Use Cases

Once the initial round of requirements elicitation is done and a discussion summary has been distributed and reviewed, the requirements analyst is ready to begin creating use cases. A *use case* is a description of a specific interaction that a user may have with the software. Use cases are deceptively simple tools for describing the behavior of the software.

A use case contains a textual description of all of the ways that the intended users could work with the software through its interface. Use cases do not describe any internal workings of the software, nor do they explain how that software will be implemented. They simply show the steps that the user follows to use the software to do his work. All of the ways that the users interact with the software can be described in this manner.

A typical use case includes these sections, usually laid out in a table. Table 6-2 shows a template for describing a use case.

TABLE 6-2. Use case template

Name	**Use case number and name**
Summary	Brief description of the use case
Rationale	Description of the reason that the use case is needed
Users	A list of all of the categories of users that interact with this use case
Preconditions	The state of the software before the use case begins
Basic Course of Events	A numbered list of interactions between the user and one or more users
Alternative Paths	Conditions under which the basic course of events could change
Postconditions	The state of the software after the basic course of events is complete

Name, Summary, Rationale, and Users

Each use case must begin with information that allows the reader to uniquely identify it. Every use case has a descriptive name and a unique identifying number. The number is used as a way to refer to a specific use case in the SRS (see below). In addition to identifying information, each use case has a summary, or a brief description of what the use case does.

The "Rationale" section of the use case contains one or more paragraphs that describe why the use case is needed. This serves as an important quality check to ensure the correctness of the use case. While it is important that the team agrees on the behavior of the software, it is equally important that they also agree on why the software is being created.

Each use case represents a series of interactions between the software and one or more users. The users are divided into categories based on the way they interact with the software; the "Users" section lists the kinds of users that interact with this use case. If all categories of users interact with this particular use case, it should list "any user" in this section.

Preconditions and Postconditions

Any software that is being executed can be thought of as being in a state of operation. When the software is in a certain state, it means that a specific set of operations are available to the user that are not available when the software is in other states. For example, a word processor could have an existing document loaded, a new document displayed, or it could be displaying no document at all. It could be showing a configuration window or a dialog box. These are all different, distinct states that the word processing software can be in. There are certain actions that are only available to the user when the software is in a particular state. For example, the user can enter text into the document if the word processor has an existing document loaded, but not if it is displaying a dialog box.

The *precondition* is the state of the software at the beginning of the use case. This represents the entry criteria for the use case: if the software is not in that state, the use case does not apply. The *postcondition* is the state that the software is left in after the use case is completed. In the use case, each of these states can be described in words ("the word processor is loaded but no document is being edited"), although it is also possible to create a name for each state and refer to it by name.

Basic Course of Events

The *basic course of events* is the core of the use case. Table 6-3 shows a very simple basic course of events for a word processor's search-and-replace feature.

TABLE 6-3. Basic course of events for search-and-replace

Precondition	A document is loaded and being edited.
Basic course of events	1. The user indicates that the software is to perform a search-and-replace in the document. 2. The software responds by requesting the search term and the replacement text. 3. The user inputs the search term and replacement text and indicates that all occurrences are to be replaced. 4. The software replaces all occurrences of the search term with the replacement text.
Postcondition	All occurrences of the search term have been replaced with the replacement text.

The basic course of events consists of a series of steps. The first step will generally be the action that the user takes in order to initiate the use case. The remaining steps are a series of interactions between the user and the software. Each interaction consists of one or more actions that the user takes, followed by a response by the software. In this case, the software is responding to the user entering the search term and replacement text and indicating that the replacement should occur.

This basic course of events does not contain words like "click," "button," "text box or "window." User interface design elements should be left out of use cases to allow the designer as many options as possible. For example, this particular use case could be implemented as a pop-up dialog box that contains text boxes for the search term and replacement text, and a button that says "Replace All." (This is how many word processors, including Microsoft Word, do it). But that's not the only way to satisfy this use case. In the Emacs text editor, for example, the user hits Meta-X and enters "replace-string" on the

bottom line of the window, followed by the search term and the replacement term. Either implementation would satisfy this use case.

Sometimes there is functionality that is replicated in many use cases. For example, say the use cases for a program to play music files includes UC-15 (see Table 6-4), a use case that allows a user to explore and edit information in one of his audio files.

TABLE 6-4. Basic course of events for "UC-15: Edit Audio File Information"

Precondition	An audio file is highlighted.
Basic course of events	1. The user indicates that the software is to display information about the audio file. 2. The software responds by displaying the information fields (track name, artist, length, genre, and year) associated with the audio file. 3. The user indicates that one of the information fields is to be replaced and specifies a replacement value. 4. The software updates the audio file with the new value.
Postcondition	Same as precondition state, except the audio file has an updated information field.

If the requirements analyst intends that the user be given the option to select an audio file in many different use cases and edit that file at any time, each of those use cases may have a use case step or an alternative path that reads: "The user indicates that the file is to be edited. Use case UC-15 is executed." In this way, multiple use cases can be extended to include this functionality. The requirements analyst needs to describe it only once, which makes the use cases clearer. This technique has the benefit of giving the designers and programmers hints about where they can reuse code when the software is being developed.

Alternative Paths

Often, a use case has one basic course of events, as well as several alternative courses that are very similar and that share many of the same steps. In these cases, they are documented as different *alternative paths* (rather than in separate use cases), to show that they are closely related. An alternative path provides behavior that is similar to the basic course of events but that differs in one or more key behaviors.

For example, an alternative path for the search-and-replace use case would be to only replace the first occurrence of the search string. Table 6-5 shows the alternative path for this behavior, as well as an alternative path for searching without replacement and another one for aborting the operation:

TABLE 6-5. Alternative paths for search-and-replace

Alternative paths	1. In Step 3, the user indicates that only the first occurrence is to be replaced. In this case, the software finds the first occurrence of the search term in the document being edited and replaces it with the replacement text. The postcondition state is identical, except only the first occurrence is replaced and the replacement text is highlighted. 2. In Step 3, the user indicates that the software is only to search and not replace, and does not specify replacement text. In this case, the software highlights the first occurrence of the search term and the use case ends. 3. The user may decide to abort the search-and-replace operation at any time during Steps 1, 2, or 3. In this case, the software returns to the precondition state.

As the requirements analyst defines additional alternative paths, it may become clear that one of them is more likely to be used than the basic course of events. In this case, it may be useful to swap them—make the alternative path into the basic course of events, and add a new alternative path to describe the behavior previously called the basic course of events.

Table 6-6 shows a final use case for a search-and-replace function, which is numbered UC-8 in this example.

TABLE 6-6. Use case for a simple search-and-replace function

Name	**UC-8: Search**
Summary	All occurrences of a search term are replaced with replacement text.
Rationale	While editing a document, many users find that there is text somewhere in the file being edited that needs to be replaced, but searching for it manually by looking through the entire document is time-consuming and ineffective. The search-and-replace function allows the user to find it automatically and replace it with specified text. Sometimes this term is repeated in many places and needs to be replaced. At other times, only the first occurrence should be replaced. The user may also wish to simply find the location of that text without replacing it.
Users	All users
Preconditions	A document is loaded and being edited.
Basic course of events	1. The user indicates that the software is to perform a search-and-replace in the document. 2. The software responds by requesting the search term and the replacement text. 3. The user inputs the search term and replacement text and indicates that all occurrences are to be replaced. 4. The software replaces all occurrences of the search term with the replacement text.
Alternative paths	1. In Step 3, the user indicates that only the first occurrence is to be replaced. In this case, the software finds the first occurrence of the search term in the document being edited and replaces it with the replacement text. The postcondition state is identical, except only the first occurrence is replaced, and the replacement text is highlighted. 2. In Step 3, the user indicates that the software is only to search and not replace, and does not specify replacement text. In this case, the software highlights the first occurrence of the search term and the use case ends. 3. The user may decide to abort the search-and-replace operation at any time during Steps 1, 2, or 3. In this case, the software returns to the precondition state.
Postconditions	All occurrences of the search term have been replaced with the replacement text.

Develop Use Cases Iteratively

As the use cases are developed, additional information about how the software should behave will become clear. Exploring and writing down the behavior of the software will lead a requirements analyst to understand various aspects of the users' needs in a new light, and additional use cases and functional requirements will start to become clear as well. As this happens, they should be written down with a name, number, and summary—once they are in this form, the analyst can apply the four-step process to complete them.

The first step in developing use cases is identifying the basic ones that will be developed. The list of features in the vision and scope document is a good starting point, as there will usually be at least one use case per feature (usually more than one). This will probably not be the final set of use cases—additional ones will probably be discovered during the development of the use cases.

Many requirements analysts have found that a four-step approach is effective in developing use cases. Table 6-7 contains a script that describes this approach.

TABLE 6-7. Use case development script

Name	**Use case development script**
Purpose	A four-step approach to use case development
Summary	This approach to developing use cases allows the information to be gathered and documented naturally, in a way that lends itself to an iterative approach of alternating iteration, documentation, and verification of use cases.
Work products	*Output* Use Cases
Entry criteria	A requirements analyst has received feedback from elicitation and is ready to develop use cases.
Basic course of events	1. Identify the basic set of use cases. Assign a name and number to each use case. 2. Add a rationale and summary to each use case. Identify which users will interact with each use case, and add them as well. Create a master list of user categories that identifies all of the information known about each kind of user: titles, roles, physical locations, approximate number of users in the category, organizational policies they must adhere to, and anything else that makes someone part of their category. Where possible, add precondition and postcondition states to the use cases. 3. Define the basic course of events and the alternative paths for each use case. Finish adding the precondition and postcondition states. If additional users and use cases are discovered, add them as well (starting with just a name and number, and then adding the other information as in Step 2). 4. Verify each use case, ensuring that all paths make sense and are correct. Go through each step with user representatives to make sure that they accurately represent the way they expect the users to interact with the software. Look for any steps that are awkward for the user that could be made more efficient. Finish all use cases that were added in Step 3.
Exit criteria	The use cases are complete and no additional information has been uncovered, which may lead to additional use cases being developed. If additional use cases have been discovered, return to Step 1 to fill them in.

A requirements analyst defining a set of use cases for this software would start by creating one use case for each feature. Initially, each of these would have a name and a number. The numbering system does not matter, as long as it is unique. (A number such as "UC-1" is sufficient.) The requirements analyst should create a new use case document with a blank template for each of these use cases, filling in the name and number for each of them and proceeding through each of the four steps to create a complete set of use cases.

NOTE

An expanded use case format and a great deal of practical information on developing use cases for software projects can be found in *Use Cases: Requirements in Context* by Daryl Kulak, Eamonn Guiney, and Erin Lavkulich (Addison Wesley, 2000).