Functional Requirements and Use Cases

By Ruth Malan, Hewlett-Packard Company, and
Dana Bredemeyer, Bredemeyer Consulting,
Email: ruth_malan@hp.com and dana@bredemeyer.com

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).

One strategy for quickly penetrating a market, is to produce the core, or stripped down, basic product, and adding features to variants of the product to be released shortly thereafter. This release strategy is obviously also beneficial in information systems development, staging core functionality for early releases and adding features over the course of several subsequent releases.

In many industries, companies produce product lines with different cost/feature variations per product in the line, and product families that include a number of product lines targeted at somewhat different markets or usage situations. What makes these product lines part of a family, are some common elements of functionality and identity. A platform-based development approach leverages this commonality, utilizing a set of reusable assets across the family.

These strategies have important implications for software architecture. In particular, it is not just the functional requirements of the first product or release that must be supported by the architecture. The functional requirements of early (nearly concurrent) releases need to be explicitly taken into account. Later releases are accommodated through architectural qualities such as extensibility, flexibility, etc. The latter are expressed as non-functional requirements.

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

Each *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 thus 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.

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 Diagram

The use case structure is graphically summarized in a *use case diagram* (UML, 1999, pp. 3-83 to 3-88), which also shows which actors interact with which use cases.

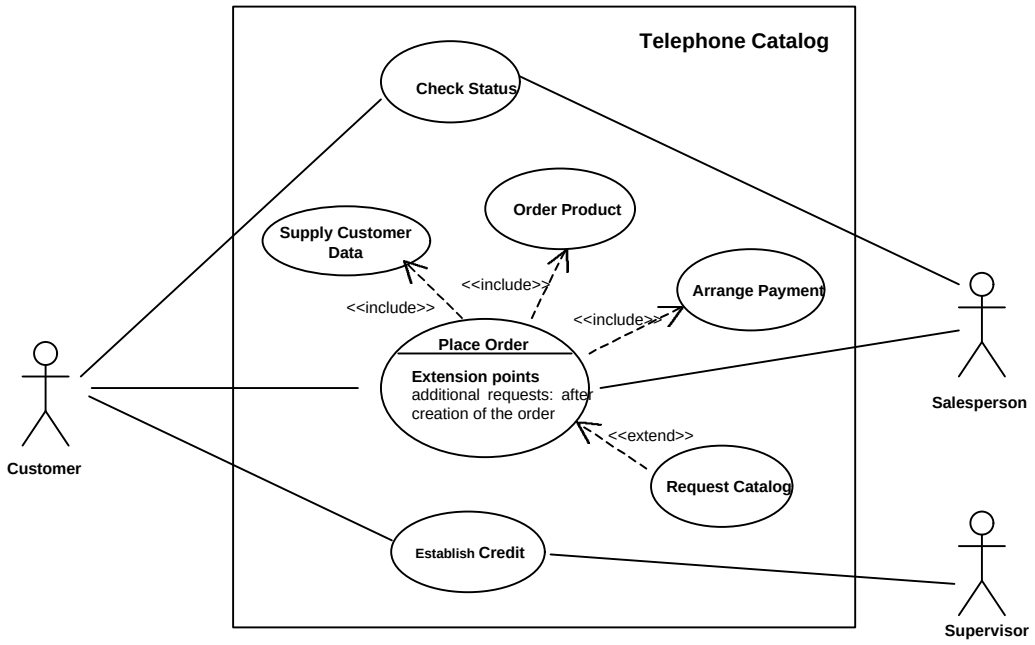

Figure 1. Example use case diagram (adapted from the UML V1.3 document)

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.

Table 1 : Use Case Template (from Coleman, 1998)

Including a Use Case

Included cases are full use cases in their own right, and therefore can be expressed using the use case template (Table 1). Including a sub-use case in a step is expressed by the keyword **INCLUDE**. For example, if Select_Product were a use case it could be used by the following interaction:

INCLUDE Select_Product

Extending a Use Case
The following is Derek Coleman's template for a Use Case Extension (Coleman, 1998), modified to include the condition that is evaluated when the first extension point is reached (UML, 1999 page 2-123):

Use Case Extension	<extension identifier> **extends** <use case identifier> The extension name includes a unique identifier for the extension and a reference to the use case to which the extension applies.
Change	*Goal to be achieved by extension* This section documents the variant of the use case in terms of the assumption that discharges.
Condition	*The condition that must be satisfied if the extension is to take place* The condition is evaluated when the first changed step is reached.
Steps	*Changes to use case steps.* The changes are expressed in terms of new or altered steps that apply to a use case at an *extension point* (i.e. a reference to a step) if some condition is true. <step reference> <changes to step>
Variations *(optional)*	…
Non-Functional	…
Issues	…

Table 2 Template for a Use Case Extension

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[1].
- 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.

The Role of Use Cases in the Architecting Process

The core technical phases of the architecting process are Architectural Requirements, System Structuring, and Architecture Validation (see http://www.bredemeyer.com/how.htm). The following sub-sections describe the role of use cases in each of these phases. Of course, other activities and considerations are brought to bear in each of these phases, and only those relating to use cases are discussed here.

Architectural Requirements
Use cases capture the functional requirements of the system. Since the architecture must support the functional requirements of current and planned systems, it is important that the architects have a good understanding of what is required, at least at the level of abstraction relevant to architecture. A representative set of use cases, covering the major goals of the users and important "corner cases" may be considered architecturally significant. Fine details of low-level use cases would not be. Architects who have little experience in the domain (e.g., it is an entirely new product area), will find it valuable to work

[1] Goals are related to user intentions in Constantine and Lockwood's work.

with more detailed use cases, but in general it is sufficient to keep to a fairly limited set of more abstract use cases.

System Structuring
In creating the meta-architecture[2], the architect's domain knowledge and gross understanding of the functional requirements is most important. Use cases play a role to the extent that they enhance the architect's domain knowledge. This is also largely true when identifying components and their responsibilities and relationships in creating the conceptual architecture. In doing so, the architect is paying a lot of attention to balancing various non-functional (i.e., quality) requirements. For example, responding to a maintainability requirement by isolating things that are likely to change together (such as responsibilities for interfacing to external systems, platform or vendor dependencies, etc.). Another force of consideration, is what functionality is related to what to cluster highly cohesive, tightly coupled responsibilities together.

During logical architecture, the steps in the use cases are fleshed out in collaboration diagrams (or sequence diagrams). This allows the architect to verify the component responsibilities and operations/methods identified so far, and to create new operations as necessary to support the use case step. As this process unfolds, the architect refines the components' interface structure, deciding which operations to cluster together on an interface, etc. The collaboration diagrams (or sequence charts) provide a means for the architect to consider run-time qualities such as performance, security, etc. (For example, doing a back-of-the-envelope calculation or building a skeletal prototype of the architecture to check that the inter-component communication times are not prohibitive, or offering a reasoned argument for how the interactions in the collaboration diagram will be secure.)

Architecture Validation
The validation team validates that the architecture does indeed support the use cases in the requirements set performing a kind of "thought experiment" walking through the collaboration diagrams for the use case steps. They may also bring up new use cases (for planned future systems, or use cases that were pruned from the architectural requirements set because they were similar to other use cases, or too detailed, etc.), to assess the impact on the architecture.

Similarly, in system evolution or migration, use cases for new functionality can be used to assess the impact of adding the functionality on the architecture (do components have to be added or just changed, etc.).

Conclusion

Use cases are useful in capturing and communicating functional requirements, and as such they play a primary role in product definition. An architecturally relevant subset of the use cases for each of the products to be based on the architecture also plays a valuable role in architecting. They direct the architects to support the required functionality, and provide the starting points for collaboration diagrams (or sequence diagrams) that are helpful in component interface design and architecture validation.

There are some shortcomings to use cases, and these include:

i. Use cases do not offer a means to reflect commonality/variability across products in a product line or family.
ii. Many teams are not able to decide on the appropriate level of abstraction to which to take the use cases, and experience an uncontrolled and time-consuming proliferation in their use cases. It may be that a more traditional list of core functions and differentiating features is more succinct, though we have seen such documents also expand into hundreds of pages!
iii. By using a non-functional field on the use case, use case-specific non-functional requirements can be captured. However, many non-functional requirements are not specific to use cases (e.g., maintainability, reuse, etc.) or are fulfilled by a conjunction of use cases.

[2] See glossary.

iv. Inexperienced teams may revert to functional decomposition by driving component identification too closely off the use cases (Meyer, 1997). The danger is that the system will not be very extensible.

Recommended Reading on Use Cases

Booch, G., I. Jacobson and J. Rumbaugh, *The Unified Modeling Language User Guide.* Addison-Wesley, 1999, pp. 219-241.

Christerson, Magnus, "From Use Cases to Components", *Rose Architect*, 5/99. http://www.rosearchitect.com/cgi-bin/viewprint.pl

Cockburn, Alistair, "Structuring Use Cases with Goals", *Journal of Object-Oriented Programming*, Sep-Oct, 1997 and Nov-Dec, 1997. Also available on http://members.aol.com/acockburn/papers/usecases.htm

Cockburn, Alistair, "Basic Use Case Template", Oct .1998. Available on http://members.aol.com/acockburn/papers/uctempla.htm

Coleman, Derek, "A Use Case Template: Draft for discussion", *Fusion Newsletter*, April 1998. http://www.hpl.hp.com/fusion/md_newsletters.html

Constantine, Larry. "What Do Users Want? Engineering usability into software", http://www.foruse.com

Pols, Andy, "Use Case Rules of Thumb: Guidelines and lessons learned", *Fusion Newsletter*, Feb. 1997. available at http://www.hpl.hp.com/fusion/md_newsletters.html.

UML Specification. http://www.rational.com/uml/index.jtmpl. We have referenced V1.3 Alpha R5, March 1999 in this paper.

Use Case Bibliography

Booch, G., I. Jacobson and J. Rumbaugh, *The Unified Modeling Language User Guide.* Addison-Wesley, 1999, pp. 219-241.

Christerson, Magnus, "From Use Cases to Components", *Rose Architect*, 5/99. http://www.rosearchitect.com/cgi-bin/viewprint.pl

Cockburn, Alistair, "Structuring Use Cases with Goals", *Journal of Object-Oriented Programming*, Sep-Oct, 1997 and Nov-Dec, 1997. Also available on http://members.aol.com/acockburn/papers/usecases.htm

Cockburn, Alistair, "Basic Use Case Template", Oct .1998. Available on http://members.aol.com/acockburn/papers/uctempla.htm

Cockburn, Alistair, "More About Use Cases" web page http://members.aol.com/acockburn/papers/OnUseCases.htm

Cockburn, Alistair, "Use Cases in Theory and Practice", presentation. Slides available on http://members.aol.com/humansandt/usecases/usecasetalk.ppt

Coleman, Derek, "A Use Case Template: Draft for discussion", *Fusion Newsletter*, April 1998. Available at http://www.hpl.hp.com/fusion/news/apr98.ppt.

Collins-Cope, Mark, "The Requirements/Service/Interface (RSI) Approach to Use Case Analysis", Ratio Group white paper available on http://www.ratio.co.uk/RSI.htm

Ett, William, "Preparing a Use Case Model", http://source.asset.com/stars/loral/cleanroom/oo/scsuc1.htm

Fowler, Martin, "Use and Abuse Cases", *Distributed Computing*, 4/98. http://www.DistributedComputing.com

Jacobson, I., M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use-Case Driven Approach*, Addison-Wesley, 1992.

Jacobson, I., G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

Korson, Timothy, "The Misuse of Use Cases (Managing Requirements)", 3/98. Available on http://www.software-architects.com/publications/korson/Korson9803om.htm

Korson, Timothy, "Configuring a Use case Process (Managing Requirements, Part 2)". Available on http://www.software-architects.com/publications/korson/usecase2.htm

Meyer, Bertrand, "OOSC2: The Use Case Principle", Objected-Oriented Software Construction 2nd ed. 1997. Available on http://www.elj.com/elj/v1/n2/bm/usecases/

Pols, Andy, "Use Case Rules of Thumb: Guidelines and lessons learned", *Fusion Newsletter*, Feb. 1997. available at http://www.hpl.hp.com/fusion/news/feb97.ppt.
Rosenberg, D. and K. Scott, *Use Case Driven Object Modeling with UML*, Addison-Wesley, 1999.
Schneider, G. and J. Winters, *Applying Use Cases: A Practical Guide*, Addison-Wesley, 1998.

The following papers are available on the web, without author ascription:

"Applying Use Case Modeling", Blueprint Technologies White Paper. http://www.blueprint-technologies.com/training/whitepapers/requir.html
"Requirements Architecture: A Use Case Driven Approach", Blueprint Technologies White Paper. http://www.blueprint-technologies.com/training/whitepapers/requir.html
UML Specification. http://www.rational.com/ We have referenced V1.3 Alpha R5, March 1999 in this paper.
"Use Case Fundamentals", 1998. http://members.aol.com/acockburn/papers/AltIntro.htm
"Use Case Modelling: Capturing user requirements". http://www.zoo.co.uk/~z0001039/PracGuides/pg_use_cases.htm

Glossary

Actor
Actors, in use case parlance, are parties outside the system that interact with the system. They may be users or other systems. Each actor defines a coherent set of roles users of the system can play (UML, 1999). 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 to satisfy its goal.

Architecture
The term software architecture is used both to refer to the high-level structure of software systems and the specialist discipline or field distinct from that of software engineering.

The architecture of a software system identifies a set of components that collaborate to achieve the system goals. The architecture specifies the "externally visible" properties of the components—i.e., those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on (Bass et al., 1998). It also specifies the relationships among the components and how they interact.

Conceptual Architecture
The intent of the conceptual architecture is to direct attention at an appropriate decomposition of the system without delving into the details of interface specification and type information. Moreover, it provides a useful vehicle for communicating the architecture to non-technical audiences, such as management, marketing, and many users. The conceptual architecture identifies the system components, the responsibilities of each component, and interconnections between components. The structural choices are driven by the system qualities, and the rationale section articulates and documents this connection between the architectural requirements and the structures (components and connectors or communication/co-ordination mechanisms) of the architecture.

Features
Features are the differentiating functionality of a product. This functionality may not be available in other products, or it may not be available with the same quality characteristics.

Functional Requirements
Functional requirements capture the intended behavior of the system—or *what* the system will do. This behavior may be expressed as services, tasks or functions the system is required to perform.

Logical Architecture
The logical architecture is the detailed architecture specification, precisely defining the component interfaces and connection mechanisms and protocols. It is used by the component designers and developers.

Meta-architecture
The meta-architecture is a set of high-level decisions that will strongly influence the structure of the system, but is not itself the structure of the system. The meta-architecture, through style, patterns of composition or interaction, principles, and philosophy, rules certain structural choices out, and guides selection decisions and tradeoffs among others. By choosing communication or co-ordination mechanisms that are repeatedly applied across the architecture, a consistent approach is ensured and this simplifies the architecture. (See http://www.bredemeyer.com/how.htm and http://www.bredemeyer.com/whatis.htm.)

Non-functional Requirements
Non-functional requirements or system qualities, capture required properties of the system, such as performance, security, maintainability, etc.—in other words, *how well* some behavioral or structural aspect of the system should be accomplished.

Product Line
Product lines consist of basically similar products with different cost/feature variations per product.

Product Family
Product families include a number of product lines targeted at somewhat different markets or usage situations. What makes the product lines part of a family, are some common elements of functionality and identity.

Qualities
System qualities, or non-functional requirements, capture required properties of the system, such as performance, security, maintainability, etc.—in other words, *how well* some behavioral or structural aspect of the system should be accomplished.

Scenario
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 error conditions, security breaches, etc.). Scenarios may be depicted using sequence diagrams.

Use Case
A use case defines a goal-oriented set of interactions between external actors and the system under consideration. That is, use cases capture *who* (actors) does *what* (interactions) with the system, for what *purpose* (goal). A complete set of use cases specifies all the different ways to use the system, and thus defines all behavior required of the system--without dealing with the internal structure of the system.