Use-Case Modeling Guidelines

Version 0.01

Table of Contents

1. Document Control Information 3
   1.1 Revision History 3
   1.2 Reviewers and Approvers 3

2. Introduction 4
   2.1 Purpose 4
   2.2 Scope 4
   2.3 References 4
   2.4 Overview 4

3. Use-Case Modeling Guidelines 4
   3.1 Evolution of a Use Case 4
   3.2 Use-Case Model Structure 4

4. How to Describe a Use Case 6
   4.1 General Style 6
   4.2 Use Case Name 6
   4.3 Use Case Brief Description 7
   4.4 Use Case Flows of Events Style 7
   4.5 Basic Flow 7
   4.6 Alternative Flows 7
   4.7 Sub-flows 7
   4.8 Preconditions and Post-conditions 8
   4.9 Extension Points 8
   4.10 Use of Scenarios 8
   4.11 Use of Glossary Terms 8
   4.12 Additional Guidelines 9

5. Use Case Checkpoints 10

1. Document Control Information

1.1 Revision History

Date	Version	Description	Author

1.2 Reviewers and Approvers

Reviewer	Functional Area

2. Introduction

2.1 Purpose

This document describes use-case modeling guidelines, and may aid teams that are new to using a use case driven approach for requirements management. It is simply a guide. It also provides information on how use cases may be realized in a modeling tool such as RSA or Rose. This use-case modeling guideline is being created to provide project standards for the development and maintenance of requirements with use cases, and to ensure a consistent look and feel of all use-case related artifacts.

2.2 Scope

2.3 References

- Use Case Modelling, Kurt Bittner, Ian Spence. June 2003.
- Rational Unified Process
- OMG Unified Modeling Language Specification, Version 2.0. (http://www.omg.org)

2.4 Overview

This set of guidelines is organized into two sections; the first describes the preferred way of *modelling* the Use-Cases, the second part provides guidelines for the *content* of the Use-Case model and for *naming* the elements within the model.

3. Use-Case Modeling Guidelines

3.1 Evolution of a Use Case

The development of a use case is an iterative process. A use case evolves from the spark of an idea to a fully described description of how a system behaves.

3.1.1 Discovered

Initially, a use case is "discovered." A use case is discovered once it has been named based upon the goal an actor is trying to achieve.

3.1.2 Briefly Described

Immediately after discovering a use case (usually while the name is being discussed), a brief description of the use case is created. The brief description describes the goal of the use case and the value provided to the actor. Two or three sentences will usually suffice.

3.1.3 Outlined

The next stage of evolution is to outline the use case. The outline contains a bulleted list of just the basic flow, and identification (not outline) of possible alternative flows. This enables scenarios to be identified, and helps to provide an understanding of the possible size and complexity of the use case.

3.1.4 Fully Described

The final stage is to fully describe the use case. This in itself is done iteratively. Particular flows for development in an iteration are identified, then fully described (and implemented). Subsequent iterations identify additional flows that are then also fully described (and implemented). The complete use case will be fully described at the completion of the project.

3.2 Use-Case Model Structure

3.2.1 Actor Guidelines

Each concrete Use Case will be involved with at least one Actor.

Actors will be given intuitive and descriptive actor name(s) that correspond to their roles.

3.2.2 *Keep this section* Communicates Association

Relationships between actors and use cases are called communicates-associations. A communicates association between an actor and a use-case indicates that they interact, i.e. the actor participates in and communicates with the system containing the use case. Communicates associations **do not describe data flow**. Interactions could be mechanical, electrical, data, audible, visual, or any combination of these. For example, an actor may press a button on the system (mechanical stimulus) and the system could turn on a light (visual response).

3.2.3 <<Communicates>> relationship

A communicates-association is represented on UML diagrams as a solid line. The line epresents a two-way or whole dialog between the actor and the system.

3.2.4 <<Include>> and <<Extend>> relationships

This information is not covered in the course. These activities should be performed by architects after use cases are detailed.

<<Include>> and <<Extend>> relationships can be used to:

1. Factor out behavior that is in common for two or more use cases.
2. Factor out behavior from the base use case that is not necessary for the understanding of the primary purpose of the use case, only the result of it is important.
3. To show that there may be a set of behavior segments of which one or several may be inserted at an extension point in a base use case.

It is recommended that the use of the <<include>> and << extend>> relationships be avoided. These relationships have much more potential to clutter and confuse than they have to help simplify the Use-Case Model. The best practice is to avoid this type of decomposition initially, and consider using these relationships at a later stage in the process only when common behaviors can be found in multiple use cases. They should only be used where they add value by helping to simplify and manage the use-case model.

3.2.4.1 «include» relationship

The «include» relationship describes a behavior segment that is inserted into a use-case instance that is executing the base use case. It is a mechanism similar in nature to a sub-routine, and is most often used to factor out common behavior. The «include» relationship is only to be used when there are flows that are common to multiple use cases.

Overuse of the «include» relationship increases the complexity for the testers, designers and documentation authors because each «include» introduces another artifact that must be read in order to understand the complete requirements. Care must also be taken when using the «include» relationship that the use-case model does not become functionally decomposed.

3.2.4.2 «extend» relationship

If there is a part of a base use case that is optional, or not necessary to understand the primary purpose of the use case, that part can be factored out, and put into an additional use case in order to simplify the structure of the base use case. The addition is implicitly inserted in the base use case, using the extend relationship.

The extension is conditional, which means its execution is dependent on what has happened while executing the base use case (similar to an alternative flow). The base use case does not control the conditions for the execution of the extension. Those conditions are described within the extend relationship.

The base use case is implicitly modified by the extensions; the base use case defines a framework into which extensions can be added, but the base does not have any visibility into the specific extensions.

The base use case has no knowledge of the use case that extends it and therefore cannot see and may not access the properties of the extending use case. The extending use case knows which use case it extends and can see all the properties of the base use case. The extension use case may access and modify properties of the base use case.

The base use case should be complete in and of itself, meaning that it should be understandable and meaningful without any references to the extensions. However, the base use case is not independent of the extensions, because it cannot be executed without the possibility of following the extensions.

The base use case must identify ***extension points*** in a separate section of the use-case specification. The extending use case then indicates where it extends the base use case by referencing these named extension points in the base use case.

3.2.5 Actor-Generalization

This information was not covered in the course.

Actor generalization is used to simplify the use-case diagram and avoid the "chopstick effect" of communicates associations when multiple actors all execute the use case for the same purpose (i.e. actors playing the same role.) In general, Actor-Generalization can be used to better define the different roles played by the users of the system to be developed. This is useful in applications with different "categories" of end-users. In this way, only relevant functionality will be presented to each category of users, and we are able to control the access rights based on this grouping.

3.2.5.1 Use of Actor-Generalization

Each use-case will only be initiated by one Actor. This requirement may be overridden, in which case the use-case description must justify the decision.

3.2.6 Use-Case Generalization

This relationship is prohibited from use.

3.2.7 Use of Interaction Diagrams

In some cases, it is beneficial to include, in addition to the textual flow of events, an Interaction diagram to illustrate the "high level" flow of events of the use case. It is recommended the sequence diagram for this be drawn in Rational Rose. Include only the communication between the actors and the boundary objects (covering both the input and the output messages) and treat the system as a black box. Use boundary objects with logical names as defined in the use case flow of events, without assigning them to classes at this point.

Interaction diagrams are optional.

3.2.8 Use of Activity Diagrams

In some cases, an activity diagram can add value in helping to define, clarify and complete the flow of events in the use case. It is recommended that activity diagrams be modelled in RSA. A good rule of thumb is to consider Activity Diagrams for complex use-cases (containing several alternative or exceptional flows). The activity diagram shows a decision tree of the flows in the use-case.

Activity diagrams are optional.

4. How to Describe a Use Case

4.1 General Style

The Use Cases will be written using the template referenced in 2.3, "References."

4.2 Use Case Name

The use-case name will be unique, intuitive, and explanatory so that it clearly and unambiguously defines the observable result of value gained from the use case.

Each use-case name will describe the behavior the use case supports. The name will combine both the action being performed and the key element being "actioned". Most often, this will be a simple verb – noun combination. The use case should be named from the perspective of the actor that triggers the use case.

A good check for the use-case name is to survey whether customers, business representatives, analysts and developers all understand the names and descriptions of the use cases, and that the observable result of value from the actors perspective is defined.

4.3 Use Case Brief Description

The use case will contain a brief description. This description will be at least 1 paragraph and no more than 3 paragraphs in length. The description will cover an explanation of the key purpose, value proposition, and key concepts of the use case.

4.4 Use Case Flows of Events Style

The use-case flows will make use of tag headings to describe the steps at a high-level, followed by numbered descriptions of what the actor does and what the system does under the heading. These tagged steps will be referenced in alternative flows and sub-flows.

Tags will be in a bold font, enclosed in curly braces, and can be formatted in Microsoft Word as headings. Sub-steps (use case detail) will be numbered within each tagged step, with numbering continuing from the previous tagged step.

4.5 Basic Flow

The first step of the use case description will start with "The use case begins when the actor…"

Each time the interaction between the actor and the system changes focus (between the actor and the system), the next segment of behavior will start with a new paragraph. Begin first with an actor and then the system.

Each step in the flow of events should show a roundtrip of events, from actor to system, and from system to actor, usually in the form of messages:

- What the actor does: <Actor> messages <System> , and
- What the system does in response: <System> messages <An Actor> ...

Making each step a roundtrip ensures testability, ensures adherence to the purpose of a use case, and makes sequence diagrams far easier to develop and read.

The last step of the use case description will end with "The use case ends."

4.6 Alternative Flows

Alternative flows will be used to document regular variants, odd cases, and exceptional (error) flows.

Each alternative Flow will explicitly and clearly define all of the possible entry points into the flow, and will conclude with all of the possible exit points from the flow.

The alternative flow will also state explicitly the exit point and where the actor continues to next, i.e. whether the flow returns to a specific step in the basic flow, or ends.

Alternative flows are described in their own section, not within the basic flow. The basic flow will not reference any alternative flows.

4.6.1 Using Alternative flows

A specific alternative flow occurs at a specific step in the basic (or other) flow. A generic alternative flow can occur anywhere in another flow. For specific alternative flows, the following information is detailed:

- The start location in the basic or another sub-flow where the alternative flow is triggered.
- The condition that triggers its start.
- The actions taken in the alternative flow.
- Where the basic or another sub-flow is resumed after the alternative flow ends.

If the use case ends in the alternative flow, state that "The use case ends" in the alternative flow. For generic alternative flows, there is no start location because a generic alternative flow can begin anywhere.

Alternative flows called out in descriptions will be identified in bold, italics text.

4.7 Sub-flows

Where the flow of events becomes cluttered due to complex behavior, or where a single flow exceeds a physical printed page in length, sub-flows can be used to improve clarity and manage the complexity. Sub-flows can also facilitate reuse of flows within the same use case. Sub-flows will be written by moving a self-contained, logical

group of detailed behavior to a sub-flow, and referencing this behavior in summary form within the flow of events. A sub-flow can be thought of as an “internal include”.

The difference between an alternative flow and a sub-flow is that alternative flows insert themselves into another flow. The flow it inserts itself into has no knowledge of the alternative flow. An alternative flow may also resume at any place within the use case. A sub-flow is explicitly called from a flow. When a sub-flow complete it always returns to the line after it was called from. It is similar in concept to a subroutine call in some programming languages.

Sub-flows are not listed as part of a scenario. By definition, if the calling flow is in the scenario, the sub-flow is also in the scenario. As with any other flow, a sub-flow may have alternative flows.

Sub-flows are not described as part of a scenario. By definition, if the calling flow is in the scenario, the sub-flow is also in the scenario.

As with any other flow, a sub-flow may have alternative flows.

4.7.1 Using Sub-flows

Sub-flows are described in their own section of the use case. Sub-flows have an identifier in front of them (S1, S2, and so on) to identify them as sub-flows.

A sub-flow is explicitly called from a flow. When a sub-flow completes, it always returns to the line after it was called, therefore, sub-flows do not need a line that says where they return. It is similar in concept to a subroutine call in some programming languages. Each sub-flow will explicitly and clearly define all of the possible entry points into the flow.

Sub-flows called out in descriptions will be identified in bold, italics text.

4.8 Preconditions and Post-conditions

The use case specification will include a set of conditions (also referred to as assumptions) that are expected to be true before the use case begins (preconditions), and after the use case has ended (post-conditions). Note that the use case may end in a number of ways, and each post-condition should be described accordingly.

4.8.1 Use of Preconditions

Preconditions describe the state the system must be in before the use case can start. All preconditions must be true. Preconditions reduce the need for validation inside the use case, but cannot be used to describe things outside the system.

4.8.2 Use of post-conditions

Post-conditions describe the state of the system at the end of the use case. Post conditions are guaranteed true at the end of the use case, regardless of how the use case ends. Any (but at least one) of the post-condition will be true.

4.9 Extension Points

4.9.1 Include Relationship

4.9.2 Extend Relationship

4.10 Use of Scenarios

Scenarios represent an instance of a use case. It is one flow through a use case. Document as many scenarios as are required to understand the system being developed, but architecturally significant, and high-risk use cases must be documented.

4.11 Use of Glossary Terms

All terms used in a use case will be defined in the project's Glossary. If a term exists in a use case that does not exist in the glossary, the term needs to either:

1. Be added to the glossary, including a brief description (maximum of one paragraph).
2. Be changed in the use case to reflect the correct term defined in the glossary.

Terms that are specific only to a particular use case may be defined in the glossary section of that use case.

Items defined in the glossary will be underlined.

4.12 Additional Guidelines

4.12.1 Inline conditional and repetitive behavior

The use of conditional and looping statements like IF, THEN, ELSE, WHILE, REPEAT, UNTIL, etc. in the flows of events, make identifying scenarios difficult, and are therefore prohibited. All conditional and repetitive behavior must be expressed in alternative flows. Activity diagrams can be used to visualize alternative flows.

The following statements are permitted in alternative flows:

- IF – used to express the condition the alternative flow is executed
- RESUME – used to express where the alternative flow resumes in the use case.

4.12.2 Consistent Use of Actor Name(s)

The use case specification will be written using consistent actor name(s). Do not refer generically to "the actor," instead use the actual name used to uniquely identify or define the actor. Care will be taken to ensure actor naming is clear and unambiguous.

Actor names will be in bold text.

4.12.3 Consistent use of the imperative: Will

System requirements within the use cases will be written using the imperative. The term "Will" has been chosen in favor of "Shall" and "Must" to describe requirements consistently. The use of passive terms that imply the requirement is optional or undefined such as "should", "possibly", "etc", "might" or "may" will be avoided.

4.12.4 Use of "Action" Terms

4.12.4.1 Define where the system is responsible for presenting the Action Option

The use case will explicitly state where the system is responsible for presenting an action as an available option for the actor to select. In most cases, the available options should be presented as part of the basic flow, and be referenced as the entry point in the first statement in the corresponding alternative flow.

4.12.4.2 Consistent use of the term throughout the Use Case

The use of terms such as New, Modify, Cancel, Delete, OK, and Print will be consistent throughout the use case: The same logical action will not be referred to using different terminology. Special care will be taken to ensure that the Action Terms used in the Alternative Flows match those used in the basic flow.

4.12.5 Use of placeholders for missing detail (TBD or None)

Where information is not yet defined or not yet decided, the use case will include a reference to the issue or element and will include the placeholder "TBD". If the section will be blank, indicate this with the placeholder "None".

4.12.6 Definition of and Reference to Supplementary Specifications

Where there are additional requirements that cannot be described naturally during the flow of events, these will be defined as supplementary requirements. For those that are specific to a use case, these will be defined in the Special Requirements section of the use case specification.

For those requirements that are applicable system-wide, especially those of a non-functional nature, these requirements will be defined in the separate supplementary specification document.

4.12.7 User Interface

The user interface will not be specified in the use case. The use case will be agnostic to the user interface.

Words to **AVOID**

Click	Drag	Form
Open	Close	Drop
Button	Field	Drop-down
Pop-up	Scroll	Browse
Record	Windw	

Words to **Use**

Prompots	Chooses
Initiates	Specifies
Submits	Selects
Starts	Displays
Informs	

4.12.7.1 Crosscheck with UI Prototype/ Design

The use case contents will be cross-checked against the UI Prototype/ Design to ensure no system require ments are missing from the use case or the UI Prototype/ Design. Where changes are required to the use case, the use case must be updated to reflect the UI design requirements.

5. Use Case Checkpoints

1. *Question:* **Is each concrete use case involved with at least one actor?**

   Answer: If not, something is wrong; a use case that does not interact with an actor is superfluous, and you should remove it.

2. *Question:* **Is each use case independent of the others?**

   Answer: If two use cases are always activated in the same sequence, you should probably merge them into one use case.

3. *Question:* **For an included use case: does it make assumptions about the use cases that include it?**

   Answer: Such assumptions should be avoided, so that the included use case is not affected by changes to the including use cases.

4. *Question:* **Do any use cases have very similar behaviors or flows of events?**

   Answer: If so - and if you wish their behavior to be similar in the future - you should merge them into a single use case. This makes it easier to introduce future changes. Note: you must involve the users if you decide to merge use cases, because the users, who interact with the new, merged use case will probably be affected.

5. *Question:* **Has part of the flow of events already been modeled as another use case?**

   Answer: If so, you can have the new use case use the old one.

6. *Question:* **Is some part of the flow of events already part of another use case?**

   Answer: If so, you should extract this subflow and have it be used by the use cases in question. Note: you must involve the users if you decide to "reuse" the subflow, because the users of the existing use case will probably be affected.

7. *Question:* **Should the flow of events of one use case be inserted into the flow of events of another?**

   Answer: If so, you model this with an extend-relationship to the other use case.

8. *Question:* **Do the use cases have unique, intuitive, and explanatory names so that they cannot be mixed up at a later stage?**

   Answer: If not, you change their names.

9. *Question:* **Do customers and users alike understand the names and descriptions of the use cases?**

   Answer: Each use-case name must describe the behavior the use case supports.

10. *Question:* **Does the use case meet all the requirements that obviously govern its performance?**

    Answer: You must include any (nonfunctional) requirements to be handled in the object models in the use-case Special Requirements.

11. *Question:* **Does the communication sequence between actor and use case conform to the user's expectations?**

12. *Question:* **Is it clear how and when the use case's flow of events starts and ends?**

13. *Question:* **Behavior might exist that is activated only when a certain condition is not met.**

    Answer: Is there a description of what will happen if a given condition is not met?

14. *Question:* **Are any use cases overly complex?**

    Answer: If you want your use-case model to be easy to understand, you might have to split up complex use cases.

15. *Question:* **Does a use case contain disparate flows of events?**

    Answer: If so, it is best to divide it into two or more separate use cases. A use case that contains disparate flows of events will be very difficult to understand and to maintain.

16. *Question:* **Is the subflow in a use case modeled accurately?**

17. *Question:* **Is it clear who wishes to perform a use case? Is the purpose of the use case also clear?**

18. *Question:* **Does the brief description give a true picture of the use case?**