Use Cases

Massimo Felici

School of informatics

Use Cases

- Support **requirements engineering activities** and the requirement process
- Capture what a system is supposed to do, i.e., systems **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

Slide 1: Use Cases

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 modelling 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).

Slide 1: Use Cases

Required Readings

- UML course textbook, Chapter 3 on Use Cases

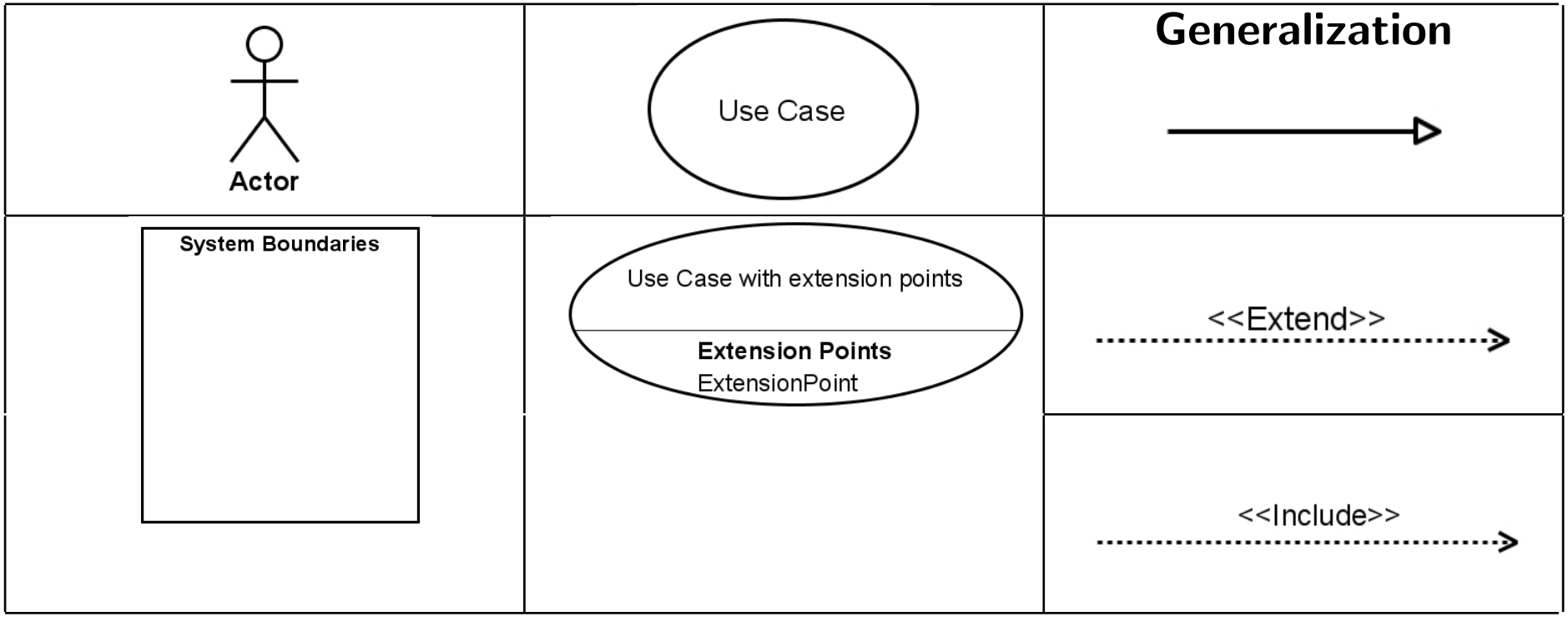
The Benefits of Use Cases

- 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 at a Glance



Slide 4: Use Cases at a Glance

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

Slide 4: Use Cases at a Glance

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

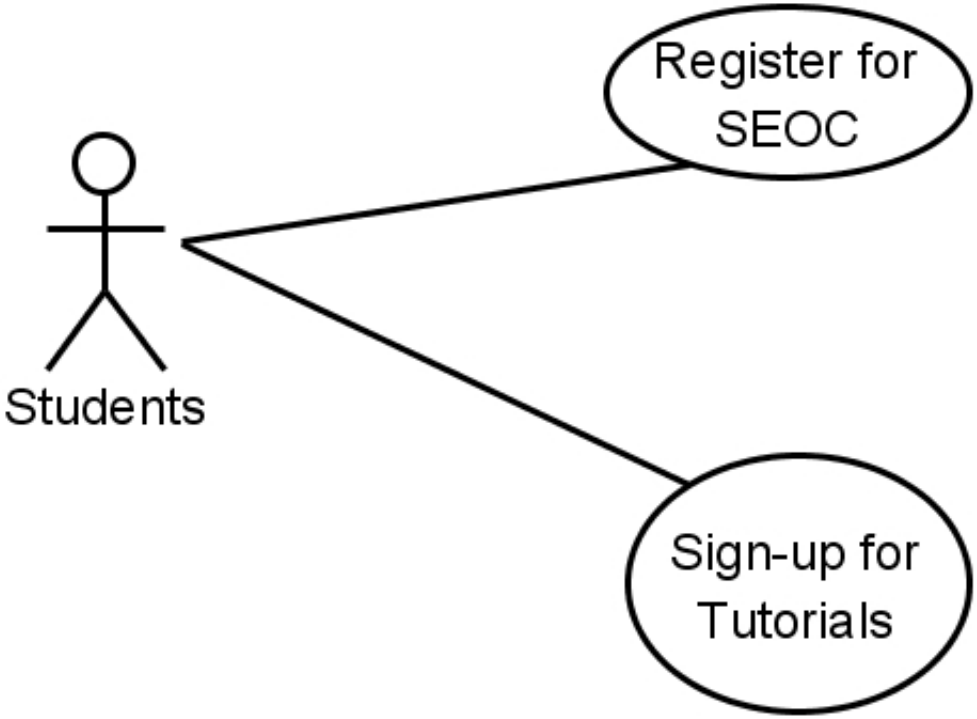
- Finding nonhuman actors
  - Incorporating other systems (e.g., databases)
  - Ignoring internal components
  - Input/Output Devices
- Roles of the Actors
- Naming the Actors

Slide 5: Actors and Use Cases

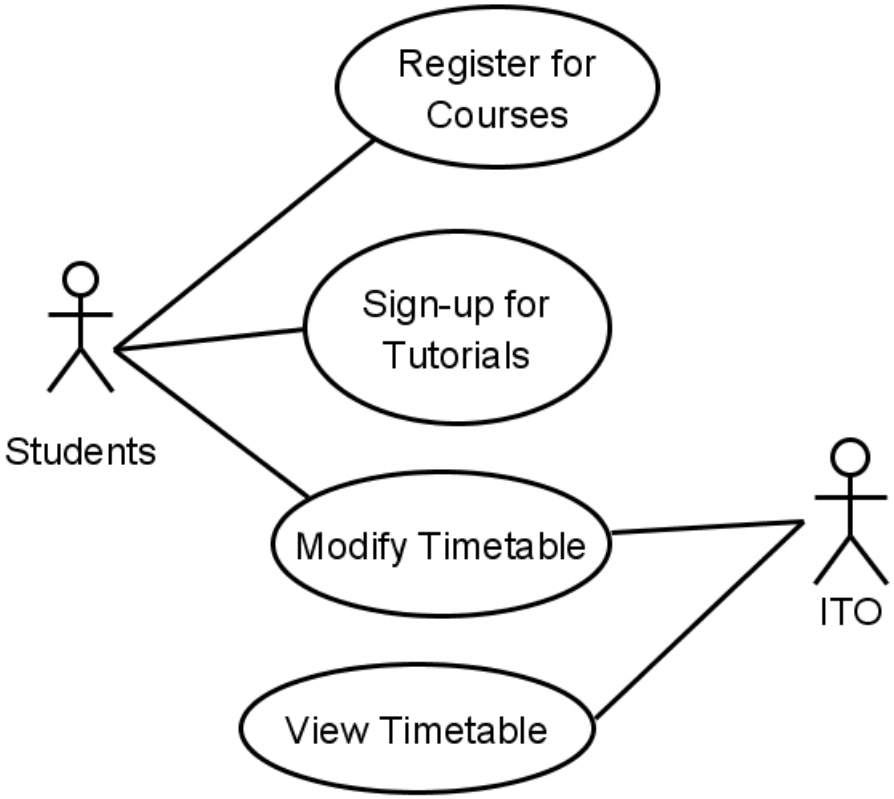
Here are some general hints:

- 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.

Actors and Use Cases



Actors and Use Cases



Finding Actors

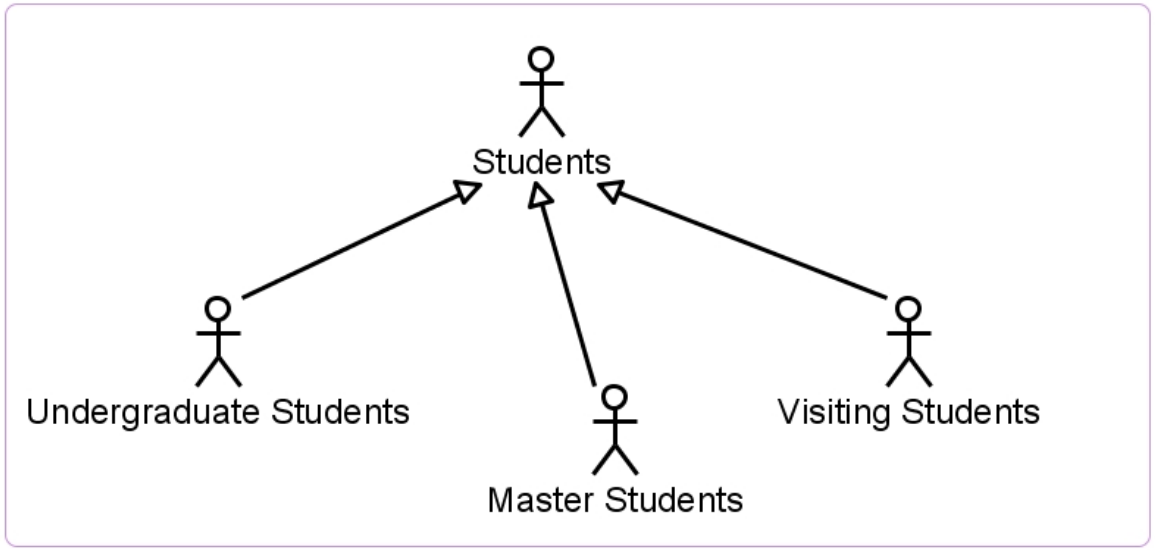
The actors can be identified by answering a number of questions:

- Who will use the main functionality of the system?
- Who will need support from the system to do their daily tasks?
- Who will need to maintain and administer the system, and keep it working?
- Which hardware devises does the system need to handle?
- With which other systems does the system need to interact?
- Who or what has an interest in the results (the value) that system produces?

Slide 8: Finding 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.

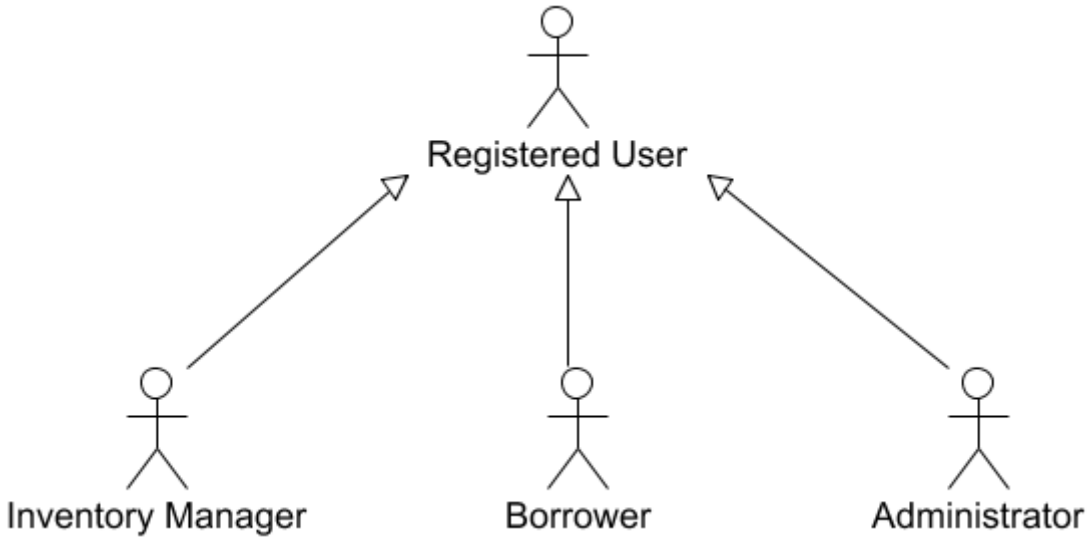
Generalizations between Actors



Slide 9: Generalizations 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



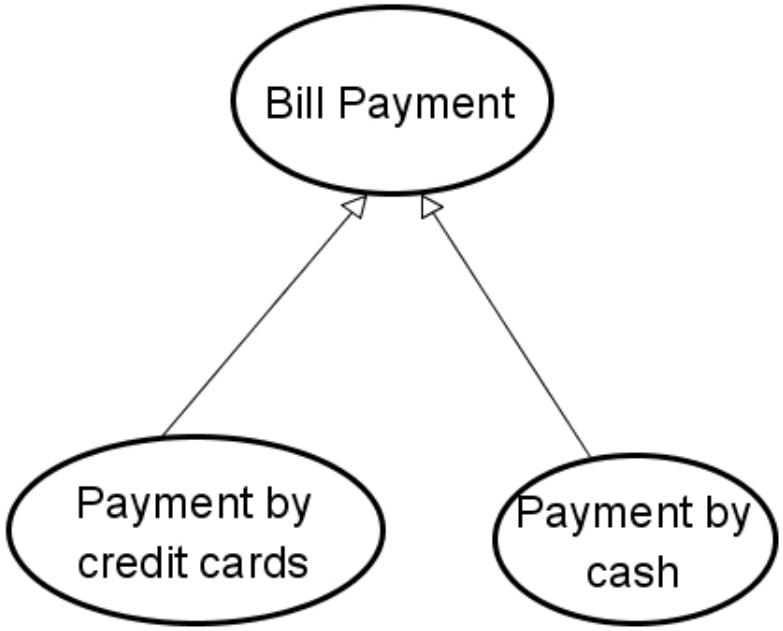
Finding Use Cases

For each identified actor, ask the following questions:

- Which functions does the actor require from the system? What does the actor need to do?
- Does the actor need to read, create, destroy, modify, or store some kind of information in the system?
- Does the actor have to be notified about events in the system, or does the actor need to notify the system about something? What do those events represent in terms of functionality?
- Could the actor's daily work be simplified or made more efficient bu adding new functions to the system?

Generalizations between Use Cases

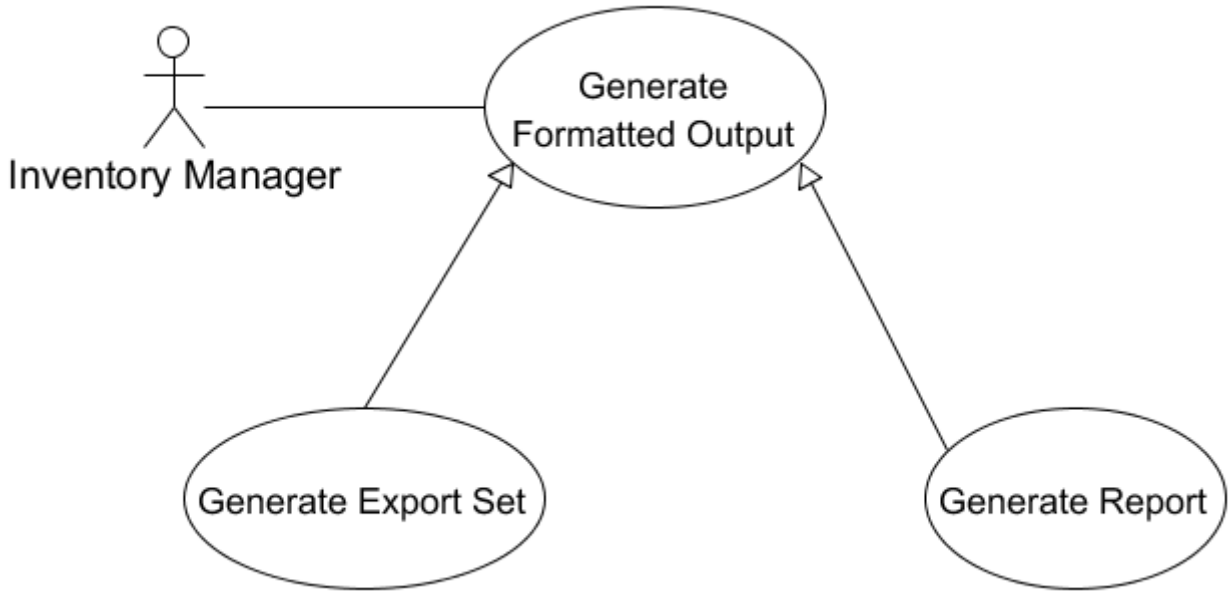
Payment, for instance, is a generalization of Payment by credit cards and payment by cash



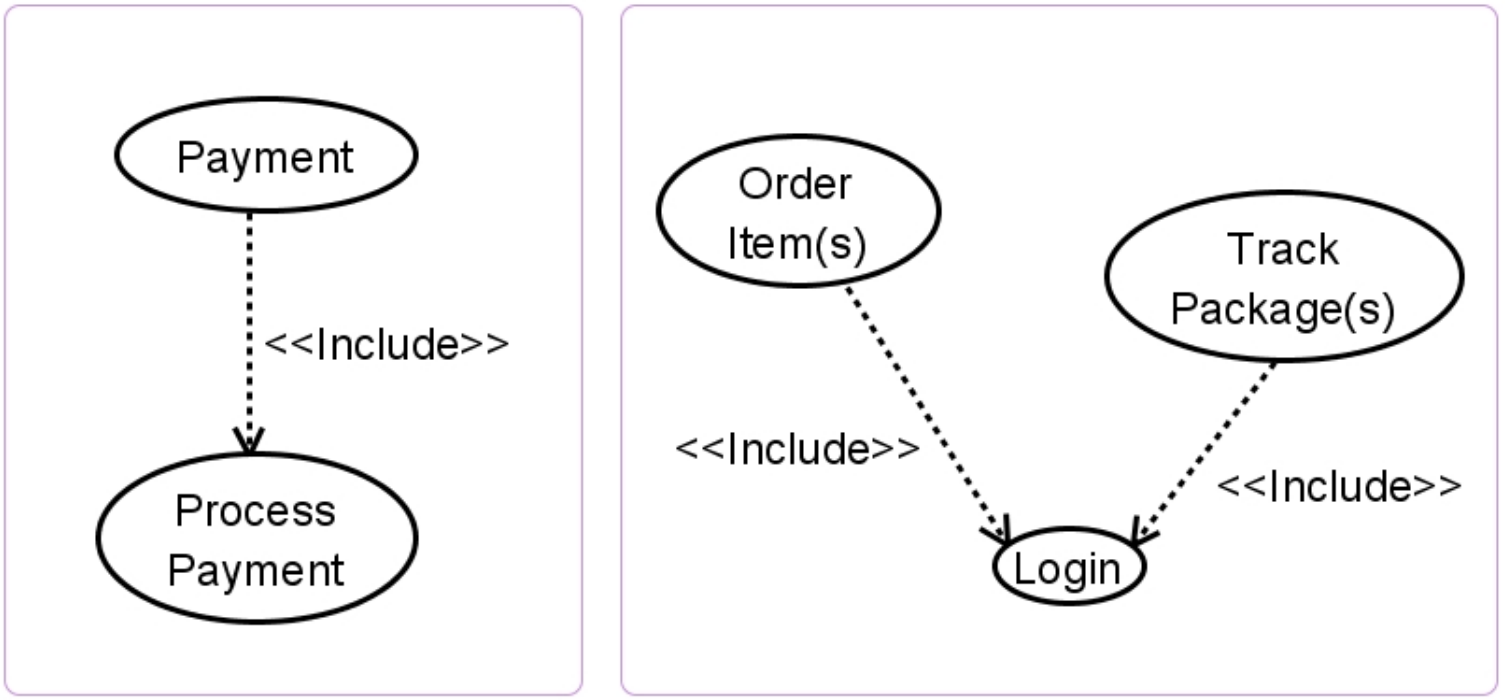
Slide 11: Generalizations between Use Cases

- Indicate that the more specific use case receives or inherits the actors, behaviour sequences, and extension points of the more general use case

- Generalization is often implemented by inheritance.



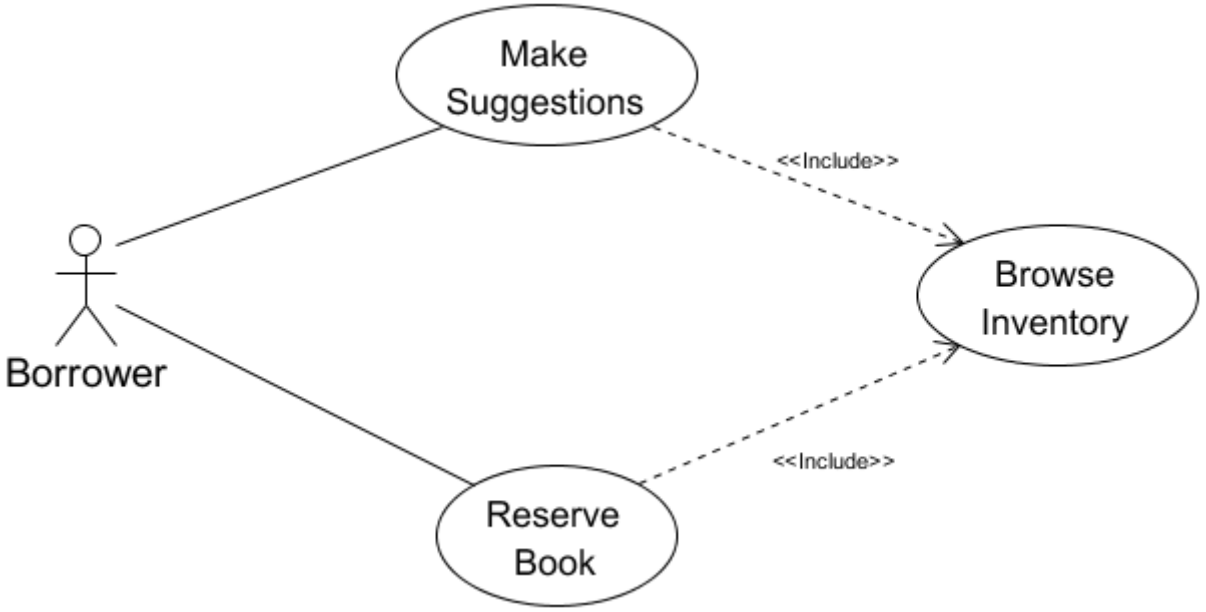
<<include>> Relationship



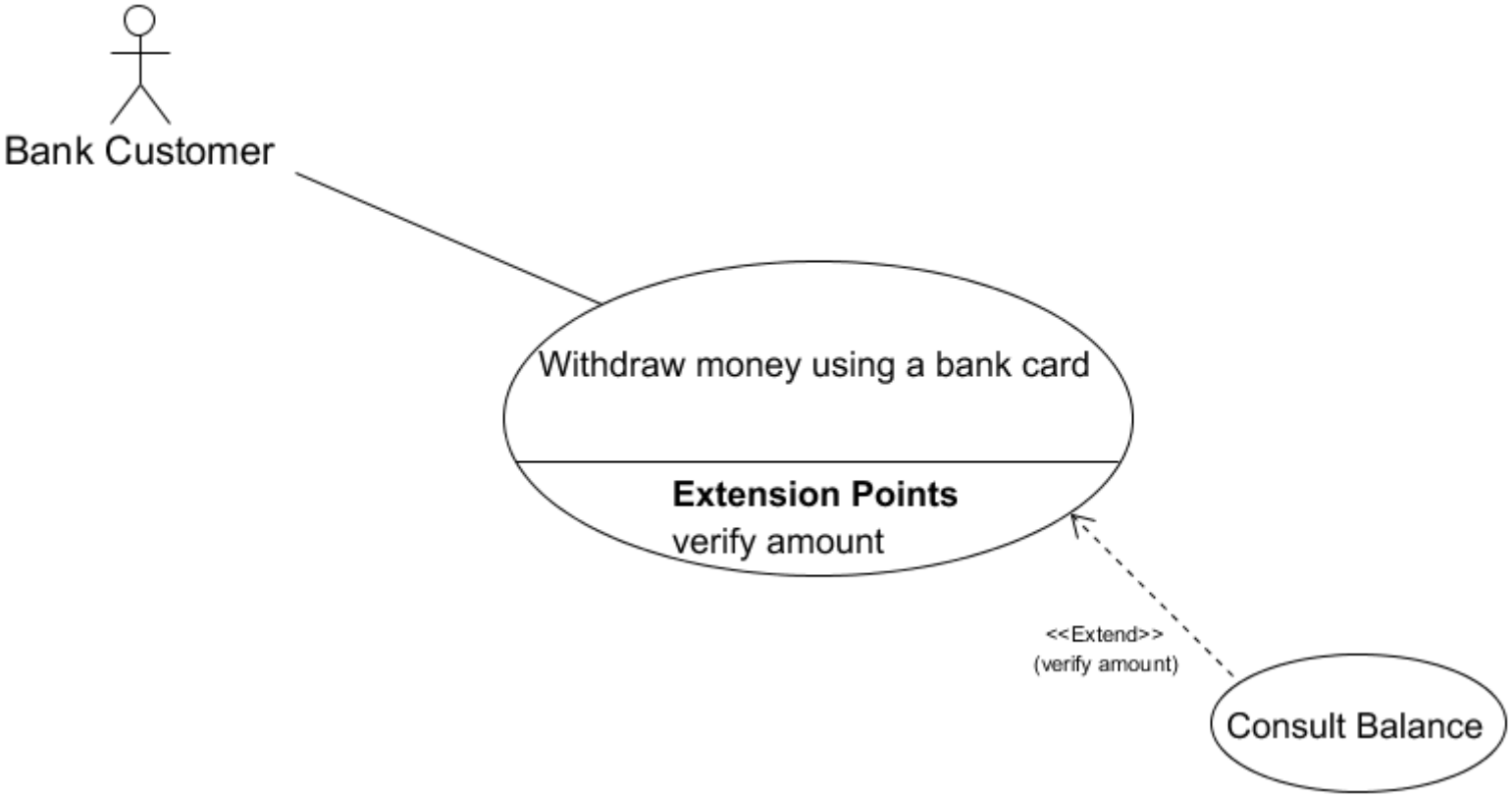
Slide 12: <<include>> Relationship

- 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
- An include relationship shows how common behaviour in a number of use cases can be described in a shared use case that is included in the other use cases

Slide 12: <<include>> Relationship



<<extend>> Relationship



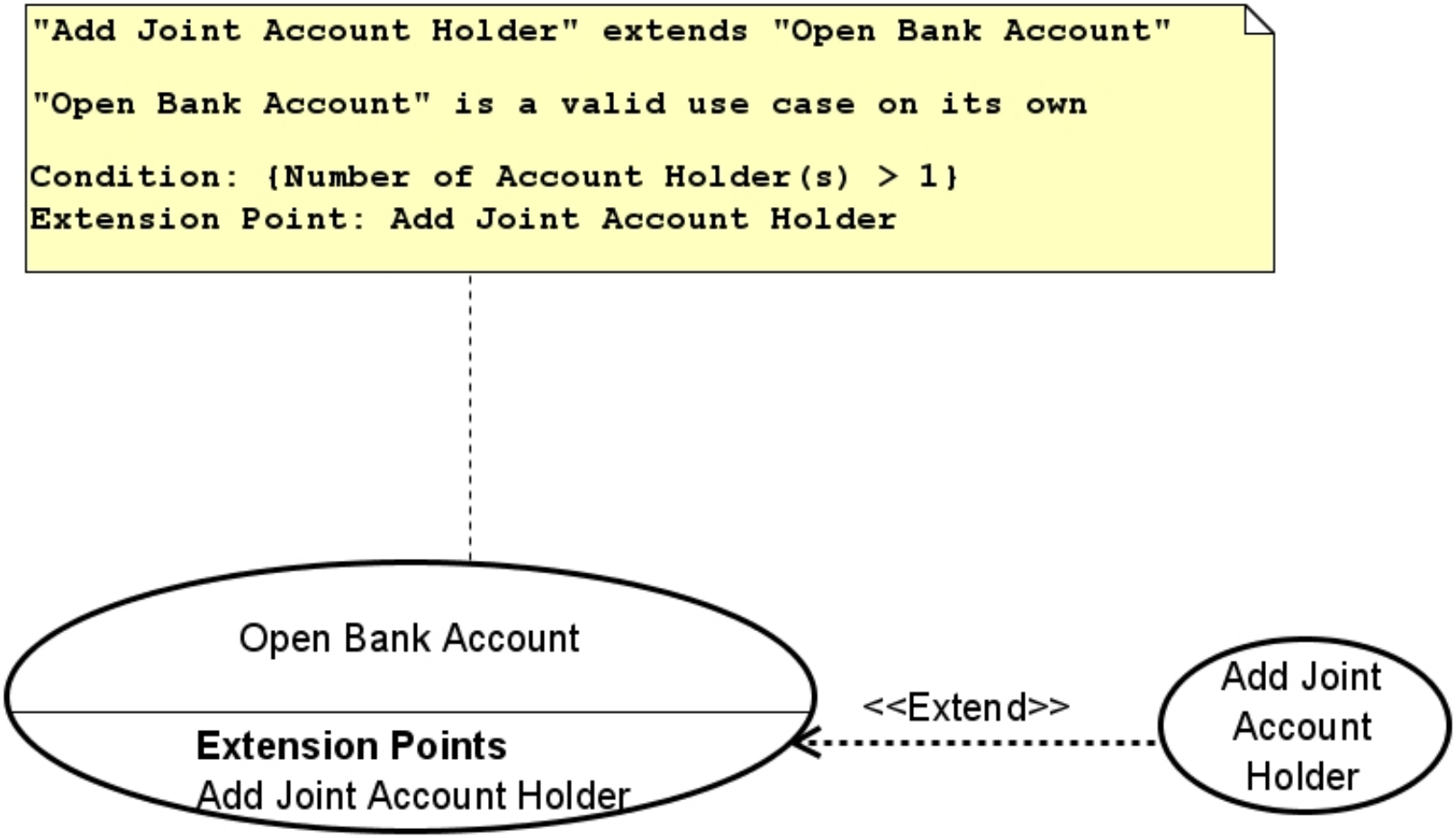
Slide 13: <<extend>> Relationship

- The <<extend>> relationship holds when use cases extend, i.e., optionally provide other functionalities to extended use cases

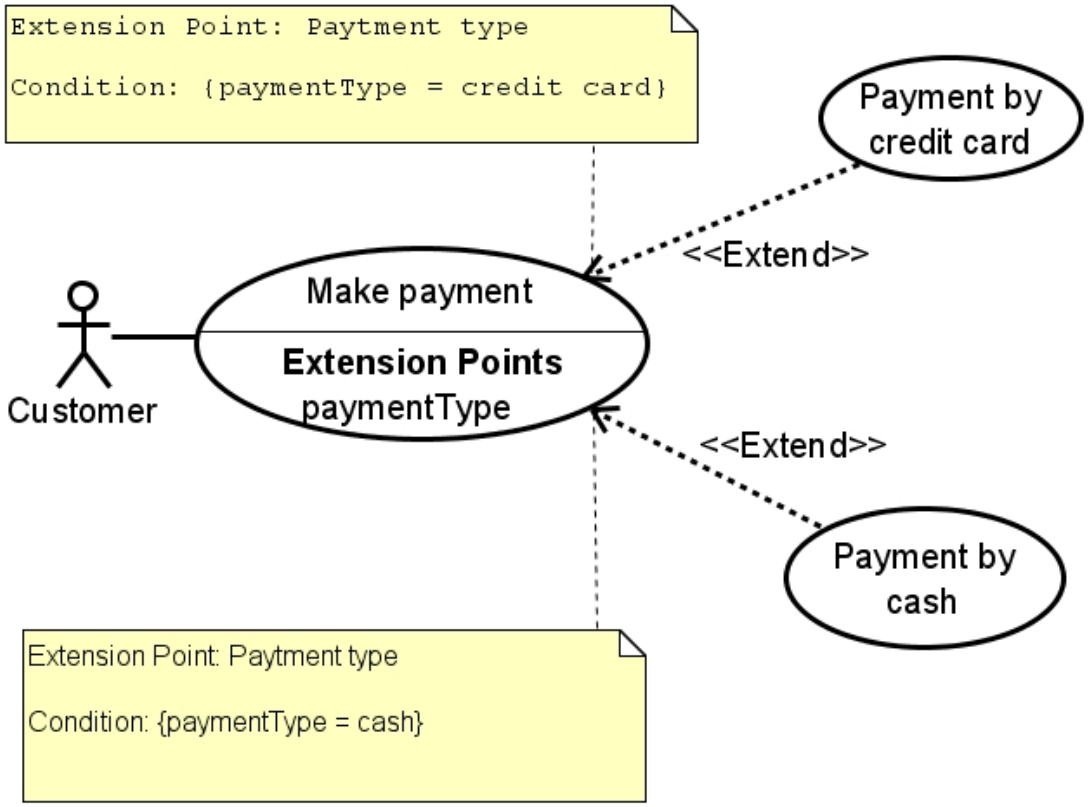
- A use case may be extend by (i.e., completely reuse) another use case, but this is optional and depends on runtime conditions or implementation decisions

- Any use case you want to extent must have clearly defined extensions points

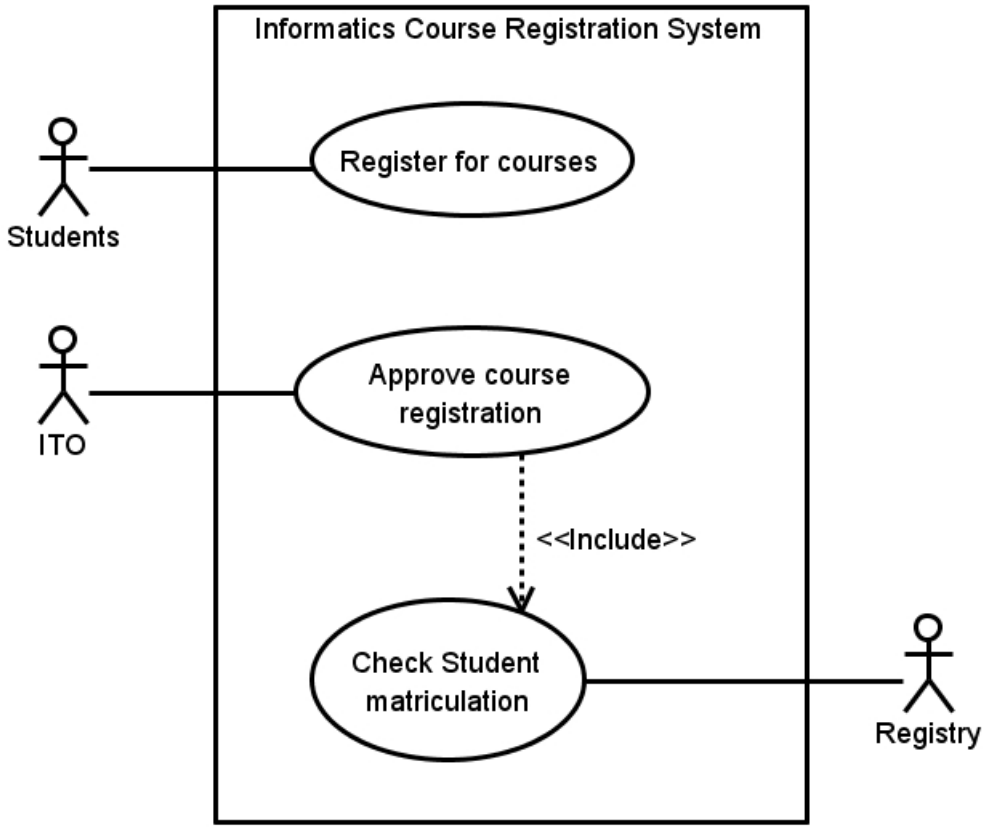
Slide 13: <<extend>> Relationship



<<extend>> Relationship



System Boundaries



Slide 15: 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 complements each use case in the diagram
- 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
- Some types of information are, e.g.: actors, related requirements, preconditions, successful/failed end conditions

Use Case Descriptions

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

Use Case Descriptions

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

A Basic Use Case Template

Use Case [number ♯]	*the name is the goal as a short active verb phrase*	
Goal in Context	*a longer statement of the goal, if needed*	
Preconditions	*what we expect is already the state of the world*	
Success End Condition	*the state of the world upon successful completion*	
Failed End Condition	*the state of the world if goal abandoned*	
Primary Actor	*a role name or description for the primary actor*	
Secondary Actors	*other systems relied upon to accomplish use case*	
Trigger	*the action upon the system that starts the use case*	
Description	**Step**	**Action**
	1 ...	
Extensions or **Variations**	**Step**	**Branching Action**
	...	*condition causing branching* *action or name of sub-use case*

Slide 19: Using the 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 projects scope

4. Identify the open issues and a deadline for the implementation

5. Identify all the systems to which you have to build interfaces

How to Create 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

Slide 20: How to Create Use Cases

Simple questions or checklist to 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.

Step 2. 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

Slide 20: 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.

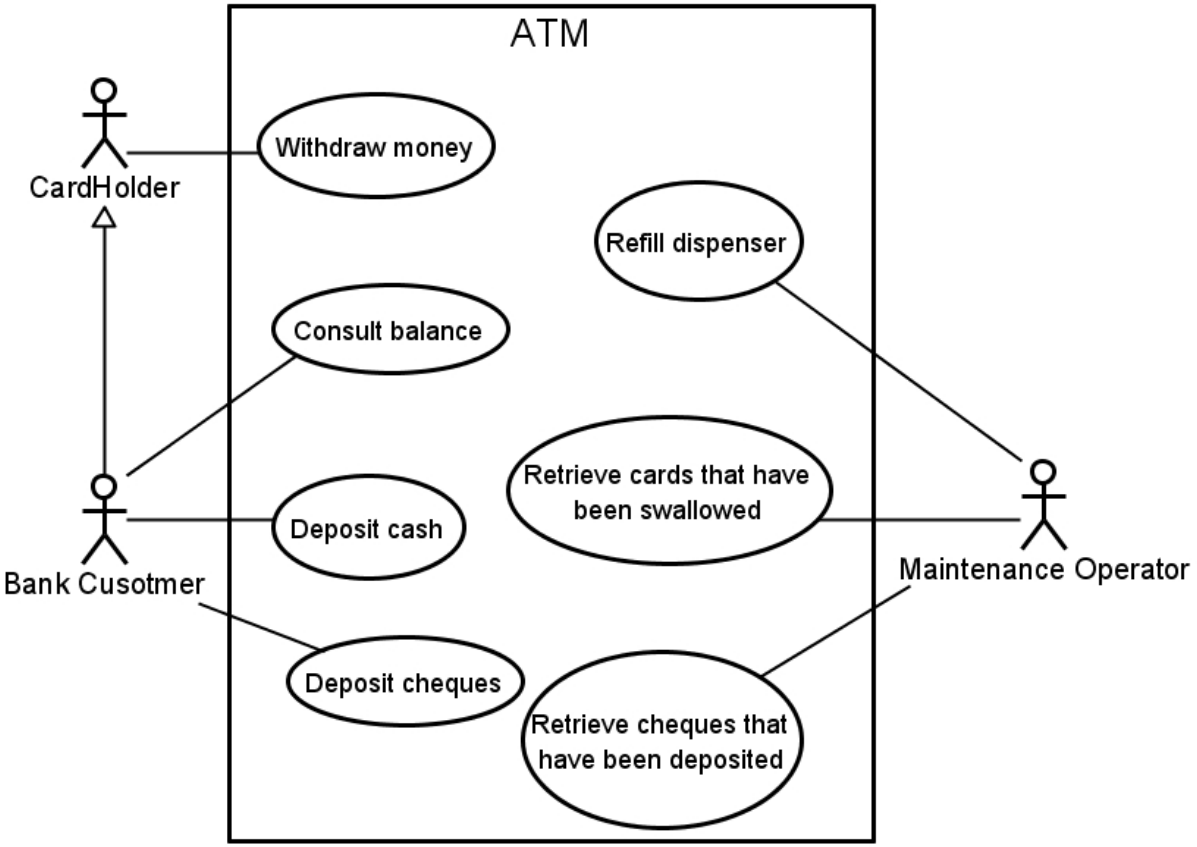
Slide 20: Building the Right System

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.

An ATM System



Use Case Description: Withdraw money

Use Case [number ♯1]	*Withdraw money*	
Goal in Context	*This use case allows a card holder, who is not a customer of the bank, to withdraw money if his or her daily limit allows it*	
Preconditions	*The ATM is well stocked and in service*	
Success End Condition	*the CardHolder withdraws the required money*	
Failed End Condition	*...*	
Primary Actor	*CardHolder*	
Secondary Actors	*The ATM Bank, The CardHolder's Bank*	
Trigger	*The CardHolder introduces the card in the ATM*	
Description	**Step**	**Action**
	1	
Extensions or **Variations**	**Step**	**Branching Action**

Required Readings

- UML course textbook, Chapter 3 on Use Cases

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

