# CS10102302
# UML与用例建模

赵君峤 长聘副教授

计算机科学与技术学院

同济大学

# What is the UML?

- UML stands for **Unified Modeling Language**.
- The UML is a language for
  - visualizing (可视化)
  - specifying (详述)
  - constructing (构造)
  - documenting (文档化)

the artifacts of a software-intensive system.

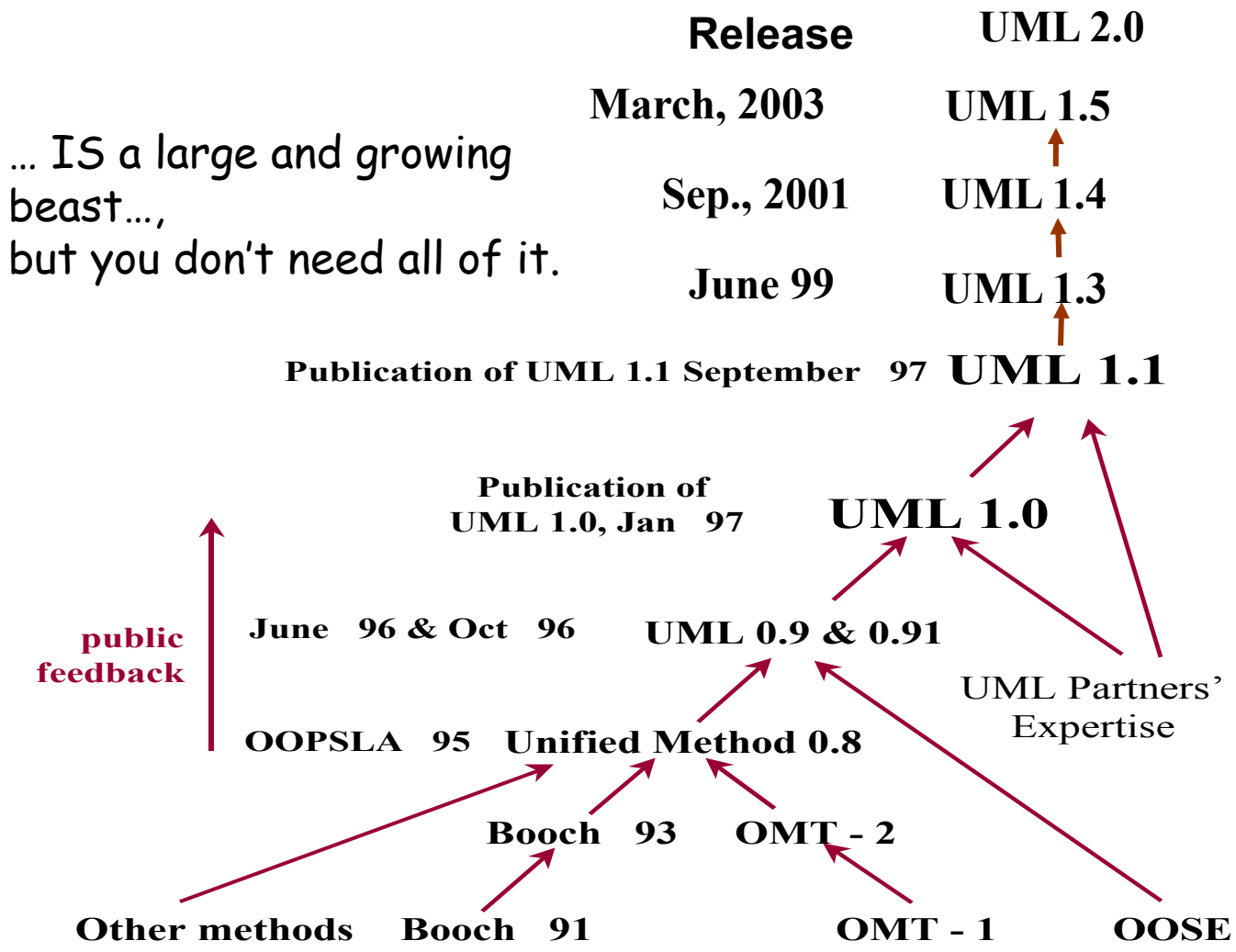
UML is the de facto standard notation for software design and analysis…

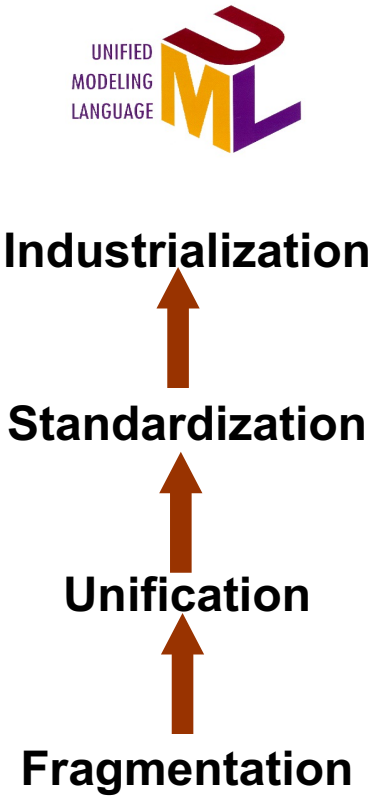
Booch

Rumbaugh

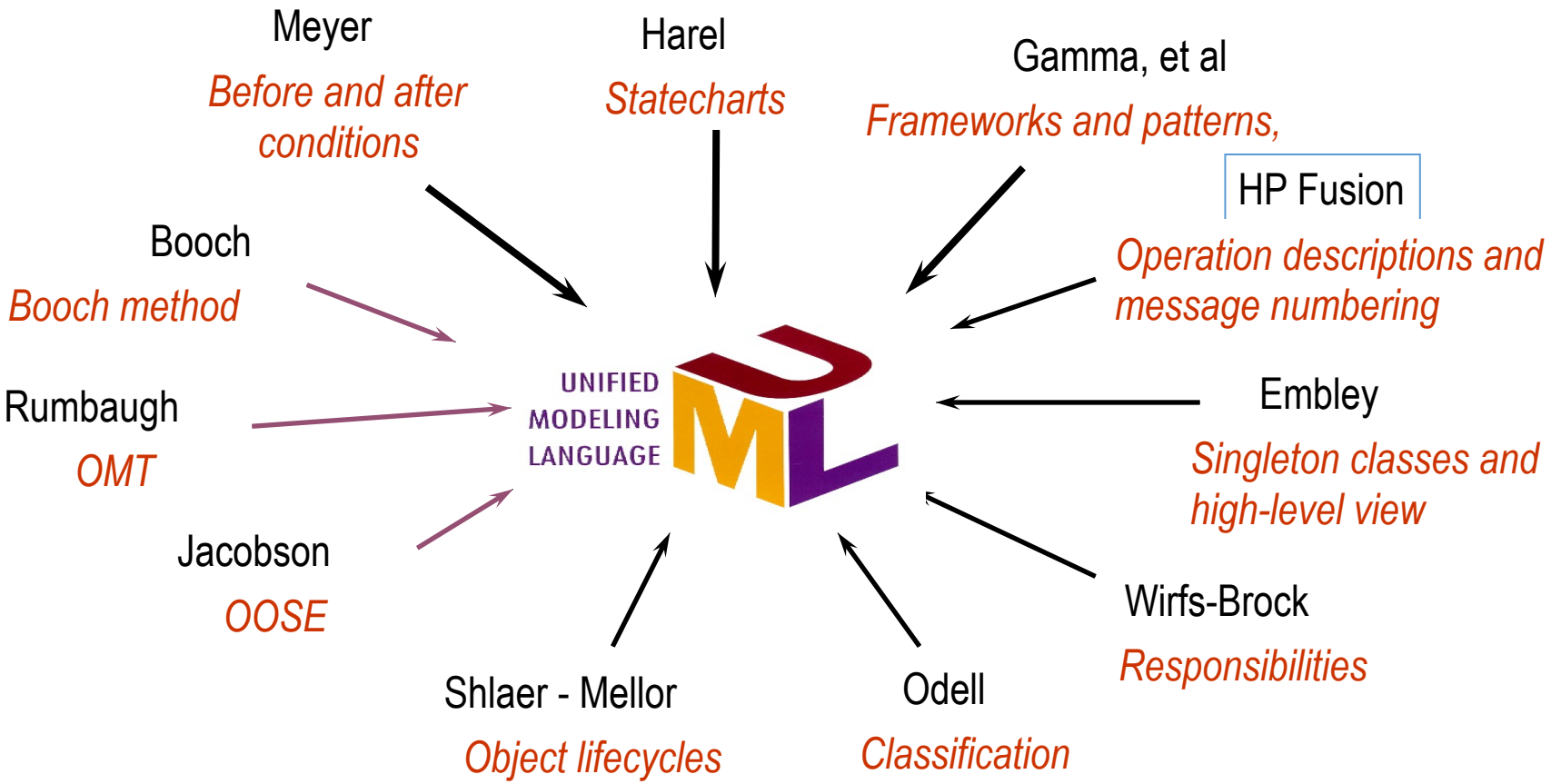
Jacobson

… IS a large and growing beast…,
but you don't need all of it.



# Contributions to the UML



# UML constructs

- 基本构造块 (basic building blocks)
  - 事物 (things)
    - 结构事物 (structural things )
      - class, interface, collaboration, use case, active class, component, node
    - 行为事物 (behavioral things )
      - interaction, state machine
    - 分组事物 (grouping things )
      - package
    - 注释事物 (annotation things )
      - note
  - 关系 (relationships)
    - 依赖 (dependency)
    - 关联 (association)
    - 泛化 (generalization)
    - 实现 ( realization )
  - 图 (diagrams )
- 规则 (rules )
- 公共机制 (common mechanisms )

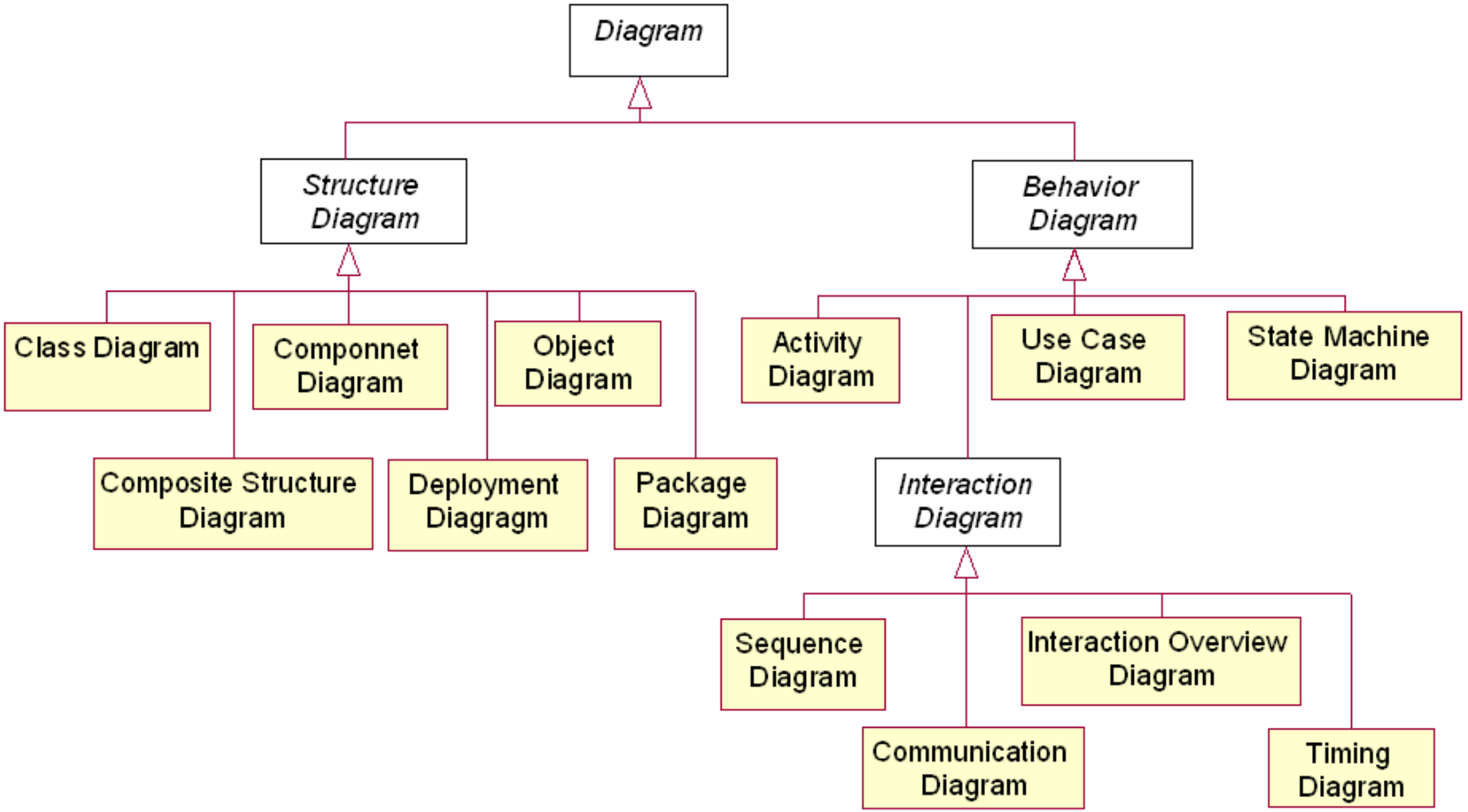
## … UML Diagrams

- UML was conceived as a language for modeling software. Since this includes requirements, UML supports world modeling (...at least to some extent).

- UML offers a variety of diagrammatic notations for modeling static and dynamic aspects of an application.

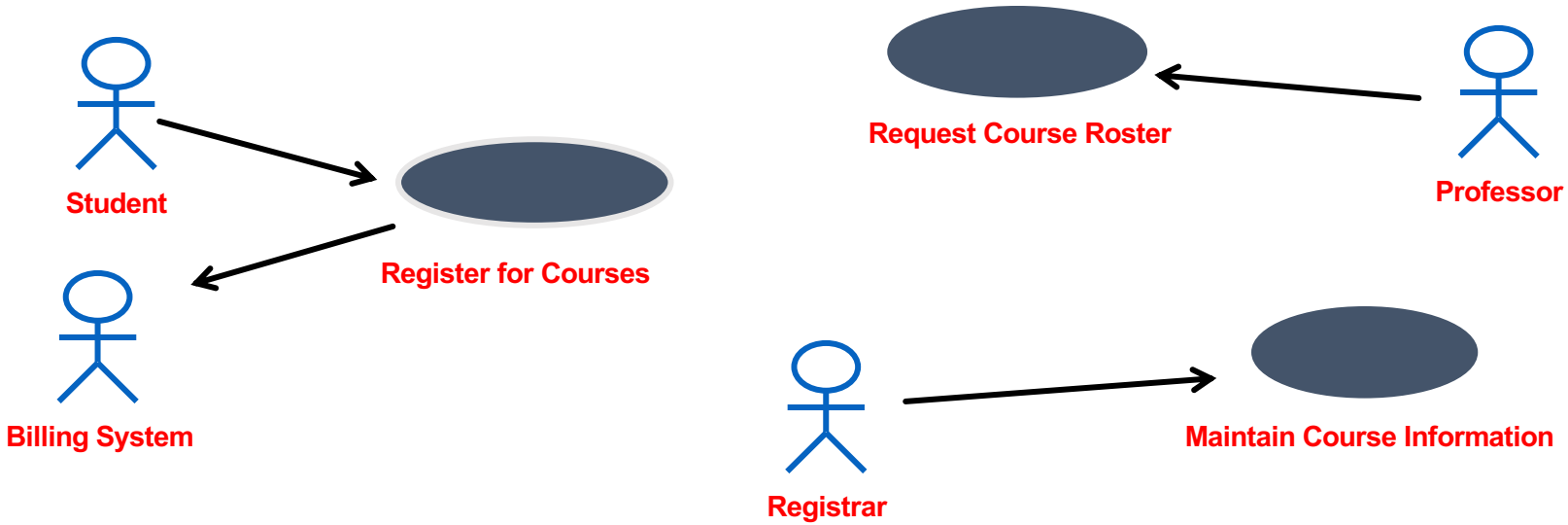
- The list of notations includes use case diagrams, class diagrams, interaction diagrams – describe sequences of events, package diagrams, activity diagrams, state diagrams, …more...

# Classification of Diagrams in the UML 2.0



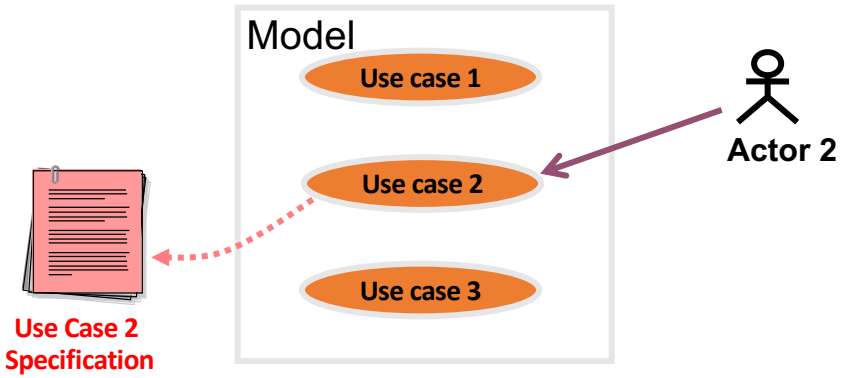
# Use-Case Diagram

- A use-case diagram is created to visualize the interaction of your system with the outside world.

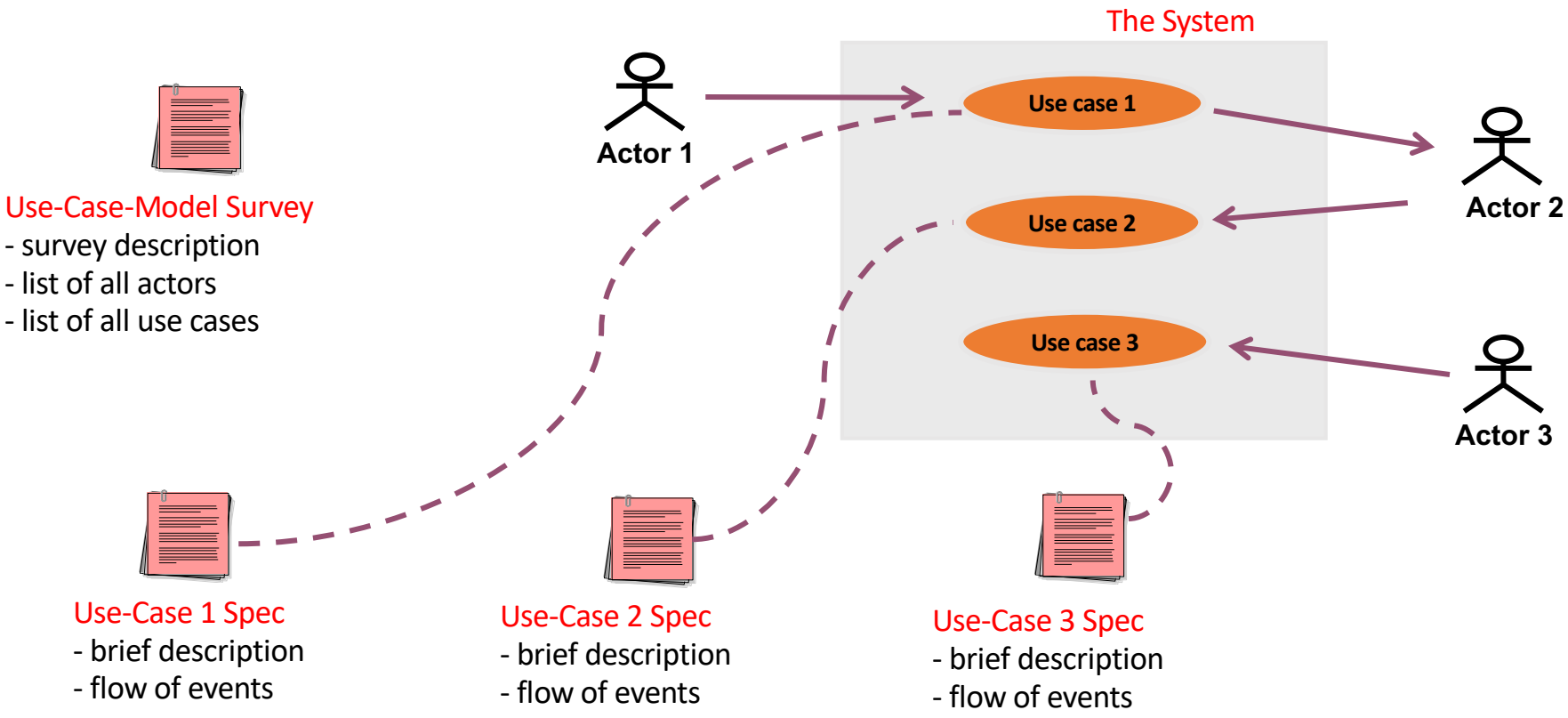


# What Is Use-Case Modeling?

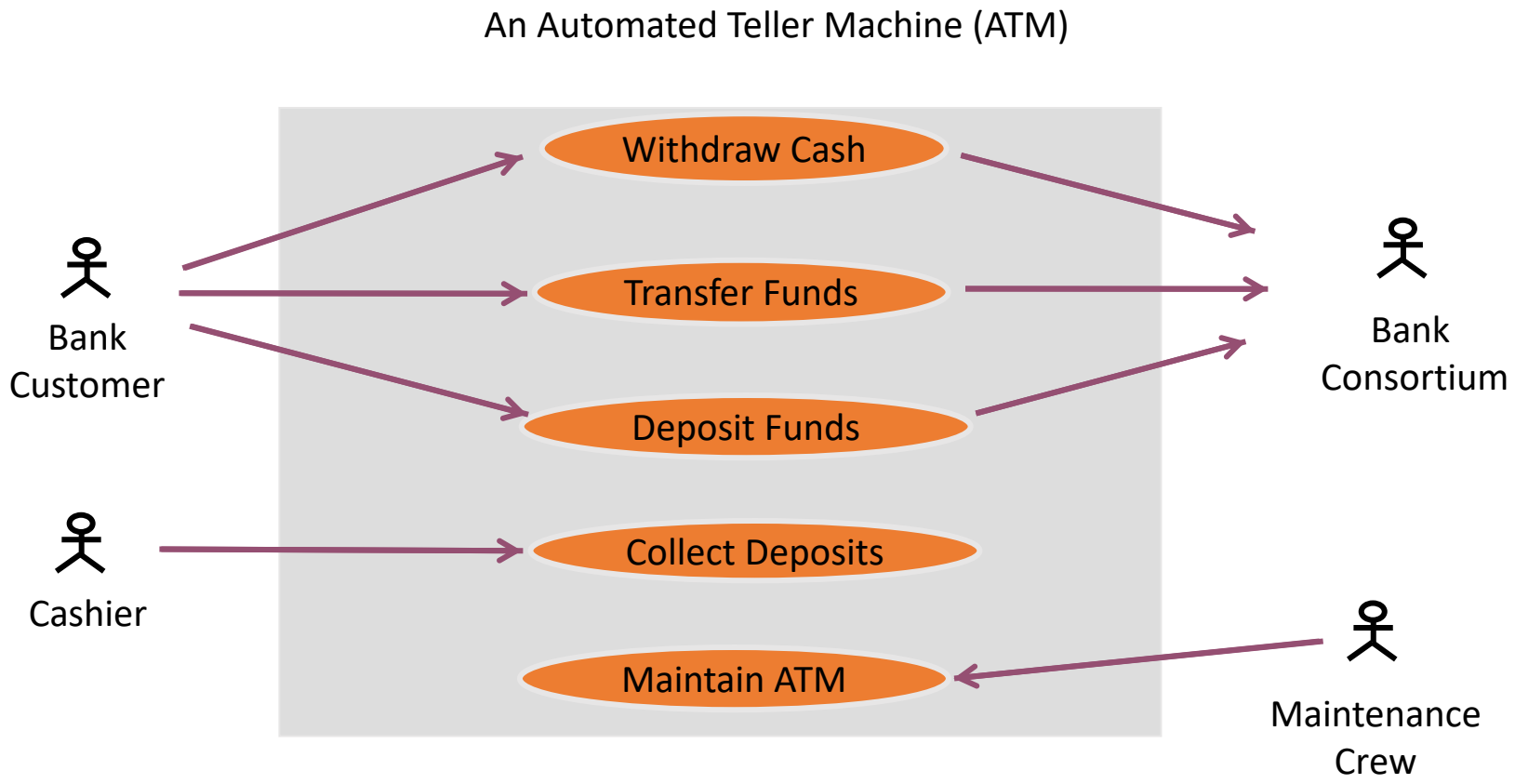
- Links stakeholder needs to software requirements.
- Defines clear boundaries of a system.
- Captures and communicates the desired behavior of the system.
- Identifies who or what interacts with the system.
- Validates/verifies requirements.
- Is a planning instrument.



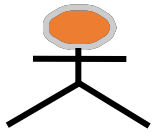
# A Use-Case Model is Mostly Text



# Use-Case Diagram



# Major Use-Case Modeling Elements



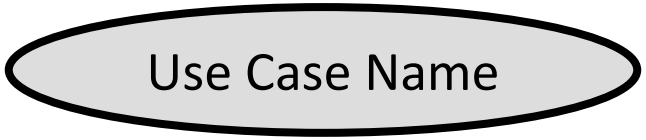

*Actor*

Someone/something outside the system, acting in a role that interacts with the system


*Use case*

Represents something of value that the system does for its actors

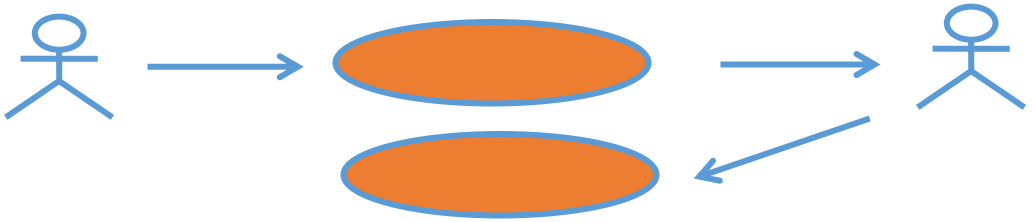
# What Is a Use Case?




A use case defines a sequence of actions performed by a system that yields an observable result of value to an actor.

# Use Cases Contain Software Requirements

- Each use case
  - Describes actions the system takes to deliver something of value to an actor.
  - Shows the system functionality an actor uses.
  - Models a dialog between the system and actors.
  - Is a complete and meaningful flow of events from the perspective of a particular actor.

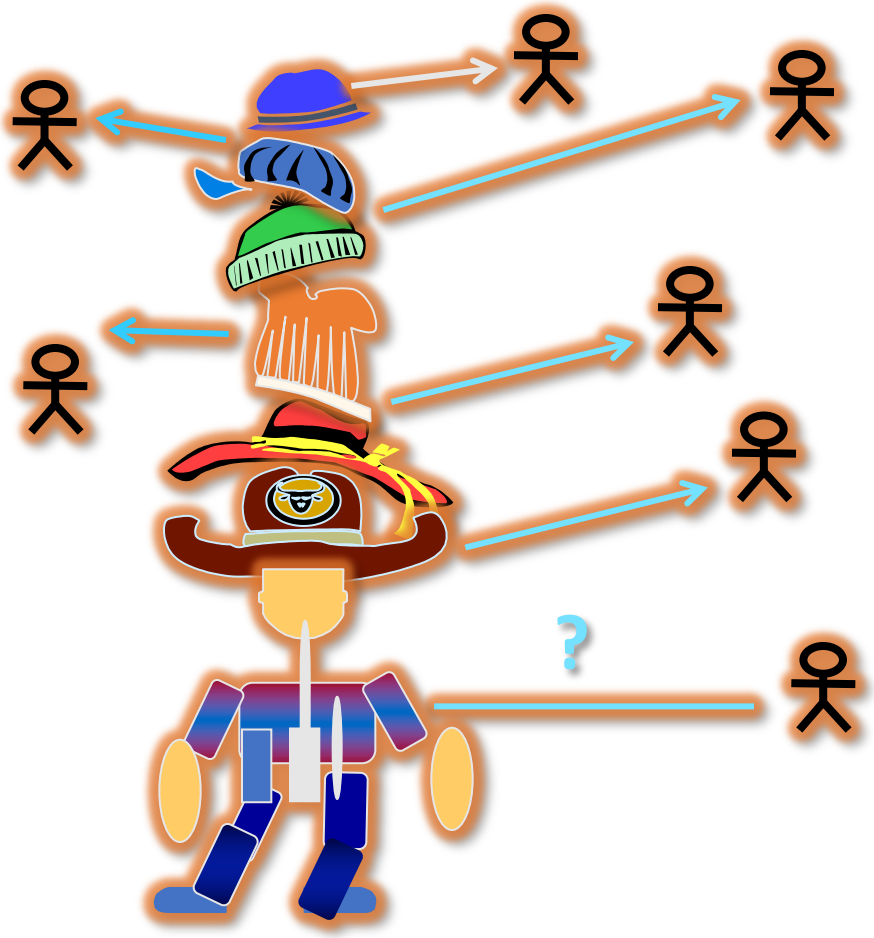


# Benefits of Use Cases

- Give context for requirements.
  - Put system requirements in logical sequences.
  - Illustrate why the system is needed.
  - Help verify that all requirements are captured.
- Are easy to understand.
  - Use terminology that customers and users understand.
  - Tell concrete stories of system use.
  - Verify stakeholder understanding.
- Facilitate agreement with customers.
- Facilitate reuse: test, documentation, and design.

# Define Actors: Focus on the Roles

- An actor represents a role that a human, hardware device, or another system can play in relation to the system.
- Actor names should clearly denote the actor's role.



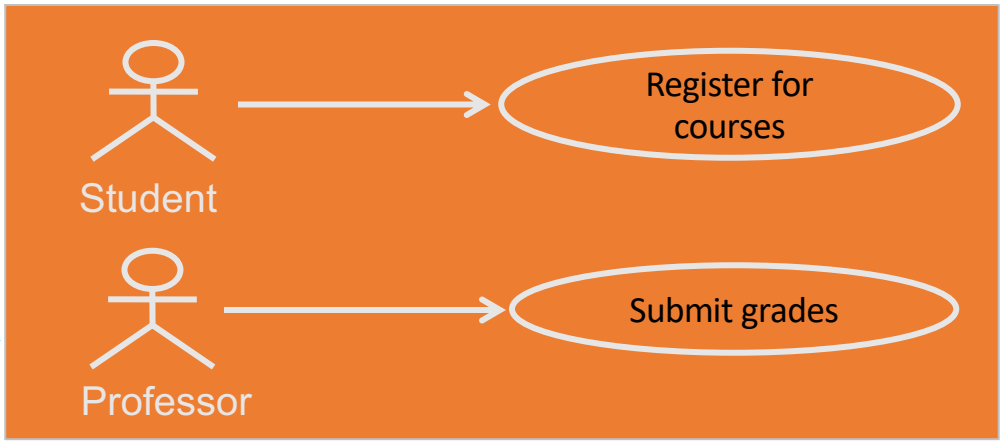
# Actors and Roles

**Charlie**: Is employed as a math professor and is an economics undergraduate.

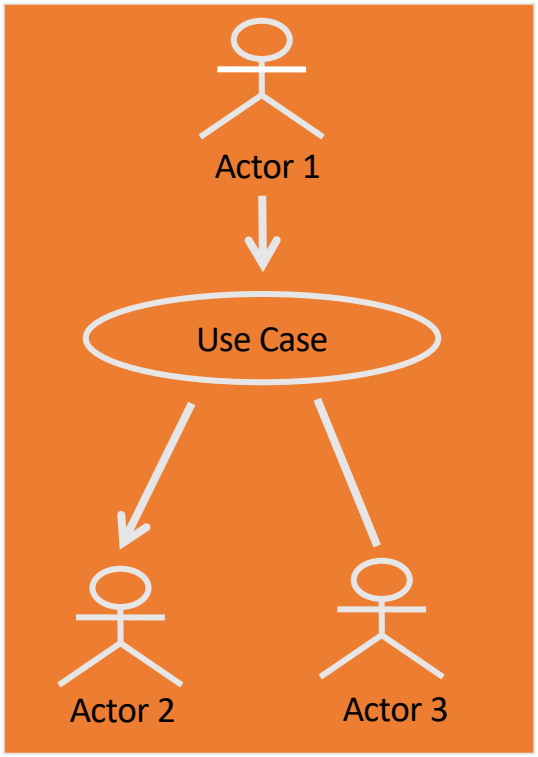
**Jodie**: Is a science undergraduate.

Charlie and Jodie both act as a Student.

Charlie also acts as a Professor.

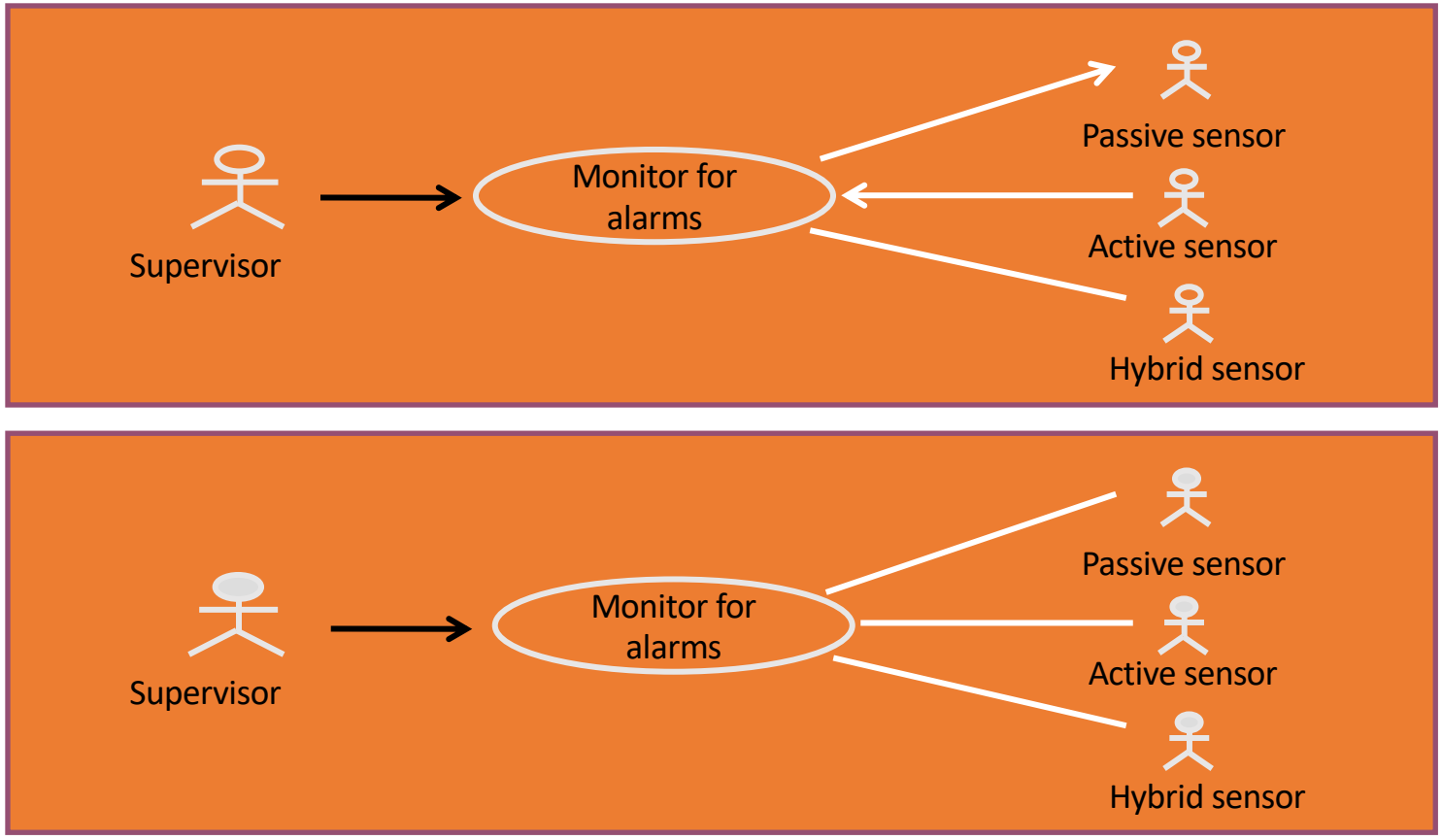


# Communicates-Association




- A channel of communication between an actor and a use case.
- A line is used to represent a communicates-association.
  - An arrowhead indicates who initiates each interaction.
  - No arrowhead indicates either end **can** initiate each interaction.

# Arrowhead Conventions



# A Scenario Is a Use-Case Instance


Scenario 1
Log on to system.
Approve log on.
Enter subject in search.
Get course list.
Display course list.
Select courses.
Confirm availability.
Display final schedule.

Scenario 2
Log on to system.
Approve log on.
Enter subject in search.
**Invalid subject.**
**Re-enter subject.**
Get course list.
Display course list.
Select courses.
Confirm availability.
Display final schedule.

# How Should I Name a Use Case?

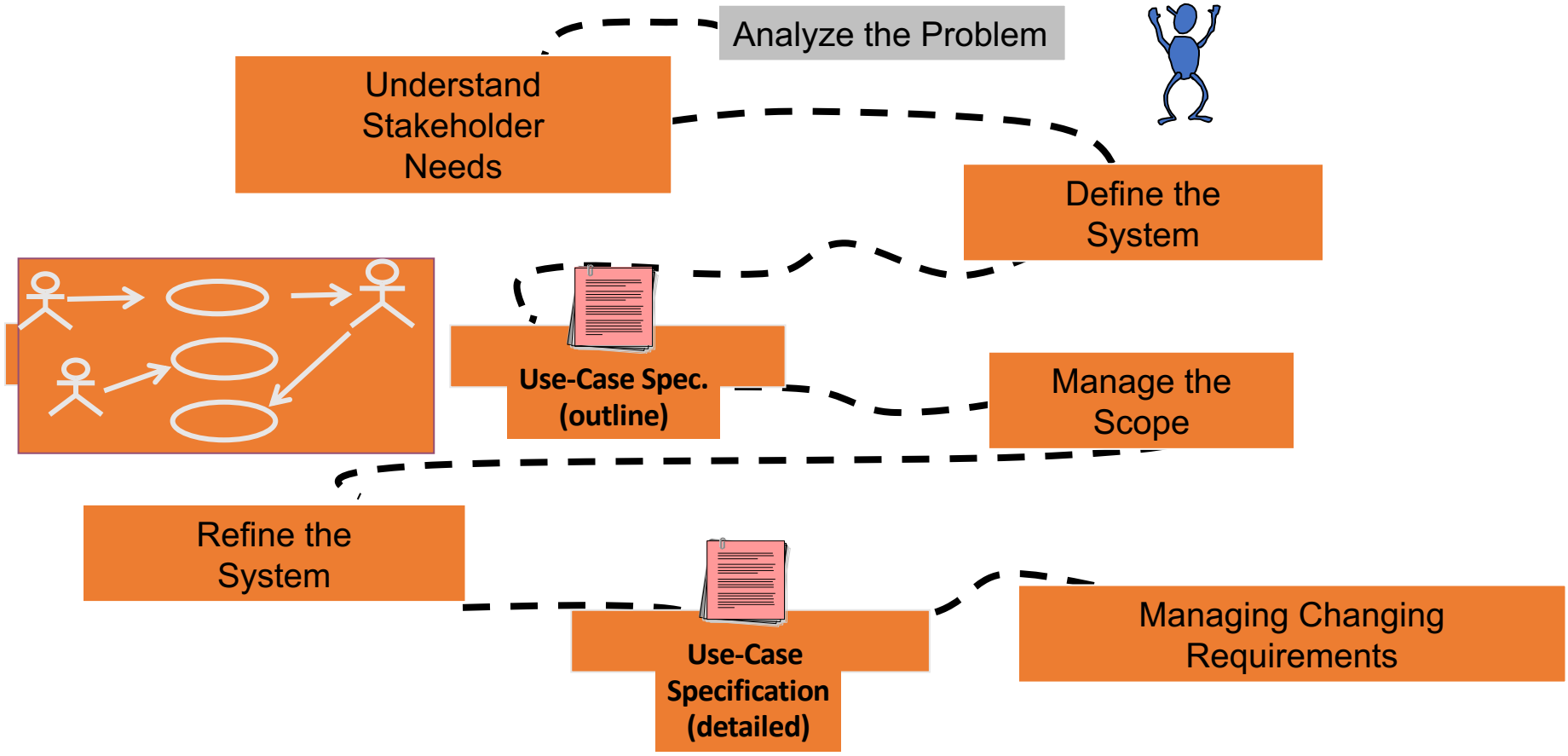
- Indicate the value or goal of the actor.
- Use the active form; begin with a verb.
- Imagine a to-do list.
- Examples of variations
  - Register for Courses
  - Registering for Courses
  - Acknowledge Registration
  - Course Registration
  - Use Registration System

Which variations show the value to the actor? Which do not? Which would you choose as the use-case name? Why?

# Steps for Creating a Use-Case Model

- Find actors and use cases.
  - Identify and briefly describe actors.
  - Identify and briefly describe use cases.
- Write the use cases.
  - Outline all use cases.
  - Prioritize the use-case flows.
  - Detail the flows in order of priority.

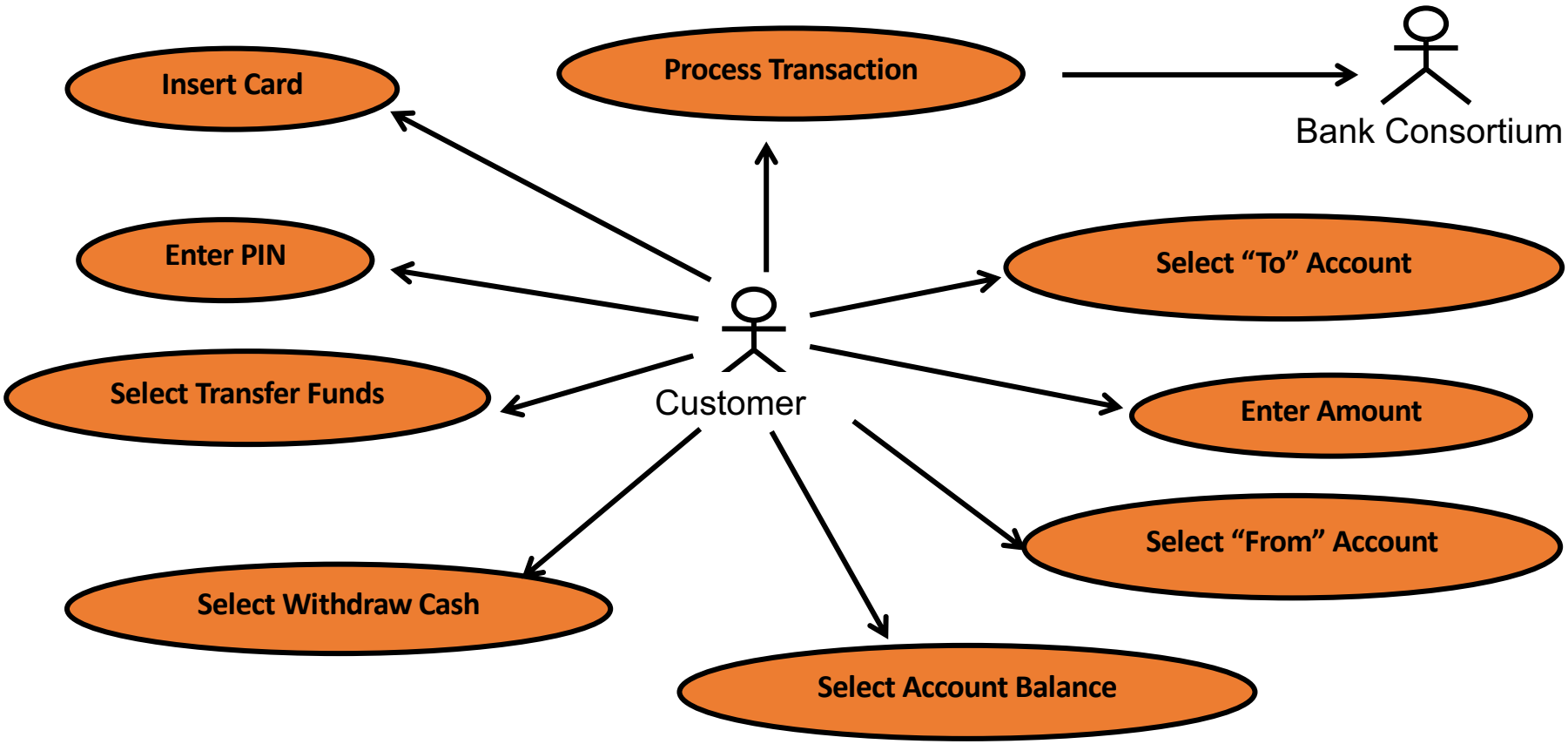
# Where Do Use Cases Fit into the RM Process?



# Functional Decomposition

- Is breaking down a problem into small, isolated parts.
  - The parts work together to provide the functionality of the system.
    - Often do not make sense in isolation.
- Use cases:
  - Are NOT functional decomposition.
  - Keep the functionality together to describe a complete use of the system.
  - Provide context.

# Functional Decomposition: An Example



# Avoid Functional Decomposition

Symptoms

- Very small use cases
- Too many use cases
- Uses cases with no result of value
- Names with low-level operations
  - “Operation” + “object”
  - “Function” + “data”
  - Example: “Insert Card”
- Difficulty understanding the overall model

Corrective Actions

- Search for larger context

“Why are you building this system?”

- Put yourself in user’s role

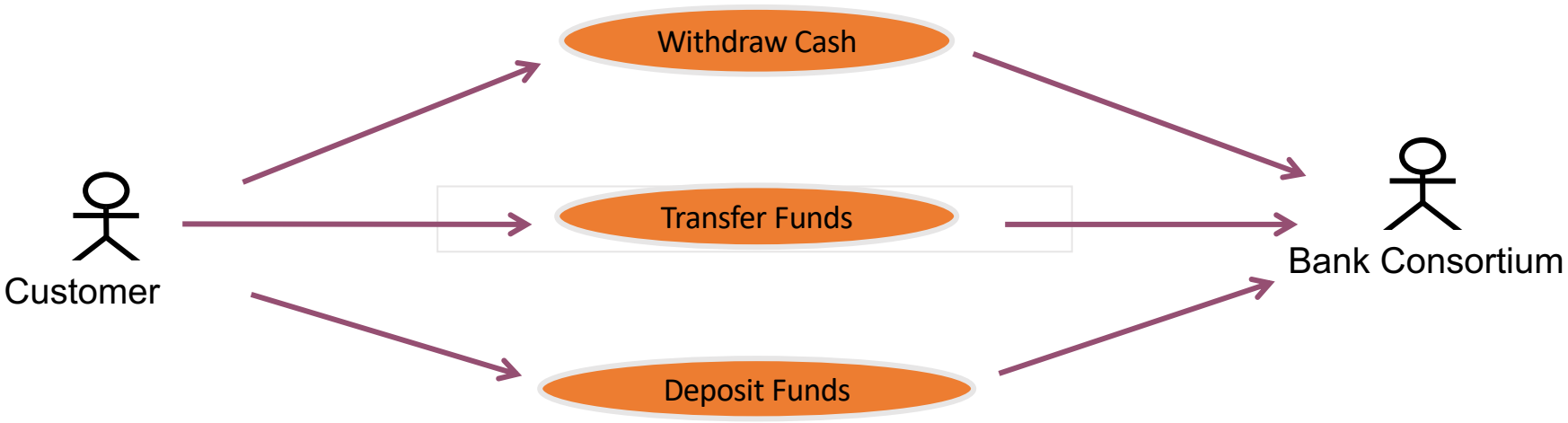
“What does the user want to achieve?”

“Whose goal does this use case satisfy?”

“What value does this use case add?”

“What is the story behind this use case?”

# Functional Decomposition: A Corrected Example

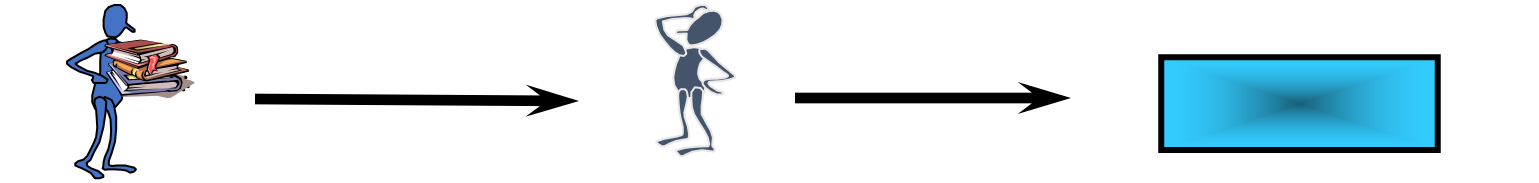


# Steps for Creating a Use-Case Model

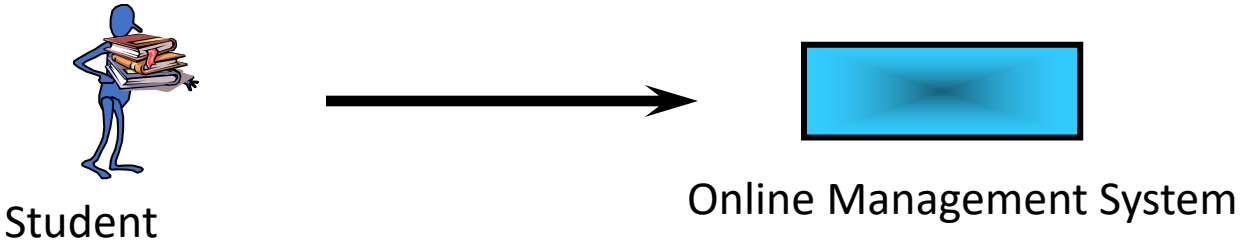
- Find actors and use cases.
  - Identify and briefly describe actors.
  - Identify and briefly describe use cases.
- Write the use cases.
  - Outline all use cases.
  - Prioritize the use-case flows.
  - Detail the flows in order of priority.

# Find Actors

Who is pressing the keys (interacting with the system)?




The student never touches the system; the manager operates it.
Or, are you building an Internet application?



# Identify Actors

- Who/what uses the system?
- Who/what gets information from this system?
- Who/what provides information to the system?
- Where in the company is the system used?
- Who/what supports and maintains the system?
- What other systems use this system?

# Description of an Actor

Text

Name

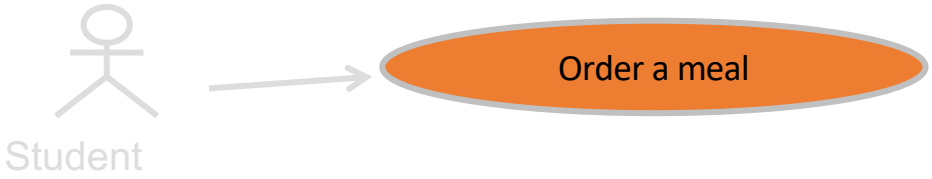
Student

Brief description

A person who eat in a dinning hall.

Relationships with

use cases

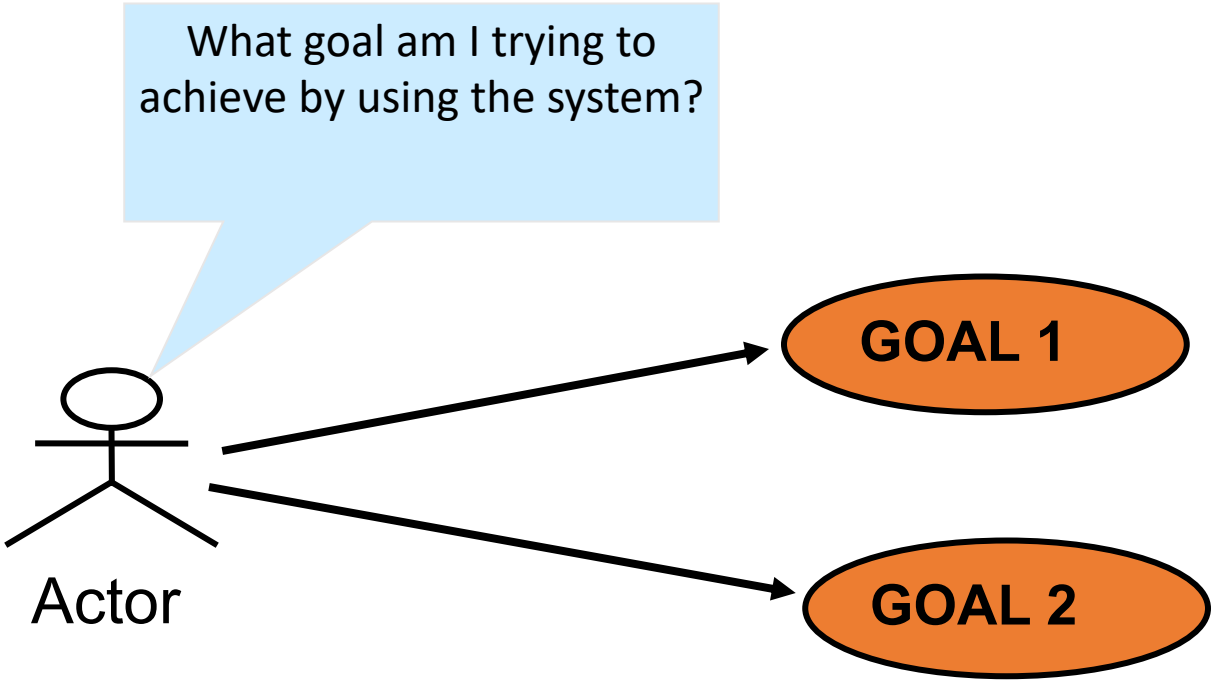



Use-Case-Model Survey

# Checkpoints for Actors

- Have you found all the actors? Have you accounted for and modeled all roles in the system's environment?
- Is each actor involved with at least one use case?
- Can you name at least two people who would be able to perform as a particular actor?
- Do any actors play similar roles in relation to the system? If so, merge them into a single actor.

# Find Use Cases



# 寻找用例的方法

- 和用户交互。
- 基本策略：把自己当作actor，与设想中的系统进行交互。

  考虑：
  - 系统交互的目的是什么？
  - 需要向系统输入什么信息？
  - 希望由系统进行什么处理并从它得到何种结果？
- **确定Use Case和确定actor不能截然分开。**

- Jacobson 提出以下问题：
  - 参与者的主要任务是什么？
  - 参与者要了解系统的什么信息？需要修改系统的什么信息？
  - 参与者是否需要把系统外部的变化通知系统？
  - 参与者是否希望系统把异常情况的变化通知自己？

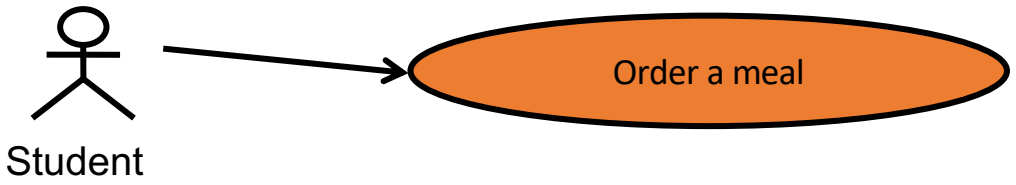
- 寻找用例时需要注意的问题：
  - 不要一开始就去捕捉所有的细节。
  - 全面地认识和定义每一个use case，**要点是以穷举的方式考虑每一个actor与系统的交互情况。**

# Identify Use Cases

- What are the goals of each actor?
  - Why does the actor want to use the system?
  - Will the actor create, store, change, remove, or read data in the system? If so, **why**?
  - Will the actor need to inform the system about external events or changes?
  - Will the actor need to be informed about certain occurrences in the system?
- Does the system supply the business with all of the correct behavior?

# Description of a Use Case

Text description of a use case.

| | |
|---|---|
| Name | Order a meal |
| Brief description | The student selects the food they wish to eat |
| Relationships with actors |  |

# Checkpoints for Use Cases

- The use-case model presents the behavior of the system; it is easy to understand what the system does by reviewing the model.
- All use cases have been identified; the use cases collectively account for all required behavior.
- All features map to at least one use case.
- The use-case model contains no superfluous behavior; all use cases can be justified by tracing them back to a functional requirement.
- All **CRUD** use cases have been removed. ✓
  - **C**reate, **R**etrieve, **U**pdate, **D**elete

# Use Case图的建立步骤

(1) 找出系统外部的参与者和外部系统，确定系统的边界和范围；

(2) 确定每一个参与者所期望的系统行为；

(3) 把这些系统行为命名为Use Case；

(4) 使用泛化、包含、扩展等关系处理系统行为的公共或变更部分；

(5) 编制每一个Use Case的脚本；

(6) 绘制Use Case图；

(7) 区分主事件流和异常情况的事件流，可以把表示异常情况的事件流作为单独的Use Case处理；

(8) 细化Use Case图，解决Use Case间的重复与冲突问题。

# 简洁用例的例子- Briefly Described Use Cases

- 简洁用例:

  简洁的一段摘要，主要是成功场景

- 处理销售:

  客户带着要购买的货物到达收款处，出纳员使用POS系统记录每一个购买的货物。系统提供总价和详细条目。客户输入付款信息供系统验证并记录。系统更新库存，客户得到收条并带着货物离开。

# 临时用例的例子- Outlined Use Cases Example

- 临时用例：
  - 非正式、随意的格式
  - 非正式段落，覆盖各种场景

## 退货处理

*主要成功场景:*

客户带着要退的货物到达收款处，出纳员使用POS系统记录每一个要退货的货物,...

*候选场景:*

若信用验证失败，通知客户并要求使用其他付款方法

若系统检测到与外界计税系统通信失败, ...

# 详细用例的例子- Fully Described Use Cases Example

## Use Case Description

Name: Place Order

Precondition: A valid user has logged into the system.

Description:

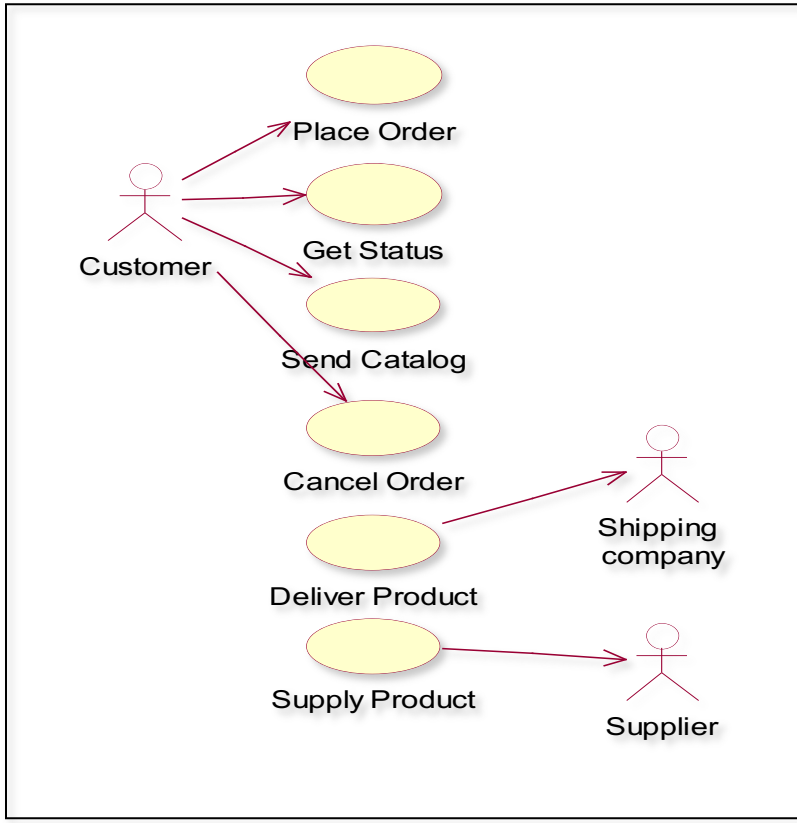
1. The use case starts when the customer selects Place Order.
2. The customer enters his or her name and address.
3. If the customer enters only the zip code, the system will supply the city & state.
4. The customer will enter product codes for the desired products.
5. The system will supply a product description and price for each item.
6. The system will keep a running total of items ordered as they are entered.
7. The customer will enter credit card payment information.
8. The customer will select Submit.
9. The system will verify the information, save the order as pending, and forward payment information to the accounting system.
10. When payment is confirmed, the order is marked Confirmed, an order ID is returned to the customer, and the use case ends.

Exceptions:

In step 9, if any information is incorrect, the system will prompt the customer to correct the information.

Postcondition: The order has been saved in the system and marked confirmed.

## Use Case Diagram



# 主成功流程

1. 顾客插卡
2. ATM读卡
3. ATM提示选择语言
4. 顾客选择英语
5. ATM提示输入密码
6. 顾客输入密码
7. ATM显示待选服务
8. 顾客选择取款
9. ATM提示输入取款数额，必为50的倍数
10. 顾客输入数值
11. ATM通知银行中央系统客户的取款数额
12. 银行中央系统接受请求，并通知ATM新的账户余额
13. ATM输出现金
14. ATM询问顾客是否要收据
15. 顾客回答需要收据
16. ATM输出并打印收据
17. ATM记录该项交易

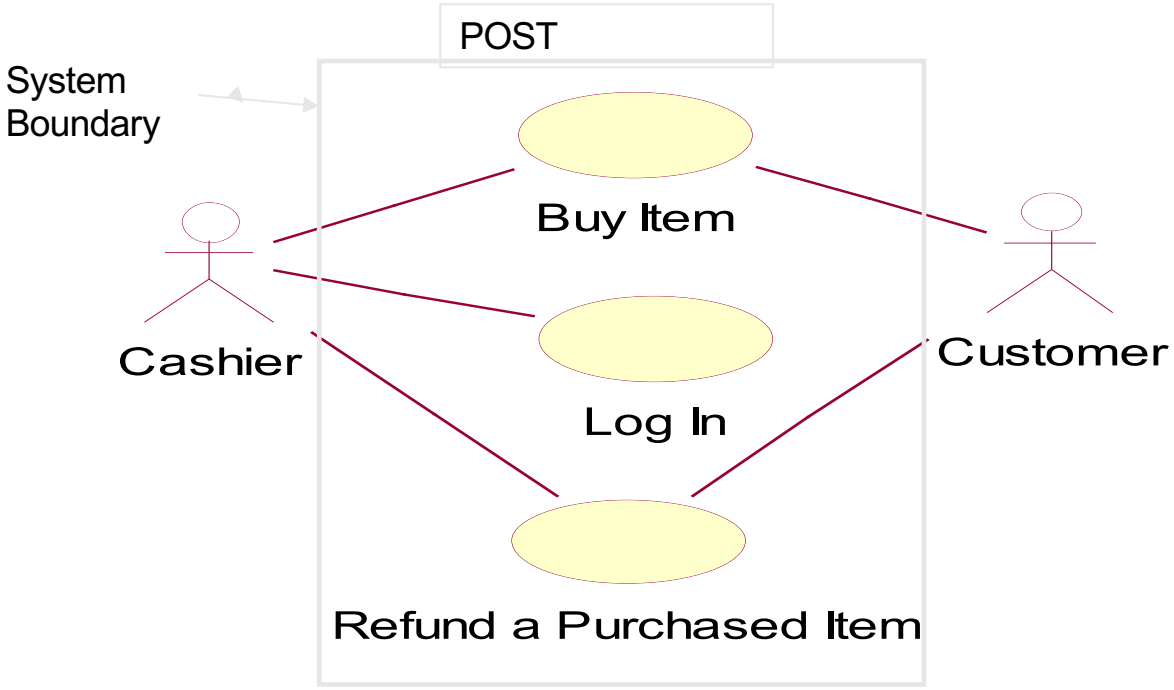
例外情况：

- Card reader broken or card scratched 读卡异常
- Card for an ineligible bank 非本机接受卡种
- Incorrect PIN 错误密码
- Customer does not enter PIN in time 输入密码超时
- ATM is down 机器停机
- Host computer is down, or network is down 主机坏或网络断
- Insufficient money in account 账户余额不足
- Customer does not enter amount in time 输入数额超时
- Not a multiple of $50 数额不合要求
- Amount requested is too large 数额过大
- Network or host goes down during transaction 交易期间系统坏
- Insufficient cash in dispenser ATM现金量不足
- Cash gets jammed during dispensing 出闭口夹纸
- Receipt paper runs out, or gets jammed 收据打印纸用完
- Customer does not take the money from the dispenser 顾客忘取现金

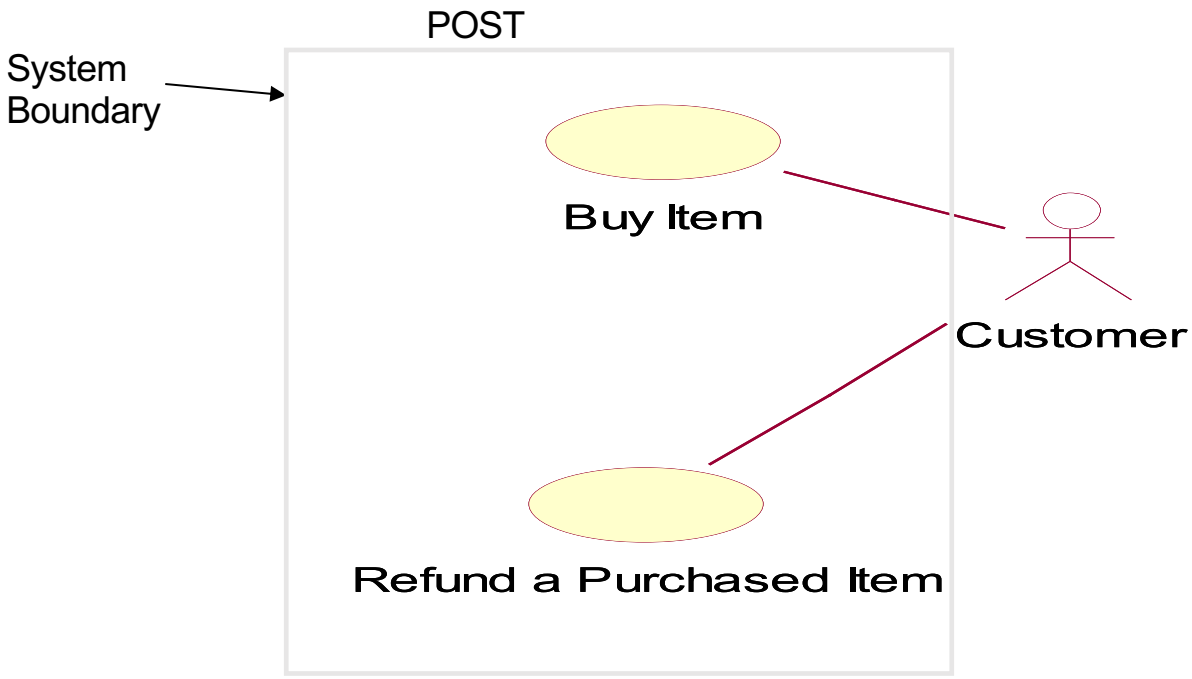
## Setting the system boundary 设定系统边界

- 系统边界会对用例以及Actor的定义有所影响
- 思考以下系统的用例图：
  - 记录销售及付款情况的计算机系统
  - 用于零售店
  - 包括硬件设备，如计算机、条码扫描装置
  - 运行在系统上的软件
  - 系统目标包括：
    - 自动收款
    - 快速准确的销售情况统计及分析
    - 自动的库存管理

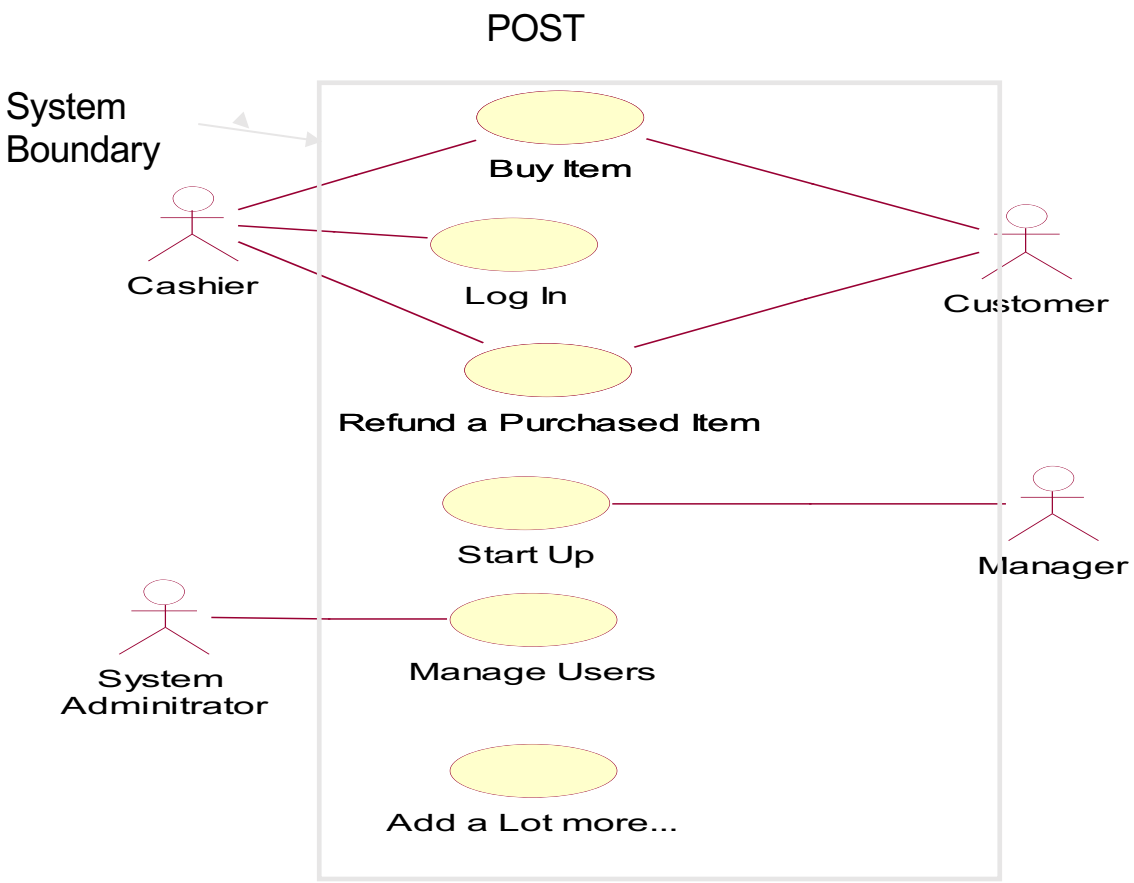
# System Boundary (系统边界定义之一)



# System Boundary (系统边界定义之二)



# System Boundary (系统边界定义之三)



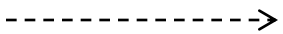
# 用例图中的一些主要图标



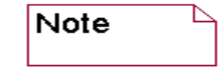


Association
关联

Generalization
泛化



Dependency
依赖



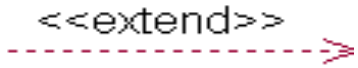

注释



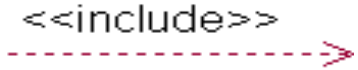
注释连接

Realize


use-case realization




extend use case




include use case

说明：UML中不使用颜色来作为图形语义的区分标记。

# Relationships between Use Case

- Use Case除了和参与者有**关联(association)**外，Use Case之间也存在着一定的关系(relationship)。包括：**泛化(generalization)关系**、**包含(include)关系**、**扩展(extend)关系**等。
- 也可以利用UML的扩展机制自定义Use Case间的关系。
- **relationship(关系)**, **association(关联)**, **generalization(泛化)**, **dependency(依赖)**的区别。
  - association, generalization, dependency都属于relationship。
  - include，extend 属于 dependency。

# Generalization(泛化关系)

- **泛化(generalization)**代表一般与特殊的关系。use case间的泛化关系与类之间的泛化关系（继承关系）类似。

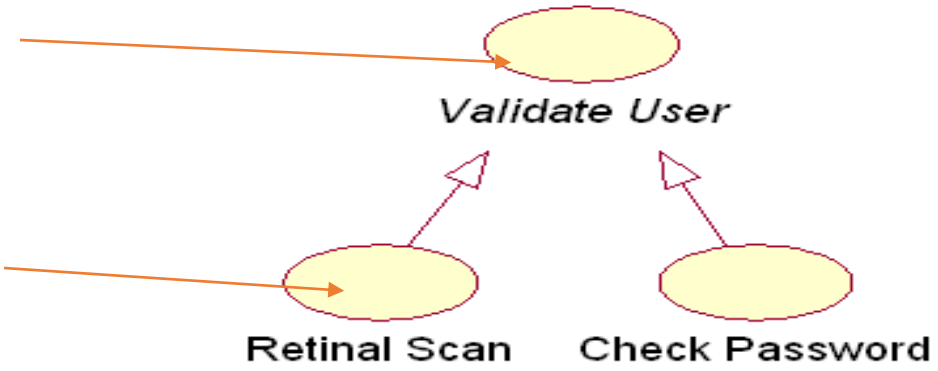
- **use case generalization** is a taxonomic relationship between a use case (the child) and the use case (the parent) that describes the characteristics the child shares with other use cases that have the same parent.

# Generalization example:

Use case behavior for Parent
Check Password:
1. Obtain password from master database
2. Ask user for password
3. User supplies password
4. Check password against user entry

parent use case

child use case




Use case behavior for child Retinal Scan
1. Obtain retinal signature from master database
2. Scan user's retina and obtain signature
3. Compare master signature against scanned signature

# More about Generalization

- **child use case** inherits the behavior and meaning of the **parent use case**;
- The **child** may add to or override the behavior and meaning of the **parent** use case;
- The **child** may be substituted any place the **parent** appears.

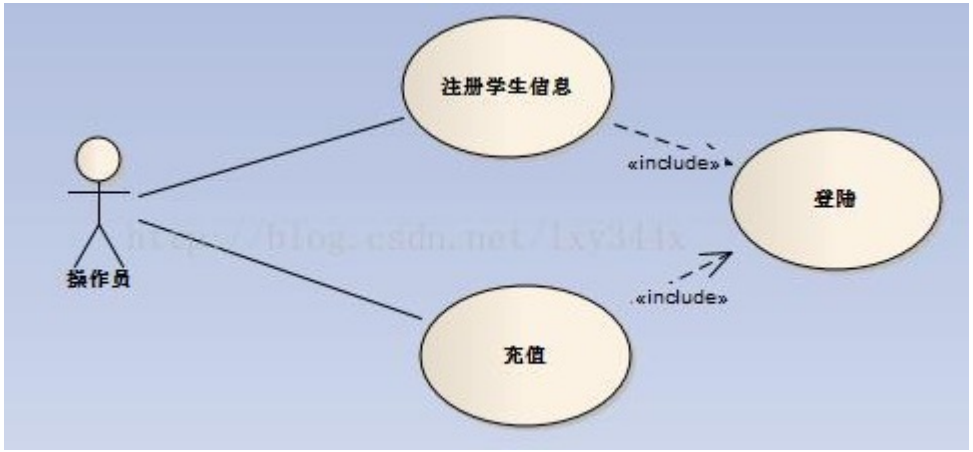
# include (包含关系)

- **包含(Include)** 关系是指一个基本Use Case的行为包含了另一个Use Case的行为。

- A relationship from a base use case to an inclusion use case, specifying how the behavior defined for the inclusion use case can be inserted into the behavior defined for the base use case.

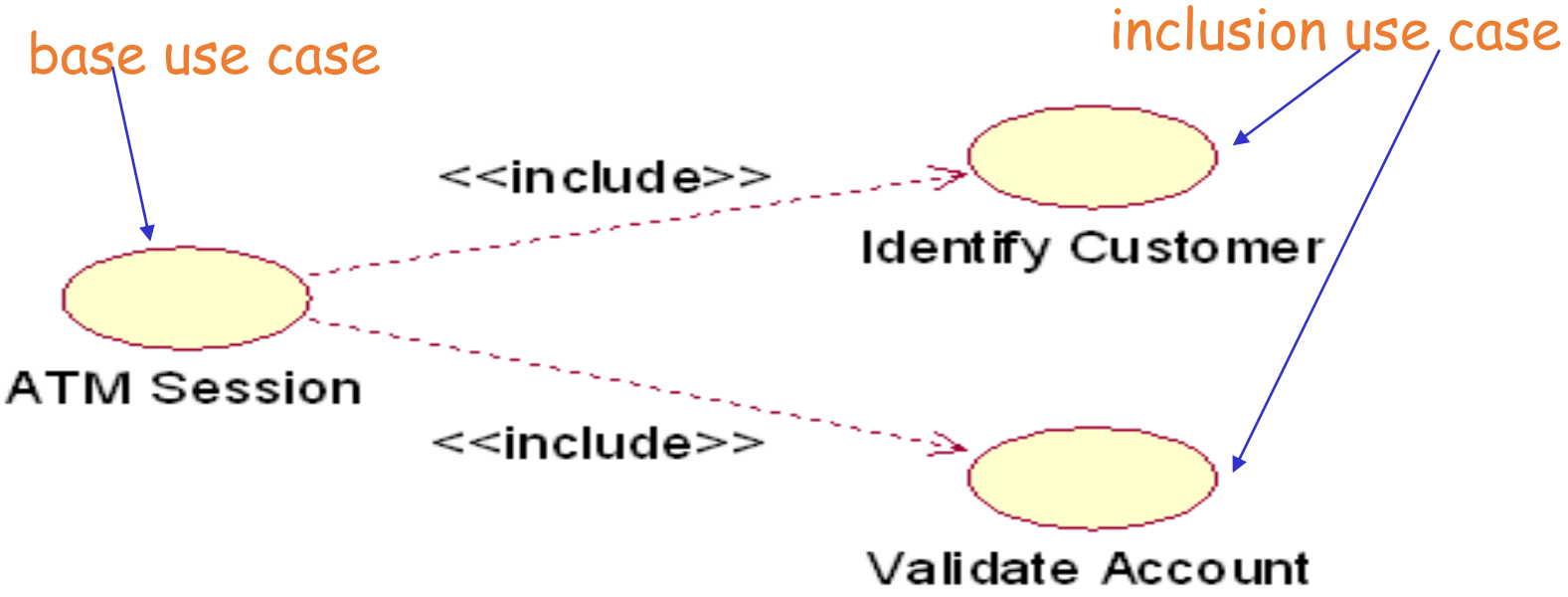
## When to use Includes？何时使用包含关系?

- You have a piece of behavior that is similar across many use cases （多个用例有共享行为）
- Break this out as a separate use-case and let the other ones "include" it （为共享行为单独创建用例)
- Examples include
  - Login (登录)




https://blog.csdn.net/lxy344x/article/details/40346525

- 包含关系的例子：


base use case
inclusion use case
<<include>>
Identify Customer
ATM Session
<<include>>
Validate Account

# extend (扩展关系)

- **扩展(extend)关系**的基本含义与泛化关系类似，如果一个用例明显地混合了两种或者两种以上的不同场景，即根据情况可能发生多种分支，则可以将这个用例分为一个基本用例和一个或多个扩展用例。

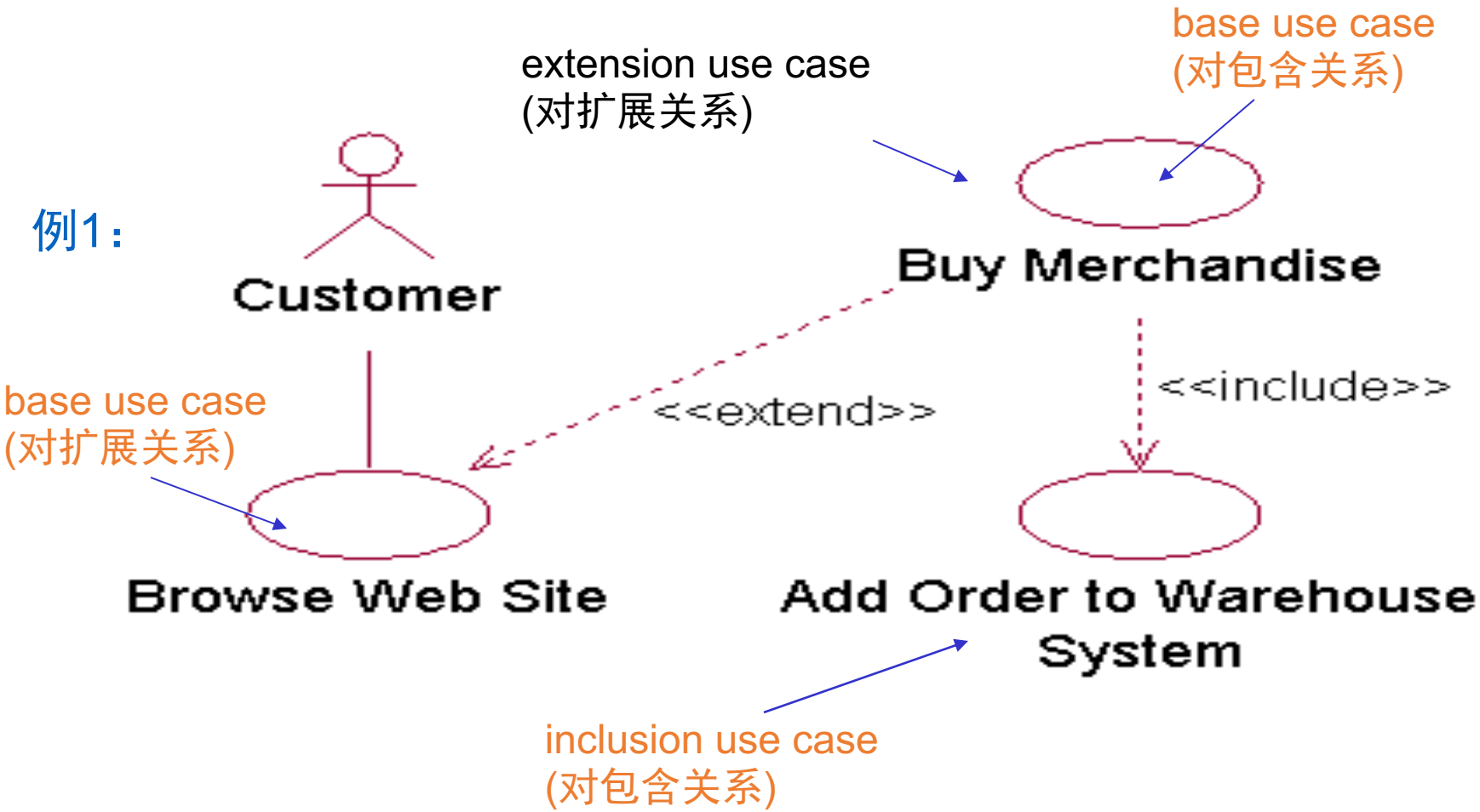
- A relationship from an extension use case to a base use case, specifying how the behavior defined for the extension use case can be inserted into the behavior defined for the base use case.

## When to use Extends？何时使用扩展关系?

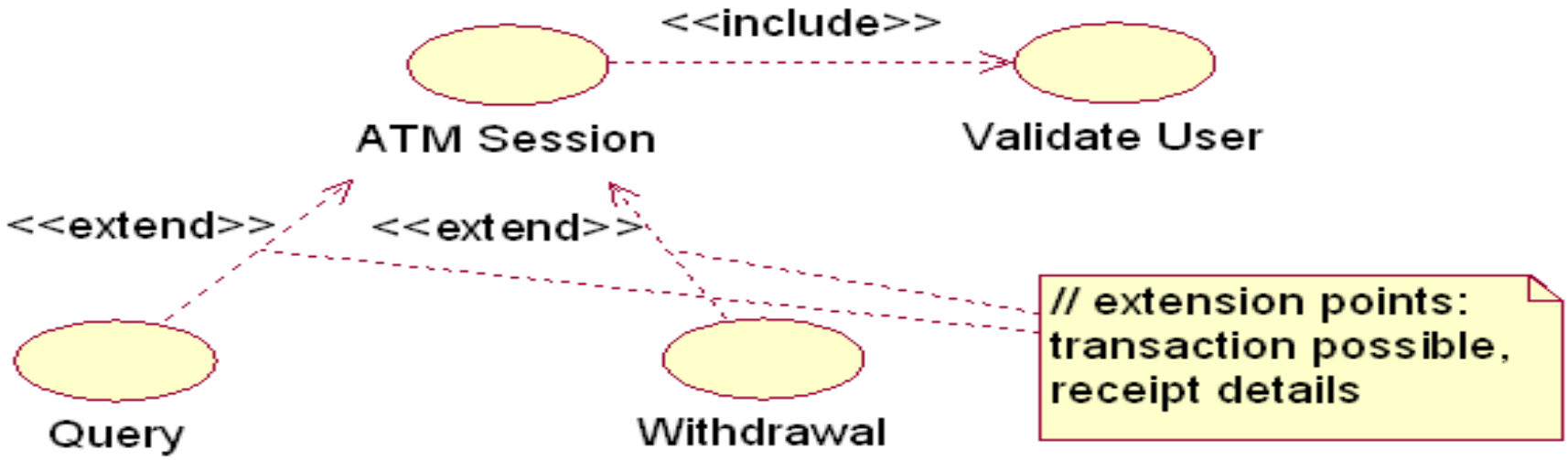
- A use case is similar to another one but does a little bit more （一个用例与另外一个用例近似，只有少许额外的活动）
- Put the normal behavior in one use-case and the exceptional behavior somewhere else （将代表普遍或基本行为的情况定义为一个用例，将特殊的、例外的部分定义为扩展用例)

# 扩展关系和包含关系的例子：

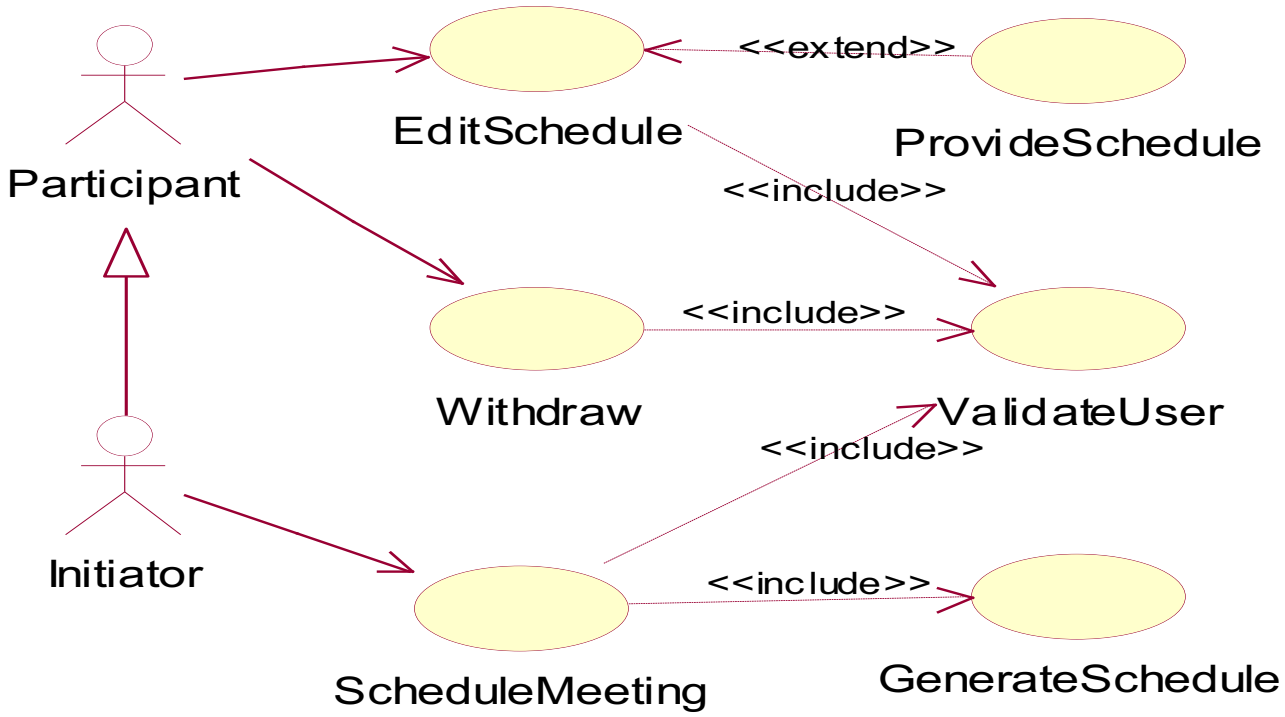
例1：



例2：


<<include>>
ATM Session
Validate User
<<extend>>
<<extend>>
// extension points:
transaction possible,
receipt details
Query
Withdrawal

例3:


Participant
EditSchedule
<<extend>>
ProvideSchedule
<<include>>
<<include>>
Withdraw
ValidateUser
<<include>>
Initiator
<<include>>
ScheduleMeeting
GenerateSchedule

# 常见问题分析

1. Use case的粒度问题，即对于一个系统的Use Case图，所包含的用例数目问题。

- 这是很多人争论的重点。例如，Jacobson说，对一个十人年的项目，他需要二十个用例。而在一个相同规模的项目中，Martin Fowler 则用了一百多个用例。

*《UML distilled》一书的作者

2. 许多应用中需要对系统访问进行控制，应该如何处理登录问题比较好?

- 有四种处理登录用例的方式:

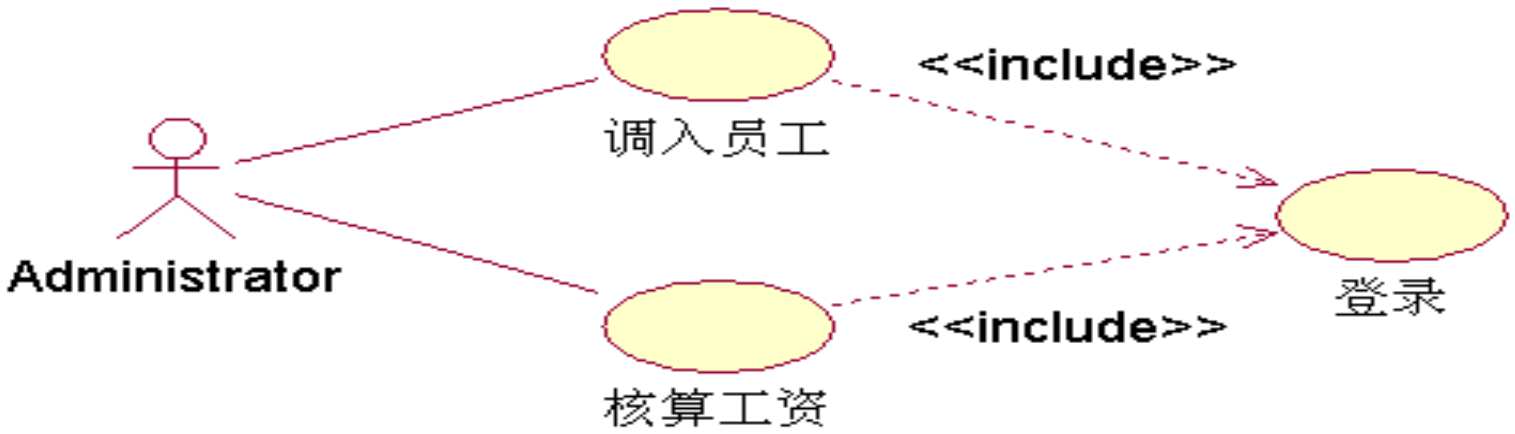
(1) 其它用例包含登录;

(2) 其它用例扩展登录;

(3) 登录独立于其它用例;

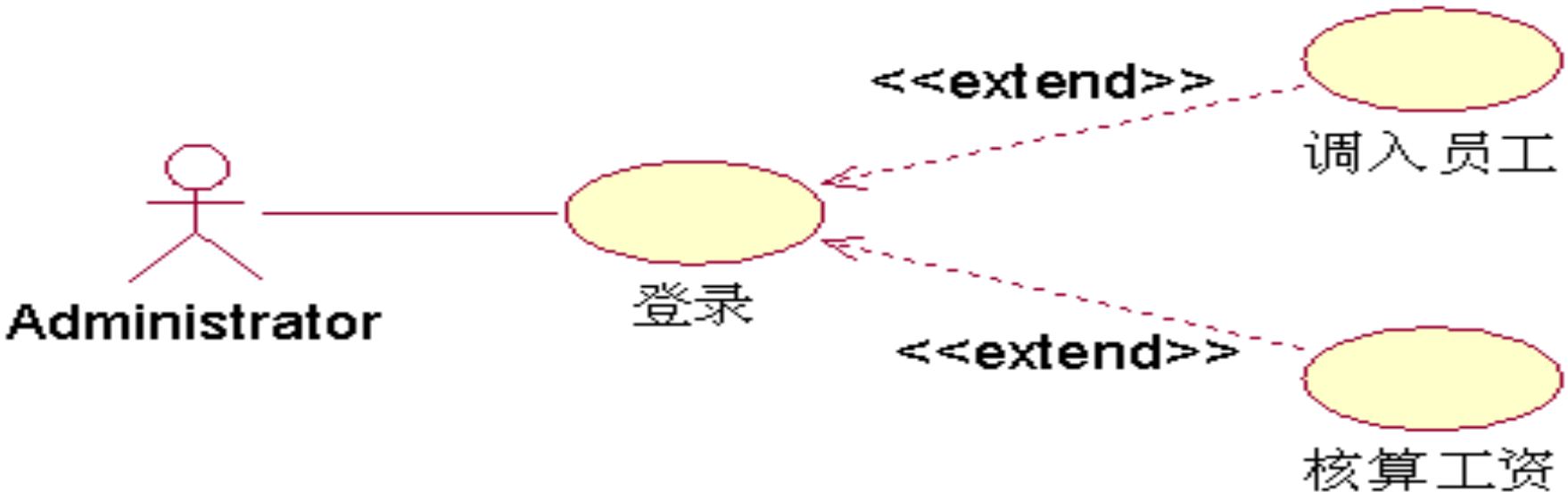
(4) 登录包含其它用例。

(1) 其它用例包含登录




用例说明: ……
这种处理方式的特点:

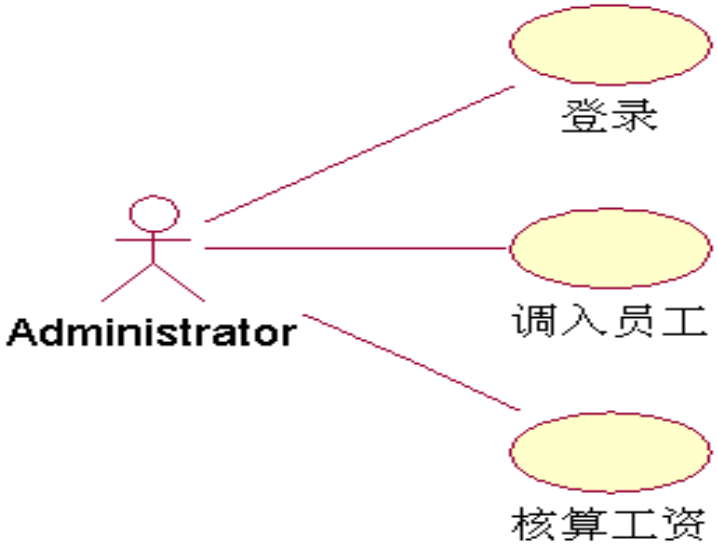
(2) 其它用例扩展登录




用例说明: .....

这种处理方式的特点:

(3) 登录独立于其它用例。



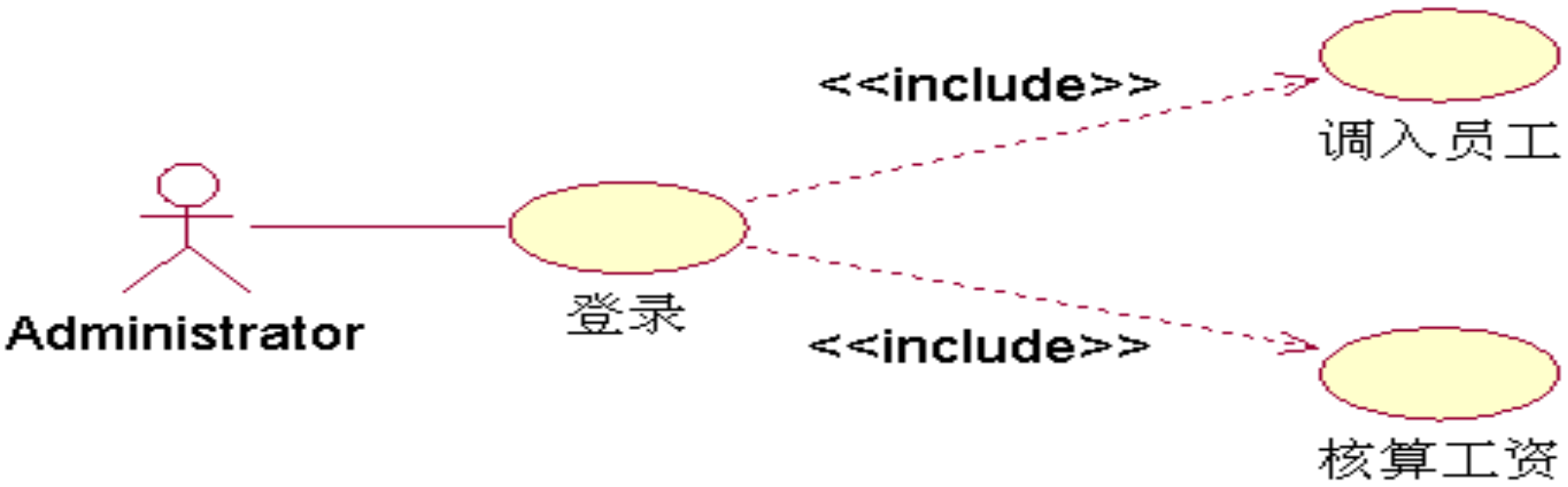

用例说明: .....
这种处理方式的特点:

- 登录用例说明:
    1. 当超级用户启动应用时用例开始
    2. 系统提示超级用户输入用户名和密码
    3. 超级用户输入用户名和密码
    4. 系统验证其是否为有效超级用户
    5. 用例结束

- 调入员工用例说明:

前置条件: 一个有效超级用户登录了系统

(4) 登录包含其它用例

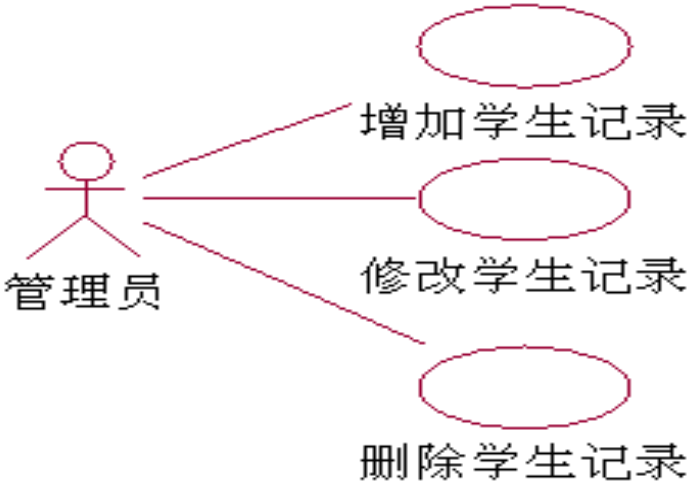



用例说明: .....

存在的问题:

3. 假设有这样需求：学生档案管理中，用户经常需要做三件事：增加一条学生记录、修改一条学生记录，删除一条学生记录。如果要画出use case图，以下2种方法哪种更合适?

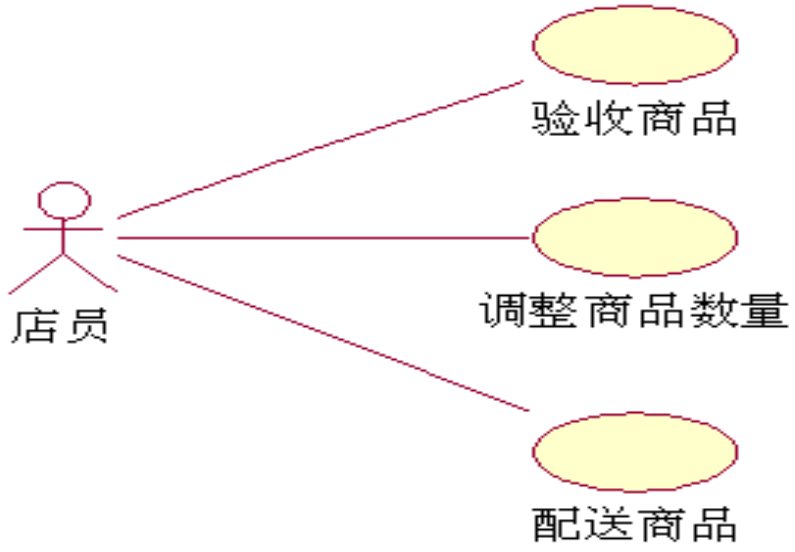

**方法1**：再分成3个脚本，分别画3个交互图。脚本1 增加学生记录；脚本2 修改学生记录；脚本3 删除学生记录。



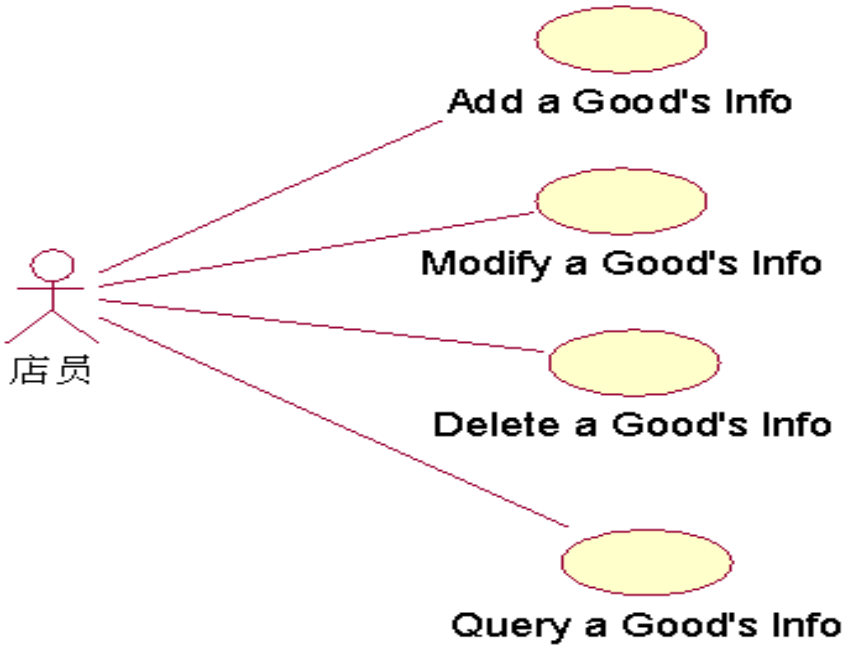

**方法2**：每个use case画一个交互图。

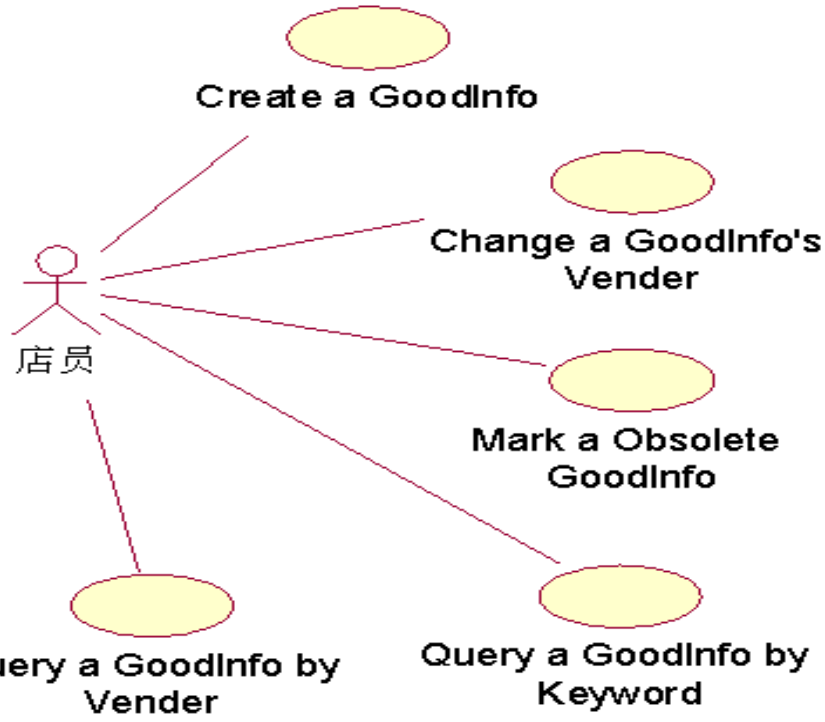
- **Create, Retrieve, Update, Delete类型用例的处理：**
  - 采用CRUD四个用例还是一个用例?
  - 从用户需求的角度考虑而不是从数据处理的角度考虑。
- 例2:


?



- 例3:


Add a Good's Info
Modify a Good's Info
Delete a Good's Info
Query a Good's Info
店员


Create a GoodInfo
Change a GoodInfo's Vender
Mark a Obsolete GoodInfo
Query a GoodInfo by Vender
Query a GoodInfo by Keyword
店员

?

# Class Exercise 问题描述

- 我们在为一家邮购公司开发订单处理软件，从供应商那里购买产品，再销售。
- 这家公司每年发布两次产品目录，并将其邮递给客户和其他感兴趣的人。
- 客户以提交商品列表并向邮购公司付费的方式购买商品。邮购公司填写帐单并把商品运送到客户的地址。
- 订单处理软件记录从收到订单直道商品被运送给客户的整个过程。
- 邮购公司将提供快捷的服务，以最快，最有效的方法来运送客户订购的产品。
- 客户可以退货，要求重新进货，但有时要付费。

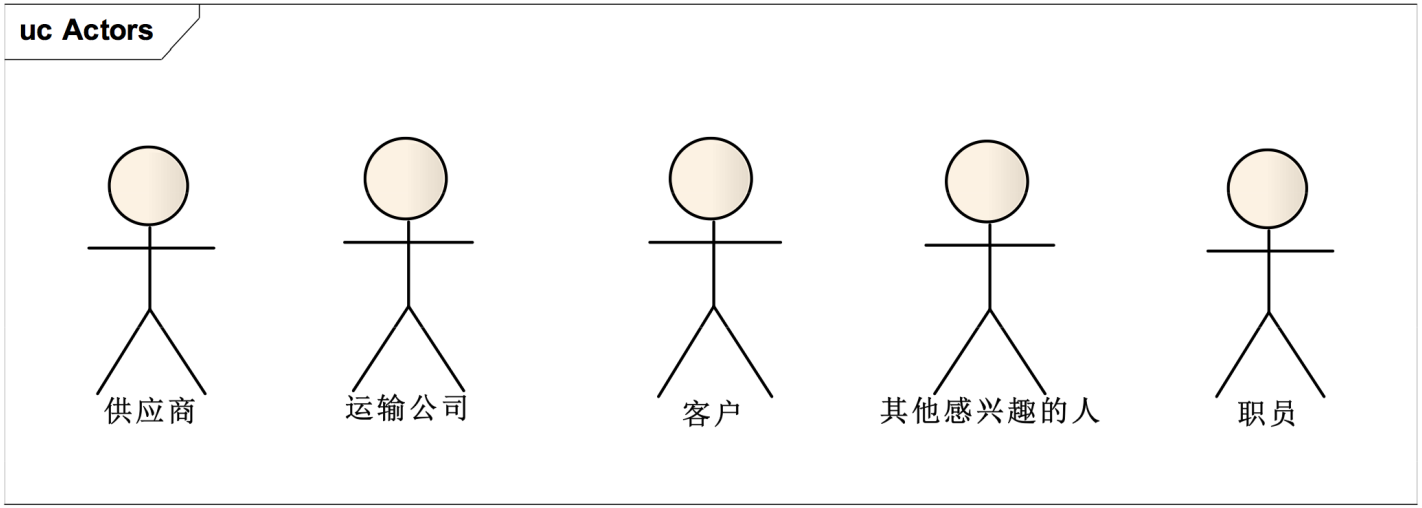
# 确定执行者— Actor

- 谁使用这个系统?
- 谁安装这个系统?
- 谁启动这个系统?
- 谁维护系统?
- 谁关闭系统?
- 哪些其他系统使用这个系统?
- 谁从这个系统获取信息?
- 谁为这个系统提供信息?
- 是否有事情自动在预计的时间发生?

# 干系人(Stakeholder)?
# 主、次参与者(Actor)? 系统？无关？

- 我们（开发人员）
- 邮购公司
- 供应商
- 产品
- 产品目录
- 客户
- 特快专递
- 其他感兴趣的人
- 商品列表
- 运输公司
- 订单处理软件
- 邮购公司职员
- Mary Smith （邮购公司客户）
- Mary Smith （邮购公司职员）
- 记帐系统
- 库存系统

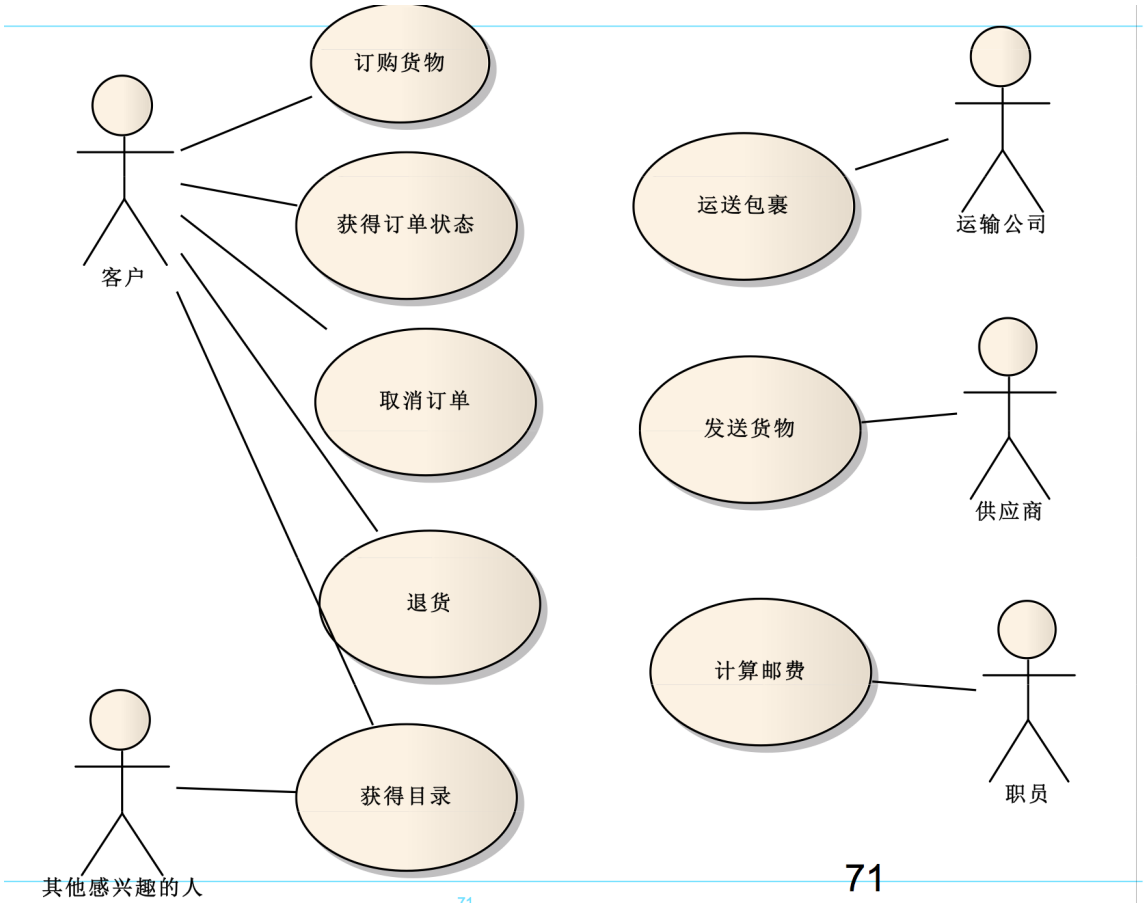
# 参与者



# 确定用例

- 执行者希望系统提供什么样的功能?
- 系统存储信息吗?
- 执行者将要创建、读取、更新或删除什么信息?
- 系统是否需要把自身内部状态的变化通知给执行者？系统必须知道哪些外部的事件？执行者将怎样通知系统这些事件?
- 其他需要考虑的用例包括启动、关闭、诊断、安装、维护、培训和改变商业过程。

# 用例


订购货物
获得订单状态
取消订单
退货
获得目录
客户
其他感兴趣的人
运送包裹
运输公司
发送货物
供应商
计算邮费
职员

# UC01：订购货物用例

前置条件：一个合法的用户已经登录到这个系统

事件流：

1. 当客户选择订购货物时，用例开始；
2. 客户输入想要购买的商品代码；
3. 系统逐项列出产品描述和价格；
4. 系统保存已经订购的产品清单；
5. 客户输入信用卡支付信息；
6. 客户选择提交；
7. 系统检验输入的信息；
   a. 如果提交的信息正确，保存订单，向记帐系统转发支付信息；
   b. 如果客户提交的信息不正确，系统提示用户修改；
8. 当支付确认后，订单就被标记上已确认，同时返回给客户一个订单ID，用例结束。
   a. 如果支付没有被确认，系统提示客户改正支付信息或取消；
   b. 如果客户选择修改信息，就回到第5步；
   c. 如果选择取消，用例结束。

后置条件：如果订单未被取消，则被系统保存起来，并被标记为已确认

# 找出可选路径的方法

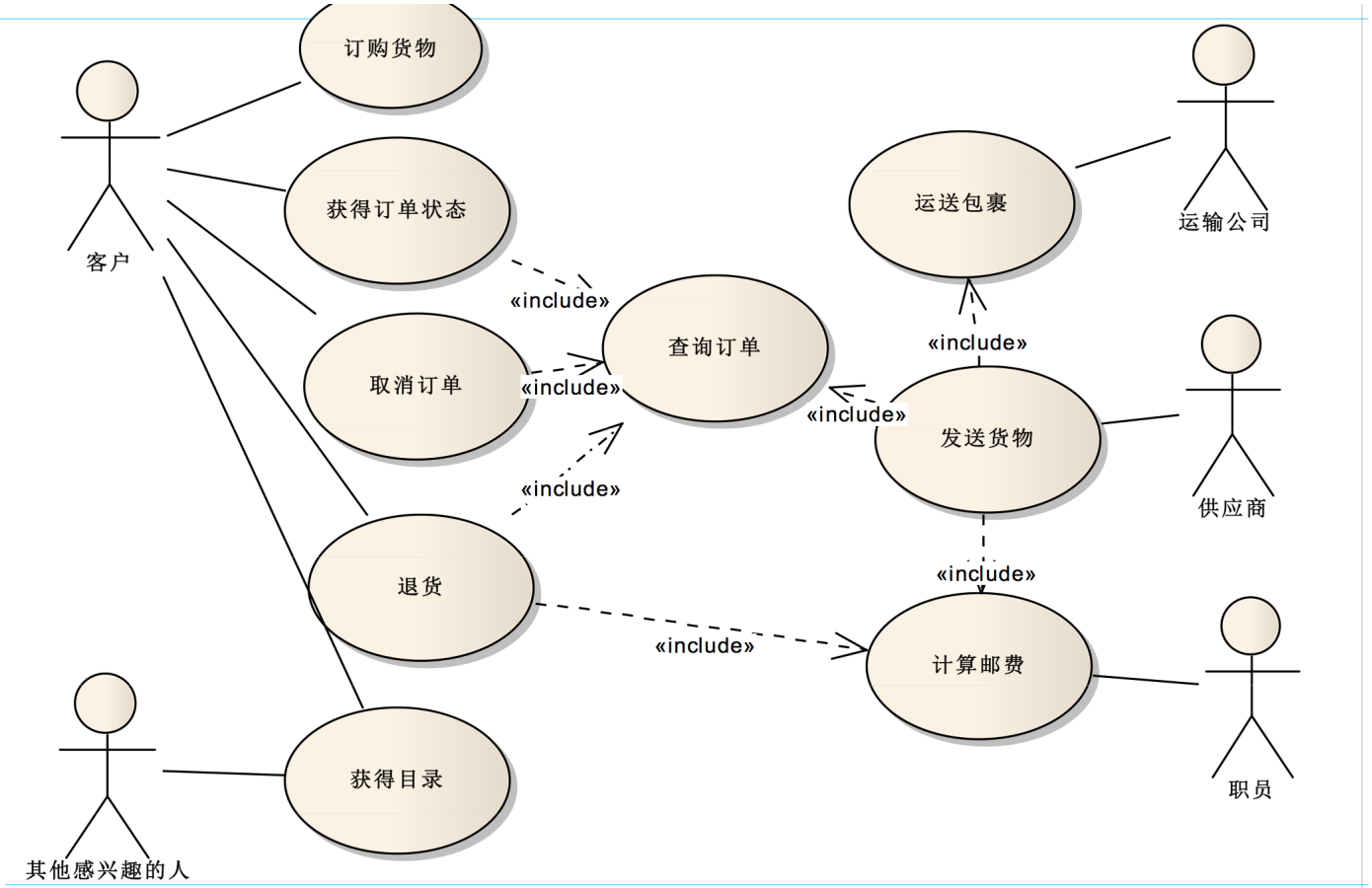
- 沿着基本路径一条一条的找，并且考虑：
  - 在这个点上还可以执行别的活动吗?
  - 在这个点上有没有什么可能出错的？
  - 有什么随时可能发生的行为吗？
- 或者，用以下大类去发现可选路径。例如：
  - 参与者退出应用程序；
  - 参与者取消指定操作；
  - 参与者请求帮助；
  - 参与者提供了“坏数据”；
  - 参与者提供了不完整数据；
  - 参与者选择了一个执行用例的可选方法；
  - 系统崩溃；
  - 系统不可用。

# 可选路径

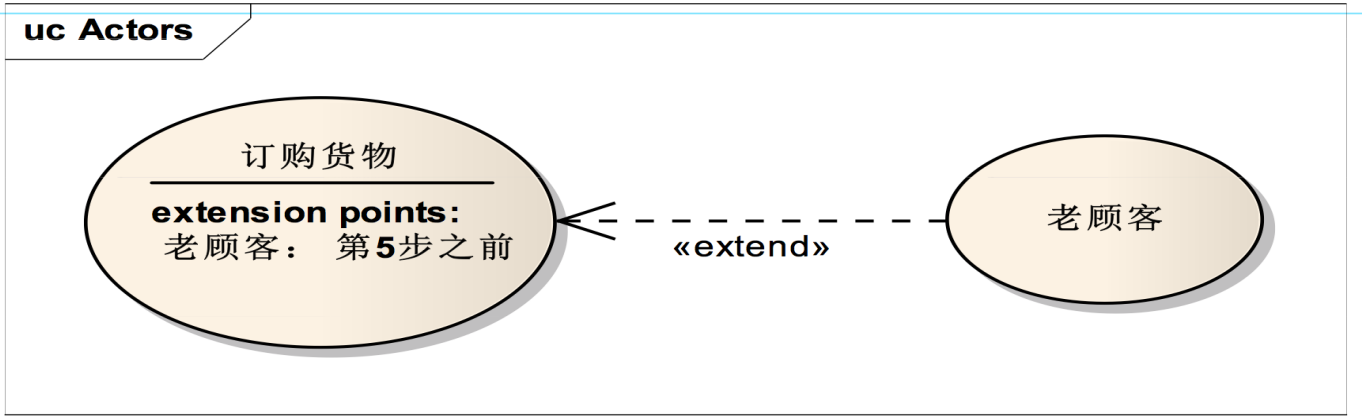
- 1. 不正确的商品代码；
- 2. 不正确的信用卡支付信息；
- 3. 填写订单过程中，用户选择取消；
- 4. 用户选择修改订单信息。

- 要用到搜索订单的用例有：
- 1. 取消订单；
- 2. 查询订单状态；
- 3. 退货。

# 修改后的用例


订购货物
获得订单状态
客户
取消订单
退货
获得目录
其他感兴趣的人
«include»
«include»
«include»
«include»
查询订单
运送包裹
运输公司
«include»
«include»
发送货物
供应商
«include»
计算邮费
职员

# 扩展用例及扩展点




UC012：老顾客打折用例
事件流：
- 顾客选择为老顾客时，用例开始；
- 系统提示客户输入顾客卡号_；
- 顾客输入卡号后，系统验证卡号合法时；
- 系统为顾客计算打折信息。

# 尝试

- 现在急需开发一个“食堂食品溯源管理系统”
  - 如何对用例进行建模?

干系人(Stakeholder)? 主、次参与者(Actor)? 系统？无关？

- 学生
- 教师
- 食堂经理
- 供应商
- 窗口阿姨
- 刷卡机
- 校区管理人员
- 服务维护人员
- 点菜系统
- …

# Reference

- 清华大学国家级精品课程 《软件工程》主讲人 刘强 副教授 刘璘 副教授
- https://www.icourses.cn/sCourse/course_3016.html
- https://www.xuetangx.com/course/THU08091000367/5883555?channel=learn_title

# 谢谢大家！

THANKS