

Lecture 4: Capturing User Requirements

Software Requirement Modeling with Use-cases



Outline

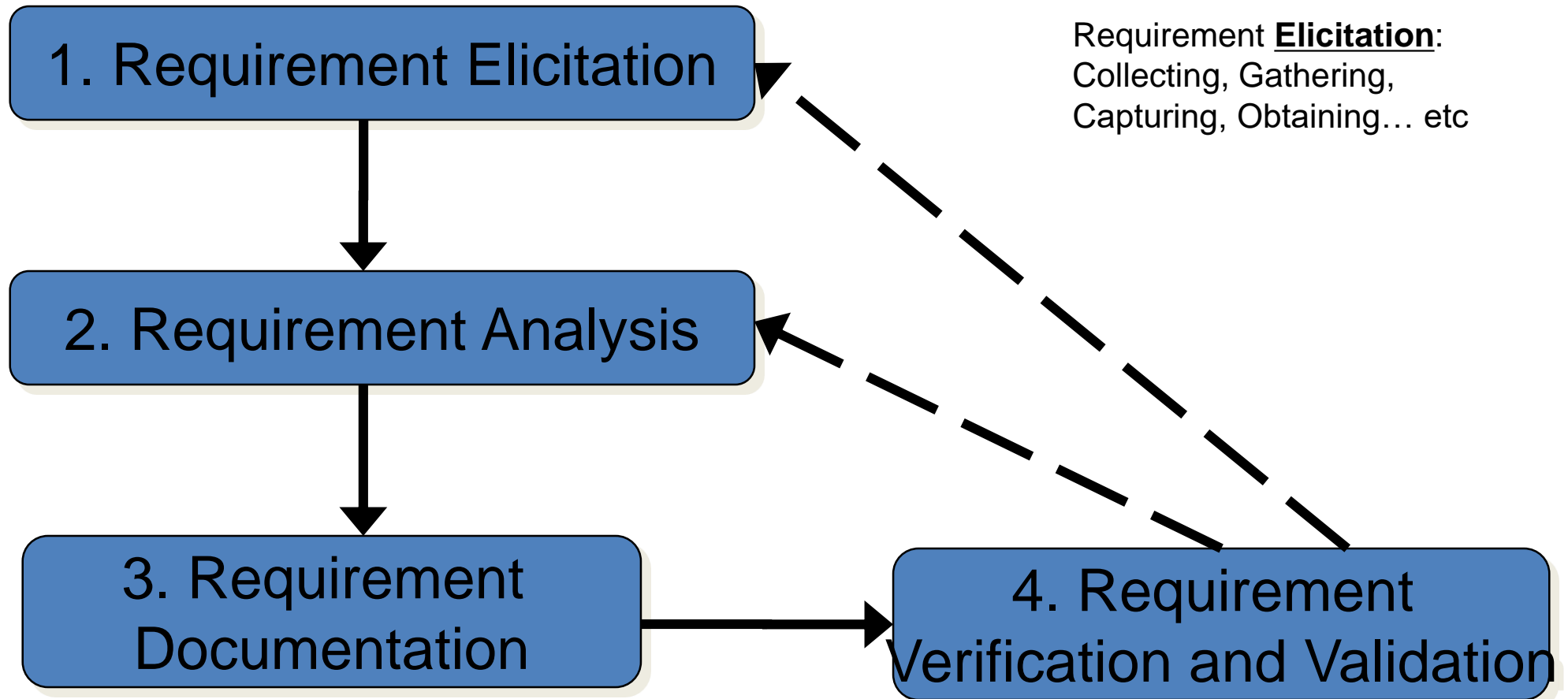
- **Introduction**
- **Use Case Diagrams**
- **Writing Use Cases**
- **Guidelines for Effective Use Cases**

Where are we if in a project?

Phase	Actions	Outcome
Initiation & Feasibility Study	Raising a business need for a software project, Risk Analysis	Initial documents, such as proposal
Requirements Elicitation	Interviewing stakeholders, exploring the system environment	Organized documentation
Requirements Analysis	Analyze the engineering aspect of the system, building system concepts	Formal specification
Design	Define architecture, components, data types, algorithms	Formal Specification
Implementation	Program, build, unit-testing, integrate, documentation	Testable system
Testing & Integration	Integrate all components, verification, validation, installation, guidance	Testing results, Working sys
Maintenance	Bug fixes, modifications, adaptation	System versions

Requirements Engineering

The Process Outline



Requirements vs. Design

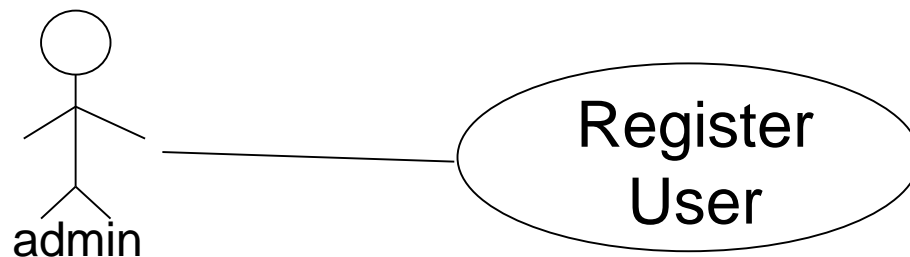
- Requirements:
 - **What** the system should do
 - More abstract



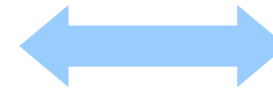
-
- Design:
 - **How** the system should do it
 - More detailed



Use Cases – visualize functional interactions



Use Case Diagram



Use Case Specification

- Typically, a use case is a **contract** of an interaction between the system and an actor.
- The Actor can be a human or an external system / machine.
- A full use-case model comprise of:
 - A Use Case Diagram, describing relations between **use-cases** (system tasks, roles) and **actors** (external parties).
 - Use Case Specifications (usually one per each use case)
 - Documents describe the use case in details

Use Cases as Means of Communication



Customers



Designers



Users

The use case should stimulate a discussion about **what** the system should do, mainly with people who are outside of the development team.

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

- ... how users interact with this system

- ... when trying to achieve their objectives.

“It is one of the key activities in requirements analysis”

Use Case Diagram Objective

1. Create a semi-formal model of the functional requirements (What the system does...)
2. Analyze and define:
 - Scope (What are the main functions?)
 - External parties (who will interact with the system?)
 - External interfaces (how they are related?)
 - Scenarios and reactions (One Big Picture)

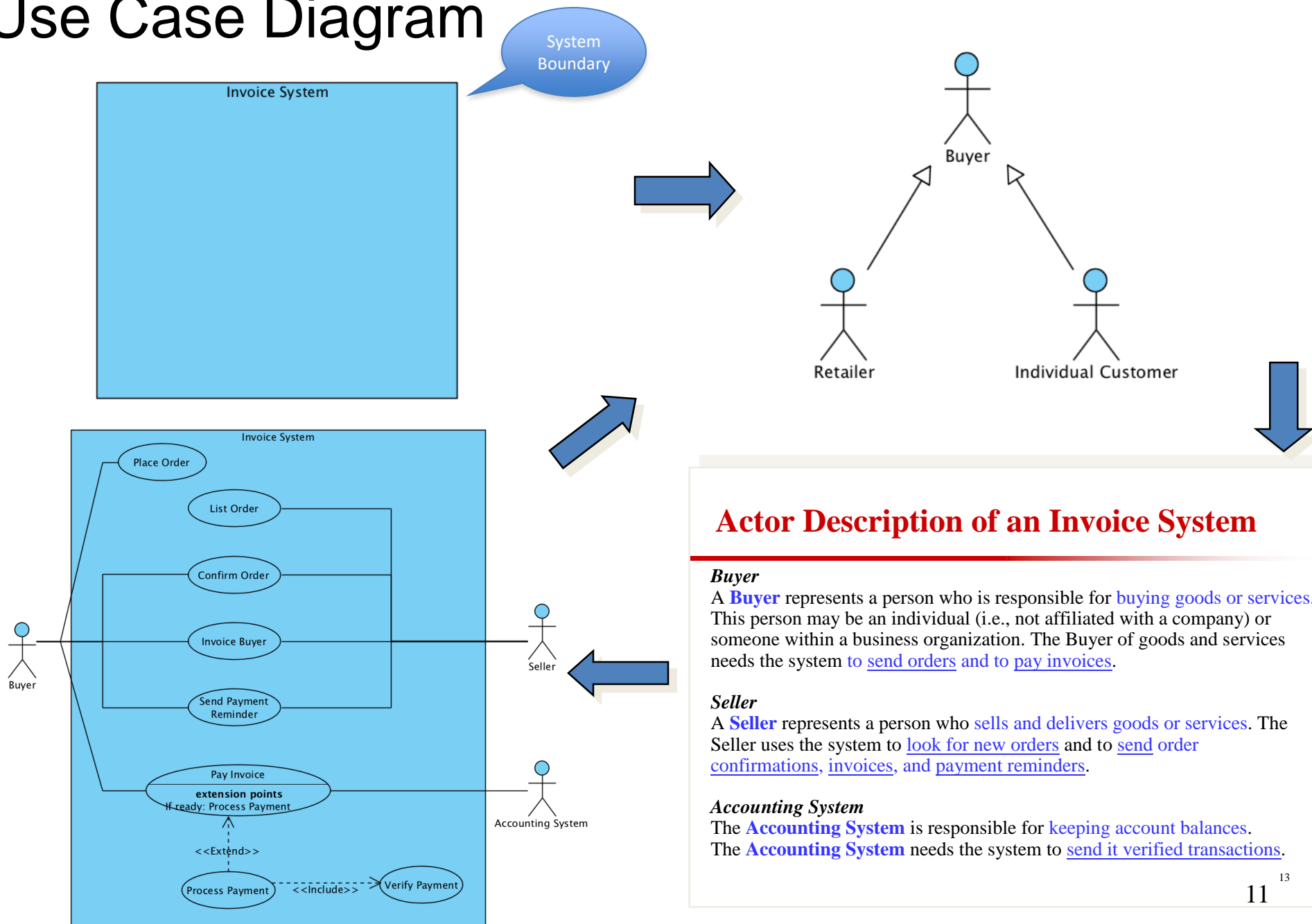
Outline

- **Introduction**
- **Use Case Diagrams**
- **Writing Use Cases**
- **Guidelines for Effective Use Cases**

Steps to Build an Use-Case Model

- Choose the **system boundary** – what are you modeling? System? Business organization?
- Identify the **primary actors** – they have user goals fulfilled by using the services of the system
- For each primary actor, **identify their user goals** – define what they want to do with the system. Describe their goals at the correct level
- **Define use cases** that satisfy user goals. Name each according to its goal. [You may find out other actors, which we call secondary actors of **that particular** use case.]

A Use Case Diagram



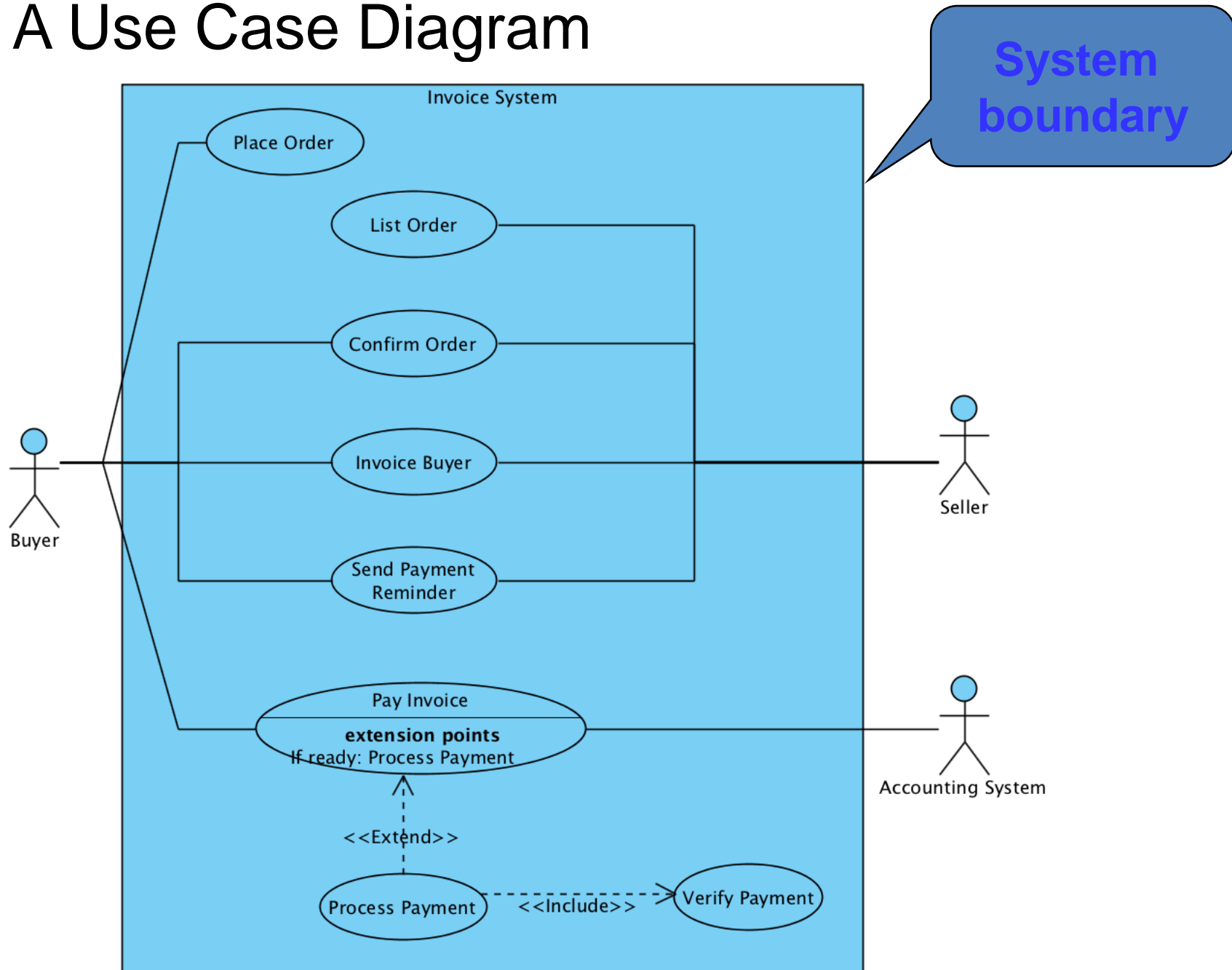
Actor Description of an Invoice System

Buyer
A **Buyer** represents a person who is responsible for buying goods or services. This person may be an individual (i.e., not affiliated with a company) or someone within a business organization. The Buyer of goods and services needs the system to send orders and to pay invoices.

Seller
A **Seller** represents a person who sells and delivers goods or services. The Seller uses the system to look for new orders and to send order confirmations, invoices, and payment reminders.

Accounting System
The **Accounting System** is responsible for keeping account balances. The **Accounting System** needs the system to send it verified transactions.

A Use Case Diagram

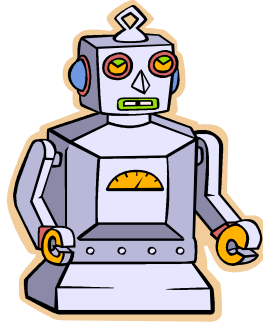


Finding Actors

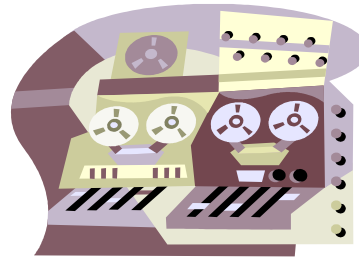
- External objects that produce/consume data:
 - Must serve as sources and destinations for data
 - Must be external to the system



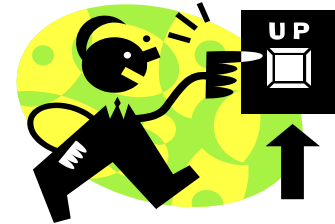
Humans



Machines



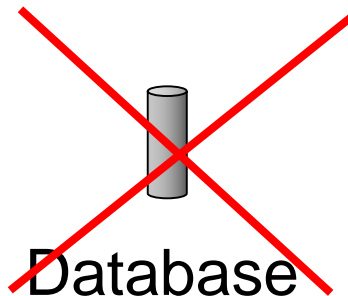
External
systems



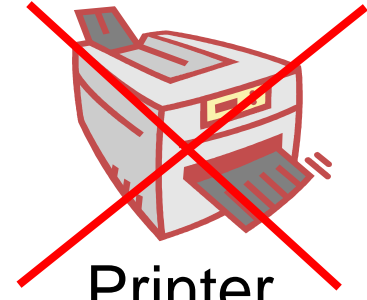
Sensors



Organizational Units

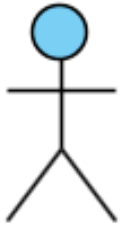


Database



Printer

Finding the Actors



Actor Name

- human/system outside the system that interacts with the system
- provides input to or takes output from the system
- a role the user can play ☐ multiple roles possible

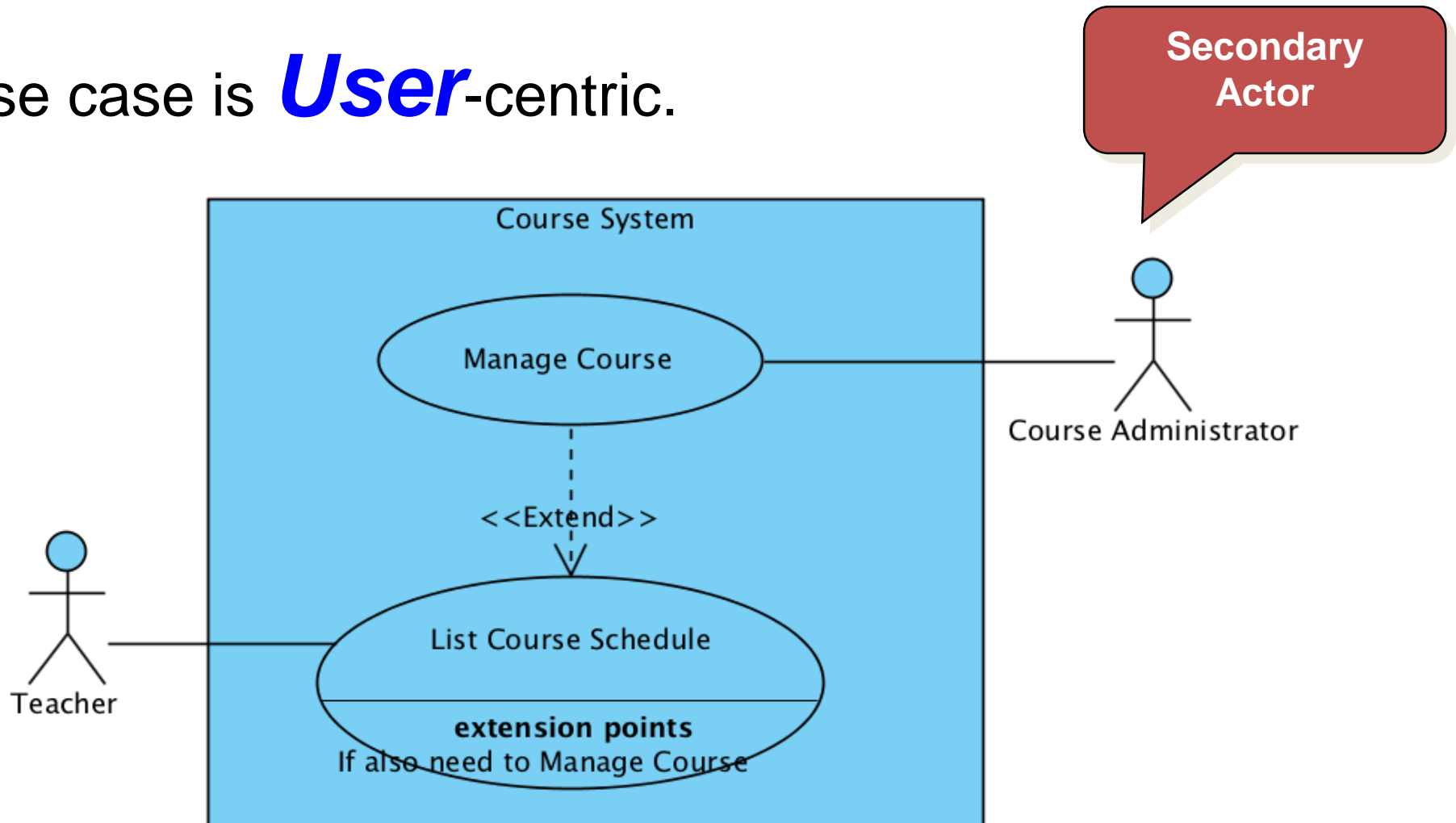
Primary actor - interacts directly with the system

Secondary actor - interacts indirectly with the system or support the use case to complete a use case for the primary actor

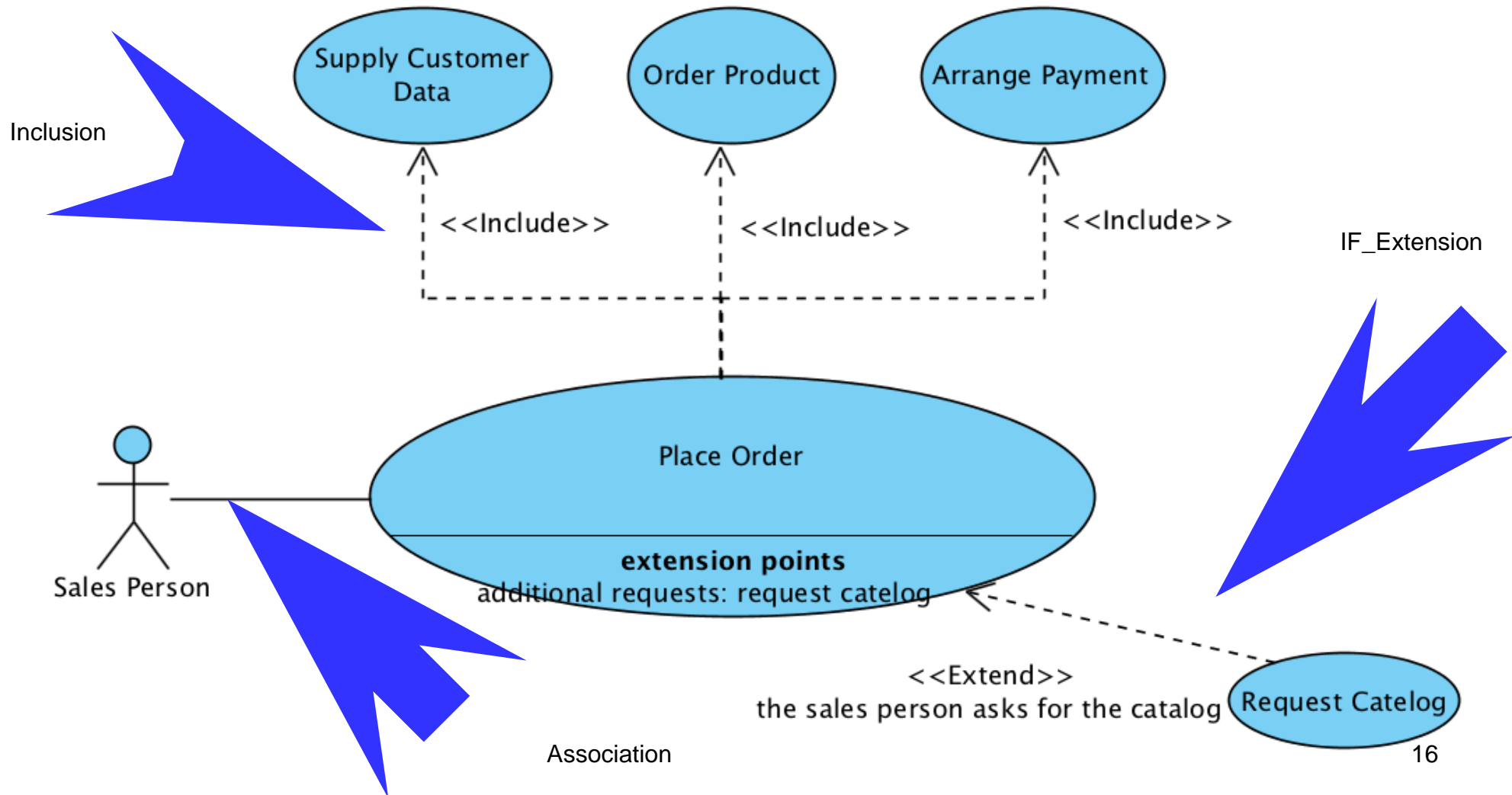
Briefly describe the role each actor plays when interacting with the system

Actors at *Use Case* Level

- Use case is *User*-centric.



Three Basic Types of Relationship in Use Case Model



Association Relationship

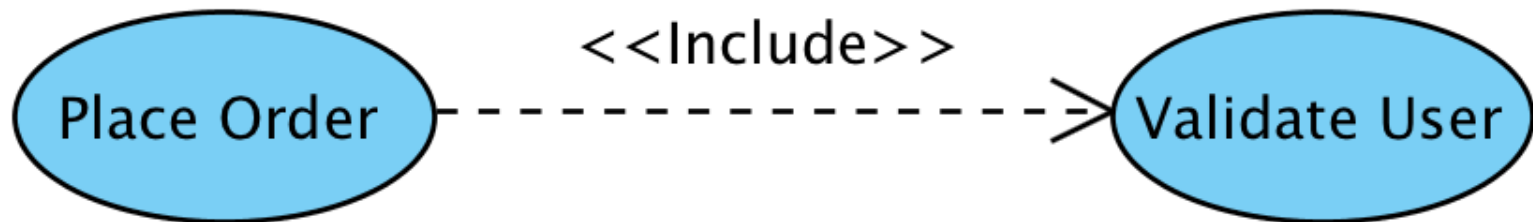
- ***Association***
 - Instances of the actor and instances of the use case communicate with each other
 - The **only** relationship between actors and use cases

<<Include>> Relationship

- Allow one to express **commonality** between several different use cases.
- Can be included in other use cases
 - Even very different use cases can share sequence of actions.
 - Enable you to avoid repeating details in multiple use cases.
- Shows the performing of a *lower-level task* with a lower-level goal.

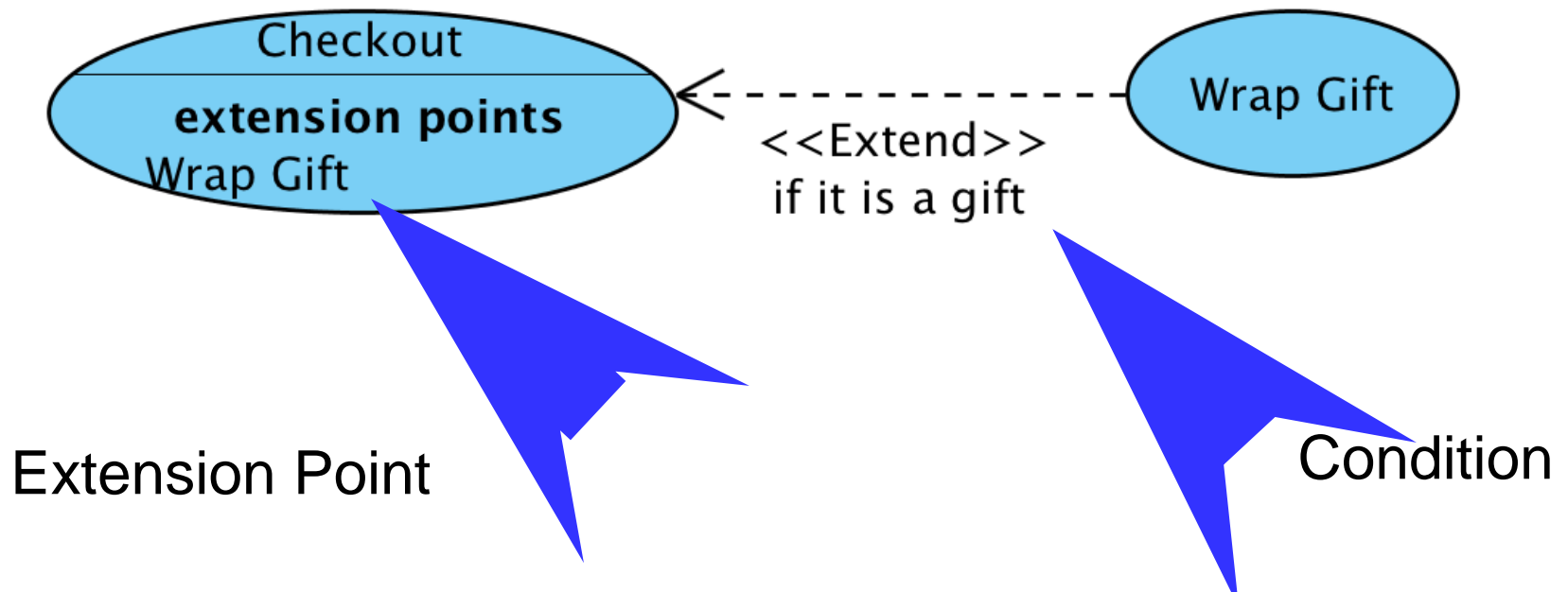
More Example

- “*place order*” includes “*validate user*”



Extend – Graphical Representation

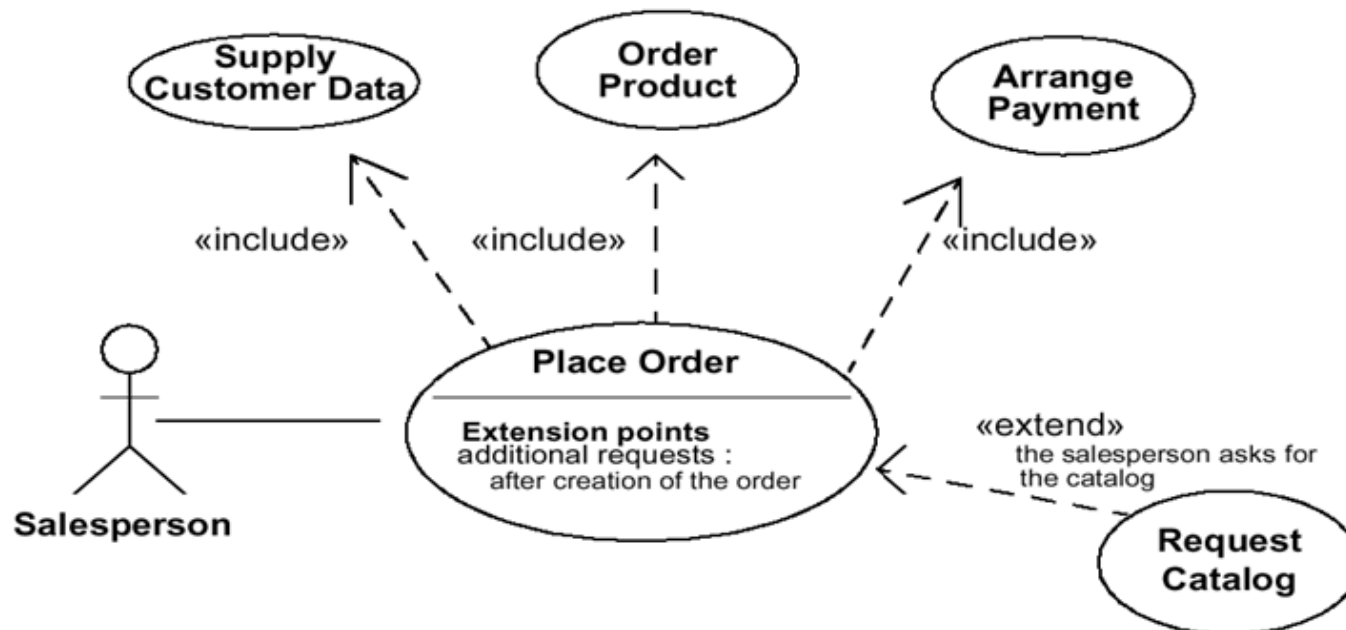
- The base use case can incorporate another use case at certain points, called extension points. (IF... THEN...)
- Here means: In “Checkout” use case,
IF “the item is a gift”, THEN perform “Wrap Gift”



<<Extend>> Relationship

- **Extend**

- Use it to show **optional** behavior explicit or to handle **exception** scenarios
- Keep the description of the basic use case simple.

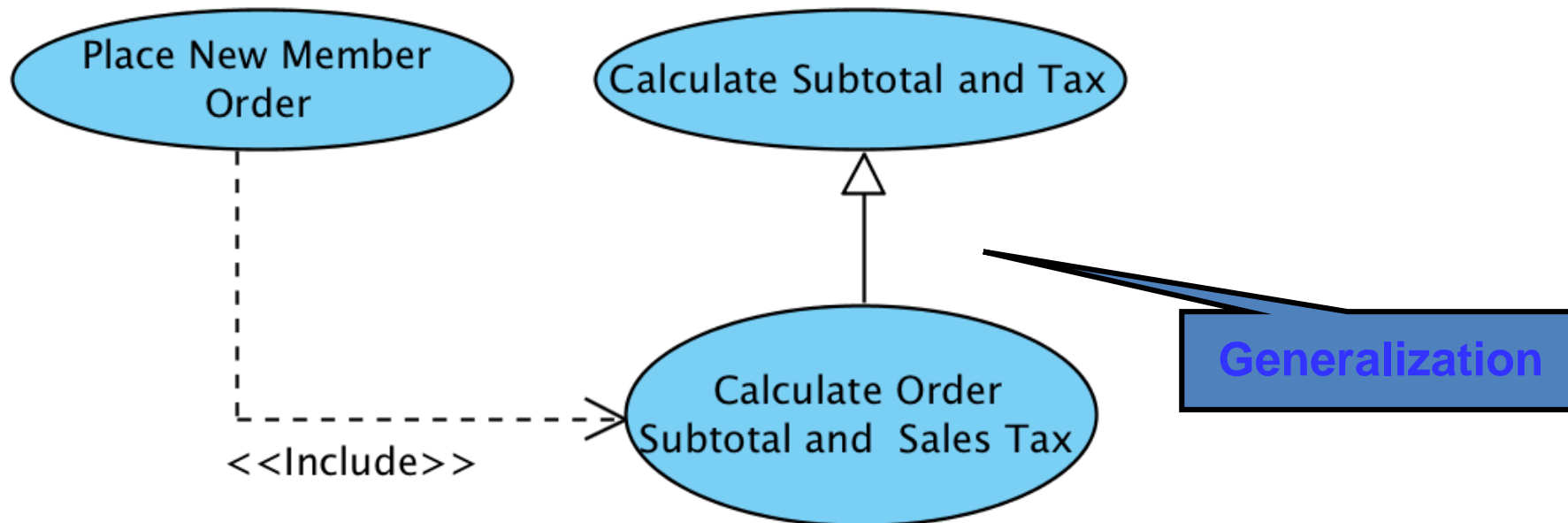


Yet another relationship:

Generalization

- **Generalization (Inheritance)**

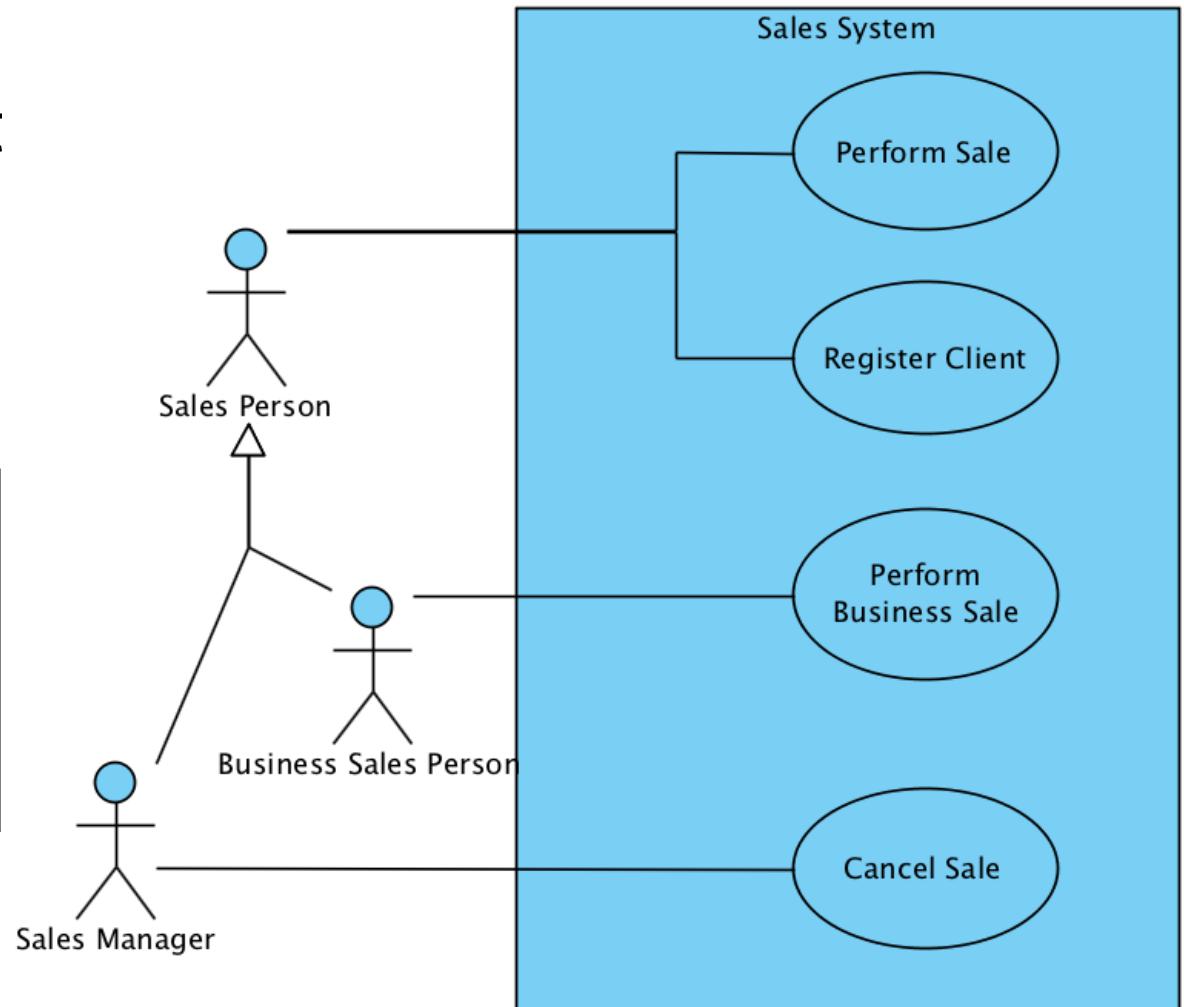
- A generalization **from** use case **A** **to** use case **B** indicates that **A inherits B**.



Actors can also be generalized

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)



Outline

- **Introduction**
- **Use Case Diagrams**
- **Writing Use Cases**
- **Guidelines for Effective Use Cases**

Structure of a Use Case Specification

Name

Actors

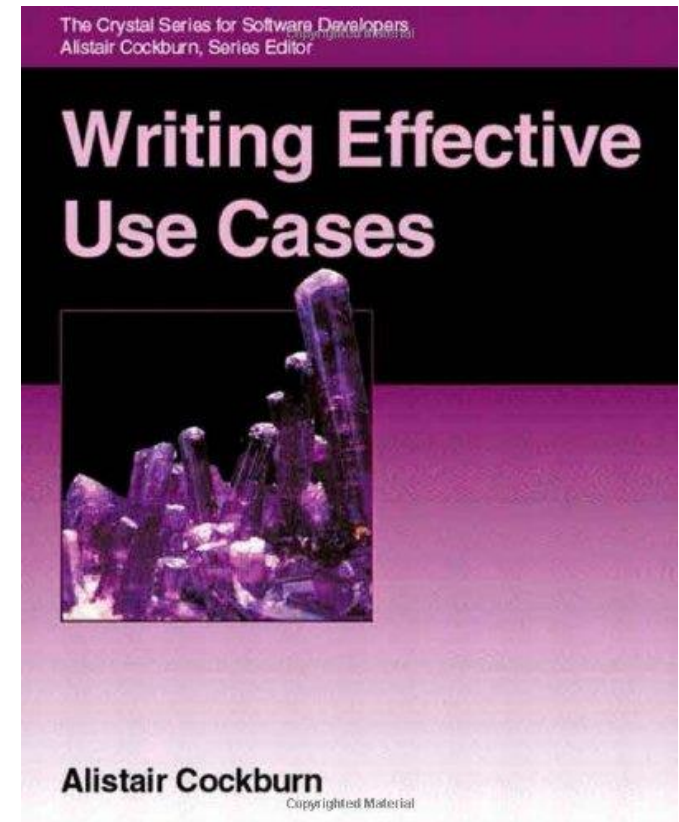
Trigger

Preconditions

Post conditions

Success Scenario

Alternatives flows



Alistair Cockburn :
“Writing Effective Use Cases”

Contents in a single use case?

- **Name:** Give a short, descriptive name to the use case.
- **Purpose:** State clearly the purpose of the use case. Give a short informal description.
- **Actors:** List the actors who can perform this use case.
- **Pre-conditions:** Describe the state of the system before the use case.
- **Flow of Events**
 - interactions between actors and the system
 - business logic
- **Not allowed paths** (if any)
- **Alternative paths** (if any)
- **Exceptions** (if any)
- **Post-conditions:** State of the system in following completion.
- **Extension Points**

Triggers

- What Triggers/Starts the use-case?
- Examples:
 - Customer reports a claim for refund
 - Customer inserts bank card
 - System clock is 10:00 (* Time is the Actor)

Pre-conditions

- What the system needs to be true before running the use-case.
 - Examples
 - User account (must) exists
 - If we want to operate on the account
 - User (must) have enough money in her account
 - If we want to withdraw the amount requested
 - There (must) be enough disk space
 - If we want to write data to the disk drive
- ...*before* we can execute the use-case

Post-Conditions

- A post-condition is the **outcome condition** of the use-case.
- Examples
 - Money was transferred to the user account (completed, past sentence)
 - User is logged in (completed, past sentence)
 - The file is saved to the hard-disk (completed, past sentence)
- **Minimal guarantee** (minimum post condition)
 - The minimal things a system can promise, holding even when the use case execution ended in failure (completed in failure, but completed)
 - Examples: Money is not transferred unless authorization is granted by the user.
- **Success guarantee** (full post condition)
 - What happens after a successful conclusion of the use-case.
 - Examples: The file is saved; Money is fully transferred.
 - Define success factors/conditions, measurable success factors.

Success Scenario

- The success scenario is the main story-line of the use-case
- It is written under the assumption that everything is okay, no errors or problems occur, and it leads directly to the desirable outcome of the use-case
- It is composed of a sequence of action **steps**
- Example:

1. Administrator enters course name, code and description
2. System validates course code
3. System adds the course to the DB and shows a confirmation message

Interaction step

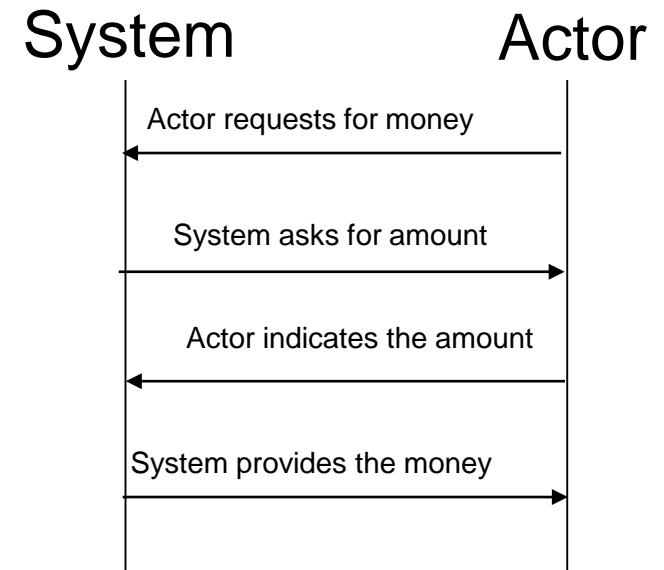
Validation Step

Internal Change Step

(plus) Interaction Step

Guidelines for Effective Writing

- Use simple grammar
- Only one side (system or actor) is doing something in a single step
- Write from an “objective” point of view
- Any step should lead to some progress
 - Bad: “User clicks the enter key”



Steps – cont' d

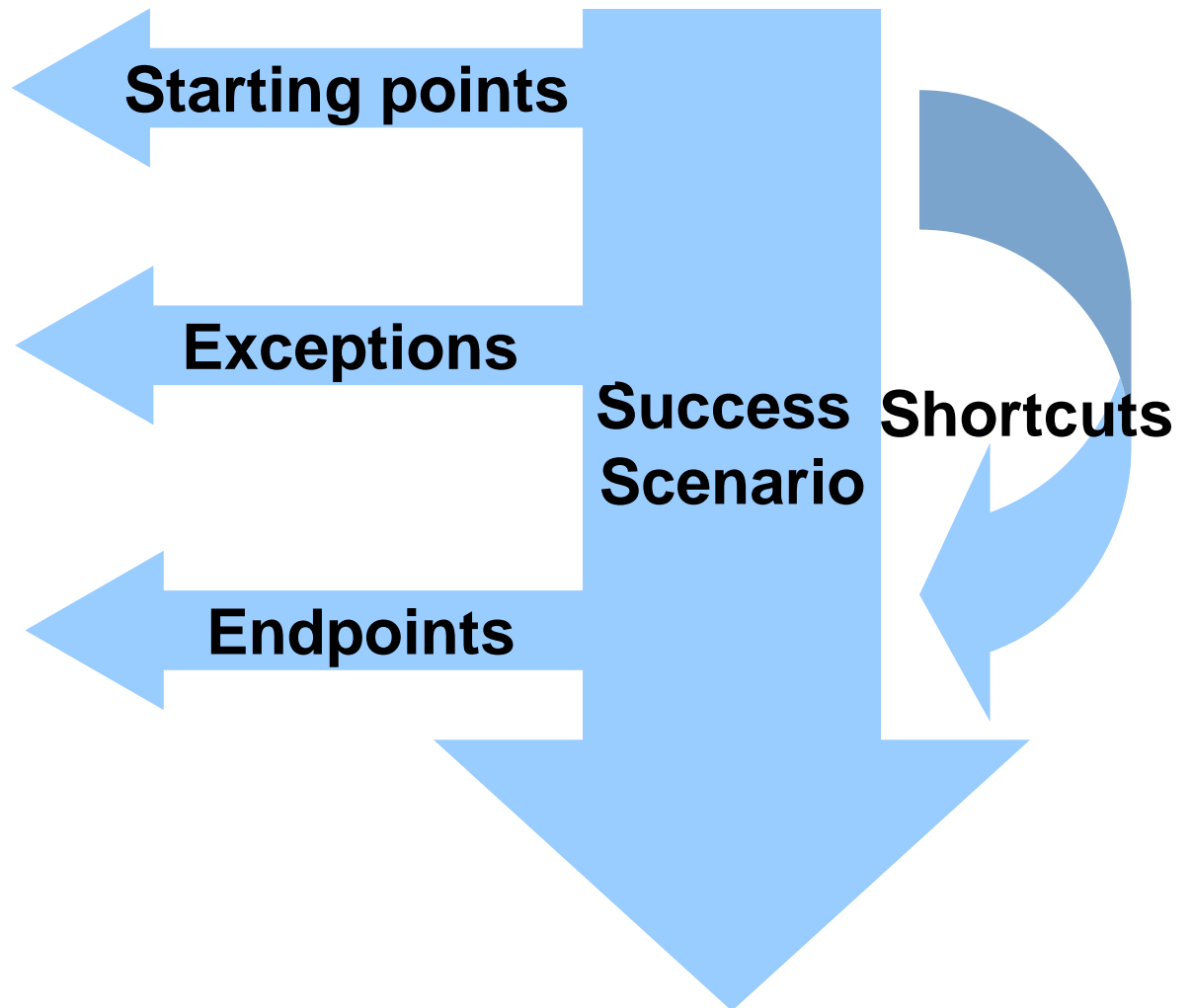
- Branches:
 - **If** an user has more than \$10,000 in her account, the system presents a list of investment options and suggestions.
 - **Otherwise**... do something else.
- Repeats:
 1. User enters the name of the item she wants to buy
 2. System shows the items
 3. User selects items to buy
 4. Systems adds the item to the shopping cart
 5. User **repeats steps 1-4 until** he is ready to check-out!

Use-Cases – Common Mistakes

- Complex diagram (complicated!)
- No system (the boundary box!)
- No actor (huh?)
- Too many user interface details (too high level)
 - “User types ID and password, clicks OK or hits Enter”
- Too much details (too low level)
 - User provides name
 - User provides address
 - User provides telephone number
 - ...

Alternative Flows

- Used to describe exceptional functionality
- Examples:
 - Errors
 - Unusual or rare cases
 - Failures
 - Starting points
 - Endpoints
 - Shortcuts



Alternative Flows - Example

- Errors:
 - “Case did not eject properly”
 - “Any network error occurred during steps 4-7”
 - “Any type of error occurred”
- Unusual or rare cases
 - “Credit card is defined as stolen”
 - “User selects to add a new word to the dictionary”
- Endpoints
 - “The system detects no more open issues”
- Shortcuts:
 - “The user can leave the use-case by clicking on the “ESC” key

Writing Include

- If a base use-case include another use-case, we can add a reference as a step:

1. System shows a homepage

2. User executes “login to the system”

OR

```
<include: login to the system>
```

Writing Extend

- Scenarios do not include direct references
- Instead, they include extension points, such as:

User enters a search string

System shows search results

Extension point: results presentations

OR

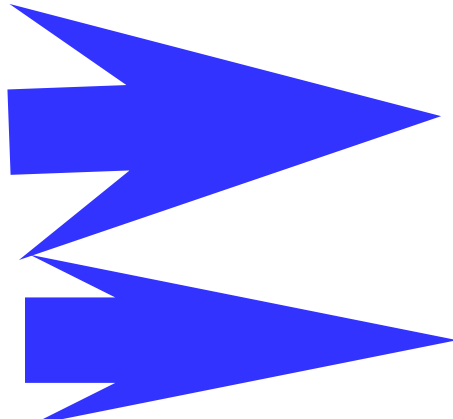
<extension point: results presentations>

- The extension use-case includes conditions in which the extension is being committed
 - Example:
 - If the user belongs to the “rich clients” group, then **perform extended use case.**
 - If additional services are required, then **perform extended use case.**

A More Formal Way to Express a Use Case (w/ Examples of Include & Generalization)

Author: Tommy

Date: 10/09/2014

Use Case Name:	Place New Member Order	
Actor(s):	Club Member	
Description:	This use case describes the process of a club member submitting a new order for SoundStage products. On completion, the club member will be sent a notification that the order was accepted.	
Reference:	MSS-1.0	
Typical Course of Events:	<p>Actor Action</p> <p>Step 1: This use case is initiated when a member submits an order to be processed.</p>  <p>Step 9: This use case concludes when the member receives the order confirmation notice.</p>	<p>System Response</p> <p>Step 2: The member's personal information such as address and phone number is validated against what is currently on file.</p> <p>Step 3: For each product being ordered, validate the product number.</p> <p>Step 4: For each product being ordered, check the availability in inventory and record the ordered product information such as the quantity being ordered.</p> <p>Step 5: Invoke the use case <i>Calculate Subtotal and Tax</i>.</p> <p>Step 6: The member's credit card information is verified based on the amount due and Accounts Receivable transaction data is checked to make sure no payments are outstanding.</p> <p>Step 7: Invoke the use case <i>Generate Warehouse Packing Order</i>.</p> <p>Step 8: Generate an order confirmation notice indicating the status of the order and send it to the member.</p>

A More Formal Way to Express a Use Case (w/ Examples of Extend)



Alternate Courses:	<p>Step 2: [Extension point: If the club member has indicated an address or telephone number change on the order, invoke the use case <i>Revise Street Address</i>].</p> <p>Step 3: If the product number is not valid, send a notification to the member requesting the member to submit a valid product number.</p> <p>Step 4: If the product being ordered is not available, record the ordered product information and mark the order as “backordered.”</p> <p>Step 6: If member’s credit card information is invalid or if member is found to be in arrears, a credit problem notice is sent to the member. Modify the order’s status to be “on hold pending payment.”</p>
Precondition:	Orders can only be submitted by members.
Postcondition:	Member order has been recorded and the Packing Order has been routed to the Warehouse.
Assumptions:	None at this time.

Actor Description of an Invoice System

Buyer

A **Buyer** represents a person who is responsible for [buying goods or services](#). This person may be an individual (i.e., not affiliated with a company) or someone within a business organization. The Buyer of goods and services needs the system [to send orders](#) and [to pay invoices](#).

Seller

A **Seller** represents a person who [sells and delivers goods or services](#). The Seller uses the system to [look for new orders](#) and to [send order confirmations](#), [invoices](#), and [payment reminders](#).

Accounting System

The **Accounting System** is responsible for [keeping account balances](#).

The **Accounting System** needs the system to [send it verified transactions](#).

Describe a Use Case:

“Pay Invoice” Use Case

Purpose: Allow buyers to pay outstanding invoice(s)

Actors: Buyer, Accounting System

Preconditions: The buyer has received the goods or services ordered and at least one invoice from the system. The buyer now plans to schedule the invoice(s) for payment.

Flow of Events

1. The **buyer** invokes the use case by beginning to **browse the invoices** received by the system. The **system checks** that the content of each invoice is **consistent** with the order confirmations received earlier.
2. The **buyer** decides to **schedule** an invoice for **payment** by the bank, and the **system generates a payment request** to transfer money to the seller's account. A buyer may not schedule the same invoice for payment twice.
3. Later, if there is enough money in the buyer's account, a **payment transaction** from the buyer's account to the seller's account. The buyer and seller are **notified of the result of the transaction**. The bank collected a fee for the transaction, which is withdrawn from the buyer's account.
4. The use case instance terminates.

Pay Invoice Use Case

Alternative Paths

In Step 2, the *buyer* may instead *ask* the *system* to *send an invoice rejection* back to the seller.

Exceptions

In Step 3, if there is not enough money in the account, the use case will *cancel the payment and notify the buyer*.

Postconditions: The use-case instance ends when the invoice has been paid or when the invoice payment was canceled and no money was transferred.

Extension Points: 1. *Register Transaction*

Use Case Specification

- A use case specification should
 - Cover the *Full sequence of steps* from the beginning of a task until the end.
 - Describe the *User's interaction* with the system ...
 - Not the actual computations the system performs.
 - As *independent* as possible from any particular user interface design (i.e. it is not about design)
 - Only include actions in which the actor interacts with the system

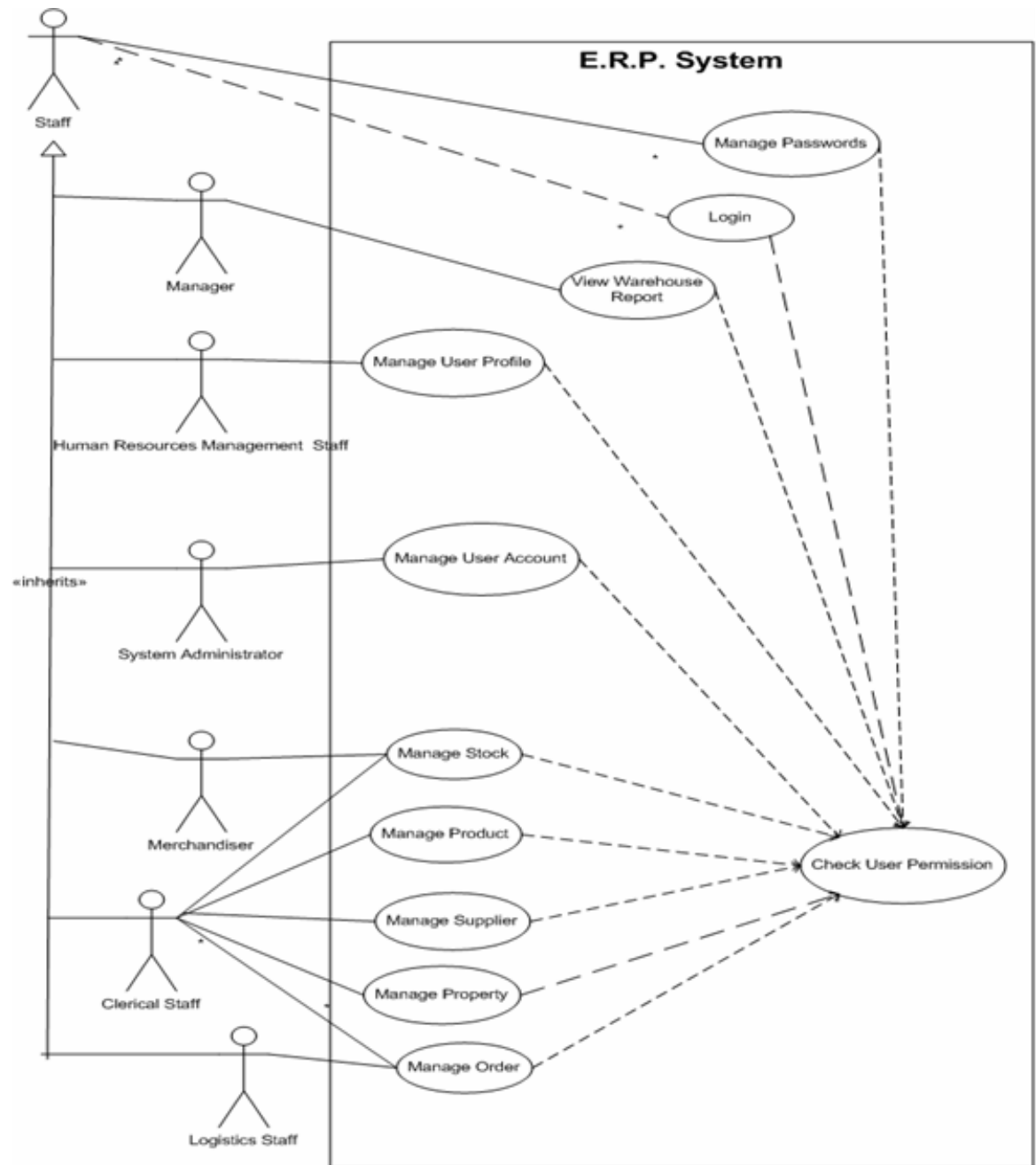
What Makes **Good** Use-Case Specification?

- **Explicitness** and Clarity (Lack of ambiguity)
 - Each requirement must be interpreted in a single manner.
- **Completeness**
 - They should cater for all **current** demands of the system.
- **Consistency**
 - Requirements should not conflict with each other. If there are, tradeoffs must be detected and discussed.
- **Avoid design**
 - Requirements should describes the “needs”, not designing solutions to the needs it.

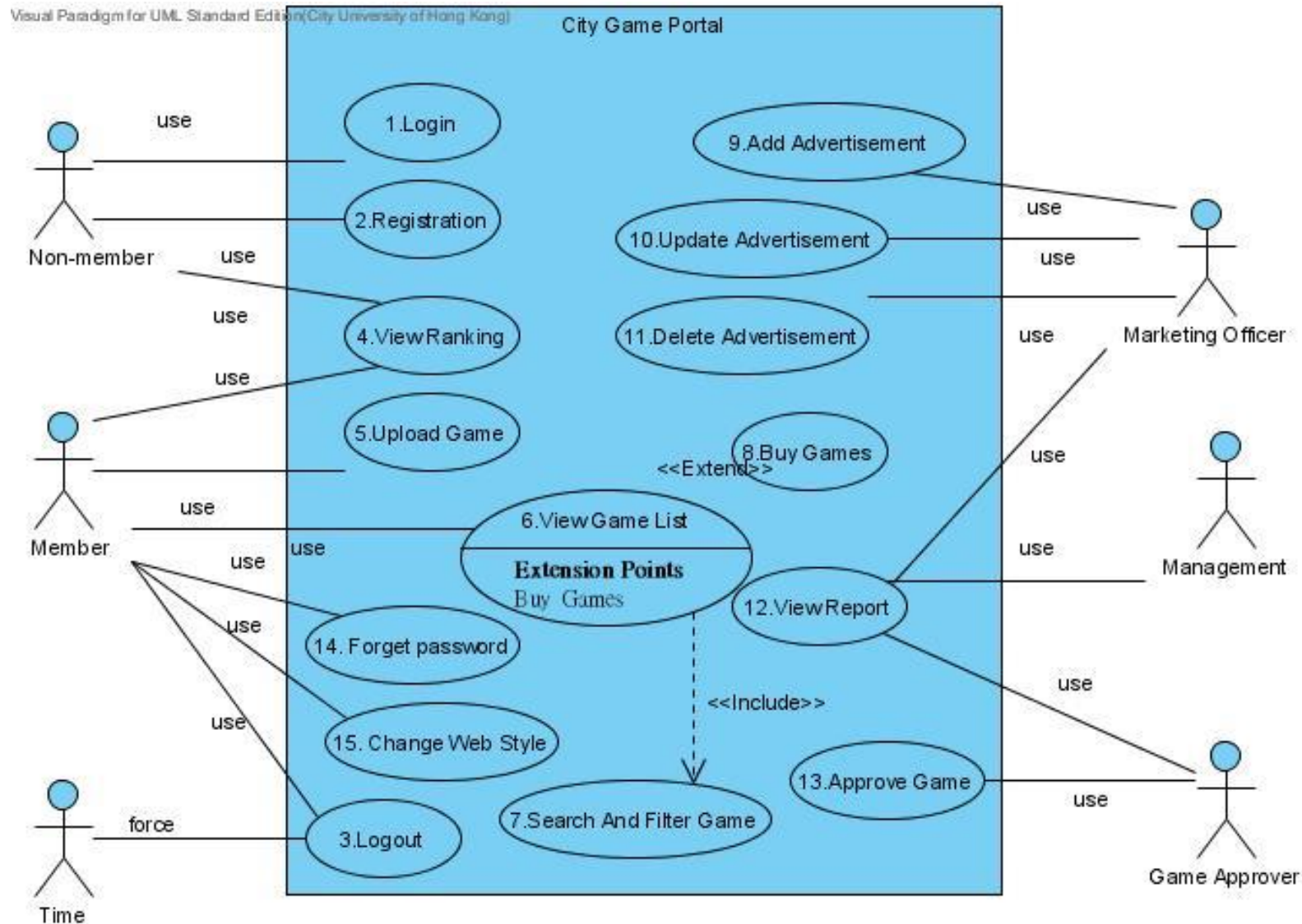
The benefits of software development based on well developed use-cases

- They can
 - help all parties to **define the scope** of the system
 - be used to **plan** the development process
 - be used to both develop and **validate the requirements**
 - form the basis for the definition of **test cases**
 - Be used to structure **user manuals**

Sample Use Case Model From Previous Students



Sample Use Case Model From Previous Students



Sample Use Case Specification From Previous Students

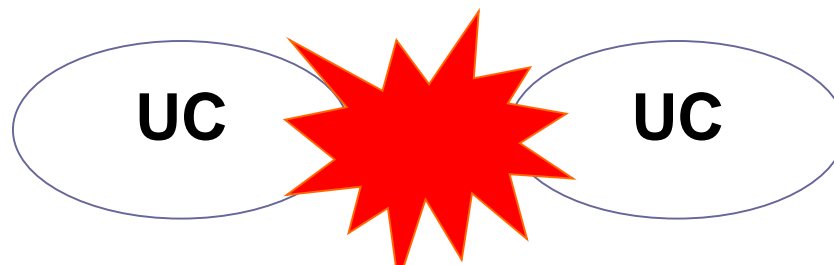
Use-Case Name:	View Game List	
Actor(s):	Primary: Member	
Description:	This use case describes the event of viewing a list of games. The member can view the games which are posted in this game web site. They can input some keywords to search what games they want. They can also spend their e-cash to buy what games they are interested in.	
Reference ID:	6	
Precondition:	The visitors must have previously logged on so that the system can identify them as a member or not.	
Trigger:	The use case is initiated when the members have logged in and then select this option from the user interface.	
Typical Course Of Events:	Actor Action	System Response
	<p>Step 1: This use case is initiated when the member has logged in and selected the view game list option.</p> <p>Step 3: The member can click the game name to see the game description.</p> <p>Step 5: the member read the game description.</p>	<p>Step 2: The system responses by displaying all the games to the member.</p> <p>Step 4: The system display the description of the game the member has clicked.</p>
Alternate Courses:	<p>Step 2a: The system display all the related games which is according to the member's input where the member has inputted the keywords of the game in the search area.</p> <p>Step 2b: If the system cannot find the related game from the member's input, the system will display an error message.</p>	
Postcondition or Results:	The member views all the games or view what the games he or she has searched.	
Implementation Constraints and Specifications:	For security reason, the member must login to view the game list.	
Assumptions:	Assume the member has logged in and has inputted the keywords when searching.	
Open Issues:	None	

Outline

- **Introduction**
- **Use Case Diagrams**
- **Writing Use Cases**
- **Guidelines for Effective Use Cases**

How to model scenarios:

- **Number Limit:**
 - The diagram should have between 3 to 10 **base** use-case. No more than 15 use cases (base + included + extending).
- **Abstraction:**
 - All use-cases should be in similar abstraction levels.
- **Size:**
 - Use cases should be described in half a page or more.
 - If a use-cases takes more than a page, consider include/extend
- **Weak dependency:**
 - If the dependency between two parts of a use-case is weak, they should be splitted.

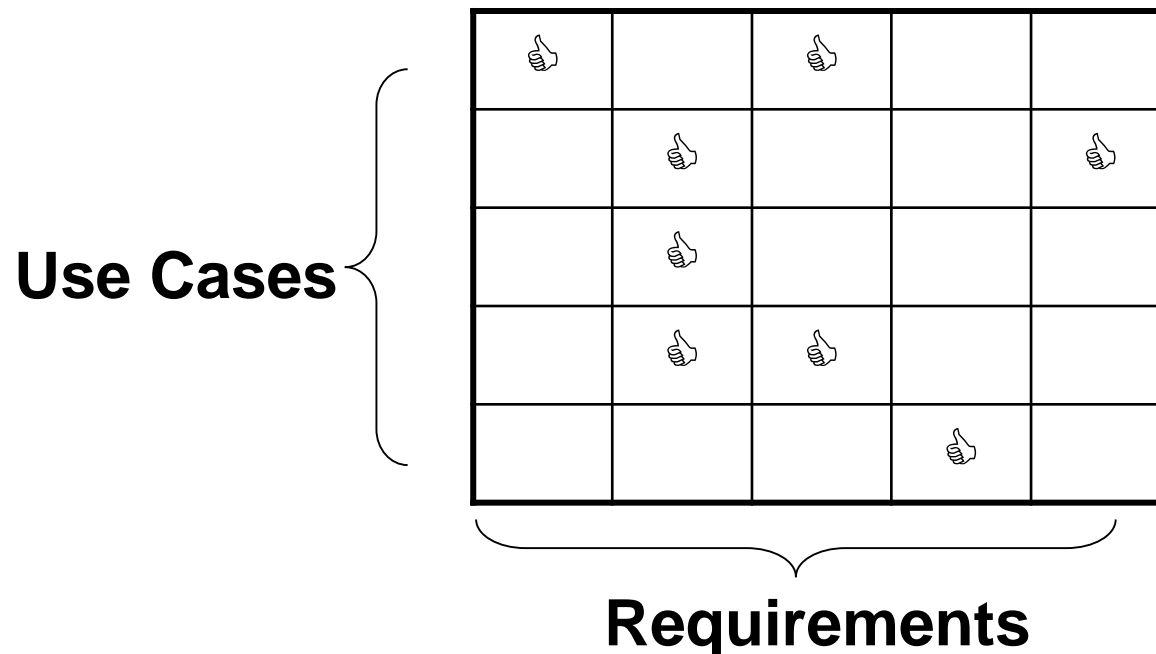


More Guidelines

- Factor out common usages that are required by multiple use cases
 - If the usage is required use <<include>>
 - If the base use case is complete and the usage may be optional or conditional consider use <<extend>>
- A use case diagram should:
 - **contain only use cases** at the same level of abstraction
 - **include only actors** which are required

When? Are we done yet?

- When every **actor** is specified.
- When every **functional requirement** has a use-case which satisfies it.
- A simple tractability matrix can help us determine it:



Use Case Summary

- ✓ Introduction
 - ▶ To Use Case Diagram
- ✓ Use Case Diagrams
 - ▶ Dual presentation of use-cases
 - ▶ Include, Extend, Inheritance
- ✓ Writing Use Cases
 - ▶ Preconditions & Post-conditions
 - ▶ Main scenario vs. Alternative Flow
- ✓ Guidelines for Effective Use Cases

Your Project

- Title/Topic done?
- What are the features/functional requirements?
- User stories, use case scenarios?
- Start working on the use cases of your project
- 1. Determine Actors (human, system) which are external to the system
- 2. Use cases and system boundary (scope) ?
- 3. What are the interactions between them?