Use Cases

Other pages in this series

Introduction

A Use Case is an example of a system being used. It might consist of an initial description of a situation followed by details of the interaction between a user and the system.

Use case modelling helps the development of a system's functional requirements before design work starts by defining the behaviour of the system from the point of view of the user. The simplicity of the use case concept and its use of natural language, allows users to participate in the specification. It helps the analyst to understand the users' needs.

Use cases help

- identify potential functions for incremental prototyping
- demonstrate the need for specific functional requirements
- in the construction of test plans.

However, there are potential problems

- concentration on user interface design
- the development of large numbers of excessively detailed use cases

Use Cases

A use case is

A behaviourally related sequence of interactions performed by an actor in a dialogue with the system to provide some measurable value to the actor. (Jacobson)

This implies:

- Use cases do not just apply to software systems.
- The use case approach defines the behaviour from the point of view of the users (the actors) of the system not the developers.
- The need to provide measurable value ensures that the use case is defined at a reasonably high level, describing a full task (from the user's viewpoint) and so avoids the production of vast numbers of use cases each defining a trivial interaction.
- There should be a single basic path through the logic of a use case. However, in practice it makes sense to consider error branches and to try to group closely related use cases together. For example, when specifying a HTML editor, it would not make sense to define previewing the page with Internet Explorer and previewing the page with Netscape Communicator as two separate use cases.

An actor is anything outside the system that the system interacts with:

- a person
- a group
- a device
- another program.

The identification of actors helps to defined the boundaries of the system.

An individual person may interact with the system in different roles over a period of time. Each role would be regarded as a separate actor.

Developing Use Cases

Use cases express functional requirements in the users' language: this counteracts the tendency to try to work out how the system will work before fully understanding what the system is meant to achieve. Use cases define the scope of a project in terms of specific user requirements. This is in contrast to traditional top-down function decomposition approaches which can produce vague specifications.

What is the problem with having a vague project scope?

For requirements analysis, a use case consists of a use case diagram, a set of descriptions and possibly illustrations of prototype screens.

Development Steps

1. Define the purpose of the system
2. Identify what the system will interface with: the actors.
3. Identify the significant ways in which the system is to be used: the use cases.
4. Define the use cases

Developing the Statement of Purpose

The statement of purpose is a concise statement of the rationale for the system. It answers the question "Why are we building a system?" and must make sense to senior management, including the project champion.

The statement of purpose should be short (around a page maximum) and also include a list of provisos. The reader should note that the statement of purpose would be included as part of the executive summary within a Feasibility Document, which would also contain a risk analysis, constraints and quality attributes.

Write a statement of purpose for a library system. As well as supporting administering loans, the system will be used by borrowers to make queries about their records and the availability of books.

Identifying Actors

An actor is anything outside the system to which the system interfaces; it could be a person, a group of persons, an organisation, another system or a piece of equipment. Sometimes actors can be found simply by reference to the statement of purpose. More often, interviews or JAD sessions are needed to elicit the actors.

An actor can be external or internal to the business. Actors outside the system boundary should be sought first. Examples of actors which are external to the business might be customers, government departments, suppliers, competitors or computer systems in other organisations.

Examples of actors which are internal to the business might be clerks, managers, operators, user departments or computer systems within the organisations.

There is usually little or no choice when identifying external actors which usually represent fixed interfaces, but the idenfication of internal actors may be less obvious.

Identify the actors for a library system. Remember that the actors are entities which interact with the system for a purpose.

Describing a Use Case

After the use cases have been identified, they can be described in more detail.

The emphasis is on the main route through the situation - the basic course: *"the most important course of events giving the best understanding of the use case. Variants of the basic course are described in alternative courses. Normally a use case has one basic course, but several alternative courses."* (Jacobson1992).

A use case is described using "external" language: describing what the actor, using the system, is doing and not describing what the system is doing. The description is often derived from whiteboard sketches or storyboards elicited during JAD sessions.

The description starts with a statement of the objective of the use case. This is followed by the steps that make up the actor-system interaction.

e.g. for the library session:

Case Feature	**Description**	**Example**
Name:	A unique identifier for the use case	Return Book
Intent:	The goal of the use case from the perspective of the primary actor or actors	To restore a borrowed book to the library
Context:	The background circumstances for the use case	The borrower has previously borrowed a book from the library.
Trigger:	The event that starts the use case	Borrower gives a book to the librarian

Basic Course:	The normal flow of the interaction	1. Enter book id 2. Handle fines 1. Calculate fines 2. Report fines to customer 3. Obtain payment 4. Discharge fines
Alternative Courses:	Normal alternatives	1. *A borrower may have no fines* omit step 2 2. *A borrower may have fines on other books* inform borrower of these 3. *A borrower may want to defer payment* tell them that they will not be able to borrow further books
Exceptions	Error alternatives	1. *non-libary book* return to borrower
Issues	(optional)	Assumes computer system is working - what is the fall-back procedure?

Is there a confused system boundary in the above use case?

A use case specification should answer these basic questions:

- Who? (the actors)
- Why? (the goal and/or context)
- When? (the triggering event)
- What? (the normal flow)
- What else? (alternative and/or exceptional flows)

Use case names should be short phrases beginning with an "action" verb. Name the use cases from the perspective of the actor, not the system

Descriptions should use active subject-verb sentences and avoid branching logic.

The web site (http://mem-bers.aol.com/acockburn/papers/OnUseCases.htm) provides some use-case resources, including a use-case template in HTML, MS Word and text formats.

Alternative Routes

The focus will be on the "bread and butter" use cases; that is on the 90% of normal sequences of interactions. Questions will naturally arise concerning errors and exceptions. It would be unwise to model these in their entirety as we will end up with a flood of use cases. A useful strategy here is to simply list such alternative courses at the bottom of the use case description.

Controlling the number and depth of Use Cases

How Many Use Cases?

A manageable number of use cases on an "average" project tends to be between ten and twenty. The number of use cases should not be too large, nor should each use case be allowed to grow too long to be understood.

A project which requires an excessively large number of use cases should be split into a number of projects if each use case is appropriate. However, it might be that the numerous use cases arise because they have been chosen at too low a level.

The definition is a guide for finding the right level of functionality for a use case:

"*A behaviourally related sequence of interactions performed by an actor in a dialogue with the system to provide some measurable value to the actor*".

"*behaviourally related*"	the interactions should as a group be a self contained business task which is an end in itself with no intervening time delays imposed by the business
"the actor"	it must be performed by a single actor in a single place, although it might result in output flows which are sent to other passive actors.

"measurable value"	the use case must achieve some goal for its that the user. If the use case does not have a business-related objective it should be rejected.
"behaviourally related sequence ... to provide some measurable value"	the use case must leave the system in a stable state; it cannot be left half done.

Consider whether the following potential use cases for a library system are appropriate:

- return a book
- scan a membership id from a library card
- stock maintenance
- go to the change password screen
- locate book details

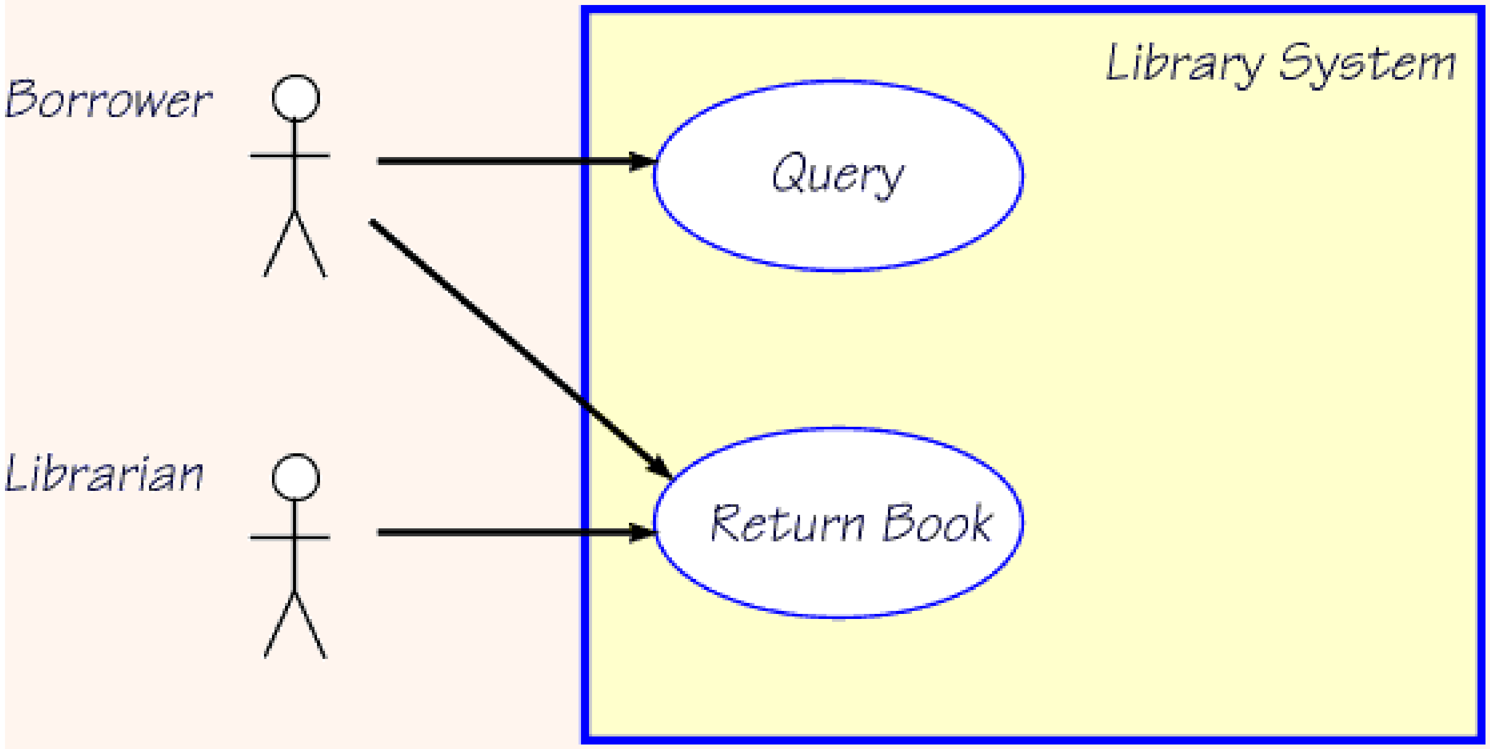
The Human Computer Interface

A dangerous tendancy with use cases is to concentrate on the details of the user interface before the business requirements are fully understood. It is essential to focus on business activities e.g. by breaking the use case into business steps, and thinking of the system as a black box which consumes events and information and provides services.

Prototype screens can usefully be appended to the use case description but these should be functional prototypes to clarify the essence of the interaction in terms of information and events. They should not be usability prototypes, which focus on cosmetics or ergonomics. Neither should they be design prototypes which facilitate discussion of design alternatives.

Diagrammatic Representation of a Use Case Model

The use-case model has a trivial notation. The labelled box that indicates the system boundary, with actors are outside of this box and the use cases (labelled ellipses) inside. Arrows connect the actors to the use cases they are involved with.



Problems with Use Case Modelling

Potential Problem	Action
Excessive detail The use-case specifications are too long.	Concentrate on 80% major requirements.Identify other 20% through incremental development.
Duplication from overlapping use cases	Merge use cases by using abstraction (e.g. merge return book & return cassette to return item)
Too many use cases / knowing when to stop	Apply 80% rule. Concentrate on interactive operational requirements
Use cases not always of equal value	prioritise according to business benefit
Missing information / variety of formats	Introduce standard template for use case descriptions

The system boundary is undefined or inconstant.	Be explicit about the system boundary. Decide whether the system refers to the program or the program and the business staff. e.g. are the librarians actors or part of the system? The actors and use cases appropriate for one system definition are unlikely to be correct for a different one.
The use cases are written from the system's (not the actors') point of view	Concentrate on the business goals.
The actor names are inconsistent.	Use a use-case diagram. Keep a data dictionary.
The actor-to-use case relationships are excessively complex.	Ensure a consistent definition of the system boundary. Focus on the user's business goals (what the user wants to use the system for), not on trivial interaction objectives. Make sure that the choice of use cases hasn't been driven by a desire to design the user interface. To make a one-to-one correspondence between use cases and screen shots, modellers often select use cases that reflect the chunks of user interface rather than user goals. This results in a spider's web of relationships in the use-case model, which have more to do with screen navigation than user goals. Use screen shots only as illustrations of possibilities
The use-case specifications are confusing	Fix the system boundary to ensure consistent scope
The use case doesn't correctly describe authority to access functions.	Define separate use cases for different user goals e.g. the librarian may borrow a book and update book records. Have a common borrow book use case for librarians and borrowers and a separate update records case.
The customer doesn't understand the use cases.	Ensure that the use cases address user goals. Re-write the use cases in problem-domain terminology.

The use cases are never finished.	Fix the system boundary Aim to describe 80% of requirement + use prototyping to capture the rest Don’t confuse use-case specification with user-interface design. The user-interface may change implying a dependency of specification on design. Moreover, the client can't sign off the specification until the user-interface is agreed - which may be well into the project.

Use Case Inspections

Formally review your use-case diagrams and specifications to catch the problems as early as possible and fix them. A similar approach to code inspections can be applied.

A staged review process can be effective.

1. As early as possible, review the use-case diagram, with simple "stubs" for the use-cases (the name and goal or brief description).
2. Formally review the final diagram, with the detailed specifications

Discuss the advantages and disadvantages of this approach over a single inspection of the completed use cases.

Ensuring effective use-case reviews

- Use a review checklist to help detect common use-case problems.
- Present the material to "tell a story." Presenting diagrams rather than text can be a problem. Give a breadth-first overview, followed by an in-depth examination of small groups of related use cases. For instance, the presenter might cover the part of the use-case diagram that is

related to one actor or to a particular business process, then go through those specifications in detail. The review then returns to the diagram to consider another related set of use cases.

- Consider non-traditional review techniques for the initial model review. For example, the "Yellow Sticky Wall Review" allows an informal review of a graphical model. The diagrams are printed and posted on the walls of the review location. The reviewers attach yellow sticky notes containing their comments.