



# 软件工程

# Software Engineering

龙 军

jlong@csu.edu.cn 18673197878

计算机学院 | 大数据研究院

# 需求挑战

---

- **软件项目中40%~60%的问题都是在需求阶段埋下的祸根**
  - 项目干系人说不清需求
  - 需求表述的二义性问题
  - 需求经常变化，项目没有时限
  - 开发人员因为误解需求而不得不大量超时返工
  - 系统测试白费了，因为测试人员并未明白产品要做什么
  - 功能都实现了，但由于产品的低性能、使用不方便或其它因素使用户不满意
  - 维护费用相当高，因为客户的许多增强要求未在需求获取阶段提出

**准确、完整和规范化的软件需求是软件开发成功的关键**

# 程序员的愤怒

---



# 大纲



中南大學  
CENTRAL SOUTH UNIVERSITY

## 第三章 软件需求

☀ 01-软件需求概述

02-需求获取

03-需求分析和建模

04-需求定义和验证

05-需求管理

@第4章.教材

# 什么是软件需求?

---

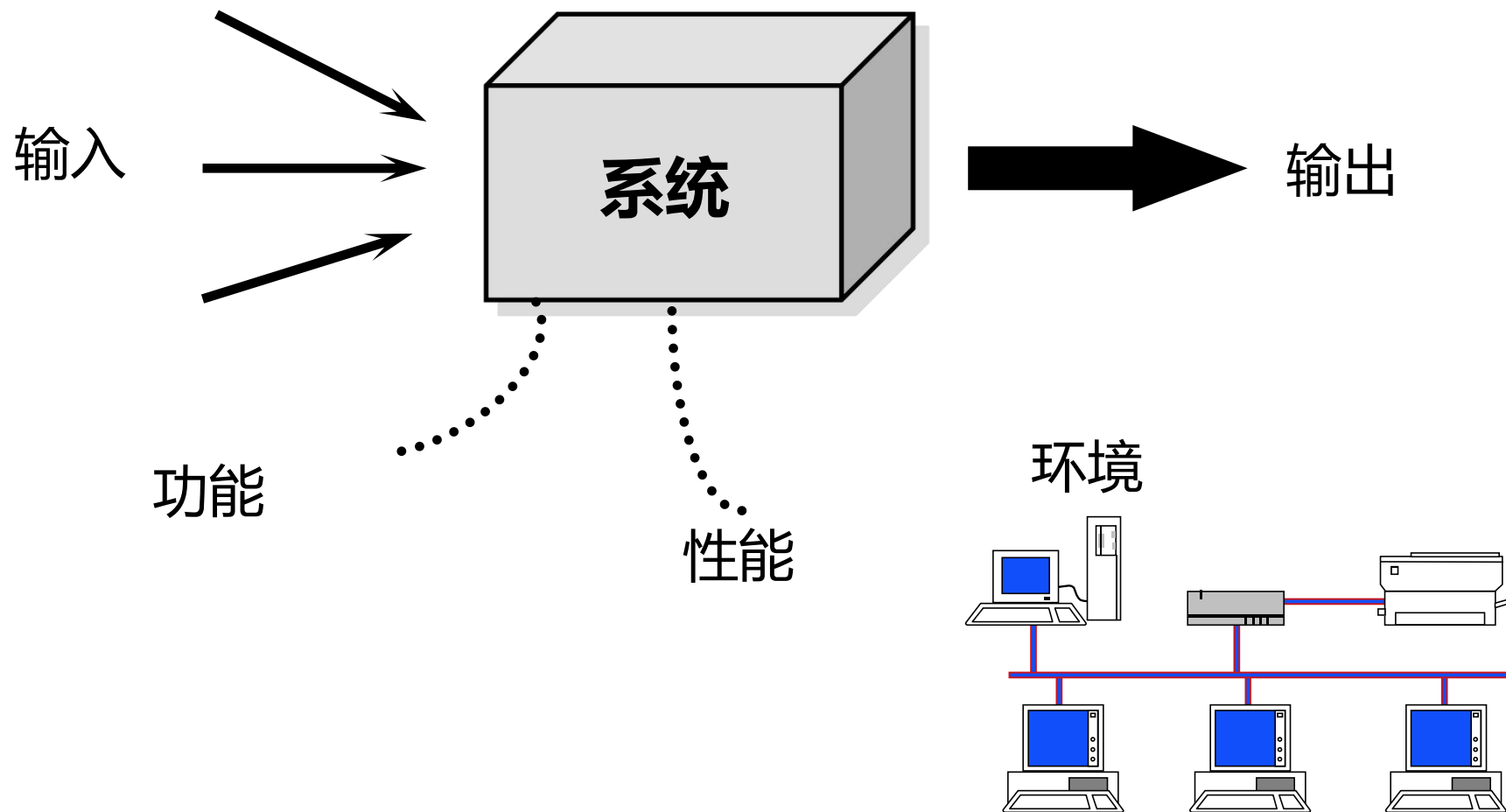
- 在软件工程中,需求分析指的是在建立系统时描写系统的目的、范围、定义和功能时要做的所有工作。
- 需求分析是软件工程中的一个关键过程。在这个过程中,系统分析员和软件工程师要确定顾客的需求。

# 什么是软件需求?

## 软件开发过程中常见的场景



# 什么是软件需求?



# 定义

---

## □ 需求

- 系统必须符合的条件或能力

## □ 软件需求

- 用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。
- 内容包括：FURPS + 模型

关注What!



# FURPS

---

- **功能性 (Functionality)** : 涉及系统必须执行的功能和操作, 包括特性、功能、安全性等。
- **易用性 (Usability)** : 关注用户如何与系统交互, 包括人性化因素、帮助、文档等。
- **可靠性 (Reliability)** : 涉及系统的稳定性和可预测性, 包括故障频率、可恢复性、可预测性等。
- **性能 (Performance)** : 包括系统的速度、效率、资源消耗、吞吐量和响应时间等。
- **可支持性 (Supportability)** : 涉及系统的可维护性、可测试性、可扩展性、适应性、兼容性等。

# FURPS +

---

- ❑ 设计约束 (design constraints) : 规定或约束了系统的设计的需求
- ❑ 实现需求 (implementation requirements) : 规定或约束了系统的编码或构建, 如技术标准、编程语言、数据库完整性策略、资源限制和操作环境
- ❑ 接口需求 (interface requirements) : 规定了系统必须与之交互操作的外部软件或硬件, 以及对这种交互操作所使用的格式、时间或其他因素的约束
- ❑ 物理需求 (physical requirements) : 规定了系统必须具备的物理特征, 用来代表硬件要求, 如物理网络配置需求
- ❑ 操作需求 (Operations requirements) 社会、健康、安全、法律及文化等。

# 设计约束

---

- 一项需求允许多种设计方案
  - 设计是在这多种方案中做出选择
- 没有选择的需求就是一个设计约束
  - 它和其它需求不同
  - 将它放在软件需求的单独一节中
  - 将每个设计约束的源标识出来
  - 记录每个设计约束的原理
- 举例
  - 必须要有某一种算法
  - 必须要用数据库

# 社会、健康、安全、法律及文化对软件的约束

---

## □ 社会

- 如软件不能危害社会的可持续发展，有责任的AI系统

## □ 健康

- 如未成年使用设限，支持健康限制的用户

## □ 安全 (Safety)

- 如自动驾驶汽车的安全性要求

## □ 法律法规

- 如个人隐私保护、不能涉黑涉黄、遵循行业规范、不违反知识产权

## □ 文化

- 如不同民族和国家的文化差异对社交软件提出不同的要求

# 软件对社会、健康、安全、法律及文化的影响

一个合理的软件系统可以对社会、健康、安全、法律和文化产生积极的影响

## □ 社会

- 如手机软件允许人们快速地共享信息，有助于提高工作效率和生活质量。

## □ 健康

- 如医疗保健系统提高治疗结果和降低感染风险，胶囊机器人进行人体探测

## □ 安全

- 如智能交通系统提高道路安全和减少交通事故

## □ 法律

- 如基于法律大模型的法律问答和案例分析系统，提高法务工作效率

## □ 文化

- 如数字孪生，保护文化遗产和传统；图书管理系统共享图书资源

# 讨论：需求

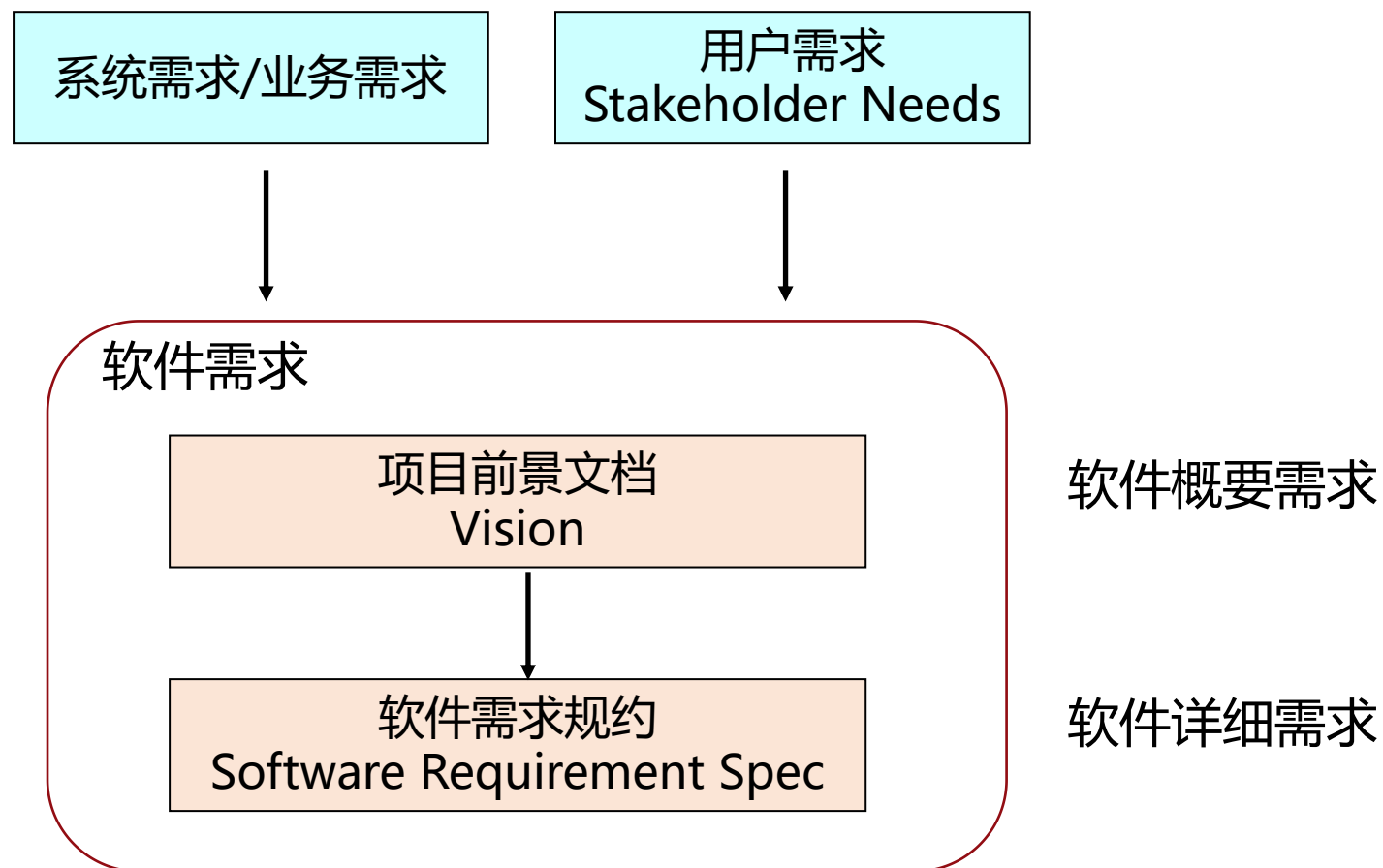
---

软件需求分为功能需求、易用性需求、可靠性需求、性能需求、支持性需求、和设计约束，请问以下需求各属于哪类需求？

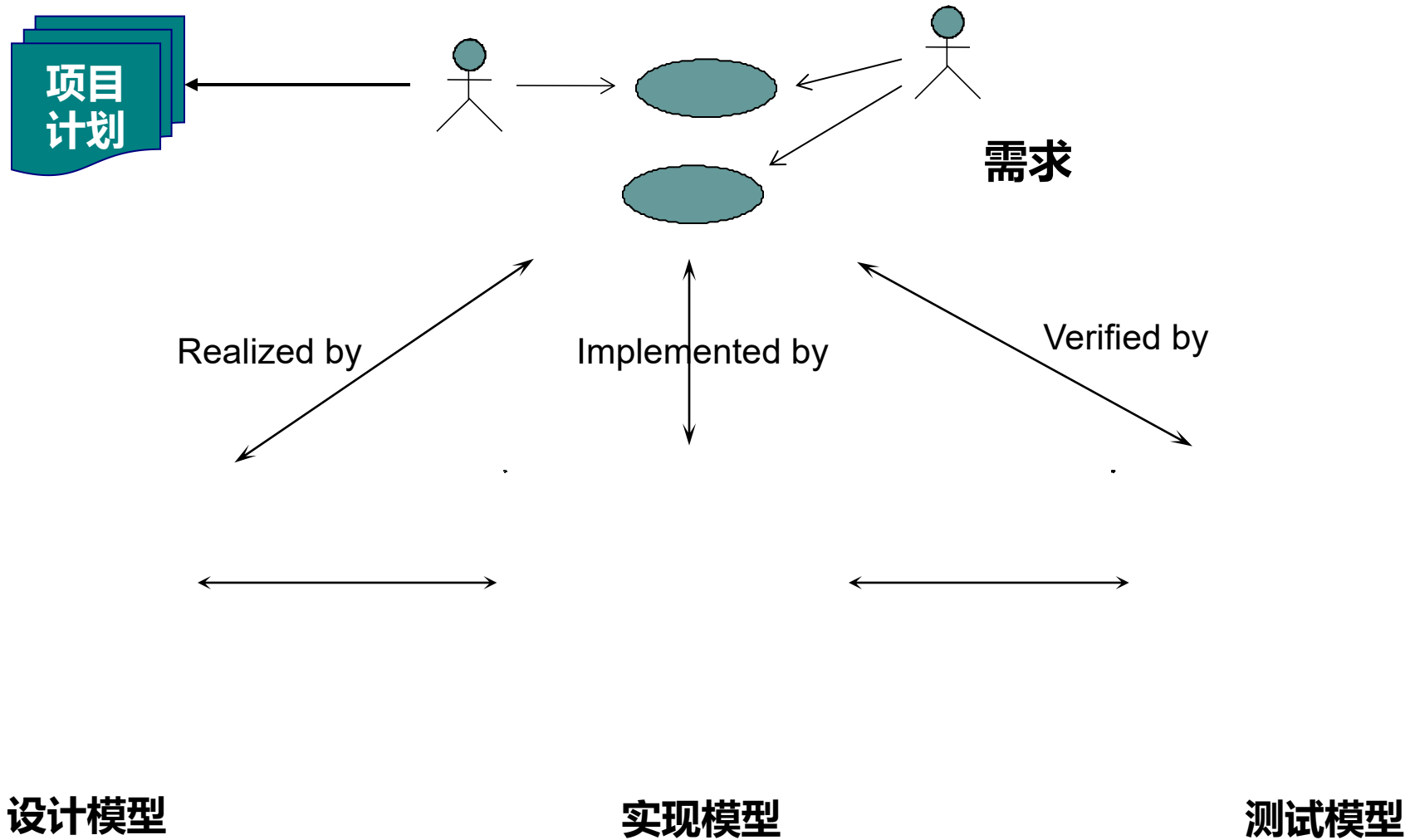
- ❑ 系统支持百万级的用户并发访问
- ❑ 系统每月宕机不超过1次
- ❑ 系统每天用户不可用的时间不超过1分钟
- ❑ 系统不易被黑客入侵
- ❑ 系统可以方便增加各种增值服务
- ❑ 系统应在2024年5月1日上线

# 软件需求的层次

软件需求可以分为四个层次：用户需求、系统需求、软件需求和设计需求。



# 需求驱动开发

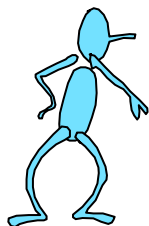




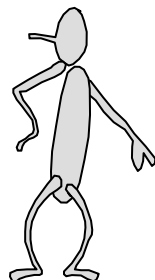
# 谁看需求?

---

客户



Client

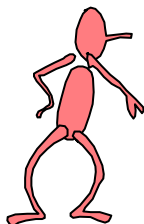


Users

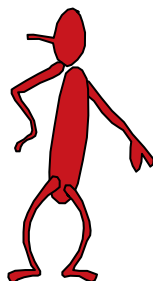
开发小组



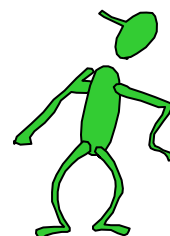
Tester



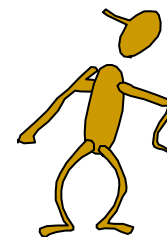
Designer



Requirements  
Specifier



Technical  
Writer



Project  
Manager

# 优秀需求具有的特性

---

## □ 单个优秀需求应具有的特性：

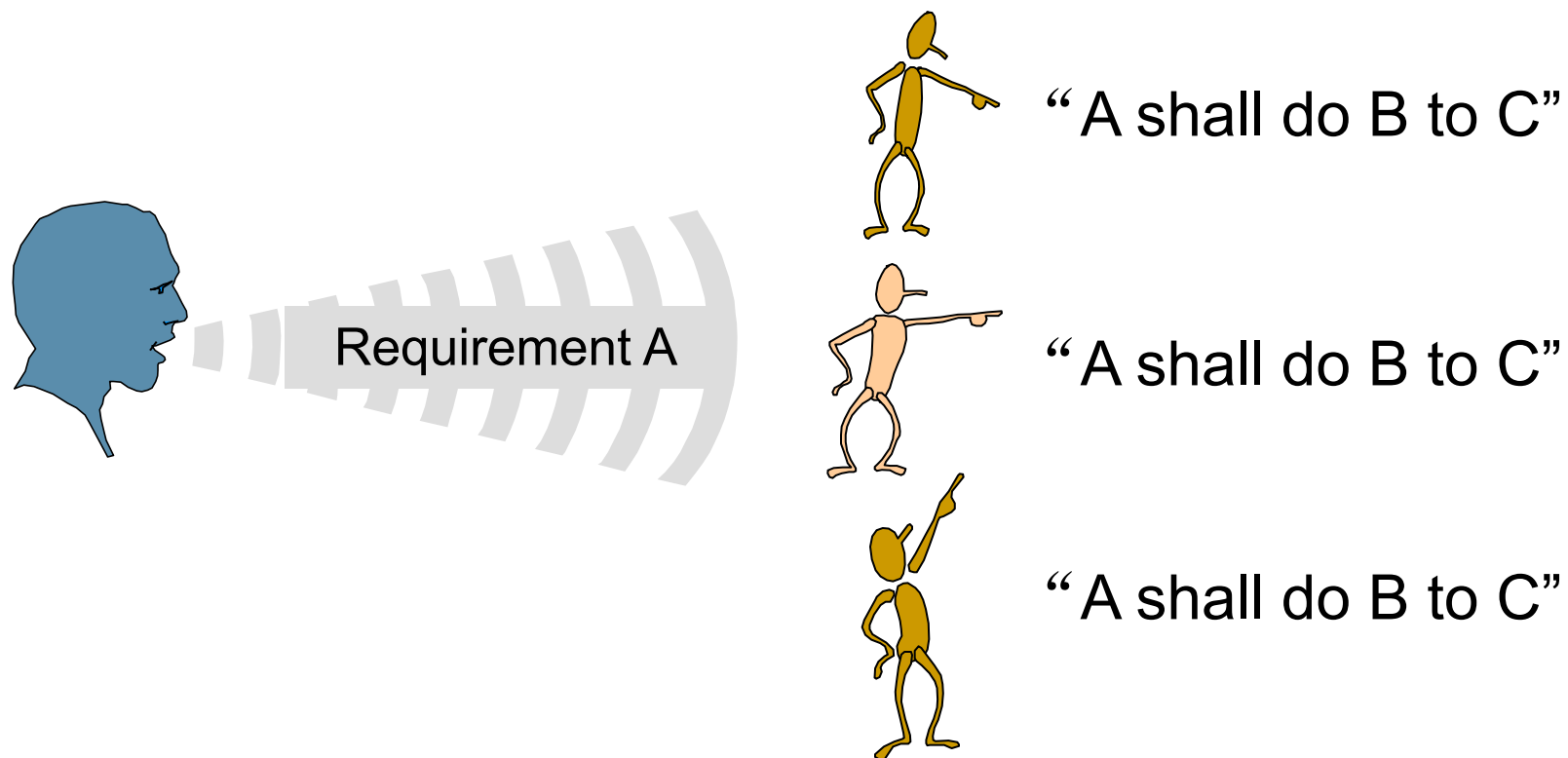
- 完整性
- 正确性
- 可行性
- 必要性
- 划分优先级 (Why?)
- 无二义性
- 可验证性

## □ 多个优秀需求应具有的特性：

- 完整性
- 可理解性/可修改性
- 一致性
- 可跟踪性

# 无二义性

- 需求是无二义的，如果：
  - 它只有一个解释



# 二义性练习

---

我没说她偷了我的钱

*Mary had a little lamb*

如何减少需求的二义性?

# 可验证性

---

## □ 需求可验证的，如果

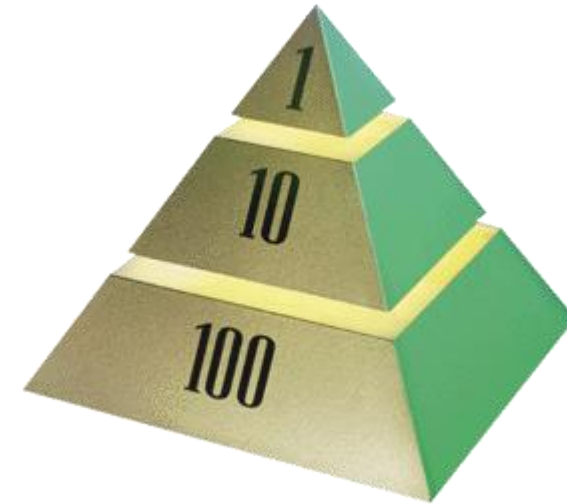
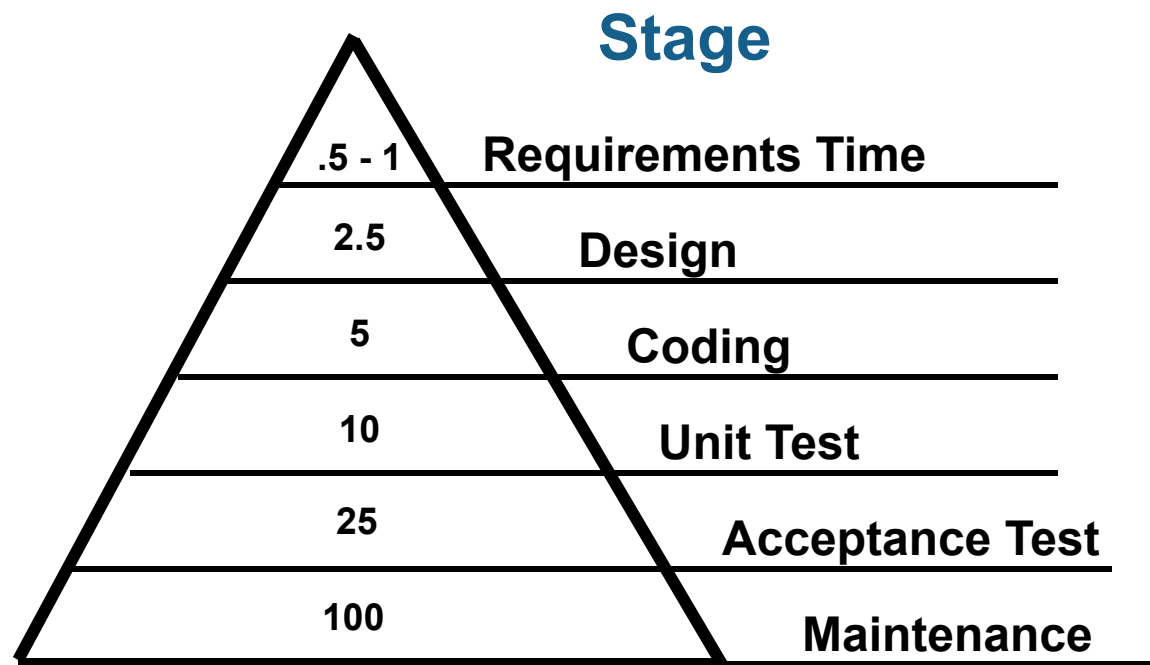
### ■ 可以用人工或机器方式检查产品是否满足需求

- The system supports up to 1,000 simultaneous users
- The system shall respond to an arbitrary query in 500 msec.
- The color shall be a pleasing shade of green
- The system shall be available 24 x 7
- The system shall export view data in comma-separated format

**这些需求可验证吗？  
如果不，如何更好地表达？**

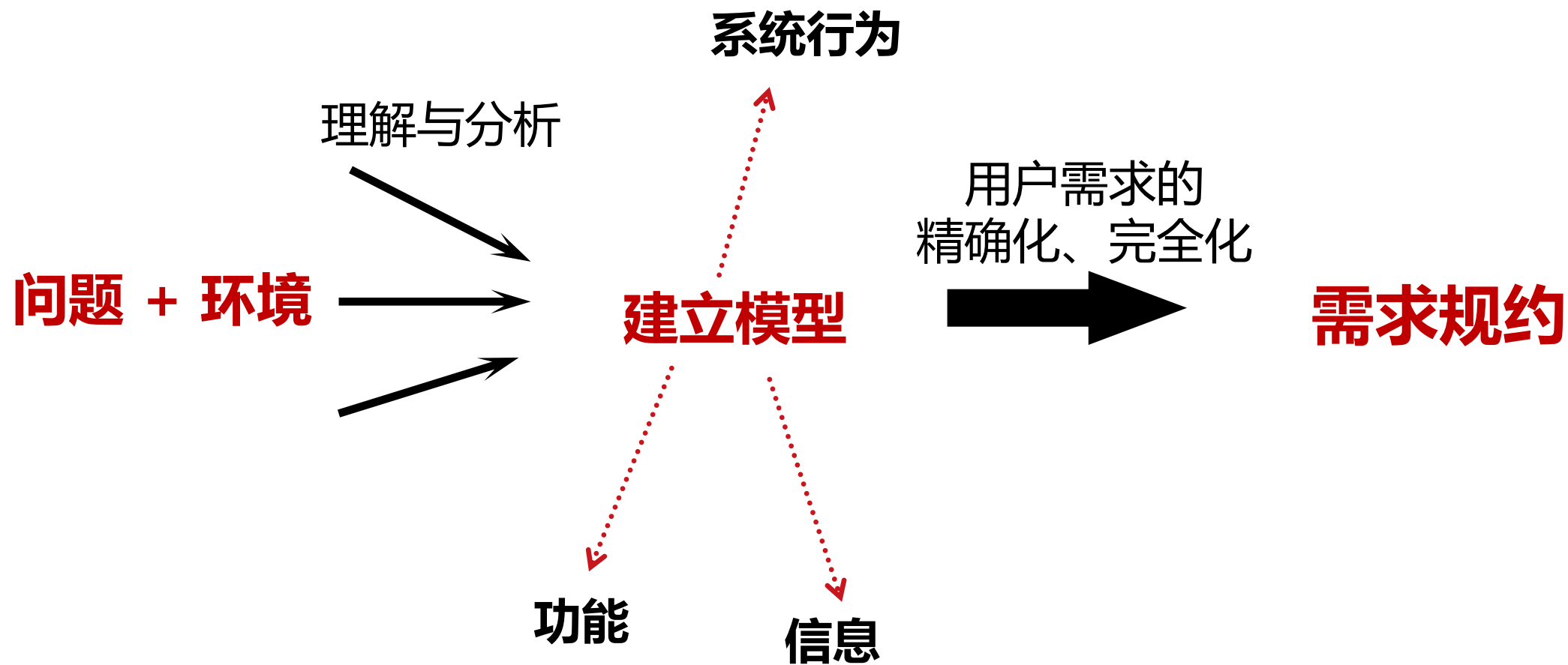
# 需求出错的高成本

## *The 1-10-100 Rule*

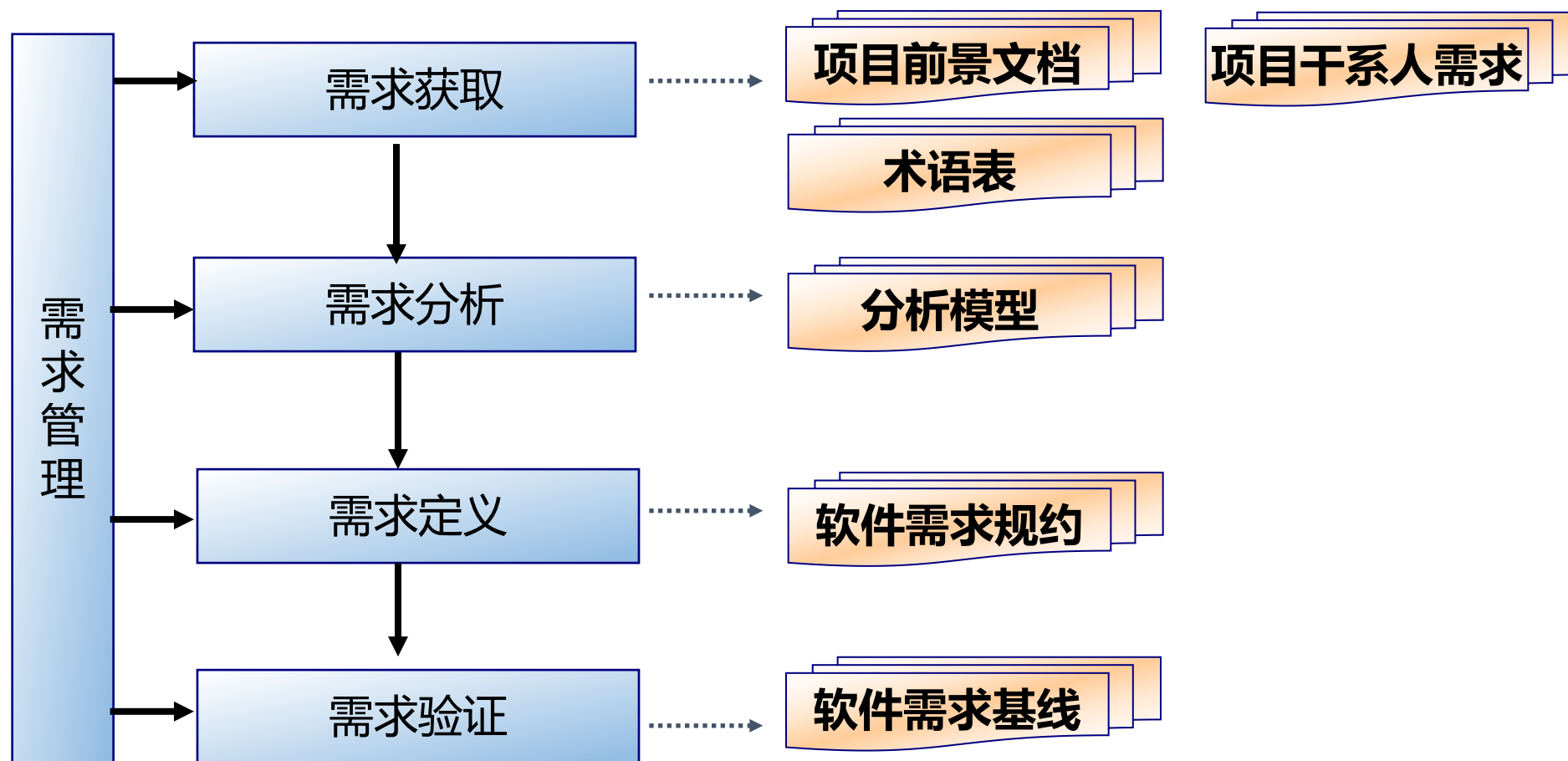


*"All together, the results show as much as a 200:1 cost ratio between finding errors in the requirements and maintenance stages of the software lifecycle."*

# 软件需求工程



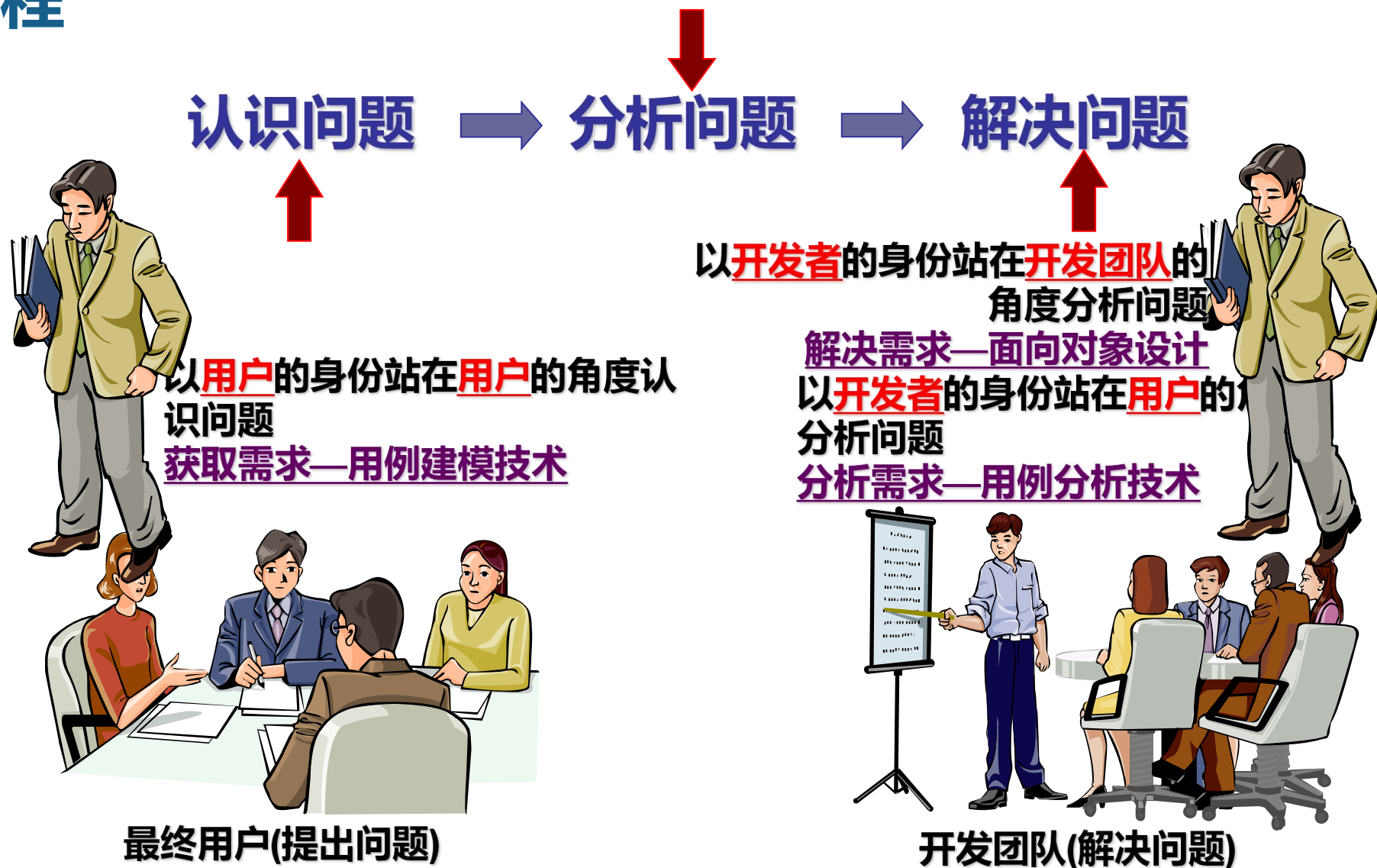
# 软件需求工程



发现、获取、组织、分析、编写和管理需求的系统方法，让客户和项目组之间达成共识。



# 软件需求工程



# 大纲



中南大學  
CENTRAL SOUTH UNIVERSITY

## 第三章 软件需求

01-软件需求概述

☀ 02-需求获取

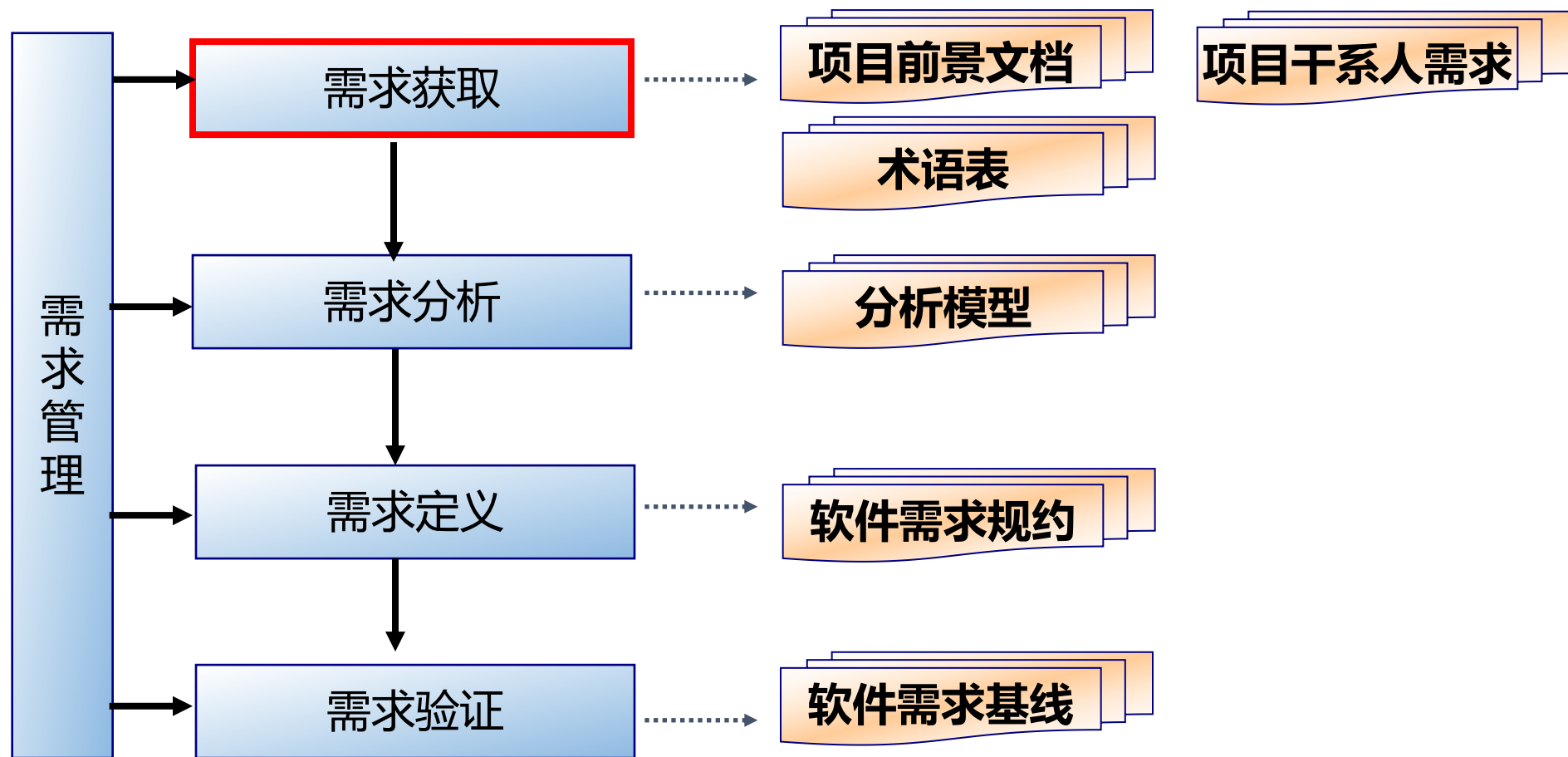
03-需求分析和建模

04-需求定义和验证

05-需求管理

@第4章.教材

# 需求工程



发现、获取、组织、分析、编写和管理需求的系统方法，让客户和项目组之间达成共识。

# 前景文档 (Vision)

---

1. 简介 Introduction
2. 定位 Positioning
  - 2.1 商机 2.2 问题说明 2.3 产品定位
3. 项目干系人和用户描述 Stakeholder and User Descriptions
4. 产品概述 Product Overview
5. 产品特性 Product Features
6. 约束 Constraints
7. 质量范围 Quality Ranges
8. 优先级 Precedence and Priority
9. 其它产品需求 Other Product Requirements
10. 文档需求 Documentation Requirements

# 1) 分析问题及根源

- 什么是问题?
  - 理解和记录客户的观点
  - 达成共识
- 什么是真正的问题?
  - 寻找问题根源，探究症结
- 避免Yes...But现象，避免IT黑洞

从业务角度



理解的

期望的

# 识别业务解决方案

---

- 分析业务需求
- 提出多种解决方案
  - 技术的或(和)非技术的
- 选择最能满足业务需求的解决方案
- 启动项目,实现方案

# 案例研究: Course Registration System

---

- Review the problem statement



Course Registration  
Requirements Document

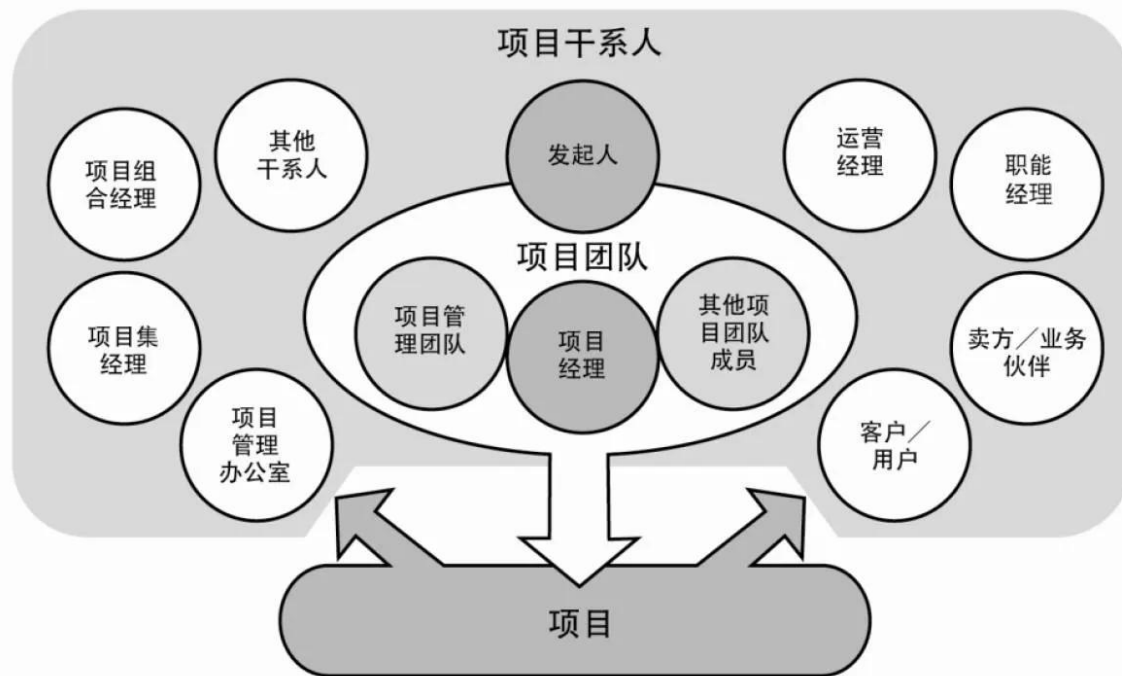
# 举例：选课系统的问题陈述 @Vision文档

<b>The problem of</b>	<b>The outdated and largely manual student registration process at Wylie College</b>
<b>affects</b>	<b>Students, professors, and College administration.</b>
<b>The impact of which is</b>	<b>A slow and costly process combined with dissatisfied students and professors.</b>
<b>A successful solution would</b>	<b>Improve the image of the College, attract more students, and streamline administrative registration functions.</b>



## 2) 识别项目干系人

- 项目干系人（Stakeholder），又称为项目涉众或利益相关人，是积极参与项目，或其利益因项目的实施或完成而受到积极或消极影响的个人和组织，他们还会对项目的目标和结果施加影响。



# 举例：选课系统的Stakeholder @Vision文档

Name	Represents	Role
IT Executive	IT Department and Wylie College as whole.	Responsible for project funding approval. Monitors project progress.
Registrar	The office of the registrar, administrative and data entry personnel.	Ensures that the system will meet the needs of the registrar, who has to manage the course registration data, including professor and student databases.
Student	Students	Ensures that the system will meet the needs of students.
Professor	Professors	Represents the interests of the faculty (professors).

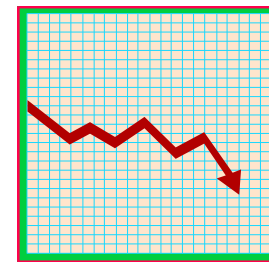
### 3) 识别项目的约束



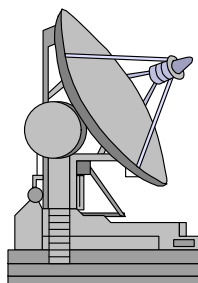
环境



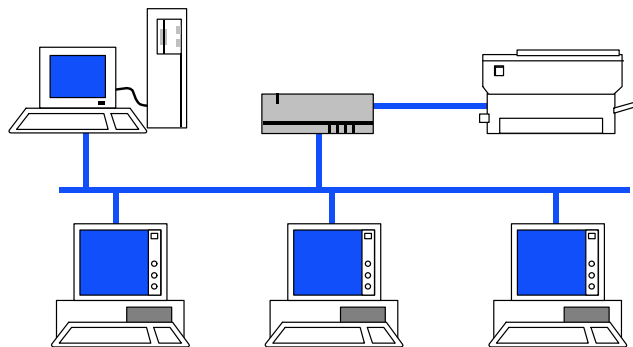
政治



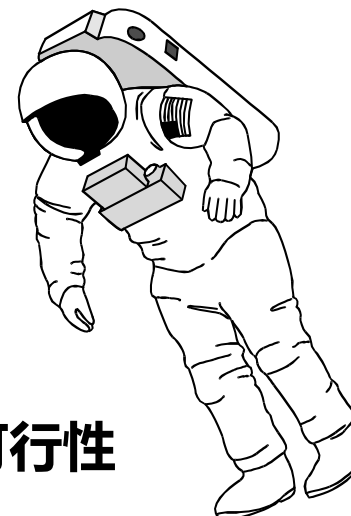
经济



技术



系统



可行性

# 举例：选课系统的约束条件 @Vision文档

---

## ❑ Assumptions and Dependencies

- The existing Billing and Course Catalog Database Systems which reside on the College DEC VAX Mainframe will continue to be supported until at least 2005.
- The external interfaces of the Billing and Course Catalog Database Systems are as defined in [2] and [3] and will not be altered.
- It is assumed that the College will continue to operate and support the existing UNIX Server and the DEC VAX Mainframe until at least 2005.
- It is assumed that additional funding will be available by 2005 to replace the legacy Billing and Course Catalog Database Systems.
- Implementation of the new registration system in time for the January 2000 school term is dependent upon funding approval by March 1st, 1999.

## ❑ Constraints

- The system shall not require any hardware development or procurement.
- The course information available is limited to the type of data supported by the existing Course Catalog Database.

## 4) 获取常用术语

---

- 定义项目用到的术语
- 有助于避免误解
- 记录于单独的术语表文件中

术语表  
Glossary

获取常用术语

- 尽早开始
- 在项目过程中持续进行

# 举例：选课系统的术语表 @Glossary文档

## □ Course

- A class offered by the university.

## □ Course Offering

- A specific offering for a course, including days of the week and times.

## □ Course Catalog

- Unabridged catalog of all courses offered by the university.

## □ Grade

- The grade for the student in a course.

## □ Report Card

- All the grades for all courses taken by a student in a given semester.

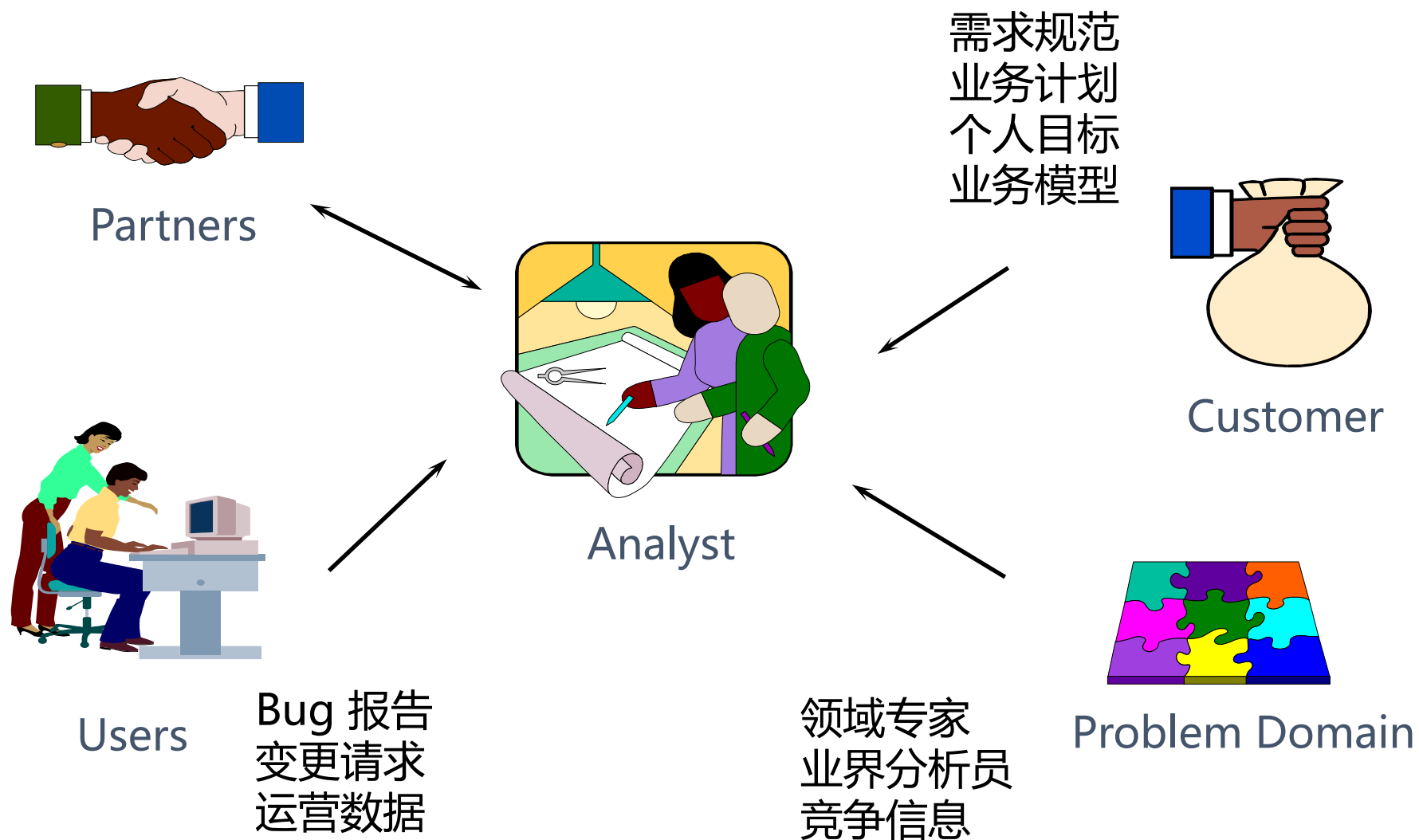
## □ Roster

- All the students enrolled in a particular course offering.

## □ Transcript

- The history of the grades for all courses for a particular student.

## 5) 识别需求的来源



## 6) 收集需求

- 访谈
- 调查问卷
- 需求研讨会
- Use-Case 研讨会
- Storyboards
- 角色扮演
- 复审现有需求
- 观察正在工作的用户
- ... ..





## 7) 产品定位

举例：选课系统的产品定位 @Vision文档

For	Wylie College students, professors, and the course registrar
Who	Attend, teach, or administer college courses
The Course Registration System	Is a tool
That	Enables online course registration and access to course and grade information
Unlike	The existing outdated mainframe registration system
Our product	Provides up-to-date information on all courses, registrations, teachers, and grades to all users from any PC connected via the College LAN or internet.

## 8) 撰写产品特性

---

举例：选课系统的产品特性 @Vision文档

- ☐ Logon
- ☐ Register for Courses
- ☐ Course Cancellations
- ☐ Student Billings
- ☐ Enter, Update, and View Professor Information
- ☐ View Student Grades
- ☐ Select Courses to Teach
- ☐ Enter, Update, and View Student Information
- ☐ Record Student Grades
- ☐ View Course Catalog Information
- ☐ View Course Schedule
- ☐ Monitor for Course Full

## 9) 定义质量范围

---

### 举例：选课系统的质量范围@Vision文档

- Availability:
  - The System shall be available 24 hours a day, 7 days a week.
- Usability:
  - The System shall be easy-to-use and shall be appropriate for the target market of computer-literate students and professors.
  - The System shall include online help for the user. Student and Professor users should not require the use of a hardcopy Manual to use the System.
- Maintainability:
  - The System shall be designed for ease of maintenance. All college-specific data should be table-driven and modifiable without recompilation of the System.
- Performance Requirements
  - The system shall support up to 2000 simultaneous users against the central database at any given time, and up to 500 simultaneous users against the local servers at any one time.
  - The system shall provide access to the legacy Course Catalog Database with no more than a 10 second latency.
  - The system shall complete 80% of all transactions within 2 minutes.

## 10) 定义文档需求

---

举例：选课系统的文档需求 @Vision文档

- ☐ User Manual
- ☐ On-line Help
- ☐ Installation Guides, Configuration, Read Me File
- ☐ Labeling and Packaging

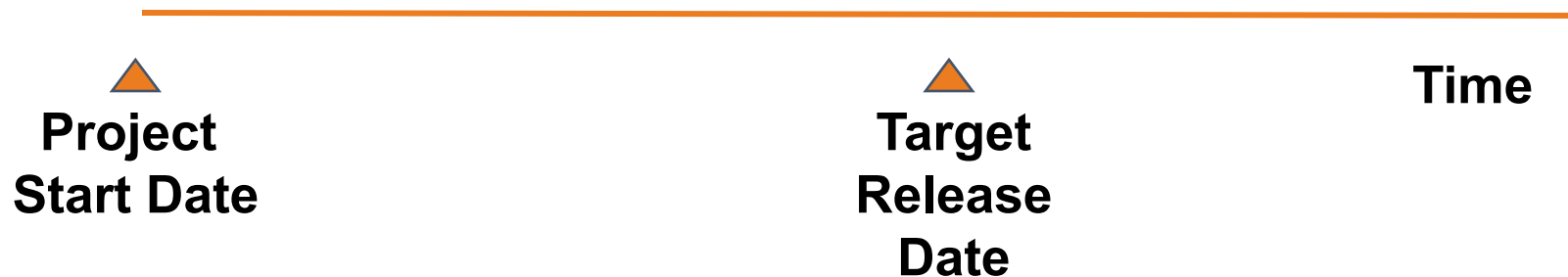
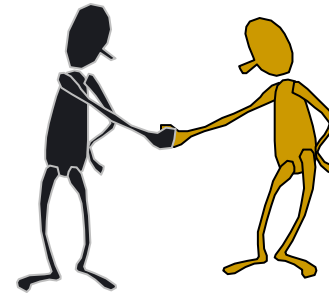
# 11) 建立项目范围

- Feature 1: The system must...
- Feature 2: The system must...
- Feature 3: The system must...
- Feature 4: The system must...
- Feature n: The system must...

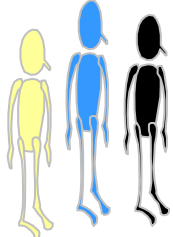

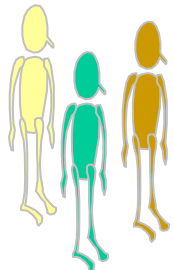
How do we know what the needs are?

How do we determine priority?

Where do we set the baseline?



# 使用需求属性排列特性的优先级

	状态	风险	重要性	工作量	成本
✓ Feature Reqt. 10	Approved	Low	High		\$\$\$
Feature Reqt. 13	Proposed	Med.	Low		\$\$
✓ Feature Reqt. 40	Approved	High	Mandatory		\$

其他属性包括稳定性、技术难度等

## 12) 划分特性优先级

---

举例：选课系统的特性优先级 @Vision文档

- Release 1 :
  - Logon
  - Register for Courses
  - Interface to Course Catalog Database
  - Maintain Student Information
  - Maintain Professor Information
- Release 2 :
  - Submit Student Grades
  - View Grades
  - Select Courses to Teach

# 大纲



中南大學  
CENTRAL SOUTH UNIVERSITY

## 第三章 软件需求

01-软件需求概述

02-需求获取

☀ 03-需求分析和建模

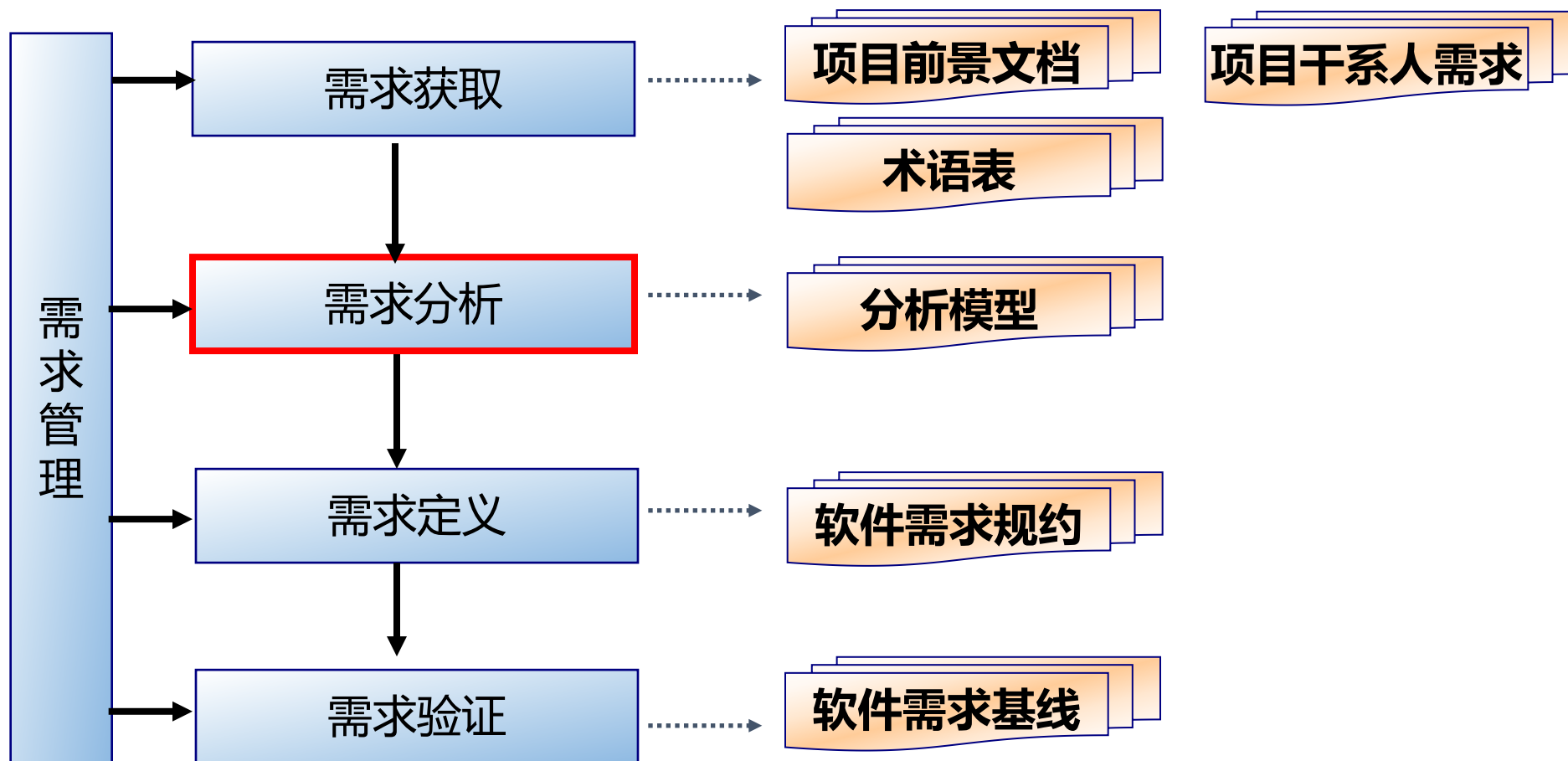
04-需求定义和验证

05-需求管理

@第4章.教材



# 需求工程



发现、获取、组织、分析、编写和管理需求的系统方法，让客户和项目组之间达成共识。

# 分析建模准则

---

- ❑ Represent the information domain
- ❑ Represent software functions
- ❑ Represent software behavior
- ❑ Partition these representations
- ❑ Move from essence toward implementation

# 分析模型

---

□ 对需求进行分析，并进行图形建模，形成分析模型(平台无关模型PIM)

## ◆ 面向对象分析模型

- 用例图
- 活动图
- 类图
- 时序图
- 通信图
- 状态机图

# 面向对象分析的步骤

## 用例驱动的软件需求分析方法



### 1. 建立用例模型

#### 1.1 识别actor和use case, 画Use-Case图

1.2 编写Use-Case Spec.

1.3 优化Use-Case图的结构



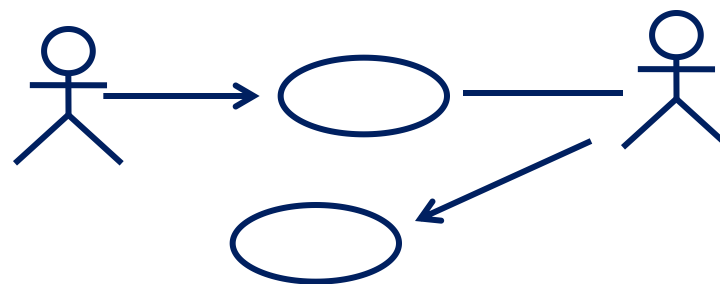
### 2. 建立概念模型



### 3. 建立分析模型

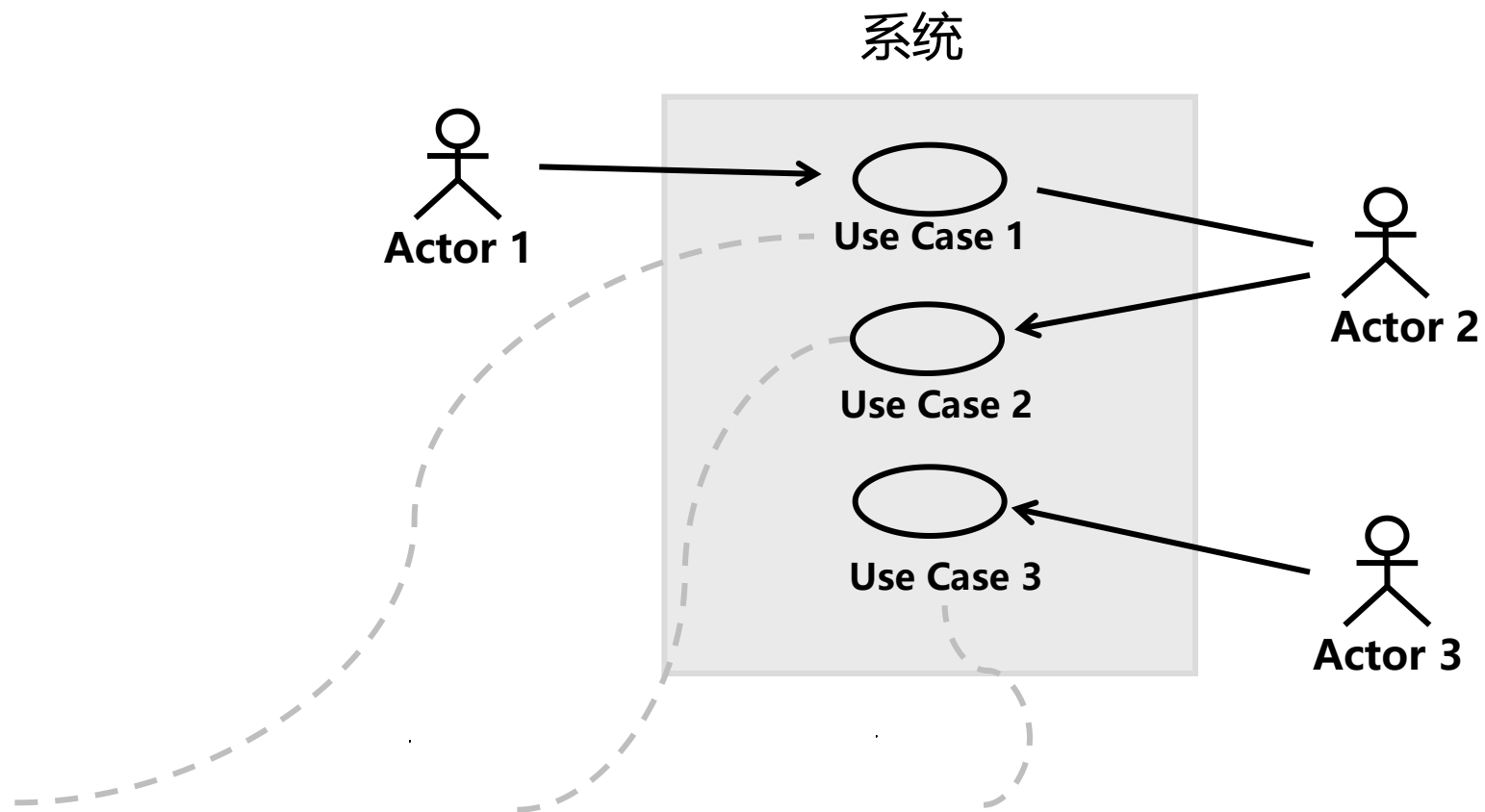
# Use Case技术

- 提供干系人的观点
- 定义功能需求
- 促进理解和讨论
  - 为什么需要系统?
  - 谁和系统交互 (actors)?
  - 用户希望如何使用系统 (use cases)?
  - 系统应该有什么接口?



Use-Case 模型

# Use-Case 模型的组成



## Use-Case 1 规约

- 简要描述
- 事件流

## Use-Case 2 规约

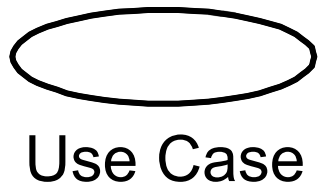
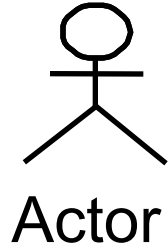
- 简要描述
- 事件流

## Use-Case 3 规约

- 简要描述
- 事件流

# Actor 和 Use Case

---



## *Actor*

和系统交互的系统外的某些人或某些东西：

- 最终用户
- 外界软件系统
- 外界硬件设备

## *Use case*

Actor想使用系统去做的事

# 如何识别Actor

---

- 谁需要在系统的帮助下完成自己的任务？
- 需要谁去执行系统的核心功能？
- 需要谁去完成系统的管理和维护？
- 系统是否需要和外界的硬件或软件系统进行交互？



# 如何识别Use Case

---

- 每个actor的目标和需求是什么？
  - actor希望系统提供什么功能？
  - actor 将创建、存取、修改和删除数据吗？
  - actor是否要告诉系统外界的事件？
  - actor 需要被告知系统中的发生事件吗？

# 避免功能分解

## 现象

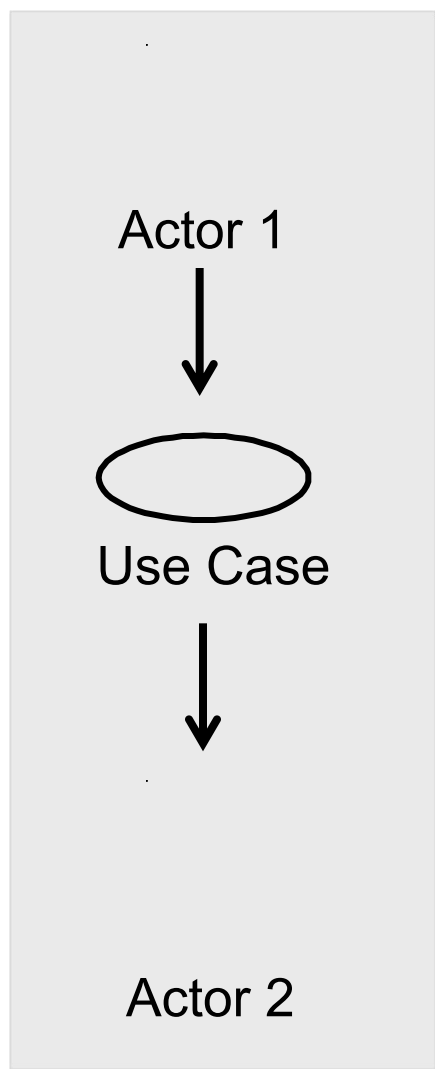
- use case太小
- use cases太多
- use case没有价值回报
- 命名以低层次的操作
  - “Operation” + “object”
  - “Function” + “data”
  - Example: “Insert Card”
- 很难理解整个模型

## 纠正措施

- 寻找更大的上下文
  - “为什么你要构建本系统?”
- 让自己站在用户的角度
  - “用户想得到什么”
  - “这个 use case 能满足谁的要求?”
  - “这个use case 能增值什么?”
  - “这个use case 背后有什么故事?”

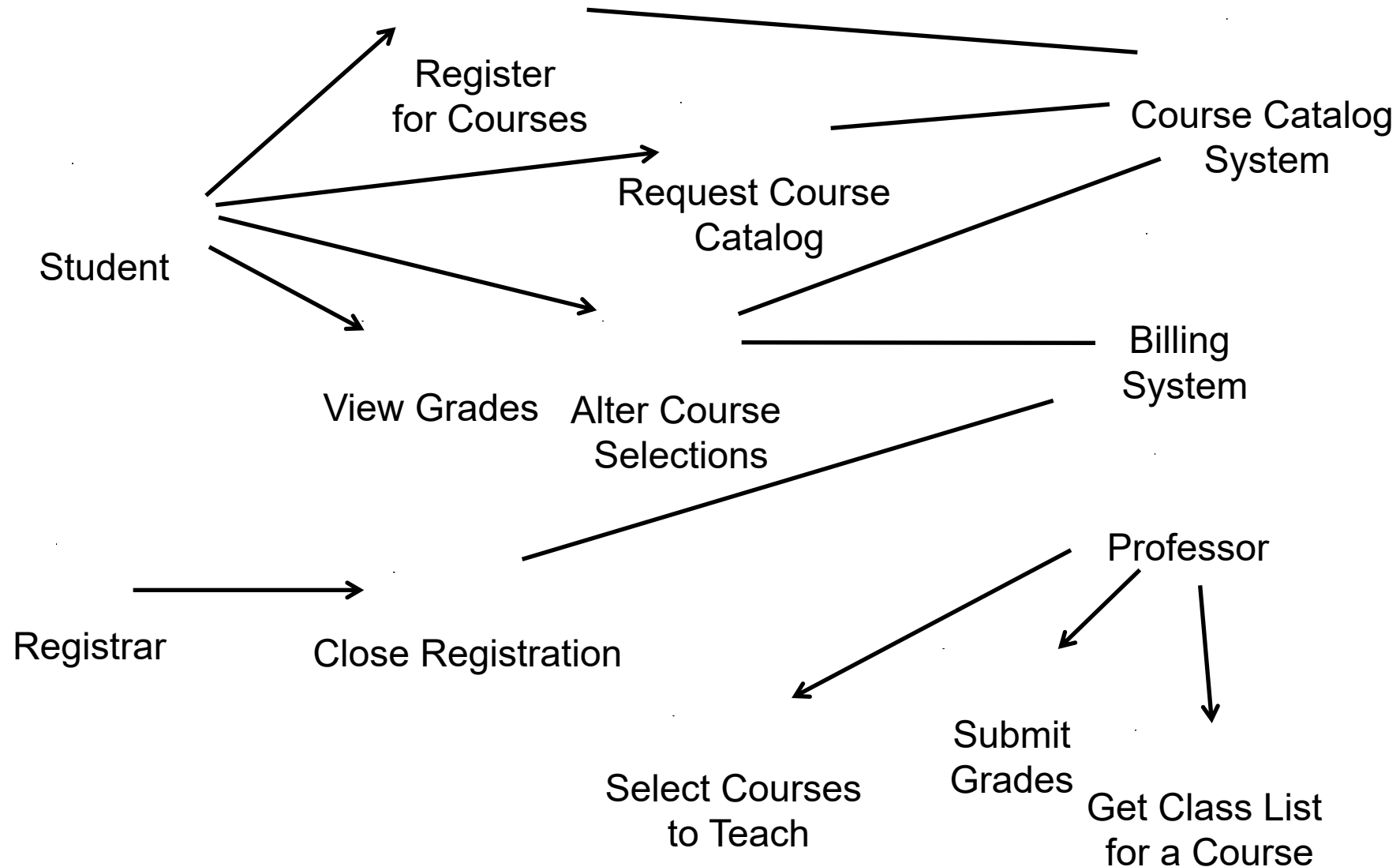
一个用例定义了和actor之间的一次完整对话。

# 通信-关联 Communicates-Association



- Actor和use case 间的通信渠道
- 用一条线表示
- 箭头表示谁启动通信

# Course Registration System的用例图



# 面向对象分析的步骤



## 1. 建立用例模型

1.1 识别actor和use case, 画Use-Case图

**1.2 编写Use-Case Spec.**

1.3 优化Use-Case图的结构



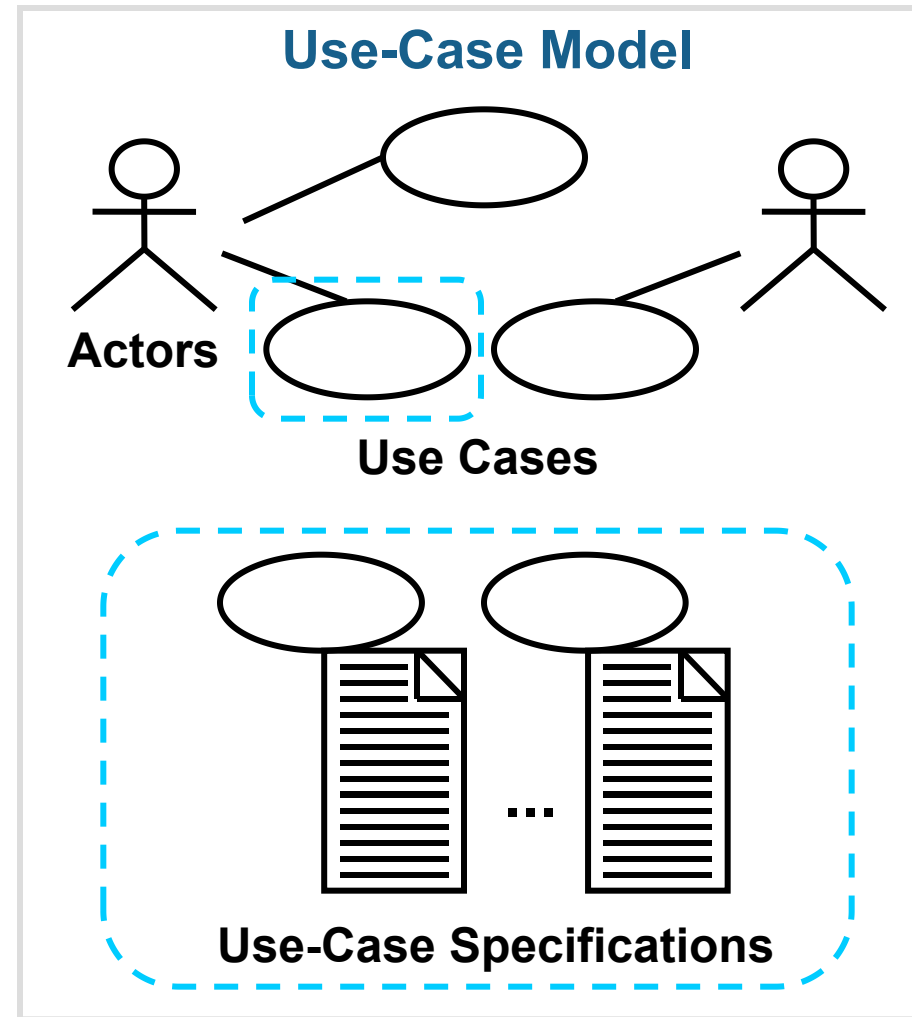
## 2. 建立概念模型



## 3. 用例分析

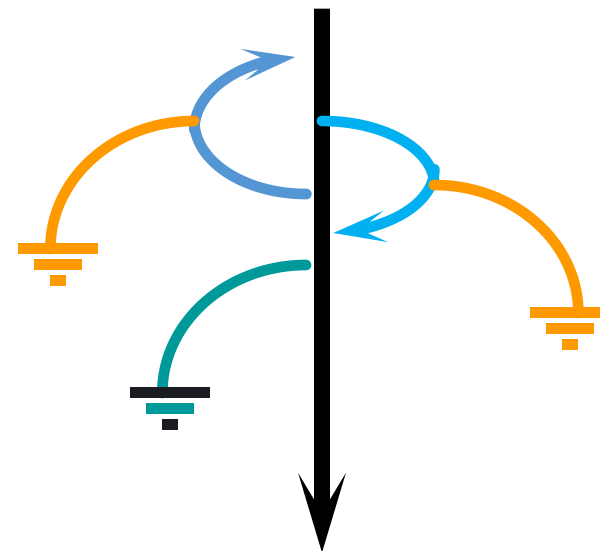
# Use-Case Specifications

- ❑ Name
- ❑ Brief description
- ❑ Flow of Events
- ❑ Relationships
- ❑ Activity diagrams
- ❑ Use-Case diagrams
- ❑ Special requirements
- ❑ Pre-conditions
- ❑ Post-conditions
- ❑ Other diagrams



# 事件流（基本流和备选流）

- 一个基本流
  - Happy day scenario
  - Successful scenario from start to finish
- 多个备选流
  - Regular variant
  - Odd cases
  - Exceptional (error) flows



# 事件流举例: Get Quote

---

## 基本流 Basic Flow

1. Customer logs on
2. Customer selects 'Get Quote' function
3. Customer selects stock trading symbol
4. Get desired quote from Quote System
5. Display quote
6. Customer gets other quotes
7. Customer logs off

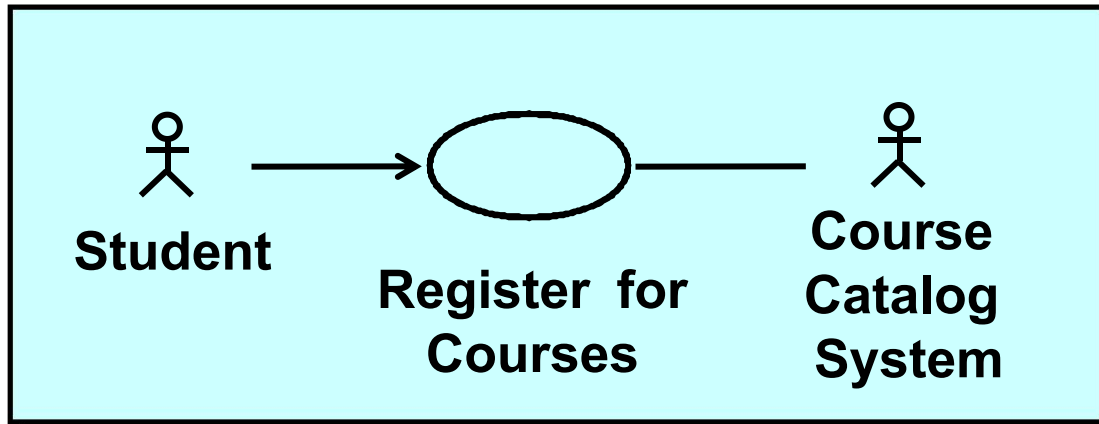
## 备选流 Alternative Flows

- A1. Unidentified Trading Customer
- A2. Quote System Unavailable
- A3. Quit

What are  
other  
alternatives?



# Scenario 是Use-Case 的实例



## 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

# 细化基本事件流

## Get Quote

### Basic Flow

#### 1. Customer Logs On

The use case starts when the Trading Customer logs on. The system validates the customer id and password. ...

#### 2. Customer Selects “Get Quote” Function

The Trading Customer chooses to “Get Quote.” The system displays the list of securities on which it has quotes.

#### 3. Customer Gets Quote

The Trading Customer selects from the list of securities or enters the trading symbol for a security. The system sends the trading symbol to the Quote System, and receives the Quote System Response.

#### 4. Customer Gets Other Quotes

If the Trading Customer wishes to get additional quotes, the use case resumes at Step 3.

#### 5. Customer Logs Off

The Trading Customer logs off the system. The use case ends.

将流分成几步

每步标识编号  
和名字

描述每步  
(1-3 句)

每步是一个来  
回的事件

# 细化通用备选流

## Identify Customer

### *Alternative Flows*

#### **A1. Unidentified Trading Customer**

In Step 1 of the Basic Flow, if the system determines that the customer password is not valid, the system displays a “Sorry, not a valid customer....” message. The use case ends.

#### **A2. Wrong Password**

In Step 1 of the Basic Flow, if the system determines that the customer password id is not valid, the system displays a “Sorry, not....”

#### **A3. Suspect Theft**

In step 1 of the Basic Flow, if the customer id is on the system’s list of stolen identification, the system displays a “Sorry, not ...” message. The system also records the date, time. And computer address...

#### **A4. Quit**

The RU e-st System allows the Trading Customer to quit at any time during the use case. The use case ends.

#### **A5. No Reply From User**

At any time during the use case, if the system asks for input from the Trading Customer....

在任何步骤  
都可能发生

# 细化特殊备选流

## Get Quote

### Alternative Flows

#### A1 Unidentified Trading Customer

In Step 1, Customer Logs On, in the Basic flow, if the system determines that the customer id and/or password are not valid,...

#### A2 Quote System Unavailable

In Step 3, Customer Gets Quote, in the basic flow, if the system is unable to communicate with the Quote System, the system...

#### A3 Quit

The RU e-st System allows the Trading Customer to quit at any time during the use case. The use case ends.

#### A4 Unknown Trading Symbol

In Step 3, Customer Gets Quote, in the Basic flow, if the system cannot recognize the trading symbol, the system notifies the customer that the trading symbol is not recognized. The use case continues at the beginning of Step 3.

#### A5 Quote System Cannot Locate Information

In Step 3, Customer Gets Quote, in the Basic Flow, if the Quote System responds that it does not have the requested information...

描述发生了  
什么

开始

条件

动作

继续

# 举例：备选流的另一种编号方法

## 基本流 Basic Flow

1. 采购员在初始申购单中添加申购子项。
2. 采购员输入供应商、交货地点、最终价格、返利、加价和费用明细。
3. 采购员增加、删除和修改申购单（包括第1、2步）直到满意为止。
4. 采购员在输入所有必要信息后，保存并完成申购单。
5. 系统检验申购单，分配申购单号，设置申购单和申购单子项状态为“未提交”。
6. 采购员提交申购单。
7. 系统设置申购单状态为“待审批”，通知采购经理审批申购单。

## 备选流 Alternative Flows

1-3a 退出：

- 1) 系统提示用户保存。
- 2) 用户选择不保存，系统放弃临时信息，申购单状态不变。

4a. 申购单信息不完整：

系统提示不能保存，回到步骤3。

# What Is an Activity Diagram?

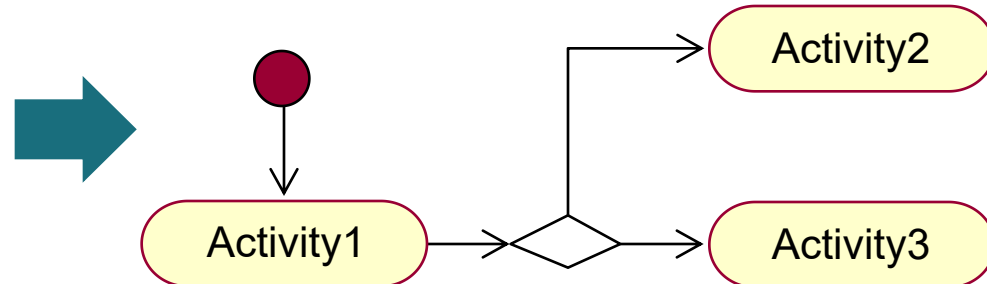
- ❑ An activity diagram in the Use-Case Model can be used to capture the activities in a use case.
- ❑ It is essentially a flow chart, showing flow of control from activity to activity.

## ***Flow of Events***

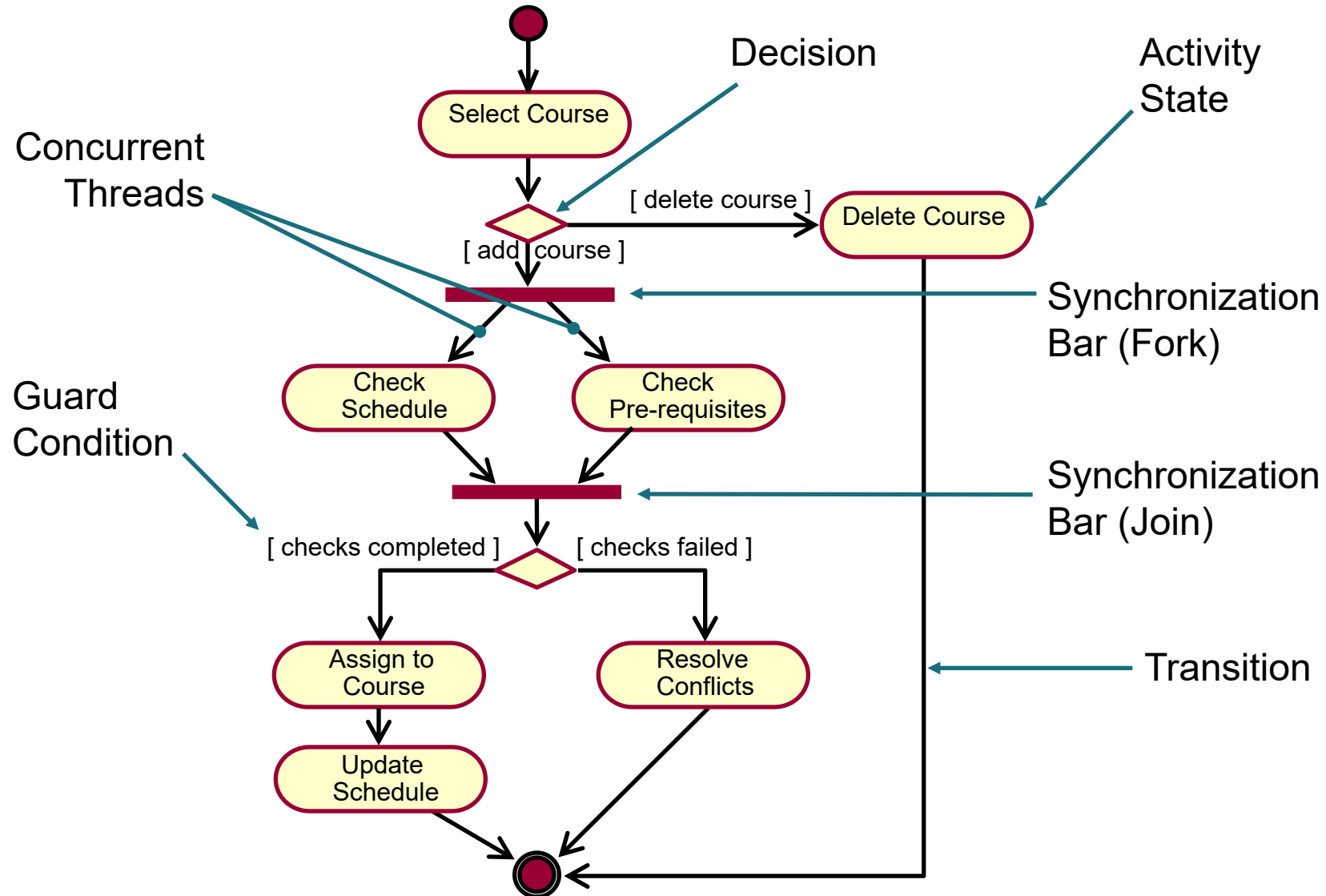
This use case starts when the Registrar requests that the system close registration.

1. The system checks to see if registration is in progress. If it is, then a message is displayed to the Registrar and the use case terminates. The Close Registration processing cannot be performed if registration is in progress.

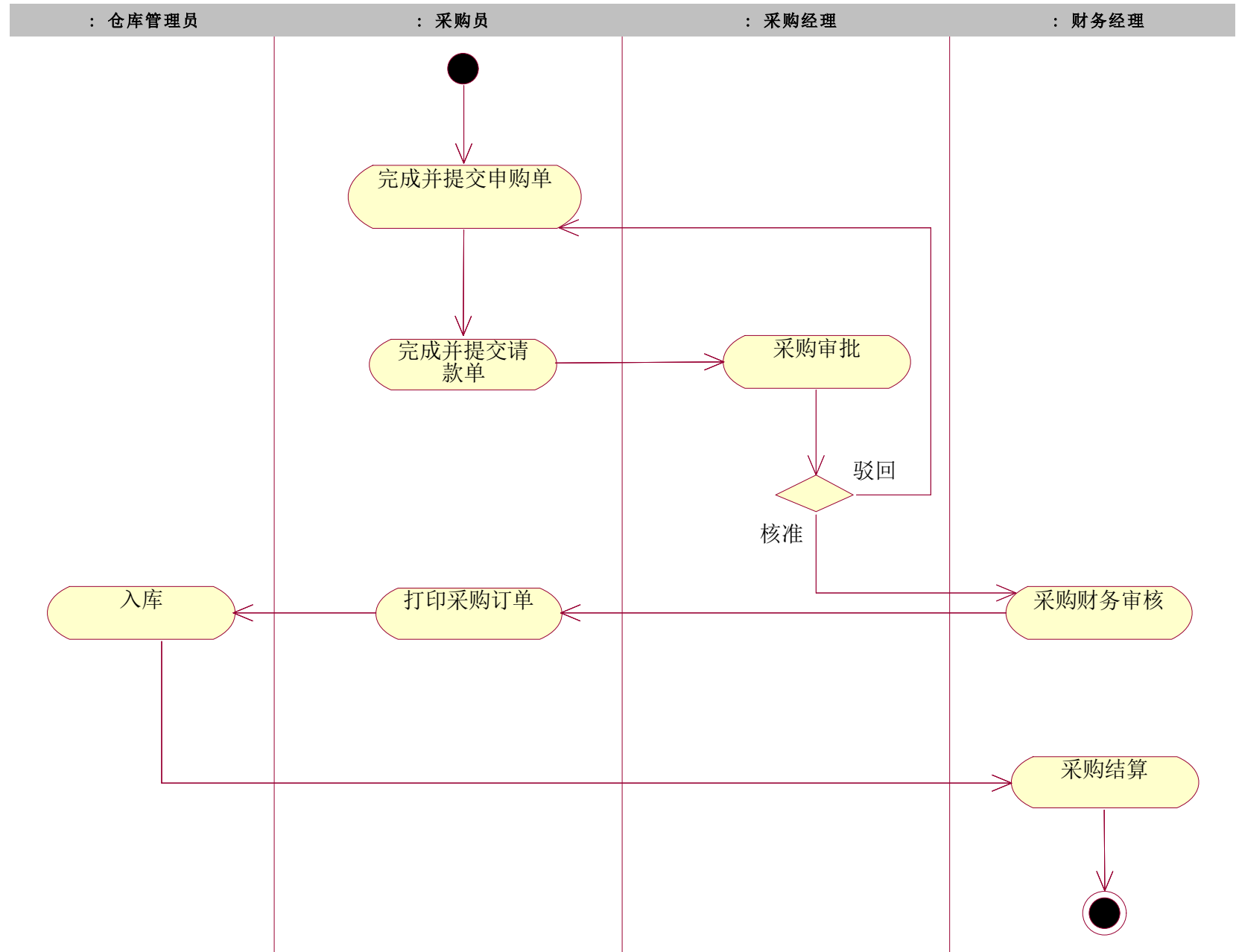
2. For each course offering, the system checks if a professor has signed up to teach the course offering and at least three students have registered. If so, the system commits the course offering for each schedule that contains it.



# Example: Activity Diagram



# Partitions





# 前置条件

---

- Use-case 启动的约束条件
- 不是触发Use-case 的事件
- 可选的: 仅当需要时才用

示例:

- 必须满足下面的前置条件, ATM 系统才能出钞:
  - 必须能进入 ATM 网络。
  - ATM 必须处于准备接受交易的状态。
  - ATM 中必须备有一些现金可供出钞。
  - ATM 必须备有足够的纸张打印至少一次交易的收据

# 后置条件

---

- 当use case结束时,后置条件一定为真
- 用于降低用例事件流的复杂性并提高其可读性
- 还可用来陈述在用例结束时系统执行的动作
- 可选的: 仅当需要时才用

示例:

- ATM 在用例结束时总是显示“欢迎使用”消息,可将此消息记录在用例后置条件中。
- 与此类似,如果 ATM 在如提取现金这样的用例结束时总会停止客户交易,则不管事件进程如何,将这种情况记录为用例后置条件。

# 其它Use Case属性

---

- 非功能需求
  - 有关本use-case的URPS+
- 业务规则
  - 在执行事件流时，使用到的重要业务规则或计算公式
- 扩展点
  - use-case可以通过另一use-case进行扩展
- 关系
  - 和actors及use-case的关联
- Use-case 图
  - 涉及本use-case的关系的可视化模型
- Other diagrams or enclosures
  - 交互、活动或其它图
  - 用户界面框图

# 面向对象分析的步骤



## 1. 建立用例模型

1.1 识别actor和use case, 画Use-Case图

1.2 编写Use-Case Spec.

**1.3 优化Use-Case图的结构**



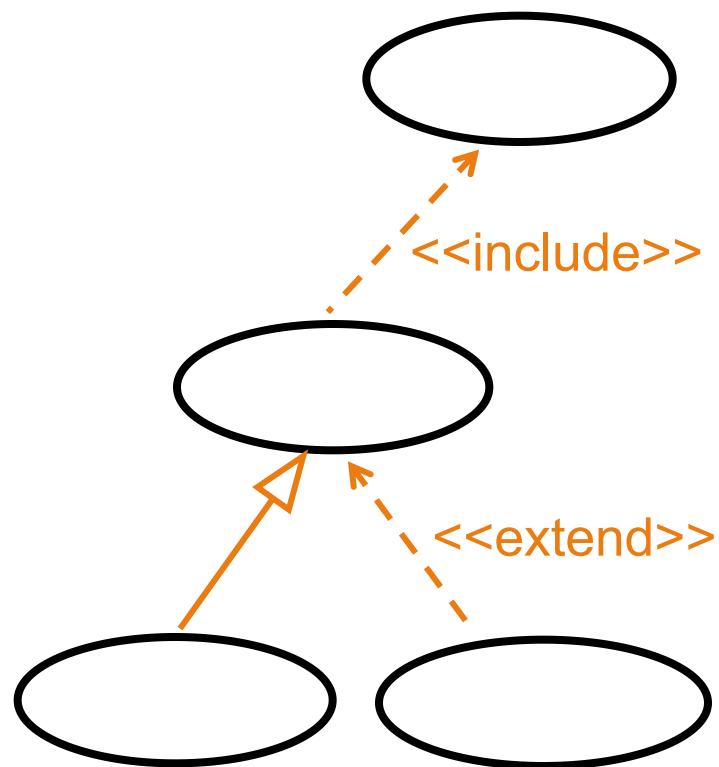
## 2. 建立概念模型



## 3. 建立分析模型

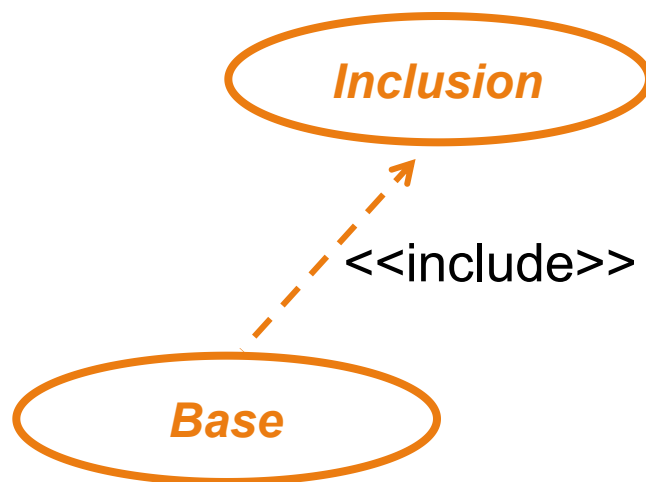
# Use Case间的关系

- Include 包含
- Extend 扩展
- Generalization 泛化

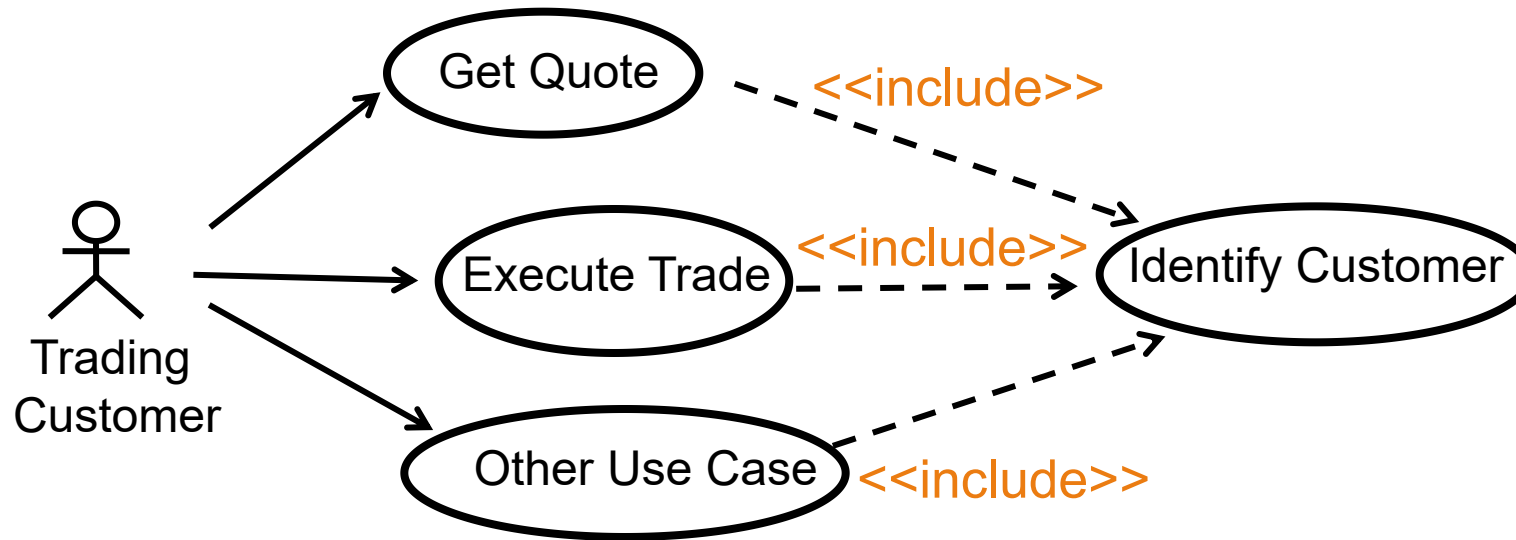


# 什么是Include 关系?

- 基本用例包含某个包含用例
- 包含用例定义的行为将显式插入基本用例



# Include 关系示例



## Get Quote Use Case

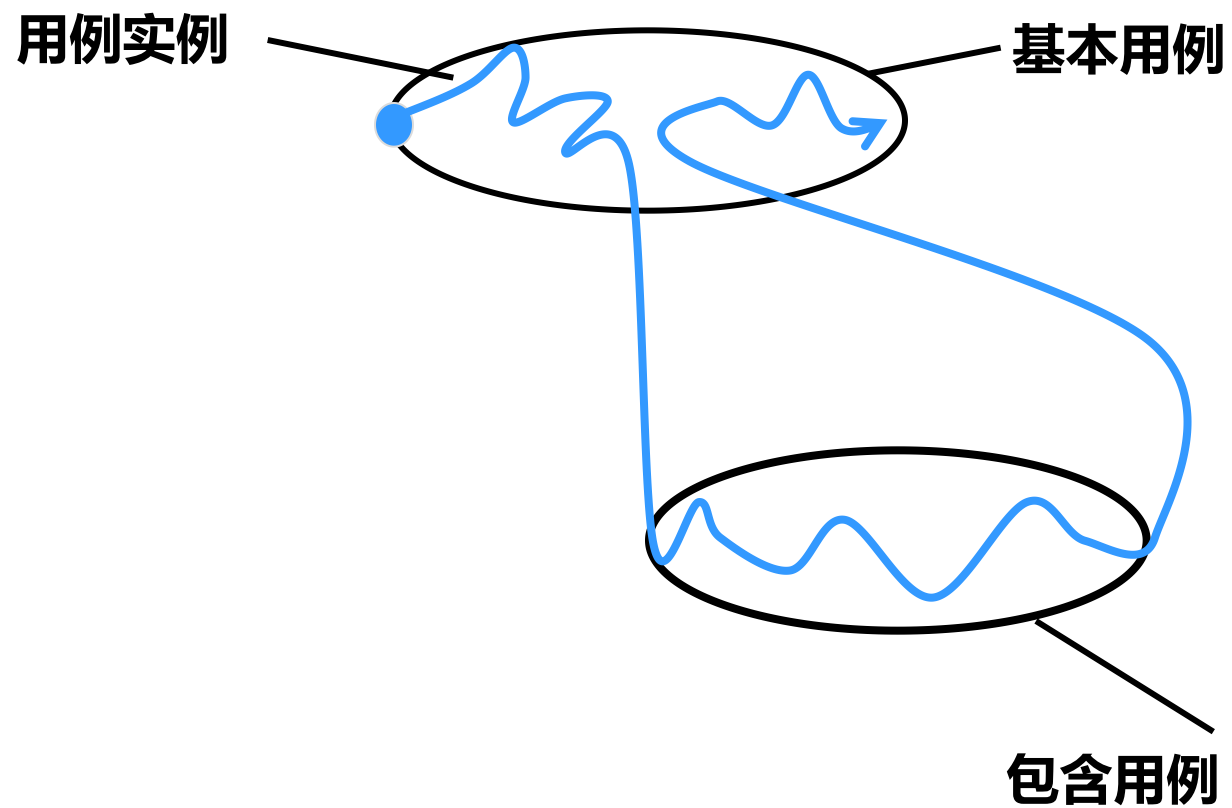
1. **Include “Identify Customer”** to verify customer’s identity
2. Display options. Customer selects “Get Quote”
3. ...

## Identify Customer Use Case

1. Log on
  2. Validate logon
  3. Enter password
  4. Verify password
- A1: Not valid logon  
A2: Wrong password  
A3: ...

# 执行Include

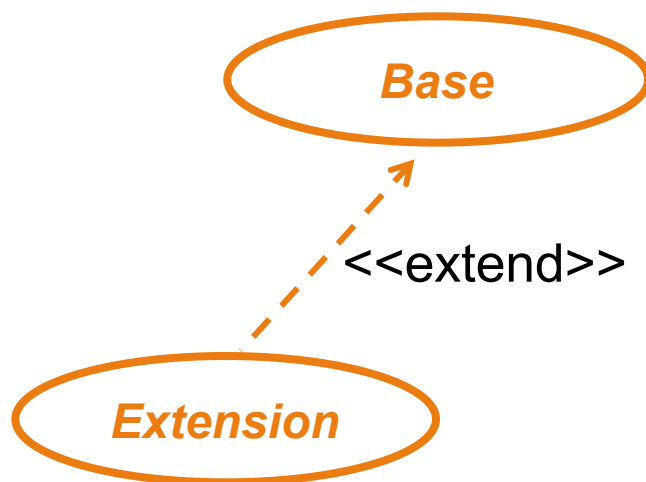
- 到达插入点时执行包含用例



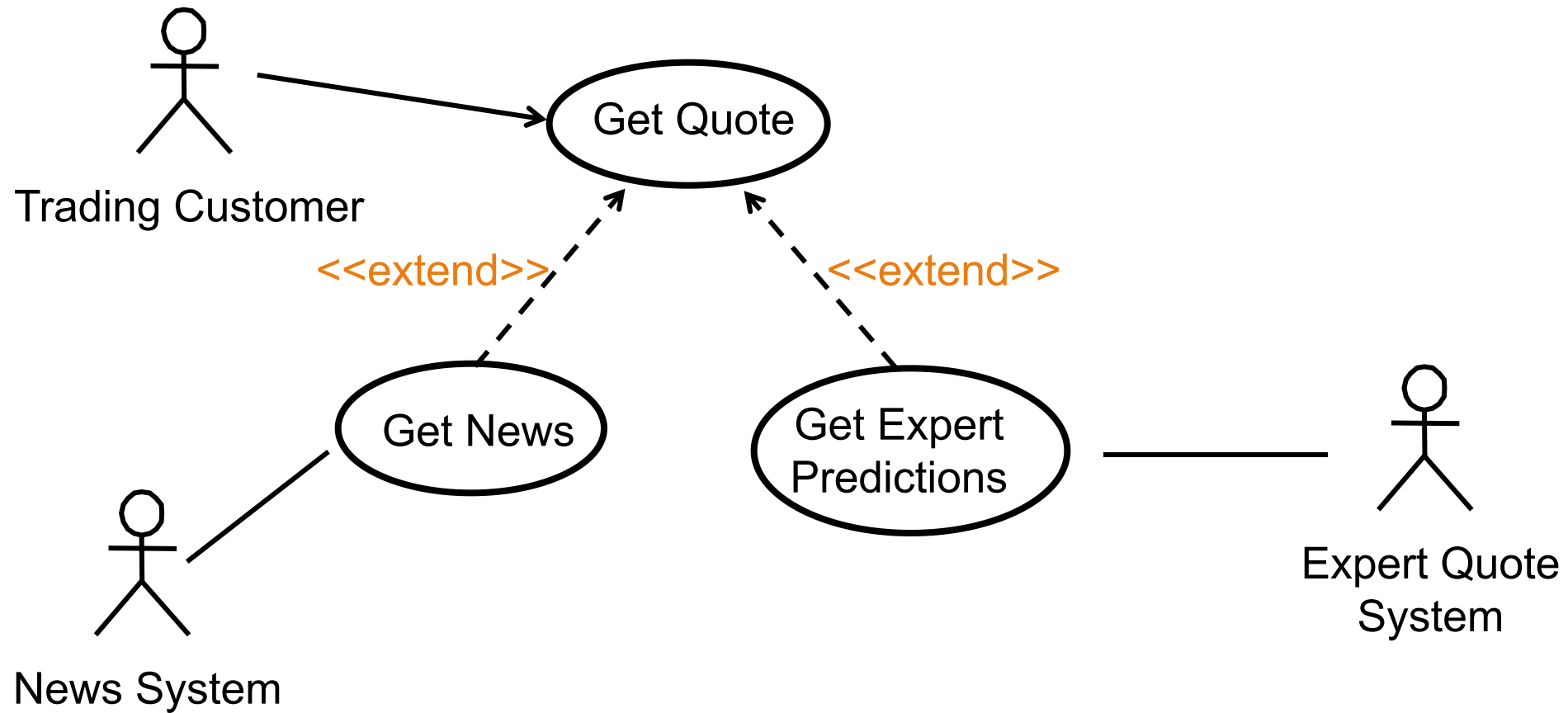


# 什么是Extend关系?

- 从基本用例到扩展用例的联接
  - 将扩展用例的行为插入基本用例
  - 只有当扩展条件为真是才进行插入
  - 在一个命名的扩展点插入基本用例



# Extend 关系示例



# Extend 关系示例(续)

## Get Quote Use Case

### Basic Flow:

1. Include “Identify Customer” to verify customer’s identity.
2. Display options.
3. Customer selects “Get Quote.”
4. Customer gets quote.
5. Customer gets other quotes.
6. Customer logs off.

A1. Quote System unavailable

...

### Extension Points:

The “Optional Services”

extension point occurs at Step 3 in the Basic Flow.

## Get News Use Case

This use case extends the Get Quote Use Case, **at extension point “Optional Services.”**

### Basic Flow:

1. **If Customer selects “Get News,”** the system asks customer for time period and number of news items.
2. Customer enters time period and number of items. The system sends stock trading symbol to News System, receives reply, and displays the news to the customer.
3. The Get Quote Use Case continues.

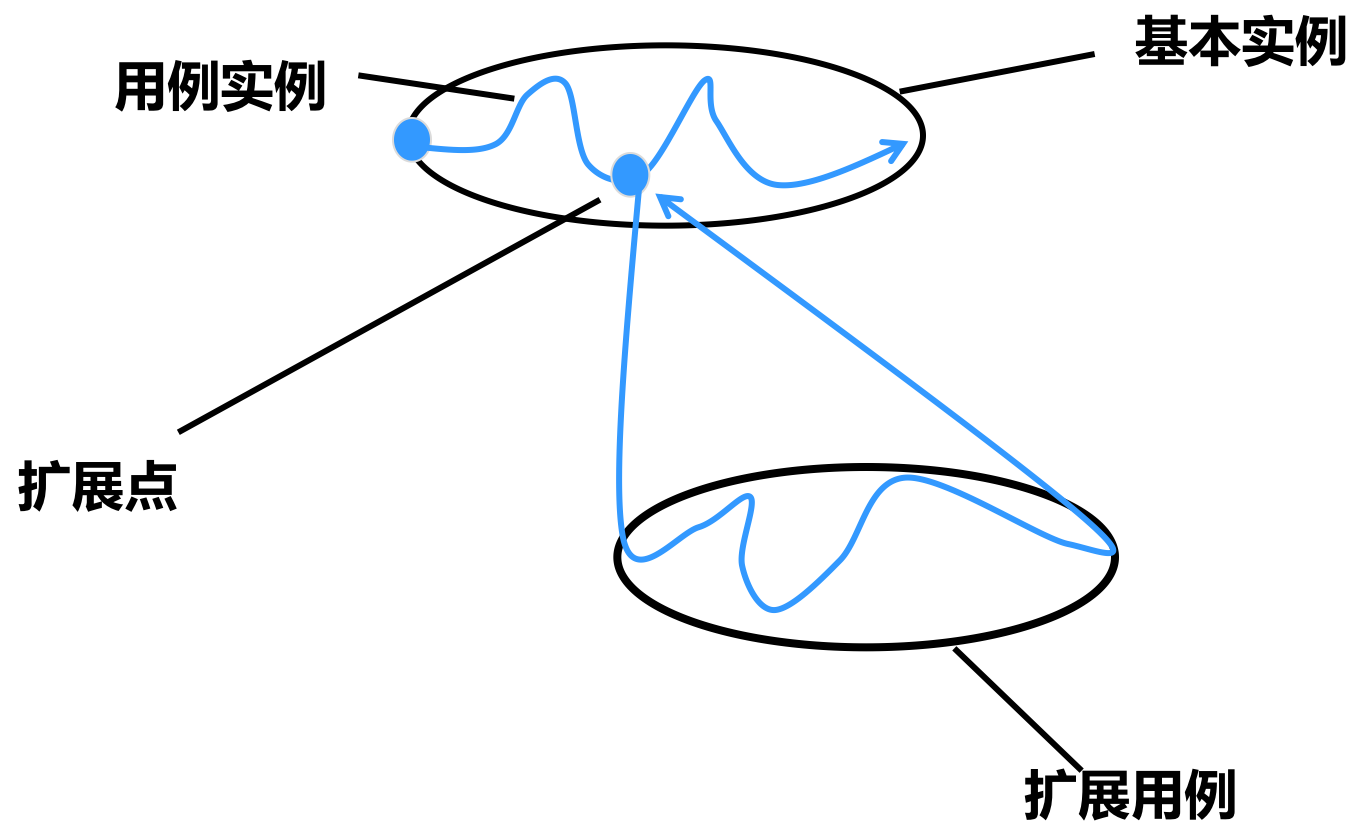
A1: News System Unavailable

A2: No News About This Stock

...

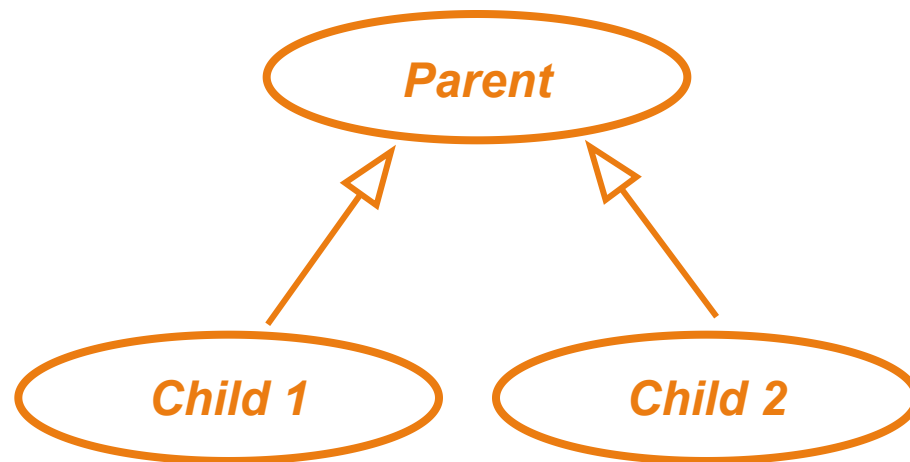
# 执行Extend

- 当到达扩展点并且扩展条件为真时执行



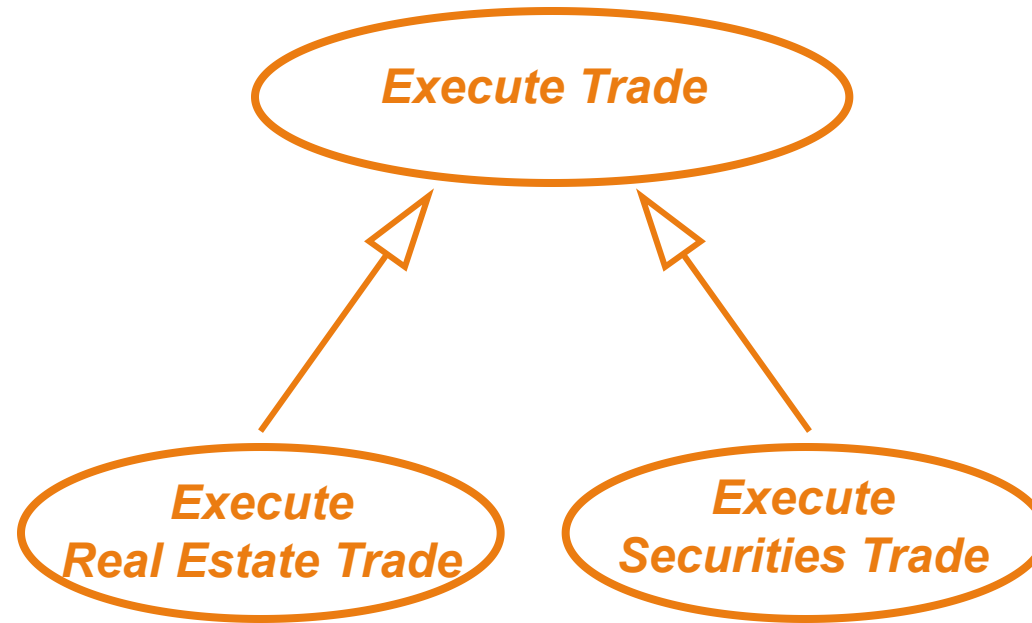
# 什么是 Use-Case Generalization?

- 从子用例到父用例的关系
  - 在父用例中描述通用的共享的行为
  - 在子用例中描述特殊的行为
  - 共享共同的目标



# Generalization示例

---



# Generalization 关系示例

## Execute Trade Use Case

Basic Flow:

1. Customer Logs On  
Include “Identify Customer”  
to verify customer identity ...
  2. Customer Selects “Trade.”  
Customer chooses trade ...
  3. Customer Selects Account  
Customer selects account.  
System displays account ...
  4. *Perform Trade*
  5. Customer Begins New Trade  
If customer wants other  
trades, repeat from Step 3 ...
  6. Display Summary  
System displays summary ...
- A1. Account Not Operational...

## Execute Securities Trade Use Case

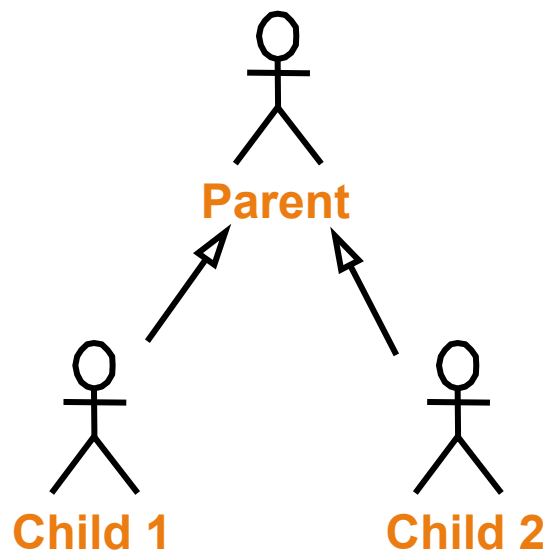
This use case is a **child** of the Execute Trade Use Case.

Basic Flow:

1. Customer Logs On
  2. Customer Selects “Trade”
  3. Customer Selects Account
  4. *Perform Trade*  
*If customer selects trade order type. The system performs...Limit Sell, Limit Buy, ...  
The system sends...*
  5. Customer Begins New Trade
  6. Display Summary
- A1: Limit Sell Order ...

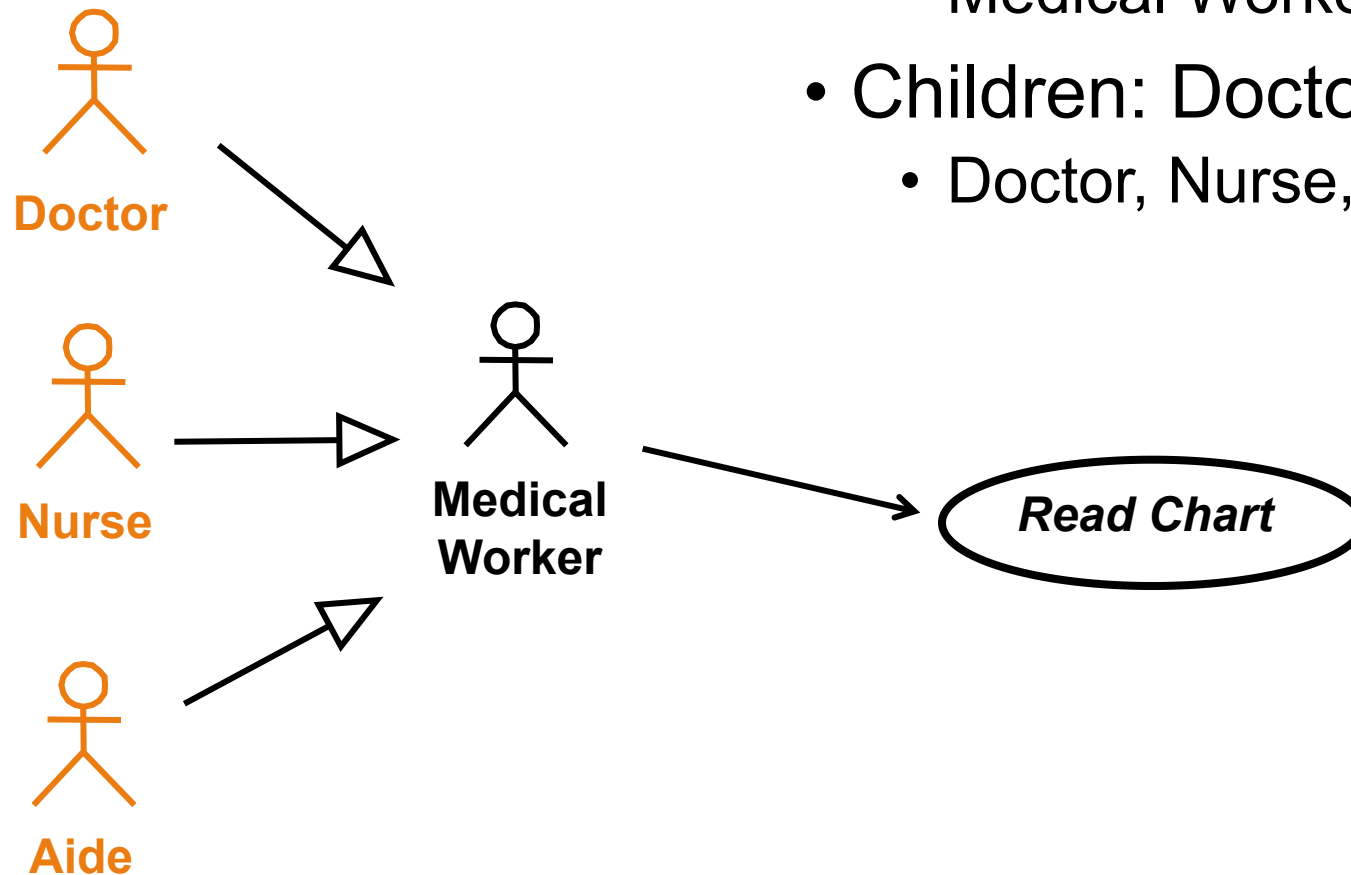
# 什么是Actor Generalization?

- Actors 可能有公共的属性
- 多个actors 和Use-Case交互时可能有公共的角色或目的





# Actor Generalization 示例



- Parent: Medical Worker
  - Medical Worker can read charts
- Children: Doctor, Nurse, Aide
  - Doctor, Nurse, and Aide can read charts

# 面向对象分析的步骤



## 1. 建立用例模型

- 1.1 识别actor和use case, 画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构



## 2. 建立概念模型

- 2.1 识别Conceptual Class**
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性, 画状态图



## 3. 建立分析模型

# 概念类图的作用

---

- When modeling the static view of a system, class diagrams are typically used in one of three ways, to model:
  - The vocabulary of a system
  - Collaborations
  - A logical database schema

# Identify Conceptual Class

---

- Strategies to Identify Conceptual Classes
  - Use a conceptual class category list
  - Finding conceptual classes with Noun Phrase
  - Use analysis patterns, which are existing conceptual models created by experts
    - using published resources such as Analysis Patterns [Fowler96] and Data Model Patterns [Hay96].

# Use a conceptual class category list

---

Category	Conceptual Classes
physical or tangible objects	Student Professor FulltimeStudent ParttimeStudent
form or abstract noun concepts	Course Course Offering Schedule
organizations	Dept

# Finding conceptual classes with Noun Phrase

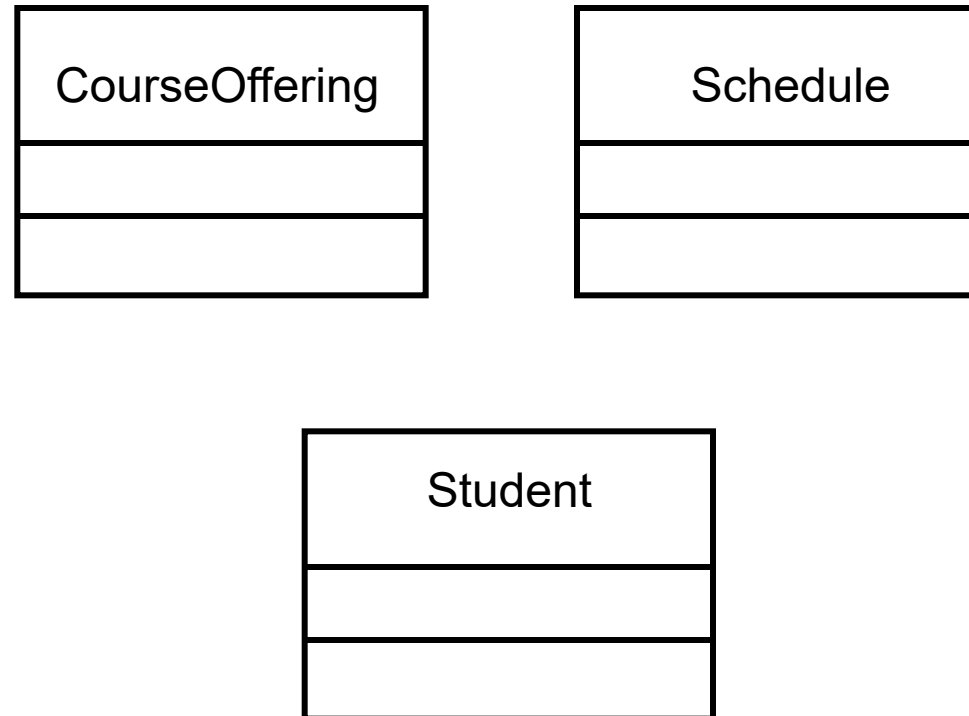
---

- ❑ Use use-case flow of events as input
- ❑ Underline noun clauses in the use-case flow of events
- ❑ Remove redundant candidates
- ❑ Remove vague candidates
- ❑ Remove actors (out of scope)
- ❑ Remove implementation constructs
- ❑ Remove attributes (save for later)
- ❑ Remove operations

# Example: Candidate Conceptual Classes

---

- Register for Courses (Create Schedule)



# A Common Mistake in Identifying Conceptual Classes

---

- ❑ Perhaps the most common mistake when creating a domain model is to represent something as an attribute when it should have been a concept.
- ❑ A rule of thumb to help prevent this mistake:
  - If we do not think of some conceptual class X as a number or text in the real world, X is probably a conceptual class, not an attribute.
- ❑ Example
  - Should store be an attribute of Sale, or a separate conceptual class Store?
  - In the real world, a store is not considered a number or text - the term suggests a legal entity, an organization, and something occupies space.
  - Therefore, Store should be a concept.



# 面向对象分析的步骤



## 1. 建立用例模型

- 1.1 识别actor和use case, 画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构



## 2. 建立概念模型

- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系**
- 2.3 增加 Conceptual Class的属性, 画状态图



## 3. 建立分析模型

# Class relationships

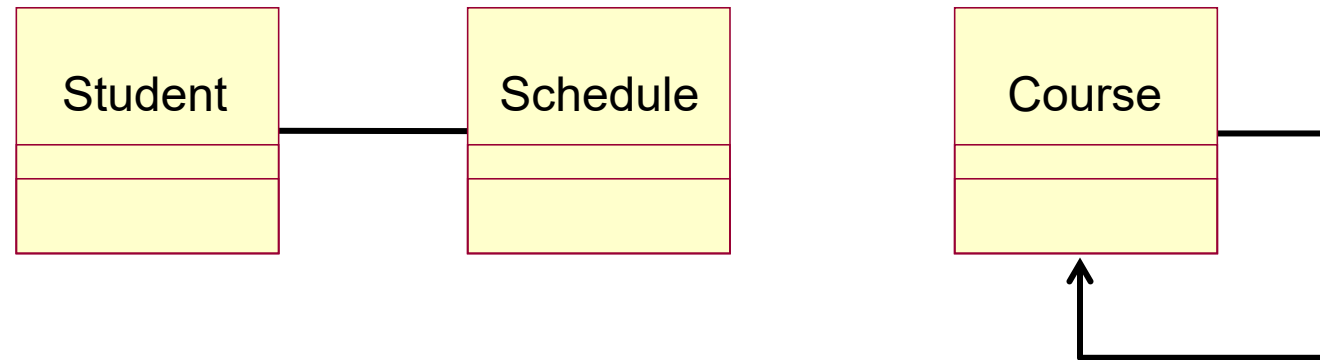
---

- Association
  - Aggregation
  - Composition
- Generalization
- Dependency

# Relationships: Association (关联)

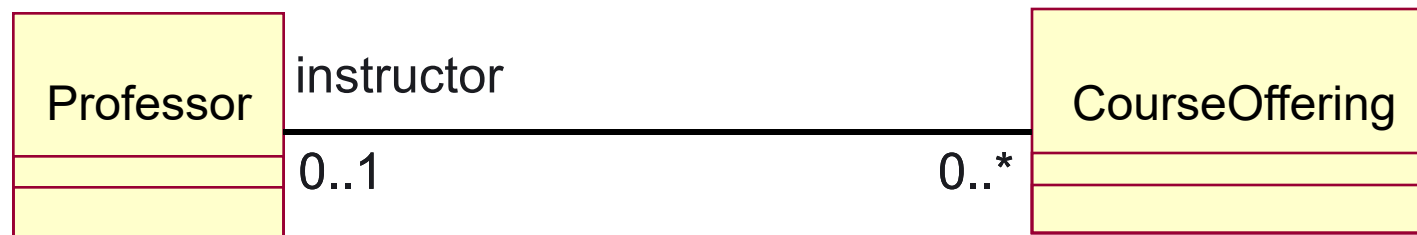
---

- ❑ The semantic relationship between two or more classifiers that specifies connections among their instances.
- ❑ A structural relationship specifying that objects of one thing are connected to objects of another thing.



# What Is Multiplicity?

- Multiplicity is the number of instances one class relates to ONE instance of another class.
- For each association, there are two multiplicity decisions to make, one for each end of the association.
  - For each instance of Professor, many Course Offerings may be taught.
  - For each instance of Course Offering, there may be either one or zero Professor as the instructor.

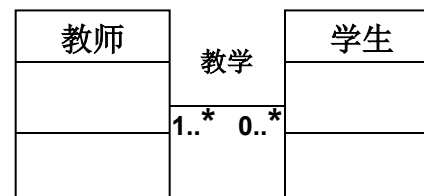
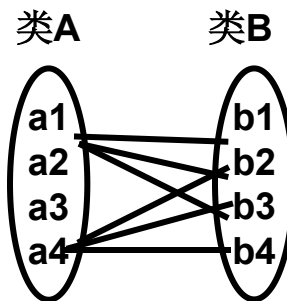
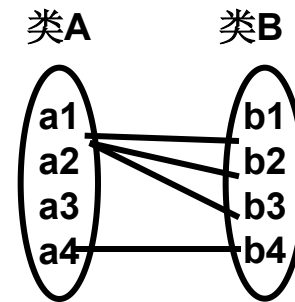
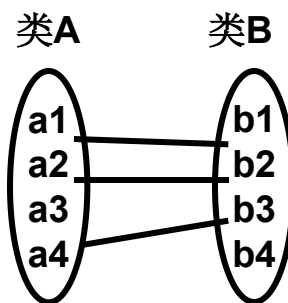
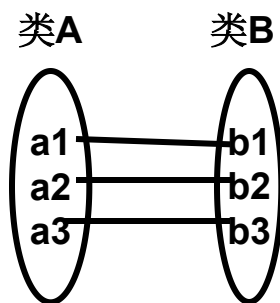


# Multiplicity Indicators

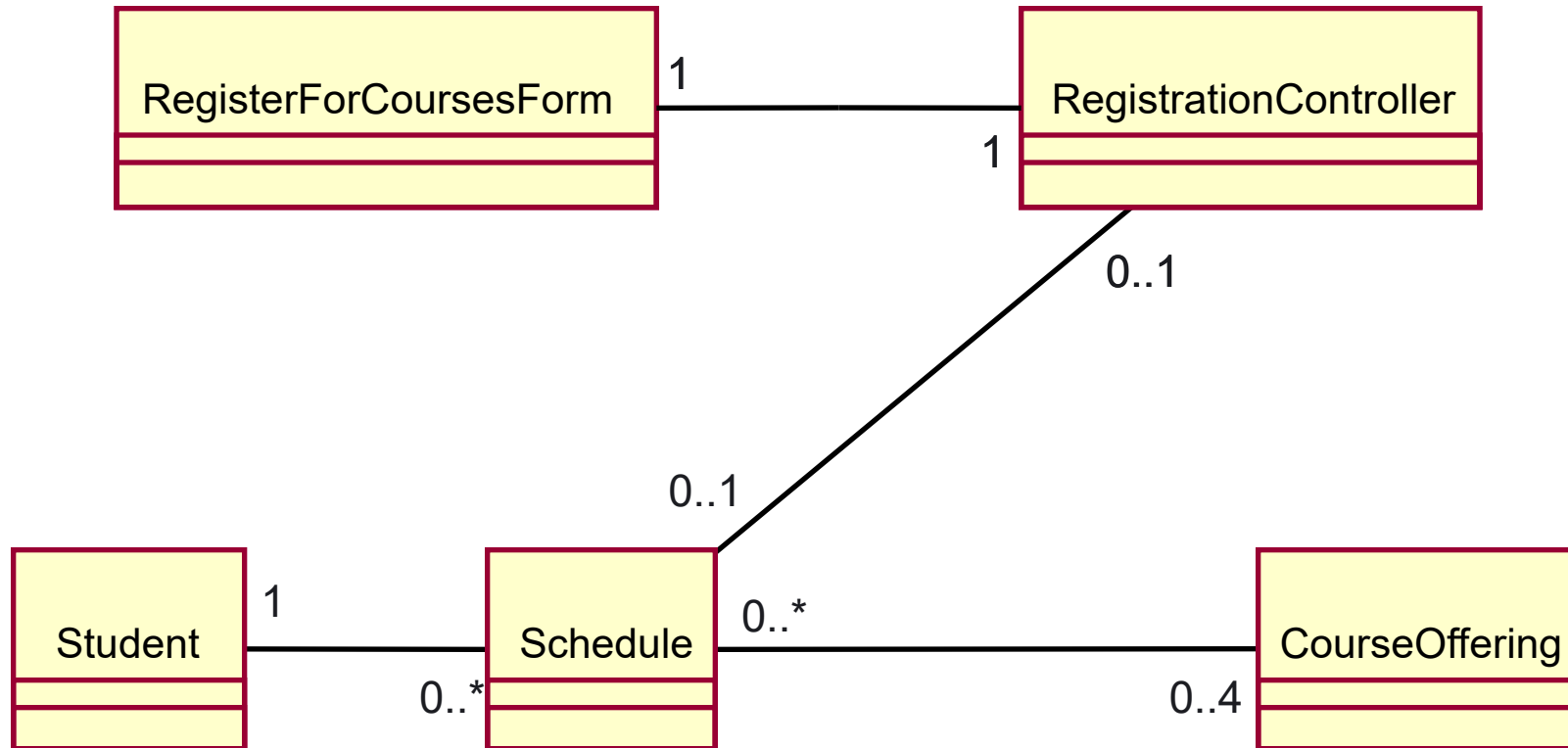
---

Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional value)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

# 举例

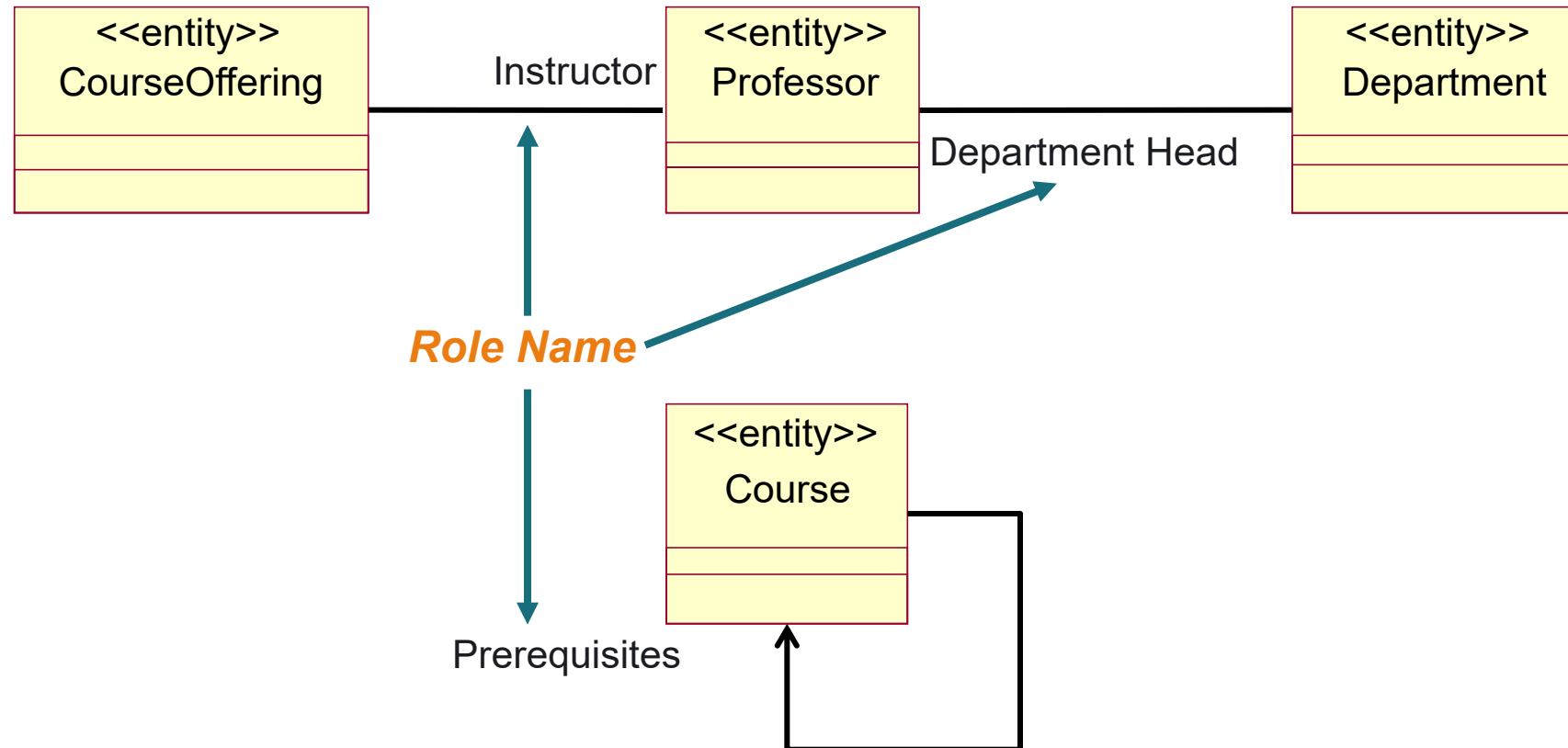


# Example: Multiplicity



# What Are Roles?

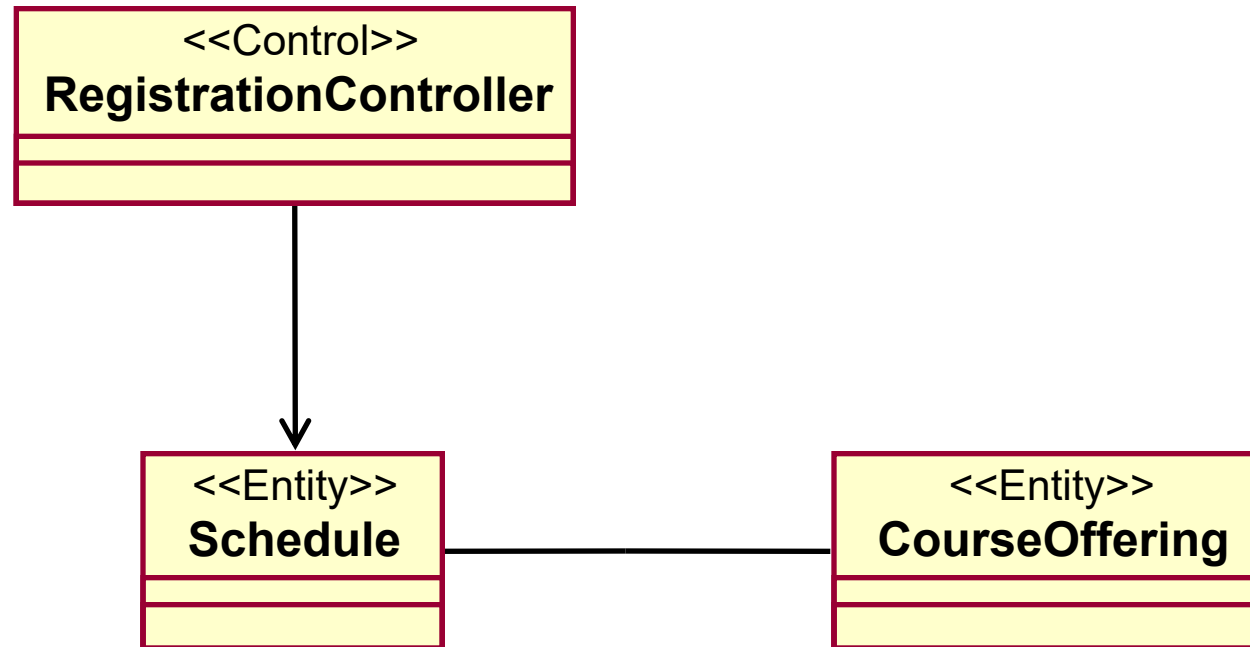
- The “face” that a class plays in the association



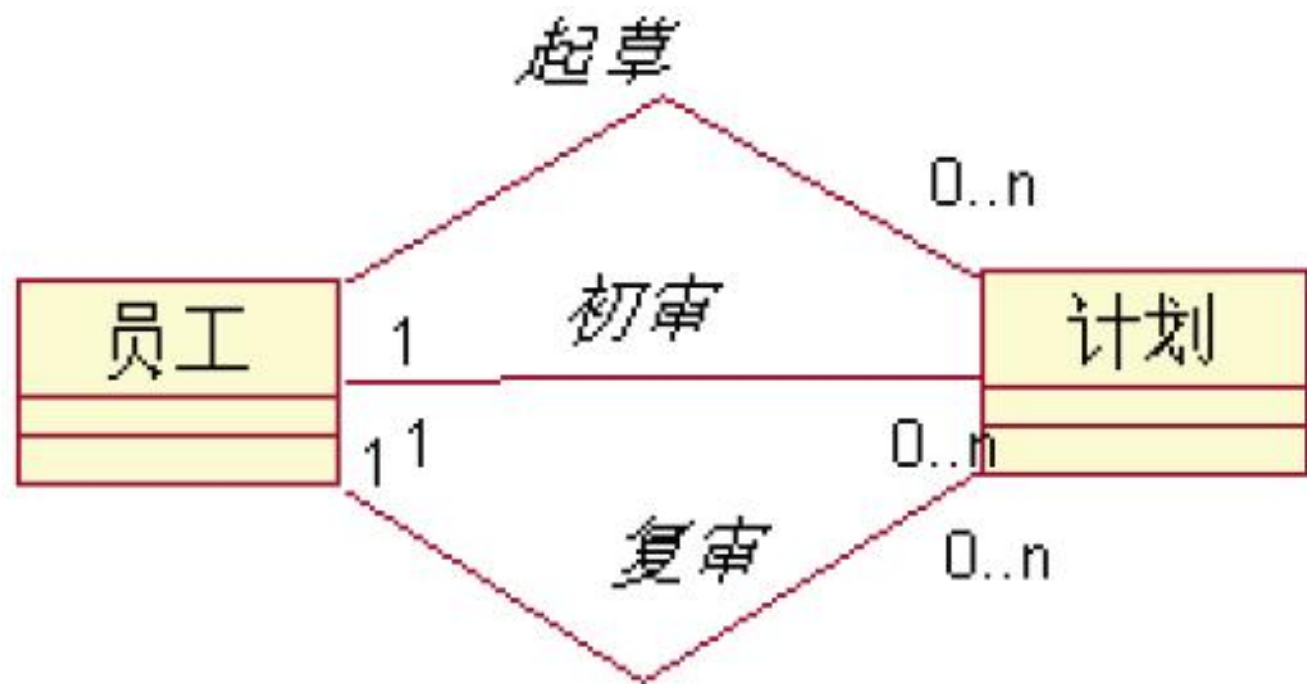


# What Is Navigability?

- Indicates that it is possible to navigate from a associating class to the target class using the association

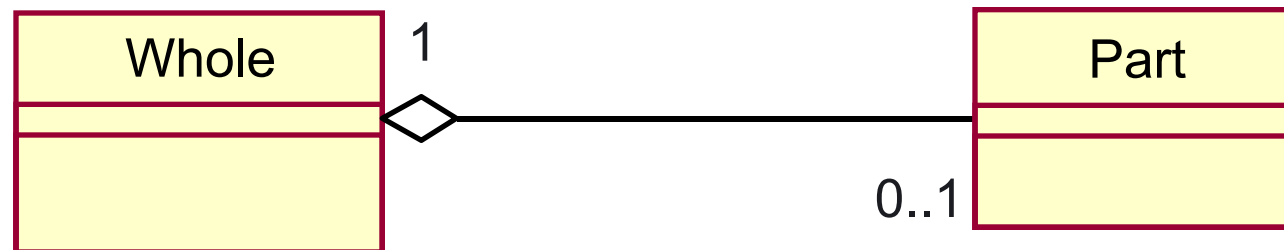


## 两个类之间关联同时可以有多条

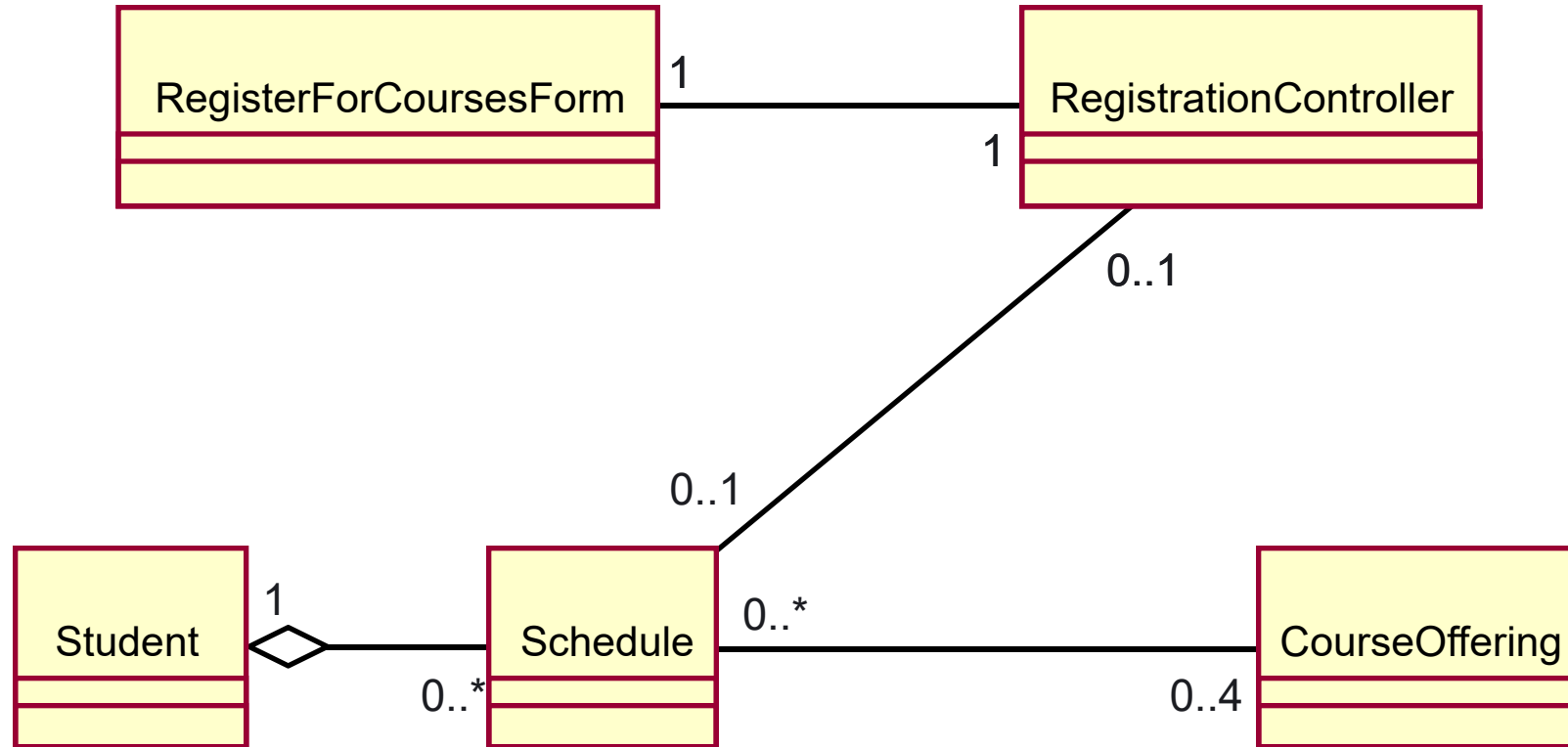


# What Is an Aggregation?

- A special form of association that models a whole-part relationship between the aggregate (the whole) and its parts.
  - An aggregation is an “is a part-of” relationship.
- Multiplicity is represented like other associations.

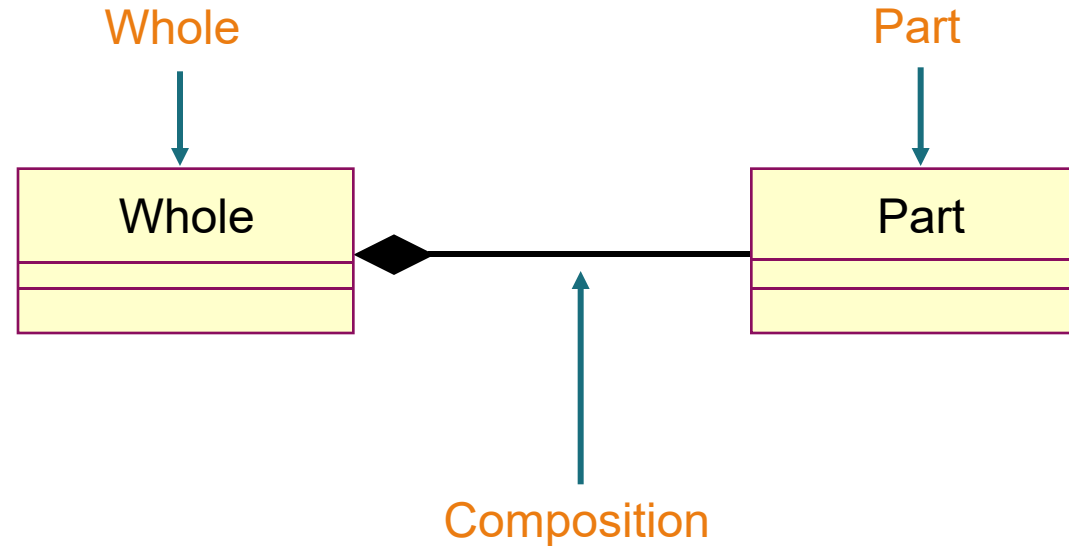


# Example: Aggregation

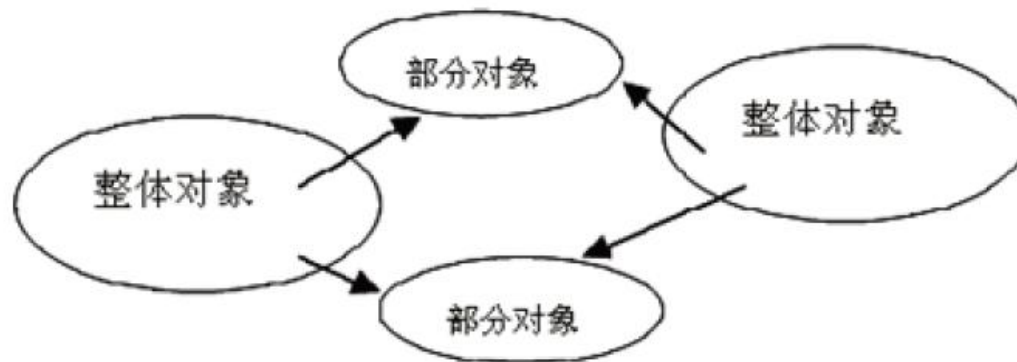


# What Is Composition?

- A form of aggregation with strong ownership and coincident lifetimes
  - The parts cannot survive the whole/aggregate

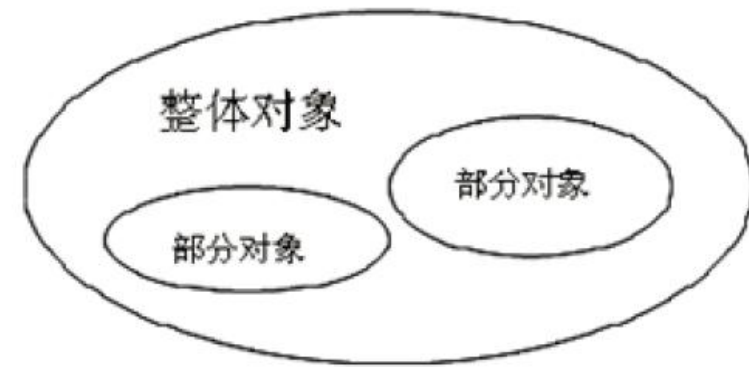


# Aggregation Vs. Composition



聚合

例：公司和雇员

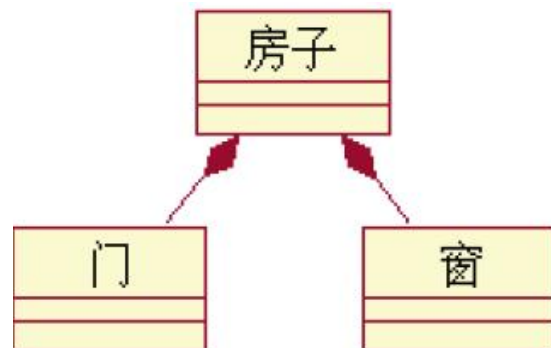


组合

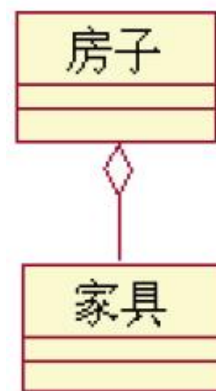
例：订单和订单项

# 举例

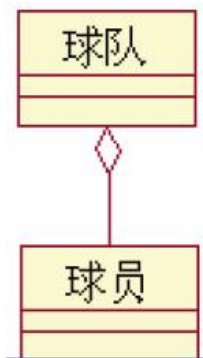
□ 组合/部分



□ 容器/内容



□ 集合/成员



# 练习

---

- 汽车和轮胎是关联、聚合、还是组合？
- 丈夫和妻子是关联、聚合、还是组合？



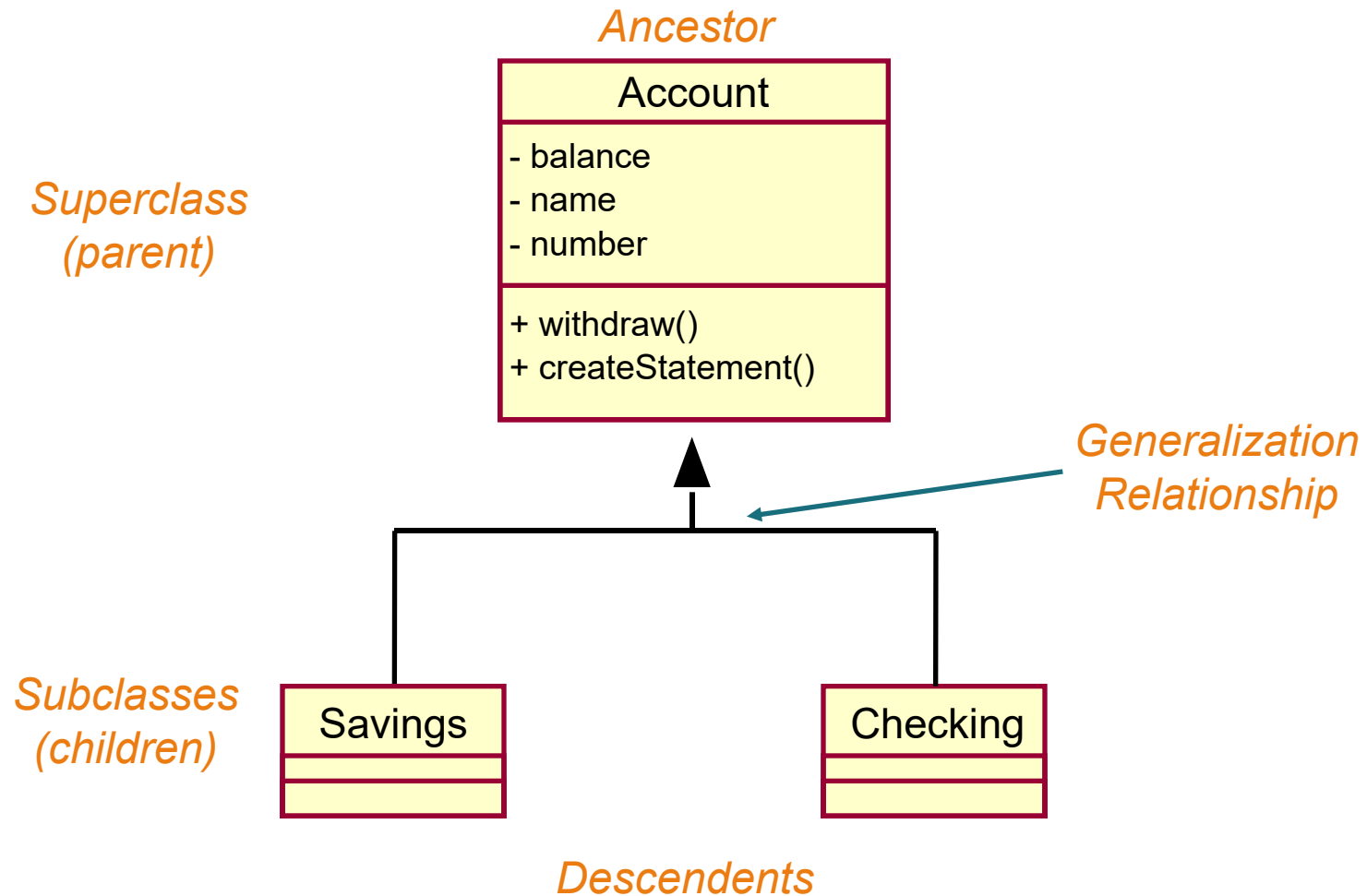
# Relationships: Generalization

---

- A relationship among classes where one class shares the structure and/or behavior of one or more classes.
- Defines a hierarchy of abstractions where a subclass inherits from one or more superclasses.
  - Single inheritance
  - Multiple inheritance
- Is an “is a kind of” relationship.

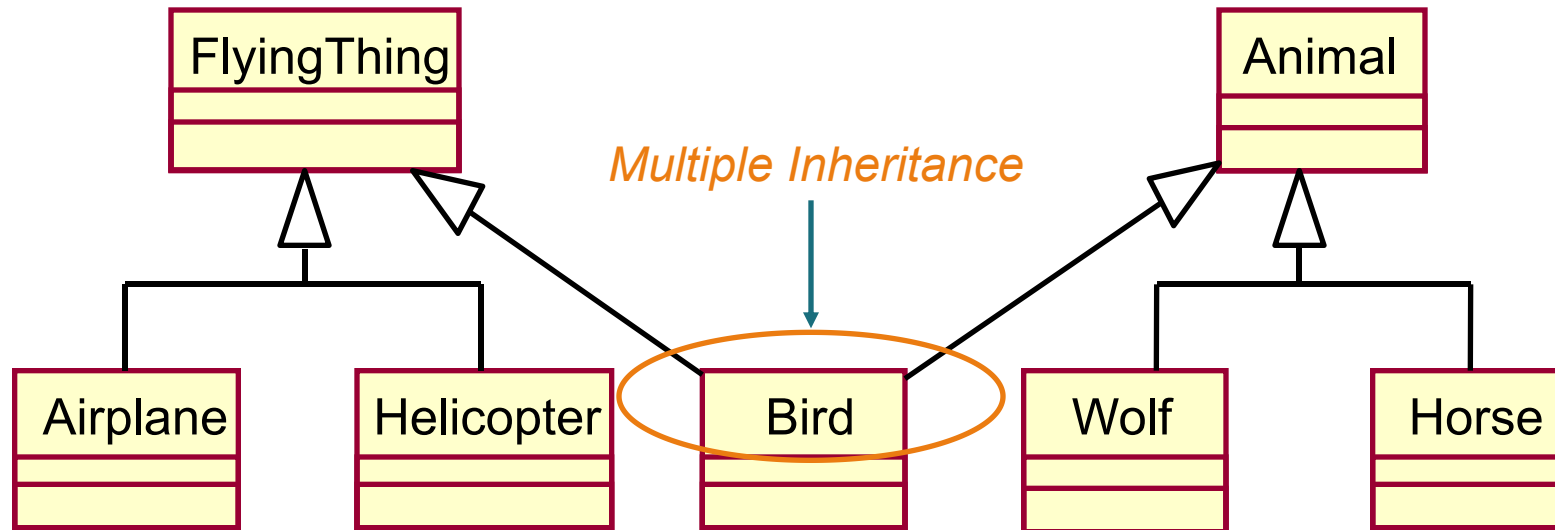
# Example: Single Inheritance

- One class inherits from another.



# Example: Multiple Inheritance

- A class can inherit from several other classes.



***Use multiple inheritance only when needed and  
always with caution!***

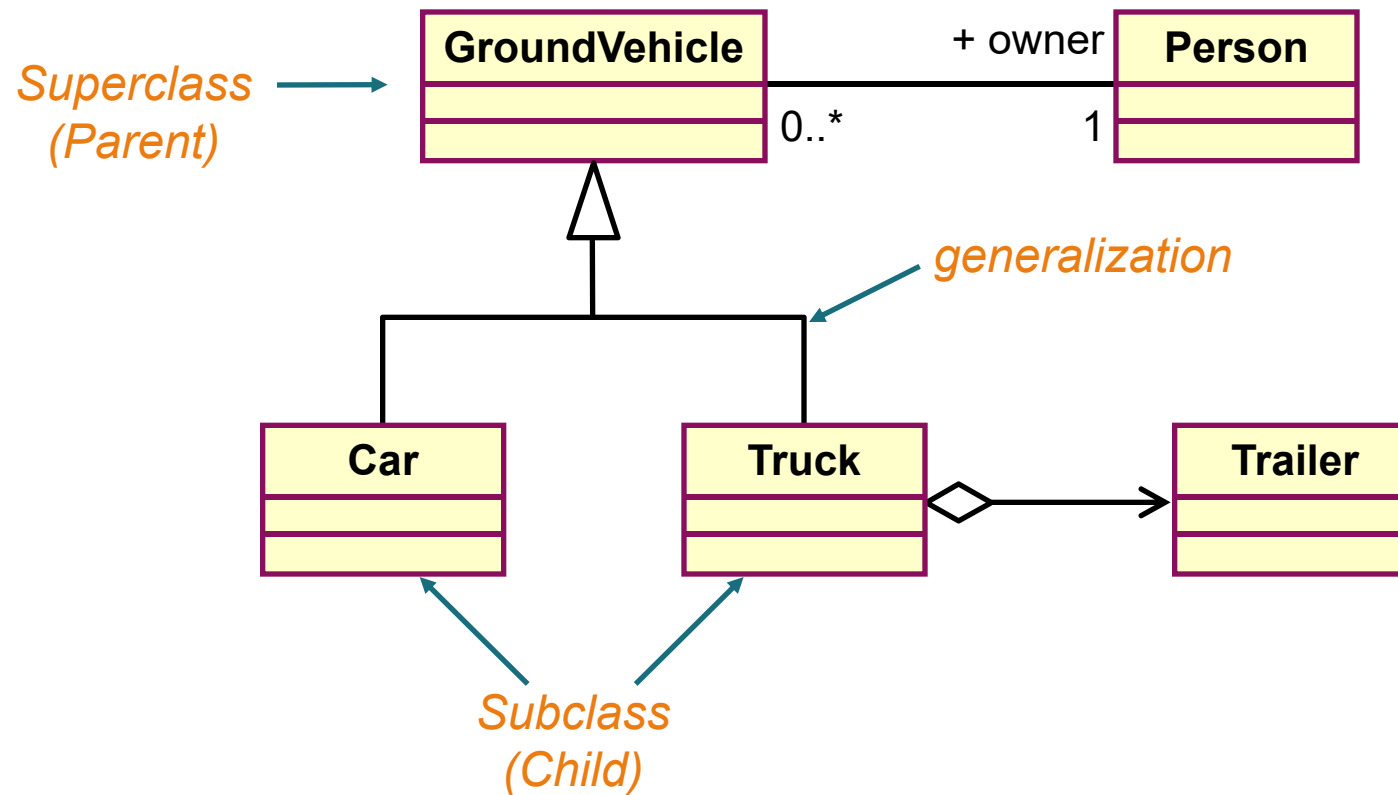
# What Is Inherited?

---

- ❑ A subclass inherits its parent's attributes, operations, and relationships.
- ❑ A subclass may:
  - Add additional attributes, operations, relationships.
  - Redefine inherited operations. (Use caution!)
- ❑ Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy.

Inheritance leverages the similarities among classes.

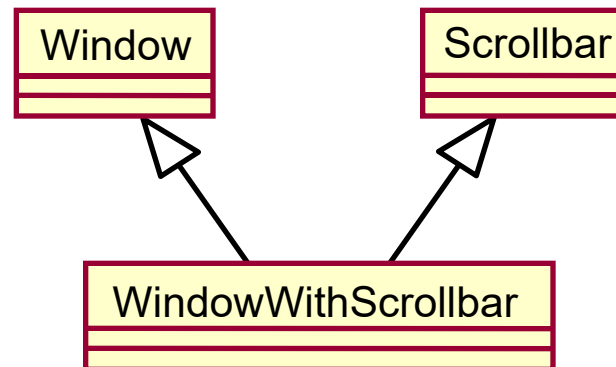
# Example: What Gets Inherited



# Generalization vs. Aggregation

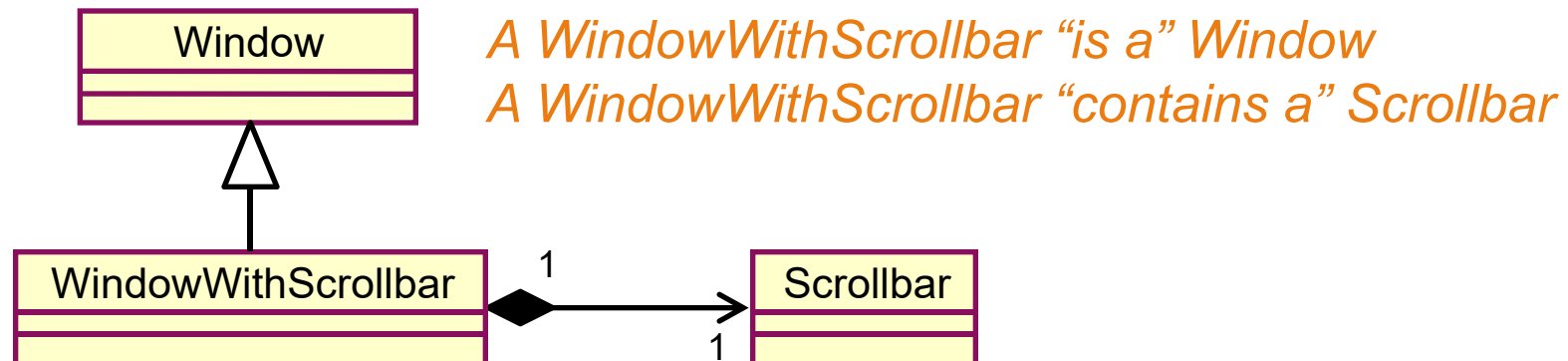
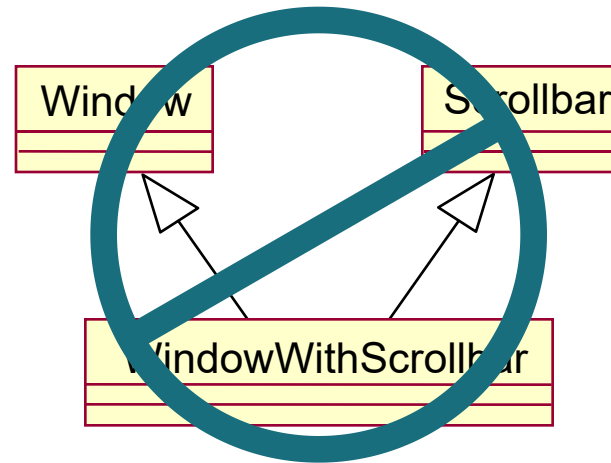
---

- Generalization and aggregation are often confused
  - Generalization represents an “is a” or “kind-of” relationship
  - Aggregation represents a “part-of” relationship



Is this correct?

# Generalization vs. Aggregation



# Liskov替换原则 —— LSP

## □ Liskov Substitution Principle (LSP)

子类型应该能替换基类型

----Barbara Liskov

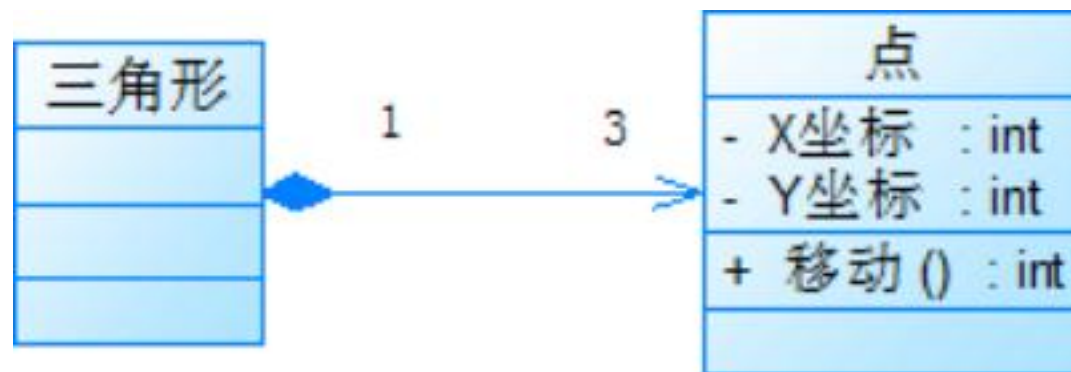
对吗?



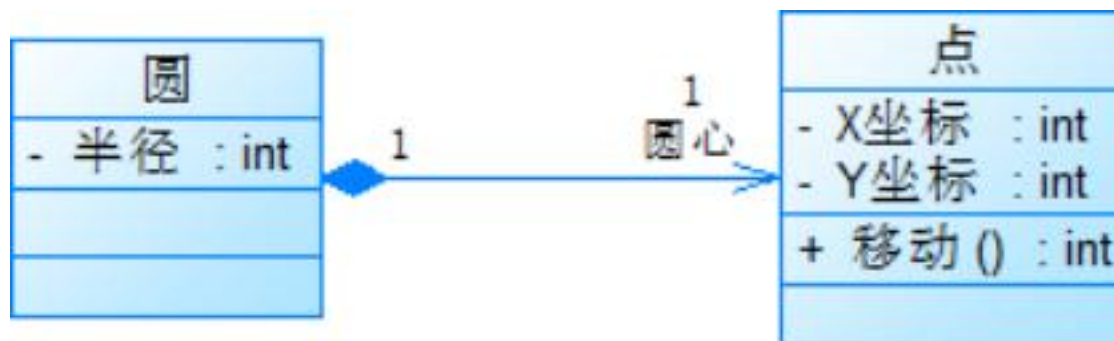
圆是一种有半径的点



# 子类能替换超类吗？



三个圆能组成三角形吗？

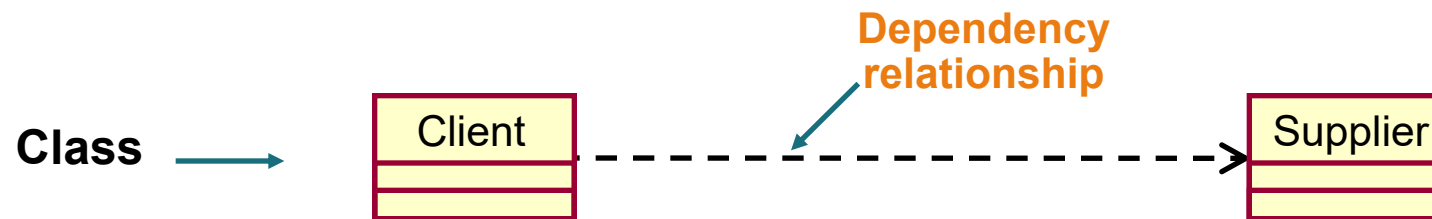


关联更能反映领域内涵

# Relationships: Dependency

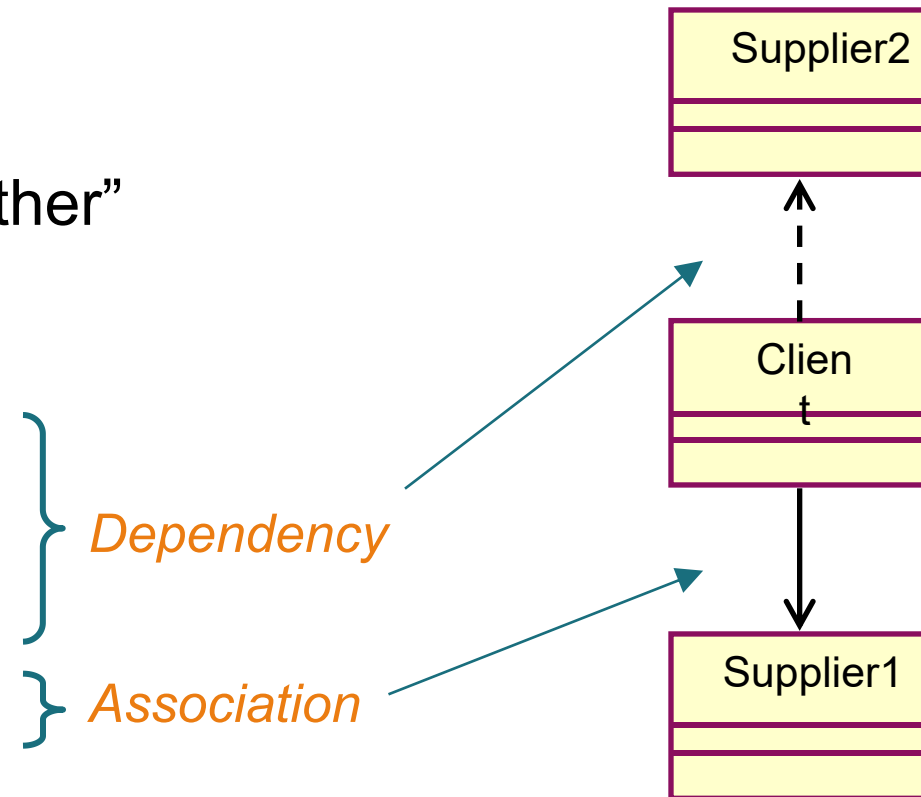
---

- A relationship between two classes where a change in one may cause a change in the other
- Non-structural, “using” relationship



# Dependencies vs. Associations

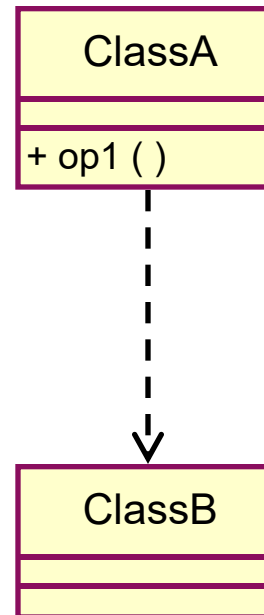
- Associations are structural relationships
- Dependencies are non-structural relationships
- In order for objects to “know each other” they must be visible
  - Local variable reference
  - Parameter reference
  - Global reference
  - Field reference



# Local Variable Visibility

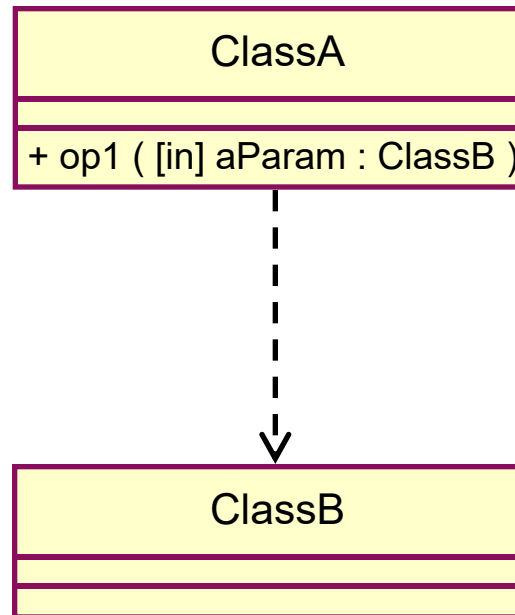
---

- The op1() operation contains a local variable of type ClassB



# Parameter Visibility

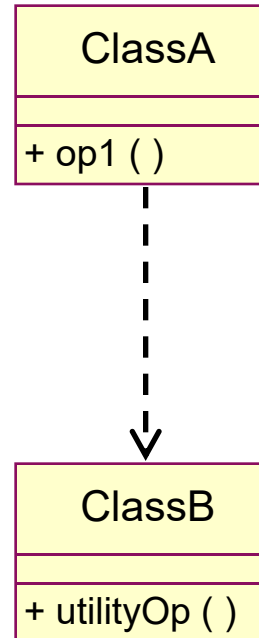
- The ClassB instance is passed to the ClassA instance



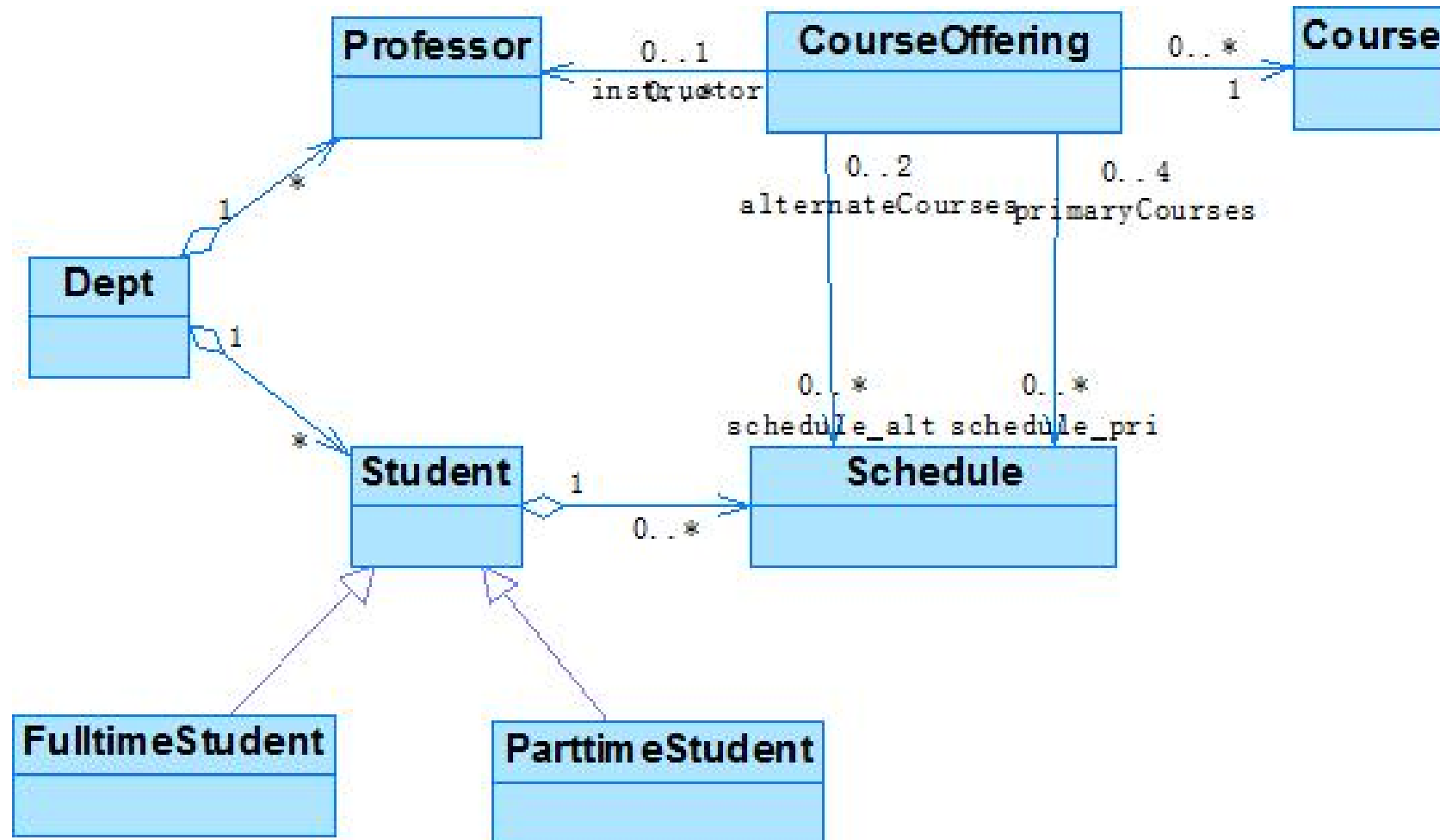
# Global Visibility

---

- The ClassUtility instance is visible because it is global



# Example: Conceptual Model



# 面向对象分析的步骤



## 1. 建立用例模型

- 1.1 识别actor和use case, 画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构



## 2. 建立概念模型

- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性, 画状态图**



## 3. 建立分析模型

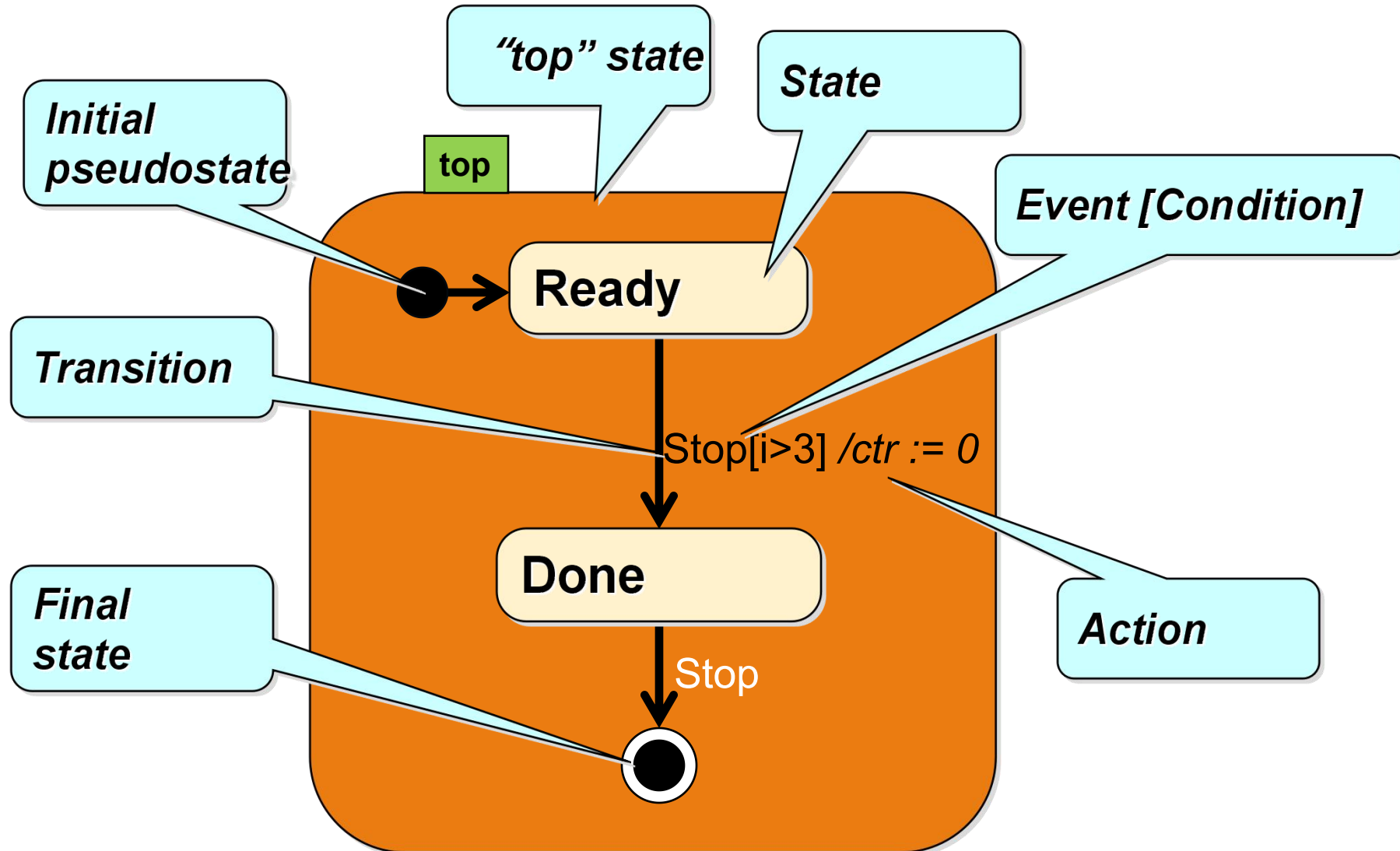


# 增加 Conceptual Class的属性

Professor
<ul style="list-style-type: none"><li>- name</li><li>- employeeID : UniqueId</li><li>- hireDate</li><li>- status</li><li>- discipline</li><li>- maxLoad</li></ul>
<ul style="list-style-type: none"><li>+ submitFinalGrade()</li><li>+ acceptCourseOffering()</li><li>+ setMaxLoad()</li><li>+ takeSabbatical()</li><li>+ teachClass()</li></ul>

识别出概念类的主要属性，可以有遗漏，  
在后续的分析与设计中，这些属性会逐渐补全。  
同时可以识别出概念类的部分操作。  
在概念模型中，类的属性比操作更为重要。

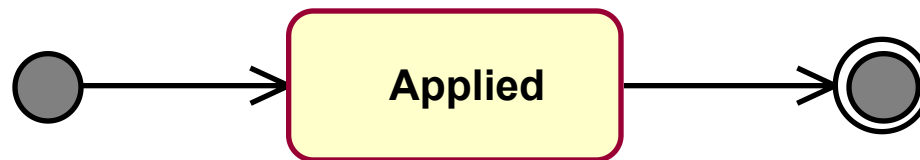
# What Are State Machine Diagrams?



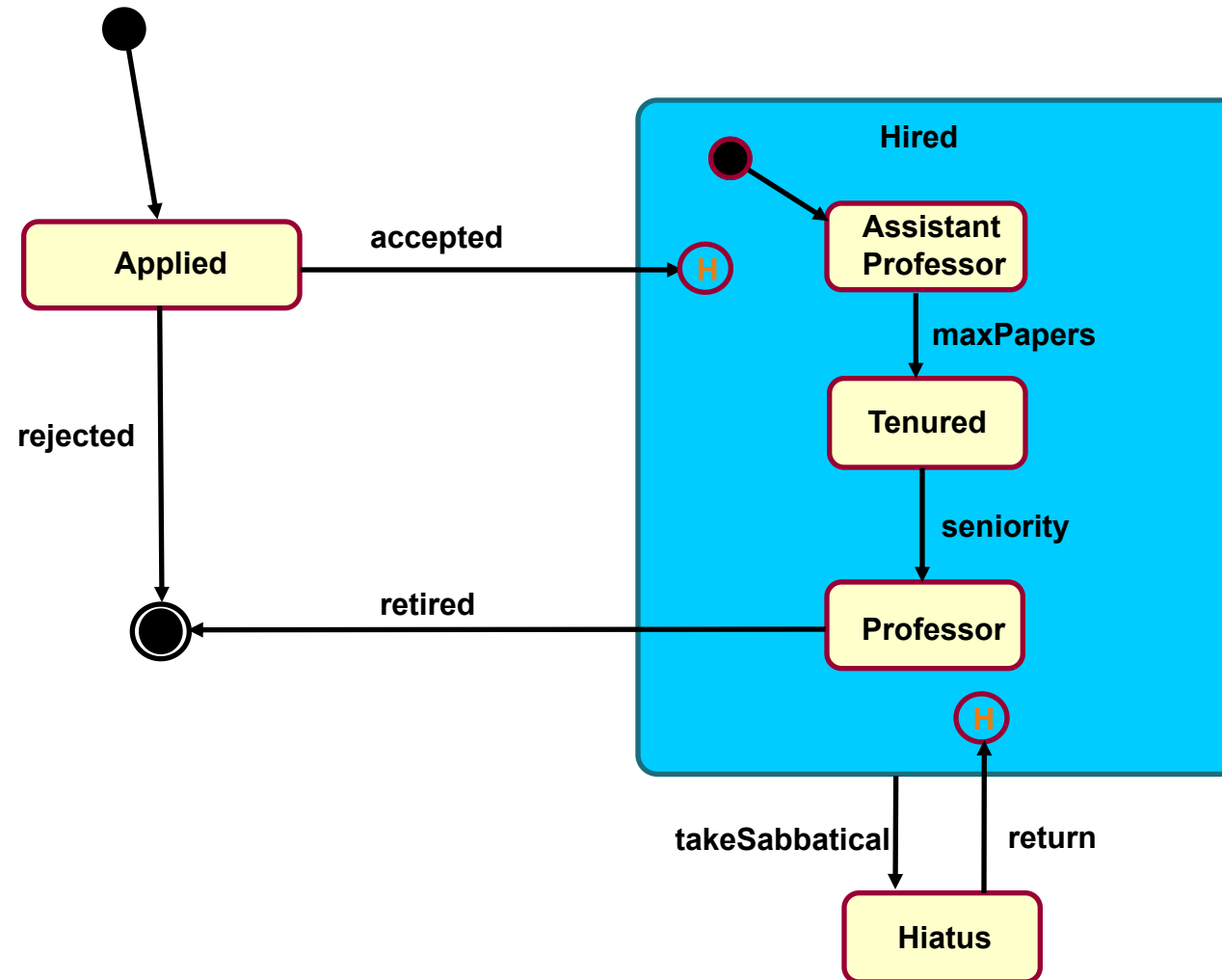
# Special States

---

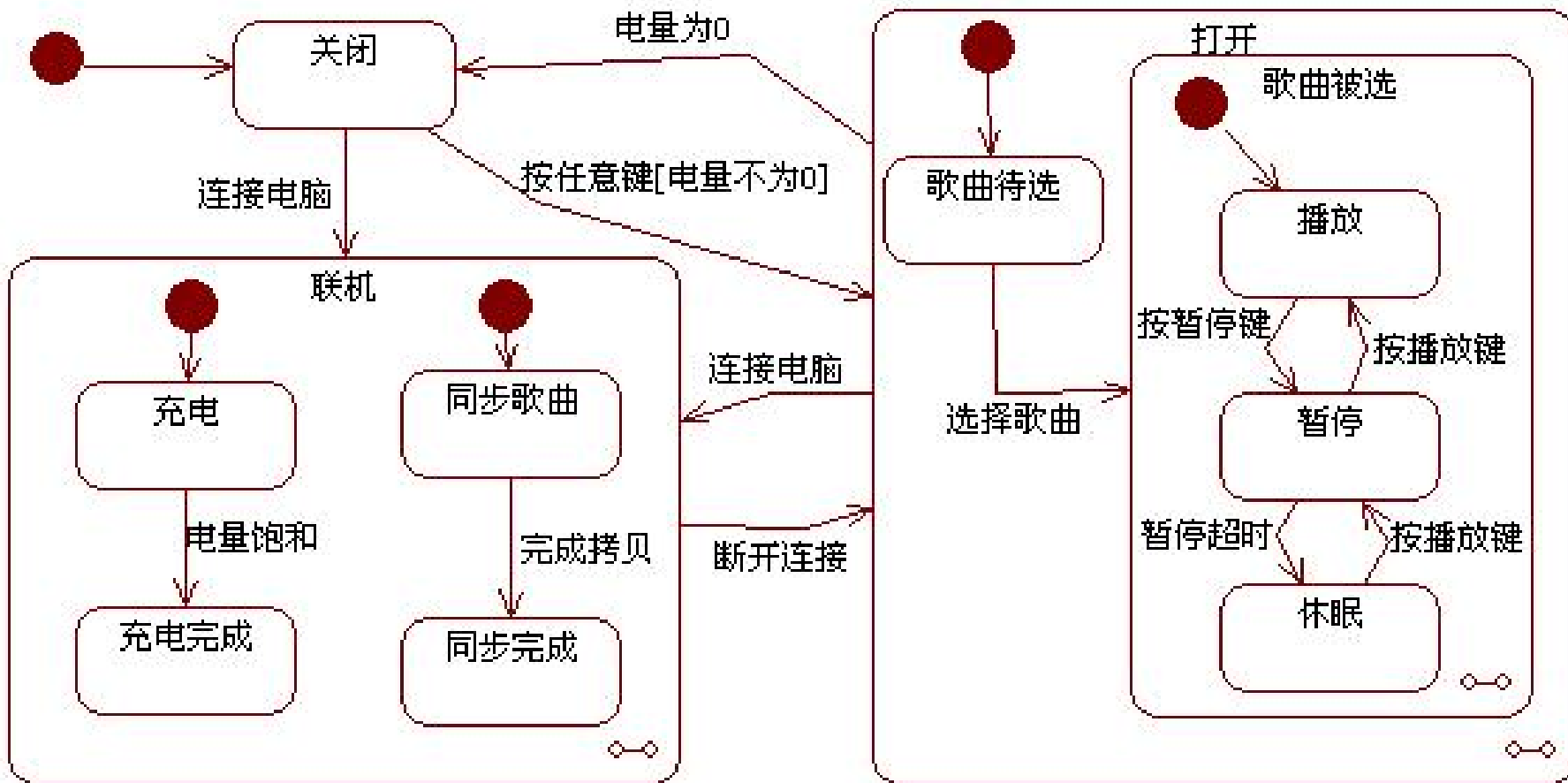
- The initial state is the state entered when an object is created.
  - An initial state is mandatory.
  - Only one initial state is permitted.
  - The initial state is represented as a solid circle.
- A final state indicates the end of life for an object.
  - A final state is optional.
  - A final state is indicated by a bull's eye.
  - More than one final state may exist.



# Example: State Machine

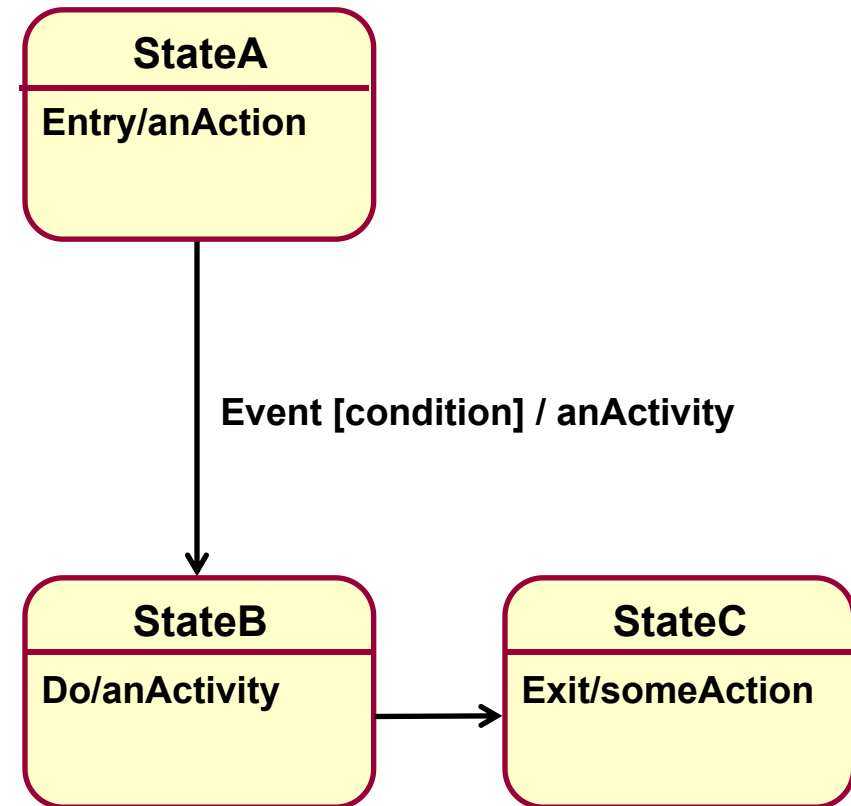


# 唱片播放器的状态图

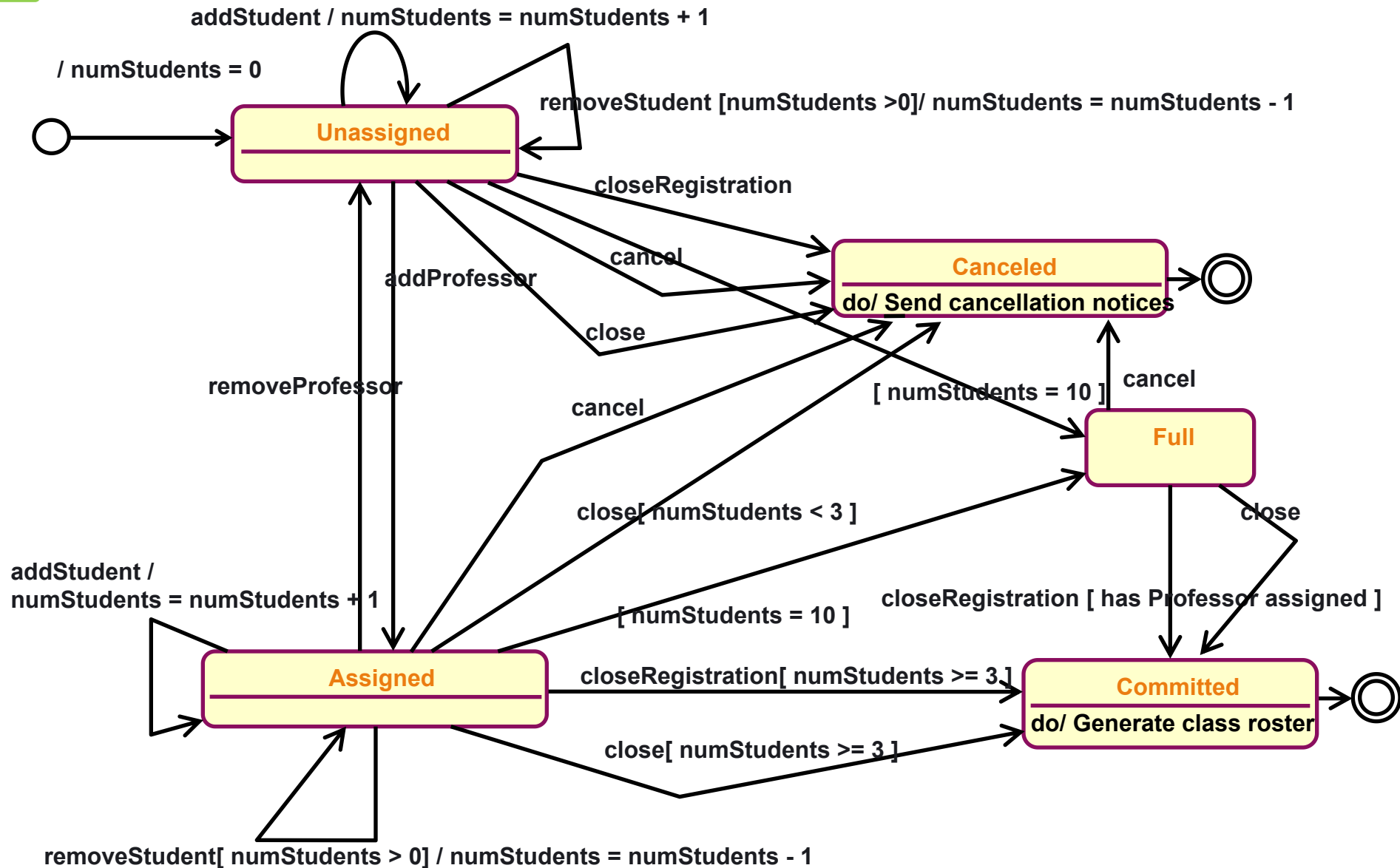


# Activities and Actions

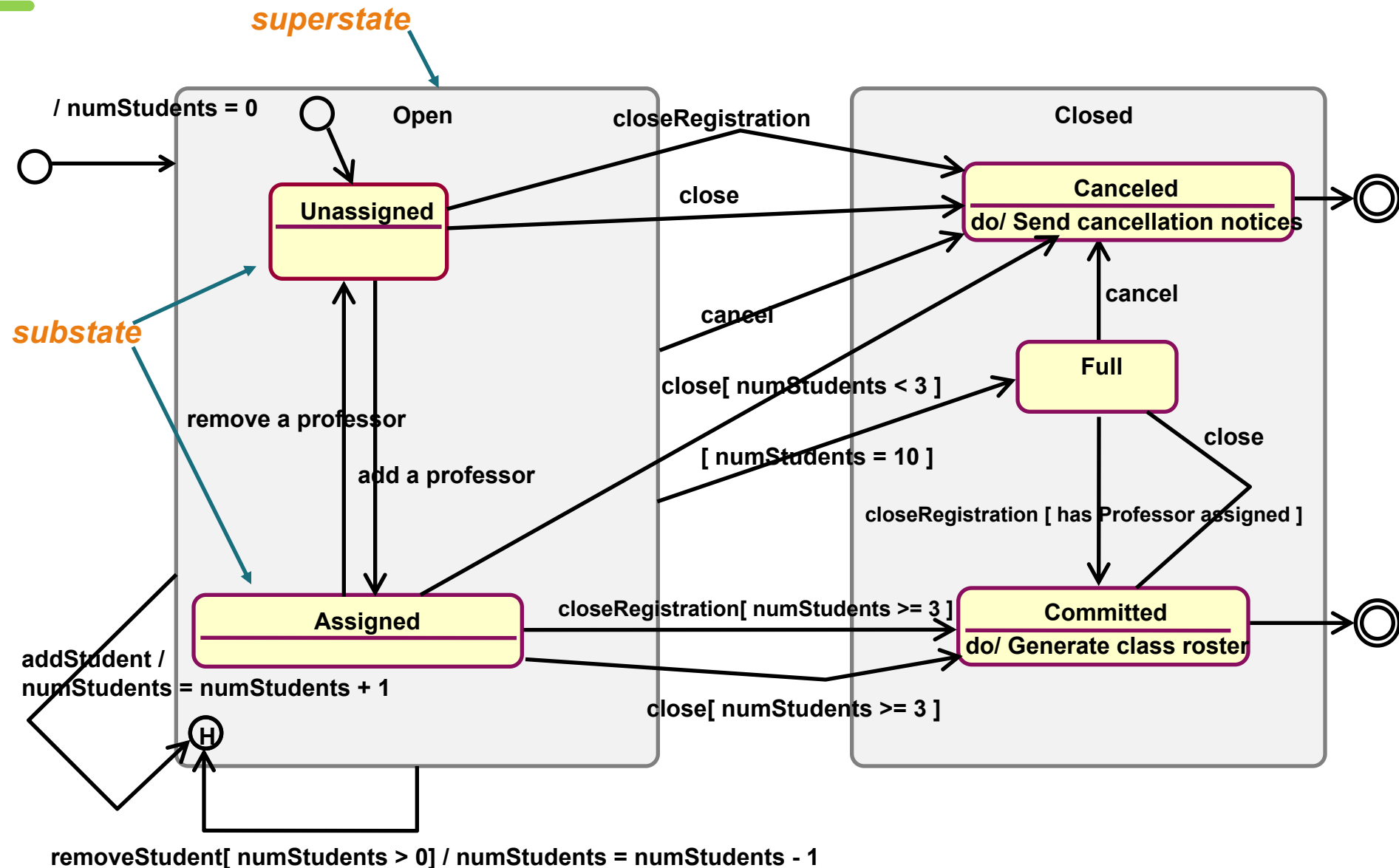
- Entry
  - Executed when the state is entered
- Do
  - Ongoing execution
- Exit
  - Executed when the state is exited
- Event [condition] / anActivity
  - Executed during transition



# Example: State Machine



# Example: State Machine with Nested States and History





# Which Objects Have Significant State?

---

- ❑ Objects whose role is clarified by state transitions
- ❑ Complex use cases that are state-controlled
- ❑ It is not necessary to model objects such as:
  - Objects with straightforward mapping to implementation
  - Objects that are not state-controlled
  - Objects with only one computational state

# 讨论

---

- “女儿” 和 “父亲” 是什么关系？
- “青蛙” 和 “蝌蚪” 是什么关系？（ Metamorphosis ）

# 面向对象分析的步骤



## 1. 建立用例模型

- 1.1 识别actor和use case, 画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构



## 2. 建立概念模型

- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性, 画状态图

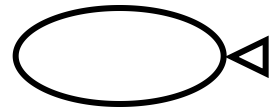


## 3. 建立分析模型

- 3.1 识别出用例实现**
- 3.2 针对每个用例实现:
  - 识别出分析类
  - 建立时序图, 生成通信图
  - 对照通信图建立类图, 完善每个分析类的属性和操作

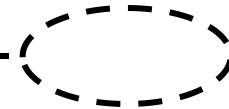
# What is a Use-Case Realization?

*Use-Case Model*

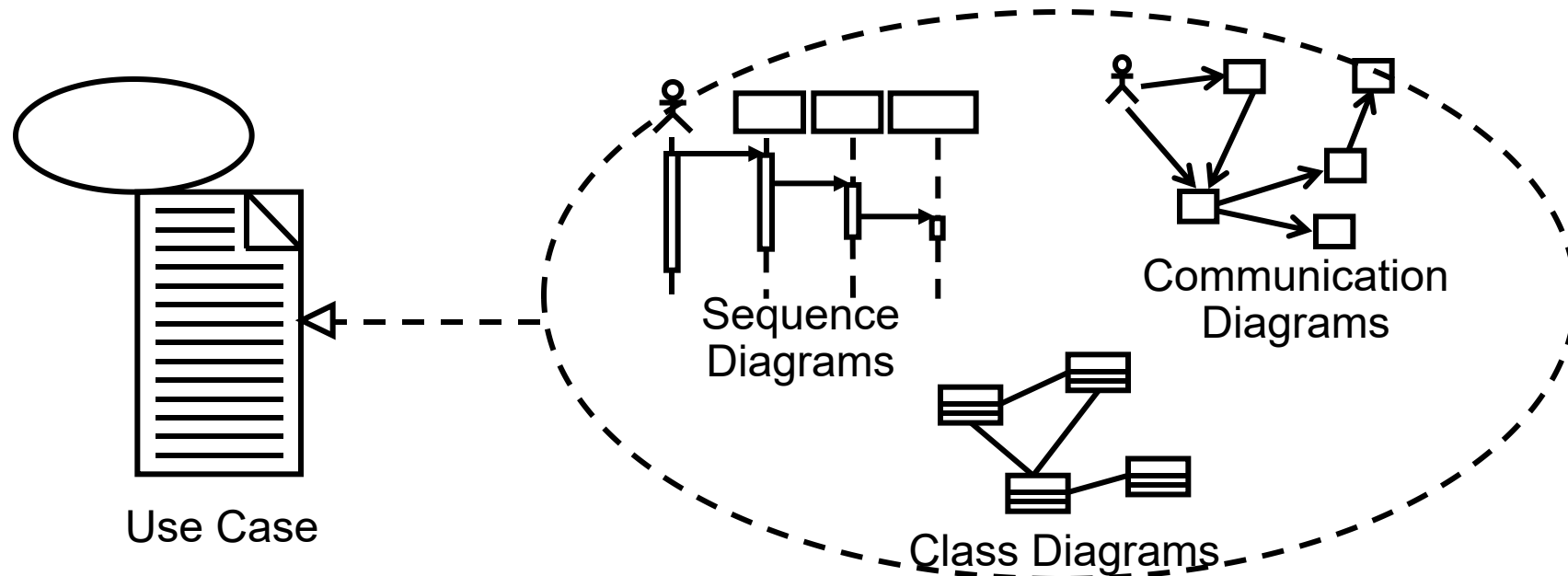


Use Case

*Analysis Model*



Use-Case Realization



# 面向对象分析的步骤



## 1. 建立用例模型

- 1.1 识别actor和use case, 画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构



## 2. 建立概念模型

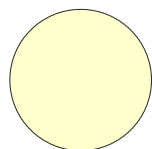
- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性, 画状态图



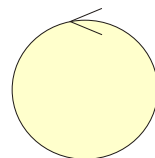
## 3. 建立分析模型

- 3.1 识别出用例实现
- 3.2 针对每个用例实现:
  - 识别出分析类
  - 建立时序图, 生成通信图
  - 对照通信图建立类图, 完善每个分析类的属性和操作

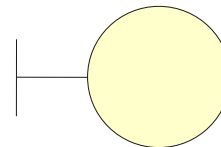
# 三种Analysis Class



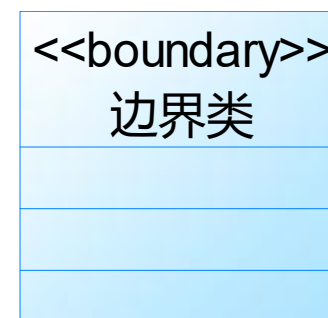
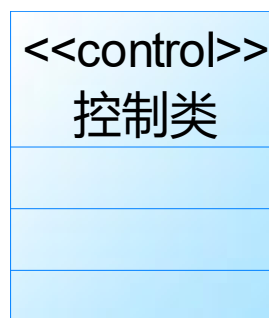
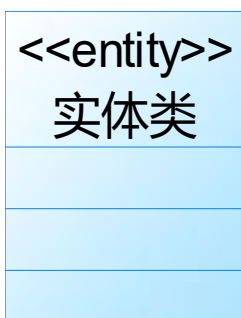
实体类



控制类



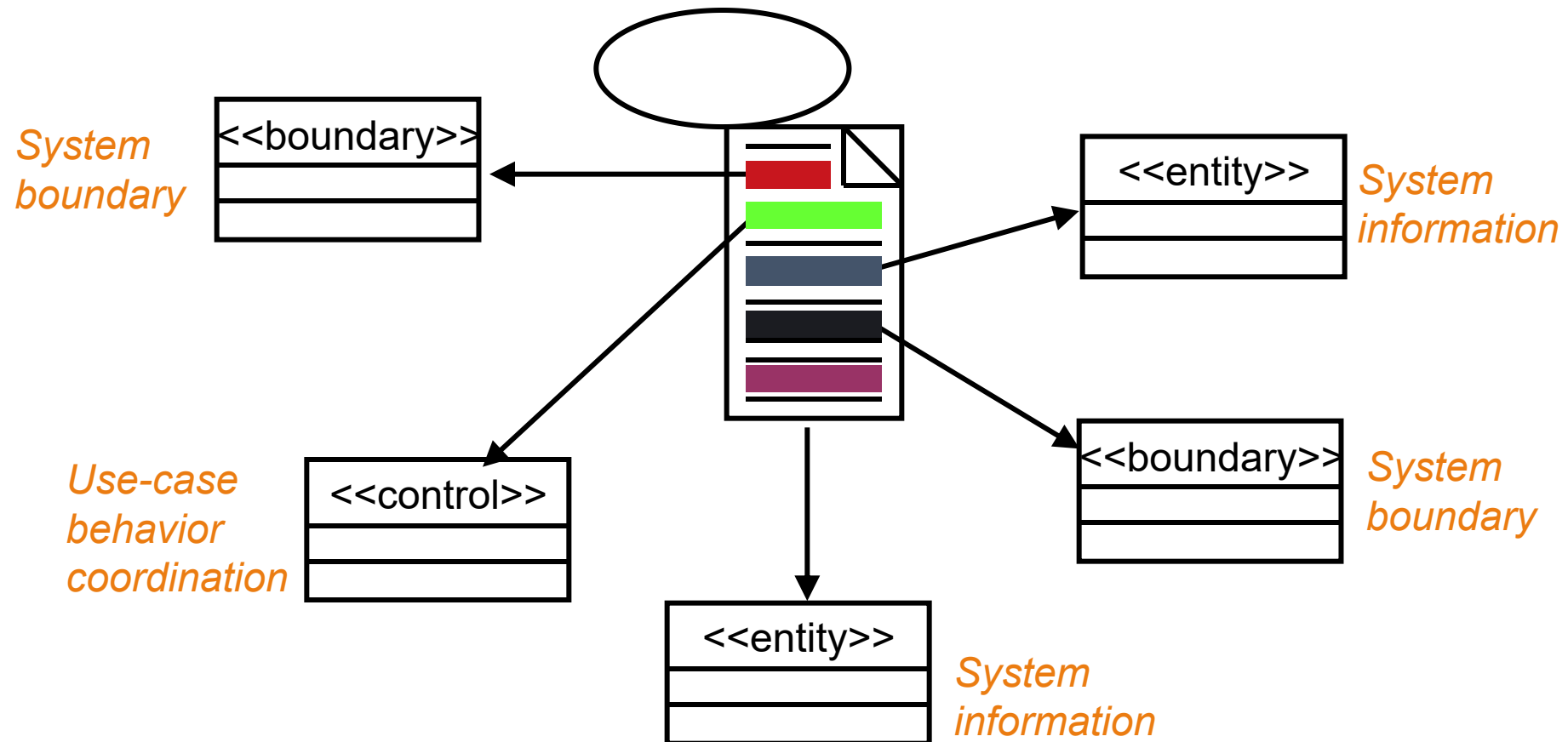
边界类



- 不同的衍型(stereotype)可采用不同的图标，也可用采用 “《》” 来区别。

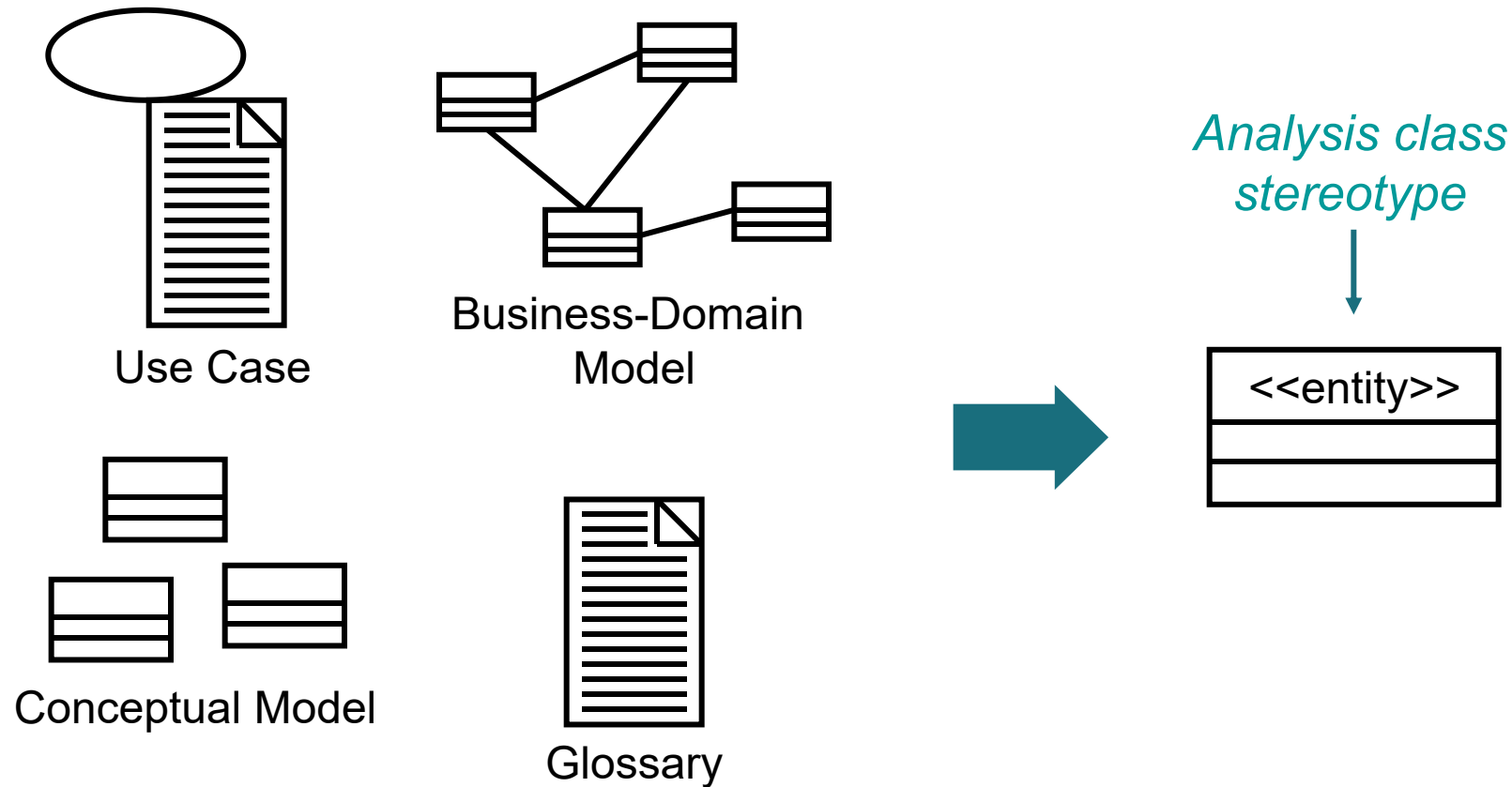
# Find Analysis Classes

- The complete behavior of a use case has to be distributed to analysis classes



# What Is an Entity Class?

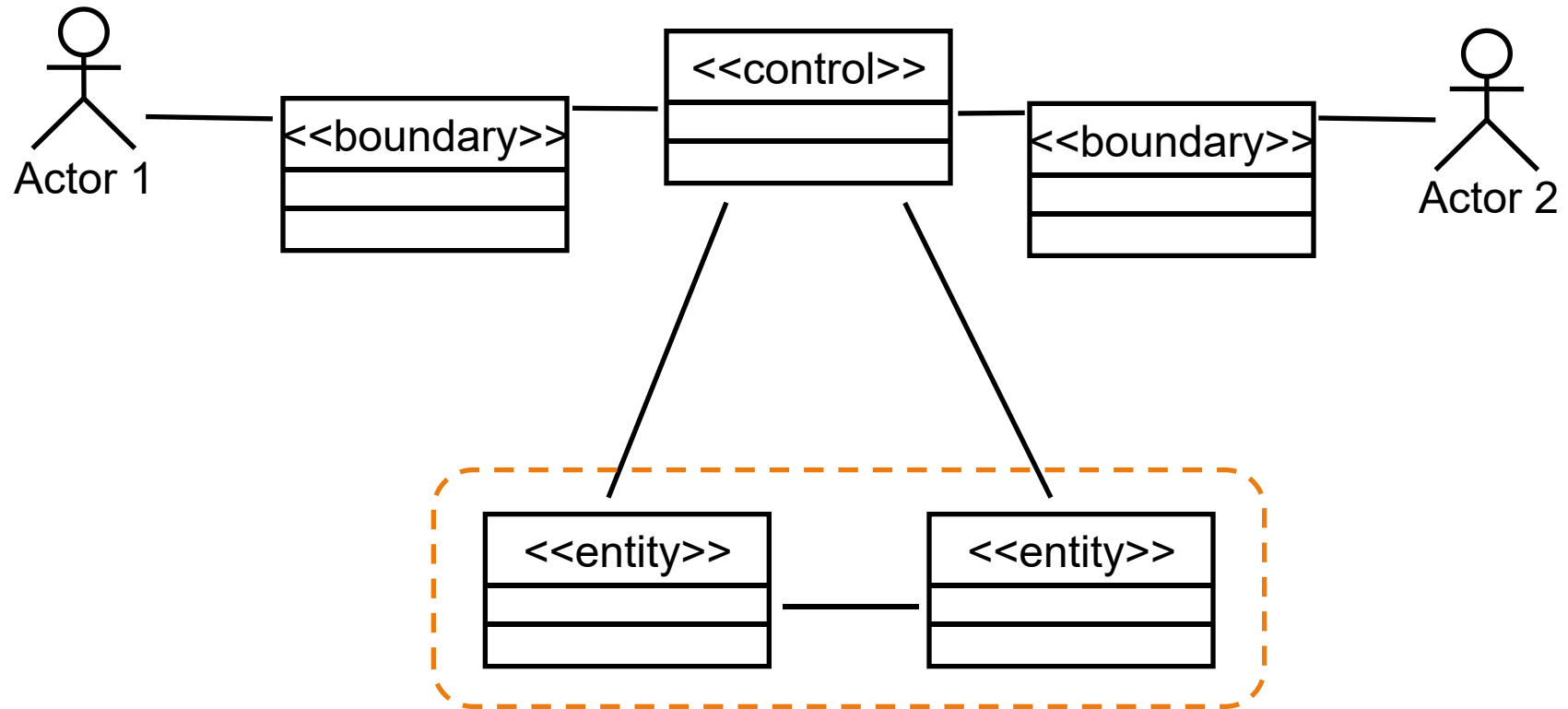
- Key abstractions of the system



Environment independent.



# The Role of an Entity Class

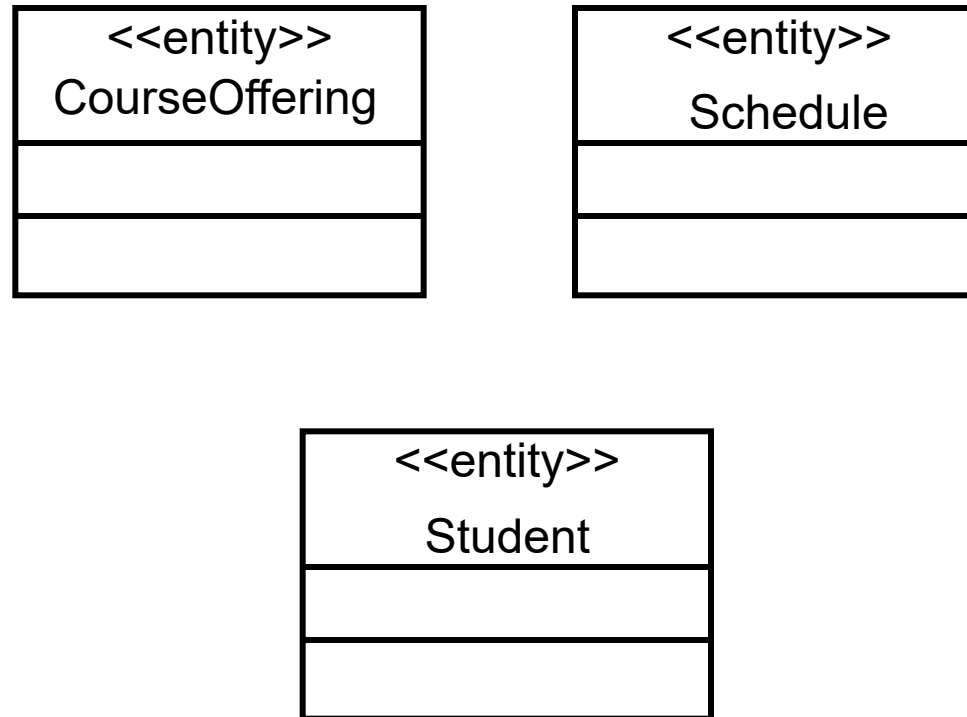


Store and manage information in the system.

# Example: Candidate Entity Classes

---

- Register for Courses (Create Schedule)



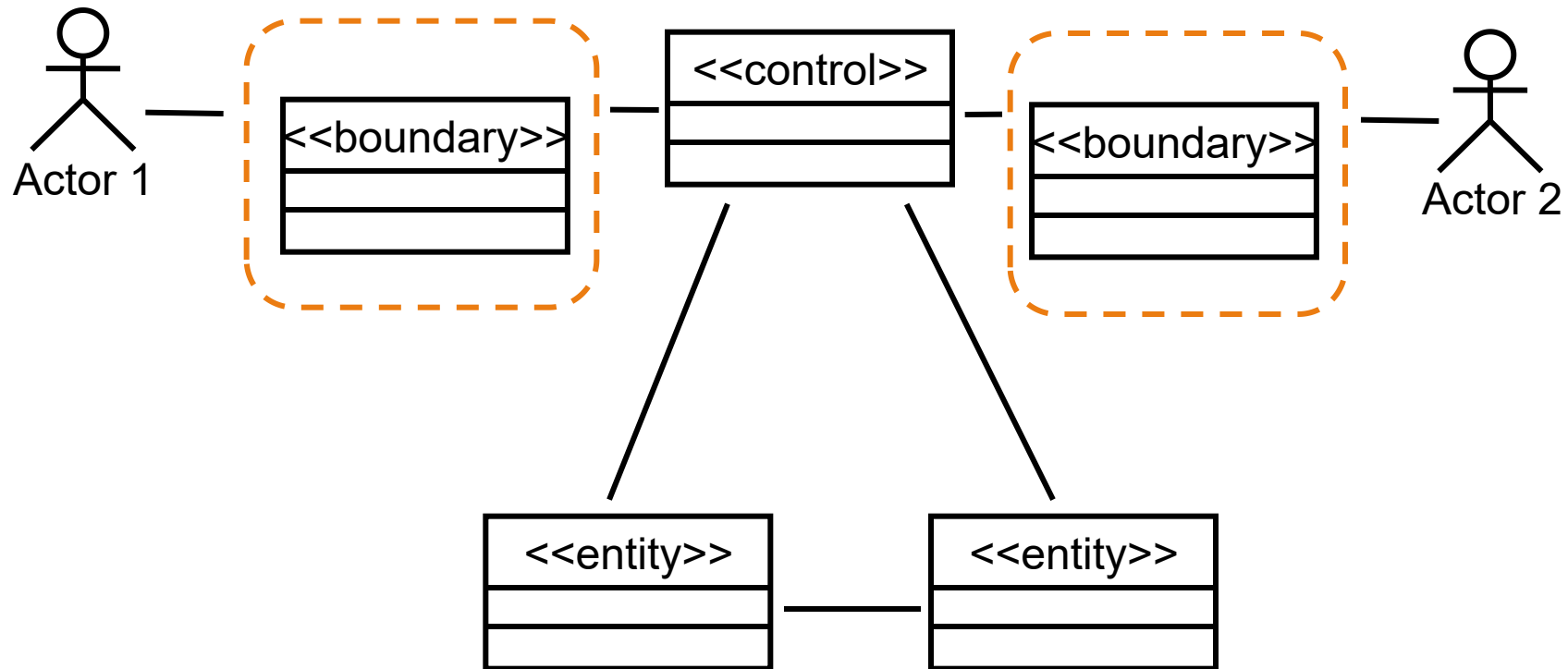
# What Is a Boundary Class?

- ❑ Intermediates between the interface and something outside the system
- ❑ Several Types
  - User interface classes
  - System interface classes
  - Device interface classes
- ❑ *One boundary class per actor/use-case pair*



Environment dependent.

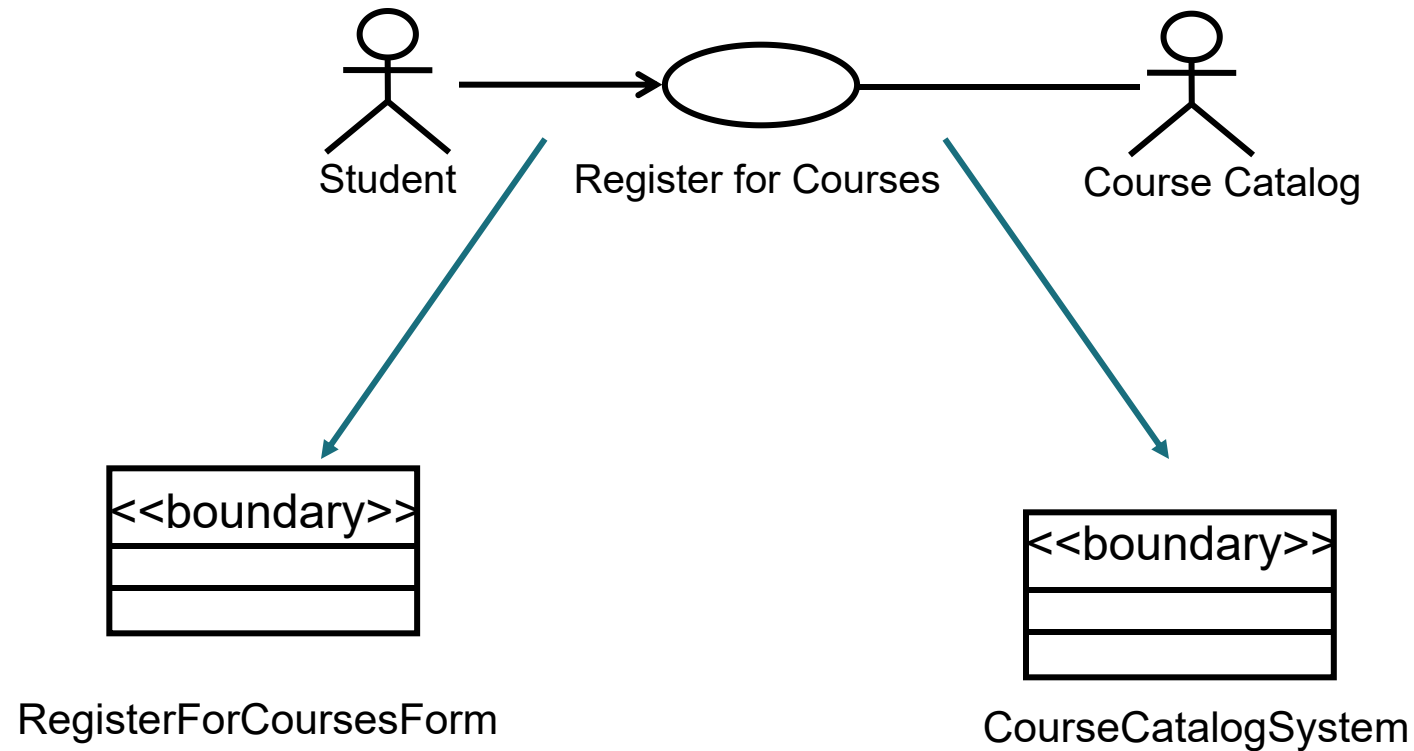
# The Role of a Boundary Class



Model interaction between the system and its environment.

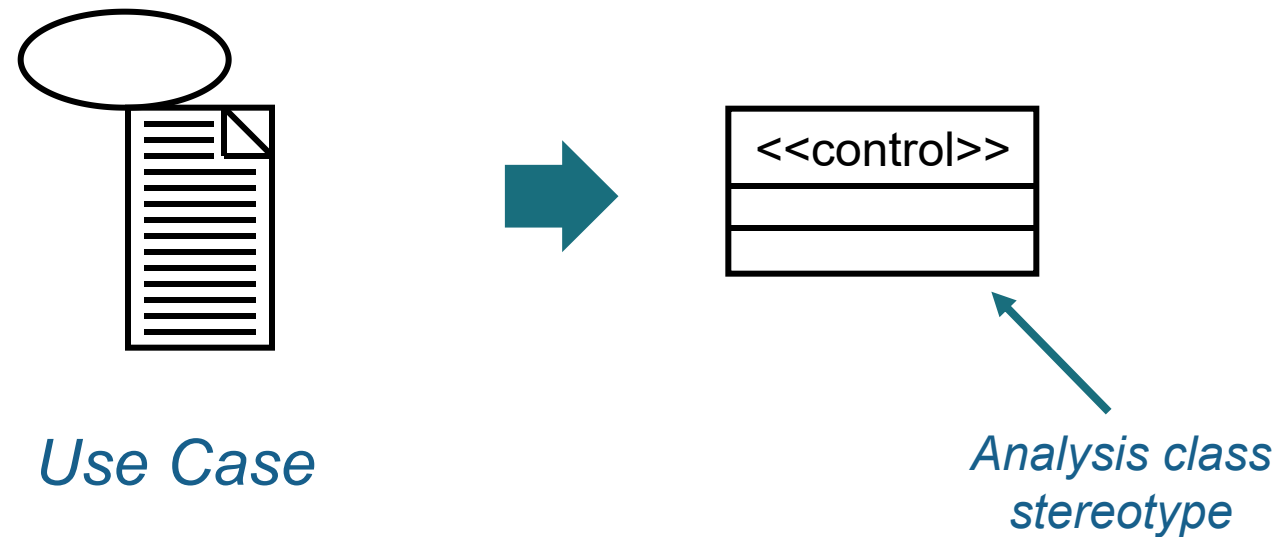
# Example: Finding Boundary Classes

- One boundary class per actor/use case pair



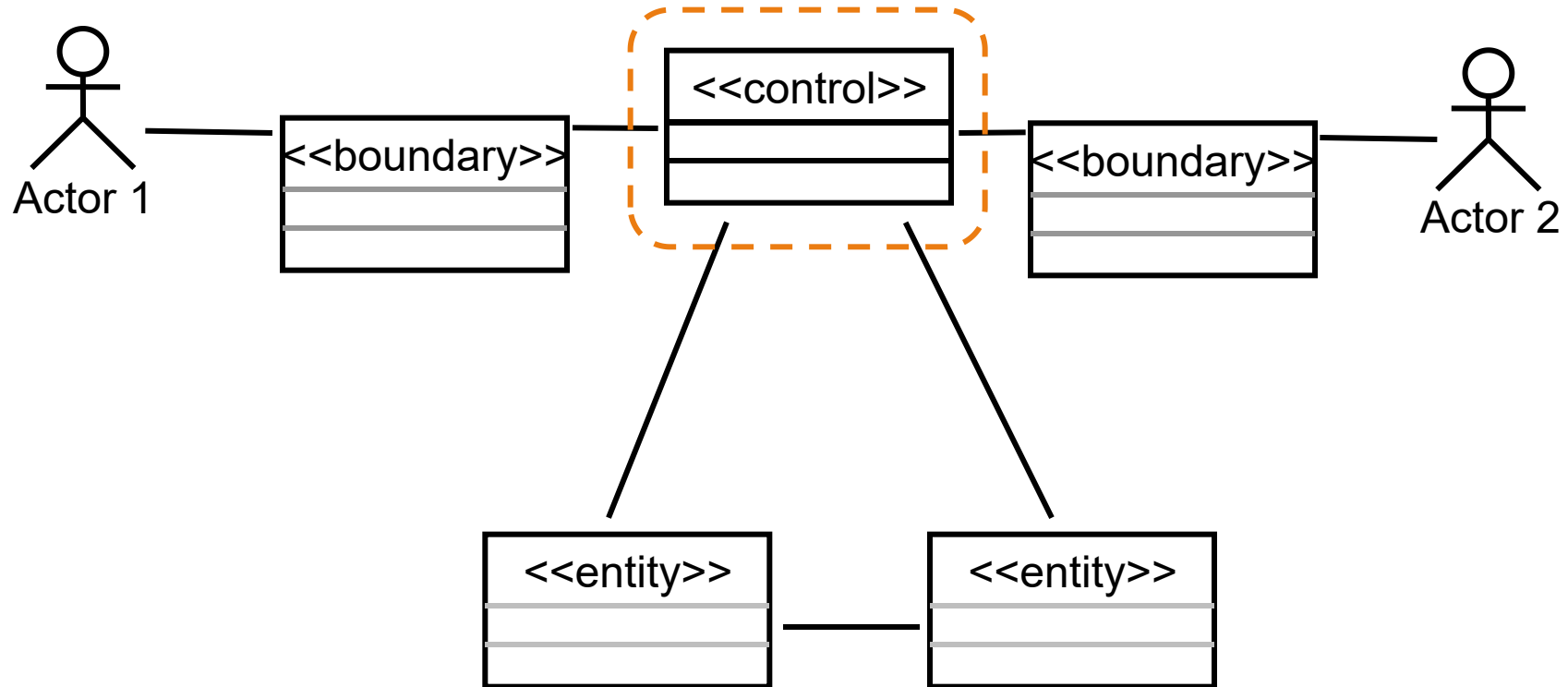
# What Is a Control Class?

- Use-case behavior coordinator
  - More complex use cases generally require one or more control cases



Use-case dependent. Environment independent.

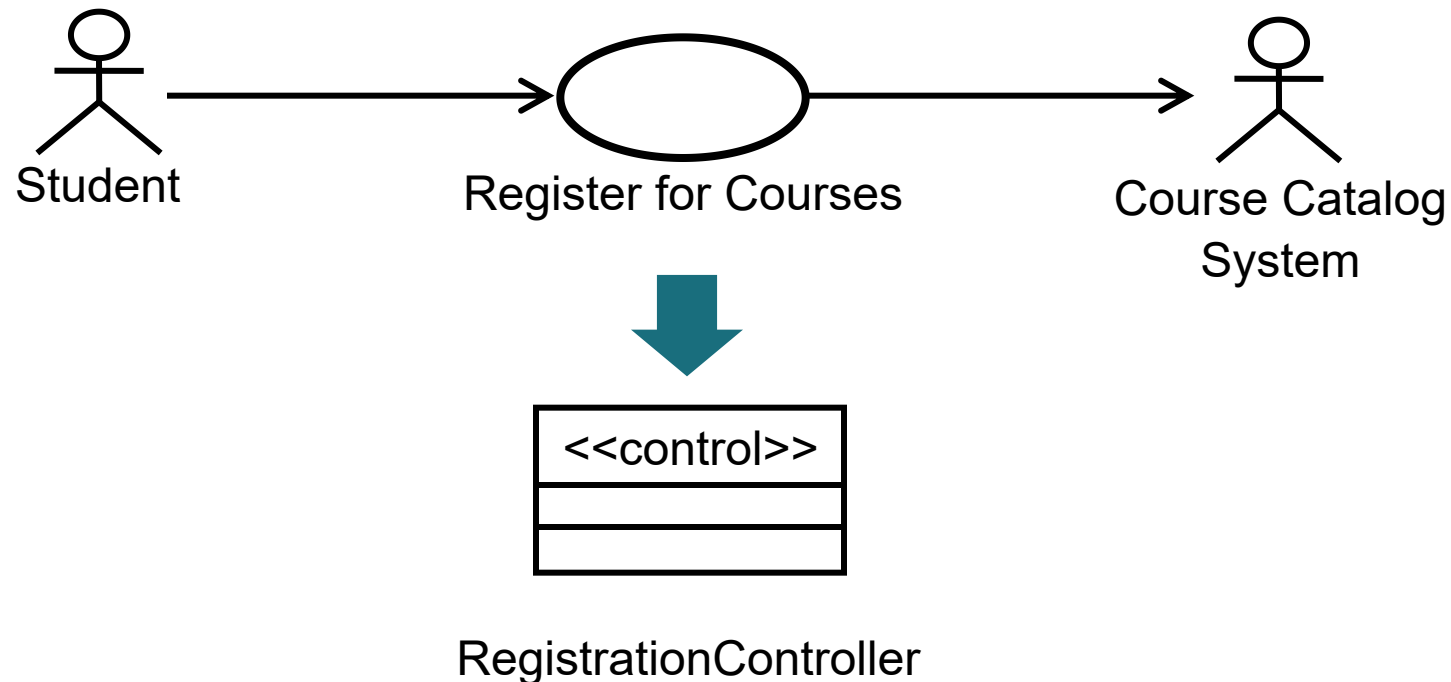
# The Role of a Control Class



Coordinate the use-case behavior.

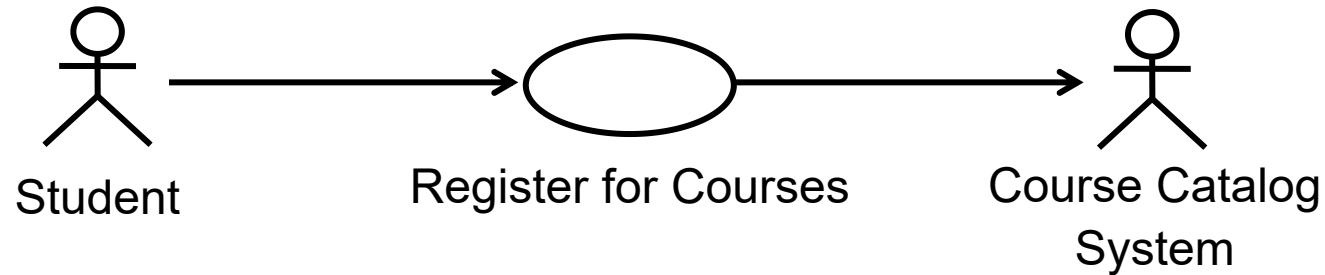
# Example: Finding Control Classes

- In general, identify one control class per use case.
  - As analysis continues, a complex use case's control class may evolve into more than one class



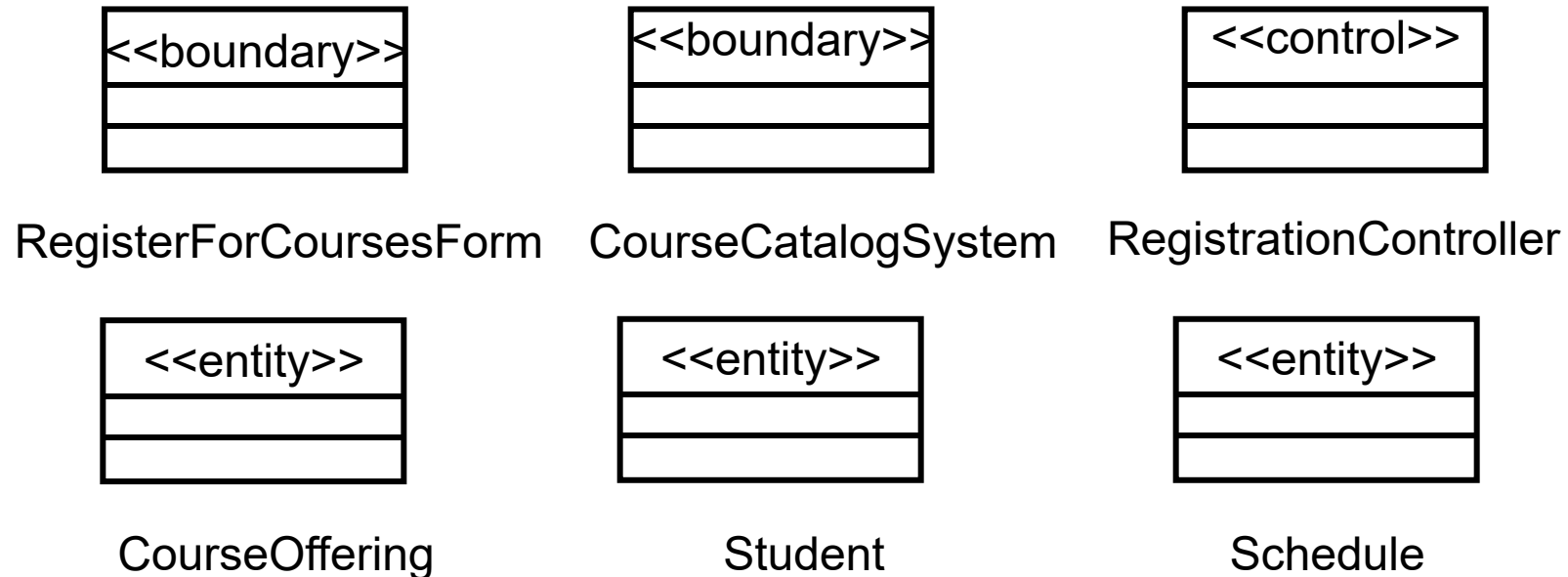


# Example: Summary: Analysis Classes



Use-Case Model

Analysis Model



# 面向对象分析的步骤



## 1. 建立用例模型

- 1.1 识别actor和use case, 画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构



## 2. 建立概念模型

- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性, 画状态图



## 3. 建立分析模型

- 3.1 识别出用例实现
- 3.2 针对每个用例实现:
  - 识别出分析类
  - **建立时序图, 生成通信图**
  - 对照通信图建立类图, 完善每个分析类的属性和操作

# Objects Need to Collaborate

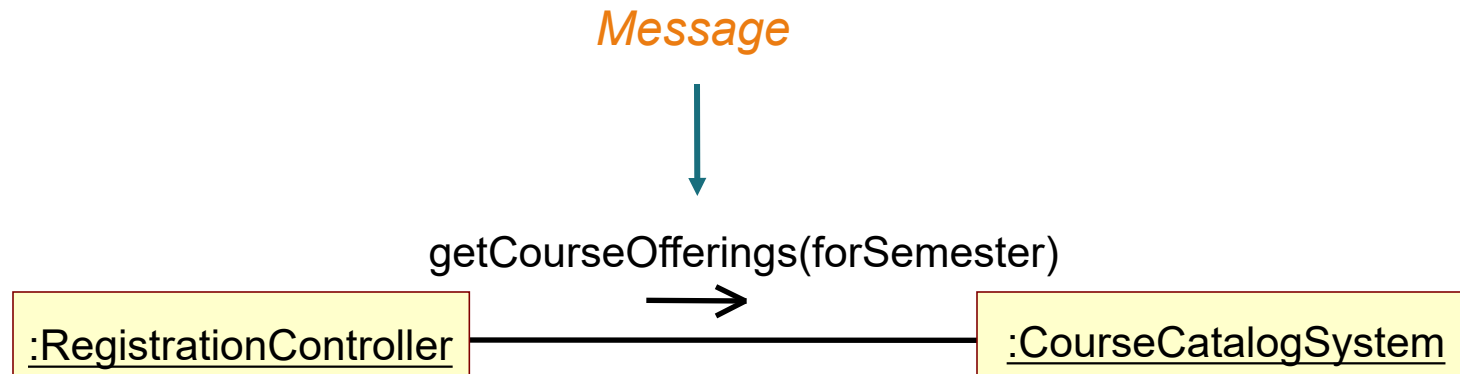
---

- ❑ Objects are useless unless they can collaborate to solve a problem.
  - Each object is responsible for its own behavior and status.
  - No one object can carry out every responsibility on its own.
- ❑ How do objects interact with each other?
  - They interact through messages.

# Objects Interact with Messages

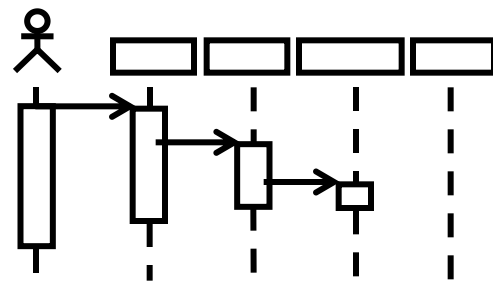
---

- A message shows how one object asks another object to perform some activity.

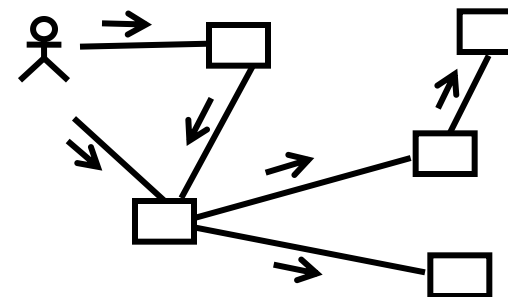


# What is an Interaction Diagram?

- ❑ Generic term that applies to several diagrams that emphasize object interactions
  - Sequence Diagram
    - Time oriented view of object interaction
  - Communication Diagram
    - Structural view of messaging objects



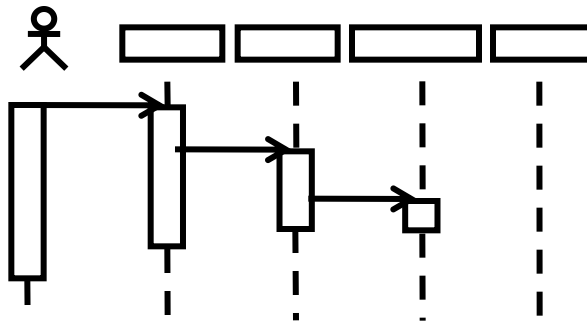
Sequence Diagrams



Communication Diagrams

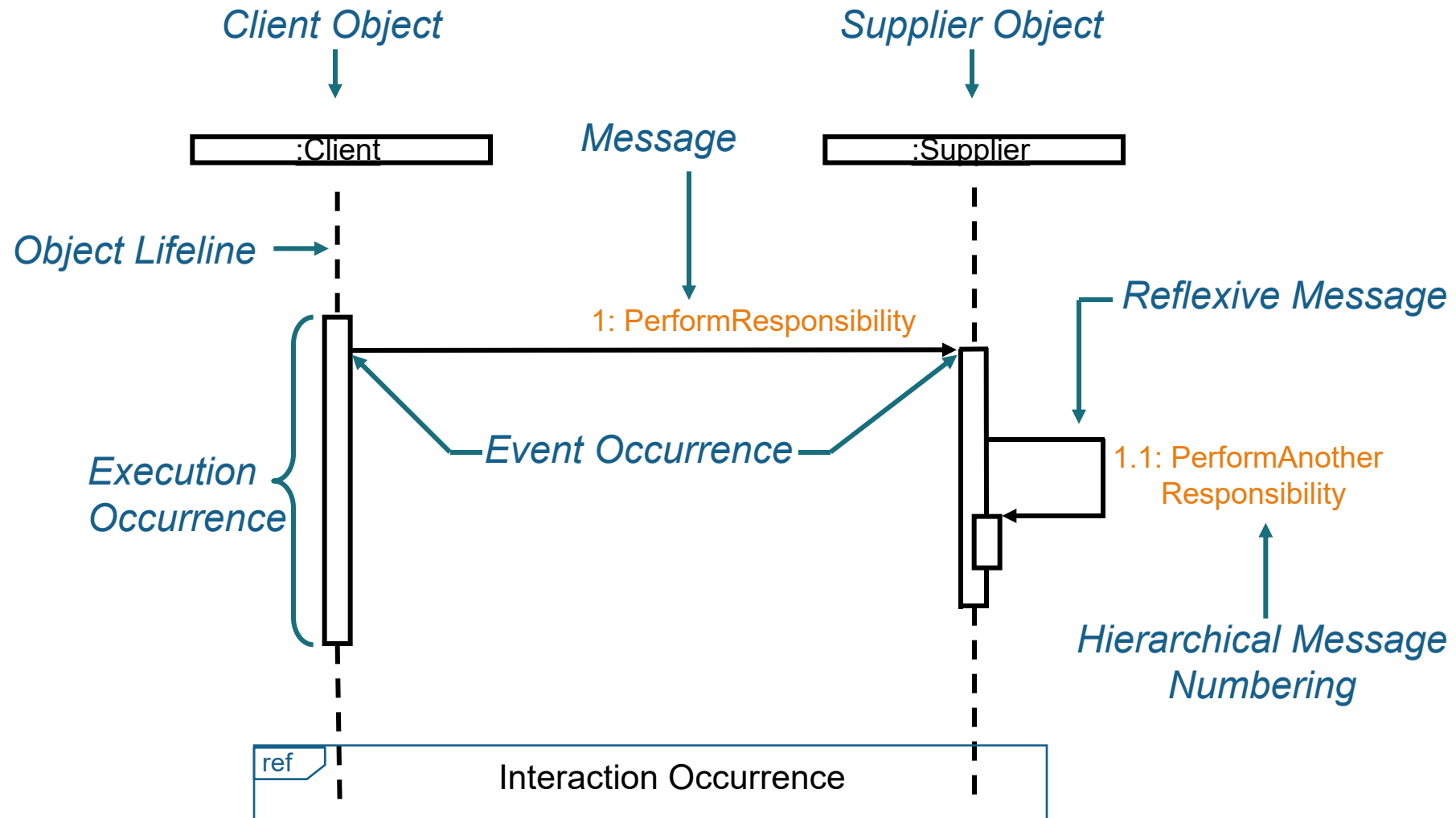
# Create Sequence Diagrams

- ❑ A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.
- ❑ The diagram shows:
  - The objects participating in the interaction.
  - The sequence of messages exchanged.

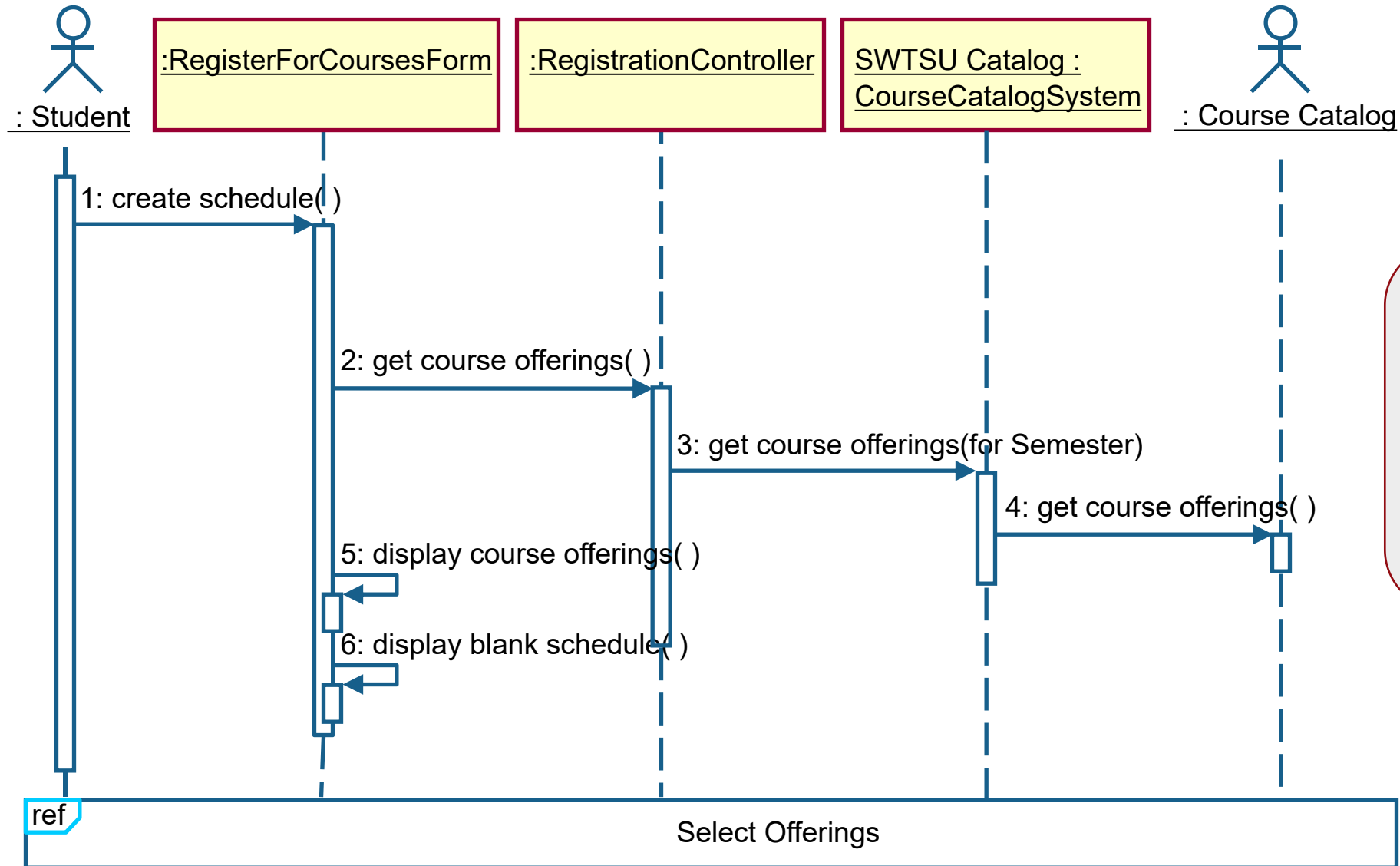


Sequence Diagram

# The Anatomy of Sequence Diagrams



# Example: Sequence Diagram



**先分工，再协作！**

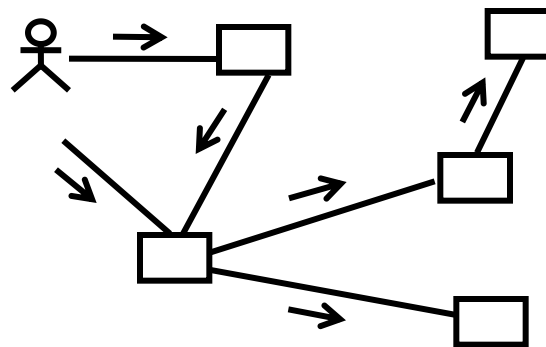
职责的分配：  
自己干自己的能干的事  
职责的内聚：  
自己只干自己的事



# Generate Communication Diagrams

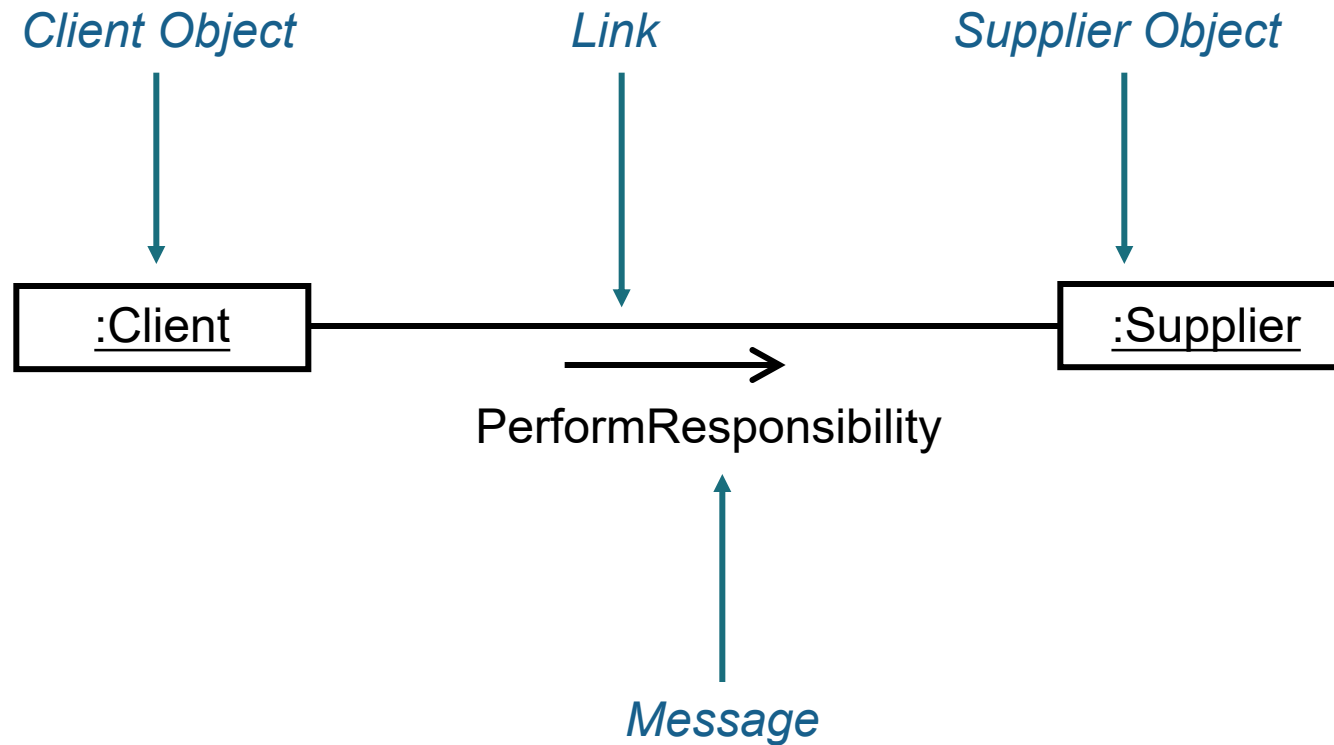
---

- A communication diagram emphasizes the organization of the objects that participate in an interaction.
- The communication diagram shows:
  - The objects participating in the interaction.
  - Links between the objects.
  - Messages passed between the objects.

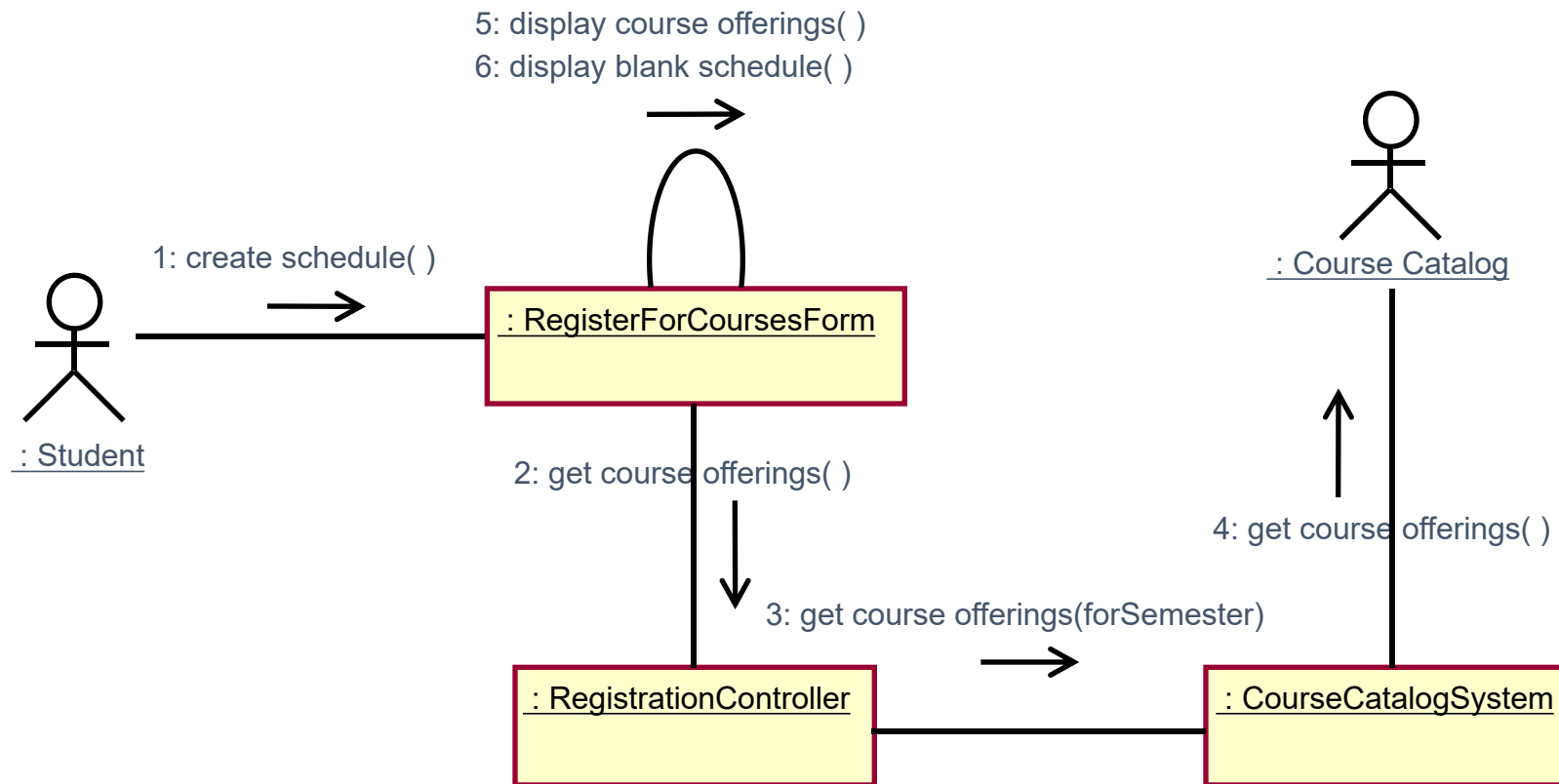


Communication Diagrams

# The Anatomy of Communication Diagrams



# Example: Communication Diagram



# 面向对象分析的步骤



## 1. 建立用例模型

- 1.1 识别actor和use case, 画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构



## 2. 建立概念模型

- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性, 画状态图



## 3. 建立分析模型

- 3.1 识别出用例实现
- 3.2 针对每个用例实现:
  - 识别出分析类
  - 建立时序图, 生成通信图
  - **对照通信图建立类图, 完善每个分析类的属性和操作**

# Finding Relationships and Operations

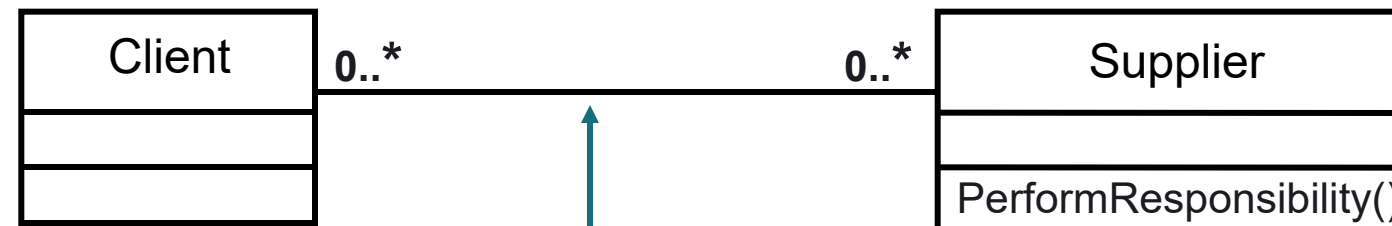
## Communication Diagram



*Link*



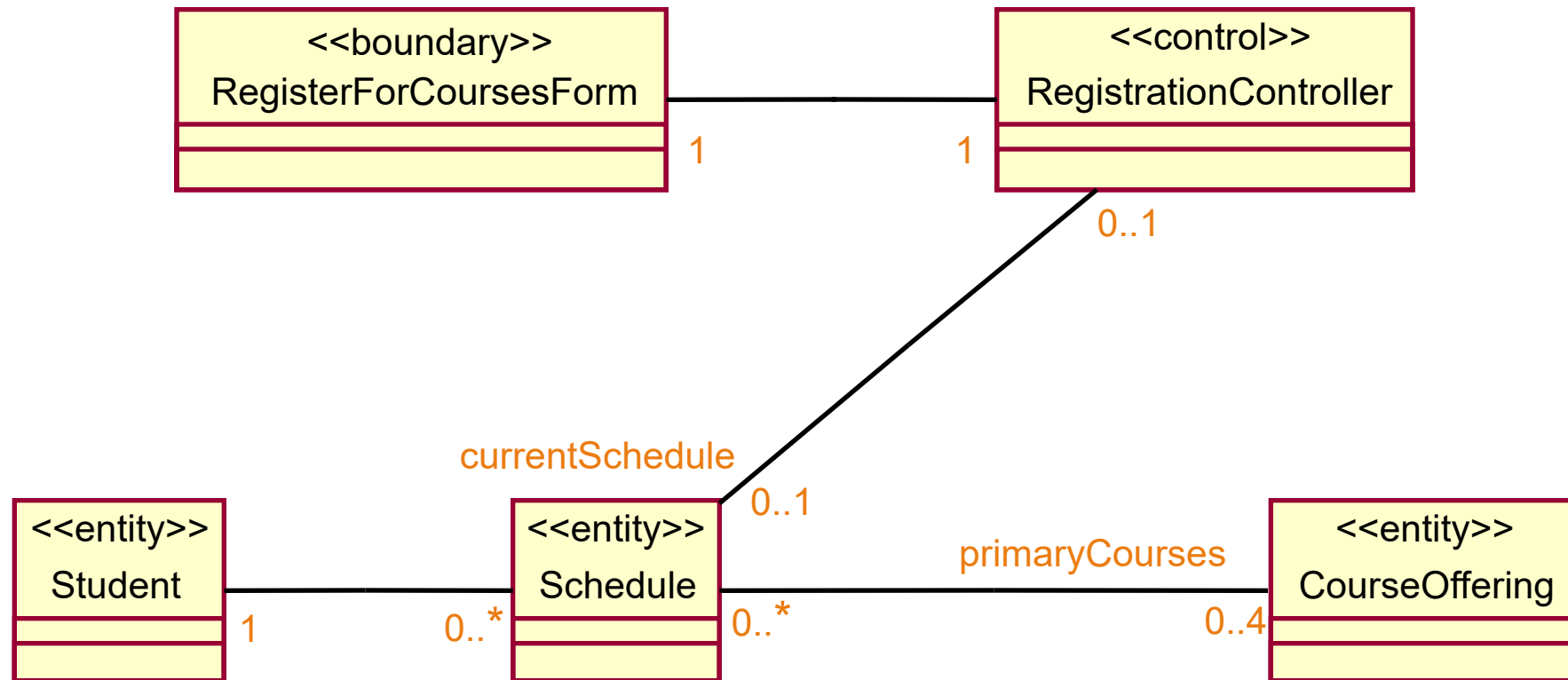
## Class Diagram



*Association*

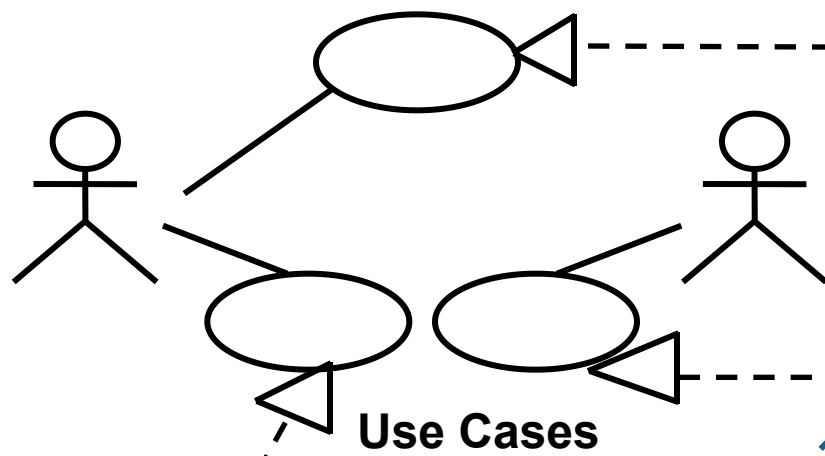
Relationship for every link!

# Example: VOPC: Finding Relationships



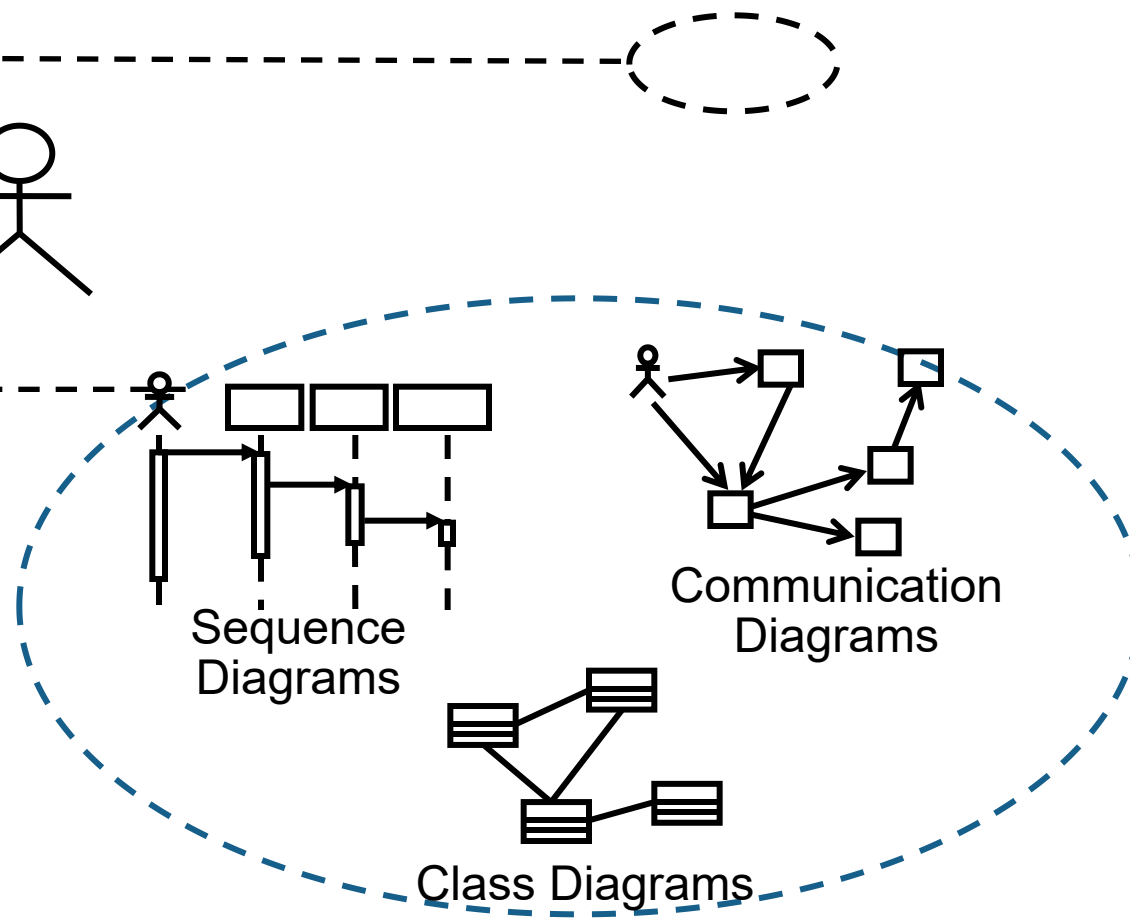
# 完成用例分析

## Use-Case Model



Use-Case realization

## Use-Case realization



# 大纲



中南大學  
CENTRAL SOUTH UNIVERSITY

## 第三章 软件需求

01-软件需求概述

02-需求获取

03-需求分析和建模

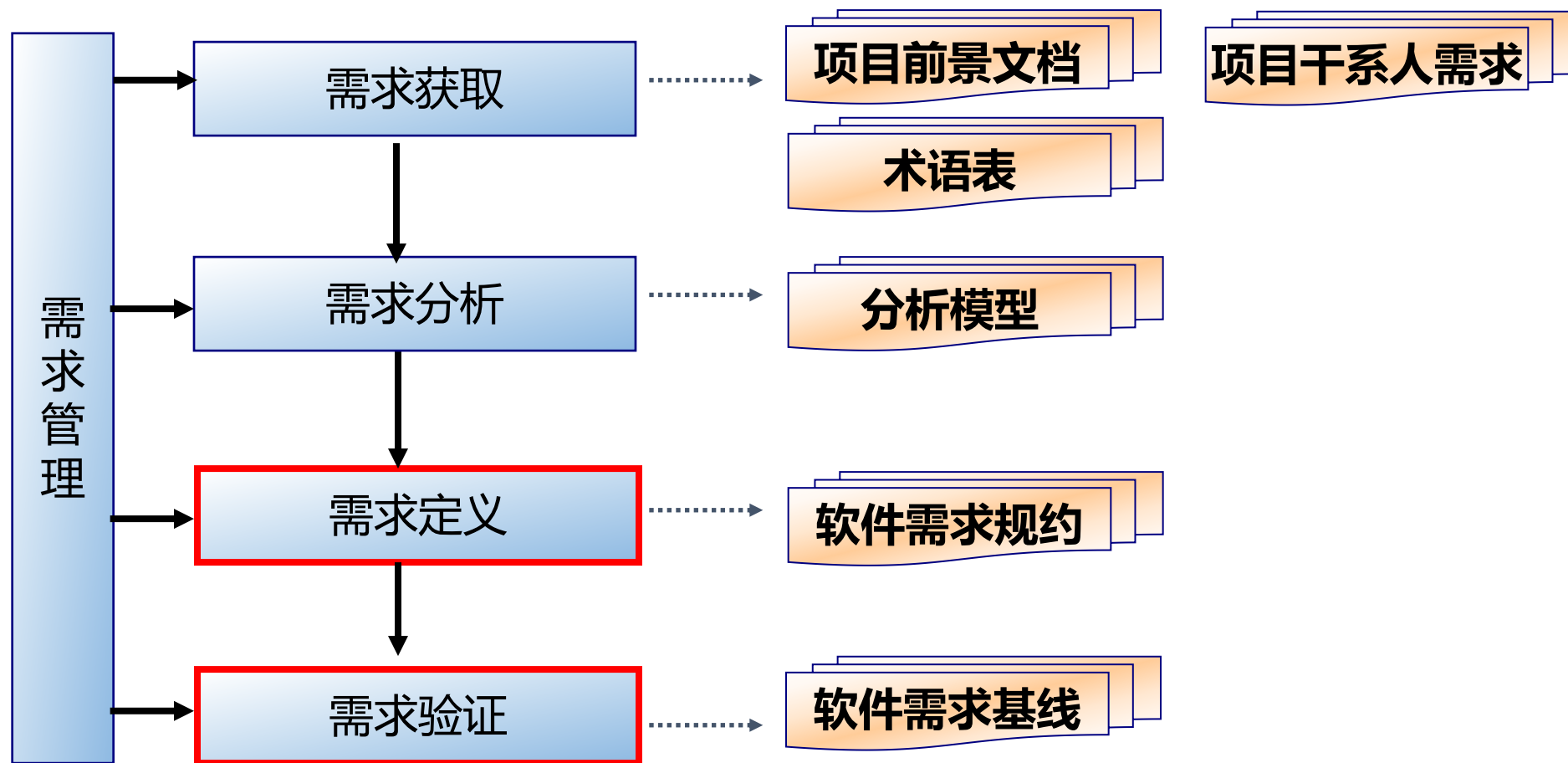
☀ 04-需求定义和验证

05-需求管理

@第4章.教材



# 需求工程



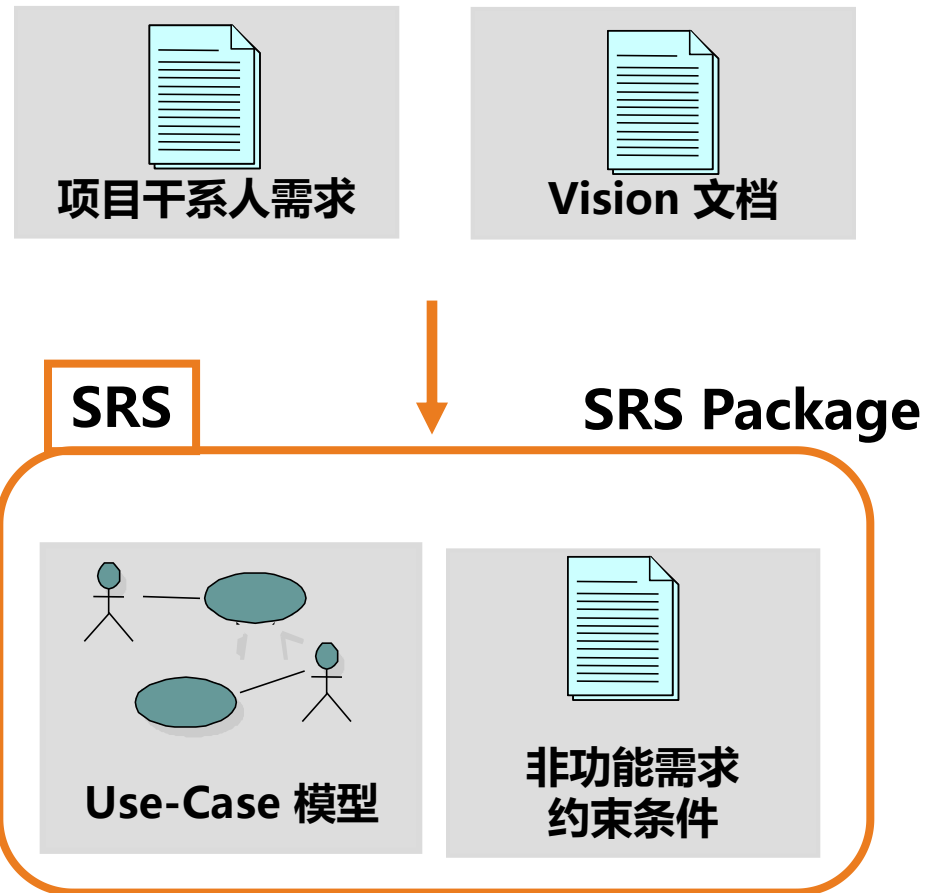
发现、获取、组织、分析、编写和管理需求的系统方法，让客户和项目组之间达成共识。

# 需求定义的任务

---

- 定义详细需求- SRS
  - 细化功能需求
  - 细化非功能需求
  - 细化约束条件
  - 设计用户界面和接口

# 成果：软件需求规约



软件需求规约 (SRS) 定义了系统的外在行为和属性。

# 细化需求

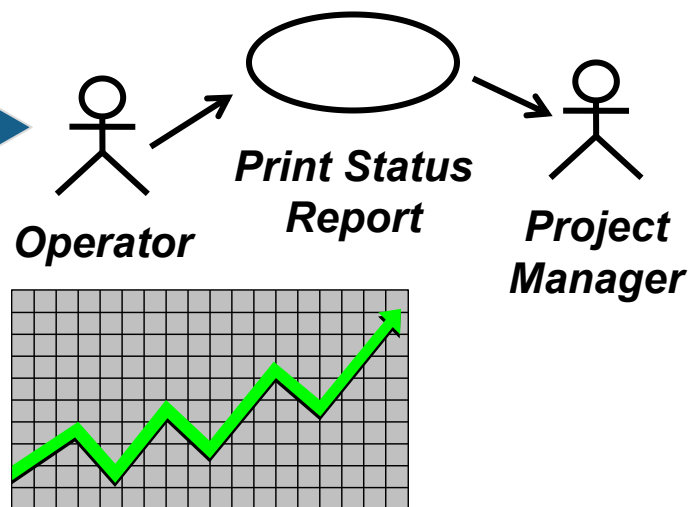
- 细化功能需求
  - 引入分析模型
- 细化非功能需求
- 细化约束条件

## 概要功能需求

*Feat 63 - the defect tracking system will provide trending information to help the project manager assess project status*

## 详细功能需求

Trending information will be charted with a line graph showing time on the x axis, and number of defects found on the y axis.



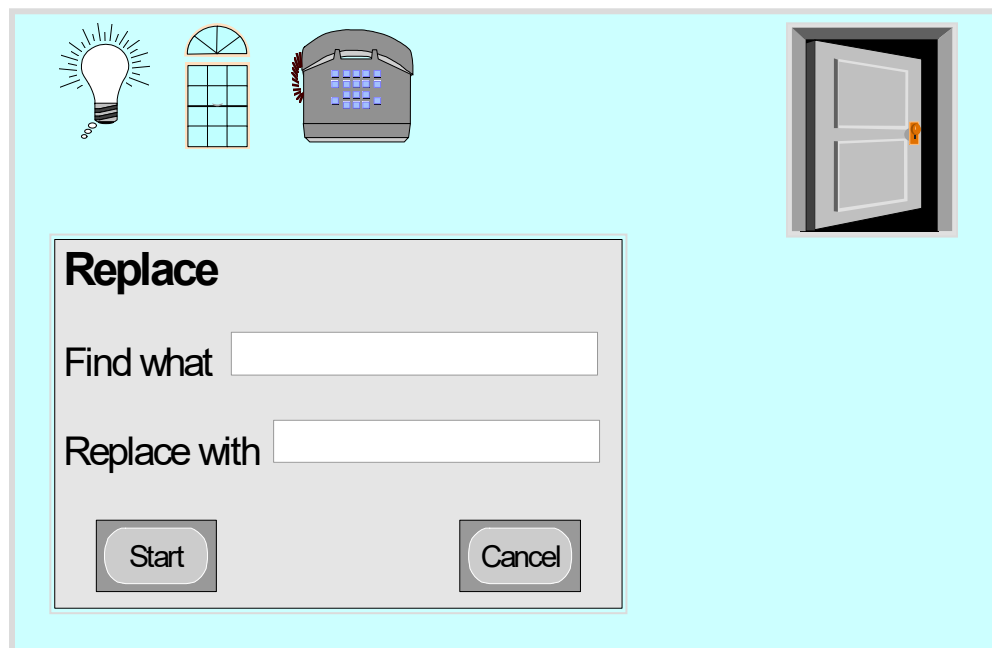
# 设计用户界面和接口

---

- 如果和外界的软件系统或硬件进行交互，则需定义通讯协议
  - 如果采用现有的协议，在功能说明中描述
  - 如果采用新的协议，在设计时进行完整描述
- 如果和用户交互，则需简要设计用户界面
  - 图纸（在纸上手绘）
  - 位图（采用绘图工具，如Visio）
  - 可执行代码，即交互式的电子界面原型
    - 采用界面原型工具，如Axure (<https://www.axure.com>)、Sketch、Zeplin
    - 或直接编程，如采用HTML或编程语言

# 如何描述用户界面

- 可在功能说明中纳入屏幕框图，也可以是一个独立的用户界面原型
- 注意不要关注太多的界面设计细节



# SRS模板

## □ 1. 简介

- 1.1 目的
- 1.2 范围
- 1.3 定义、首字母缩写词和缩略语
- 1.4 参考资料
- 1.5 概述

## □ 2. 整体说明

- 2.1 语境图 (context diagram)
- 2.2 用例模型/需求概览
- 2.3 假设与依赖关系

## □ 3. 用例报告/功能需求

- 3.1 用例1/功能1
- 3.2 用例2/功能2
- ...

## □ 4. 非功能需求

- 4.1 可靠性需求
- 4.2 性能需求
- 4.3 易用性需求
- 4.4 可支持性需求
- 4.5 设计约束
- 4.6 接口需求
  - 用户界面、硬件接口、软件接口和通信接口
- 4.7 联机用户文档和帮助系统需求
- 4.8 适用的标准
- 4.9 许可需求
- 4.10 法律、版权及其他声明

## □ 5. 支持信息

# 需求验证

---

## □ 原型确认

- 抛弃型原型确认
- 演进型原型确认

## □ 需求评审

- 评审需求文档（Vision和SRS等），及时发现缺陷，寻找改进的契机，同时从评审反馈中获得知识，补充了正规的交流和培训机制，帮助团队建立对产品的共同理解



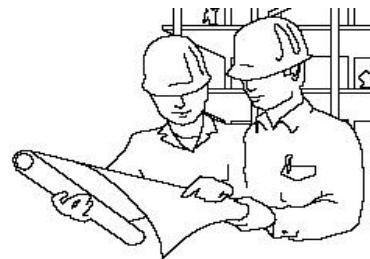


# 需求评审方法

- 审查
- 小组评审
- 走查
- 结对编程
- 同级桌查
- 轮查
- 临时评审



正式化程度



# 需求评审的输入和输出

## □ 输入

- 待评审的需求文档
- Check list

## □ 输出

### ■ 评审结论

- 通过
- 有条件通过
- 不通过

### ■ 缺陷清单



# 大纲



中南大學  
CENTRAL SOUTH UNIVERSITY

## 第三章 软件需求

01-软件需求概述

02-需求获取

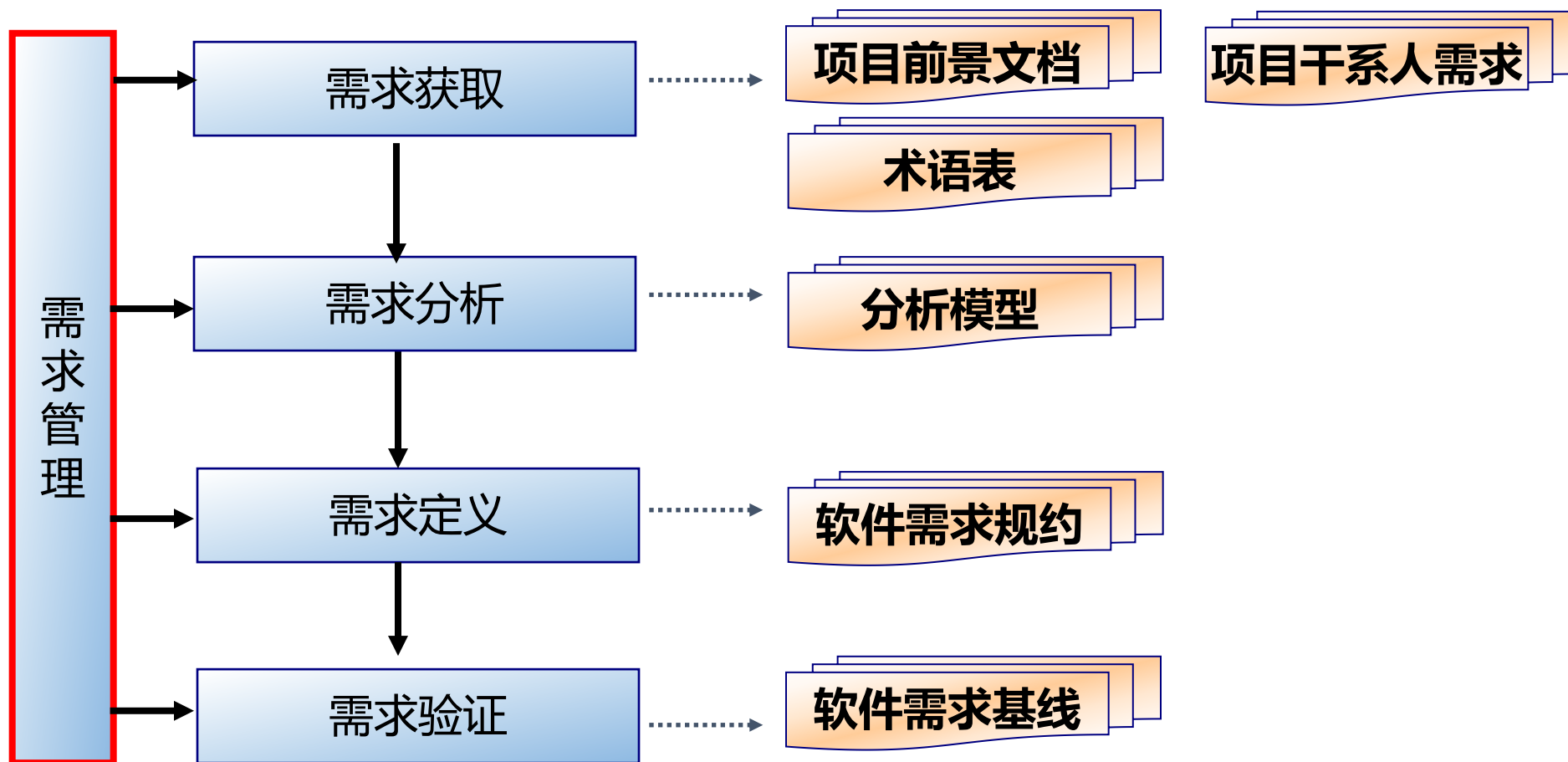
03-需求分析和建模

04-需求定义和验证

☀ 05-需求管理

@第4章.教材

# 需求工程



发现、获取、组织、分析、编写和管理需求的系统方法，让客户和项目组之间达成共识。

# 需求管理

---

- 1) 定义需求基线
- 2) 需求变更控制和版本控制 (建立新的需求基线)
- 3) 需求跟踪

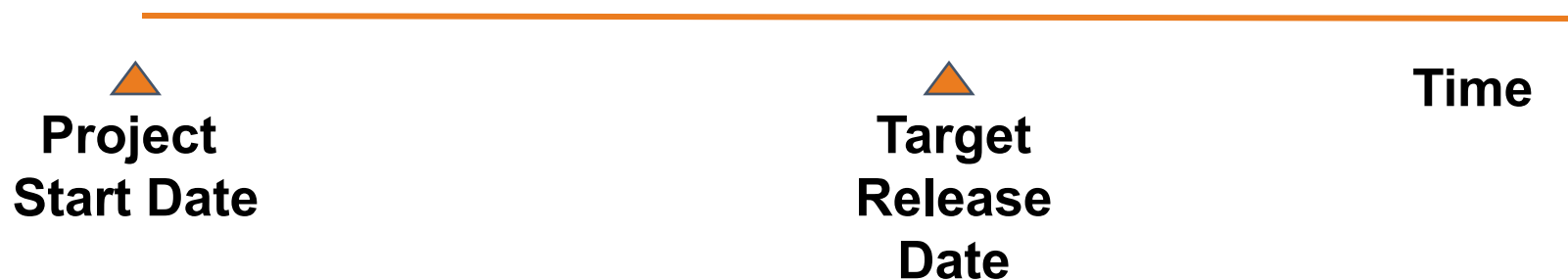
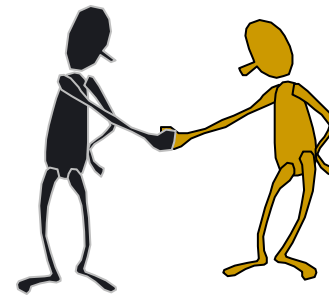
# 1) 建立需求基线

- Feature 1: The system must...
- Feature 2: The system must...
- Feature 3: The system must...
- Feature 4: The system must...
- Feature n: The system must...

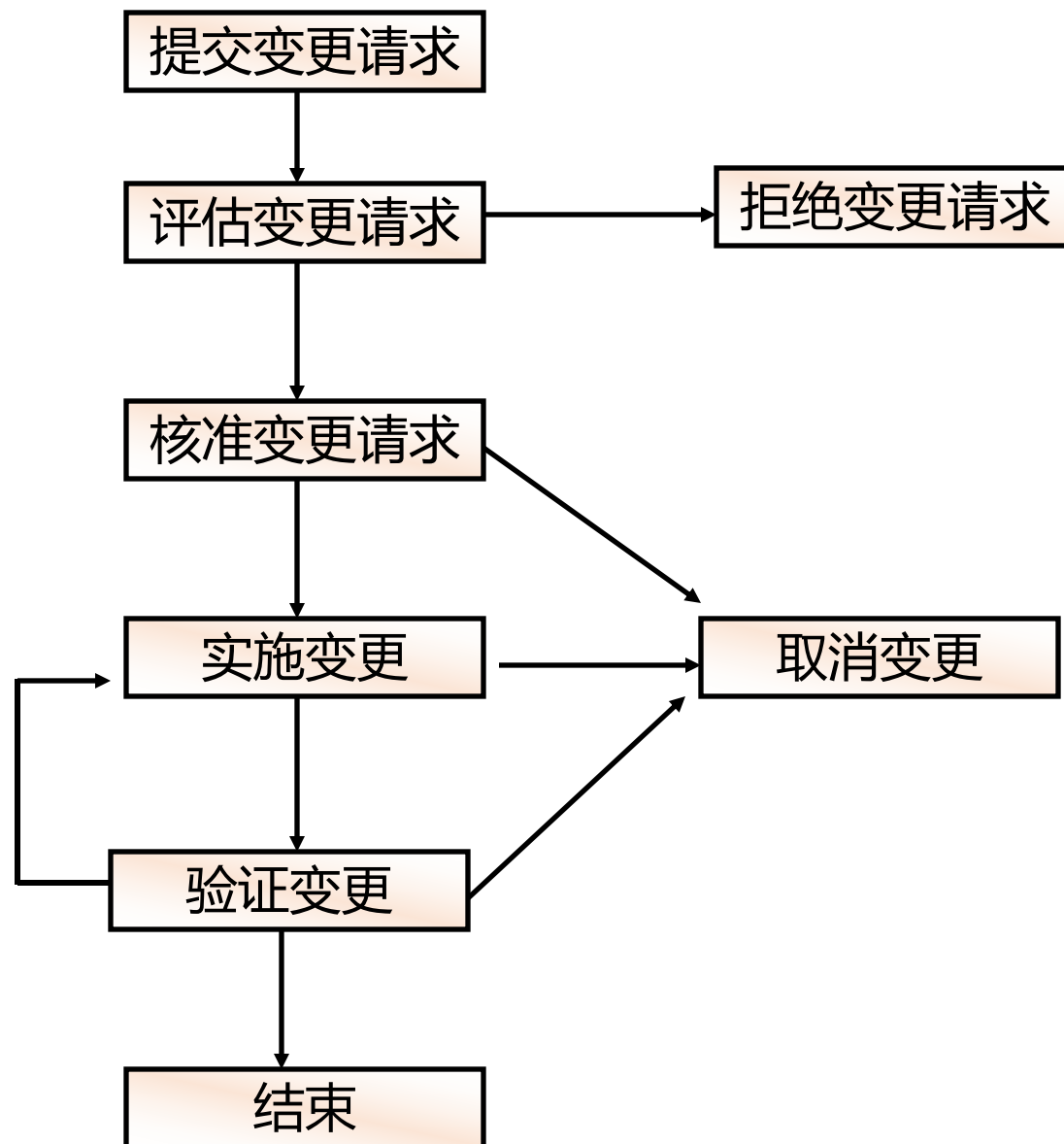
How do we know what the needs are?

How do we determine priority?

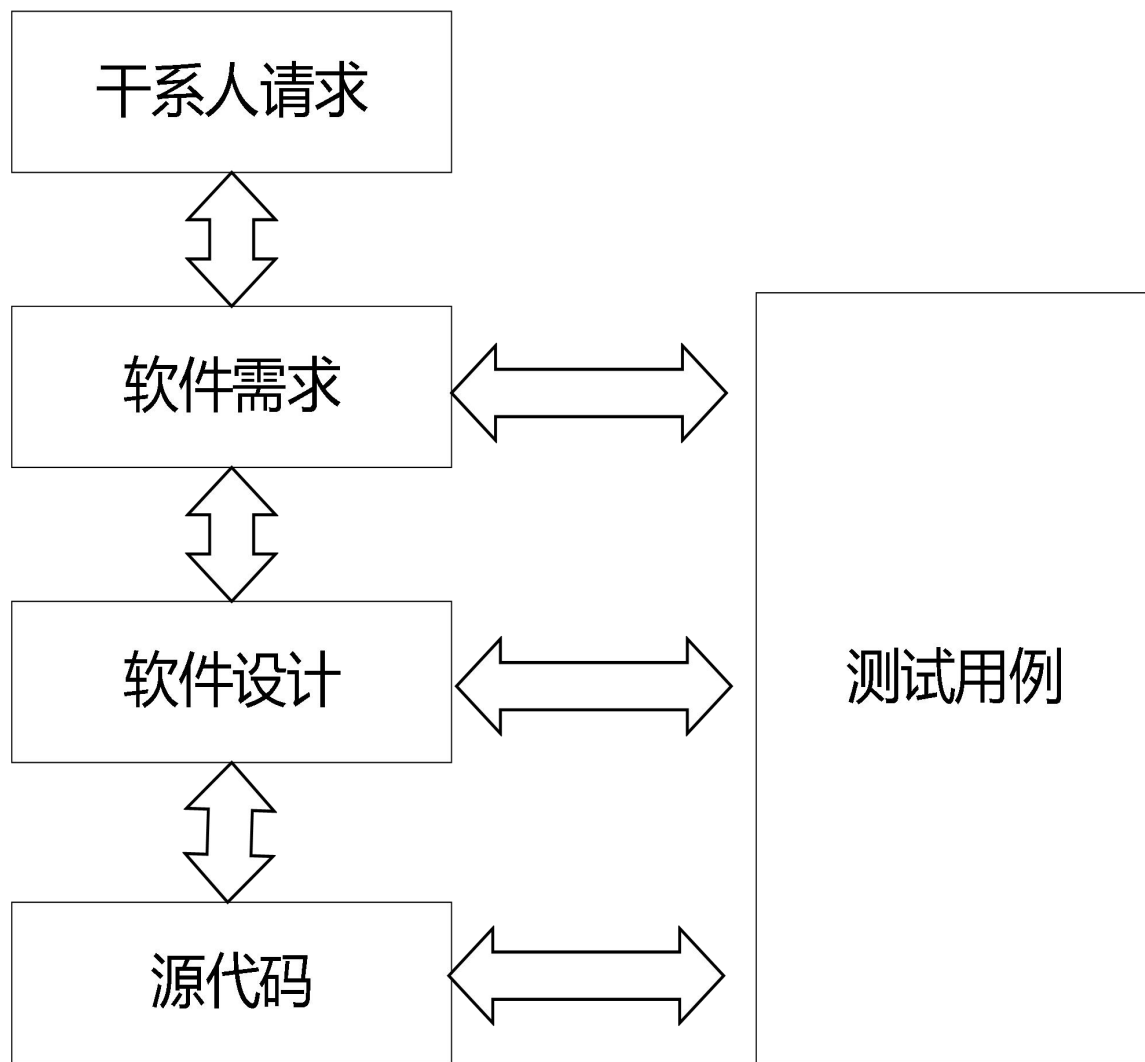
Where do we set the baseline?



## 2) 需求变更控制



### 3) 需求跟踪



#### 需求跟踪信息的作用:

- 了解需求的来源
- 掌握项目的进展情况
- 检查软件系统仅仅做了期望做的事情
- 确保软件测试的全面性
- 分析需求变更的影响
- .....