

# 第04讲 需求用例分析

徐峰磊



2022 (C) 电子与信息工程学院

# 主要内容

1

需求难题



2

用例基础

3

用例建模

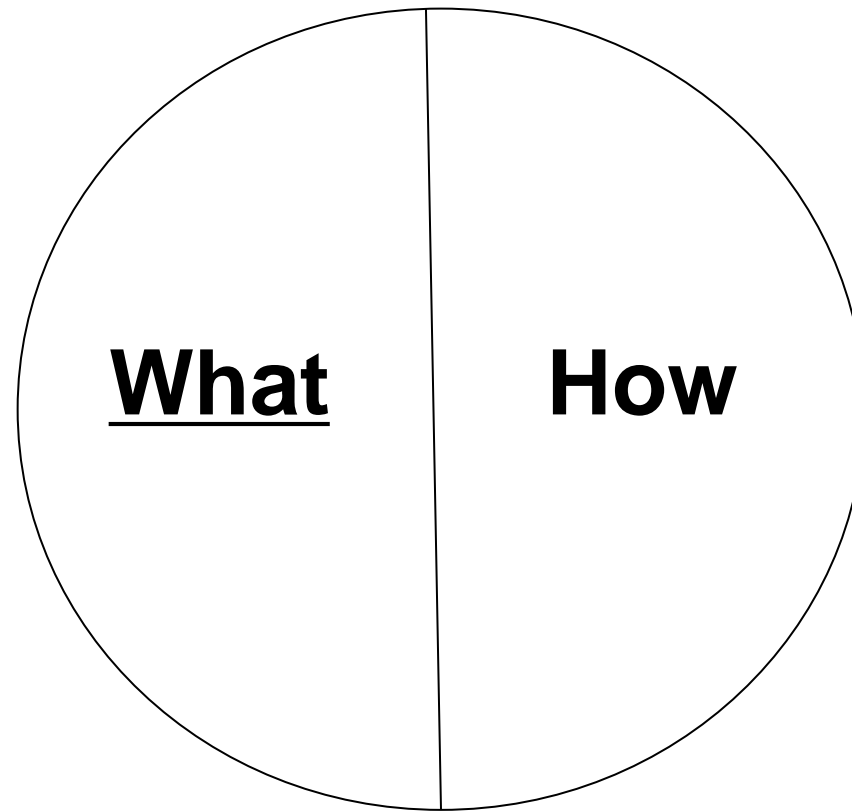
# Part1 需求难题

- 什么是需求？
- 常见的需求问题
- 如何破解需求难题？

# 什么是需求？

**Requirement**

# 需求



# 需求的定义

- **IEEE 97**

用户解决问题或达到目标所需的条件或能力。

系统或其部件要满足合同、标准、规范或其他正式规定文档所需具有的条件或能力。

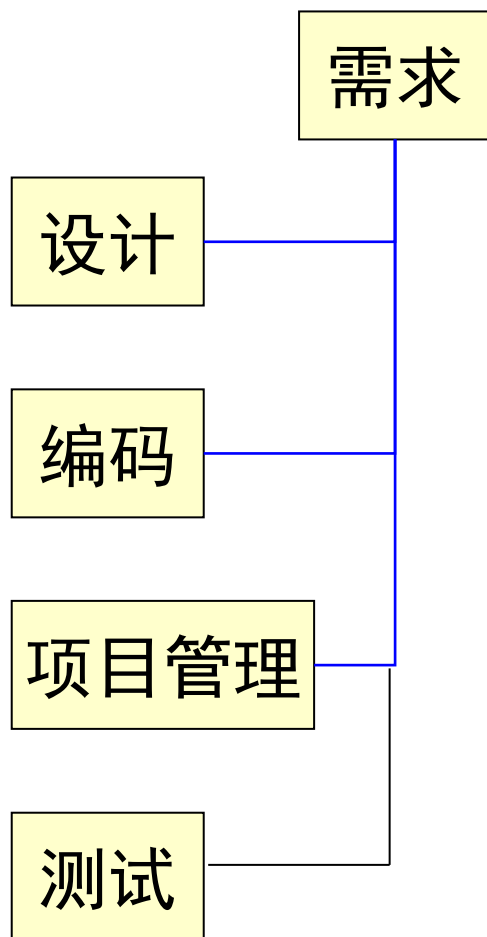
- **A. Davis 93**

从系统外部能发现的，系统所具有的满足于用户的特点、功能及属性等。

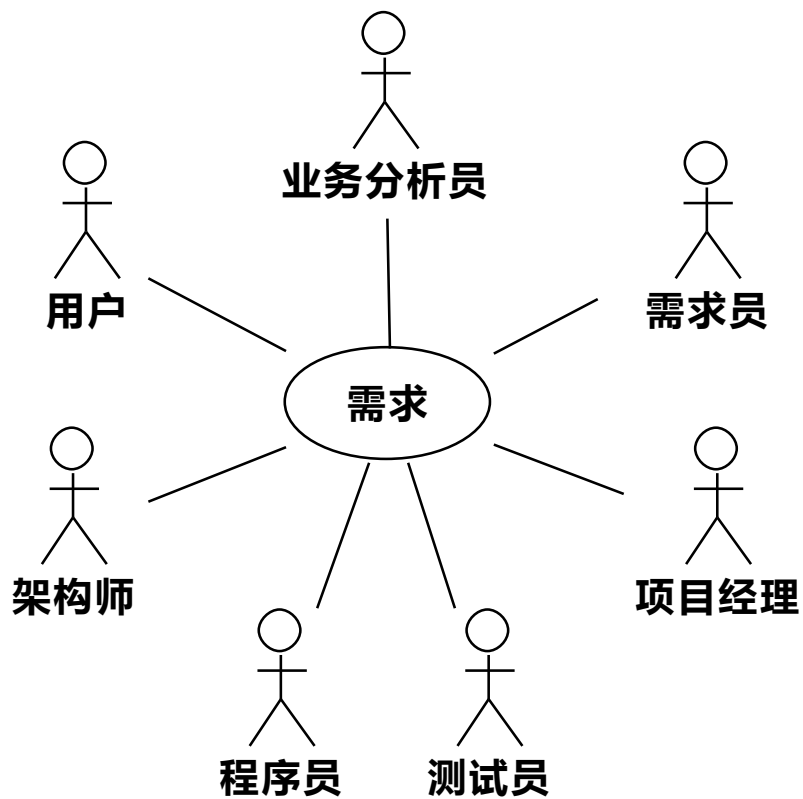
- **UML**

系统的一个期望特性、属性或行为。

# 需求是软件开发的上游



# 需求关系到所有人 \*





# 常见的需求现象-1

1. 用户提出的需求不完整、不准确。
2. 需求经常变化，工作没完没了。
3. 开发后期才发现误解了需求，开发人员被迫大量加班，修修补补。
4. 功能都实现了，但由于性能、使用方面的问题导致用户不满。

# 常见的需求现象-2

- 5. 客户的许多增强要求未及早提出，导致软件后期维护费用陡升。
- 6. 测试效果差，测试人员不明白软件要做什么
- 7. 勉强交付了客户并不满意的产品
- ...

# 常见的需求问题

1. 需求不完整，太粗略，遗漏关键信息。
2. 频繁、大量的需求变更|变化
3. 用户不积极参与
4. 模棱两可、含糊的需求
5. “锦上添花”的需求
6. 忽略了用户分类
- ...

# 低质量需求的后果

- 遗漏需求

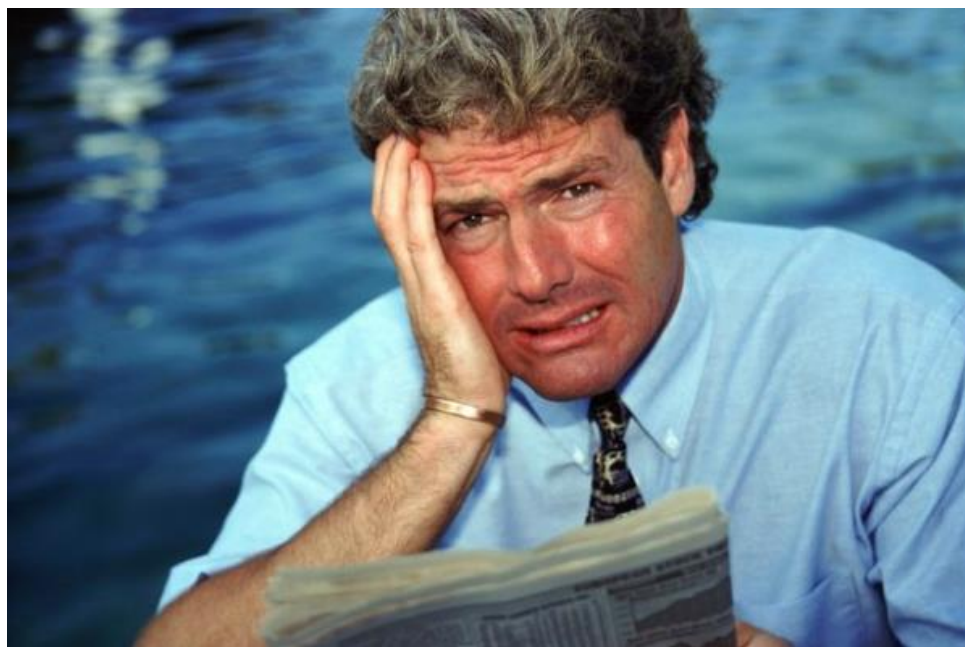
  - 项目计划、估算错误

  - 影响进度

- 需求不准

  - 返工

  - 增加成本



# 需求错误的成本

各阶段修复需求错误的成本比（Boehm 1998）：

需求 (0.5-1)

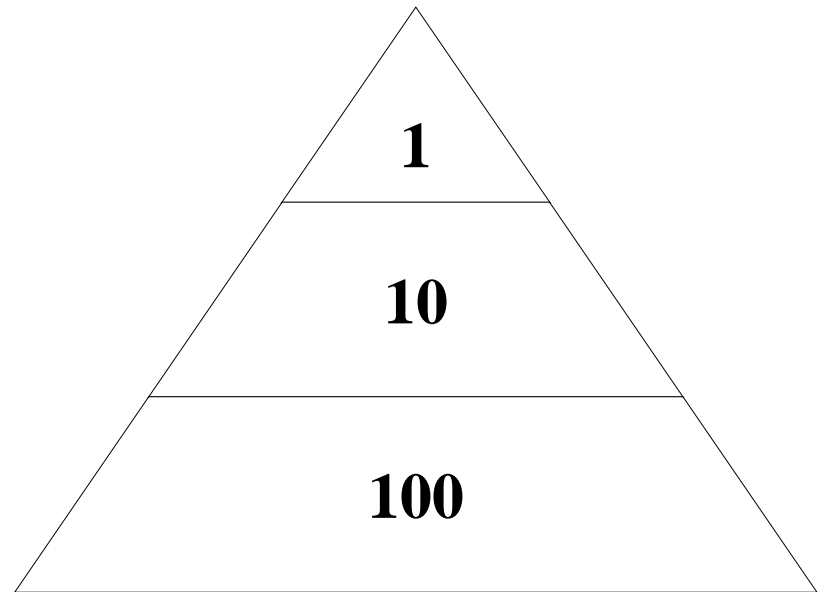
设计 (2.5)

编码 (5)

单元测试 (10)

验收测试 (25)

维护 (100)



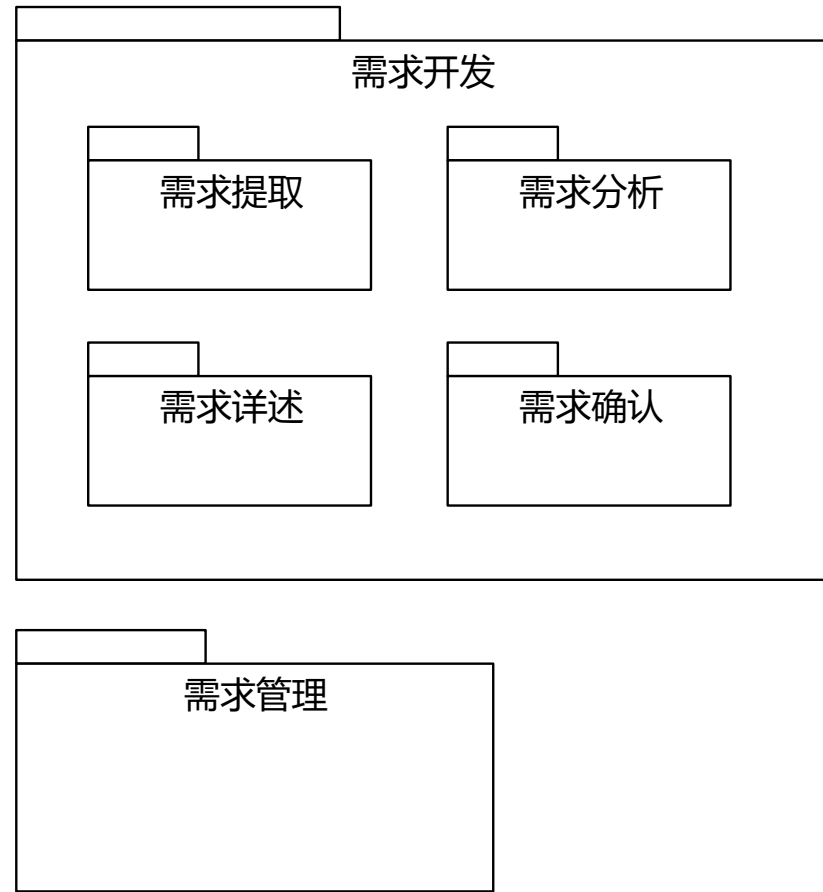
# 需求错误的影响

## 据国外研究统计

- 软件项目中 40% 到 60% 的问题都是在需求分析阶段埋下的隐患。
- 返工开销占开发总费用的 40%，而 70-80% 的返工是由需求方面的错误所导致的。

# 解决之道 – 需求工程

- 科学的
- 系统的
- 规范的
- 经济的
- 人本的



# 需求工程的价值

成功的需求开发与管理

能为组织

带来 **巨大** 回报！





# 目标 – 高质量的需求

1. 完整性
2. 正确性
3. 可行性
4. 必要性
5. 分级性
6. 无多义性/准确性
7. 可验证性
8. 一致性
9. 可扩展性
10. 可跟踪性
11. 可管理性
12. 稳定性

# 难题 – 预测需求变化



# 需求变化 – 合理的原因

- 用户不了解自己的需求

用户不是 IT 专家

没有实物，需求很抽象，空想。

- 需求本身易变

市场变化

技术变化

竞争变化

...

# 需求变化 – 不合理的原因

- 客户沟通问题

  - 理解错误

  - 业务不清、不熟悉

- 需求文档质量不高

  - 缺乏预见性

  - 遗漏关键信息

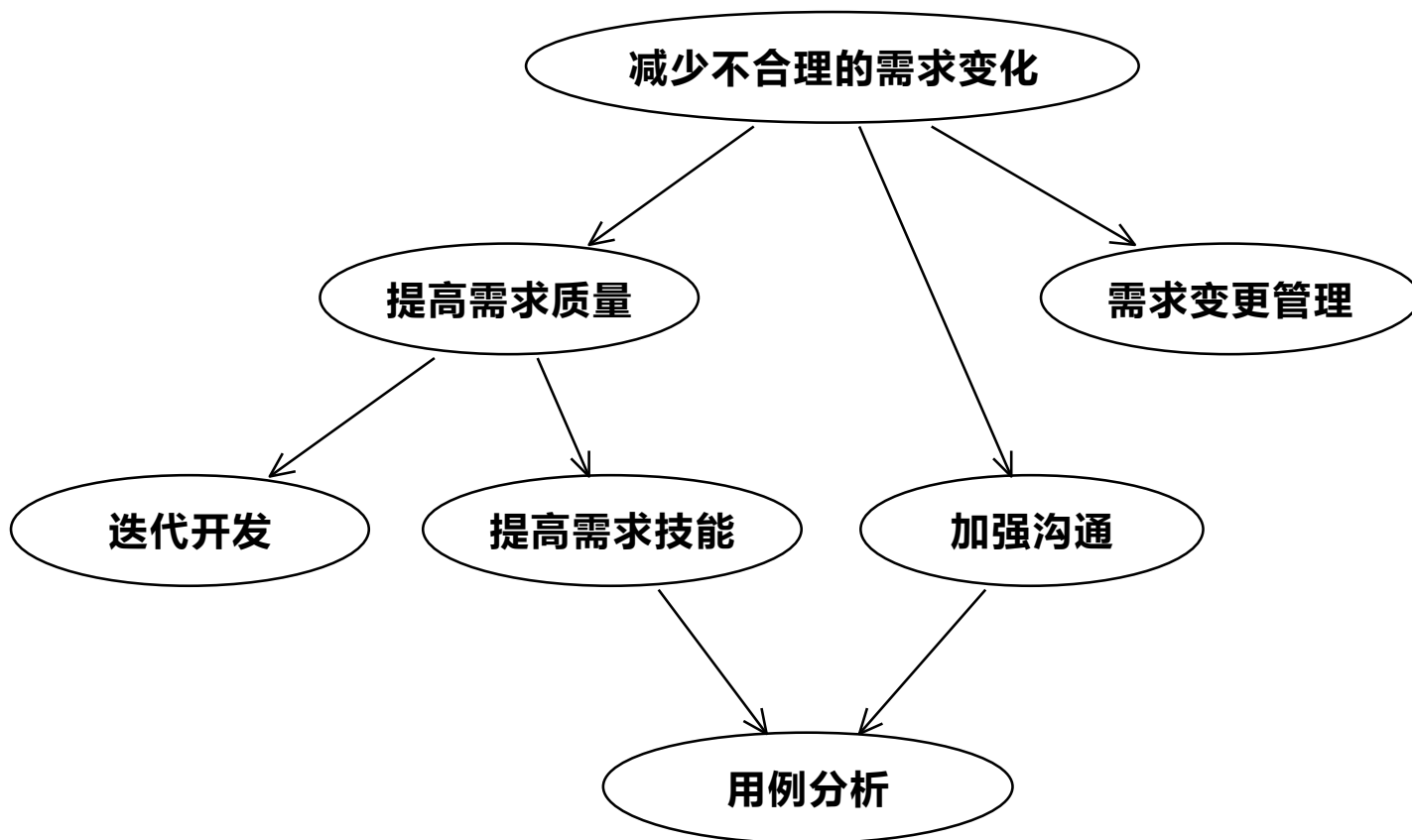
  - 需求分析技能、技术缺陷

- 需求管理缺陷

  - 变更控制

  - ...

# Part1 小结



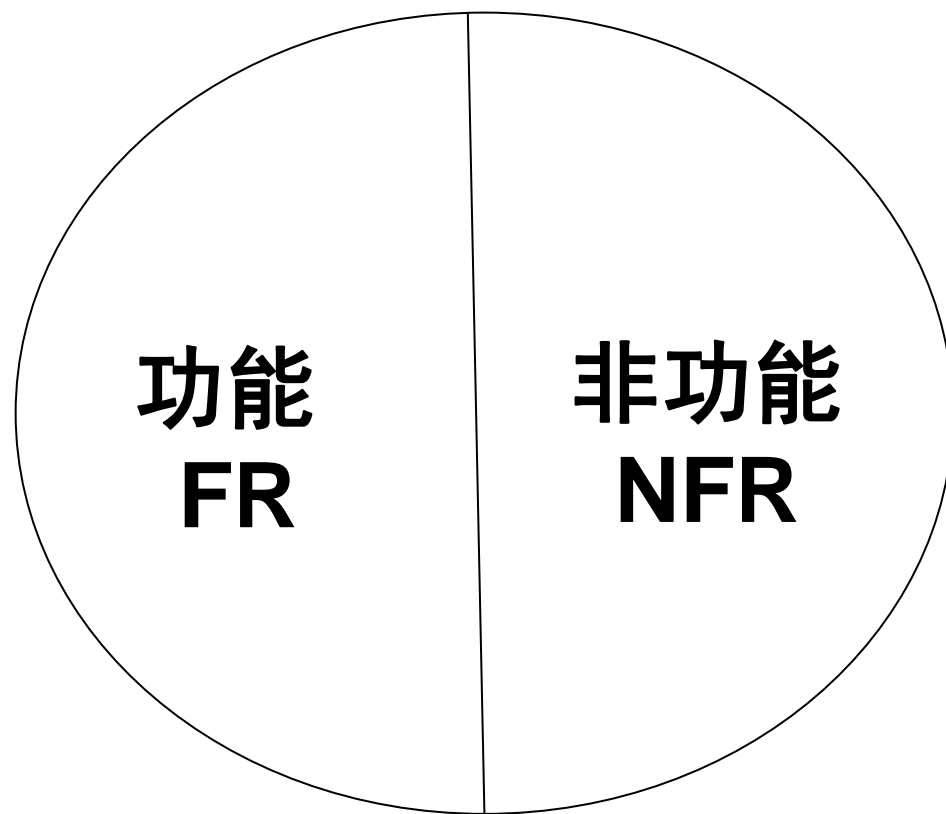
# Part 2 用例基础

- 软件需求的组成 ←
- 什么是用例？
- 用例与其他需求的比较
- 用例的价值

# 软件需求的种类

- 业务需求、业务规则
- 特性|特征
- 用例（功能）需求
- 质量属性
- 外部接口（协议）
- 各种约束
  - 解决方案约束（对设计、实现的限制）
- 数据需求

# 软件需求 – 功能|非功能





# FR (Functional Requirements)

- 1) 系统输入到输出的映射及其组合;
  - 2) 系统外部可见的行为需求。
- 与用户任务、目标、用例、服务、特性相关
  - 一般可通过描述系统的输入、输出、功能、系统的属性和系统环境的属性来完整地描述软件需求。

# NFR (Non-Functional Requirements)

- 质量属性
- 业务规则
- 约束（对解决方案自由度的限制）

约束源包括：经济、行政、技术、环境（标准、法律、安全）、进度、资源、战略、业务运作、合同等等。

# NFR-软件|系统质量属性

1. 有效性
2. 效率|性能
3. 灵活性
4. 完整性
5. 互操作性
6. 可靠性
7. 健壮性
8. 易用性
9. 可用性
10. 可维护性
11. 可移植性
12. 可重用性
13. 可测试性
- ...

# FURPS+ (Grady 1992)/1

- 功能 (Functionality)

特性集、能力、通用性、安全性等等。

- 易用性 (Usability)

人性因素、美学、一致性、文档等等。

- 可靠性 (Reliability)

失效频度/严重性、可恢复性、可预测性、精确性、MTBF 等等。

# FURPS+/2

- 性能（Performance）

速度、效率、资源使用、吞吐量、应答时间等等

- 可支持性（Supportability）

可测试性、可扩展性、适应性、可维护性、兼容性、健壮性；

可配置性、可保养性、安装能力、本地化能力等等。

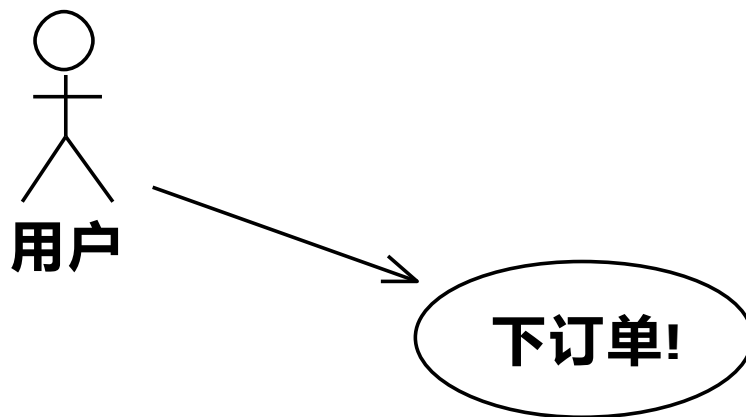
设计约束、实现、接口、物理要求等等。

# Part 2 用例基础

- 软件需求的组成
- 什么是用例？ ←
- 用例与其他需求的比较
- 用例的价值

# 什么是用例?

- 针对某个特定的外部用例，系统通过与其交互而执行的，能够产生可观测的、有价值的结果的一个动作序列。



# 用例的组成-1

1. 名称
2. 简述
3. 用例与干系者
4. 层次
5. 范围
6. 后态
7. 前态
8. 触发事件



# 用例的组成-2

9. 基本流

10. 扩展流

11. 扩展点

12. 用例图

13. 业务规则

14. 约束

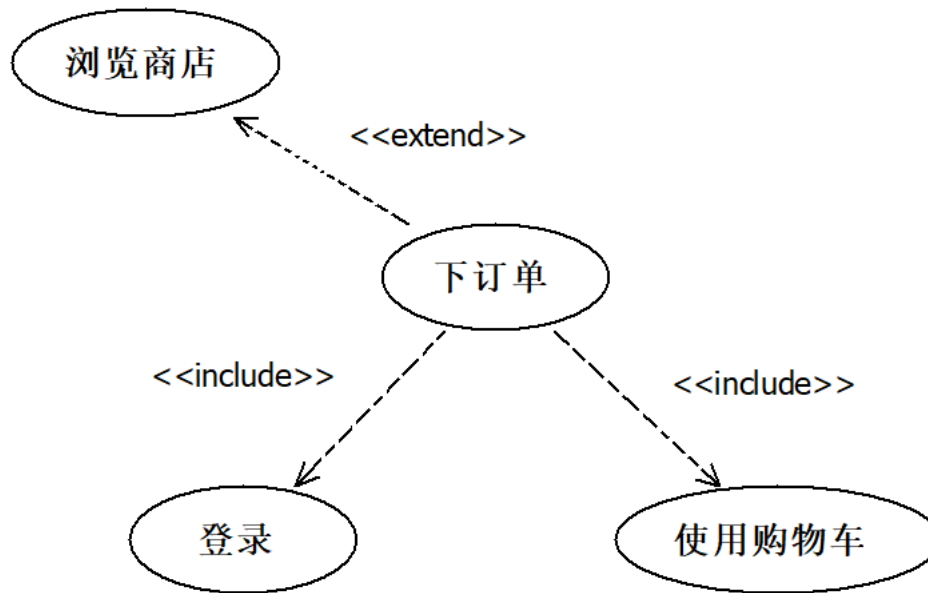
15. UI 说明

...

# 下订单/简述

- 顾客通过网店系统订购宠物，把选中的商品加入购物车；
- 登录后，下订单（含商品的种类、数量和价格，帐单地址、送货地址和信用卡等信息）
- 采取送货、信用卡支付方式。

# 例/用例图



# 下订单/基本流/填写交易信息

1. 系统显示该用户的缺省账单地址和送货地址，提供信用卡表单供用户填写。
2. 用户填写信用卡信息（卡类型、卡号和密码），并确认地址后提交。
3. 系统验证用户提交的交易信息（包括信用卡等）。
4. 系统显示完整的交易信息（包括用户已填写的信用卡信息）让用户确认。
5. 用户选择继续。

# 下订单/扩展流/验证交易信息

:填写交易信息/3 { // 验证

[变更送货地址] {

1. 系统显示空白的送货地址表单。
2. 用户输入新送货地址并提交。
3. 系统验证该送货地址。

**RESUME**

}

[信用卡信息无效] { ... }

}

# Part 2 用例基础

- 软件需求的组成
- 什么是用例？
- 用例与其他需求的比较 ←
  - FR
  - NFR
  - 特性
  - 用户故事
- 用例的价值

# 用例 vs FR

## ●FR

Input 到 Output 的映射组合（用例）

通常描述比用例简单、笼统（简述、特性）

很多功能粒度小于用例，是用例的一些步骤。

## ●用例

所有的用例名称都代表了一种功能

用例的主体是一种规范的、更为精确和细致的功能描述方法和形式

用例就是功能（大略的说法）

# 用例 vs NFR

## ●用例

不是需求的全部，还存在非功能需求限制

与每个用例有关的非功能（特殊）需求可以放在用例文本的格式模版中，与用例的动作功能描述形成互补。

## ●NFR

在用例文本描述之外，制作一份单独的文档、表格或数据库，列举除了用例之外的所有其他软件需求（NFR、FURPS+、补充规约等）。

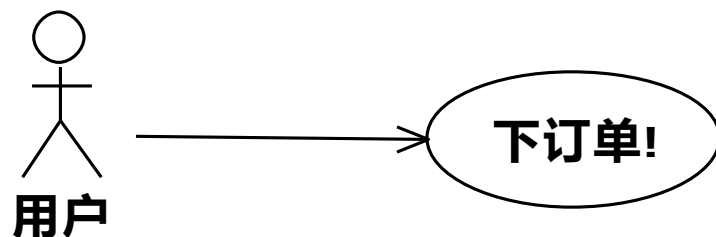


# Part 2 用例基础

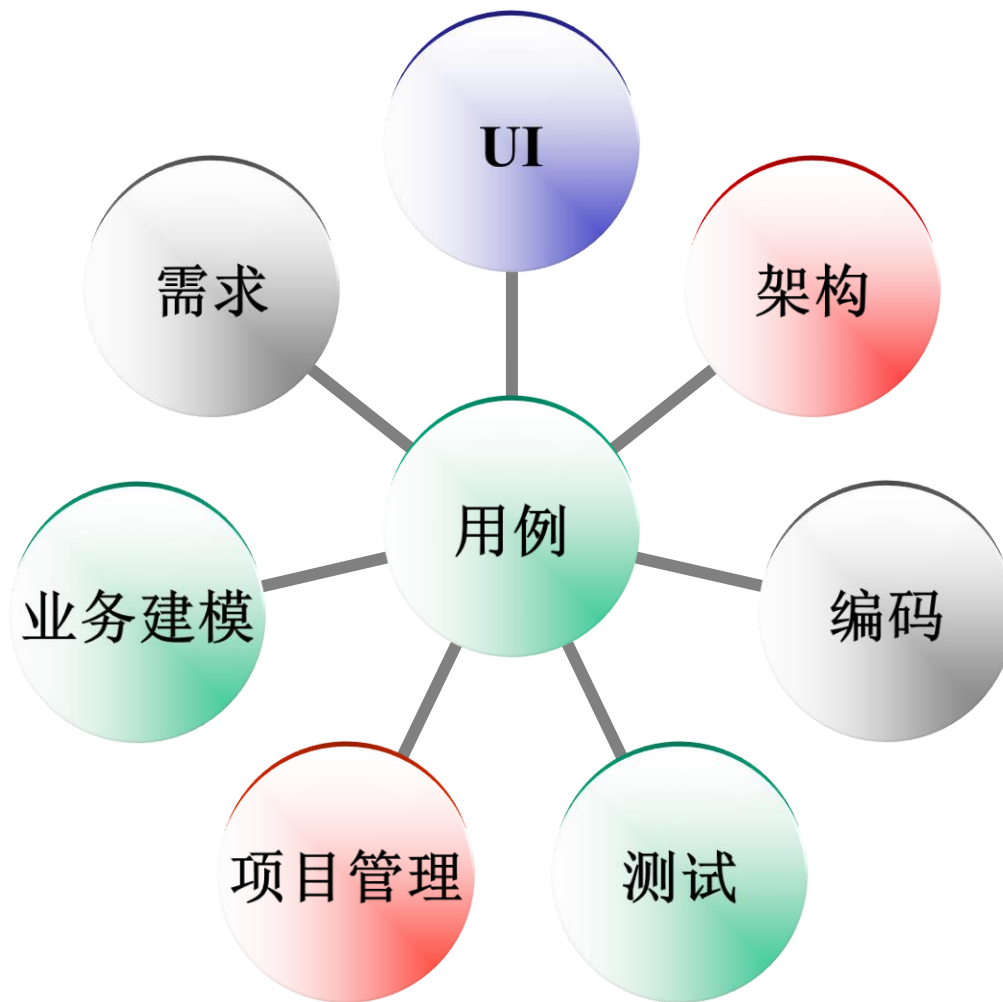
- 软件需求的组成
- 什么是用例？
- 用例与其他需求的比较
- 用例的价值 ←

# 用例的价值

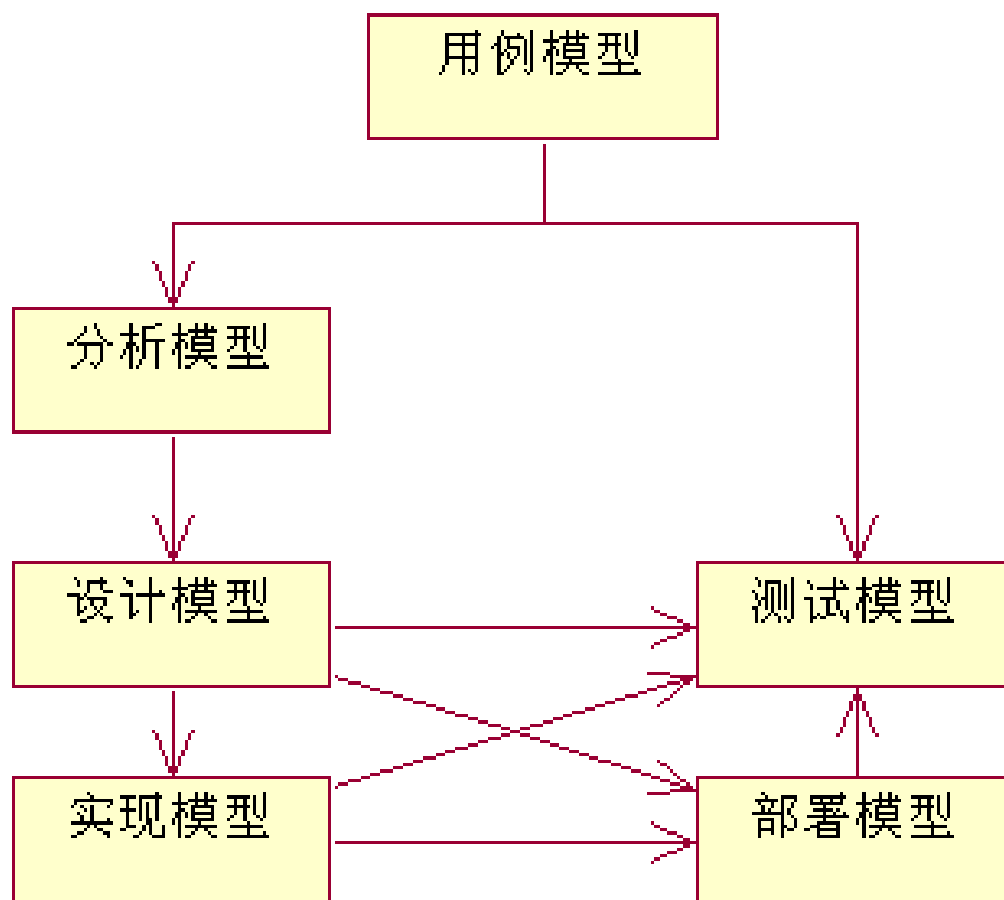
- 以用户为中心
- 最重要的需求（功能）
- 图文并茂、繁简结合、形式灵活
- 全面的字段和信息
- 规范的模版
- 清晰的层次结构



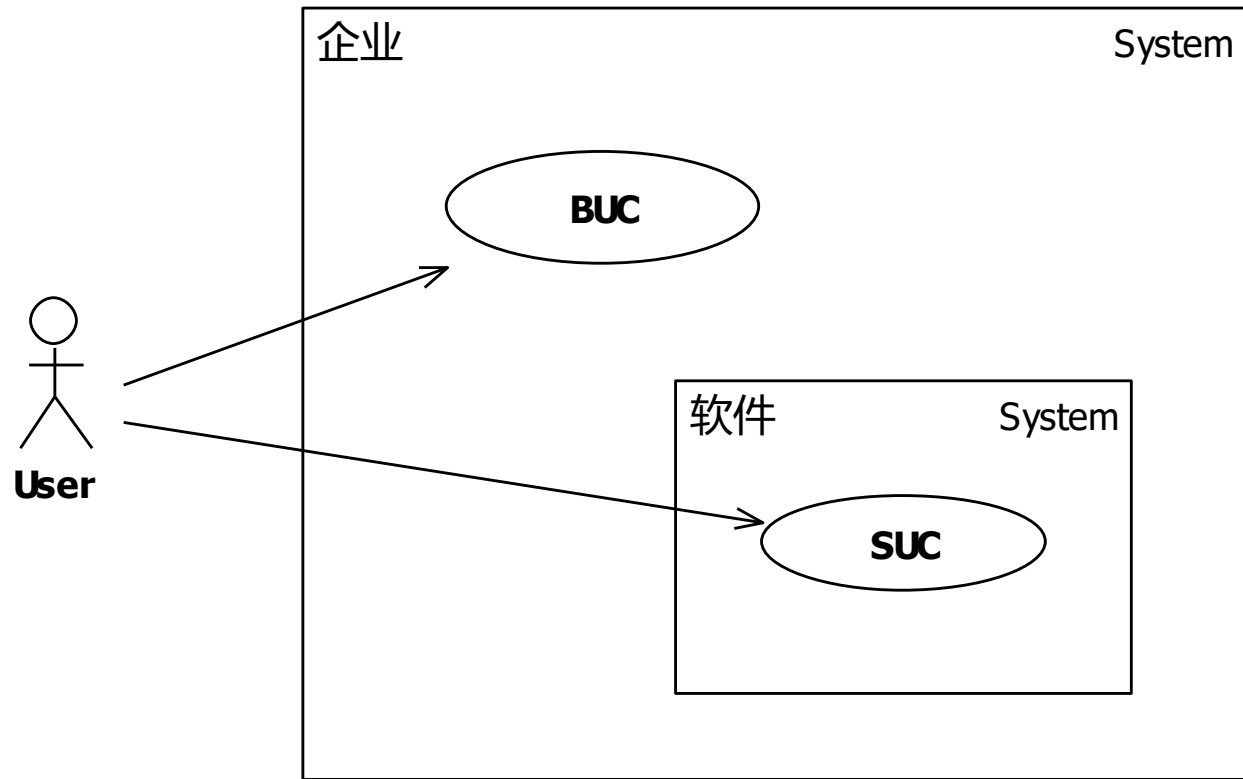
# 用例在软件开发中的核心地位 \*



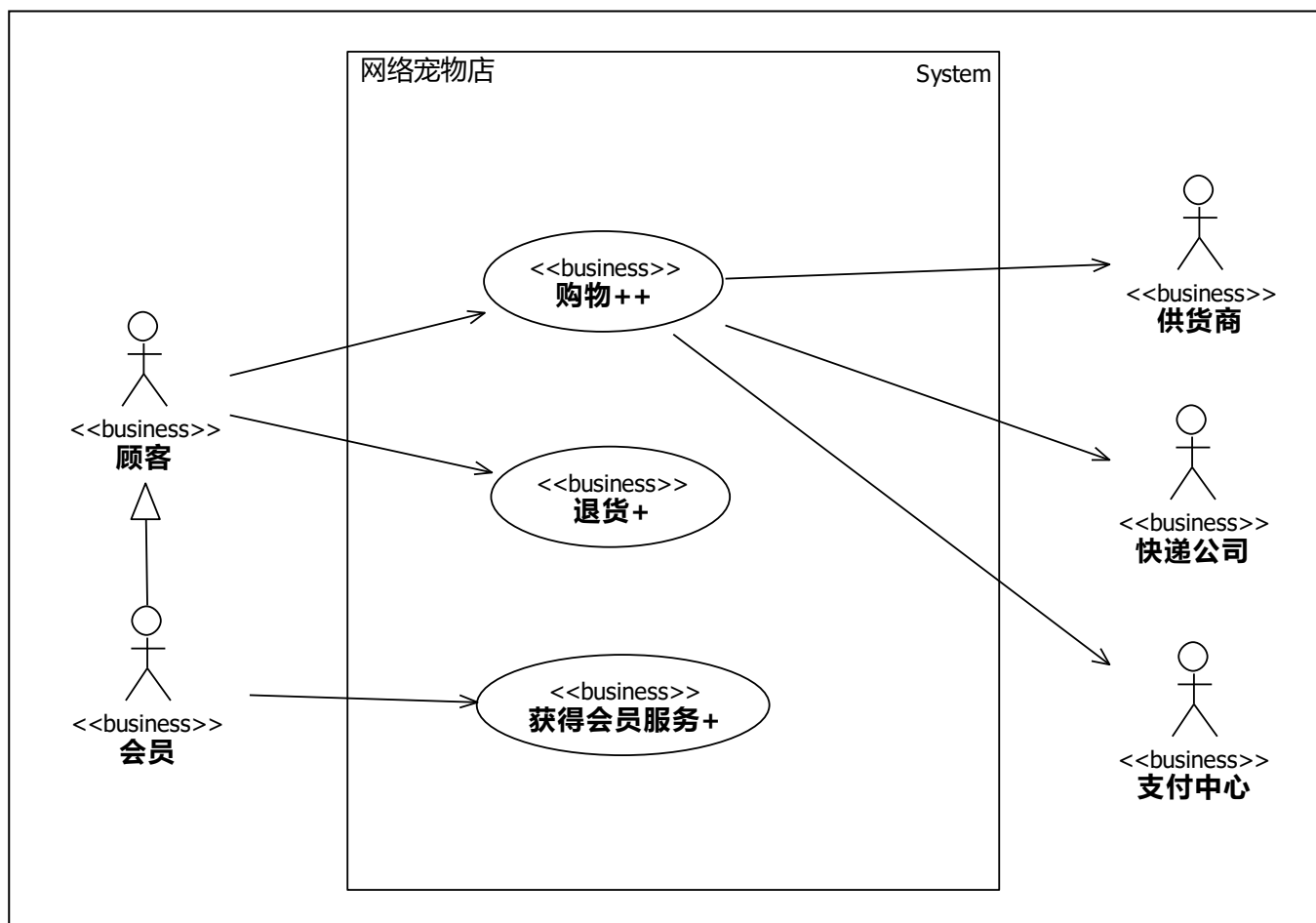
# 用例驱动软件开发



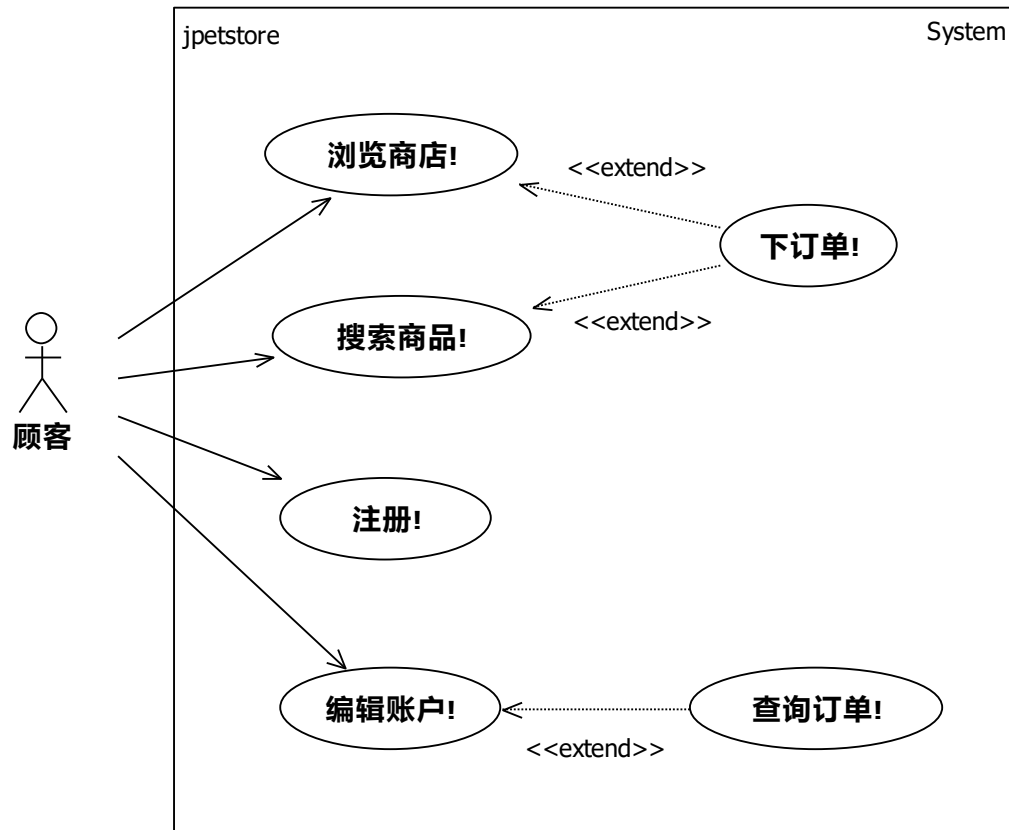
# 业务用例 vs. 软件用例



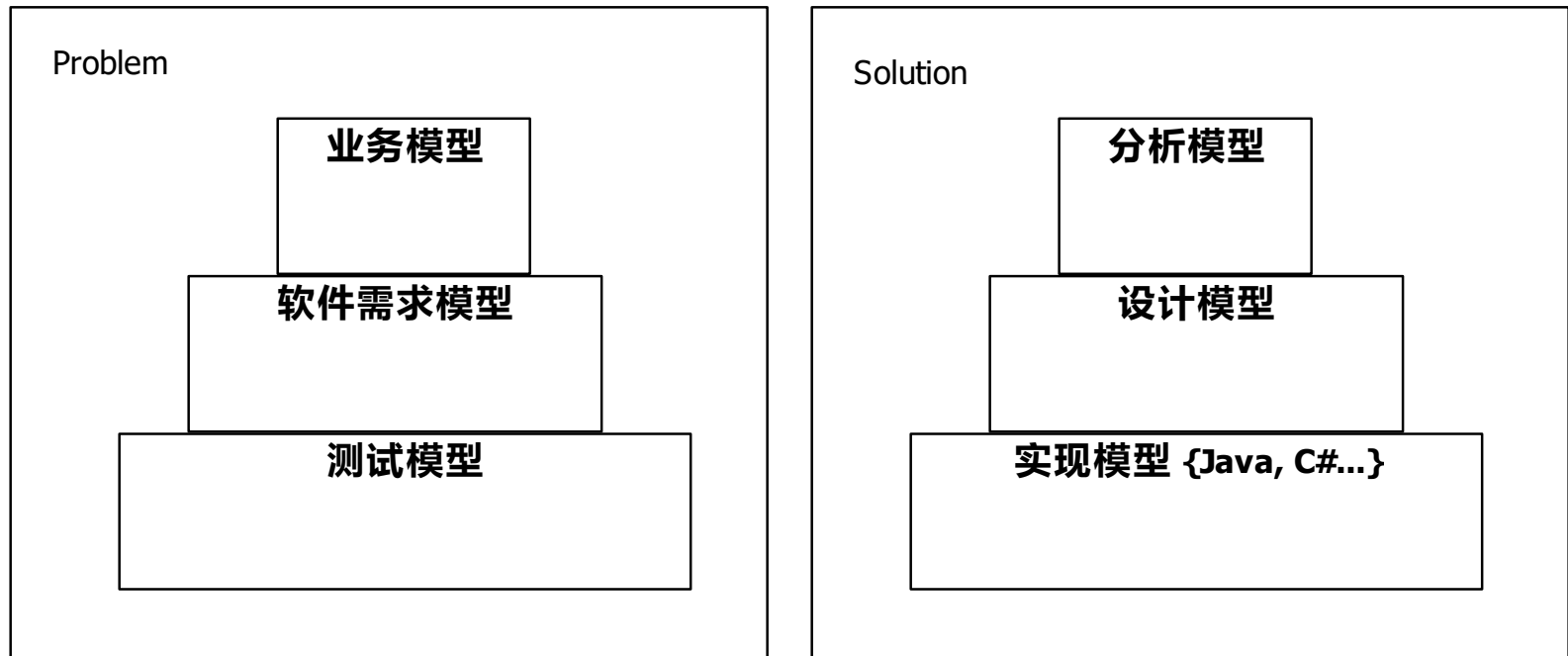
# 宠物店/业务用例图



# 宠物店/软件用例图




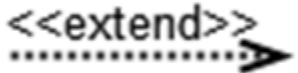
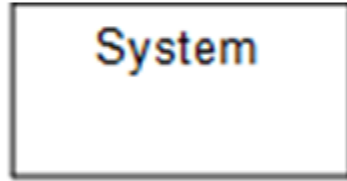


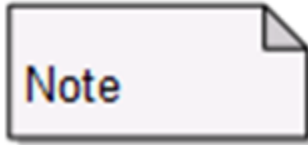



# 层次分明-软件架构





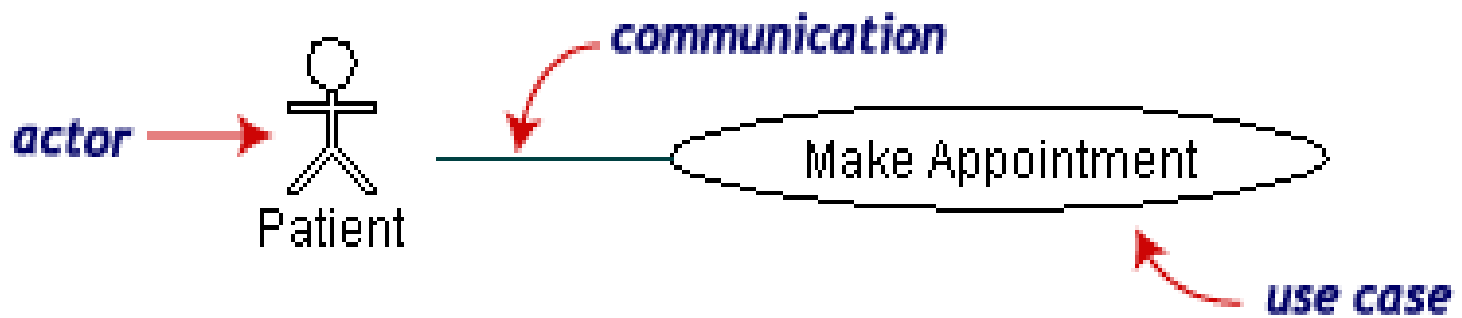
# Part 3 用例建模

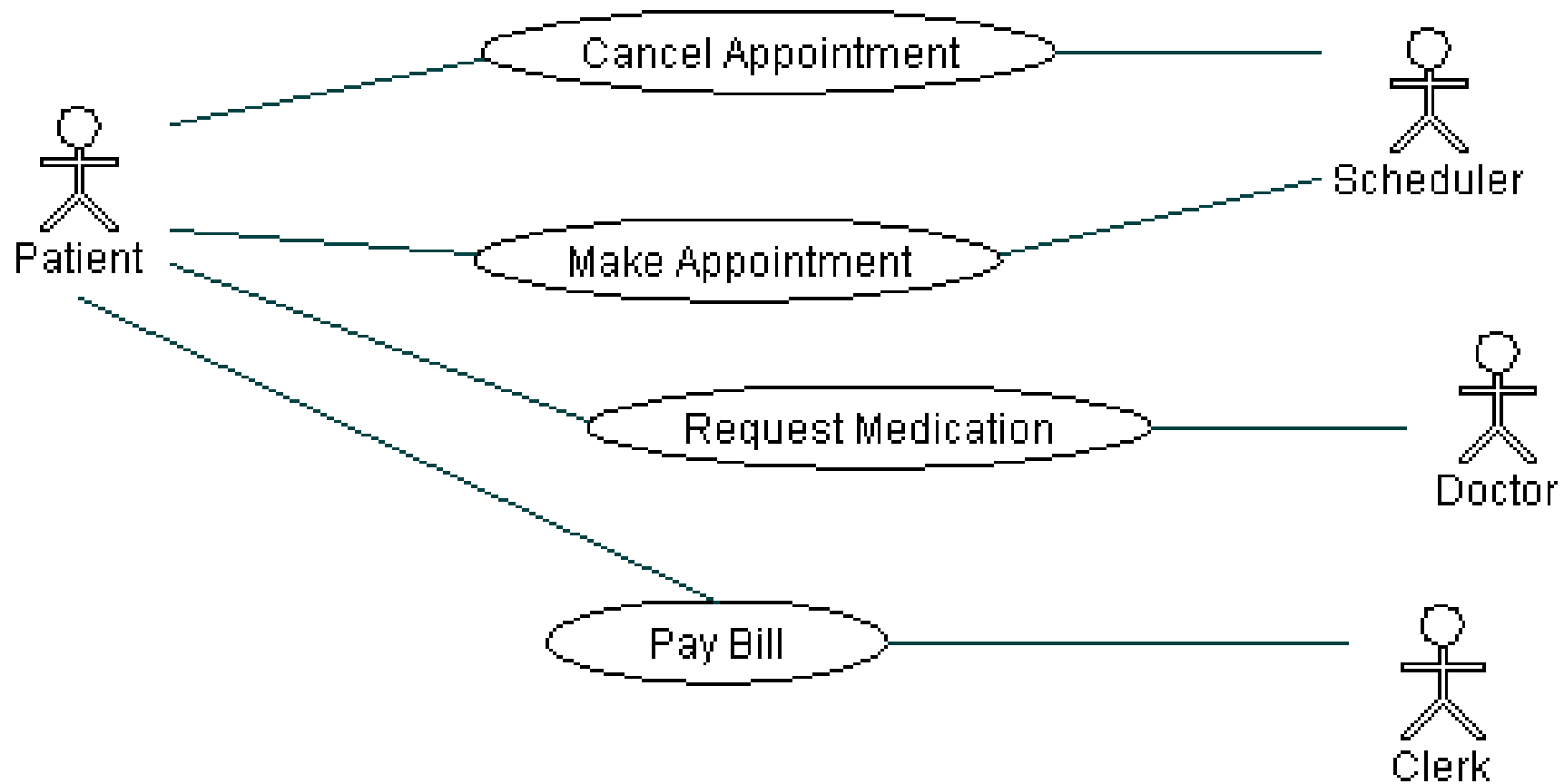
结构元素	图形符号	关系元素	图形符号
用例 (Use Case)		关联 (Association)	
参与者 (Actor)		扩展 (Extend)	
系统边界 (System Boundary)		包含 (Include)	
		泛化 (Generalization)	
注释 (Note)		注释连接 (Note link)	

- 用例图
- 用例图Use case diagrams描述了作为一个外部的观察者的视角对系统的印象。强调这个系统是什么而不是这个系统怎么工作。
- 用例图与情节紧紧相关的。情节scenario是指当某个人与系统进行互动时发生的情况。下面是一个医院门诊部的情节。

- UML的用例图由参与者、用例及它们之间的关系组成
- 它的表达方式为:

用例图 = 参与者 + 用例 + 关系





# 用例之间关系

<<extend>> ➤ 扩展

<<include>> ➤ 包含

————— ➤ 泛化

# 通过关系整理文档

- **Extend(扩展)**

通过扩展用例对基用例增加附加的行为

- **Include(包含)**

基用例中复用被包含用例的行为

提取公共步骤，便于复用

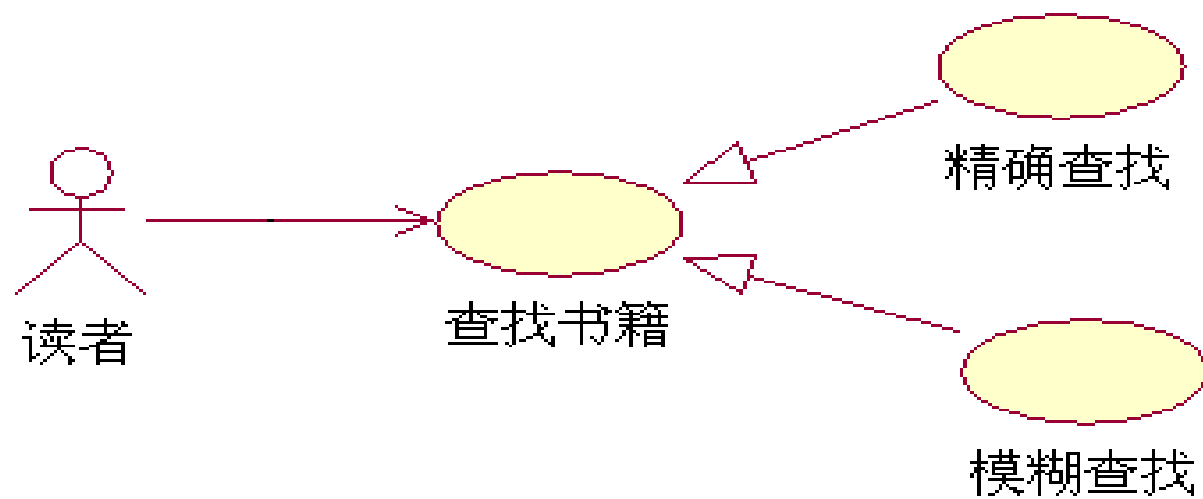
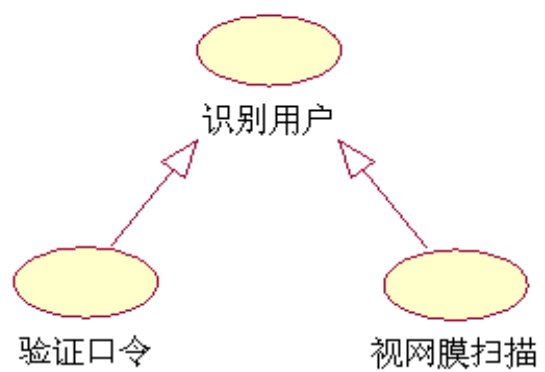
- **Generalization(泛化)**

派生用例继承泛化用例的行为并添加新行为

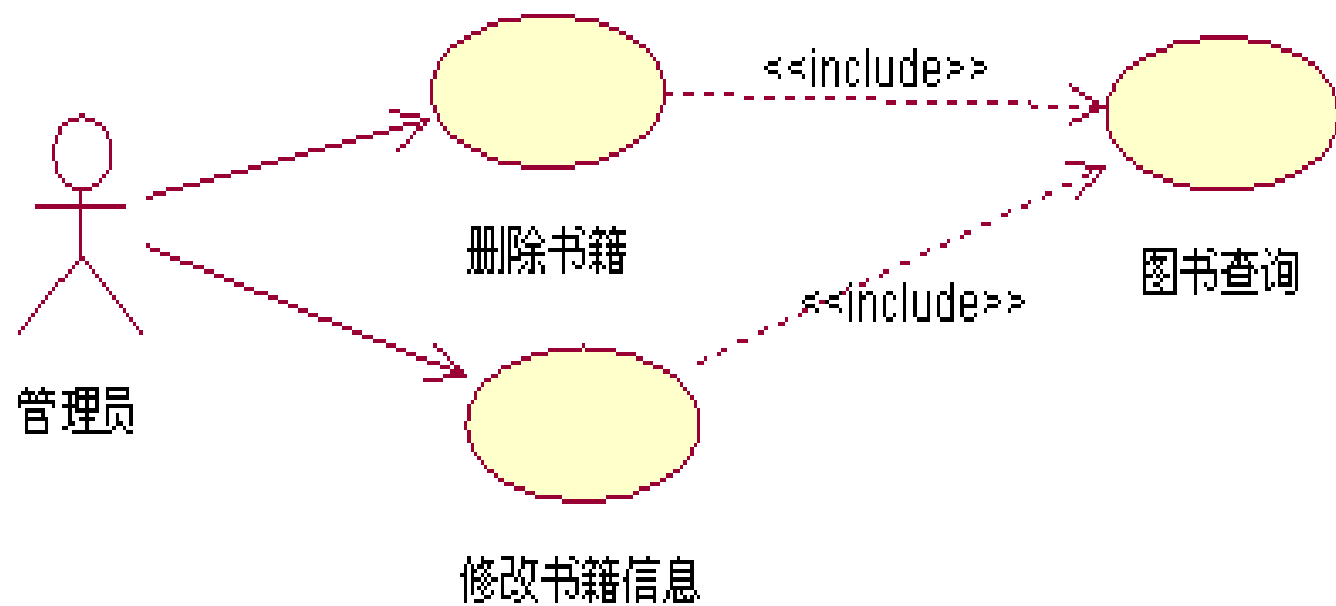
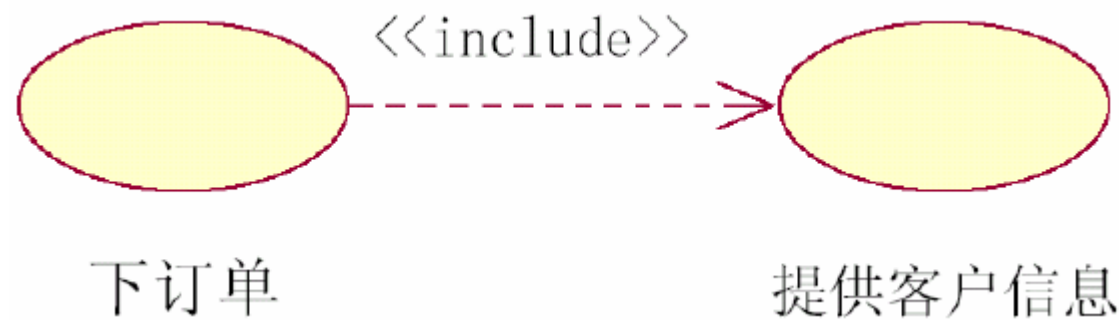
## ●1. 泛化关系(generalization)

- 当多个用例共同拥有一种类似的结构和行为的时候，我们可以将它们的共性抽象成为父用例，其他的用例作为泛化关系中的子用例。在用例的泛化关系中，子用例是父用例的一种特殊形式，子用例继承了父用例所有的结构、行为和关系，还可以添加自己的行为或覆盖已继承的行为。



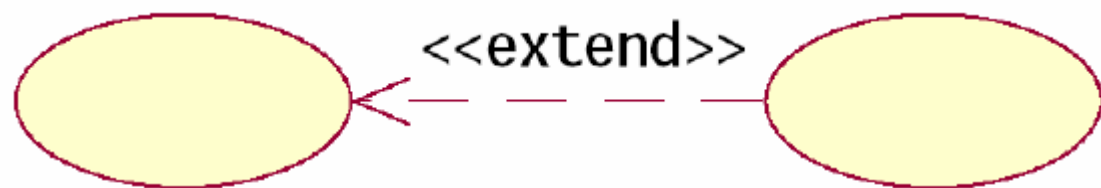


- **2. 包含关系(include)**
- **包含是指基础用例(base use case)会用到被包含用例(inclusion), 具体地讲, 就是将被包含用例的事件流插入到基础用例的事件流中。被包含用例是可重用的用例——多个用例的公用用例。**
- **包含关系是关联关系的一种。**



### ●3. 扩展关系(extend)

- 扩展关系也是关联关系的一种。假设基础用例中定义有一至多个已命名的扩展点，扩展关系是指将扩展用例的事件流在一定的条件下按照相应的扩展点插入到基础用例中。如果基础用例是一个很复杂的用例，选用扩展关系将某些业务抽象成为单独的用例可以降低基础用例的复杂性。



检查订单状态

取消订单

