Functional Requirements

Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform. In product development, it is useful to distinguish between the baseline functionality necessary for any system to compete in that product domain, and *features* that differentiate the system from competitors' products, and from variants in your company's own product line/family. Features may be additional functionality, or differ from the basic functionality along some quality attribute (such as performance or memory utilization).
Use cases have quickly become a widespread practice for capturing functional requirements. This is especially true in the object-oriented community where they originated, but their applicability is not limited to object-oriented systems.

Use Cases

A *use case* defines a goal-oriented set of interactions between external actors and the system under consideration.
Actors are parties outside the system that interact with the system (UML 1999, pp. 2.113- 2.123).
An actor may be a class of users, roles users can play, or other systems. Cockburn (1997) distinguishes between primary and secondary actors. A *primary* actor is one having a goal requiring the assistance of the system. A *secondary* actor is one from which the system needs assistance.
A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the goal. It also includes possible variants of this sequence, e.g., alternative sequences that may also satisfy the goal, as well as sequences that may lead to failure to complete the service because of exceptional behavior, error handling, etc. The system is treated as a "black box", and the interactions with system, including system responses, are as perceived from outside the system.
Thus, use cases capture *who* (actor) does *what* (interaction) with the system, for what *purpose* (goal), without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore defines all behavior required of the system, bounding the scope of the system.
Generally, use case steps are written in an easy-to-understand structured narrative using the vocabulary of the domain. This is engaging for users who can easily follow and validate the use cases, and the accessibility encourages users to be actively involved in defining the requirements.

Use cases may be described at the abstract level (business use case, sometimes called essential use case), or at the system level (system use case). The differences between these are the scope.

Scenarios

A scenario is an instance of a use case, and represents a single path through the use case. Thus, one may construct a scenario for the main flow through the use case, and other scenarios for each possible variation of flow through the use case (e.g., triggered by options, error conditions, security breaches, etc.). Scenarios may be depicted using sequence diagrams.

Structuring Use Cases

UML (1999) provides three relationships that can be used to structure use cases. These are generalization, include and extends. **An *include* relationship between two use cases means that the sequence of behavior described in the included (or *sub*) use case is included in the sequence of the base (including*)* use case**.
Including a use case is thus analogous to the notion of calling a subroutine (Coleman, 1998). The *extends* relationship provides a way of capturing a variant to a use case. Extensions are not true use cases but changes to steps in an existing use case. Typically extensions are used to specify the changes in steps that occur in order to accommodate an assumption that is false (Coleman, 1998). The extends relationship includes the condition that must be satisfied if the extension is to take place, and references to the extension points which define the locations in the base (extended) use case where the additions are to be made.
A *generalization* relationship between use cases “implies that the child use case contains all the attributes, sequences of behavior, and extension points defined in the parent use case, and participates in all relationships of the parent use case.” The child use case may define new behavior sequences, as well as add behavior into and specialize existing behavior of the parent. (UML, 1999)

Use Case Guidelines

Creation

The following provides an outline of a process for creating use cases:

- Identify all the different users of the system
- Create a user profile for each category of user, including all the roles the users play that are relevant to the system.

For each role, identify all the significant goals the users have that the system will support. A statement of the system’s value proposition is useful in identifying significant goals.

- Create a use case for each goal, following the use case template. Maintain the same level of abstraction throughout the use case. Steps in higher-level use cases may be treated as goals for lower level (i.e., more detailed), sub-use cases.
- Structure the use cases. Avoid over-structuring, as this can make the use cases harder to follow.
- Review and validate with users.

Use Case Template

Although use cases are part of UML, there is no template for writing use cases. The following is Derek Coleman's proposal for a standard use case template (Coleman, 1998), with some minor modifications.

Use Case	*Use case identifier and reference number and modification history* Each use case should have a unique name suggesting its purpose. The name should express what happens when the use case is performed. It is recommended that the name be an active phrase, e.g. "Place Order". It is convenient to include a reference number to indicate how it relates to other use cases. The name field should also contain the creation and modification history of the use case preceded by the keyword **history**.
Description	*Goal to be achieved by use case and sources for requirement* Each use case should have a description that describes the main business goals of the use case. The description should list the sources for the requirement, preceded by the keyword **sources**.
Actors	*List of actors involved in use case* Lists the actors involved in the use case. Optionally, an actor may be indicated as primary or secondary.
Assumptions	*Conditions that must be true for use case to terminate successfully* Lists all the assumptions necessary for the goal of the use case to be achieved successfully. Each assumption should be stated as in a declarative manner, as a statement that evaluates to true or false. If an assumption is false then it is unspecified what the use case will do. The fewer assumptions that a use case has then the more robust it is. Use case extensions can be used to specify behavior when an assumption is false.
Steps	*Interactions between actors and system that are necessary to achieve goal* The sequence of interactions necessary to successfully meet the goal. The interactions between the system and actors are structured into one or more steps which are expressed in natural language. A step has the form <sequence number><interaction> Conditional statements can be used to express alternate paths through the use case. Repetition and concurrency can also be expressed (see Coleman, 1997, for a proposed approach to do doing so).
Variations *(optional)*	*Any variations in the steps of a use case* Further detail about a step may be given by listing any variations on the manner or mode in which it may happen. <step reference> < list of variations separated by **or**>
Non-Functional	*List any non-functional requirements that the use case must meet.* The nonfunctional requirements are listed in the form: <**keyword**> : < requirement> Non-functional keywords include, but are not limited to **Performance**, **Reliability, Fault Tolerance**, **Frequency,** and **Priority**. Each requirement is expressed in natural language or an appropriate formalism.
Issues	*List of issues that remain to be resolved* List of issues awaiting resolution. There may also be some notes on possible implementation strategies or impact on other use cases.