Reading Technique for Use Case Construction

Create use cases to document the functionality that users of the system should be able to perform. This will require listing the functionality that the new system will provide and the external interfaces which are involved. You will have to identify actors (sets of users) who will use the system for specific types of functionality. Remember to include all of the functionality in the system, including special/contingency conditions. Follow the procedure below to generate the use cases, using the questions provided to identify faults in the requirements:

1) Read through the requirements once, finding participants involved.

   a) Identify the participants in the new requirements. Participants are the other systems and the users that interact with the system described in the requirements – that is, they participate in the system functionality by sending messages to the system and/or receiving information and instructions from it. List these participants on Form A. You may use the following questions to help you identify participants:
      - Which user groups use the system to perform a task?
      - Which user groups are needed by the system in order to perform its functions? These functions could be its major functions, or secondary functions such as system maintenance and administration.
      - Which are the external systems that use the system in order to perform a task?
      - Which components or hardware are managed by the system?

   b) Decide which of the participants are actors. Actors represent classes of users which are external to the system and interact with it in characteristic ways. Indicate these actors on the form by checking the box next to the appropriate participants. The following guidelines may be helpful:
      - Actors represent *classes* of users, and may not be the same as the *individual* users of the system.
      - Actors represent the set of external things that need to exchange information (or otherwise interact) with the system.
      - Actors are different from other participants in that their actions are non-deterministic.

   Q1.1 Are multiple terms used to describe the same participant in the requirements?
   Q1.2 Is the description of how the system interacts with a participant inconsistent with the general requirements? Are the requirements unclear or inconsistent about this interaction?
   Q1.3 Have necessary participants been omitted? That is, does the system need to interact with another system, a piece of hardware, or a type of user which is not described?
   Q1.4 Is an external system or a class of "users" described in the requirements which does not actually interact with the system?

2) Read through the requirements a second time, identifying the product functions.

   a) Identify low-level functionality in the requirements. Jot these functionalities down on a piece of scrap paper.

   b) Group functionality together to form higher-level tasks. Each of these high-level tasks will be a use case or scenario that represents, at a high-level, what kind of system functionality will be used by an actor. You may find it helpful to consider the following guidelines when group functionality together to create use cases:
      i) Group functionalities that are involved with the same subsystem.

ii) Identify processing phases – is there a sequence or ordering to the functionality? Group together functionality which is consecutive.
iii) Group functionalities that lead to a definite goal state for the system or for an actor.

c) For each group of functionality, decide how the lower-level pieces are related. On a copy of Form B, use labeled boxes to represent the lower-level pieces in a particular group. Draw arrows between the boxes to identify the flow of system control ("First this functionality happens, then this…") and use branching arrows to signify places where the flow of control could split. Remember to include exception functionality in the use case. Check the appropriate functional requirements to ensure that you have the details of the processing and flow of control correct.

d) For each use case you create, remember to signify the actor who would use the functionality, as well as the action that starts the functionality (e.g. "selecting option 2 from the main menu" might start a use case for deleting records from a database). There are spaces for recording both of these pieces of information on Form B. Finally, give the use case a descriptive name that conveys the functionality it represents. Enter the name both on the associated Form B as well as on Form A.

Q2.1 Are the start conditions for each use case specified at an appropriate level of detail?
Q2.2 Has any functionality that should appear in a use case been omitted?
Q2.3 Has the functionality been described sufficiently so that you understand how the low-level functionalities are related to each other? Does the description allow more than one interpretation as to how the functionalities are related?
Q2.4 Have necessary functionalities been omitted, according to your domain knowledge or the general description?

3) Match the actors to all of the use cases in which they participate. (Remember that if two actors participate in all of the same use cases, they probably represent a single unique actor and should be combined.) Use the third column on Form A to cross-reference the actors with the appropriate use case from the list.

Q3.1 Is it clear from the requirements which actors are involved in which use cases?
Q3.2 Based on the general requirements and your knowledge of the domain, have all of the necessary use cases been specified for each actor?
Q3.3 Are use cases described for an actor which are incompatible with the actor's description in the general requirements?
Q3.4 Have necessary actors been omitted (that is, are there use cases for which no actor is specified)?

Acknowledgements
These guidelines owe much to the following works:

Andersson, Michael, and Johan Bergstrand. (1995) *Formalizing Use Cases with Message Sequence Charts*. Masters Thesis for the Department of Communication Systems at Lund Institute of Technology.

Jacobson, Ivar, *et al.* (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Publishing Company.

And to the work of Curt Zimmerman during his stay with the Empirical Software Engineering Group.