Use Case Analysis

SWENET REQ3 Module

July 2003

Developed with support from the National Science Foundation

Topics

- Use Case Concepts and Terms
- Developing a Use Case Model
- Reviewing a Use Case Model
- Problems with Use Cases

Introduction

- In 1987, Ivar Jacobson [Jacobson 92] introduced the use case concept for modeling the behavior of a system.
- One of the most important, yet difficult, parts of software development is determining what a software system is suppose to do. That is, what are the functional requirements for the software.
- Use case modeling is technique that can help in this difficult task.
  - Use cases are now widely used to model and analyze software requirements.
  - Some believe they should be the primary requirements modeling tool, almost to the exclusion of other techniques.
- Note: This presentation of use cases is introductory in nature. The level of discussion is appropriate for the analysis of small to moderate systems (e.g., systems requiring one-half work year or less to develop), developed by students in an academic setting. It does not include certain detail and complexities that would be needed to develop larger systems, in a commercial or government setting.

Use Case Basics 1

- Use Case (UC): a structured statement of software functionality
  - a restatement of a functional software requirement
  - "the ways in which a user uses a system" [Jacobson 92]
  - "a collection of possible sequences of interactions between the system under discussion and its external actors, related to a particular goal." [Cockburn 97]
- There are lots of different opinions, approaches, techniques, and styles associated with use case modeling.
- Use cases are most helpful in requirements elicitation and analysis.
- Use cases are also helpful with system test planning.

Use Case Basics 2

- A use case represents an external view of interaction with a system.
- Use cases can be used to develop a system users manual.
- Use cases are stories about using a system (that represent the system functional needs).
- Although use cases are often used in OO analysis and are a formal part of the UML, they are not "object-oriented".
  - Use cases are "functional" in nature and work well with functional focused development methodologies.
- Use cases are like "black boxes": they describe what the system must do, not how it will do it.

Use Case Basics 3

- The number and granularity of use cases influences the time and difficulty to understand, maintain, and manage the requirements.
- Some of the advantages cited for use case modeling are:
  - Use cases are good communication tool between developers and users, clients and domain experts.
  - Use cases are a convenient way to capture and categorize the behavior of a system.
  - Use case modeling supports effective requirements elicitation. Use case construction leads very naturally to questions and research about unsupported assumptions and hidden requirements.
  - Use cases can be used to validate the behavior of a system during and after development.

Use Case Format

- *Actor-Action-Object*
  - Actor: the role that a user (or another external entity) plays with respect to the system
  - Action: functionality requested by an actor
  - Object: item acted on by an Actor
- Example Name: *Customer_Deposits_Money*

Goals and Actors

- The use case goal represents the defining purpose of a use case. It represents what is intended to be achieved by initiator of the use case.
  - e.g. a bank customer may have the goal of withdrawing money from her account.
- Actors are not only roles played by people, but organizations, software, and machines. Actors can be classified as follows:
  - primary actor: the principal actor that calls on the system to fulfill the UC goal (e.g., bank customer)
  - supporting actor: an actor that provides a service to the system (e.g., a bank accounts database that is external to the system)
  - offstage actor: an actor that has an interest in the use case, but is not primary or supporting (e.g., a bank examiner)

Use Case Scenarios

- Scenario: formatted description of the steps required for the completion of a Use Case – to achieve the UC goal. One particular story of using the system.
- Main Success Scenario: a scenario that describes the successful completion of the UC goal.
  - It is possible for there to be more than one success scenario.
- Extensions (Alternate Flows): a scenario that represents an alternate or exception to the main success scenario
  - A scenario that does not lead to the UC goal is called a failure scenario.
- A given use case would typically be made up of multiple scenarios.

Developing Use Case Model 1

- Review customer need statement or other information about functional requirements.
  - Interact with users and domain experts to better understand system needs.
- Determine the system boundary and the external entities (develop a Context Diagram).
  - A context diagram depicts the system as a single entity (a circle), the external entities as outside the system (as squares), and interaction between the system and external entities via labeled arrows. (There is an example coming up.)
- List the actors in the system. Candidate actors are:
  - people that use the system
  - other systems that use the system
  - people that install, starts up, or maintain the system

Developing Use Case Model 2

- For each actor, determine the goals (or needs) for the actor.
  - e.g. “Customer needs to establish a Bank Account” or “Customer needs to be able to get information about her account”.
- Define a Use Case for each actor need/goal.
- If a goal is too general, it should be divided into sub goals.
  - e.g. a goal of “customer shall be able to manage her account” could be subdivided as follows:
    - “Customer can get her account balance”
    - “Customer can withdraw money from her account”
    - “Customer can deposit money in her account”

Developing Use Case Model 3

- Do not identify use cases at too low of a level.
  - e.g., “customer makes a deposit” rather than “customer selects deposit menu option”
- Refine the goal/need statement into the use case format.
  - e.g., *Customer_Gets_Balance*
- Fill in a Use Case Scenario template (see next slide), including a step-by-step narrative of how the actor interacts with the system to achieve the UC goal.
  - this sometimes referred to as the “Main Success Scenario”.
- In developing use case descriptions, one needs to role play both as the actors and the system.

Use Case Template 1

- The below template can be used to describe a use case scenario.

Use Case: <Use Case Name>

Primary Actor: <has goals/needs satisfied by the UC>

Goal: <high-level description of use case purpose>

Preconditions: <list of conditions that must be true before Use Case can be executed >

Main Success Scenario:

Step	Actor Action	Step	System Reaction

Post Conditions: <list of conditions that must be true when the Scenario is complete >

Exceptions:
< list of failure conditions that could occur during execution of the scenario and a description of how the system would respond> >

Use Cases Utilized: < list of other use cases used>

Scenario Notes: < description of supporting actors, concurrency of actions, and any additional information such as nonfunctional requirements related to the UC.>

Use Case Template 2

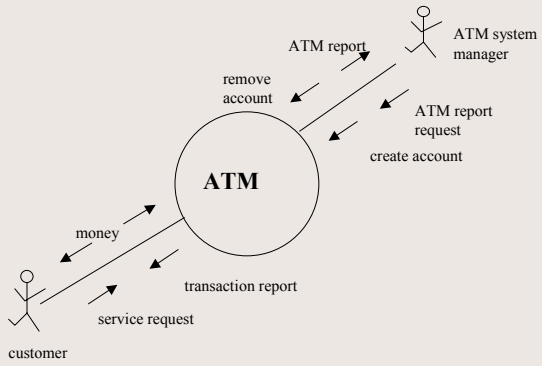
- For the *Specified Action* and *System Reaction* fields, be careful about the the level of detail in the description:
  - Include enough detail so that developers can understand the meaning of the action or reaction. e.g. “Customer requests bank loan” would be better than “Customer requests money”.
  - There is not need, especially at an early stage, to include too much detail. e.g. “Customer enters address” may be better than “Customer enters street address, city name, state name, and zip code”
- The *Exceptions* field would contain the conditions that would required special handling by the system. In use case modeling “exception handling” typically specified with alternate scenarios or scenario extensions. In our discussion, the scenarios are all rather “high-level’ and the details of how exceptions will be handled are left for later, more detailed examination.

Use Case Template 3

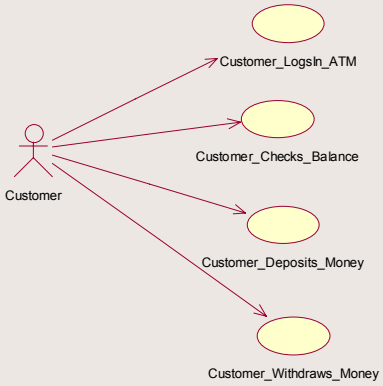
- The *Use Cases Utilized* field indicates structuring of a scenario by using "sub-interaction", where a system may react to some action by initiating another use case.
  - e.g., a use case of *Customer_Login_Account* might initiate a use case of *Manager_Checks_PIN*.
- Use case modeling is part of the evolving nature of requirements elicitation, analysis and modeling; hence, in the early stages of use case development, some items (such as Use Cases Utilized) may not be clear or complete.
- There is no standard UC template; a variety of styles and formats are used.
  - One popular alternative is to depict the scenario steps in a single column, intermixing the actor action and the system reaction.

An Example Problem

Problem Statement: Develop an Automatic Banking System (ABS) that will interact with banking customers, through an ATM, to provide automated banking services (deposit money, withdraw money, provide account information - balance, transaction information, etc.). An ABS manager can create a new account or close-out an existing account.



Use Case Diagram



Are there any missing use cases for the customer actor?

Use Case Description Example

Use Case : Customer_Deposits_Money

Primary Actor: Customer
Goal: A bank customer makes a deposit via an ATM.
Preconditions:
1. Customer has logged into the ATM
2. a service menu is displayed

Main Success Scenario:

Step	Actor Action	Step	System Reaction
1	Customer selects deposit menu item	2	ATM requests deposit amount
3	Customer enters amount of deposit	4	ATM requests the customer to place money in the deposit slot
5	Customer places money in deposit slot	6	ABS adds deposit amount to account balance and files a transaction report about the deposit.
		7	ATM asks if customer wants to request another service
8a	Customer answers NO	9a	ATM prints transaction report and logs the customer off
8b	Customer answers YES	9b	ATM displays services menu

Post Conditions:
1. Customer deposit amount added to balance
2. Customer logged off (if answer NO) or Services Menu displayed (if answer YES)

Exceptions:
1. non-positive deposit amount entered
2. failure to select deposit menu item
3. failure to enter deposit in deposit slot
4. failure to enter YES or NO

Review of Use Case Model 1

- Always carry out a review of the Use Case model.
  - A major error in the UC model (if not detected early) can cause major problems in later stages of development.
  - Where possible, include customers, users, domain experts, and other members of the development team as part of the review group.
  - Make up a checklist for the review and follow it carefully.
    - The following slides will give you some ideas for the checklist.

Review of Use Case Model 2

- Use Case Coverage
  - Every need/goal of the system is covered by at least one UC
  - Each functional requirement has at least one UC designated.
    - That is, all the goals taken together cover all of the functional requirements.
- Use Case Level and Granularity
  - UCs are not at too low/detailed level
  - UCs are not at too high/broad level
  - The level of detail is consistent across the UCs.
  - The number of UCs is appropriate for the size and complexity of the system being developed.

Review of Use Case Model 3

- Use Case Template
  - Each item in the UC Scenario template has been addressed.
  - The *Preconditions* and *Postconditions* represent system states that are appropriate for the achievement of the UC goal
  - The sequence of steps in the main success scenario correspond to expected interaction between the actors and the system.
  - The scenario does not require any non-domain information to be understood.
  - All functional elements concentrate on what the system should do and not how it should be implemented.
  - Any condition that could prevent the success scenario from being completed is listed in the *Exceptions* field.
- Use Case Diagram
  - The UC Diagram includes all actors and UCs, and their correct relationships

Problems with Use Cases 1

- Use in OO development
  - Use Cases are part of the Unified Modeling Language and are widely used in objected-oriented analysis.
  - Since use cases are derived from the functional requirements, there is a danger that when UCs are used as part of OOA, an analyst might drift into a functionally focused approach to development of the system.
- The advantages of use case modeling can be lost in a sea of UC explosion if there is not some control over the number of use cases, their scope and their level of detail.
  - A UC modeler should not lose sight of the UC model's purpose in organizing the description of system behavior and its role as a communication tool between system stakeholders.
  - As in other areas of software development, use abstraction, modularity and information hiding to reduce complexity.

Problems with Use Cases 2

- When are you finished?
  - The true answer might be not until the system is delivered. When systems are developed iteratively, one could see changes and enhancements to the UC model near the end of the development life-cycle.
  - However, if one is interested in some sort of quasi-closure at a given stage of development, the test would be agreement from the stakeholders that the UC model accurately depicts a description of system behavior.
- Scaling up for a larger system.

Summary

- Use cases are an informal, but explicit way to specify the functional requirements for a system.
- The Use Case can be used to support system test planning and the development of user manuals.
- As in any software artifact a formal review of the Use Case model is necessary to insure it is correct and complete.

References

- [Cockburn 97] Cockburn, Alistair,"Structuring Use Cases with Goals", *Journal of Object-Oriented Programming*, Sep-Oct 1997 & Nov-Dec 1997 (http://members.aol.com/acockburn/papers/usecases.htm)
- [Cockburn 01] Cockburn, Alistair, *Writing Effective Use Cases*, Addison-Wesley, 2001.
- [Fowler 92] Fowler, M., *UML Distilled*, 2nd Edition, Addison Wesley, 2001.
- [Jacobson 92] Jacobson, Ivar, *Object-Oriented software engineering: A Use-Driven approach*, Addison-Wesley, 1992.
- [Larman 02] Larman, C., *Applying UML and Patterns*, 2nd Ed., Prentice-Hall, 2002.
- [Texel 97] Texel P.P. & Williams C.B., *Use Cases Combined with Booch, OMT, UML Process and Products*, Prentice Hall, 1997