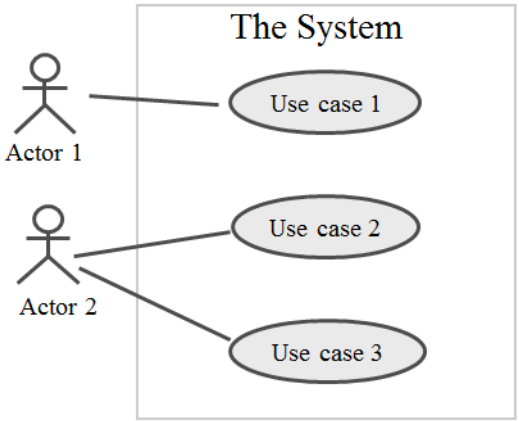
CMPS 411

Use-Case Modeling



Dr. Abdelkarim Erradi

CSE @ QU

Outline

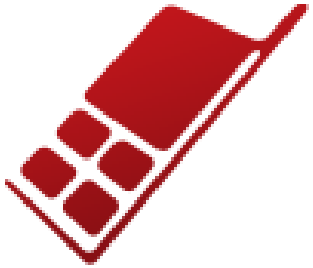
- Use Case **Model**
- Major **Concepts** in Use-Case Modeling
- Use Case Relationships
- **Benefits** of a Use-Case Model
- **How to develop a Use-Case Model?**

What Is a Use-Case Model?

- 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**

Use cases vs. internal features

- Consider software to run a cell phone:



Use Cases

- call someone
- receive a call
- send a message
- store a contact

Point of view: user

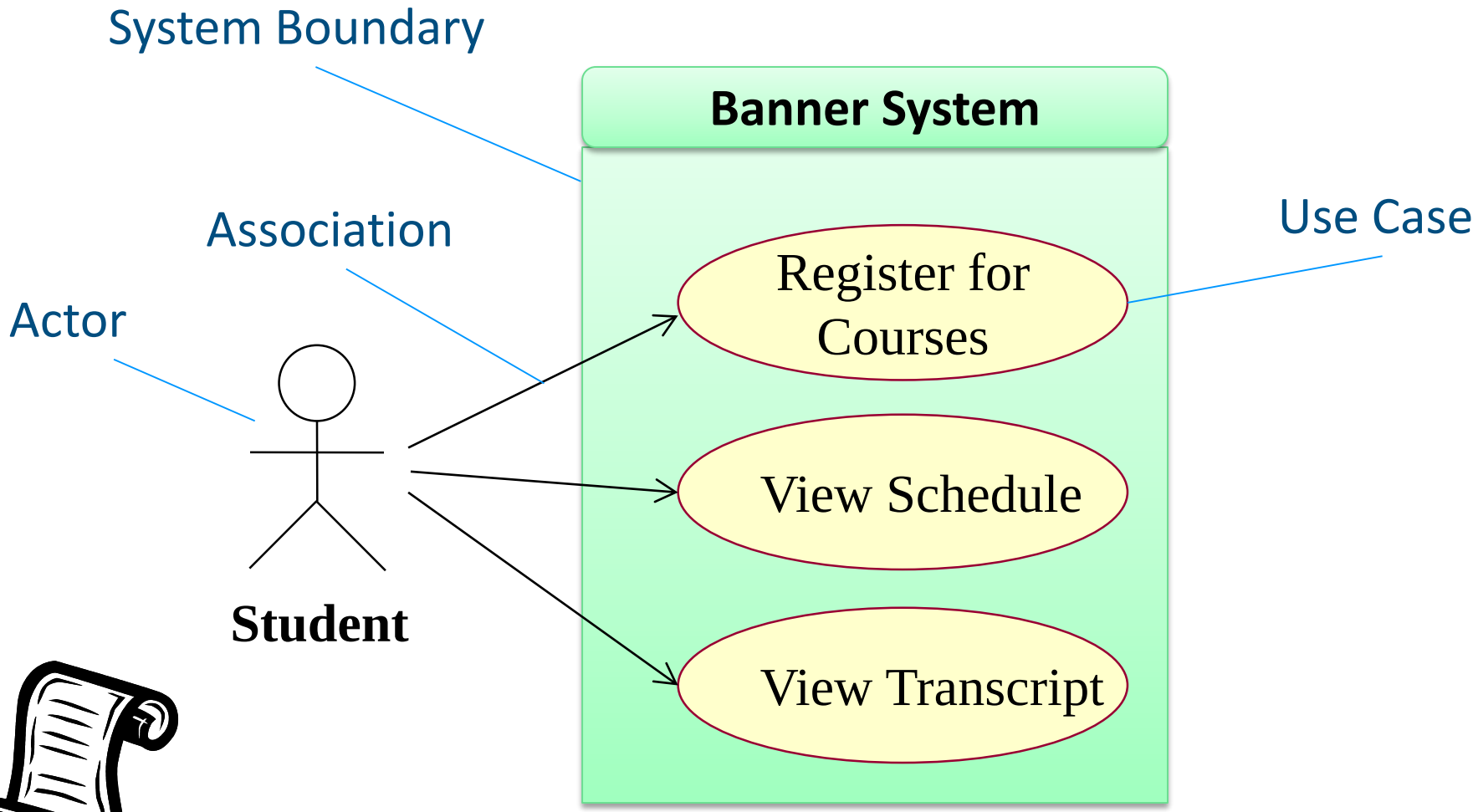
Internal Functions

- transmit / receive data
- energy (battery)
- user I/O (display, keys, ...)
- phone-book mgmt.

Point of view: designer/developer

Major Concepts in Use-Case Modeling

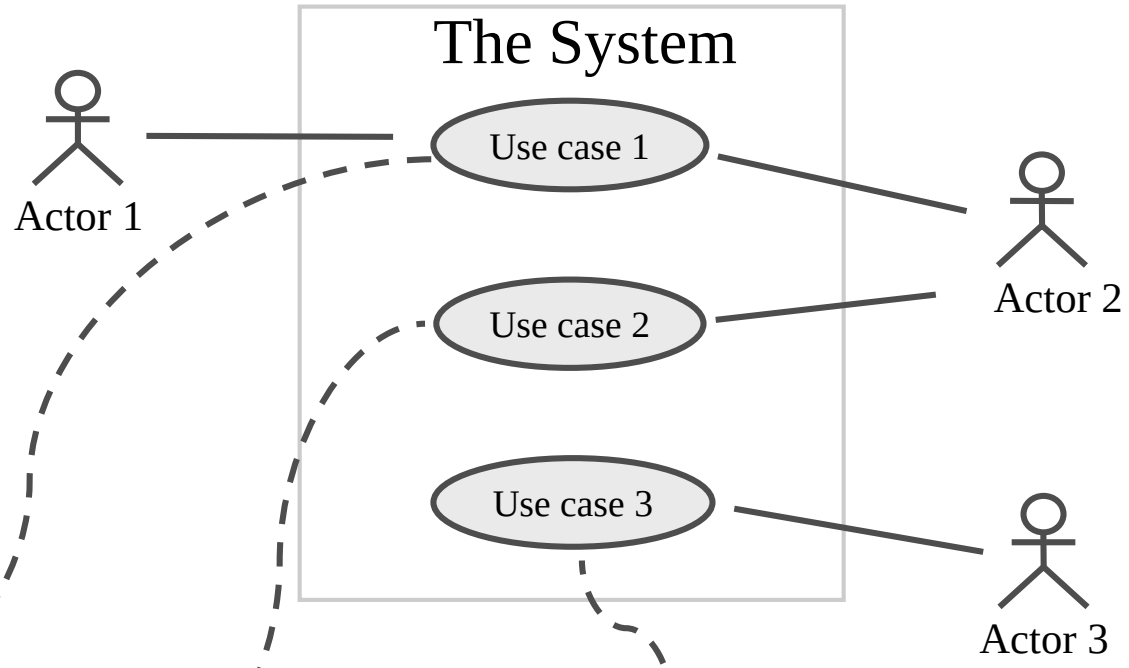
Simplified Use case model for Banner



+ A document describing the use case scenarios in details

A Use-Case Model is Mostly Text

The diagrams give a visual overview of the system

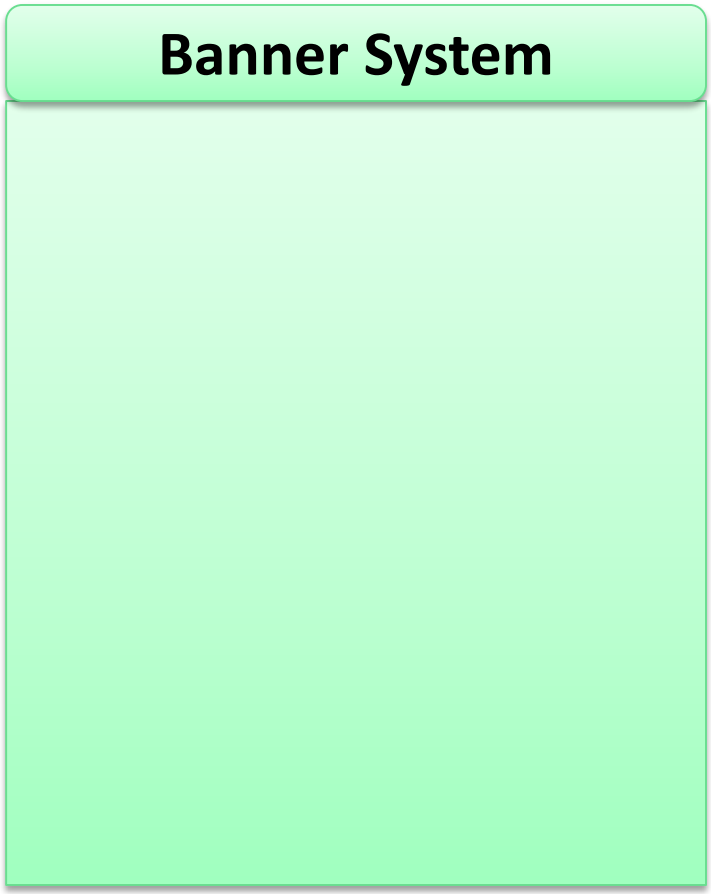


Use-Case 1 Spec
- brief description
- flow of events

Use-Case 2 Spec
- brief description
- flow of events

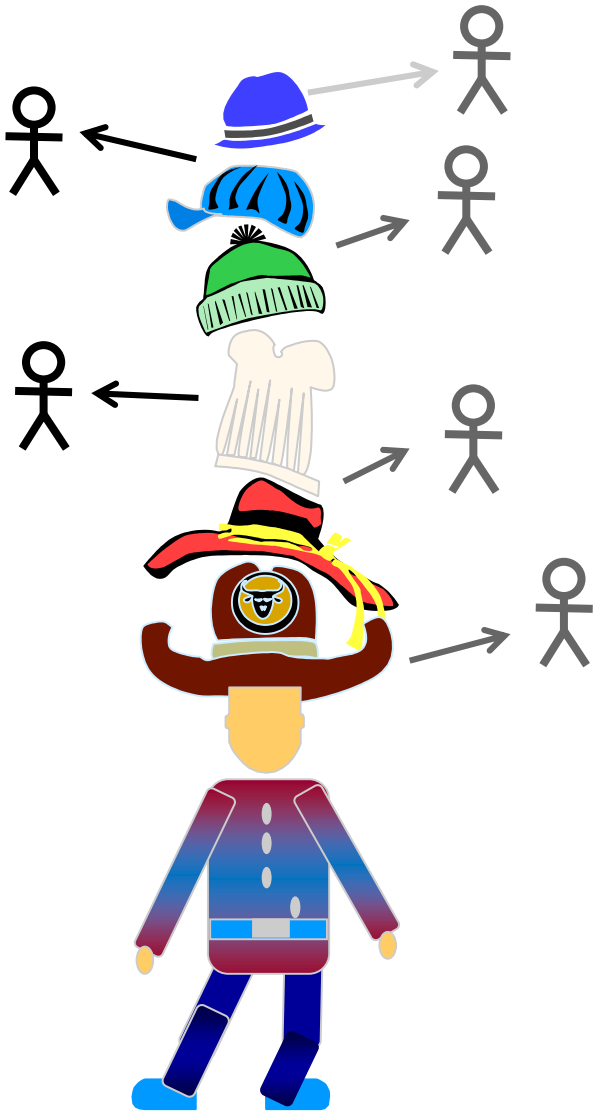
Use-Case 3 Spec
- brief description
- flow of events

System Boundary



- Marks off the system as separate from its environment
- Actors are outside

Actor



- Basically users of the system
  - Actors represent **“roles”** not individuals
- External entities (people or systems)
  - That interact with the system either by giving or receiving information or both
  - In order to achieve a desired goal

Use Case

- Describe the **goal** that the use case is intended to achieve
  - Use active verb to describe the **intended goal** to be attained by using the system

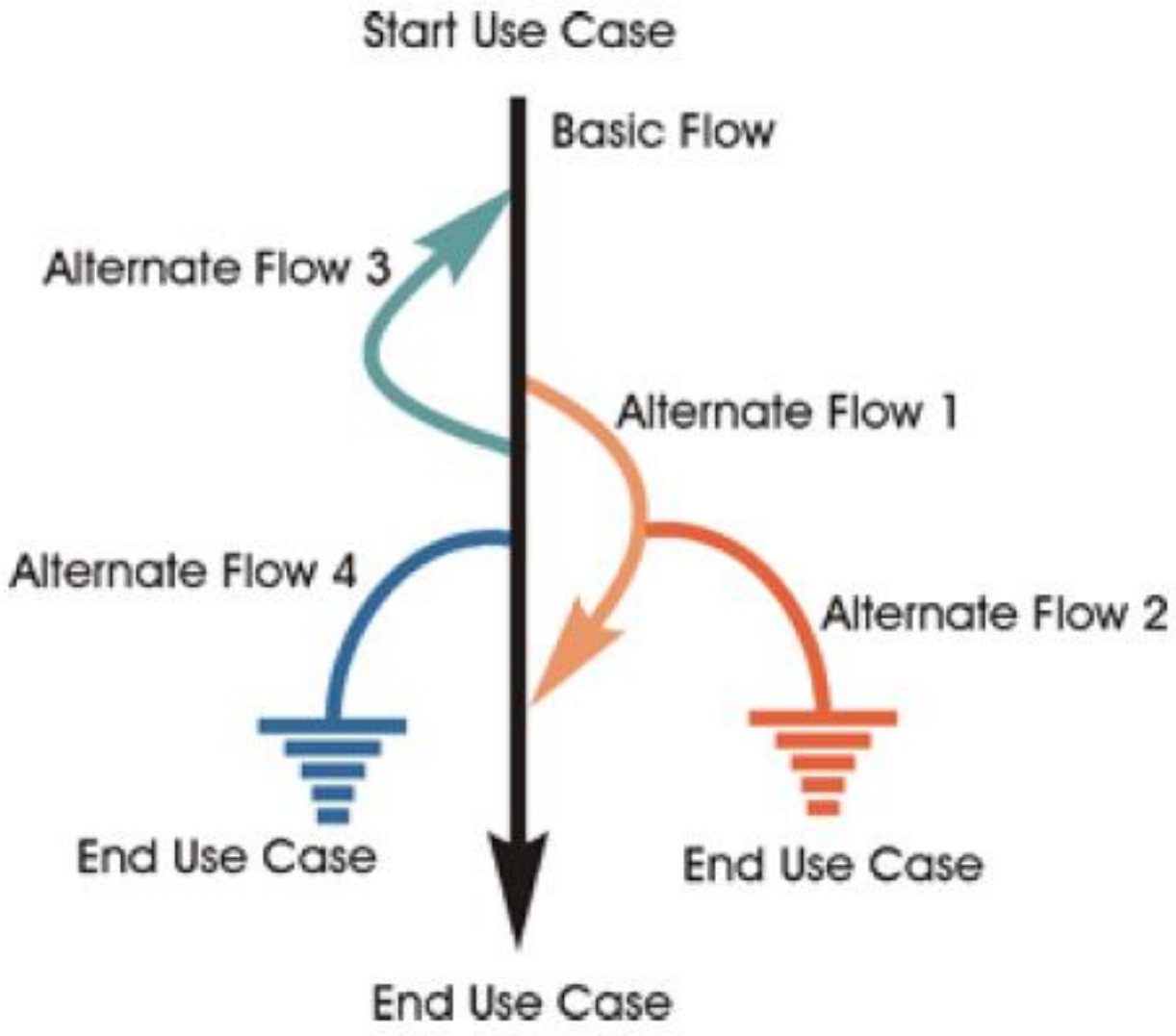
  Example: Register in an event, Issue an ID Card, Renew a Membership

  - It is the **outwardly visible and testable** activity of a system => "**external view**" of the system
  - Each use case describes a **significant chunk** of system usage

Use Case Scenarios

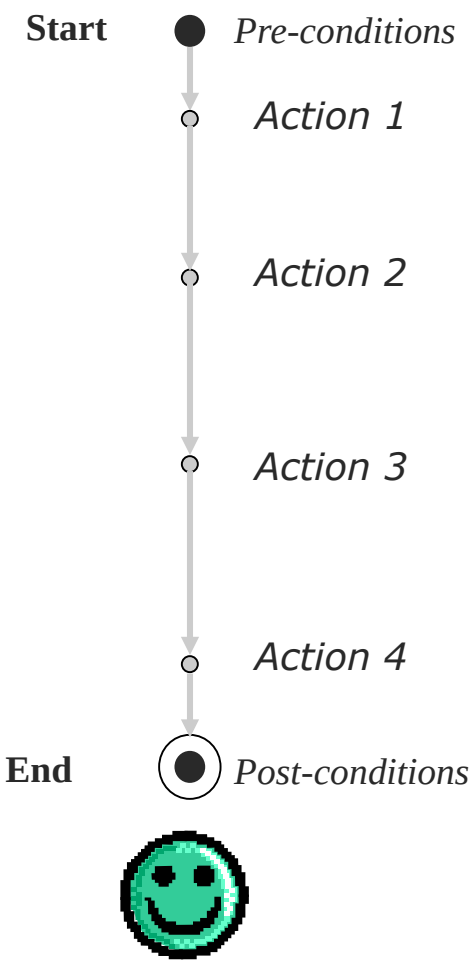
- A scenario is a sequence of steps describing the **interaction (a dialog) + the exchanged information between the actor and the system**
  - A **story of using the system** to achieve a user goal
- A use case has **one normal scenario** (happy day scenario) and several alternative flows:
  - Normal scenario describes **what "normally" happens** when the use case is performed
  - Alternative flows describe **optional or exceptional behavior** for handling errors or exceptions during the normal flow of events.

Normal Flow of Events and Alternate Flows of Events for a Use Case



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



Preconditions vs. Post-Conditions

- **Preconditions** = what *must always* be true before **beginning** a scenario. Preconditions are *not* tested within the use case but they are conditions that are *assumed to be true*. E.g. :
  - Student identified and authenticated
  - User account exists
  - User has enough money in her account
  - There is enough disk space
- **A post-condition** = success guarantee = the outcome of the use-case.

=> How to test that the use case was successful

E.g.:

  - Money was transferred to the user account
  - User is logged in
  - The file is saved to the hard-disk
  - The file is saved; Money is transferred
  - Sale is saved. Tax is correctly calculated. Inventory updated. Receipt is generated.

=> Declarations about the system state changes or outcomes rather than a description of actions to execute

→ 1.

← 2.

Normal Scenario Example

Interaction step

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

Validation Step

Internal Change Step

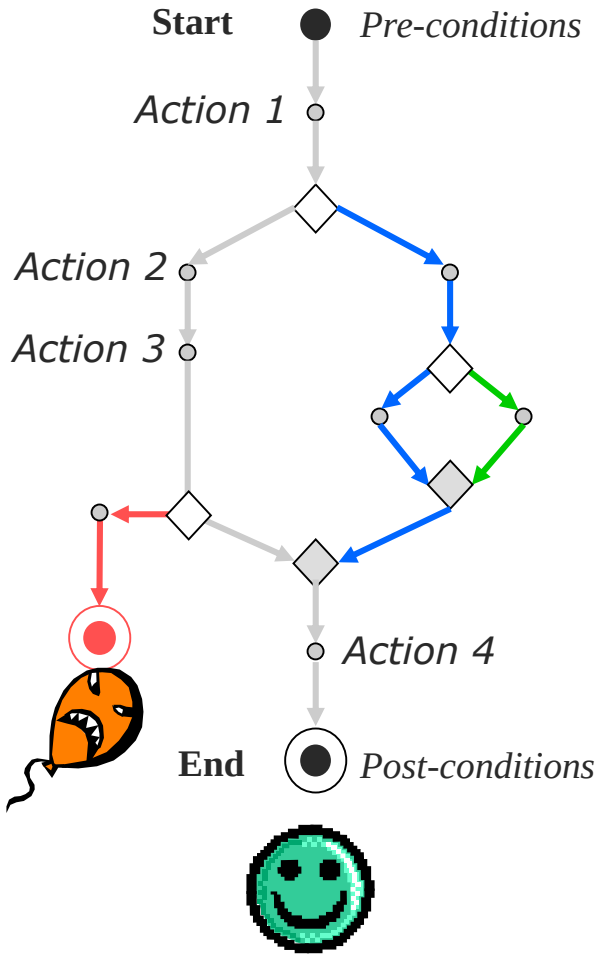
(plus) Interaction Step

Writing Alternative Flows

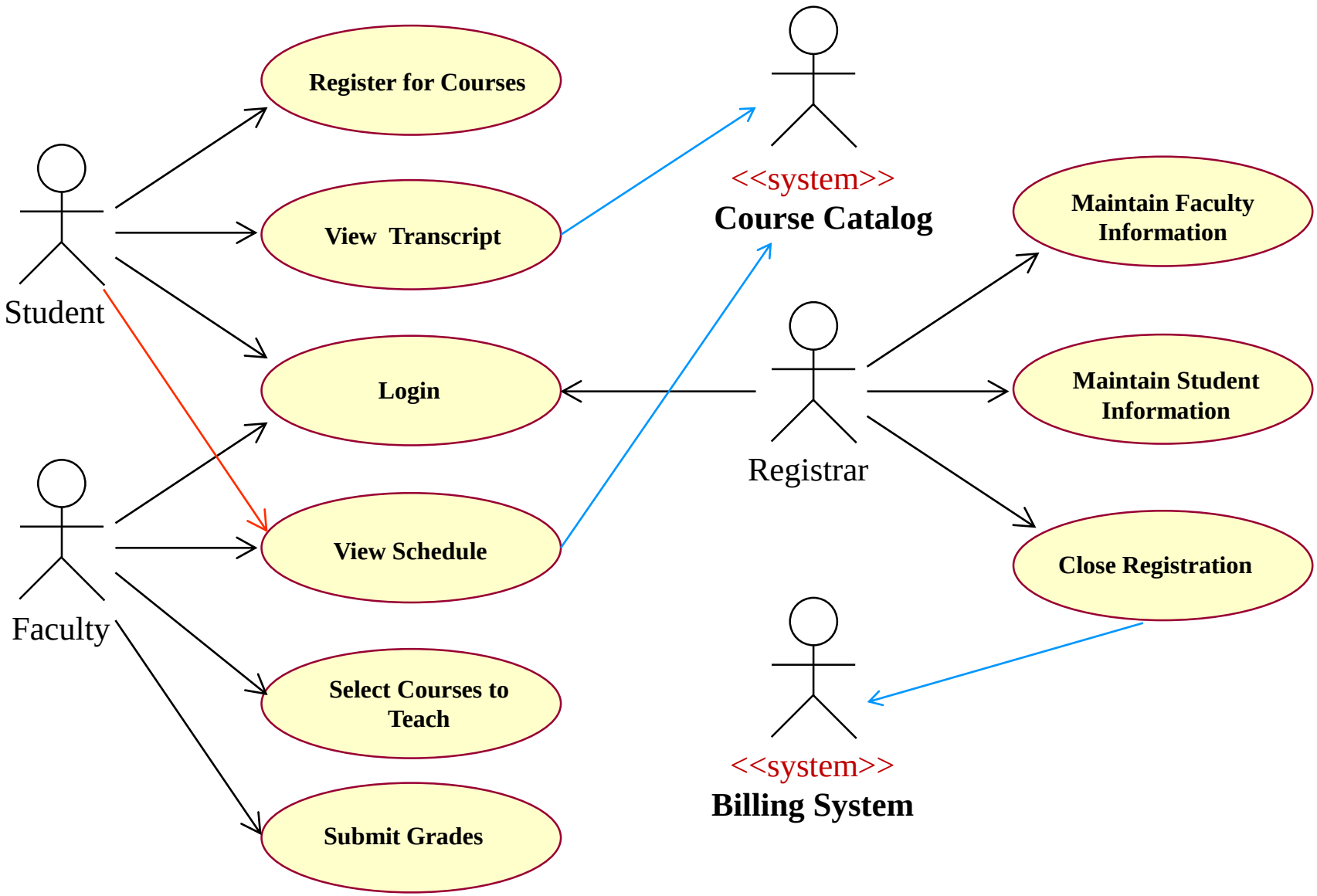
- List what can go wrong in the normal flow. e.g.:
  - *course registration closed*
  - *invalid studentId*
- Describe **what to do to handle** the identified exceptions?
  - Sometimes the exception is recoverable i.e., the **alternative flow rejoins the normal flow**

  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.



Student Registration Use Cases



Primary Actors vs. Secondary Actors

- **Primary Actor:**
  - The Actor that uses the system to achieve a goal

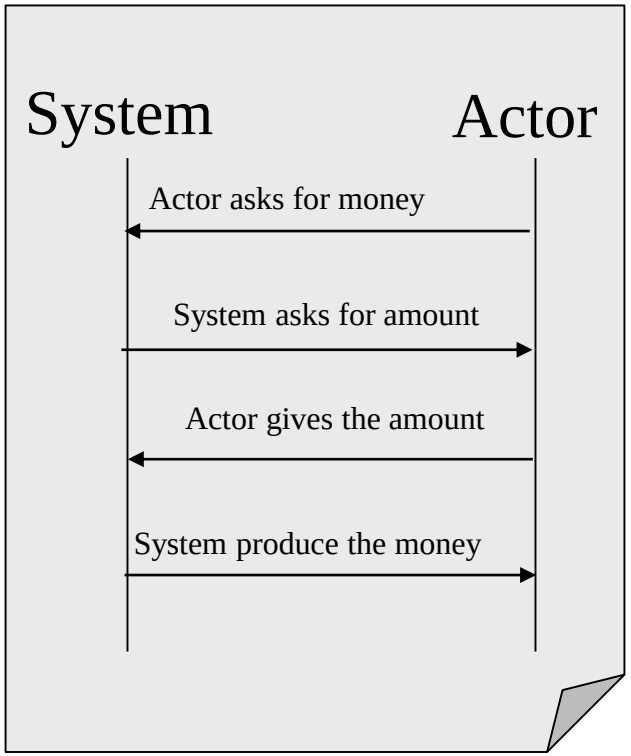
  => acts on the system

- **Secondary Actor:**
  - Actor that the system needs assistance from to achieve the primary actors goal
  - Play a supporting role

  => acted on by the system

Guidelines for Effective Scenario Writing

- Use simple and clear language
- Only one side (system or actor) is doing something in a single step
- Any step should lead to some progress
  - Bad: “User click the enter key”
- Avoid describing the user interface details
  - “User types ID and password, clicks OK or hits Enter”



Example Use case scenario

Use case Id: UC**01**	**Login**
Brief Description	User login to the Payroll System
Primary actors	Payroll Officer and HR Manager
Preconditions: The user has a valid account.	
Post-conditions: If the use case was successful, the actor is logged into the system. If not, the system state is unchanged.	
Main Success Scenario:	
Actor Action	**System Response**
1. Enters username and password	2. The system validates the entered username and password and logs the user into the system
Alternative flows: **2.a. Invalid Username/Password** If the user enters an invalid username and/or password, the system displays an error message. The user can choose to either return to the beginning of the basic flow or cancel the login, at which point the use case ends.	

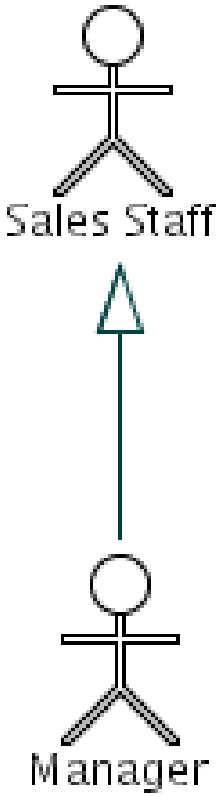
Use Case Relationships

Use Case Relationships in UML

- Possible relationships between actors:
  - **Generalization**

- Possible relationships between use cases:
  - **Include**
    - Included use case represents common behavior
  - **Extend**
    - Extending use case adds behavior
  - **Generalization**
    - One use case is a special case of another

Generalization of Actors

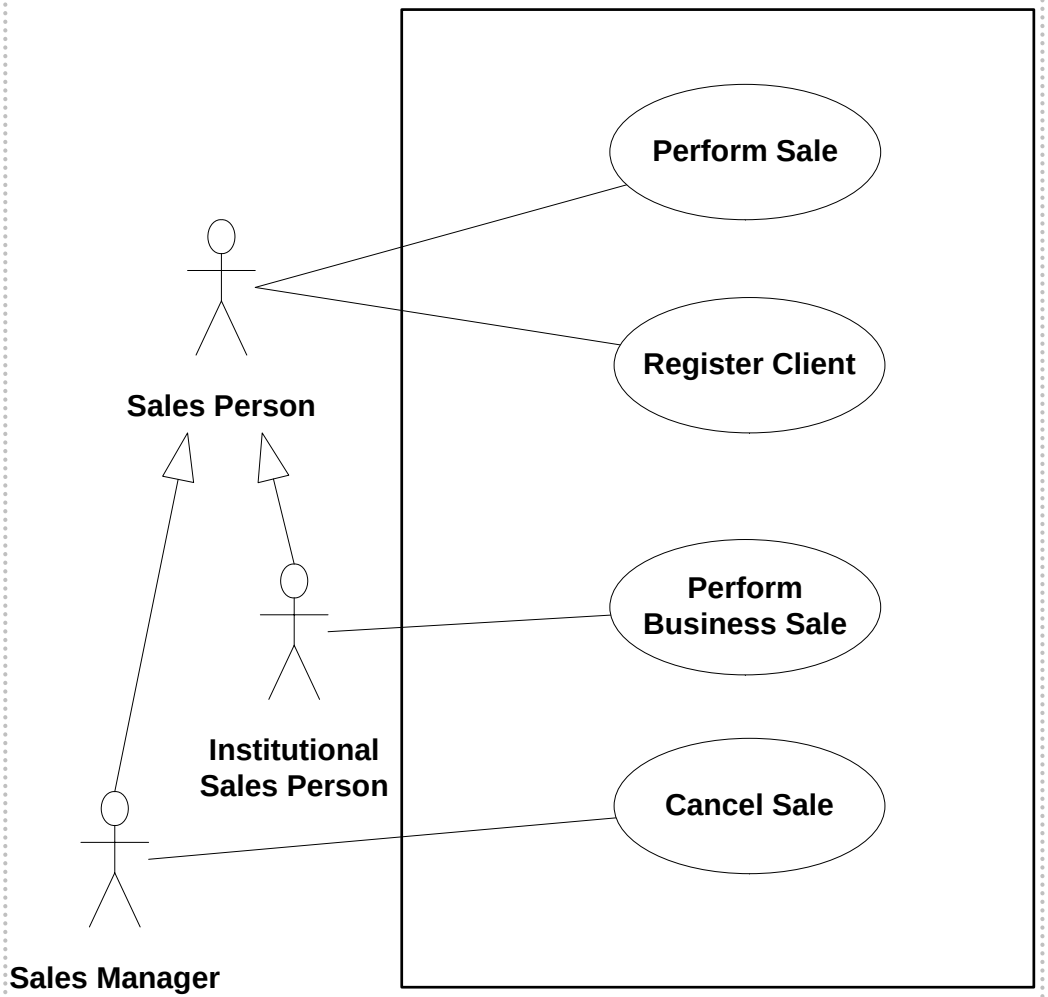


- One actor can be a specialization of another.
- Arrow points to the more general (base) actor.

Example: Generalization of Actors

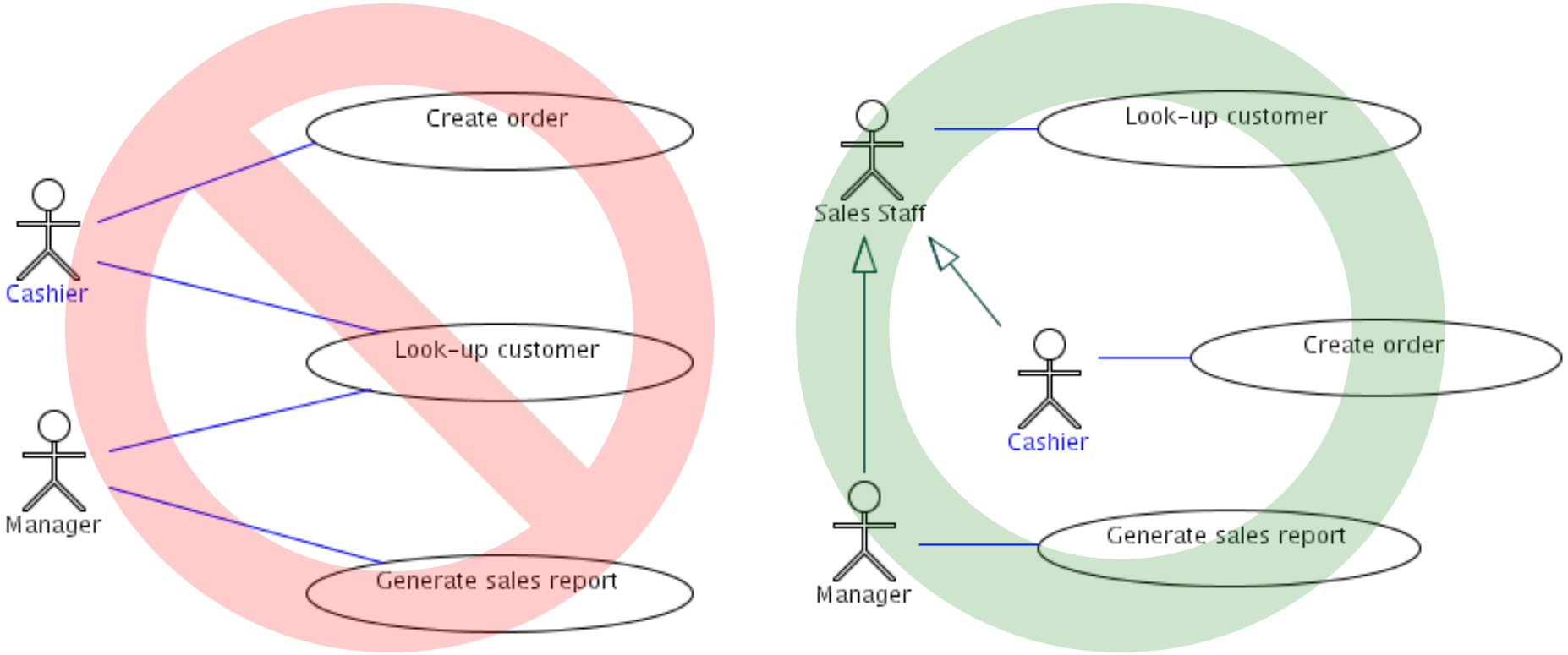
The child actor inherits all use-cases associations

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

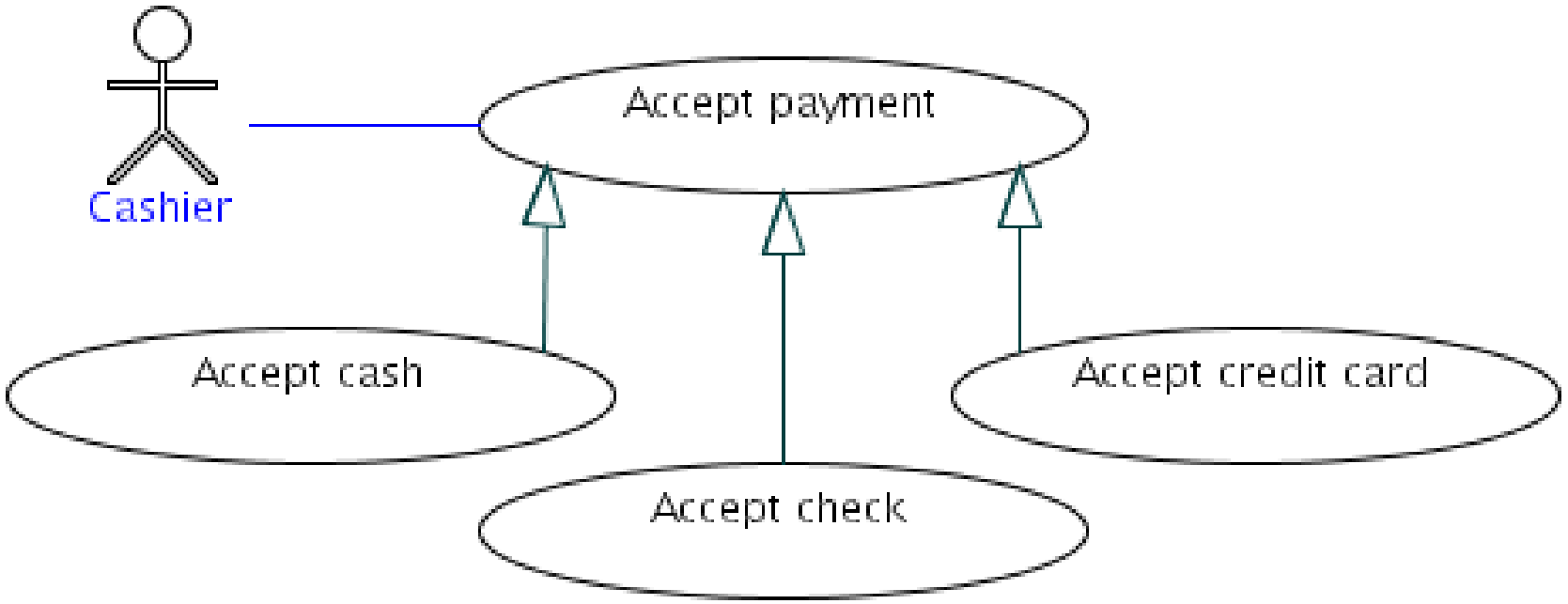


Example

Using Actor Generalization

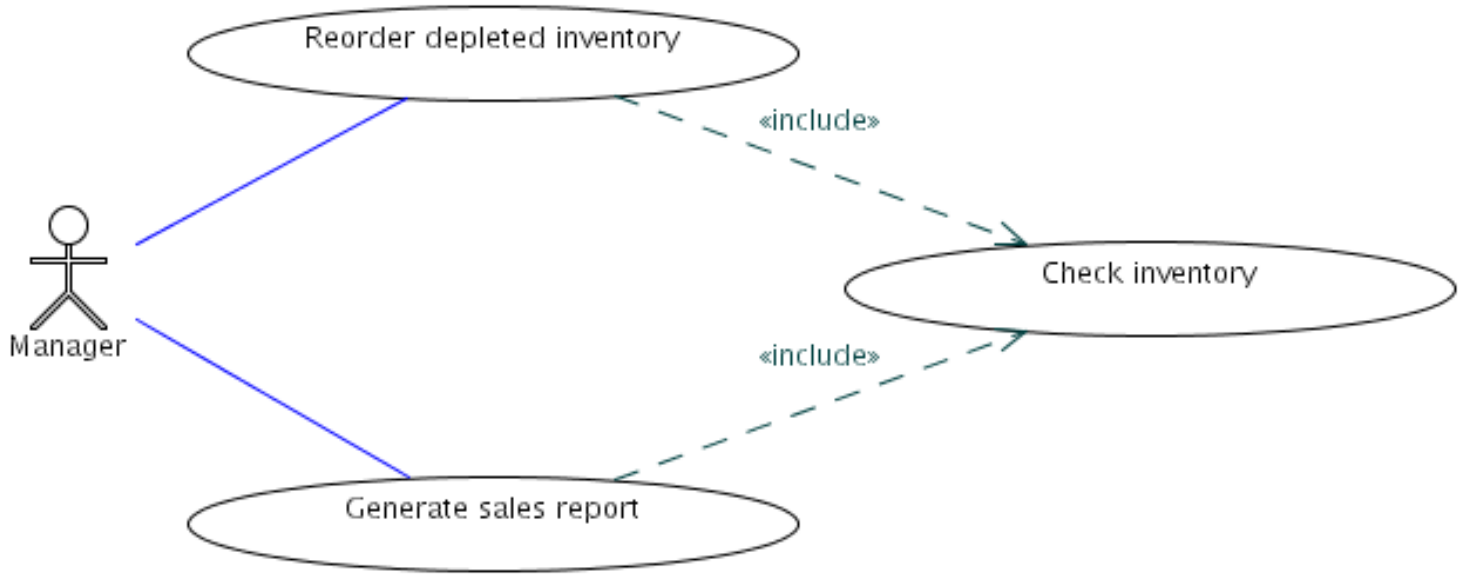


Use case Generalization



- 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)

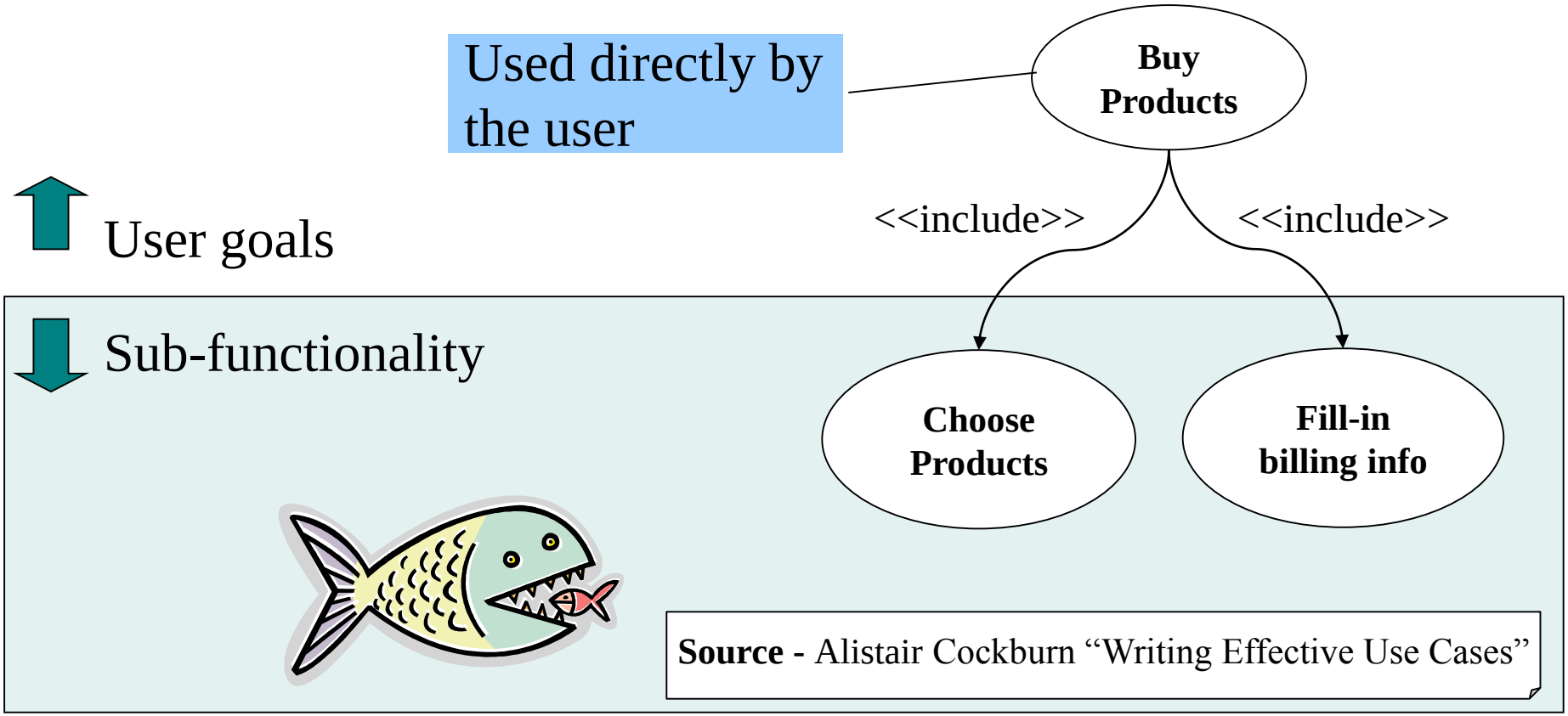
The <<include>> Relationship



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

The <<include>> Relationship

Used directly by the user

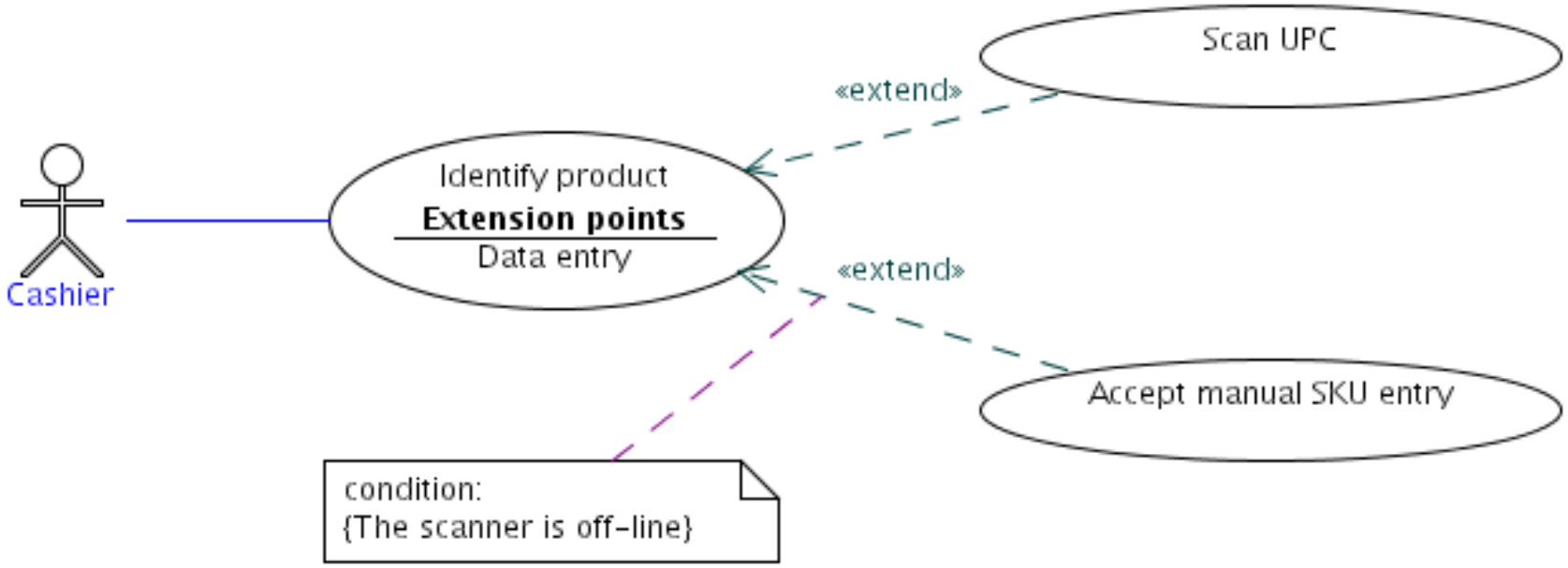


Writing Include

- If a normal scenario includes another use-case, we will describe this as a step within the normal flow:

1. <include: Login use case>
2. ….

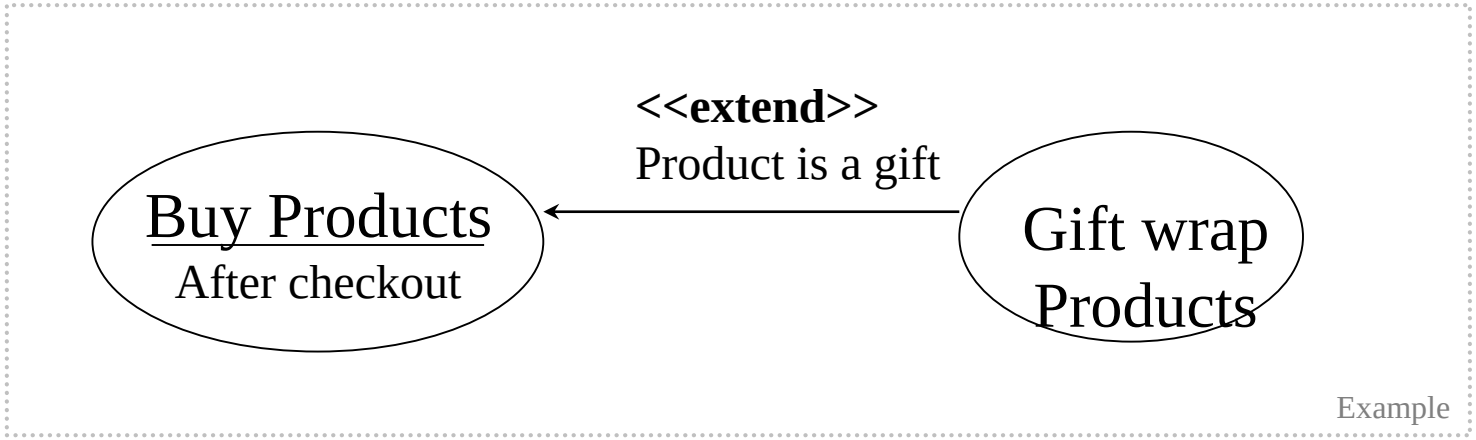
The <<extend>> Relationship



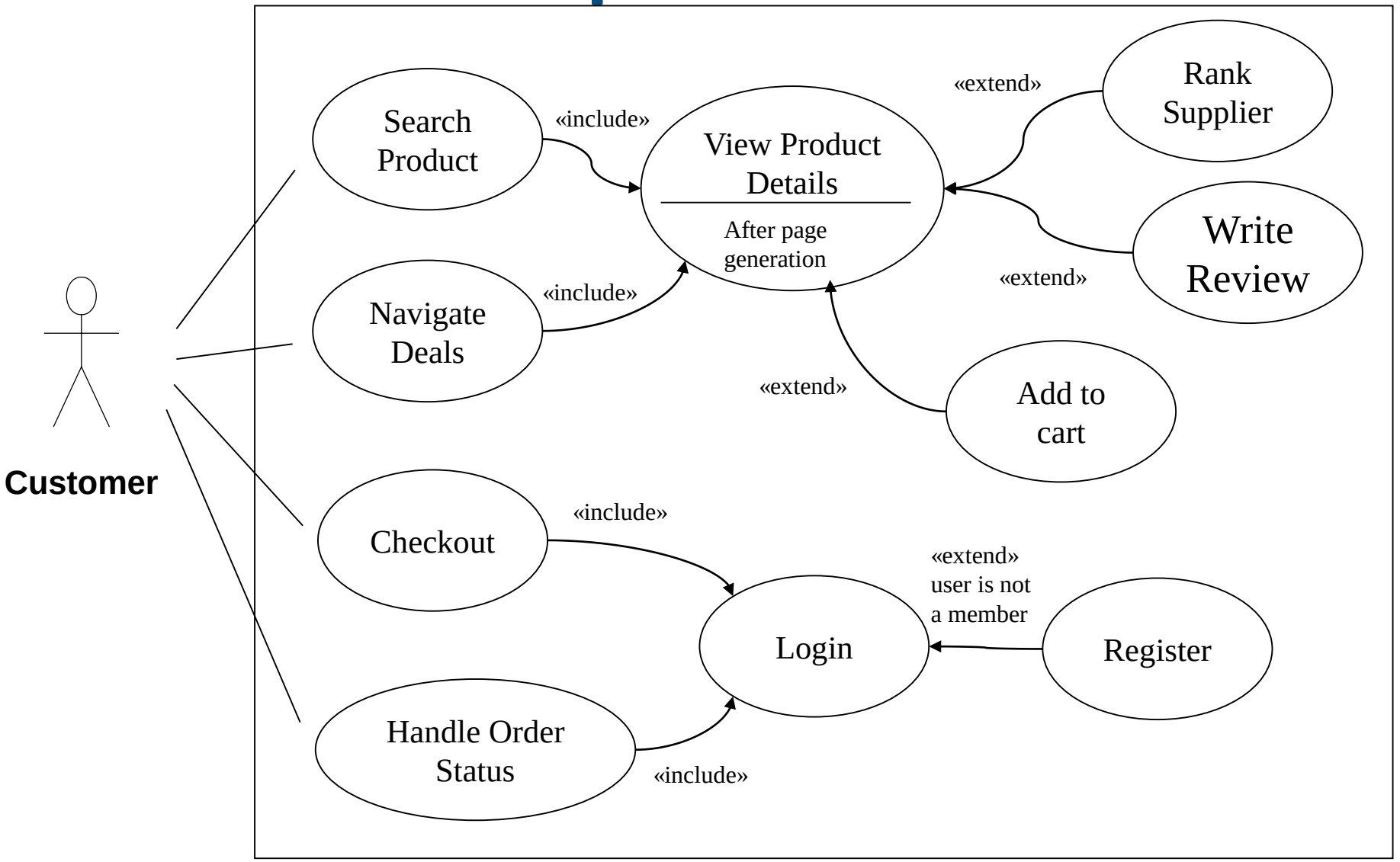
- <<extend>> relationship represent an **exceptional case**
  - The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extend>> relationship is to the extended use case

Example: <<extend> Relationship

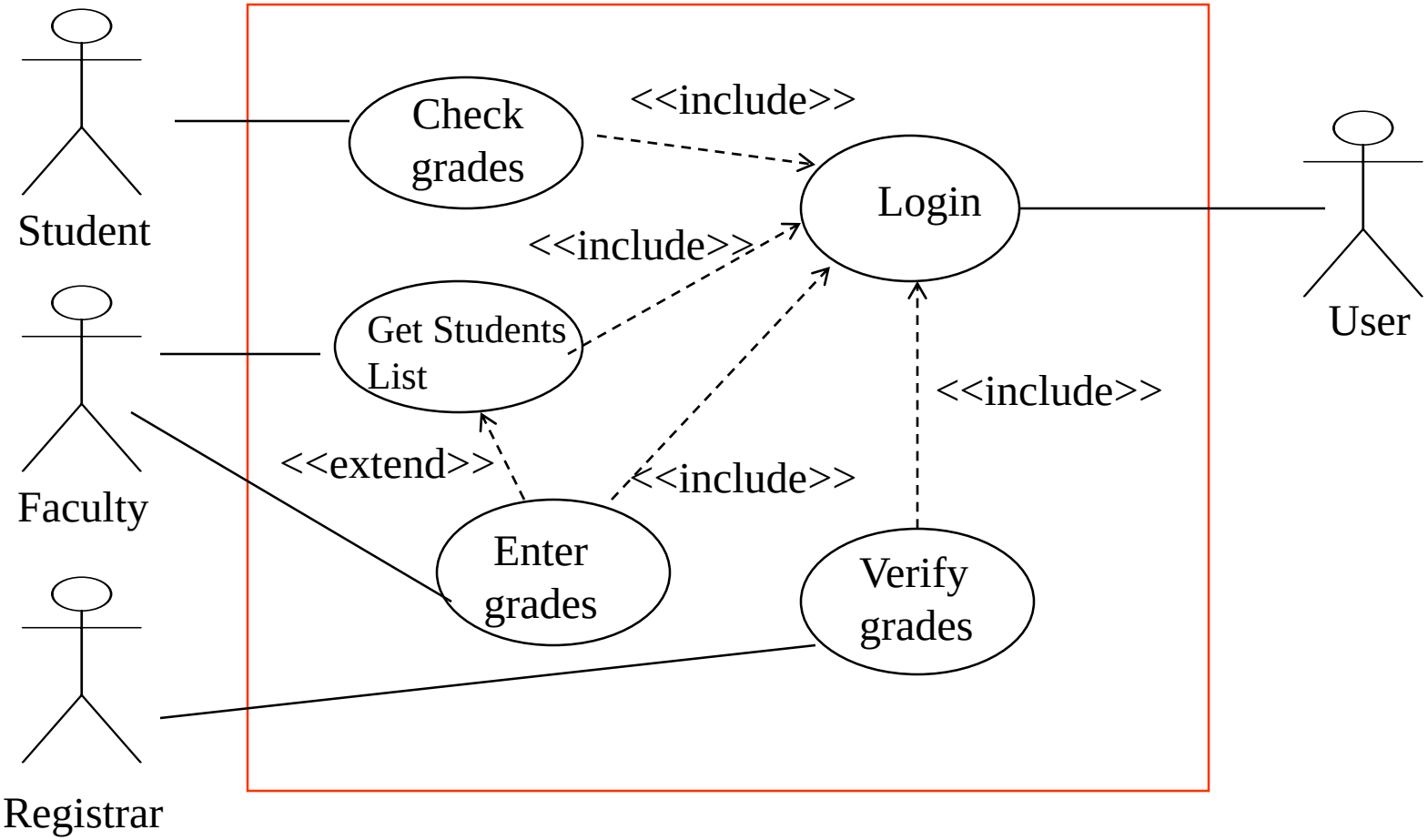
- The base use case can incorporate another use case at certain points, called extension points.
- Note the direction of the arrow
  - The base use-case does not know which use-case extends it



Example – cont'd



Example: Use Case Relationships

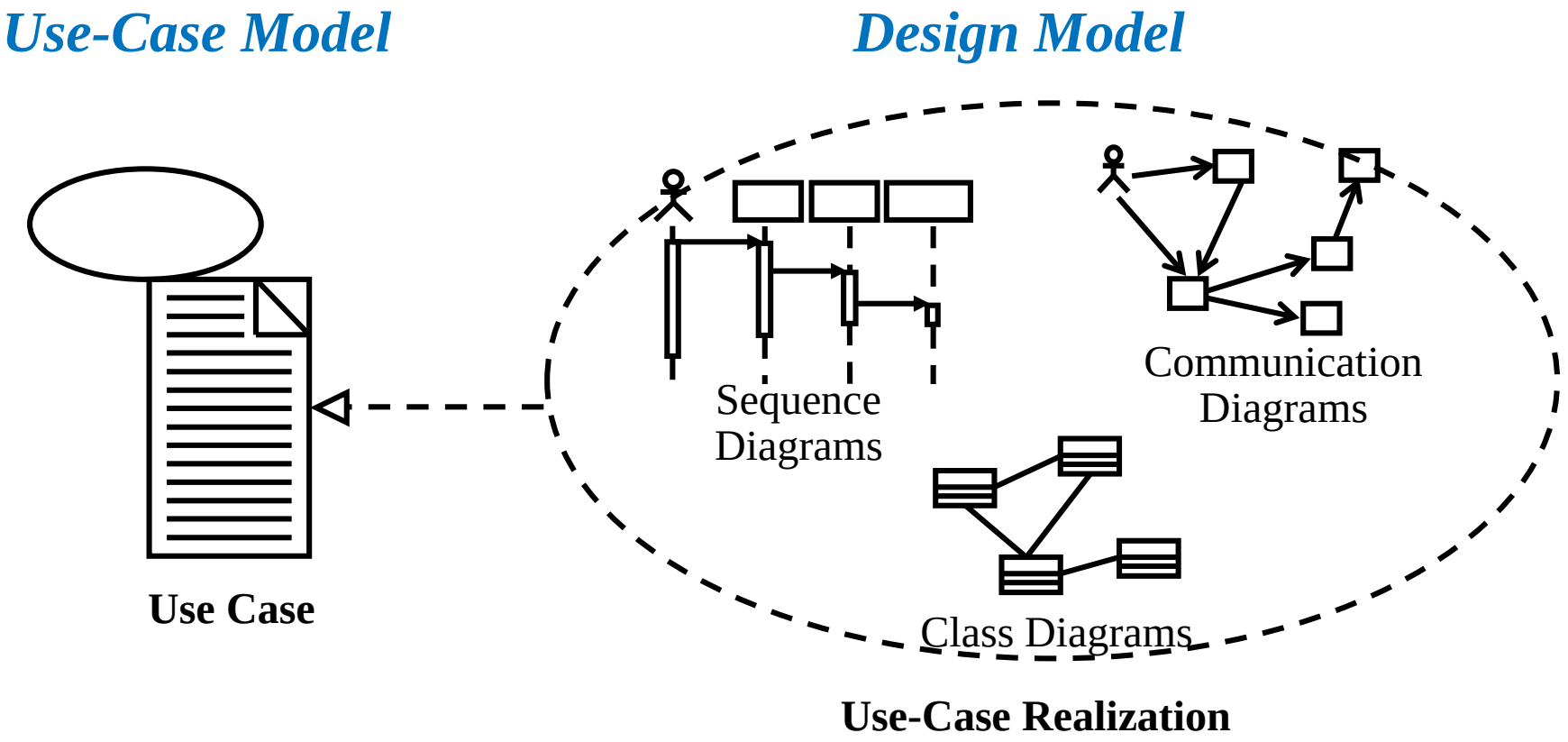


What Are the Benefits of a Use-Case Model?

Benefits

- Discover, describe and understand requirements
  - **Identification** of users and what they can do
- **Verification** that requirements are complete and consistent
- **Communication** with the end users and domain experts:
  - Ensures a mutual understanding of the requirements
- **Serve as a basis for scheduling and estimating**
  - Use cases are the **basis for the entire development process**
- Re-useable deliverable that can be used to create:
  - Test Cases
  - User Manual and online-help
  - User Interface design

Use cases drive the entire development process



How to develop a Use-Case Model?

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**

+ Walkthrough & Revise

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

Structure of a Use Case Specification

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

See the Word template provided

Alistair Cockburn "Writing Effective Use Cases"

List of NFRs that the use case must meet

Tips for writing effective Use Cases

- Actors should represent **roles** (not persons)
- Actor names should singular (not plural)
- Actor names should be consistent (e.g., if you use *Faculty* always use the same name and not *Professor*)
- Use case name should be a **verb** followed by a direct object.
- Make sure that each use case describes a significant chunk of system usage
- Do not represent communication between actors. Actors may collaborate through a use case.
- Reuse common Use Cases using <<include>> and <<extend>>
- Create detailed Use Case scenarios

More Information

- Best Use cases books:

  - Alistair Cockburn, '**Writing Effective Use Cases**' 2000.

  http://www.infor.uva.es/~mlaguna/is1/materiales/BookDraft1.pdf

  - Rebecca Wirfs-Brock, 'The Art of Writing Use Cases' 2001.

  www.wirfs-brock.com/PDFs/Art_of_Writing_Use_Cases.pdf