplish within a specified amount of time.

**timesheet** – the application that serves to facilitate hour logging by allowing the employees to fill out and submit timesheets electronically.

**tuple** – an ordered set of values. Common uses for the tuple are representing a set of value attributes in a relational database and representing an access right.

**type** – describes the legal values the attribute or variable can take.

**usability** – ease with which the user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

**use case**­ – a general sequence of interactions between one or more actors of the system.

**use case diagram** – UML notation used during requirements elicitation and analysis to represent the functionality of the system. A use case describes a function of the system in terms of a sequence of interactions between an actor and the system. A use case also includes *entry conditions* that must be true before executing the use case and the *exit conditions* that are true at the completion of the use case.

**visibilities** – specifies whether or not the attribute can be accessed by other classes.

**work breakdown structure** – a hierarchical decomposition of project work into tasks. Leaves represent tasks that are assigned to participants. Aggregates represent the work associated with a work product.