

TUM

Use Case Modeling

Prof. Bernd Brügge

Technische Universität München

Lehrstuhl für Angewandte Softwaretechnik

6 May 1998

Defining the System Boundary: What do you see?



System Identification

- ❖ Development of a system is not just done by taking a snapshot of a scene (domain)
- ❖ Definition of the system boundary
 - ◆ What is inside, what is outside?
- ❖ How can we identify the purpose of a system?
- ❖ Requirements Process:
 - ◆ **Requirements Elicitation: Definition of the system in terms understood by the customer**
 - ◆ **Requirements Analysis: Technical specification of the system in terms understood by the developer.**

Requirements Elicitation

- ❖ Very challenging activity
- ❖ Requires collaboration of people with different backgrounds
 - ◆ User with application domain knowledge
 - ◆ Developer with implementation domain knowledge
- ❖ Bridging the gap between user and developer:
 - ◆ **Scenarios: Example of the use of the system in terms of a series of interactions with between the user and the system**
 - ◆ **Use cases: Abstraction that describes a class of scenarios**

Requirements Validation

- ❖ **Critical step in the development process,**
 - ◆ **Usually after requirements engineering or requirements analysis. Also at delivery**
- ❖ **Requirements validation criteria:**
 - ◆ **Correctness:**
 - ◆ **The requirements represent the client's view.**
 - ◆ **Completeness:**
 - ◆ **All possible scenarios through the system are described, including exceptional behavior by the user or the system**
 - ◆ **Consistency:**
 - ◆ **There are functional or nonfunctional requirements that contradict each other**
 - ◆ **Realism:**
 - ◆ **Requirements can be implemented and delivered**

Requirements Validation

❖ Traceability:

- ◆ Each system function can be traced to a corresponding set of functional requirements**

❖ Tool:

◆ RequisitePro from Rational

- ◆ Stores requirements in a repository**
- ◆ Multi-user access**
- ◆ Automatically creates a requirements document from the repository**
- ◆ Provides traceability and change management throughout the project lifecycle**
- ◆ <http://www.rational.com/products/reqpro/docs/datasheet.html>**

Types of Requirements Elicitation

❖ Greenfield Engineering

- ◆ Development starts from scratch, no prior system exists, the requirements are extracted from the end users and the client**
- ◆ Triggered by user needs**

❖ Re-engineering

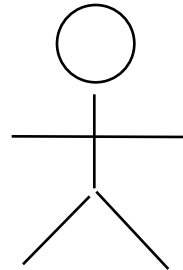
- ◆ Re-design and/or re-implementation of an existing system using newer technology**
- ◆ Triggered by technology enabler**

❖ Interface Engineering

- ◆ Provide the services of an existing system in a new environment**
- ◆ Triggered by technology enabler or new market needs**

Actors

- ❖ Actors constitute everything that is external to the system and that communicates and interacts with the system.
 - ◆ **human users, external hardware and other systems**
- ❖ Actors communicate by sending and receiving stimuli to and from the system. Each actor has a name.
- ❖ Graphical Notation: A stick figure with the name of the actor



Field Officer

Scenarios

- ❖ **“A narrative description of what people do and experience as they try to make use of computer systems and applications” [M. Carrol, Scenario-based Design, Wiley, 1995]**
- ❖ **A concrete, focused, informal description of a single feature of the system used by a single actor.**
- ❖ **Scenarios can have many different uses during the software lifecycle**

Types of Scenarios

- ❖ **As-is scenario:**
 - ◆ **Used in describing a current situation. Usually used during re-engineering. The user describes the system.**

- ❖ **Visionary scenario:**
 - ◆ **Used to describe a future system. Usually described in greenfield engineering or reengineering.**
 - ◆ **Can often not be done by the user or developer alone**

- ❖ **Evaluation scenario:**
 - ◆ **User tasks against which the system is to be evaluated**

- ❖ **Training scenario:**
 - ◆ **Step by step instructions designed to guide a novice user through a system**

How do we find scenarios?

- ❖ Don't expect the client to be verbal if the system does not exist (greenfield engineering)
- ❖ Don't wait for information even if the system exists
- ❖ Engage in a dialectic approach (evolutionary, incremental)
 - ◆ You help the client to formulate the requirements
 - ◆ The client helps you to understand the requirements
 - ◆ The requirements evolve while the scenarios are being developed

Heuristics for finding Scenarios

- ❖ **Ask yourself or the client the following questions:**
 - ◆ **What are the primary tasks that the system needs to perform?**
 - ◆ **What data will the actor create, store, change, remove or add in the system?**
 - ◆ **What external changes does the system need to know about?**
 - ◆ **What changes or events will the actor of the system need to be informed about?**

- ❖ **Insist on task observation if the system already exists (interface engineering or reengineering)**
 - ◆ **Ask to speak to the end user, not just to the software contractor**
 - ◆ **Expect resistance and try to overcome it**

Example: Accident Management System5/6/98 **13:00**

- ❖ 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, activates the “Report Emergency” function from her laptop.
- ❖ 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 `creates` 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
 - ◆ **Describe the Entry Condition**
 - ◆ **Describe the Flow of Events**
 - ◆ **Describe the Exit Condition**
 - ◆ **Describe Exceptions**
 - ◆ **Describe Special Requirements (Constraints, Nonfunctional Requirements)**

Heuristics

- ❖ **First name the use case**
 - ◆ **Use case name: ReportEmergency**

- ❖ **Then find the actors**
 - ◆ **Generalize the concrete names (“Bob”) to participating actors (“Field officer”)**
 - ◆ **Participating Actors:**
 - ◆ **Field Officer (Bob and Alice in the Scenario)**
 - ◆ **Dispatcher (John in the Scenario)**
 - ◆

- ❖ **Then concentrate on the flow of events**
 - ◆ **Use informal natural language**

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.

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.**
- ❖ 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.**

How to Specify a Use Case (Summary)

- ❖ Name of Use Case
- ❖ Actors (Description of Actors involved in use case)
- ❖ Entry condition (“This use case starts when...”)
- ❖ Flow of Events (Free form, informal natural language)
- ❖ Exit condition (“This use cases terminates when...”)
- ❖ Exceptions (Describe what happens if things go wrong)
- ❖ Special Requirements (Nonfunctional Requirements, Constraints)

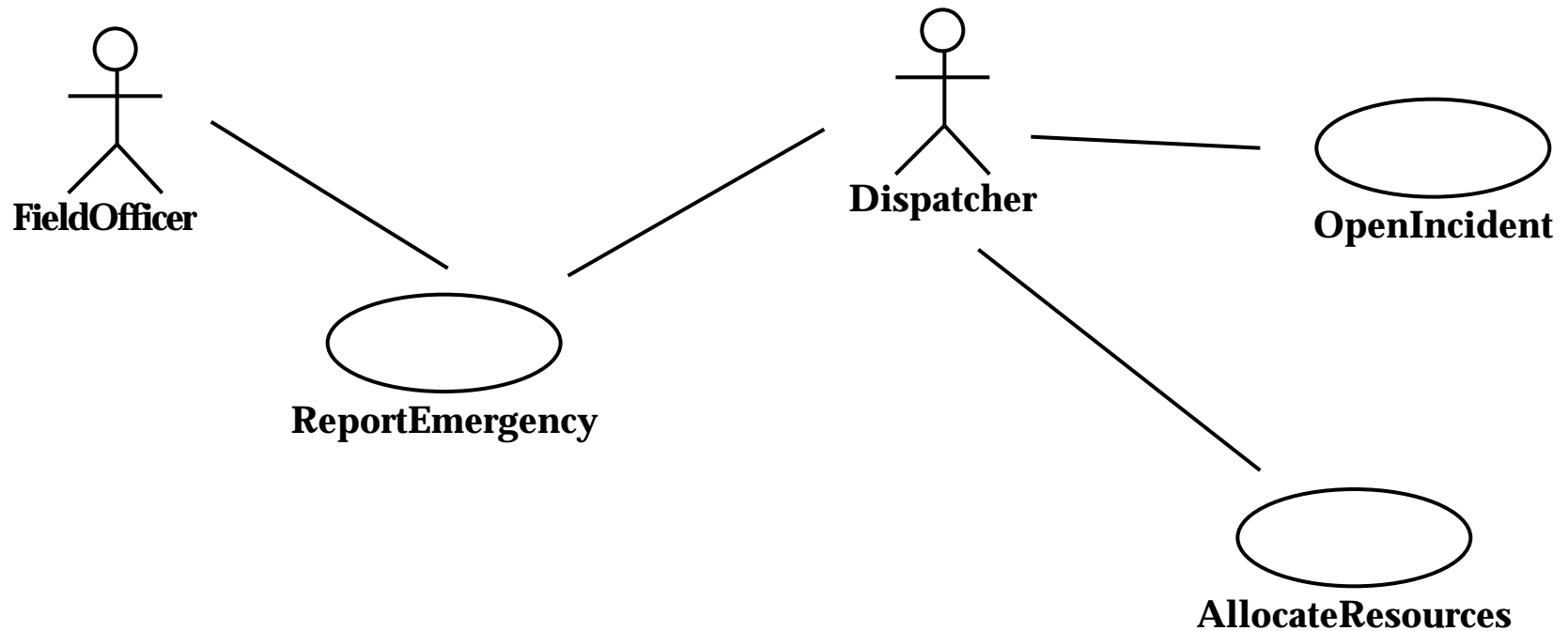
Use Cases

- ❖ 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



Use Case Associations

- ❖ **Use Case Association = Relationship between use cases**
- ❖ **Important types:**
 - ◆ **Consists of**
 - ◆ **A use cases consists of other use cases (“functional decomposition”)**
 - ◆ **Extends**
 - ◆ **A use case extends another use case**
 - ◆ **Uses**
 - ◆ **A use case uses another use case**

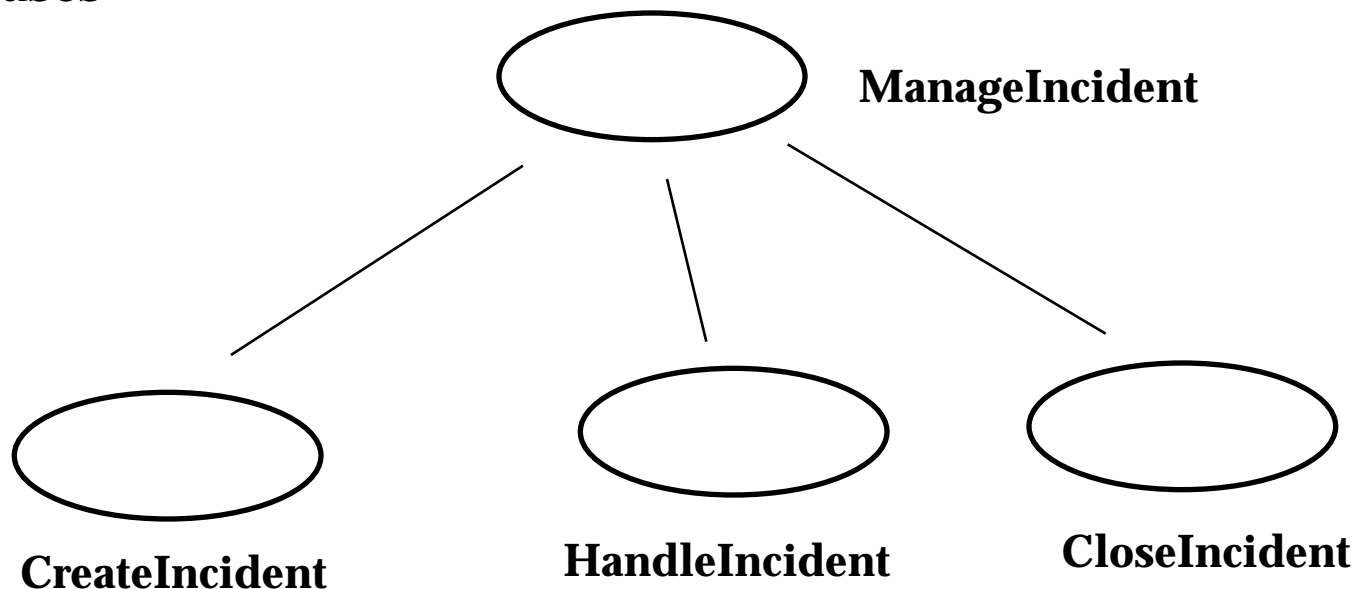
“Consists of” Association

❖ Problem:

- ♦ A function in the original problem statement is too complex to be solvable

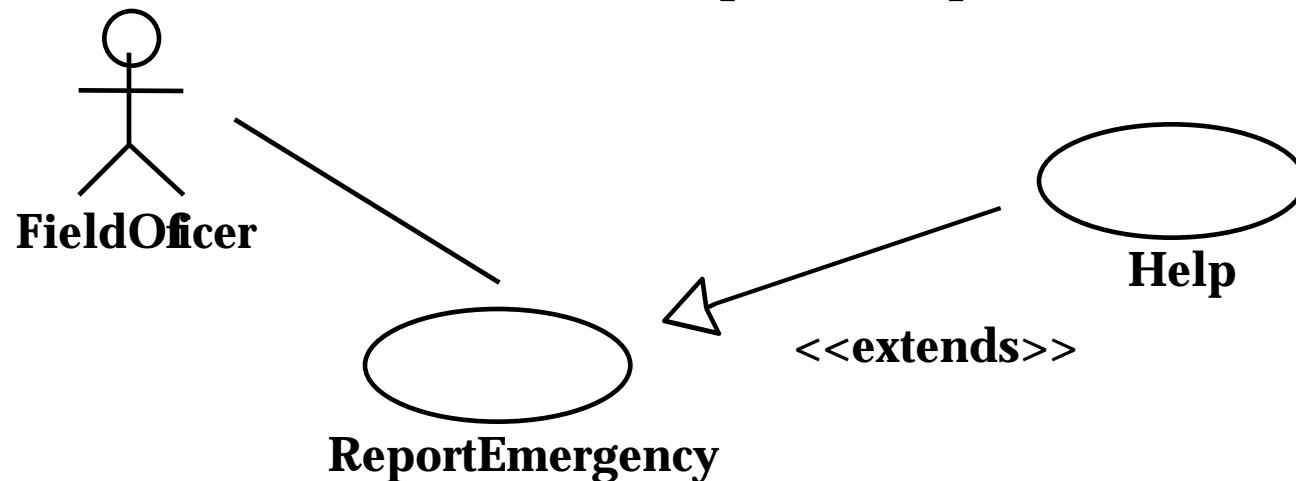
❖ Solution:

- ♦ Describe the function as the aggregation of a set of simpler functions. The associated use case is refined into smaller use cases



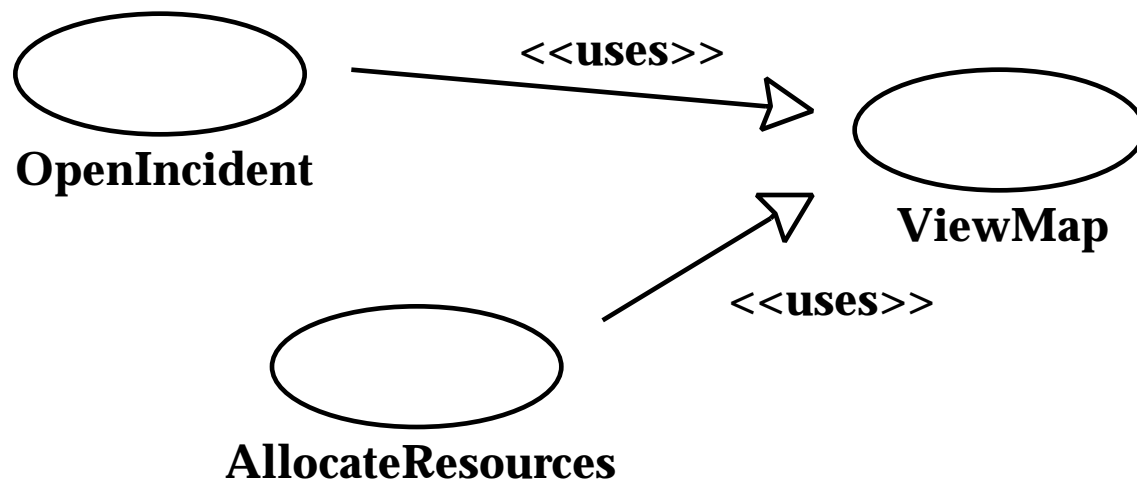
“Extends” Association for Use Cases

- ❖ Problem:
 - ♦ The functionality in the original problem statement needs to be extended.
- ❖ Solution:
 - ♦ An extends 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



“Uses” Association for Use Cases

- ❖ Problem:
 - ♦ There are already existing functions. How can we *reuse* them?
- ❖ Solution:
 - ♦ The uses association from a use case A to a use case B indicates that an instance of the use case A can perform all behavior described for the use case B
- ❖ Example:
 - ♦ The use case “ViewMap” describes behavior that can be used by the use case “OpenIncident” (“ViewMap” is factored out)



Use Case Example: Allocate a Resource

❖ *From FRIEND*

- ◆ *First Responder Interactive Emergency Navigation Database***
- ◆ *15-413 Fall 93***

❖ 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.**

Use Case Example: Allocate a Resource

❖ Entry Condition

- ♦ **The use case starts after 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**

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 mock-ups as visual support**
- ❖ **Find out what the user does**
 - ◆ **Task observation (Good)**
 - ◆ **Questionnaires (Bad)**

Is there Life after Scenarios and Use Cases?

- ❖ **Functional Decomposition says no**

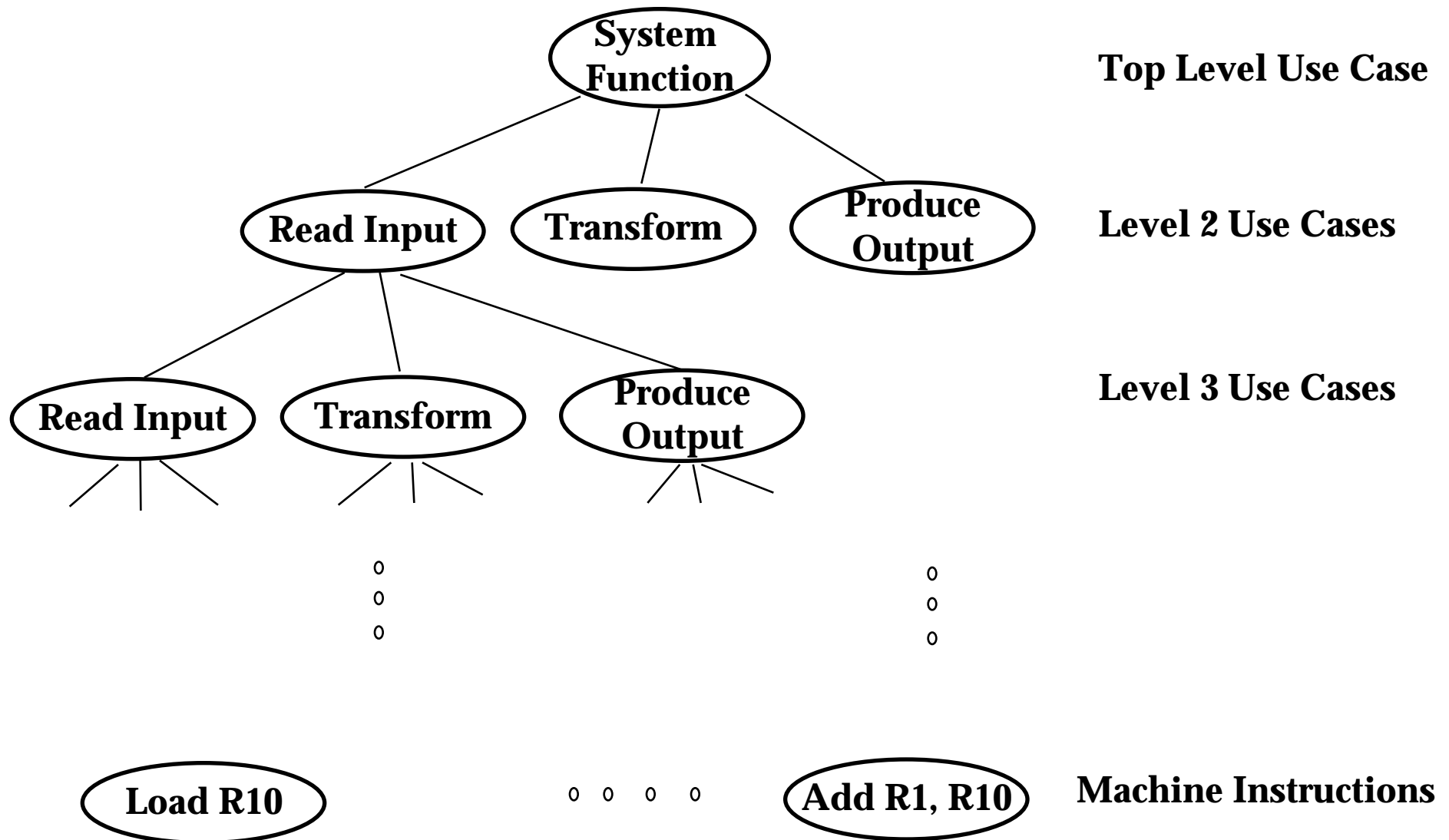
- ❖ **Every use case is refined into a set of lower level use cases. Either**
 - ◆ **The use cases consists of lower level use cases**

or

 - ◆ **The use case is extended by another use case**

- ❖ **This refinement is repeatedly done until the lowest level use cases are machine instructions that can be executed by the target machine**

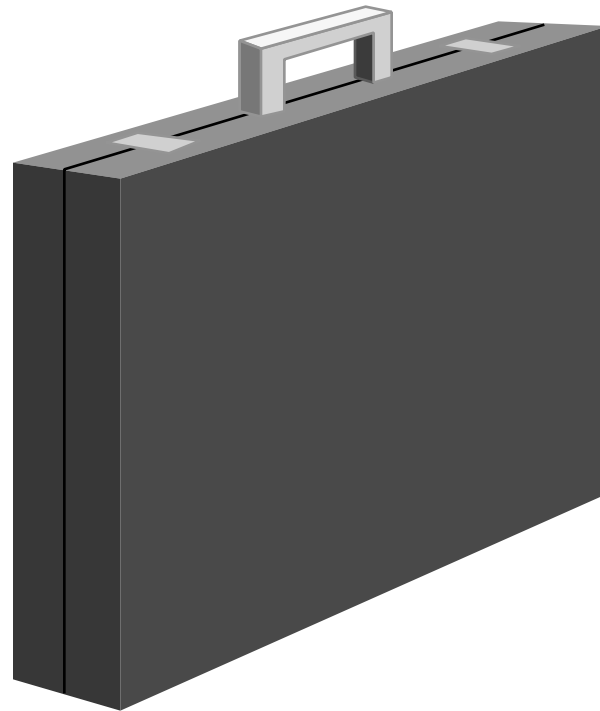
Functional Decomposition



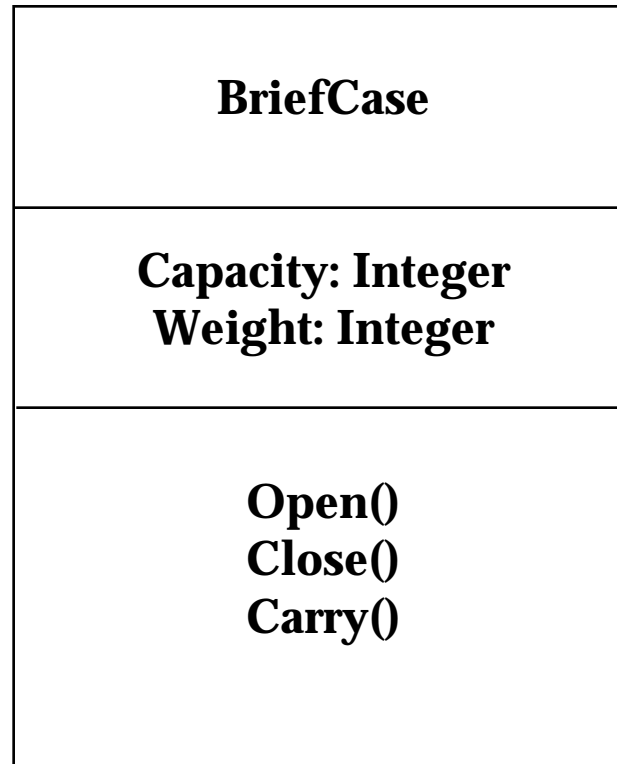
Problems with Functional Decomposition

- ❖ **High cost of recompilation**
 - ◆ **Adding a new device usually requires compilation of every file that uses the device**
- ❖ **Leads to code that is hard to maintain**
- ❖ **The better way is to start with functional decomposition and then to find objects**
 - ◆ **Object identification**
 - ◆ **Sequence Diagrams**

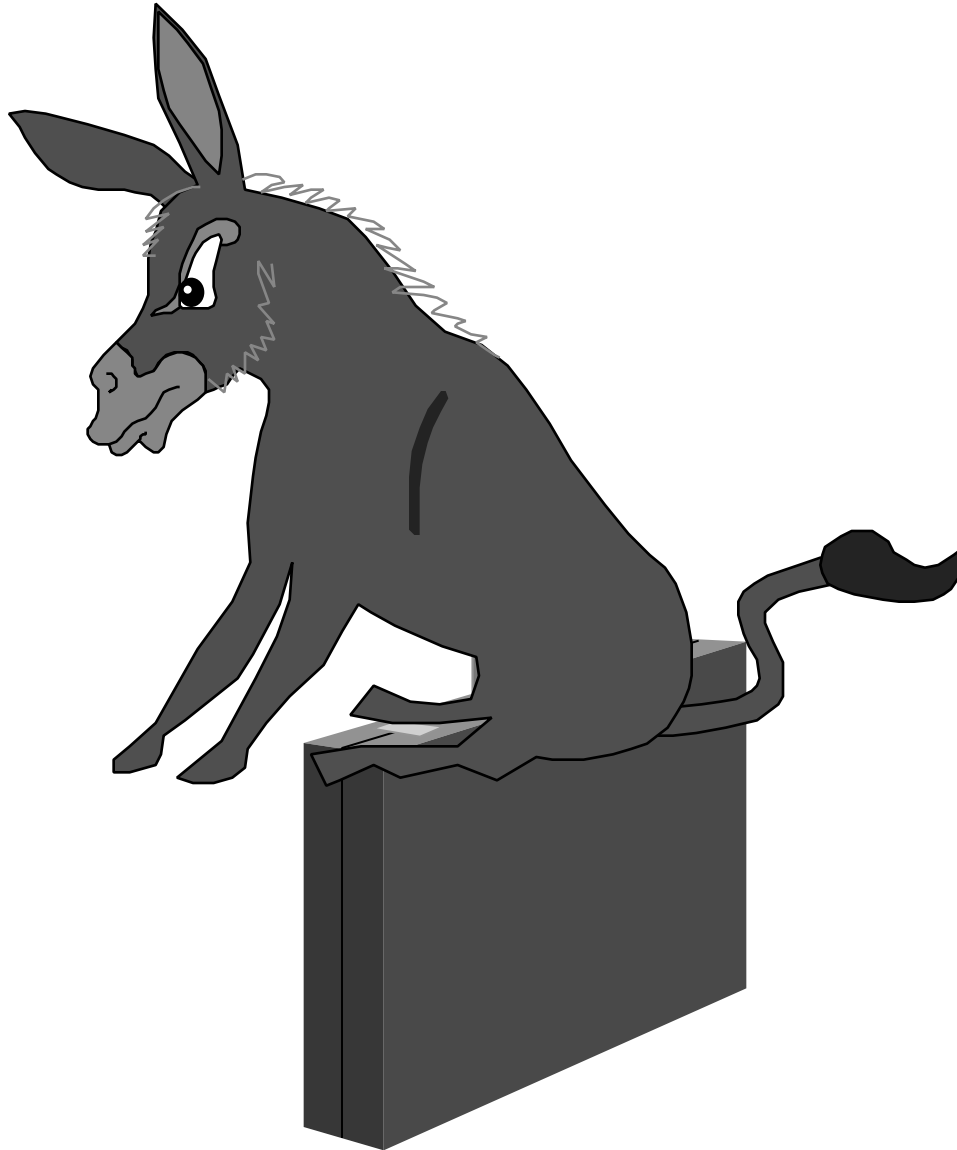
What is this Thing?



Modeling a Briefcase



A new Use Case for a Briefcase

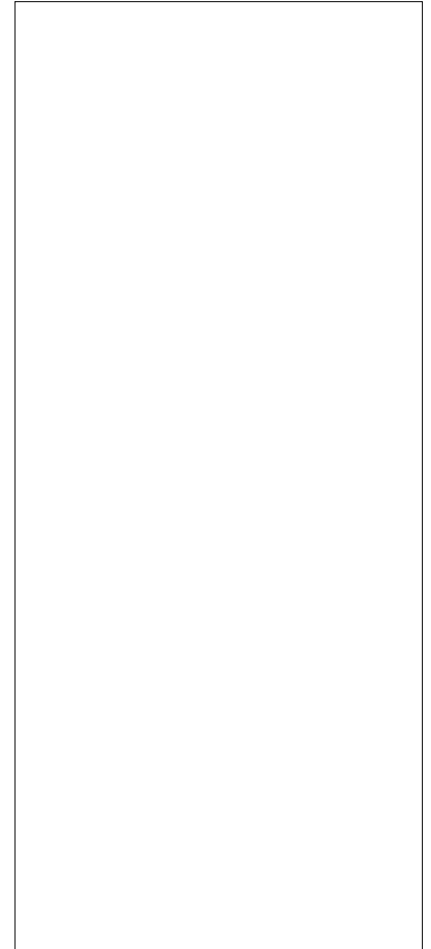
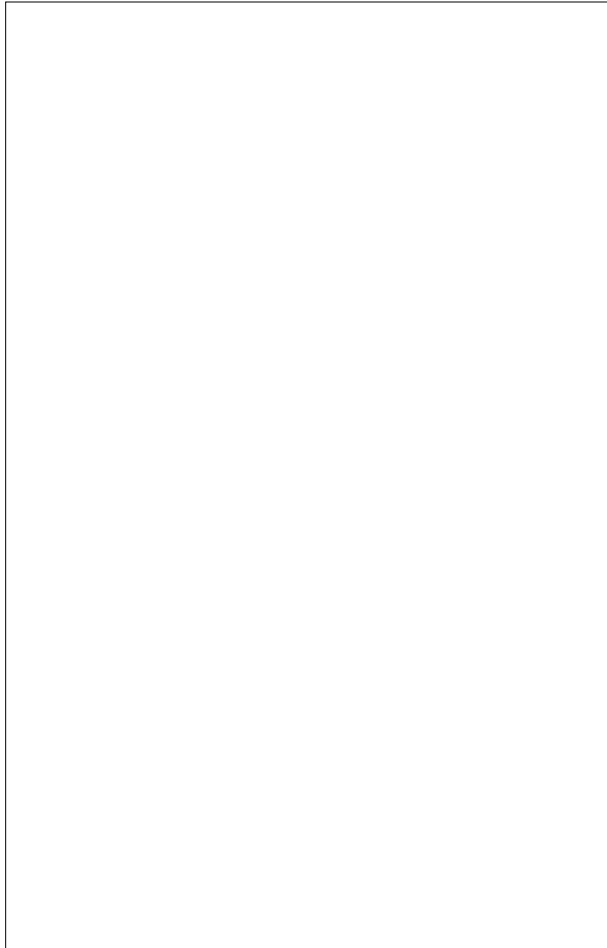


BriefCase
Capacity: Integer Weight: Integer
Open() Close() Carry() SitOnIt()

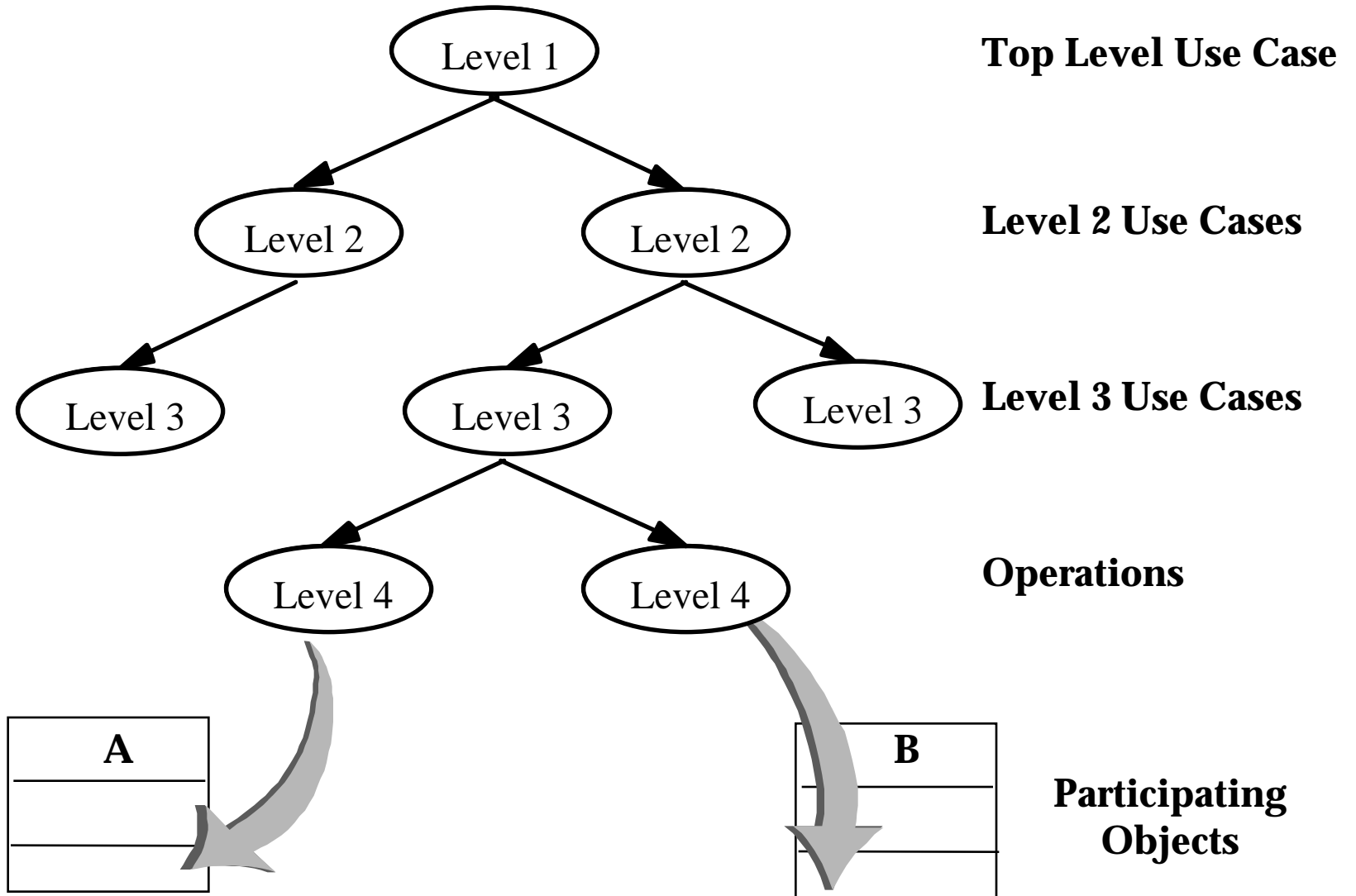
Questions

- ❖ Why did we model the thing as “Briefcase”?
- ❖ Why did we not model it as a chair?
- ❖ What do we do if the SitOnIt() operation is the most frequently used operation?
- ❖ The briefcase is only used for sitting on it during video conferences. It is never opened nor closed. Is it a “Chair” or a “Briefcase”?
- ❖ How can we live with our mistake?

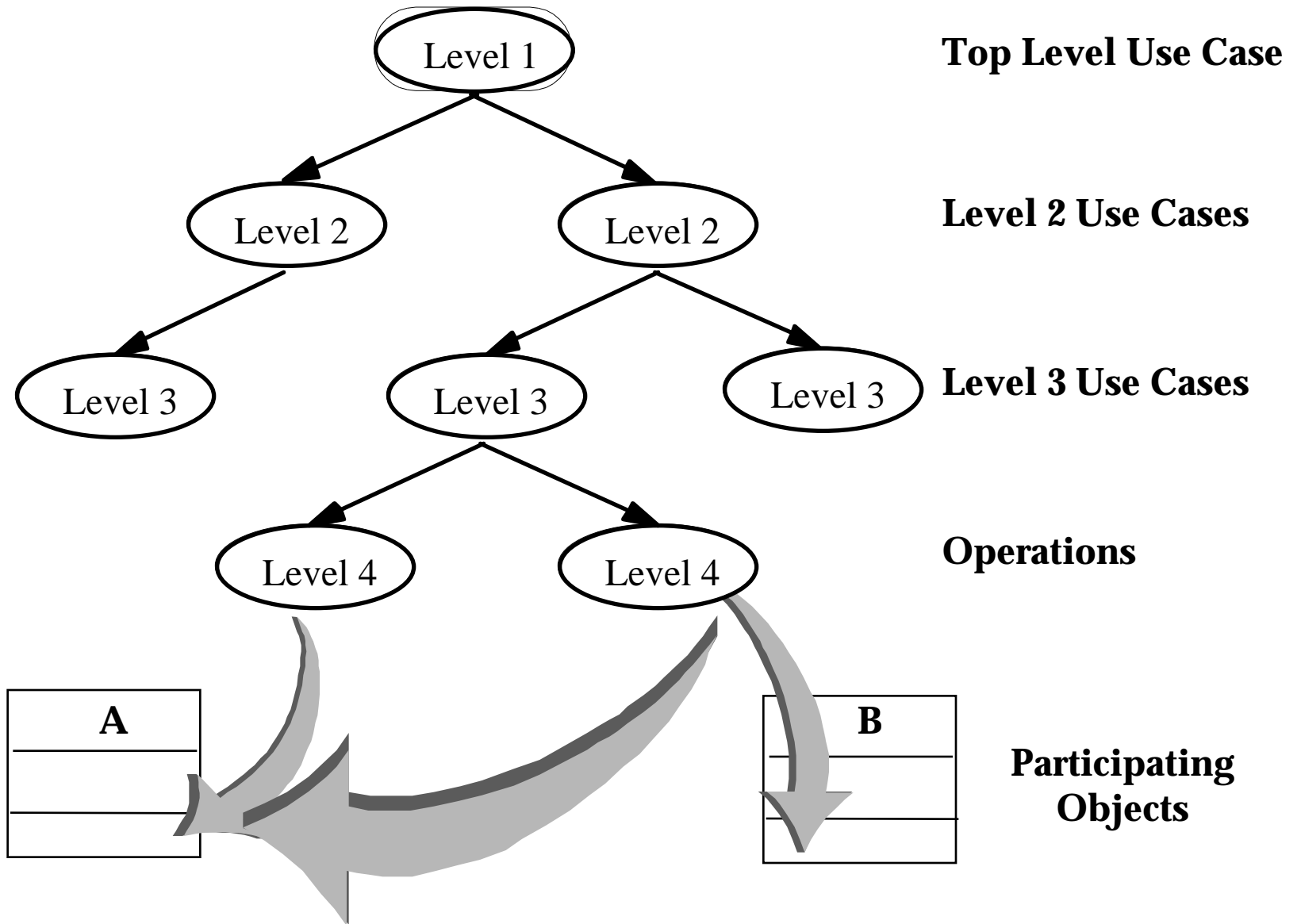
What is this?



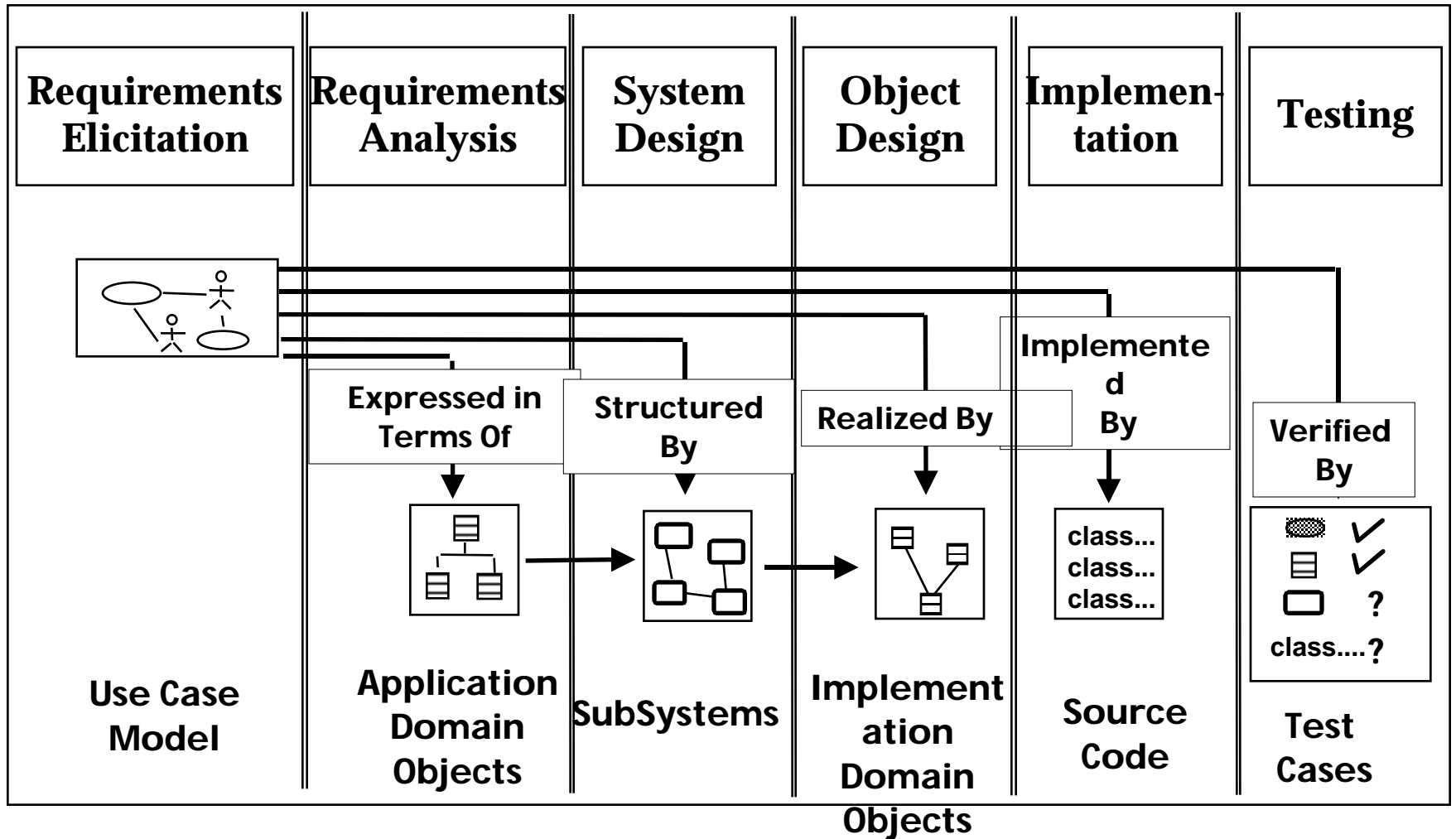
From Use Cases to Objects



Use Cases are used by more than one object



Use Cases and Lifecycle Activities



Summary

- ❖ **Requirements Elicitation:**
 - ◆ **Greenfield Engineering, Reengineering, Interface Engineering**
- ❖ **Scenarios:**
 - ◆ **Great way to establish communication with client**
 - ◆ **As-Is Scenarios, Visionary scenarios, Evaluation scenarios Training scenarios**
- ❖ **Use cases: Abstraction of scenarios**
- ❖ **Pure functional decomposition is bad:**
 - ◆ **Leads to unmaintainable code**
- ❖ **Pure object identification is bad:**
 - ◆ **May lead to wrong objects, wrong attributes, wrong methods**
- ❖ **The key to successful analysis:**
 - ◆ **Start with use cases and then find the participating objects**
 - ◆ **If somebody asks “What is this?”, do not answer right away. Return the question or observe: “What is it used for?”**