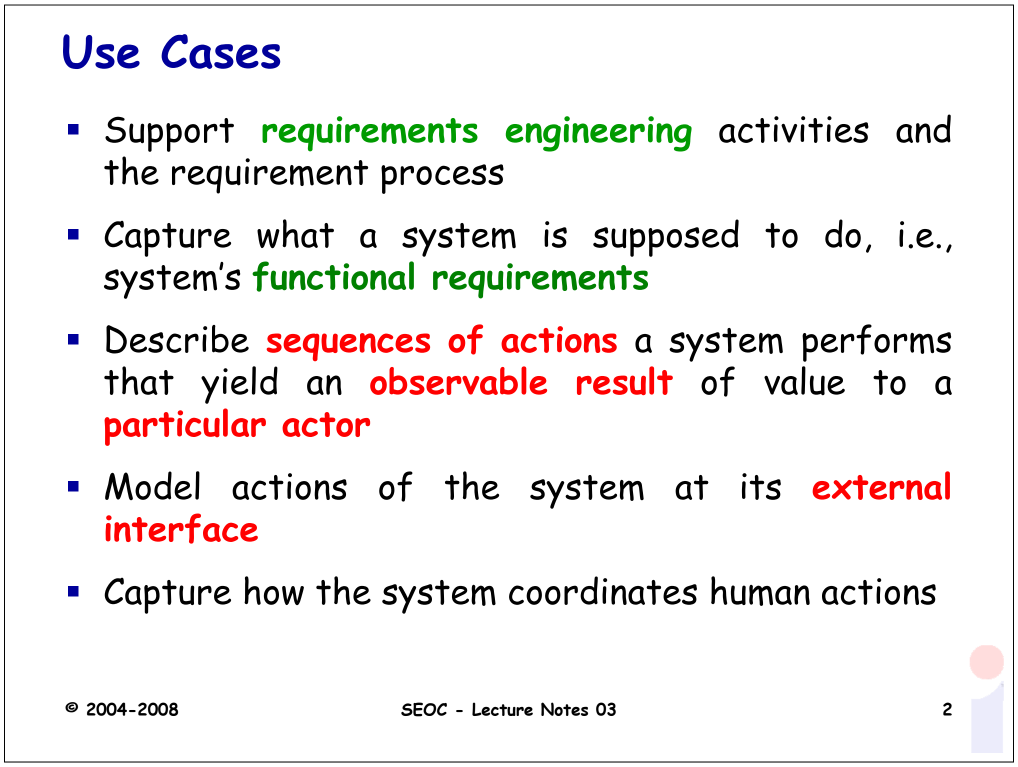
Use Cases

Massimo Felici

IF 3.46 0131 650 5899

mfelici@inf.ed.ac.uk



Use Cases

- Support **requirements engineering** activities and the requirement process
- Capture what a system is supposed to do, i.e., system's **functional requirements**
- Describe **sequences of actions** a system performs that yield an **observable result** of value to a **particular actor**
- Model actions of the system at its **external interface**
- Capture how the system coordinates human actions

Use cases provide a high level view of the system. They capture to a certain extent system structures. Use case describe **sequences of actions** a **system performs** that yield **an observable result of value** to **a particular actor**.

• **Sequence of actions**: set of functions, algorithmic procedures, internal processes, etc.

• **System performs**: system functionalities

• **An observable result of value** to a user

• **A particular actor**: individual or device

Use Cases modeling is an effective means of communicating with users and other stakeholders about the system and what is intended to do.

Use Cases support a relationship with scenarios and relevant activities (e.g., testing).

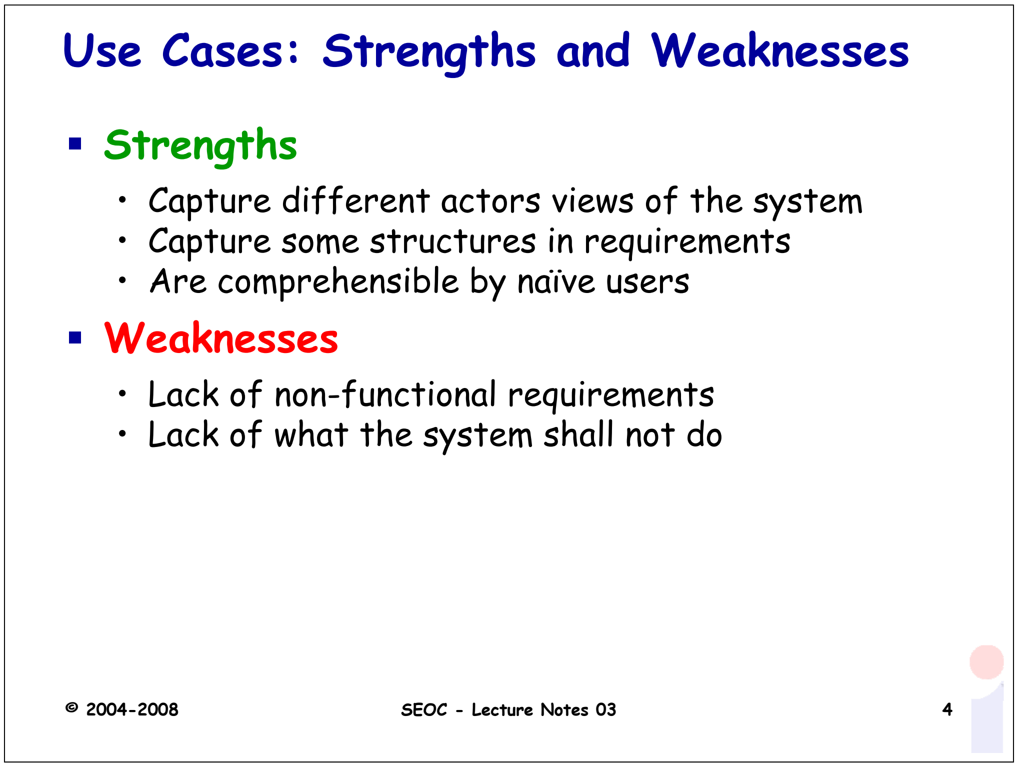
Readings

• **UML course textbook**

  • Chapter 3 on Use Cases

The Benefits of Use Cases

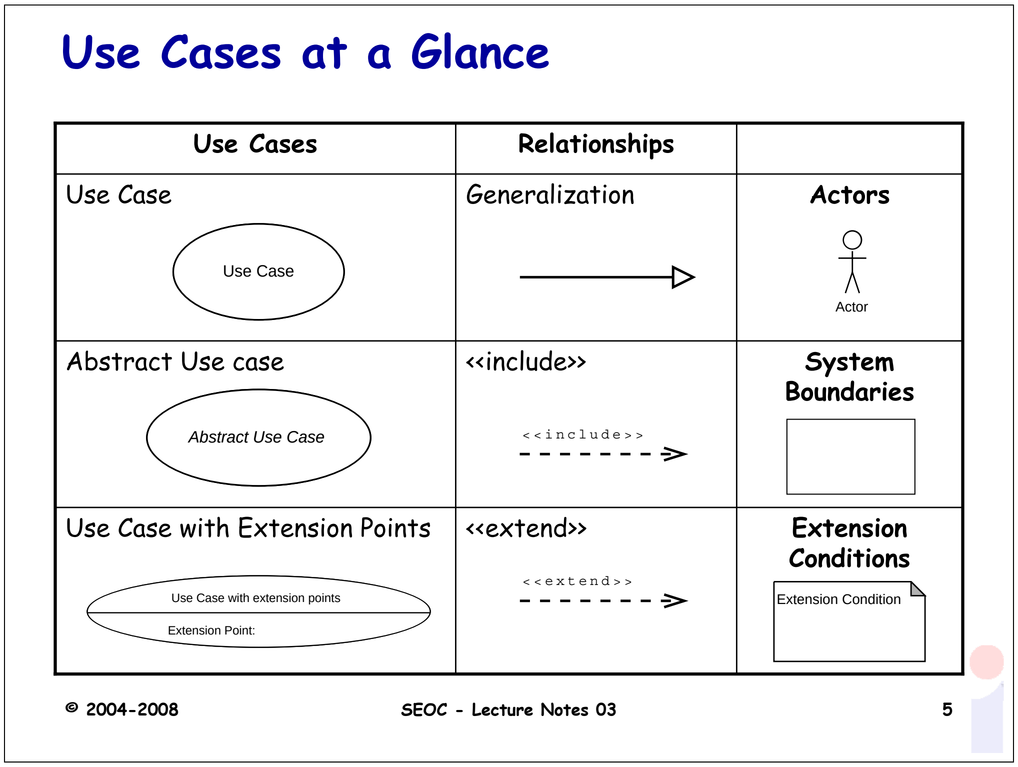
- Relatively **easy** to write and easy to read
- Comprehensible by **users**
- Engage the **users** in the requirements process
- Force developers to think through the design of a system from a **user viewpoint**
- Identify a **context** for the requirements of the system
- Critical tool in the **design**, **implementation**, **analysis** and **testing process**
- Rapid **change** allows exploratory approach
- Serve as inputs to the user documentation



Use Cases: Strengths and Weaknesses

- **Strengths**
  - Capture different actors views of the system
  - Capture some structures in requirements
  - Are comprehensible by naïve users
- **Weaknesses**
  - Lack of non-functional requirements
  - Lack of what the system shall not do

Use Cases help in **structuring systems.** For example, the scheduler and patient more or less form a sub-system – look at delegating appointment management to a single component or sub-system.



Use Cases at a Glance

Use Cases	Relationships	
Use Case Use Case	Generalization	**Actors** Actor
Abstract Use case *Abstract Use Case*	<<include>> <<include>>	**System Boundaries**
Use Case with Extension Points Use Case with extension points Extension Point:	<<extend>> <<extend>>	**Extension Conditions** Extension Condition

Anatomy of Use Cases: Basic Diagrams

• **Actors** are represented as stick figures

• **Use Cases** as ellipses

• **Lines** represent associations between these things

• **Use Case diagrams** show who is involved with what.

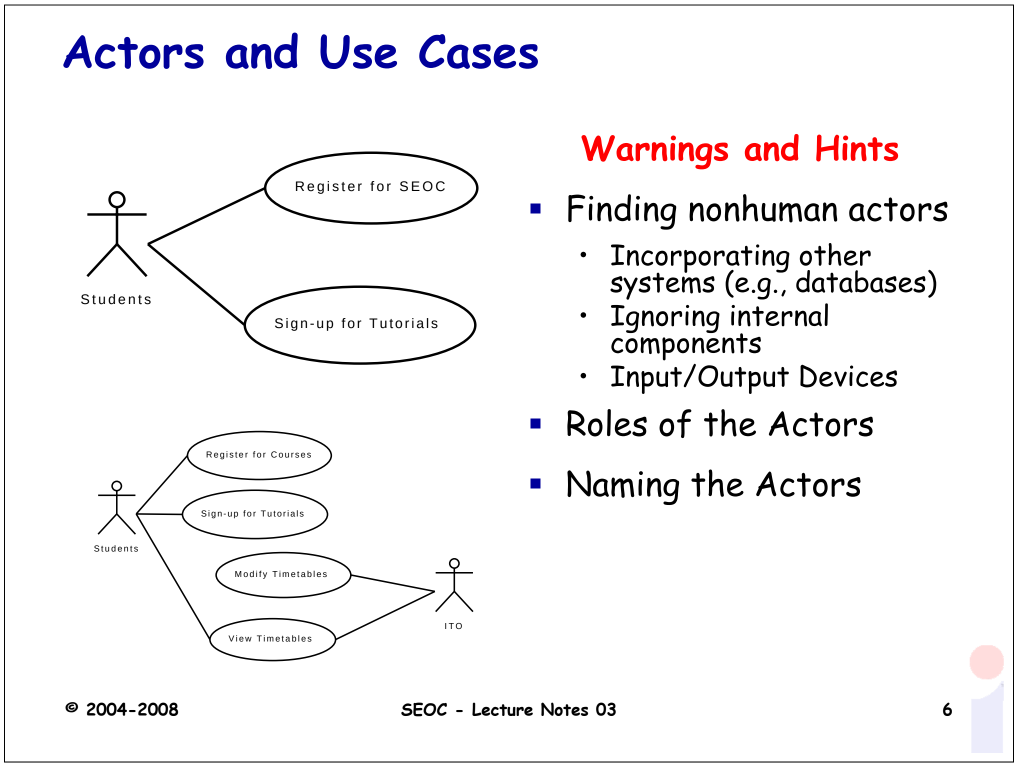
Use Cases Basics

• **Actors:** An Actor is **external** to a system, **interacts** with the system, may be a **human user or another system**, and has a **goals** and **responsibilities** to satisfy in interacting with the system.

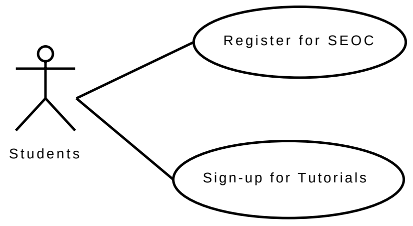
• **Use Cases:** identify **functional requirements**, which are described as a sequence of steps describe **actions** performed by a system capture **interactions** between the system and actors.

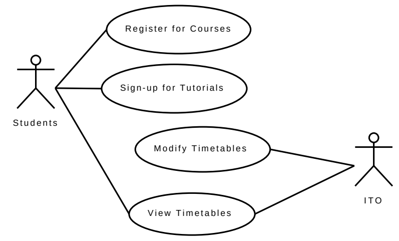
• **Relationships:** Actors are connected to the use cases with which they interact by a line which represents a relationship between the actors and the use cases.

• **System Boundaries:** Identify an implicit separation between actors (external to the system) and use cases (internal to the system)



Actors and Use Cases





Warnings and Hints

- Finding nonhuman actors
  - Incorporating other systems (e.g., databases)
  - Ignoring internal components
  - Input/Output Devices
- Roles of the Actors
- Naming the Actors

Despite the simplicity of use cases, it is difficult to identify the involved actors and use cases. One of the common issue is the completeness of the involved actors and relevant use cases. This is often due to a lack of understanding of the system and its requirements. Hence, use cases help to discuss an high-level structured view of the system, its functionality and the relevant actors around the system. Another common difficulty is the identification of the trade-offs between generality and specificity. On the one hand, general use cases could lack information about the system functionalities. On the other hand, detailed use cases could try to over specify some design aspects. Here are some general hints:

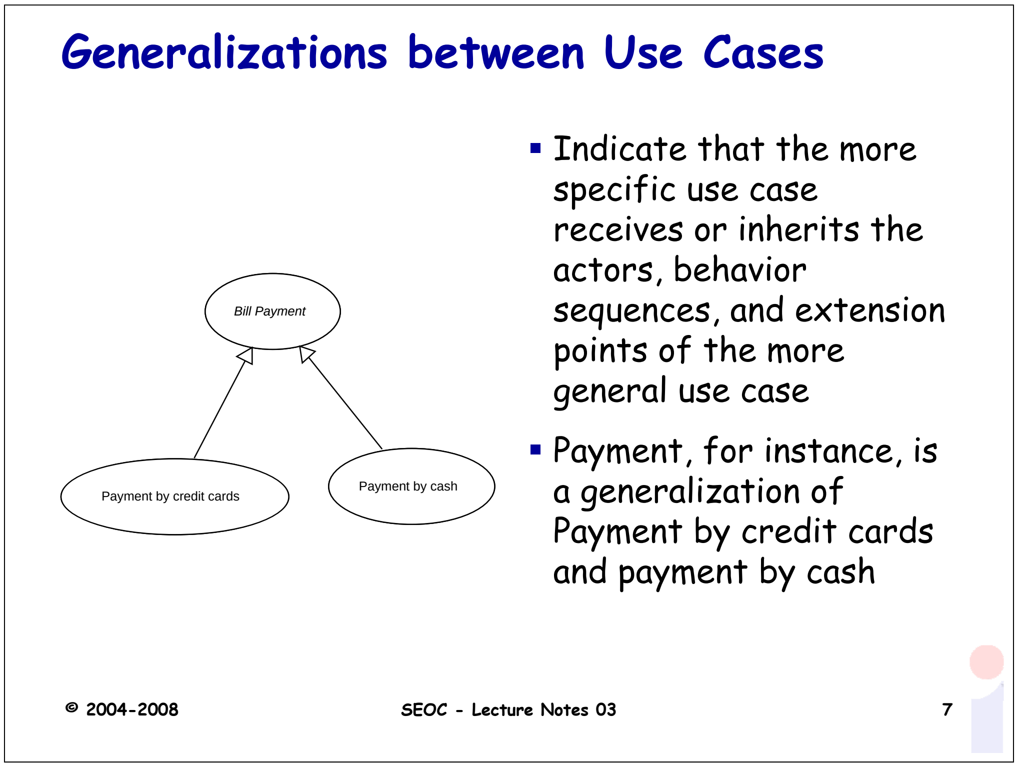
• take care to identify generic actors who do a particular task, or cover a particular role with respect to the system

• Do not get confused with job titles

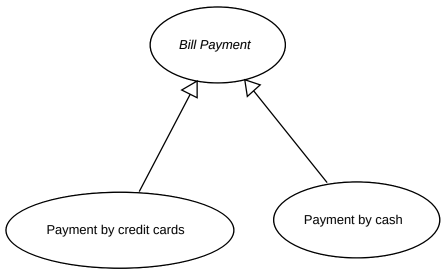
• use case diagrams should not be too complex

• aim for reasonably generic use cases

• try not be too detailed at first.



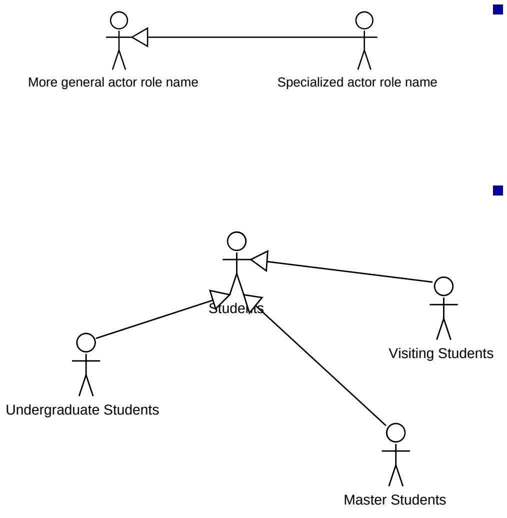
Generalizations between Use Cases



- Indicate that the more specific use case receives or inherits the actors, behavior sequences, and extension points of the more general use case
- Payment, for instance, is a generalization of Payment by credit cards and payment by cash

Generalization is often implemented by **inheritance**.

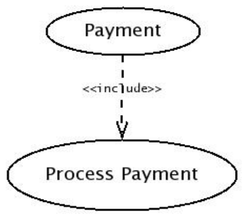
Generalization between Actors

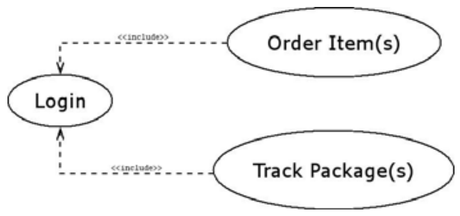


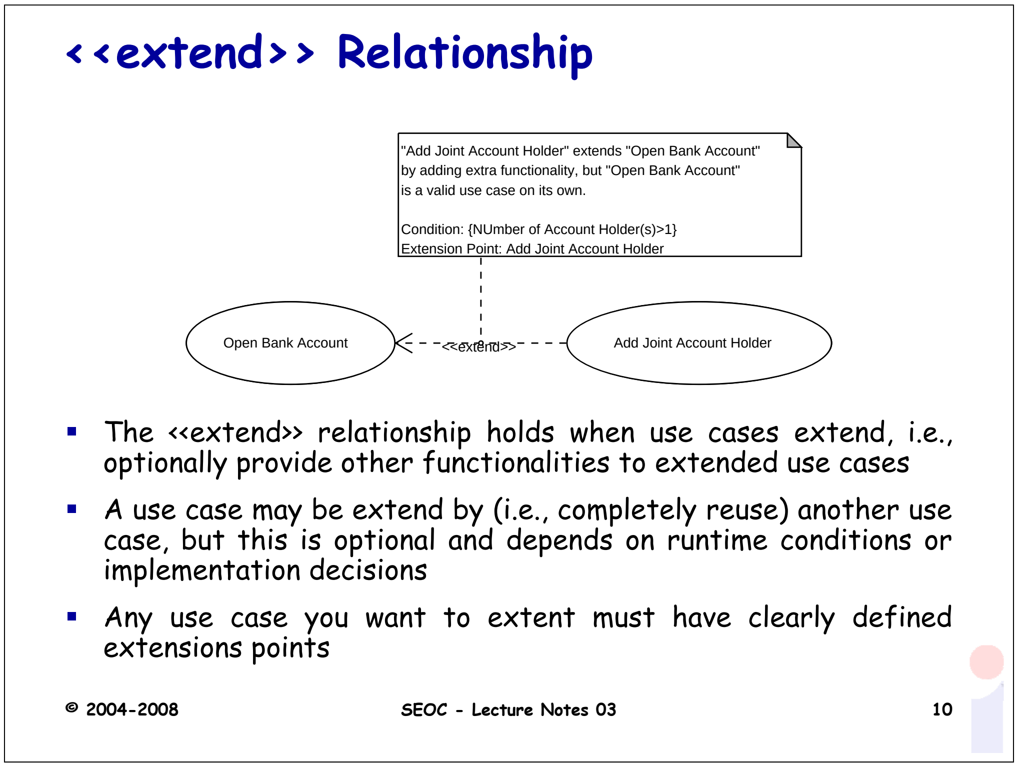
- Actors may be similar in how they use the system (e.g., project and system managers)
- An Actor generalization indicates that instances of the more specific actor may be substituted for instances of the more general actor

<<include>> Relationship

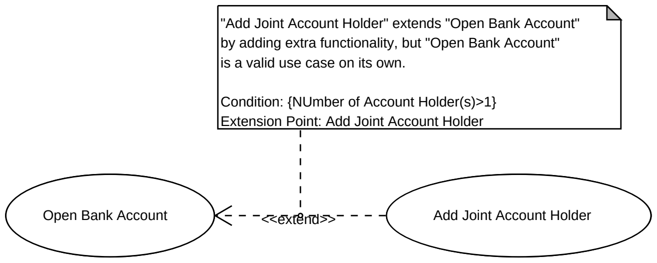
- The <<include>> relationship holds when one use case is included in others
- The <<include>> relationship declares that a use case reuses another one being included
- The included use case is (typically not complete on its own) a required part of other use cases







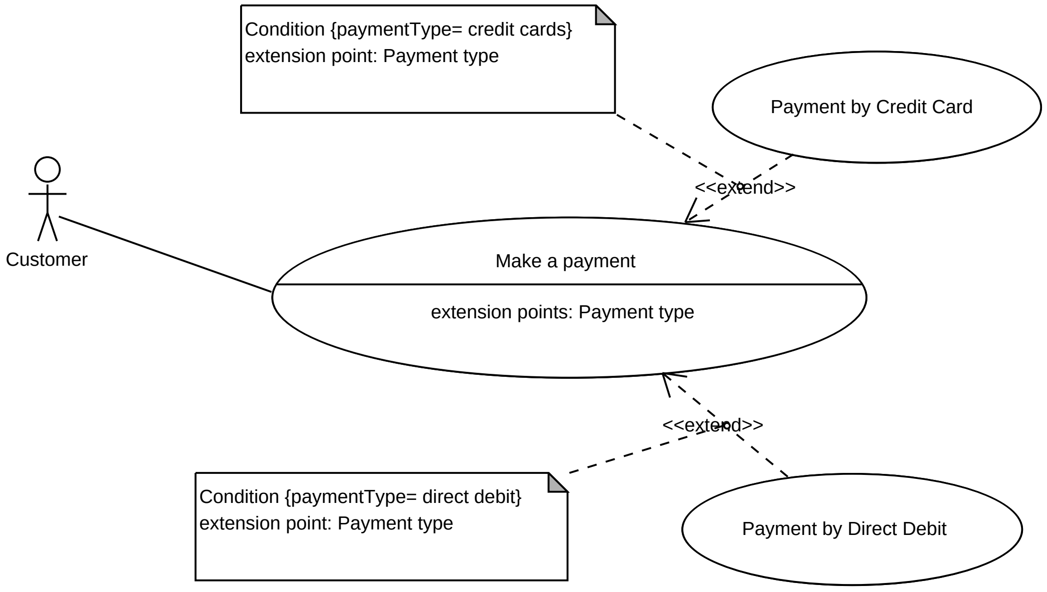
<<extend>> Relationship

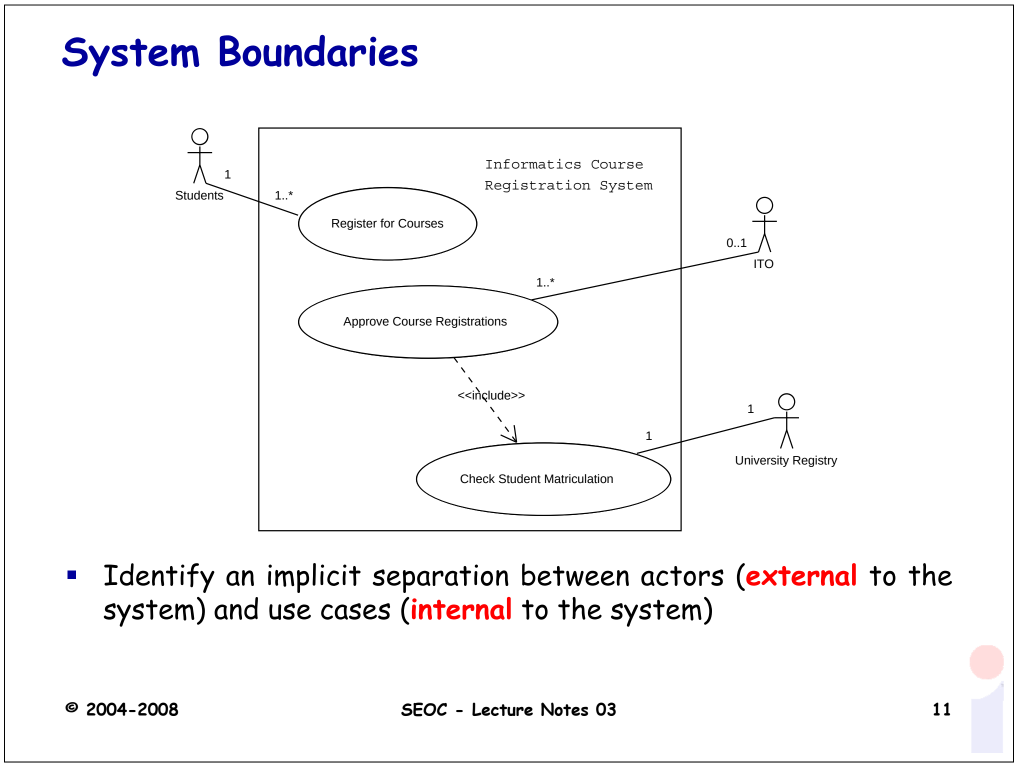

"Add Joint Account Holder" extends "Open Bank Account" by adding extra functionality, but "Open Bank Account" is a valid use case on its own.
Condition: {NUmber of Account Holder(s)>1}
Extension Point: Add Joint Account Holder
Open Bank Account
<<extend>>
Add Joint Account Holder

- The <<extend>> relationship holds when use cases extend, i.e., optionally provide other functionalities to extended use cases
- A use case may be extend by (i.e., completely reuse) another use case, but this is optional and depends on runtime conditions or implementation decisions
- Any use case you want to extent must have clearly defined extensions points

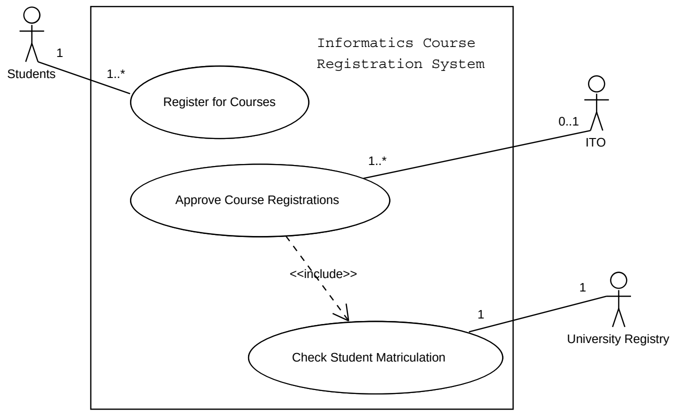
Another example of <<extend>> relationship.

.


Condition {paymentType= credit cards}
extension point: Payment type
Payment by Credit Card
<<extend>>
Customer
Make a payment
extension points: Payment type
<<extend>>
Condition {paymentType= direct debit}
extension point: Payment type
Payment by Direct Debit



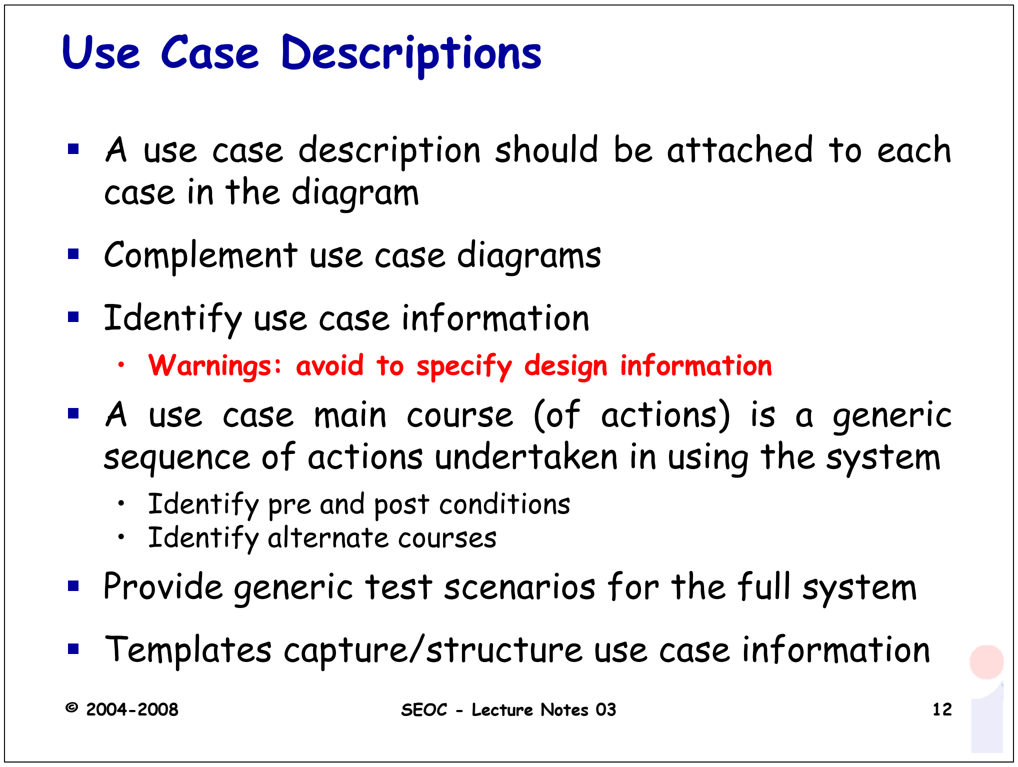
System Boundaries



- Identify an implicit separation between actors (**external** to the system) and use cases (**internal** to the system)

The system boundaries identify what is part of the system and the actors interacting with it. The boundaries affect the functionalities that the system has to implement/support. Therefore, there are both technical (whether the system needs to implement a specific functionality or the functionality is provided by an external actor) as well as business implications (e.g., financial).

Note that it is possible to specify multiplicities between actors and use cases. It is useful to capture various information (e.g., concurrency) already in the use cases. However, it is useful initially to maintain the use case diagrams as general as possible in order to avoid (or commit) to particular design during early stages of the requirements process.



Use Case Descriptions

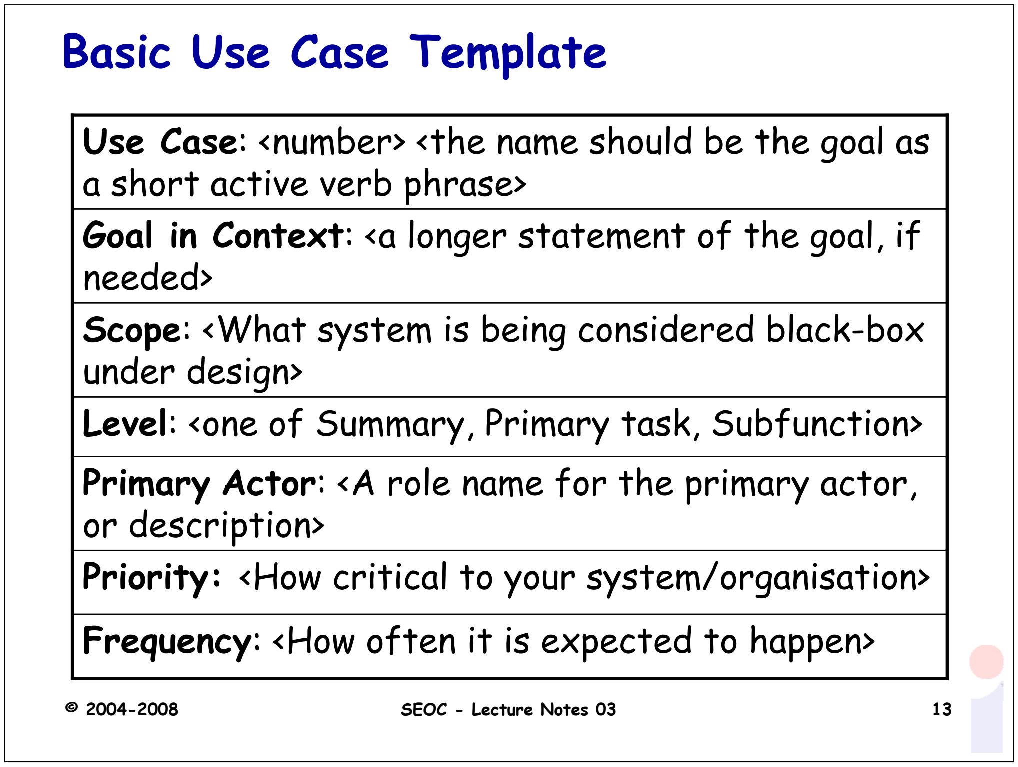
- A use case description should be attached to each case in the diagram
- Complement use case diagrams
- Identify use case information
  - Warnings: avoid to specify design information
- A use case main course (of actions) is a generic sequence of actions undertaken in using the system
  - Identify pre and post conditions
  - Identify alternate courses
- Provide generic test scenarios for the full system
- Templates capture/structure use case information

© 2004-2008 SEOC - Lecture Notes 03 12

Some types of information are, e.g.: actors, related requirements, preconditions, successful/failed end conditions.

Description Example 1.
Use Case name: Register for Courses
Description: This use cases allows students to register for informatics courses. The student uses the Informatics Course Registration System, an online system, for selecting the courses to attend for the forthcoming semester.
Main course:
1.This use case starts when a student visits the system web page
    1.1 The system provides the list of available courses in the forthcoming semester
2.The student identifies the courses and select them
3.The student confirm the selection, which is then recorded

Description Example 2.
Use Case name: request an appointment with a GP (General Practitioner).
Description: An system allows patients to request appointments with GPs.
Main course:
1.A patient requests appointment to the system
2.The system queries a scheduler for available GPs and times
3.The system responds with GPs and times
4.The system negotiates with Patient on suitable GP/time
5.The system confirms GP/time with the Scheduler
6.The scheduler responds with confirmation of appointment (e.g. booking number)
7.The system communicates confirmation to Patient



Basic Use Case Template

Use Case: <number> <the name should be the goal as a short active verb phrase>
Goal in Context: <a longer statement of the goal, if needed>
Scope: <What system is being considered black-box under design>
Level: <one of Summary, Primary task, Subfunction>
Primary Actor: <A role name for the primary actor, or description>
Priority: <How critical to your system/organisation>
Frequency: <How often it is expected to happen>

Readings.

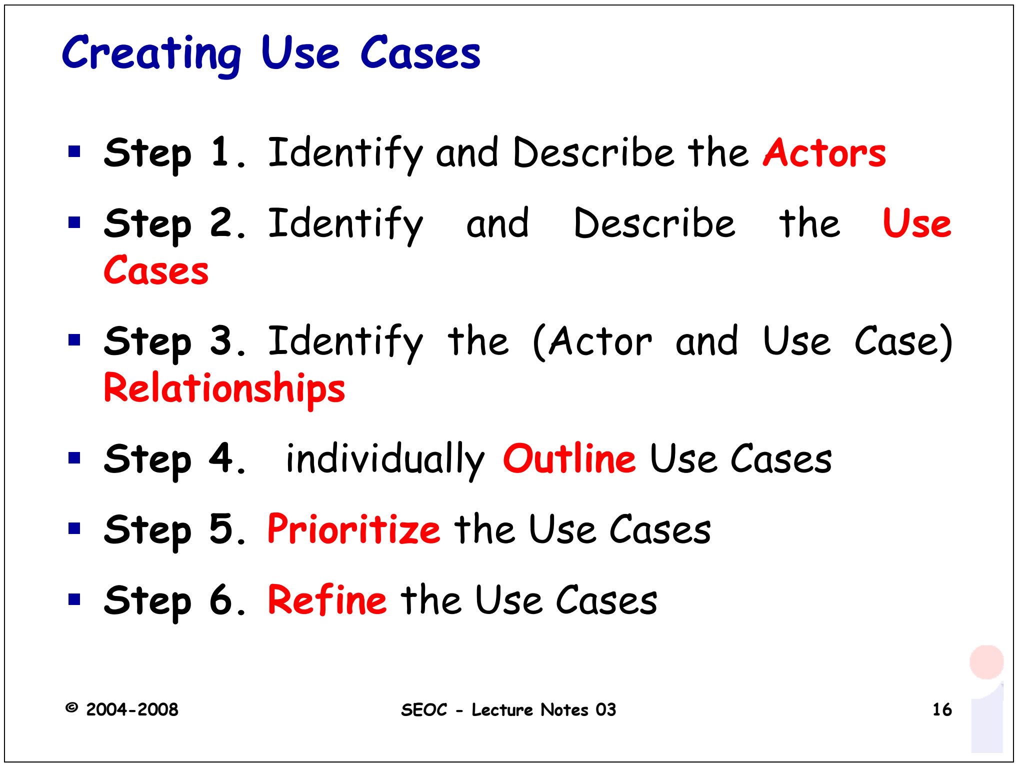
• Alistair Cockburn. Structuring Use Cases with Goals. The paper introduces a **Basic Use Case Template**.

Another Use Case Template

Use Case: Use case identifier and reference number and modification history
Description: Goal to be achieved by use case and sources for requirements
Actors: List of actors involved in use case
Assumptions: Conditions that must be true for use case to terminate successfully
Steps: Interactions between actors and system that are necessary to achieve the goal
Variations (optional): any variations in the steps of a use case
Non-Functional (optional): List of non-functional requirements that the use case must meet.
Issues: List of issues that remain to be solved

Using a Use Case Template

1. Learn to fill in all the fields of the template in several passes
2. Stare at what you have so far
3. Check your project's scope
4. Identify the open issues and a deadline for the implementation
5. Identify all the systems to which you have to build interfaces



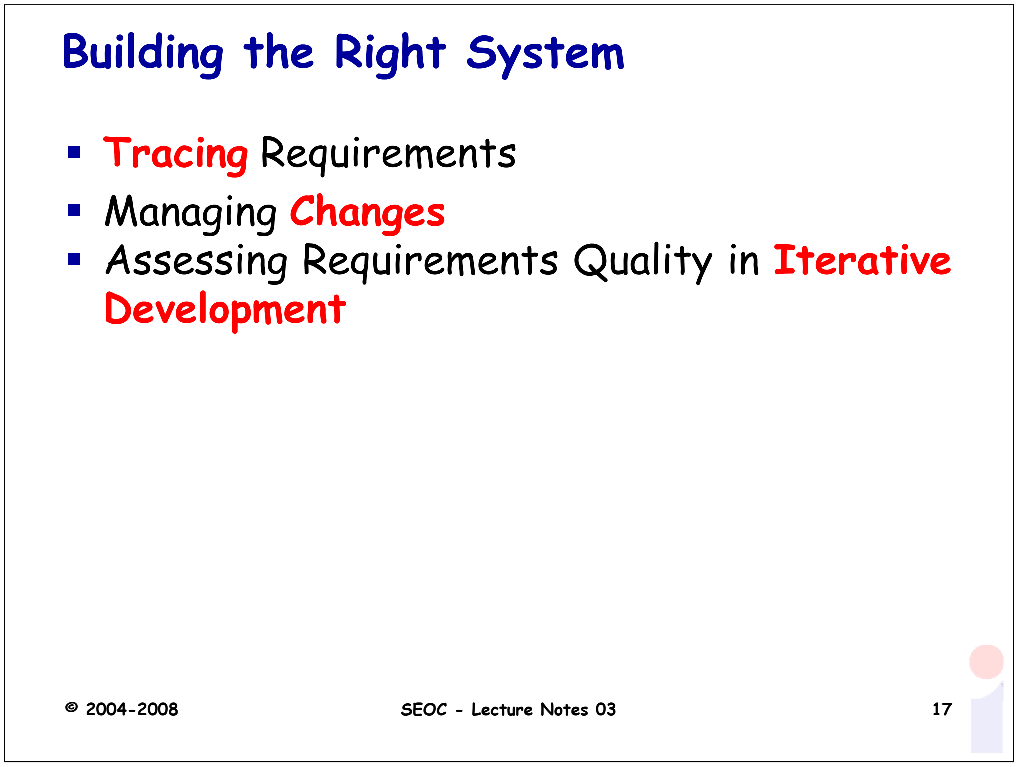
Creating Use Cases

- **Step 1.** Identify and Describe the Actors
- **Step 2.** Identify and Describe the Use Cases
- **Step 3.** Identify the (Actor and Use Case) Relationships
- **Step 4.** individually Outline Use Cases
- **Step 5.** Prioritize the Use Cases
- **Step 6.** Refine the Use Cases

© 2004-2008 SEOC - Lecture Notes 03 16

Simple questions or checklists support the specification of use cases.

Step 1. Identify and Describe the Actors: who uses the system? who manages the system? who maintains the system? Who provides information to the system? Who gets information from the system? etc.
Identify and Describes the Use Cases: What will the actor use the system for? Will the actor create, store, change, remove or read information in the system? etc.
Step 3. Identify the Actor and the Use Case Relationships
Step 4. Outline the individual Use Cases
Step 5. Prioritize the use cases: for instance, on the basis of utility or frequency of use depending on the process this may be closely linked to what is needed in the process
Step 6. Refine the Use Cases: Develop each use case (starting with the priority ones) develop the associated use case structure the use case



Building the Right System

- Tracing Requirements
- Managing Changes
- Assessing Requirements Quality in Iterative Development

UML supports traceability links from use cases to implementation. This allows the mapping of high level functional requirements to design and code.

Orthogonality problem: the structure of requirements and the structure of design and implementation are different. These structures emerge as requirement dependencies and system architecture respectively. Unfortunately, the complexity of such structures may increase the project risk (e.g., increasing cost and effort, late development, etc.) as well as affecting system features. A lack of understanding of system requirements and their allocation to the system design could result un poorly designed object oriented systems (e.g., high coupling and low cohesion).

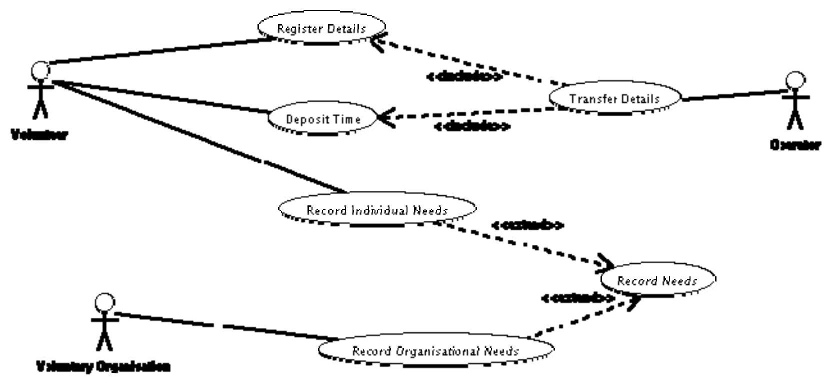
Further traceability links allow to relate use cases to test cases. A scenario, or an instance of use case, is an use case execution wherein a specific user executes the use case in a specific way. Note that a use case is not a test case - a use case usually involves many different test cases.

Stakeholders interaction, business constraints, implementation issues, system usage and so on may trigger requirements changes. Successive refinement, rather than absolute completeness, or specificity, is the goal.

VolBank: Creating Use Cases

- Who are the main actors in the VolBank example?
- Can you identify all the main use case names in the system?
- What opportunities are there to structure the use case diagram?
- Can you see any non-functional requirements that are present in the specification?
- How well are non-functional requirements represented in the use case diagram?

VolBank: Incomplete Use Cases



VolBank: Using Use Case Template

Use Case: 01 - deposit time
Goal in Context: *The VolBank system allows volunteers to deposit their availabilities in terms of time*
Scope: *volunteers' profiles are unavailable*
Level: Primary task
Primary Actor: Volunteers
Priority: *It supports one of the major functionalities of the VolBank system*
Frequency: *Every time volunteers provide information about their availability*

Readings

- **UML course textbook**
  - Chapter 3 on Use Cases
- Alistair Cockburn. Structuring Use Cases with Goals.
  - The paper introduces a **Basic Use Case Template**

Summary

- Use Cases in UML capture (to a certain extent) system requirements and support requirements engineering activities and processes
- Use Case notations and examples
- Describing use cases
- Developing use cases