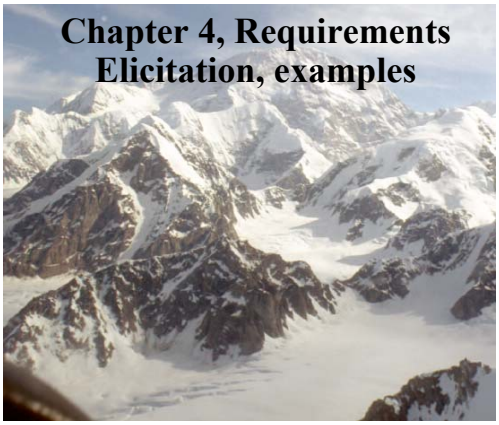
Object-Oriented Software Engineering
Using UML, Patterns, and Java

Chapter 4, Requirements Elicitation, examples



Example: Selection of Software Lifecycle Activities for a specific project

The Hacker knows only one activitity

Implementation

Activities used this lecture

Requirements Elicitation | **Analysis** | **System Design** | **Object Design** | **Implementation** | **Testing**

Each activity produces one or more models

Defining the System Boundary is Often Difficult

What do you see here?

ARENA: The Problem

- The Internet has enabled virtual communities
  - **Groups of people sharing common of interests but who have never met each other in person. Such virtual communities can be short lived (e.g people in a chat room or playing a multi player game) or long lived (e.g., subscribers to a mailing list).**
- Many multi-player computer games now include support for virtual communities.
  - **Players can receive news about game upgrades, new game levels, announce and organize matches, and compare scores.**
- Currently each game company develops such community support in each individual game.
  - **Each company uses a different infrastructure, different concepts, and provides different levels of support.**
- This redundancy and inconsistency leads to problems:
  - **High learning curve for players joining a new community,**
  - **Game companies need to develop the support from scratch**
  - **Advertisers need to contact each individual community separately.**

ARENA: The Objectives

- Provide a generic infrastructure for operating an arena to
  - **Support virtual game communities.**
  - **Register new games**
  - **Register new players**
  - **Organize tournaments**
  - **Keeping track of the players scores.**
- Provide a framework for tournament organizers
  - **to customize the number and sequence of matchers and the accumulation of expert rating points.**
- Provide a framework for game developers
  - **for developing new games, or for adapting existing games into the ARENA framework.**
- Provide an infrastructure for advertisers.

Example: Accident Management System

- What needs to be done to report a “Cat in a Tree” incident?
- What do you need to do if a person reports “Warehouse on Fire?”
- Who is involved in reporting an incident?
- What does the system do, if no police cars are available? If the police car has an accident on the way to the “cat in a tree” incident?
- What do you need to do if the “Cat in the Tree” turns into a “Grandma has fallen from the Ladder”?
- Can the system cope with a simultaneous incident report “Warehouse on Fire?”

Scenario Example: Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.
- Alice enters the address of the building, a brief description of its location (i.e., north west corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appear to be relatively busy. She confirms her input and waits for an acknowledgment.
- John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

Observations about Warehouse on Fire Scenario

- Concrete scenario
  - **Describes a single instance of reporting a fire incident.**
  - **Does not describe all possible situations in which a fire can be reported.**
- Participating actors
  - **Bob, Alice and John**

Next goal, after the scenarios are formulated:

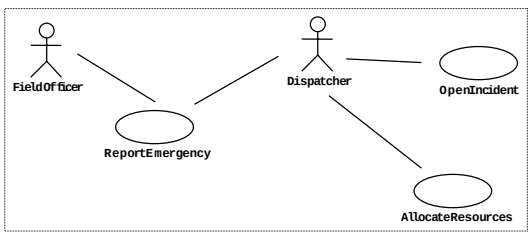
- Find all the use cases in the scenario that specifies all possible instances of how to report a fire
  - **Example: "Report Emergency " in the first paragraph of the scenario is a candidate for a use case**
- Describe each of these use cases in more detail
  - **Participating actors**
  - **Describe the Entry Condition**
  - **Describe the Flow of Events**
  - **Describe the Exit Condition**
  - **Describe Exceptions**
  - **Describe Special Requirements (Constraints, Nonfunctional Requirements**

Use Cases

- A use case is a flow of events in the system, including interaction with actors
- It is initiated by an actor
- Each use case has a name
- Each use case has a termination condition
- Graphical Notation: An oval with the name of the use case

Use Case Model: **The set of all use cases specifying the complete functionality of the system**

Example: Use Case Model for Incident Management



Heuristics: How do I find use cases?

- Select a narrow vertical slice of the system (i.e. one scenario)
  - **Discuss it in detail with the user to understand the user's preferred style of interaction**
- Select a horizontal slice (i.e. many scenarios) to define the scope of the system.
  - **Discuss the scope with the user**
- Use illustrative prototypes (mock-ups) as visual support
- Find out what the user does
  - **Task observation (Good)**
  - **Questionnaires (Bad)**

Use Case Example: ReportEmergency

- Use case name: ReportEmergency
- Participating Actors:
  - **Field Officer (Bob and Alice in the Scenario)**
  - **Dispatcher (John in the Scenario)**
- Exceptions:
  - **The FieldOfficer is notified immediately if the connection between her terminal and the central is lost.**
  - **The Dispatcher is notified immediately if the connection between any logged in FieldOfficer and the central is lost.**
- Flow of Events: **on next slide.**
- Special Requirements:
  - **The FieldOfficer's report is acknowledged within 30 seconds. The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.**

Use Case Example: ReportEmergency Flow of Events

- The **FieldOfficer** activates the "Report Emergency" function of her terminal. FRIEND responds by presenting a form to the officer.
- The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form, at which point, the **Dispatcher** is notified.
- The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the emergency report.
- The FieldOfficer receives the acknowledgment and the selected response.

Another Use Case Example: Allocate a Resource

- Actors:
  - *Field Supervisor:* **This is the official at the emergency site....**
  - *Resource Allocator:* **The Resource Allocator is responsible for the commitment and decommitment of the Resources managed by the FRIEND system. ...**
  - *Dispatcher:* **A Dispatcher enters, updates, and removes Emergency Incidents, Actions, and Requests in the system. The Dispatcher also closes Emergency Incidents.**
  - *Field Officer:* **Reports accidents from the Field**

Another Use Case Example: Allocate a Resource

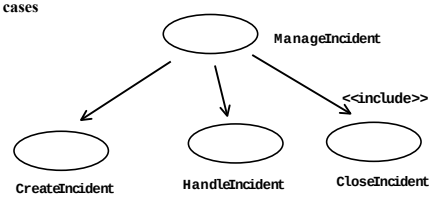
- *Use case name:* AllocateResources
- *Participating Actors:*
  - **Field Officer (Bob and Alice in the Scenario)**
  - **Dispatcher (John in the Scenario)**
  - **Resource Allocator**
  - **Field Supervisor**
- *Entry Condition*
  - **The Resource Allocator has selected an available resource.**
  - **The resource is currently not allocated**
- *Flow of Events*
  - **The Resource Allocator selects an Emergency Incident.**
  - **The Resource is committed to the Emergency Incident.**
- *Exit Condition*
  - **The use case terminates when the resource is committed.**
  - **The selected Resource is now unavailable to any other Emergency Incidents or Resource Requests.**
- *Special Requirements*
  - **The Field Supervisor is responsible for managing the Resources**

Use Case Associations

- A use case model consists of use cases and use case associations
  - **A use case association is a relationship between use cases**
- Important types of use case associations: Include, Extends, Generalization
- Include
  - **A use case uses another use case ("functional decomposition")**
- Extends
  - **A use case extends another use case**
- Generalization
    - **An abstract use case has different specializations**

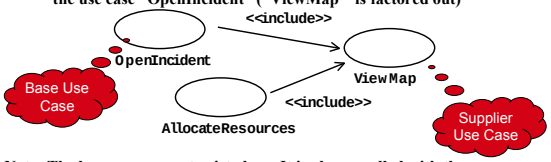
<<Include>>: Functional Decomposition

- Problem:
  - **A function in the original problem statement is too complex to be solvable immediately**
- Solution:
  - **Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into smaller use cases**



<<Include>>: Reuse of Existing Functionality

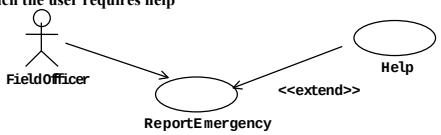
- Problem:
  - **There are already existing functions. How can we *reuse* them?**
- Solution:
  - **The *include association* from a use case A to a use case B indicates that an instance of the use case A performs all the behavior described in the use case B ("A delegates to B")**
- Example:
  - **The use case "ViewMap" describes behavior that can be used by the use case "OpenIncident" ("ViewMap" is factored out)**



Note: The base case cannot exist alone. It is always called with the supplier use case

<Extend>> Association for Use Cases

- Problem:
  - **The functionality in the original problem statement needs to be extended.**
- Solution:
  - **An *extend association* from a use case A to a use case B indicates that use case B is an extension of use case A.**
- Example:
  - **The use case "ReportEmergency" is complete by itself , but can be extended by the use case "Help" for a specific scenario in which the user requires help**



Note: The base use case can be executed without the use case extension in extend associations.

Generalization association in use cases

- Problem:
  - **You have common behavior among use cases and want to factor this out.**
- Solution:
  - **The generalization association among use cases factors out common behavior. The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.**
- Example:
  - **Consider the use case "ValidateUser", responsible for verifying the identity of the user. The customer might require two realizations: "CheckPassword" and "CheckFingerprint"**