CMPS411

Software engineering

Lecture- 4

Introduction to UML and Requirements Analysis with Use Cases

Topics covered

- System Modeling
- Use case Models
- Use case Diagram
- Use Case Relationships
- Use Case specification
- Normal scenario: Actor Action – System Response Table
- How to develop a Use-Case Model

Unified Modeling Language (UML)

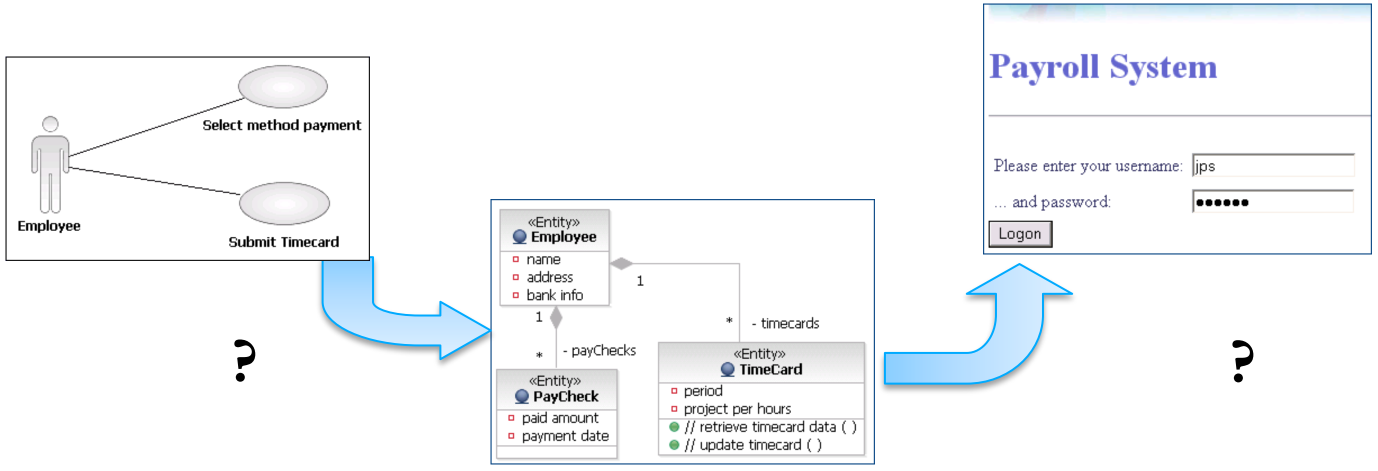
UML Is a Language for Specifying

UML Is a Language for Constructing

UML Is a Language for Documenting

UML Is a Language for Visualizing

A picture is worth a thousand words!



Payroll System

Please enter your username: jps

... and password: ••••••

Logon

Scenario-Based Modeling

“[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases).” Ivar Jacobson

(1) What should we write about?

(2) How much should we write about it?

(3) How detailed should we make our description?

(4) How should we organize the description?

Scenarios

- A <u>scenario</u> is an *instance* of a use case
  - It expresses a *specific occurrence* of the use case
    - a specific actor ...
    - at a specific time ...
    - with specific data.

What Is a Use-Case Model?

•A model that describes a system’s functional requirements in terms of use cases

•It shows the system’s intended functionality (use cases) and its environment (actors)

•It shows how the users can use the system to achieve their goals

•A *use case* is a typical sequence of actions that a user performs in order to complete a given task

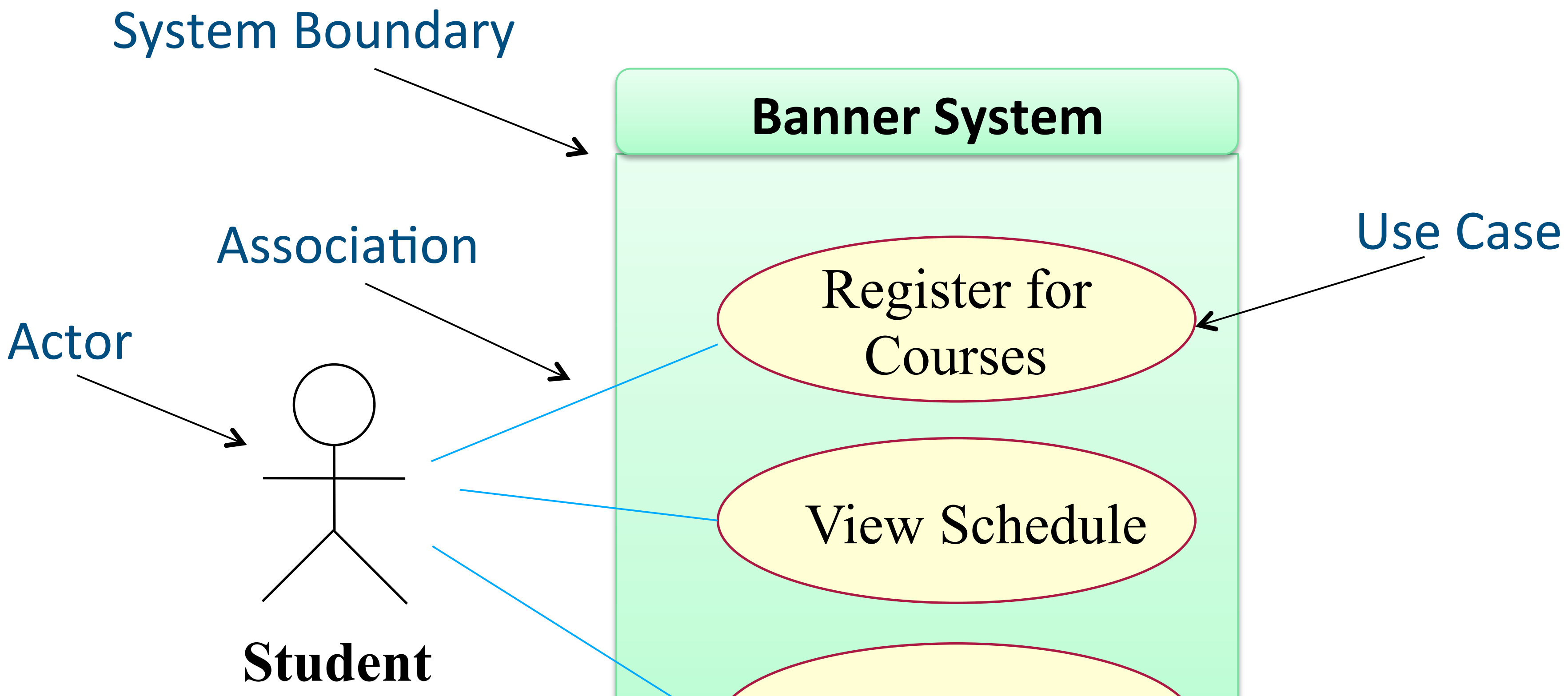
•The objective of *use case analysis* is to model the system

… from the point of view of how users interact with this system
… when trying to achieve their objectives.

Use-Case Model and System's Functionality

- A model that describes a system's **functional requirements** in terms of **use cases**
  - shows the **system's intended functionality** (use cases) and **its environment** (actors)
  - shows how the users can **use the system** to **achieve their goals**

Simplified Use case model for Banner



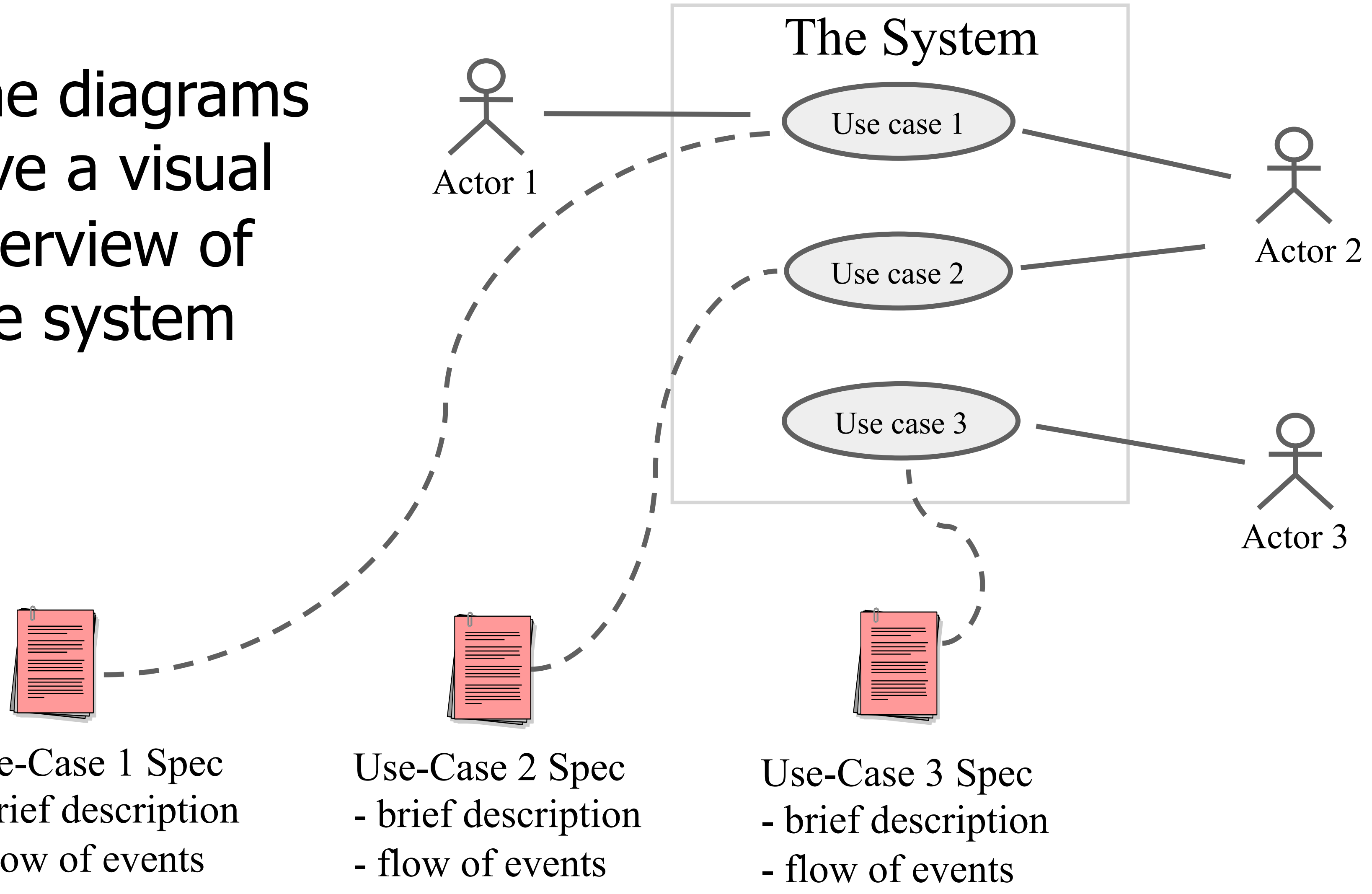
+ A document describing the use case scenarios in details

What is Use Case

- *A use-case is a sequence of actions a system performs that yields an observable result of value to a particular actor.*
- Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself
- A **use case** is a collection of related success and failure scenarios that describe actors using a system to support a goal
- A set of use cases should describe all possible interactions with the system
- A use case tells a story of <u>actors</u> using a <u>system</u>. (e.g."Rent Videos")
- System is a 'black box' that provides functionality to the user. How it does it is not relevant (at this stage)

A Use-Case Model is Mostly Text

The diagrams give a visual overview of the system



Define Use Cases

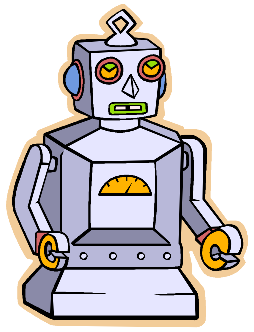
Use Cases: Identify Actors

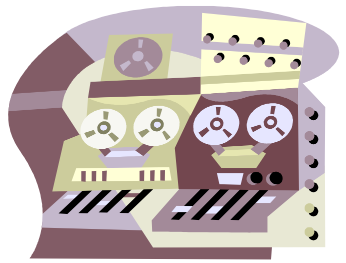
- Who or what (the role) is/will be providing input to the system?
- Who or what (the role) is/will be receiving output from the system?
- Who or what (the role) is right next to the system boundary passing input into the system?
- Who or what (the role) is right next to the system boundary getting output from the system?

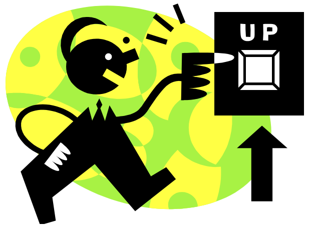
Finding Actors

- External objects that produce/consume data:
  - Must serve as sources and destinations for data
  - Must be external to the system
  - Usually nouns, such as student, staff, camera, external computer, car, etc.

Human

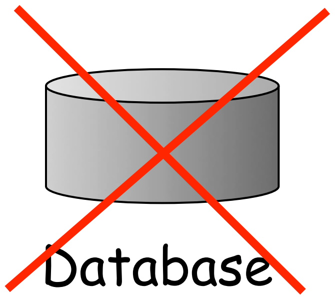

Machine

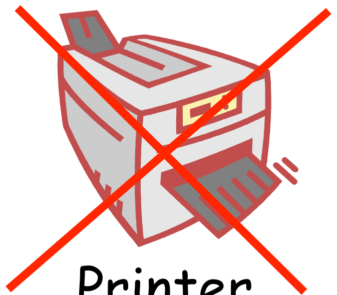

External system



Sensor

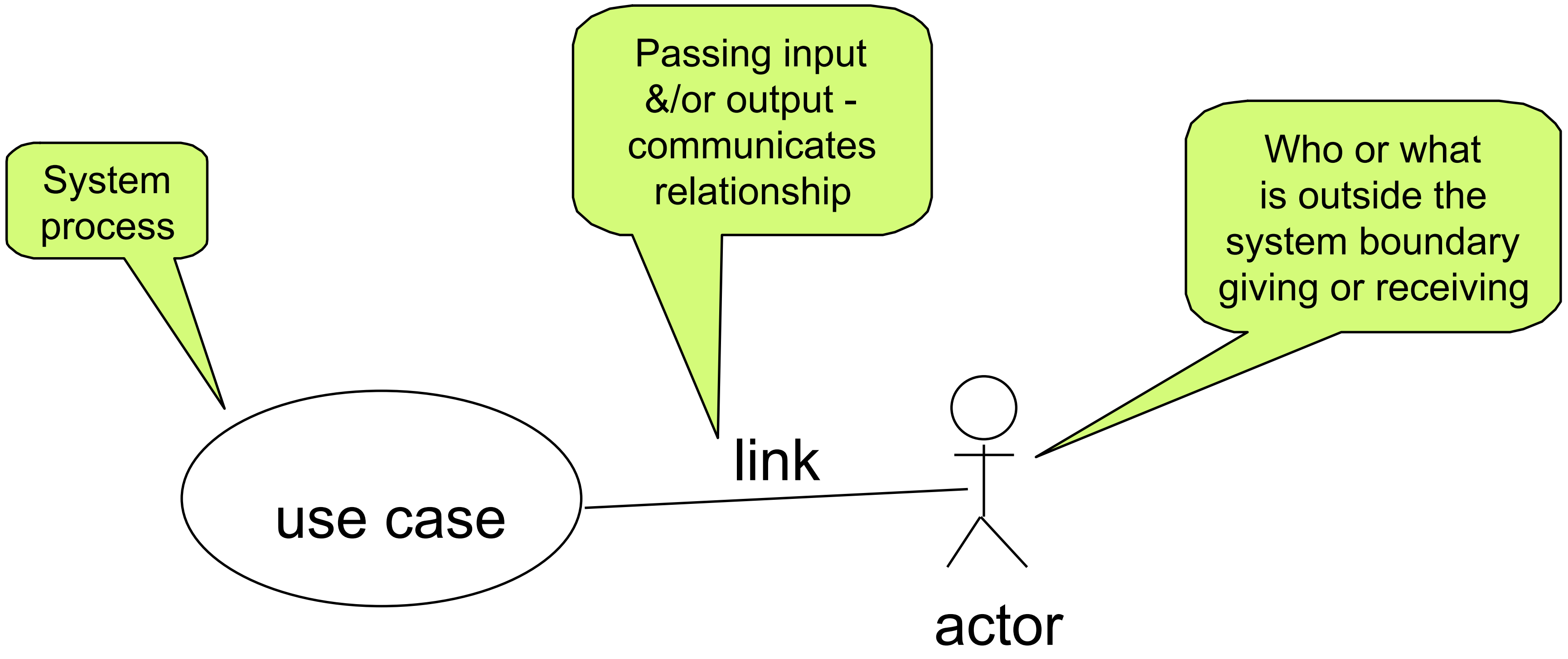
Organizational Units


~~Database~~


~~Printer~~

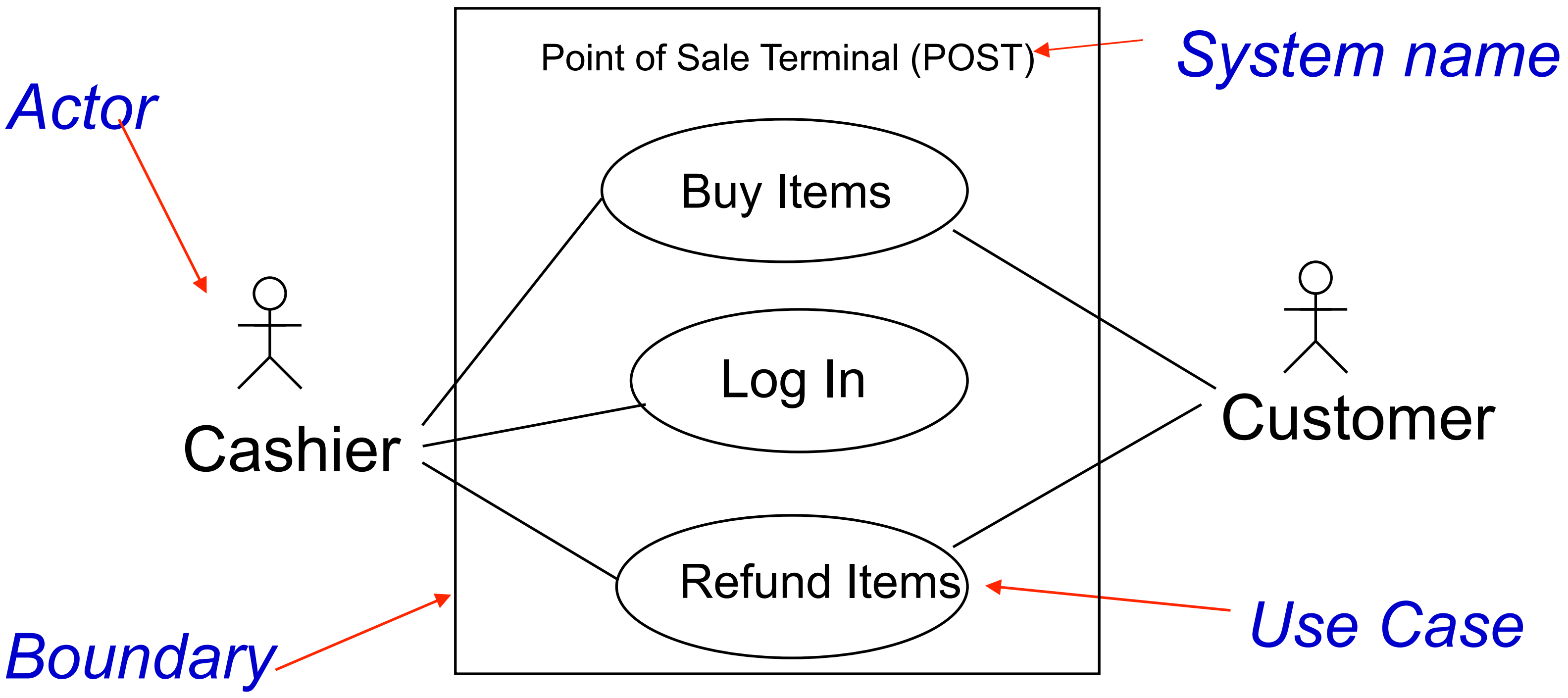
Draw a Use Case Diagram

- Shows all the use cases, actors and associations


System process
Passing input &/or output - communicates relationship
Who or what is outside the system boundary giving or receiving
use case
link
actor

The UML Icons for a use case, actor and association

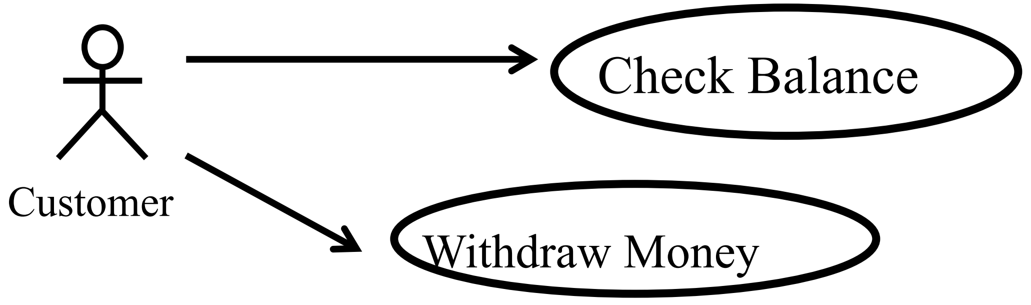
A Sample Use Case Diagram



Partial Use Case Diagram

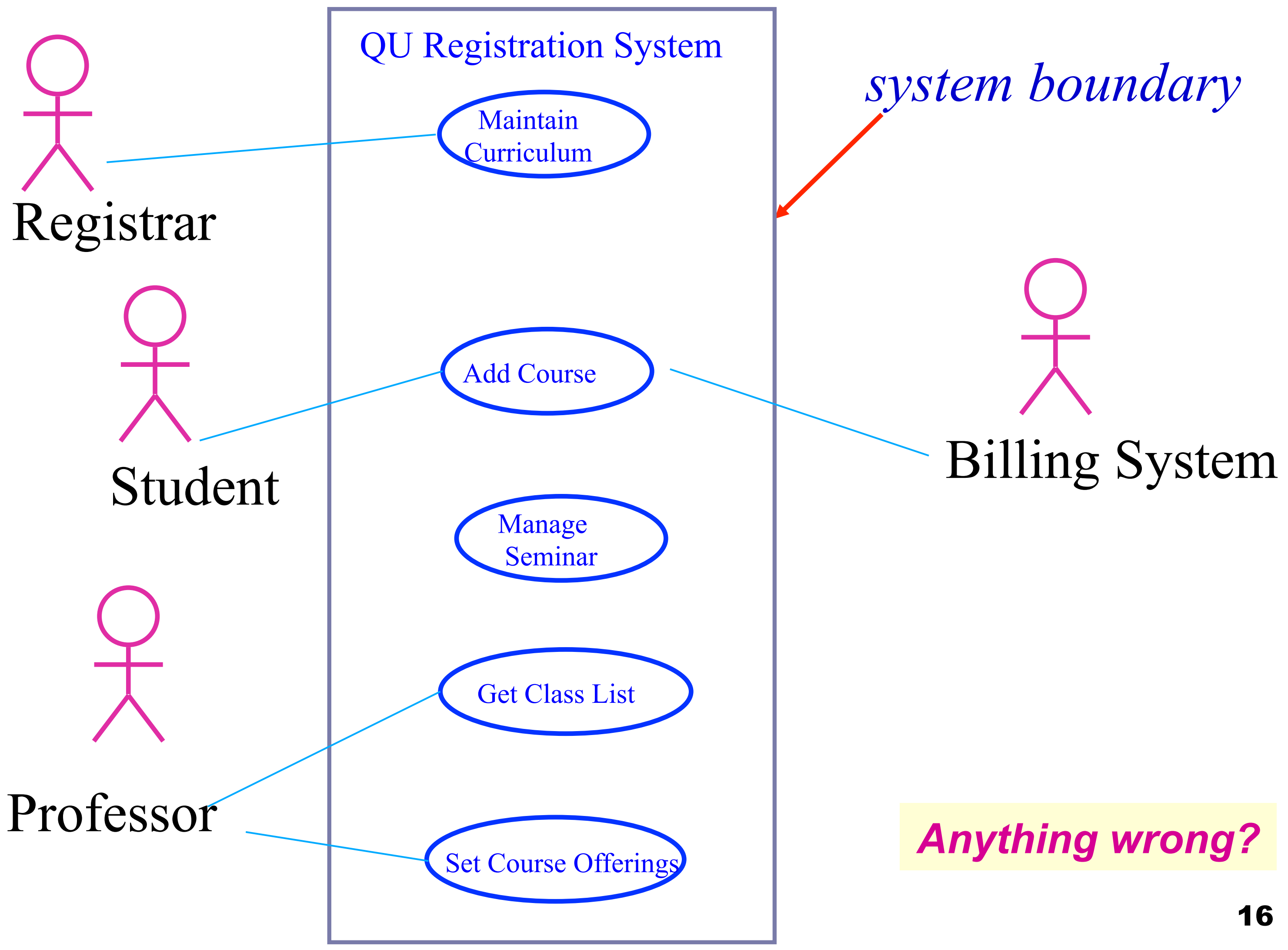
Major Concepts in Use-Case Modeling

- An **actor** represents **Someone or something outside the system that interacts with it**
  - Someone <u>or</u> something **outside** the system that interacts with the system either by giving or receiving information or both.
  - Actors represent "roles" not individuals
- A **use case** is a **sequence of actions a system performs that yields an observable result** of value to a particular actor.
  - describes *what* a system does, but it does not specify *how* it does it



Use Case Diagram: University System

- Use case diagrams depict the relationships between actors and use cases

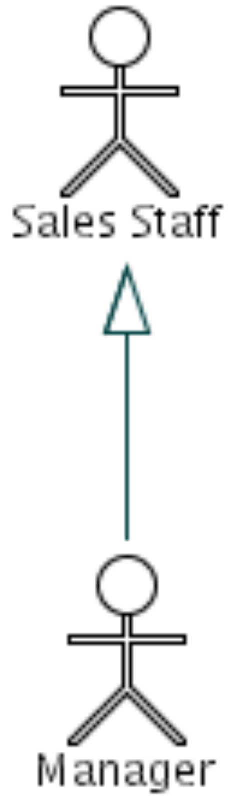


Anything wrong?

Use Case Relationships in UML

- Generalization
  - One actor is a special kind of another actor (specialized actor)
  - One use case is a special case of another use case (specialized use case)
- <<include>>
  - One use case invokes the other use case
  - Included use case represents common behavior
  - Represents a common behavior among several use cases
  - Decompose complicated use case
  - Centralize common behavior
- <<extend>>
  - One use case is a variation of the other
  - Extending use case adds behavior
  - Use cases representing exceptional flows can extend more than one use case.

Generalization of Actors

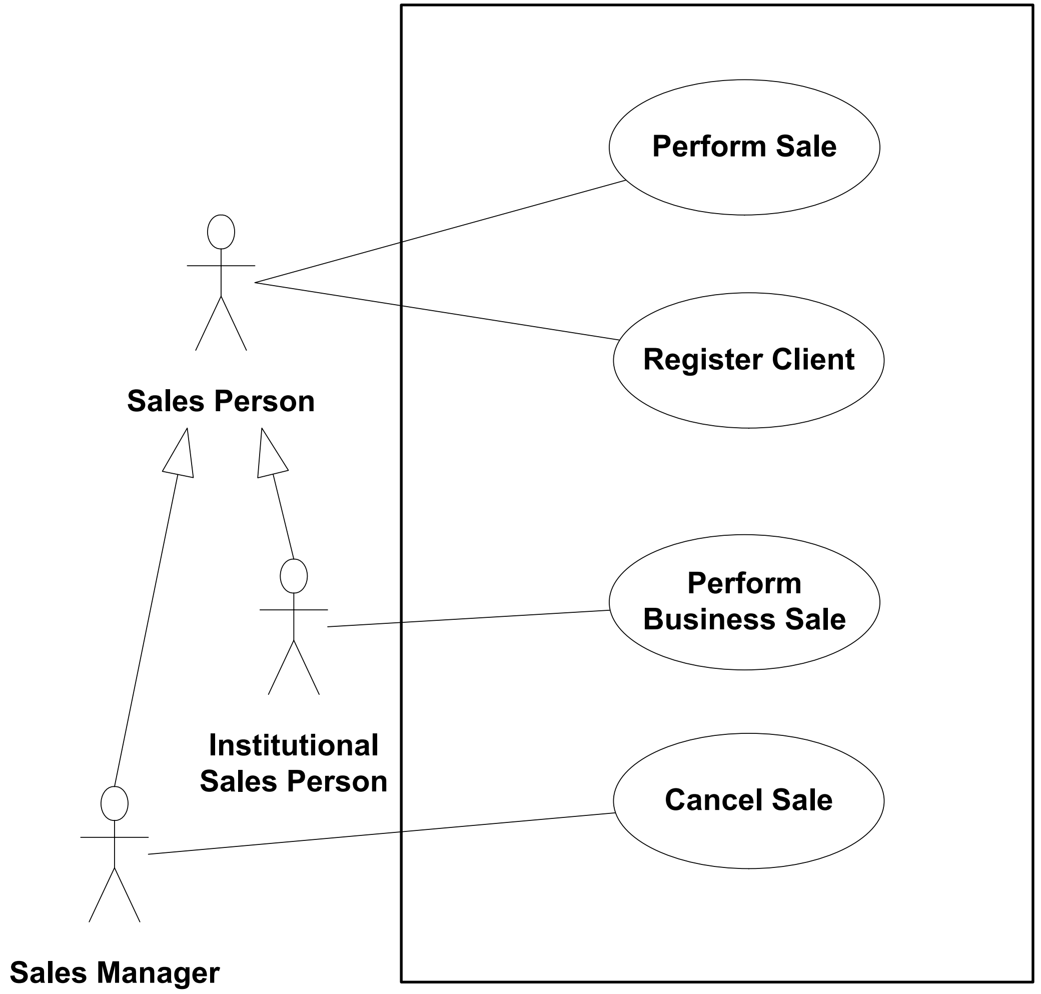


- One actor can be a specialization of another.
- Arrow points to the more general (base) actor.

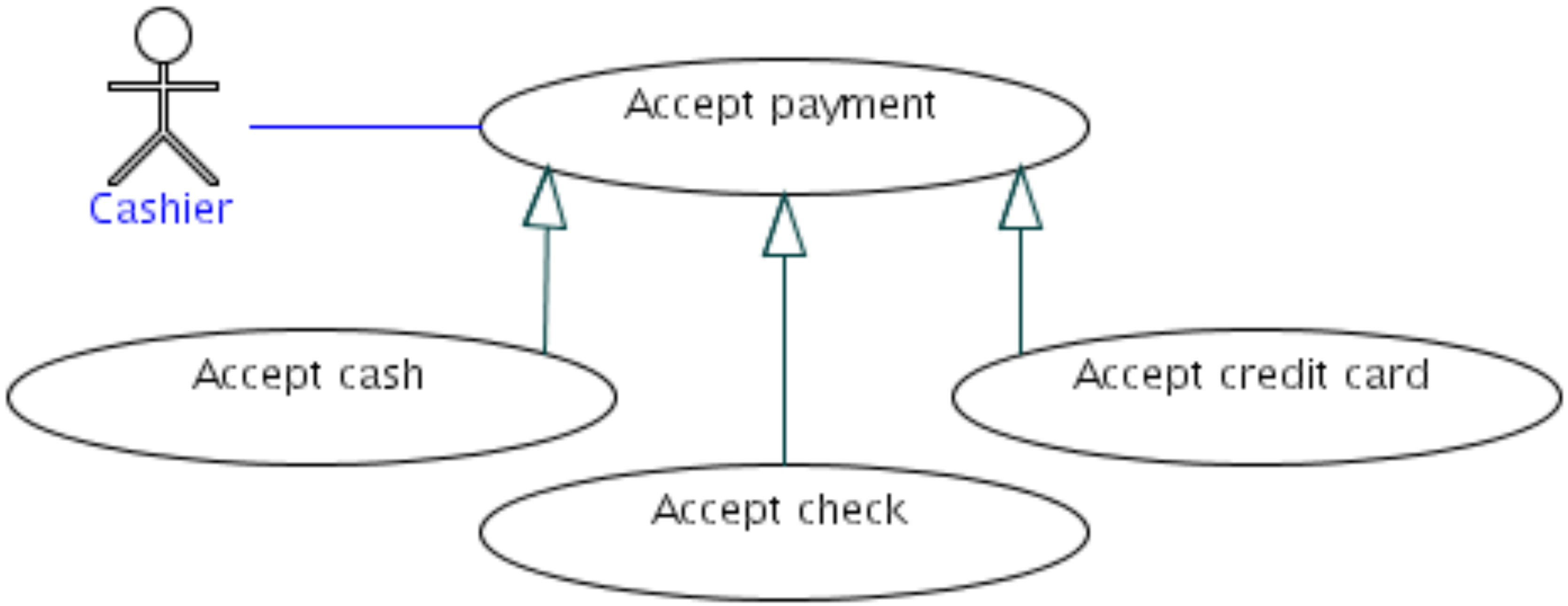
Example: Generalization of Actor

•The child actor inherits all use-cases associations

Should be used if (**and only if**), the specific actor has more responsibility than the generalized one (i.e., associated with more use-cases)

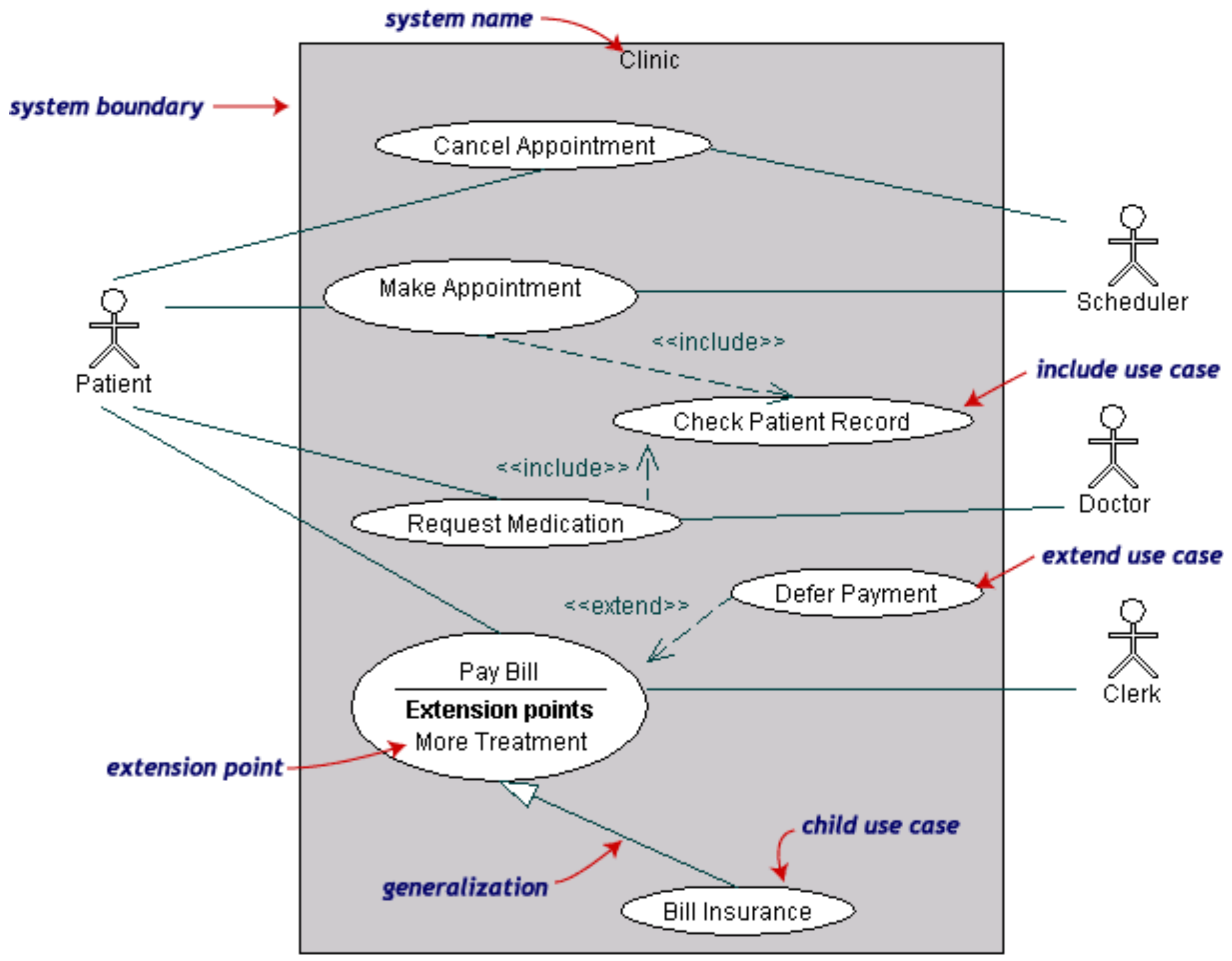


Use case Generalization

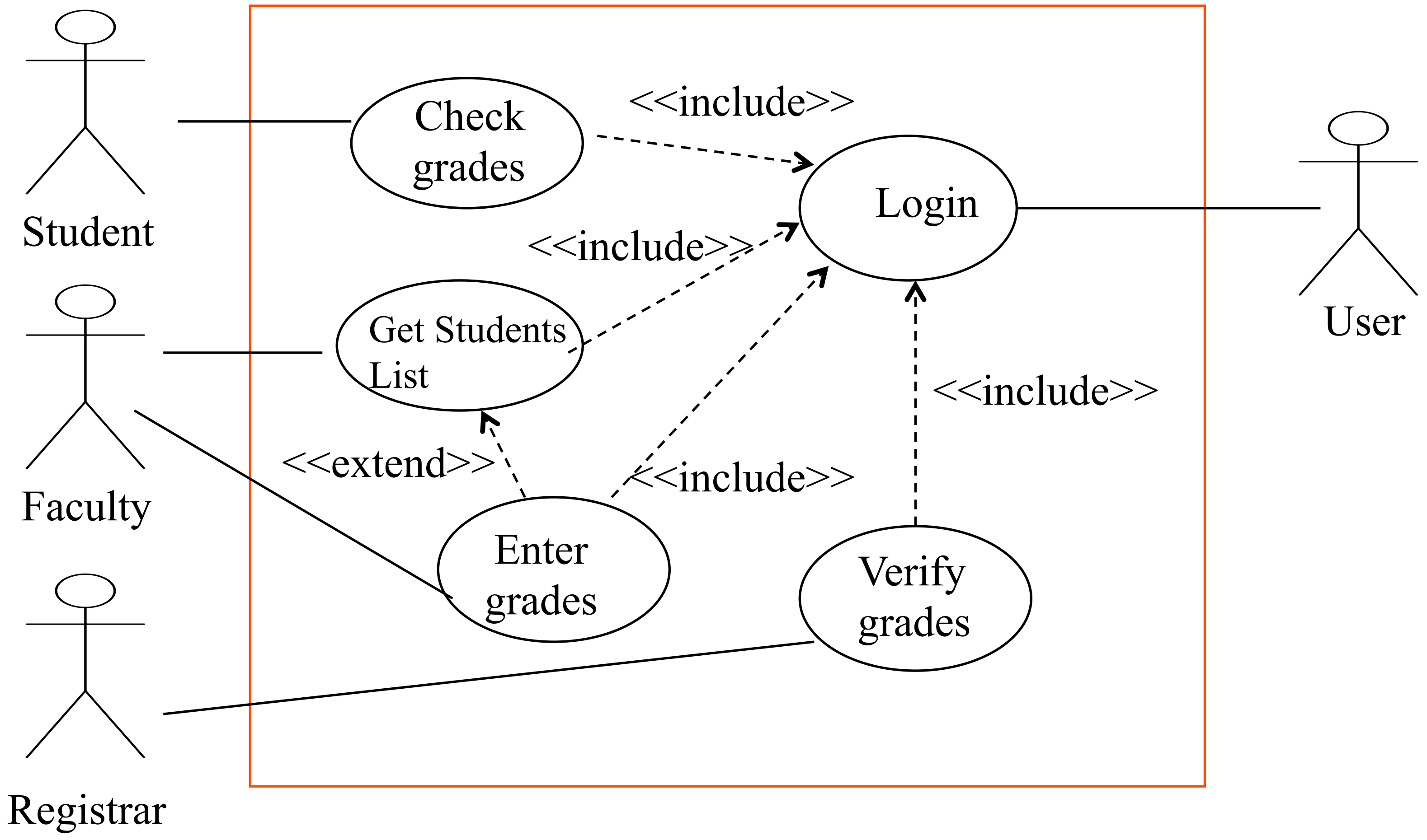


- One use case is simply a special kind of another
- Shows inheritance and specialization. The child use case inherits:
  - The interaction (described in the textual description)
  - Use case links (associations, include, extend, generalization)

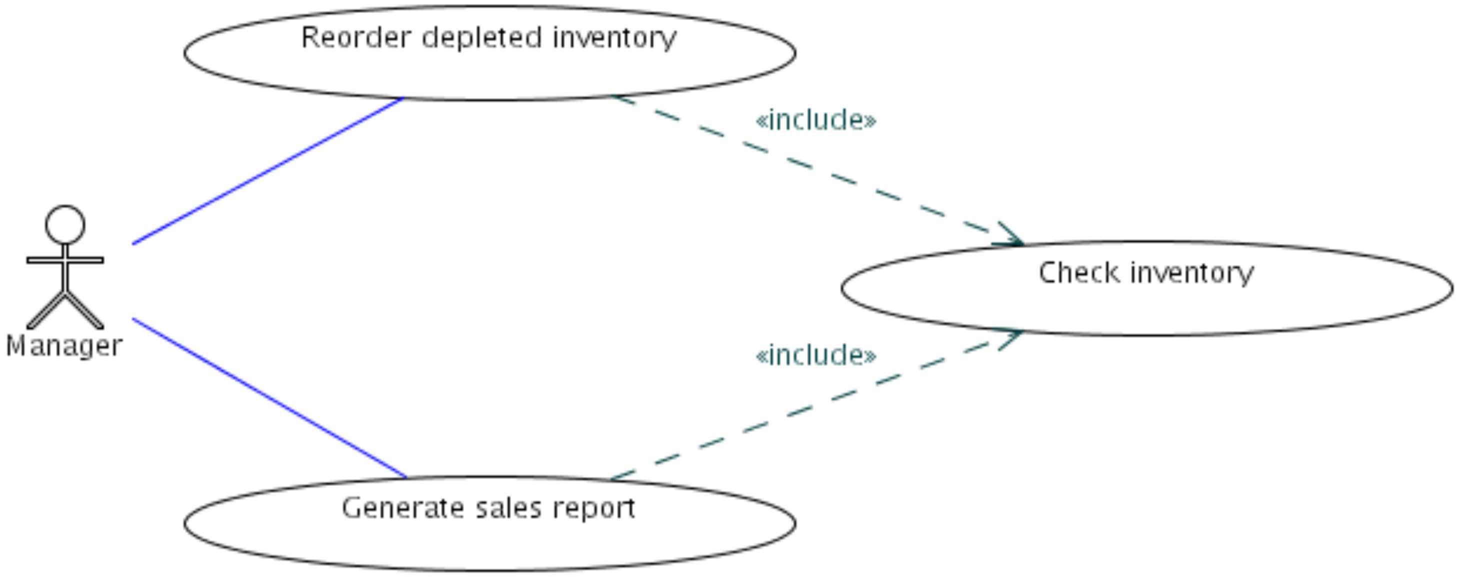
- Generalization = one is a special kind of the other
- Includes = **one invokes the other**
- Extend = **one is a variation of the other**



Example: Use Case Relationships

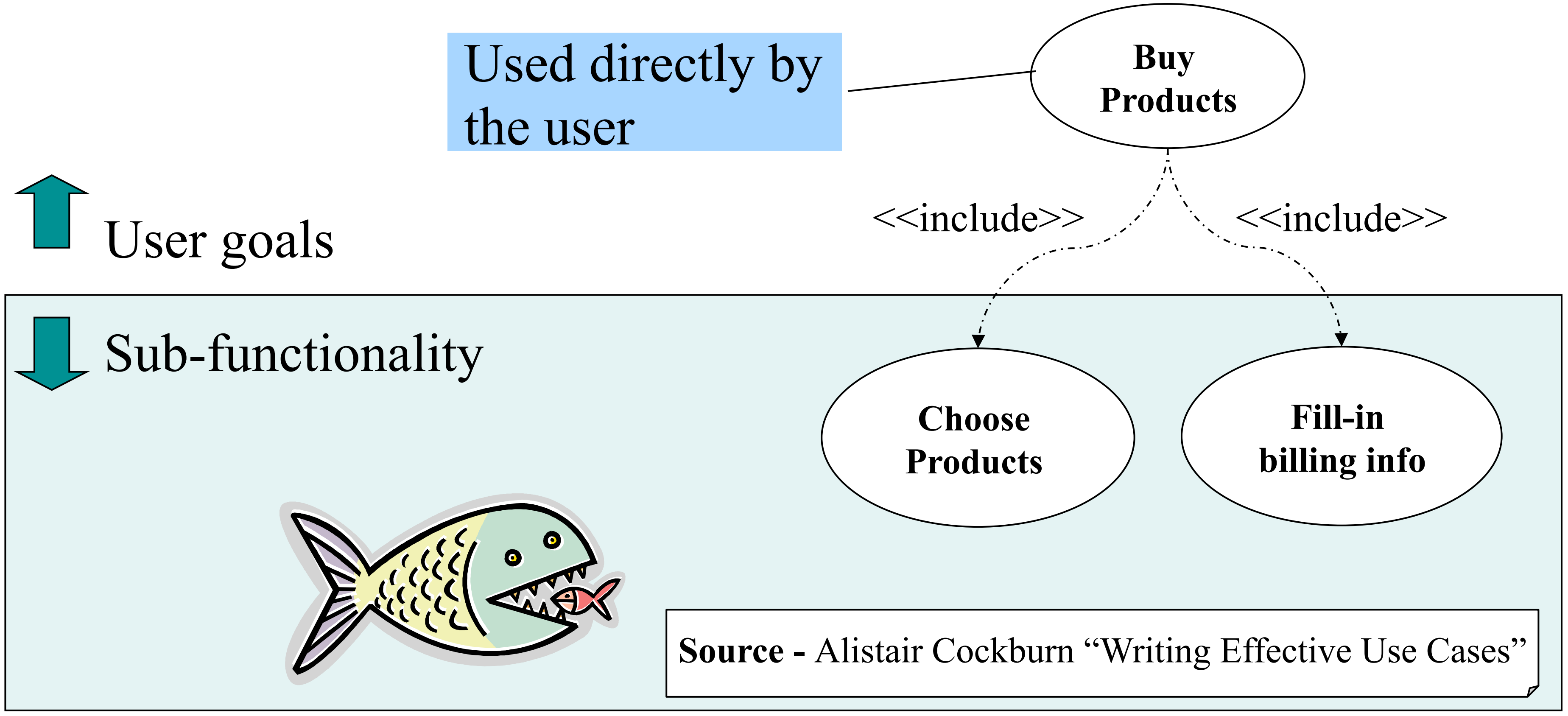


The *<<include>>* Relationship



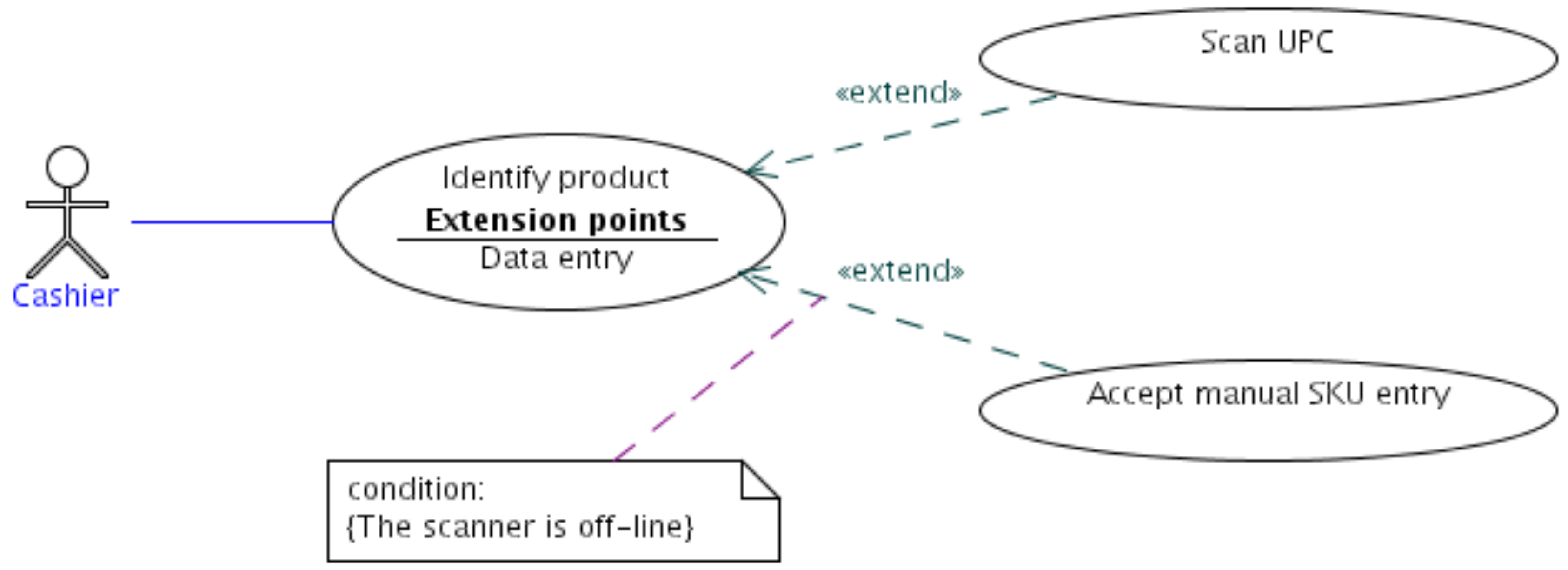
- `<<include>>` relationship represents a **common behavior among several use cases**
  - Decompose complicated use case
  - Centralize common behavior

The <<*include*>> Relationship



Source - Alistair Cockburn "Writing Effective Use Cases"

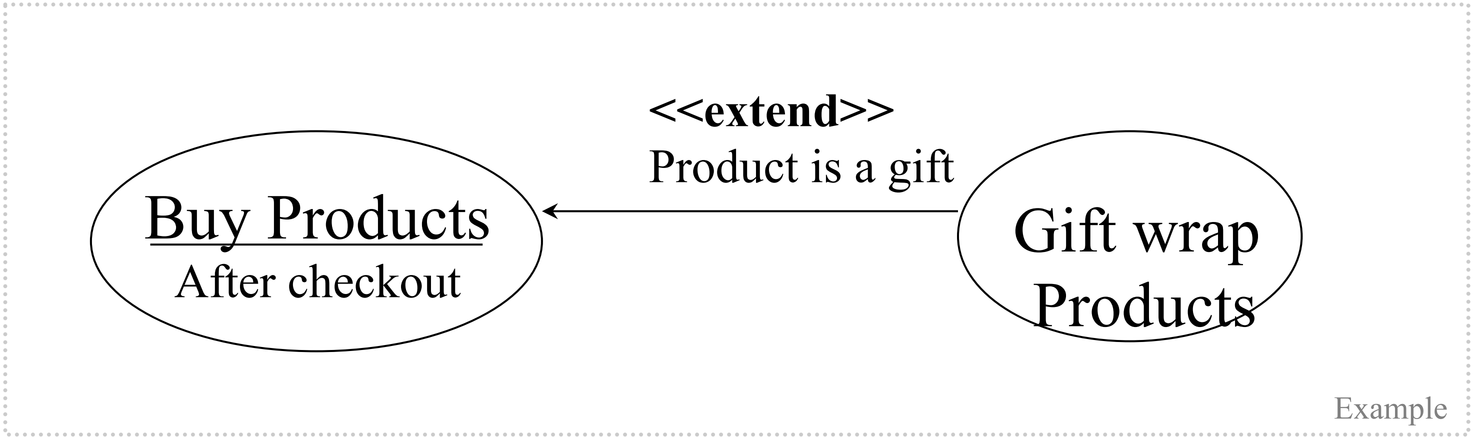
The *<<extend>* Relationship



- <<extend>> relationship represent an **exceptional case**
  - The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extend>> relationship is to the extended use case

Example: <<extend> Relationship

- The base use case can incorporate another use case at certain points, called extension points.
- Note the direction of the arrow
  - The base use-case does not know which use-case extends it



Use Case Diagram: Basic Rules

- A use case must be associated either with an actor or another use case
- An actor cannot be directly connected with another actor unless it is an association relationship (Generalization)
- A use case name must be **active verb**
- An actor name must be generic **noun**
- An actor cannot exist without being associated with at least either one use case or another actor in Generalization relationship.

Use Case Specification: Basic Rules

- There will be no alternative flow for the actor action
- You can define alternative flow ONLY for the system response
- Do not write “The system” beginning of every system response -- start directly with a verb.
- Make actor action and system response compact and concise --do not just copy from the requirement document and paste in the table.
- The flow of system response and actor actions should be in logical order
- Specify the name of the actor in each Actor Action.
- Alternative flow usually starts with “If”
- Use one sequence numbering system for all actor actions and system responses.

Summary of Rules for Actors

- **Give meaningful business relevant names for actors** – For example if your use case interacts with an outside organization its much better to name it with the function rather than the organization name. (e. g.: Airline Company is better than PanAir)
- **Primary actors should be to the left side of the diagram** – This enables you to quickly highlight the important roles in the system.
- **Actors model roles (not positions)** – In a hotel both the front office executive and shift manager can make reservations. So something like "Reservation Agent" should be used for actor name to highlight the role.
- **External systems are actors** – If your use case is send email and if interacts with the email management software then the software is an actor to that particular use case.
- **Actors don't interact with other actors** – In case actors interact within a system you need to create a new use case diagram with the system in the previous diagram represented as an actor.
- **Place inheriting actors below the parent actor** – This is to make it more readable and to quickly highlight the use cases specific for that actor.

Summary of Rules for Use Cases

- **Names begin with a verb** – An use case models an action so the name should begin with a verb.
- **Make the name descriptive** – This is to give more information for others who are looking at the diagram. For example “Print Invoice” is better than “Print”.
- **Highlight the logical order** – For example if you’re analyzing a bank customer typical use cases include open account, deposit and withdraw. Showing them in the logical order makes more sense.
- **Place included use cases to the right of the invoking use case** – This is done to improve readability and add clarity.
- **Place inheriting use case below parent use case** – Again this is done to improve the readability of the diagram.

Summary of Rules for Relationships/ Associations

- There can be 5 relationship types in a use case diagram.
  - Association between actor and use case
  - Generalization of an actor
  - Extend between two use cases
  - Include between two use caGeneralization of a use case
  - ses
- Arrow points to the base use case when using <<extend>>
- <<extend>> can have optional extension conditions
- Arrow points to the included use case when using <<include>>
- Both <<extend>> and <<include>> are shown as dashed arrows.
- Actor and use case relationship doesn't show arrows.

Use Case Specification

Structure of a Use Case Specification

- Name
- Brief Description
- Actors
- Trigger
- Preconditions
- Post conditions
- Normal Scenario
- Alternatives Flows
- Non-Functional (optional)

List of Non-Functional (quality) Requirements (NFRs) that the use case must meet

Example Use case Specification

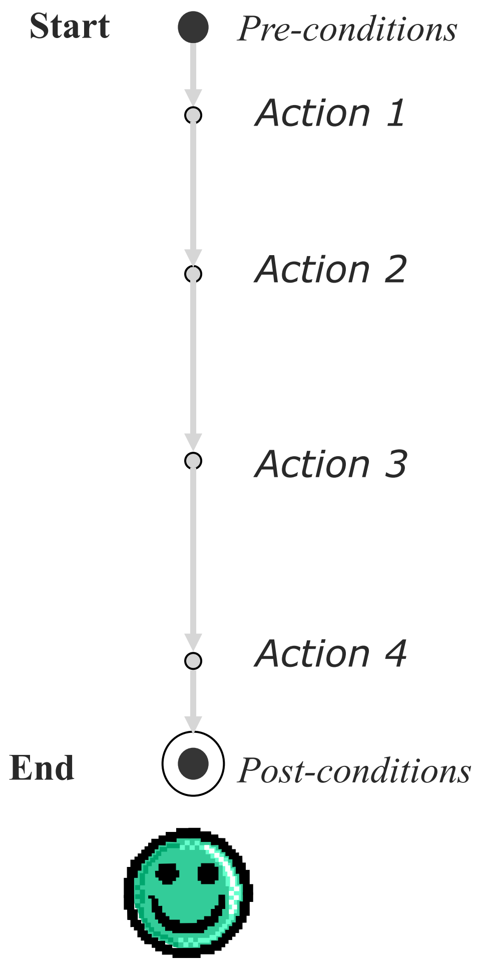
- Name
  - **Withdraw cash**
- Brief Description
  - **Customer withdraws cash from a bank account using an ATM**
- Actors
  - **Customer**
- Trigger
  - **Customer inserts card to ATM**
- Pre-conditions
  - **Actor has a bank card and a valid pin**
- Post-conditions
  - **Account was debited**
- Normal Scenario
  - …
- Alternative flows

Triggers

- What starts the use-case?
- Examples:
  - Customer reports a claim
  - Customer inserts card
  - System clock is 10:00
- Usually the first step of a user case in the trigger event that starts the scenario
  - E.g., customer arrives to a checkout with items to purchase

Writing the Normal Scenario

- The normal scenario is written under the assumption that everything is okay, no errors or problems occur. It describes:
  - Pre-conditions : what must be true before the use case starts
  - *The **interaction** and* what **data are exchanged** between *the actor and the system*
  - The data **validation** performed by the system
  - **State change** by the system (e.g., recording or modifying something)
  - Post-conditions = what will be true upon successful completion



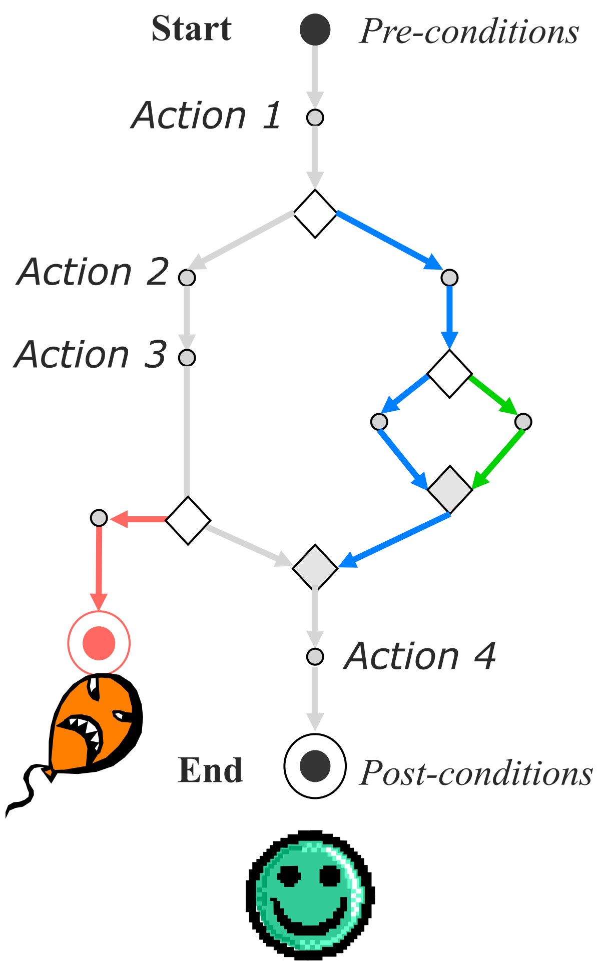
Writing Alternative Flows

- List what can go wrong in the normal flow. e.g.:
  - *course registration closed*
  - *invalid studentID*
- Describe **what to do to handle** the identified exceptions?
  - Sometimes the exception is recoverable i.e., the **alternative flow rejoins the normal flow**

  e.g., if the course is full the system can display alternative courses then the normal flow resumes

  - Or the exception could be non-recoverable and ends the use case

  e.g., if the registration is close, display a message and the use case ends.



Steps to Develop Use Case

Use Case Writing Process:

1. Identify **actors** and their **goals**
2. Write the **normal scenario**
3. Identify and list **possible failure conditions** that could occur
4. Describe **how the system handles each failure**
5. **Revise**

- Start out at a high level and add detail as you go
- It is an iterative, incremental process

An Exercise

Exercise: Use Case Diagram

The QU wants to computerize its registration system

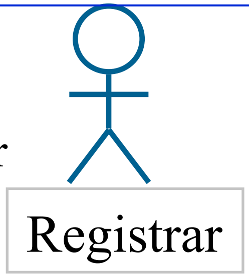
- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course class lists after students register
- Users of the registration system are assigned passwords which are used at logon validation

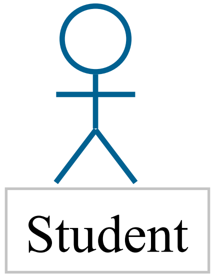
Actors in Use Case Diagram

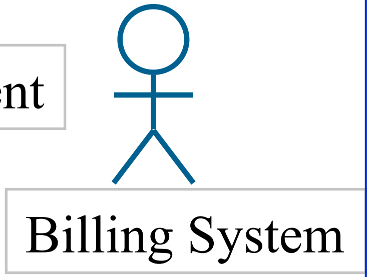
- An actor is someone or some thing that must interact with the system under development

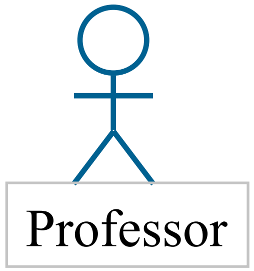
The QU wants to computerize its registration system

- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course class lists after students register
- Users of the registration system are assigned passwords which are used at logon validation







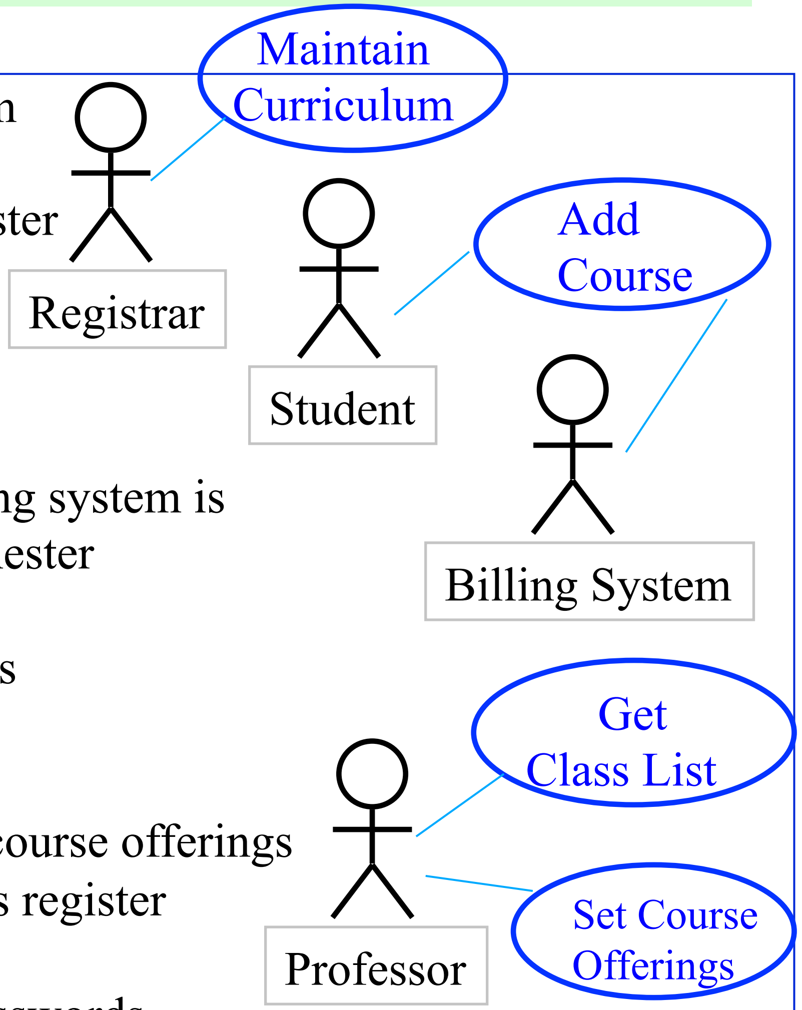


Use Cases

A use case is a sequence of interactions between an actor and the system

The QU wants to computerize its registration system

- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course class lists after students register
- Users of the registration system are assigned passwords which are used at logon validation



Formal Way to Document Normal Scenario in a Use Case

Actor-System Use Case Specification Table

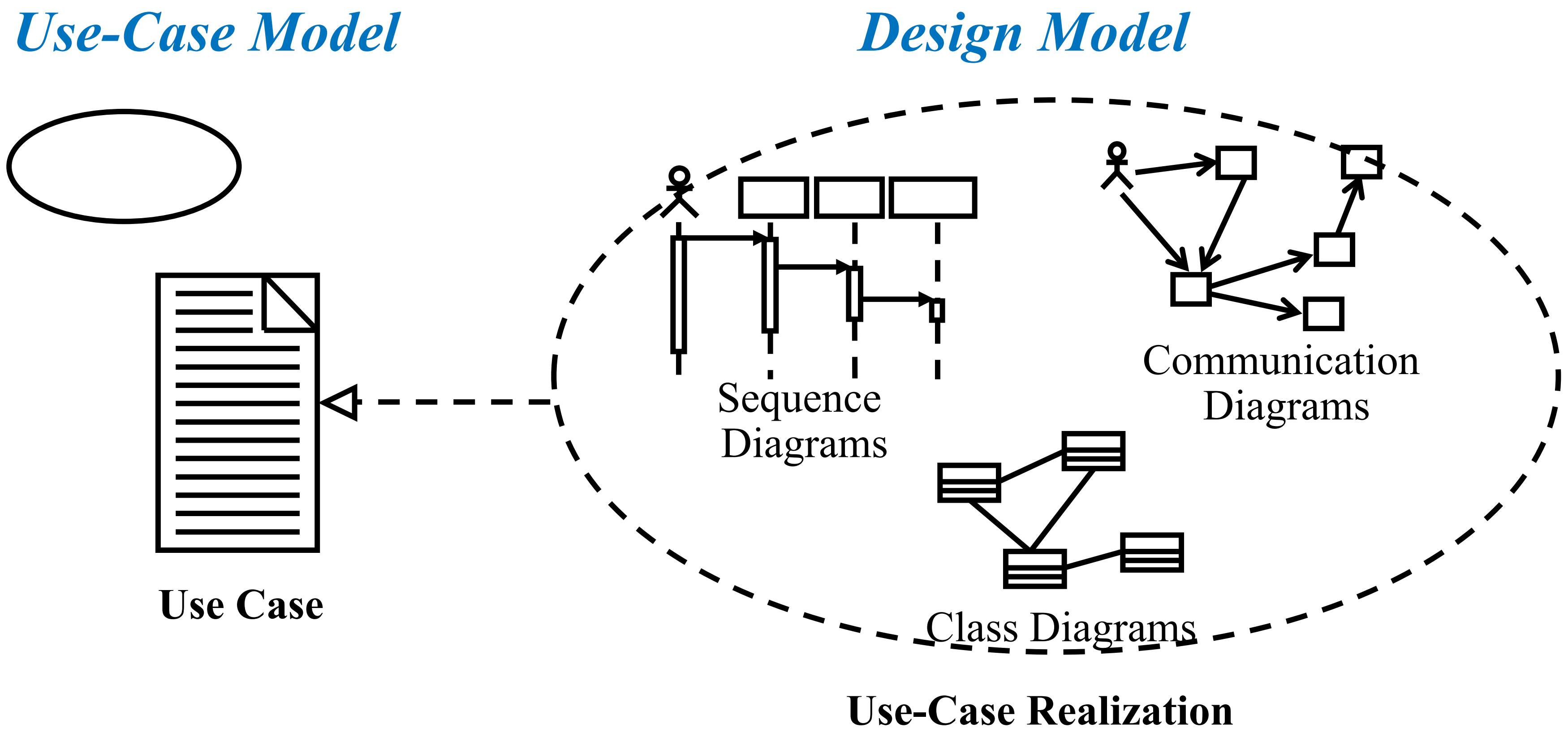
Actor Action	System Response
1. This use case begins when a customer arrives at a POST checkout with items to purchase	
2. The Cashier records the identifier from each item, if there is more than one of the same item the Cashier can enter the quantity as well	3. Determine the item price and add the item information to the running sales transaction
4. On completion of the item entry, the cashier indicates to the POST that the item entry is complete	5. Calculate and present the sale total
6.Cashier tells the Customer the total	
7. The Customer gives a cash payment, the cash tendered, possibly greater than the sale total	
8. The Cashier records the cash received amount	9. Show the balance due back to the customer, Generates a receipt
10. The Cashier deposits the cash received and extracts the balance owing, gives to Customer	
	11. Log and complete the sale

Actor-System Use Case Specification Table

Actor Action	System Response
1. Student arrives at admin office and choses a tutorial from the available list	
2. Student requests allocation to choice of tutorial from Admin staff	
3. Admin Assistant enters Student Name and Number into Tutorial List	4. Check vacancy and record student in tutorial
5. Student leaves allocated to tutorial	

4a. If there is no vacancy, ask to choose another tutorial

Use Cases Drive the Development Process



Common Mistakes with Use Cases

Mistake

- Represent individual steps
  - print report
  - read data
  - total amounts
- Represent individual operations
  - check user id
- Represent individual transactions
  - buy items with cash
  - buy items on credit
  - buy items by cheque

Fix

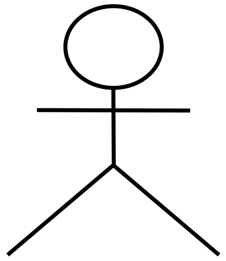
- Include in the use case
- Include in the use case
- Use the «include» relationship

In the first instance, always check the use case starts and ends at the system boundary.

Common Use Case pitfalls

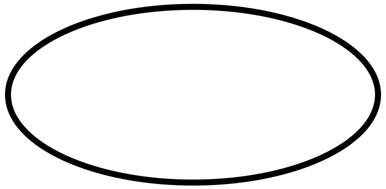
- Unclear system boundary
- The Use Case is written from the system view (e.g., describing how and not what)
- The actor names are inconsistent
- There are too many Use Cases
- The use-case scenarios are too long or confusing
- Incorrect description of the Use Case functionality
- The customer doesn't understand the use cases

Checkpoints: Actors



- Have all the actors been identified?
- Is each actor involved with at least one use case?
- Is each actor really a role? Should any be merged or split?
- Do two actors play the same role in relation to a use case?
- Do the actors have intuitive and descriptive names? Can both users and customers understand the names?

Checkpoints: Use-Cases



- Is each use case involved with at least one actor?
- Do the use cases have unique, intuitive, and explanatory names so that they cannot be mixed up at a later stage?
- Do any use cases have very similar flows of events?
- Do customers and users alike understand the names and descriptions of the use cases?

Summary

- UML introduction
- Use Case diagram
  - <<include>>, <<extend>>
- Use Case Specification
- Pre-condition, post-condition
- Normal scenario
- Alternative flows
- Actor-System Use Case Specification Table