# 软件工程
# 第五章　需求获取

乔立民

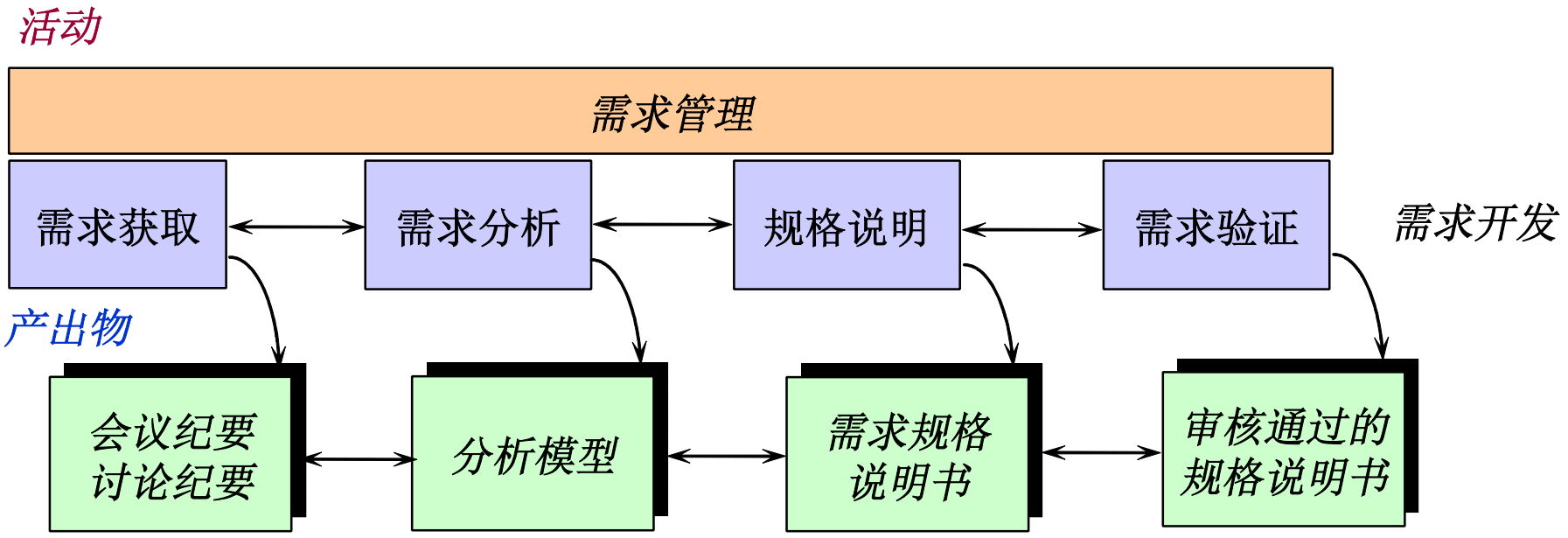
**qlm@hit.edu.cn**

**2011年4月20日**

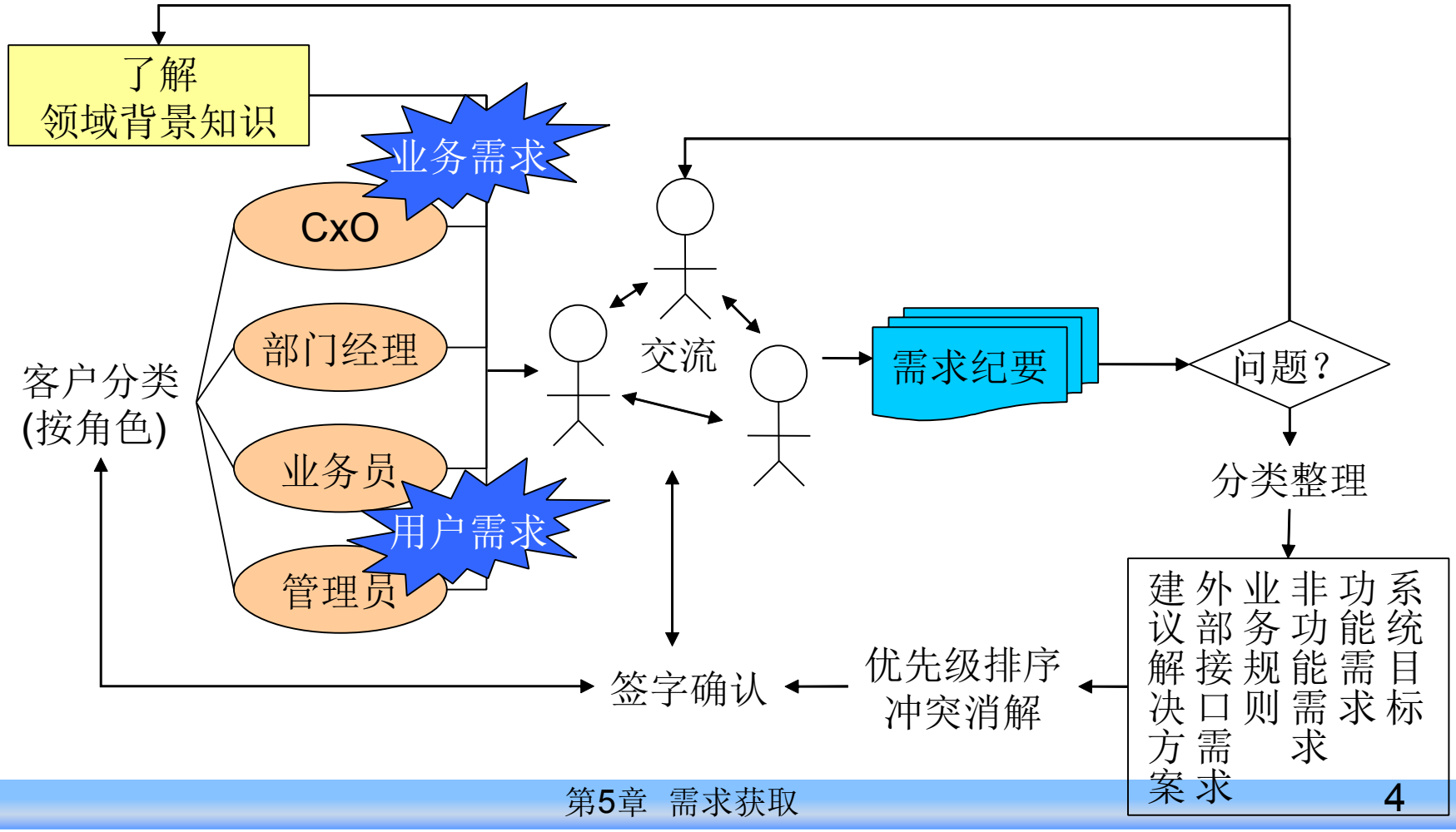
# 本章内容

- **5.1** 需求获取的挑战
- **5.2** 需求获取的途径
- **5.3** 需求获取技术
- **5.4** 需求获取管理

# 需求工程的总体流程


活动
需求管理
需求获取
需求分析
规格说明
需求验证
需求开发
产出物
会议纪要
讨论纪要
分析模型
需求规格
说明书
审核通过的
规格说明书

# 需求获取的基本步骤


了解
领域背景知识
业务需求
CxO
部门经理
客户分类
(按角色)
业务员
用户需求
管理员
交流
需求纪要
问题？
分类整理
系统目标
非功能需求
业务规则
外部接口需求
建议解决方案
优先级排序
冲突消解
签字确认

# 需求获取的基本步骤

- 第1步：了解相关背景和领域/行业的知识，确定产品所期望的用户类；
- 第2步：与客户企业或组织的高层人员进行交流，了解实际用户任务和目标以及这些任务所支持的业务需求；
- 第3步：与客户企业或组织的底层人员进行交流，获取每个用户类的详细的用户需求；
- 第4步：整理需求纪要，发现新问题，并重复1-3步；
- 第5步：需求分类和组织，形成系统需求，区别功能需求、非功能需求、约束条件、业务规则、外部接口需求、建议解决方法和附加信息；
- 第6步：优先排序和解决冲突；
- 第7步：整理出需求规格说明，并与客户协商确认。

# “看似简单，实际却很难…”

——“需求获取？不就是问问题吗？这有什么难的？”

# [案例分析1]“他们忙，没有时间与你讨论需求…”

- **“银弹”公司的CEO 王神话约见软件开发小组李敏捷，商讨为公司开发新系统的事情…**

| | |
|---|---|
| 王神话 | 我们的“银弹”经常出故障，客户总抱怨我们的产品质量！我们需要建立一套化学制品跟踪信息系统，可以记录并查询化学药品的使用情况…你们小组能在五个月内开发出该系统吗？ |
| 李敏捷 | 我已经明白这个项目的重要性了，但在我制定计划前，我们必须收集一些系统的需求。 |
| 王神话 | 你什么意思？我不是刚告诉你我的需求了吗？ |
| 李敏捷 | 你只说明了整个项目的概念与目标，这些高层次的业务需求并不能为我们提供足够的详细信息以确定究竟要开发什么样的软件，以及需要多长时间。我需要一些分析人员与一些知道系统使用要求的化学专家进行讨论，然后才能真正明白达到业务目标所需的各种功能和用户的要求。 |
| 王神话 | 那些化学专家都非常忙，没有时间与你们详细讨论各种细节，你不能让你的手下的人说明要做的系统吗？ |
| 李敏捷 | 如果我们只是凭空猜想用户要求，结果不会令人满意。 |
| 王神话 | 行了，行了，我们没有那么多时间，我来告诉你需求，请马上开始开发系统，并随时将你们的进展情况告诉我。 |

# (1) “Yes, But”综合症

…当你把新开发的系统展示给用户时…

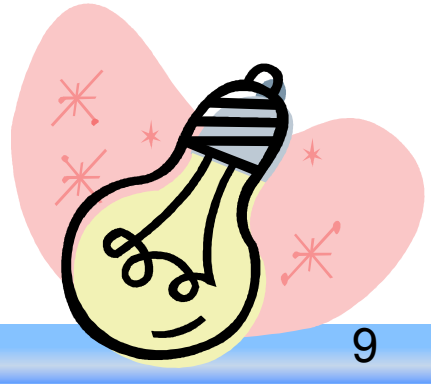
—— **“Wow**，太酷了！这正是我们想要的，你做了一个了不起的系统！”

…五分钟后…

—— **“Yes, but,** 嗯….这个模块是怎么回事？…如果你把它修改成这样岂不是会变得更好？…如果那样的话，我觉得我会更喜欢它…”

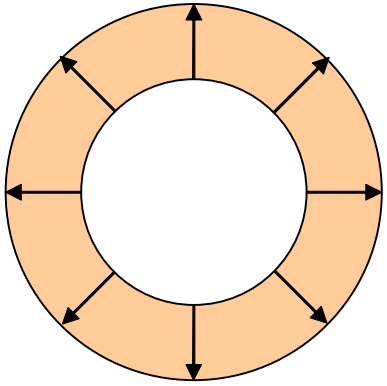
# (1) “Yes, But”综合症

- **“Yes, But”是软件开发中最经常遇到的问题，这种反应实际上是人的一种自然反应。**
  - 不管之前他多么认同你的设计，在没有看到真正的系统之前，用户决不可能完全理解你的设计；——软件本质上的“无形性”造成的必然结果
  - 机械设计里的每一步都是看的见摸得到的，用户从最开始就能与设计人员同步理解，所以不存在“Yes, But”问题
  - 在软件设计里，需求获取阶段的一个重要目标就是如何尽早的把“But”后面的部分发现出来



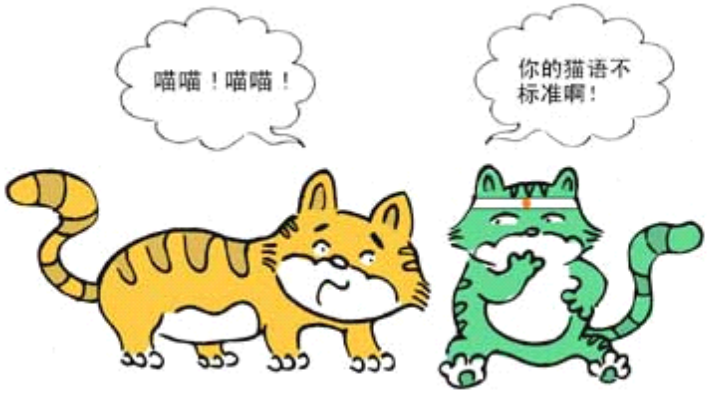
# (2) “Undiscovered Ruins”综合症

- “我们已经发现了所有的需求，现在让我们开始着手开发吧…”
- “知道的越多，不知道的也越多”
- **“Undiscovered Ruins”**：请问尚未被发现的废墟有多少呢？
- 需求是永无止境的



# (3) “User and Developer”综合症

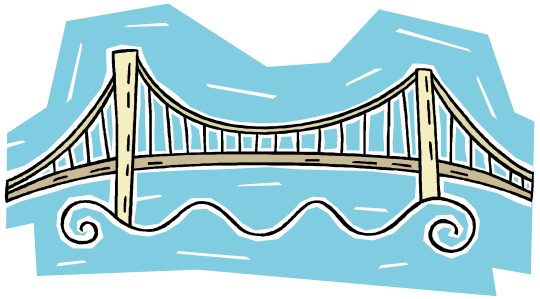
- 中国猫：喵喵~喵~
- 日本猫：涅呀~涅呀~




- 软件开发中，开发人员与用户处于不同的知识、技术层面，所关注的目标不同，双方在沟通时必然存在**communication gap(**交流的鸿沟**)**。

# (3) “User and Developer”综合症

- **“理解用户需求”**这一目标驱使软件开发人员从他们所沉溺的**“01”**世界转入到现实中的世界；
- 巨大的鸿沟**(gap)**导致开发人员与用户之间无法充分的相互理解；
- 为了在两个截然不同的世界之间架起一座桥梁，有必要学习一些技术以便于有效的获取和理解客户需求。



# 小结

| 问题 | 解决方案 |
| --- | --- |
| “Yes, But”综合症：直到开发人员将用户描述的东西交给他们，用户才认为他们知道自己要什么 | 尽早提供可选择的启发技术：应用用例、角色扮演、开发原型等方法 |
| “Undiscovered Ruins”综合症：用户不知道自己需要什么，或知道但不知如何表达 | 将用户当作领域专家来认识和激励，尝试其他交流和启发技术 |
| “User and Developer”综合症：分析员认为自己比用户更了解用户的需求 | 把分析员放在用户的位置上，试着角色扮演一小时或一天 |

# 本章内容

- **5.1** 需求获取的挑战
- **5.2** 需求获取的途径
- **5.3** 需求获取技术
- **5.4** 需求获取管理

# 需求获取途径

- **需求获取的关键：**
  - 沟通和交流
- **所要避免的问题：**
  - 交流障碍、沟通不全、意见冲突
- **所要必备的条件：**
  - 较高的技术水平、丰富的实践经验、较强的人际交往能力
- **可能采取的手段：**
  - 用户访谈、现场考察、专家咨询、会议讨论、…

# 需求获取途径

- 面对面访谈**(face-to-face interviewing)**
- 专题讨论会**(workshop)**
- 现场观察**(observing on the scene)**
- 头脑风暴**(brainstorming)**

- 重点注意业务单据等资料的收集、整理！！---分析的依据
- 多种方法要复合在一起使用，效果更好

# 面对面访谈

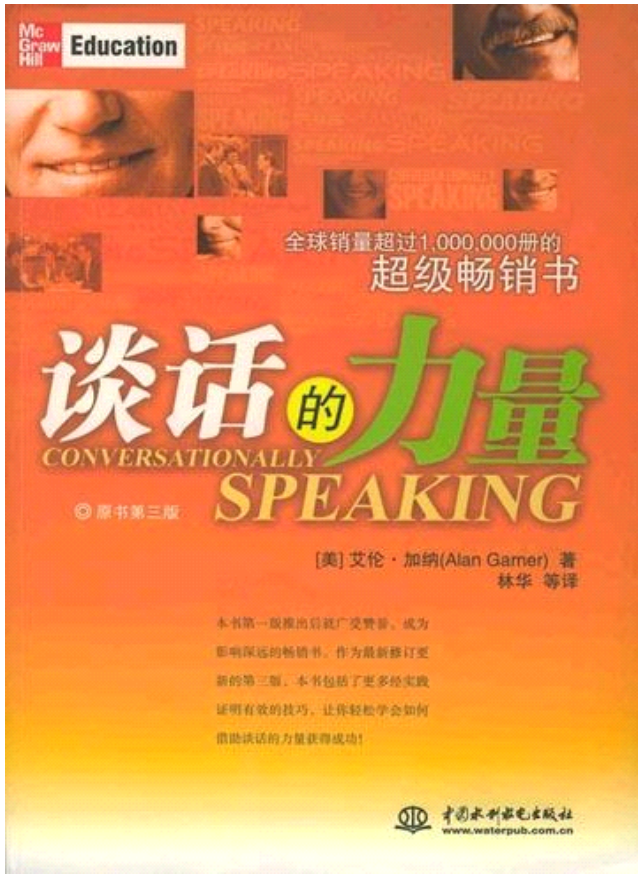
- 需求获取中最直接的方法：用户面谈**(interviewing)**
- “看起来很美”，但“做起来并不容易”

——需求分析者个人的偏见、事先的理解、以往的经验积累是导致面谈失败的最重要原因

——在面谈时，忘掉一切以往所作的事情，通过问题启发，倾听对方的陈述

——不要把自己放在“专家”的位置上

# 如何提问？




- “每个人都能提问题，但并不等于人人都会提问题…”
- 封闭式问题：
  - 对错判断或多项选择题，回答只需要一两个词
- 开放式问题：
  - 这种问题需要解释和说明，同时向对方表示你对他们说的话很感兴趣，还想了解更多的内容。
- 通过提问题增强你对谈话进展和方向的控制
- 问题不能过于宽泛
- 最开始的问题不能太难
- 不能在提问之前就已经表示不赞同
- 谈话之前有意识的准备一些备用问题

# 访谈问题的分类

- 上下文无关的问题**(context-free questions)**：充分理解用户的问题，不涉及具体的解决方案
  - 客户是谁？
  - 最终用户是谁？
  - 不同用户的需求是否不同？
  - 这种需求目前的解决方案是什么？
- 解决方案相关的问题**(solution-context questions)**：通过这类问题，探寻特定的解决方案并得到用户认可
  - 你希望如何解决这个问题？
  - 你觉得该问题这样解决如何？

# 面谈之前

- 确立面谈目的
- 确定要包括的相关用户
- 确定参加会议的项目小组成员
- 建立要讨论的问题和要点列表
- 复查有关文档和资料
- 确立时间和地点
- 通知所有参加者有关会议的目的、时间和地点

# 面谈之中

- **Step 1**：事先准备一系列上下文无关的问题，并将其记录下来以便面谈时参考；
- **Step 2**：面谈前，了解一下要面谈的客户公司的背景资料，不要选择自己能回答的问题而浪费时间；
- **Step 3**：面谈过程中，参考事先准备的面谈模板，以保证提出的问题是正确的。将答案记录到纸面上，并指出和记录下未回答条目和未解决问题；
- **Step 4**：面谈之后，分析总结面谈记录。

# 面谈之后

- 复查笔记的准确性、完整性和可理解性
- 把所收集的信息转化为适当的模型和文档
- 确定需要进一步澄清的问题域
- 向参加会议的每一个人发出此次面谈的**minutes(**会议纪要**)**

# 面对面访谈的优缺点分析

- 优点：
  - 人们很愿意谈论自己的工作，并且总是很喜欢接受访谈；
- 缺点：
  - 大多数人都采用专业术语和“行话”，而太多的专业术语让需求工程师难以理解，往往造成很多误解；
  - 有些需求对用户来说太普通了，以至于他们不自觉地认为这些需求太基本，不值得去提。但它们对需求工程师来说却不是显而易见的。这往往会造成某些需求被忽略；

# 本章内容

- **5.1** 需求获取的挑战
- **5.2** 需求获取的途径
- **5.3** 需求获取技术
  - 5.3.1 业务调研
  - 5.3.2 基于用例的需求获取
- **5.4** 需求获取管理

# 业务调研

- 调研准备
  - 了解背景、领域知识
  - 确定调研组织、对象
  - 准备调研提纲，制定调研计划
  - 设计调研问卷
- 调研过程
  - 访谈、记录
  - 填写问卷
  - 整理调研材料（思考，回访）
- 撰写调研报告
- 确认调研结果

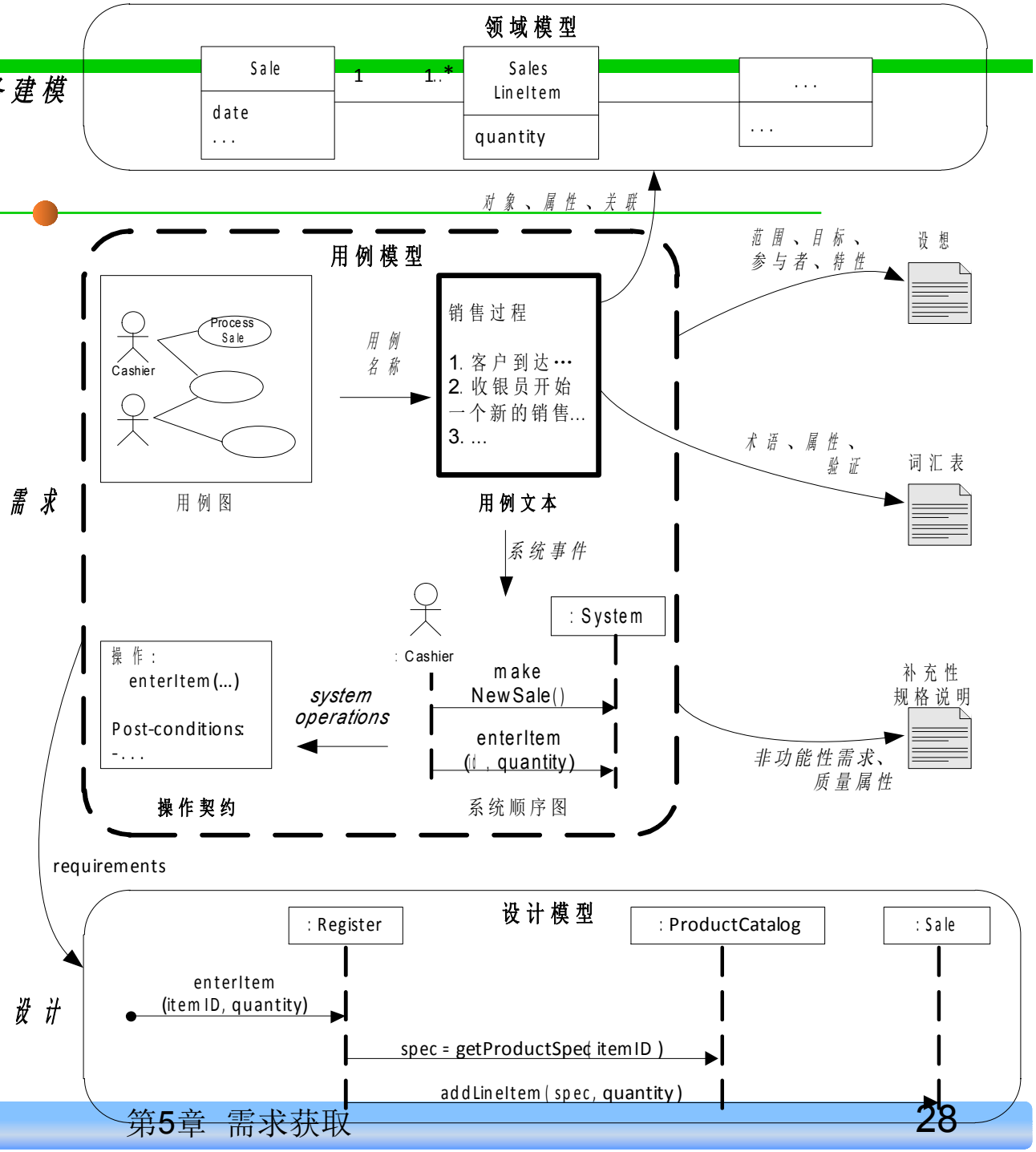
# 业务调研案例

- 河南同力水泥集团信息系统工程
  - 调研问卷
  - 调研计划
  - 调研资料
  - 调研报告

# 本章内容

- **5.1** 需求获取的挑战
- **5.2** 需求获取的途径
- **5.3** 需求获取技术
  - 5.3.1 业务调研
  - 5.3.2 基于用例的需求获取
- **5.4** 需求获取管理

# 基于用例的需求获取


领域模型
业务建模
Sale
date
...
1
1..*
Sales LineItem
quantity
...
...
对象、属性、关联
用例模型
Process Sale
Cashier
用例图
用例名称
销售过程
1. 客户到达…
2. 收银员开始一个新的销售...
3. ...
用例文本
范围、目标、参与者、特性
设想
术语、属性、验证
词汇表
需求
系统事件
: System
: Cashier
操作：
enterItem(...)
Post-conditions:
-...
system operations
make NewSale()
enterItem (it , quantity)
操作契约
系统顺序图
补充性规格说明
非功能性需求、质量属性
requirements
设计模型
: Register
: ProductCatalog
: Sale
设计
enterItem (itemID, quantity)
spec = getProductSpec( itemID )
addLineItem( spec, quantity)

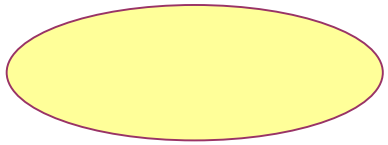
# 基本概念

- 参与者(actor)：是某些具有行为的事物，可以是人（由角色标识）、计算机系统或者组织，例如 收银员
- 场景(scenario)：是参与者和系统之间的一系列特定的活动交互，也称为用例实例(use case instance)
- 用例(use case)：就是一组相关的成功和失败场景集合，用来描述参与者如何使用系统来实现其目标

- 用例是文本文档，而非图形；
- 用例建模主要是编写文本的活动，而非制图。

# 用例的特征

- 用例：站在用户角度定义软件系统的外部特征
- 四大特征：
  - 行为序列(sequences of actions)：一个用例由一组可产生某些特定结果的行为构成，这些行为是不可再分解的(接收用户输入、执行、产生结果)
  - 系统执行(system performs)：系统为外部角色提供服务；
  - 可观测到的、有价值的结果(observable result of value)：用例必须对用户产生价值；
  - 特定的角色(particular actor)：某人、某台设备、某外部系统、等等，能够触发某些行为。



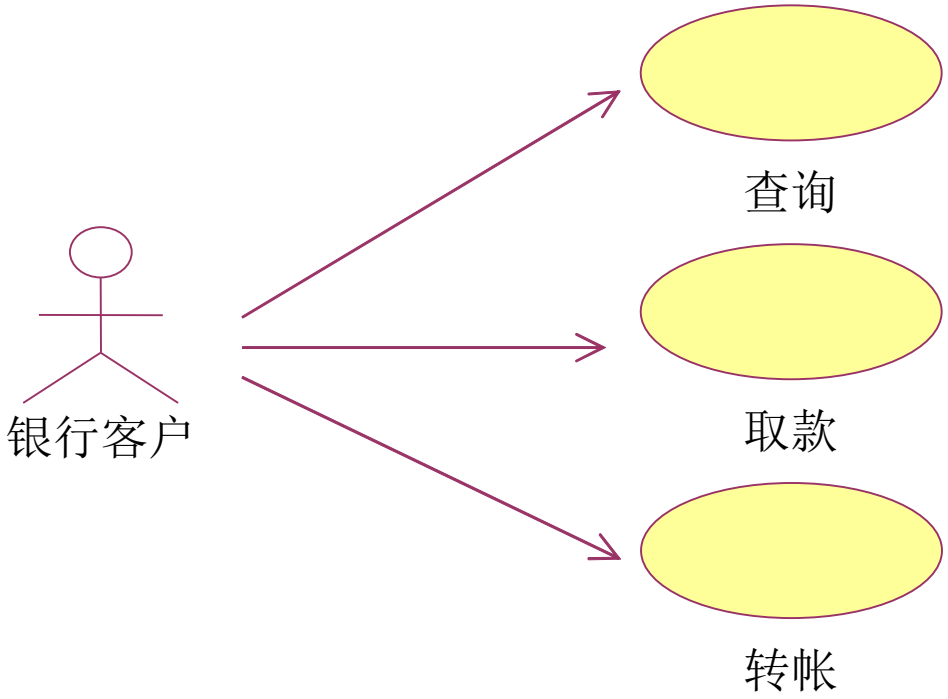
Use case

# 用例方法的基本思想

- 用例方法的基本思想：从用户的角度来看，他们并不想了解系统的内部结构和设计，他们所关心的是系统所能提供的服务，也就是被开发出来的系统将是如何被使用的。
- 用例模型主要由以下模型元素构成：
  - 参与者(Actor)：存在于被定义系统外部并与该系统发生交互的人或其他系统，代表系统的使用者或使用环境。
  - 用例(Use Case)
  - 通讯关联(Communication Association)：用于表示参与者和用例之间的对应关系，它表示参与者使用了系统中的哪些服务(用例)、系统所提供的服务(用例)是被哪些参与者所使用的。

# 示例：ATM系统的用例

- 参与者：银行客户
- 用例：银行客户使用自动提款机来进行银行帐户的查询、取款和转帐交易

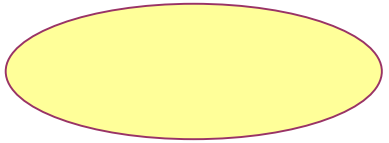


# 关于“通讯关联”的几点说明

- **通讯关联表示的是参与者和用例之间的关系：**
  - 箭头表示在这一关系中哪一方是对话的主动发起者，箭头所指方是对话的被动接受者；
  - 如果不想强调对话中的主动与被动关系，可以使用不带箭头的关联实线。
  - 通讯关联不表示在参与者和用例之间的信息流，并且信息流向是双向的，它与通讯关联箭头所指的方向没有关系。

# 用例的内部剖析

- 用例= 椭圆 + 名字？—— **NO**！
- 用例=文本！



Use case

用例名：
参与者及关注点：
主成功场景：
　事件1
　事件2
　……
扩展
前置条件：
后置条件：
……

# 如何发现用例

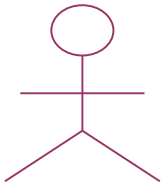
- **Step 1**：确定系统边界
- **Step 2**：识别并描述参与者**(actor)**；
- **Step 3**：确定每个参与者目标，识别用例**(use case)** ；
- **Step 4**：识别参与者与用例之间的通讯关联**(Association)**；
- **Step 5**：给出每一个用例的详细描述
- **Step 6**：细化用例模型

# Step 1：确定系统边界

- 系统目标
- 系统范围

# Step 2：识别并描述参与者

- **通过以下问题来识别Actor：**
  - 谁使用这个系统的功能？
  - 谁从该系统获得信息？
  - 谁向该系统提供信息？
  - 该系统需要访问(读写)那些外部硬件设备？
  - 谁来负责维护和管理这个系统以保证其正常运行？
  - 该系统需要与其他系统进行交互吗？

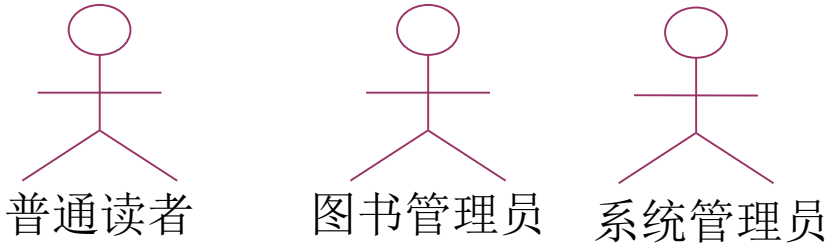


参与者

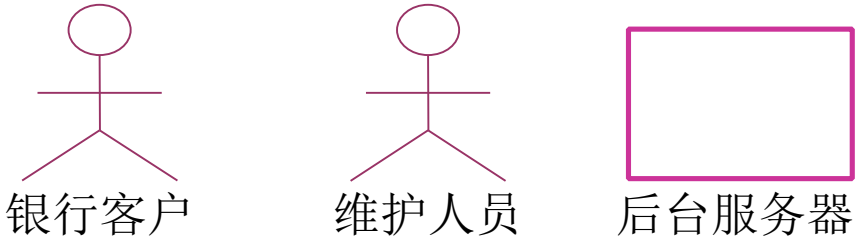
其它系统

# 识别并描述参与者

- **例1：对一个图书馆管理系统来说，有哪些参与者？**
  - 普通读者
  - 图书管理员
  - 系统管理员




- **例2：对ATM系统来说，有哪些参与者？**
  - 银行客户
  - ATM维护人员
  - 后台服务器

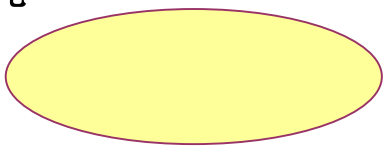


# 特殊的参与者：系统时钟

- 有时候需要在系统内部定时的执行一些操作，如检测系统资源使用情况、定期生成统计报表等等；
- 但这些操作并不是由外部的人或系统触发的；
- 对于这种情况，可以抽象出一个系统时钟或定时器参与者，利用该参与者来触发这一类定时操作；
- 从逻辑上，这一参与者应该被理解成是系统外部的，由它来触发系统所提供的用例对话。

# Step 3：识别用例(use case)

- 找到参与者之后，据此来确定系统的用例，主要是看各参与者需要系统提供什么样的服务，或者说参与者是如何使用系统的。
- 寻找用例可以从以下问题入手(针对每一个参与者)：
  - 参与者使用该系统执行什么任务？
  - 参与者是否会在系统中创建、修改、删除、访问、存储数据？如果是的话，参与者又是如何来完成这些操作的？
  - 参与者是否会将外部的某些事件通知给该系统？
  - 系统是否会将内部的某些事件通知该参与者？



Use case

# 识别用例（目标识别）

- **例1：对图书馆管理系统来说，有哪些用例？**
  - 图书管理员
    - 管理读者信息
    - 管理图书信息
    - 登记借书
    - 登记还书
  - 普通读者：
    - 预订图书
    - 取消预订
    - 查询浏览图书信息
- **例2：对ATM系统来说，有哪些参与者？**
  - 银行客户
    - 查询
    - 取款
    - 转装
  - ATM维护人员
    - 维护系统
  - 后台服务器
    - 周期性操作

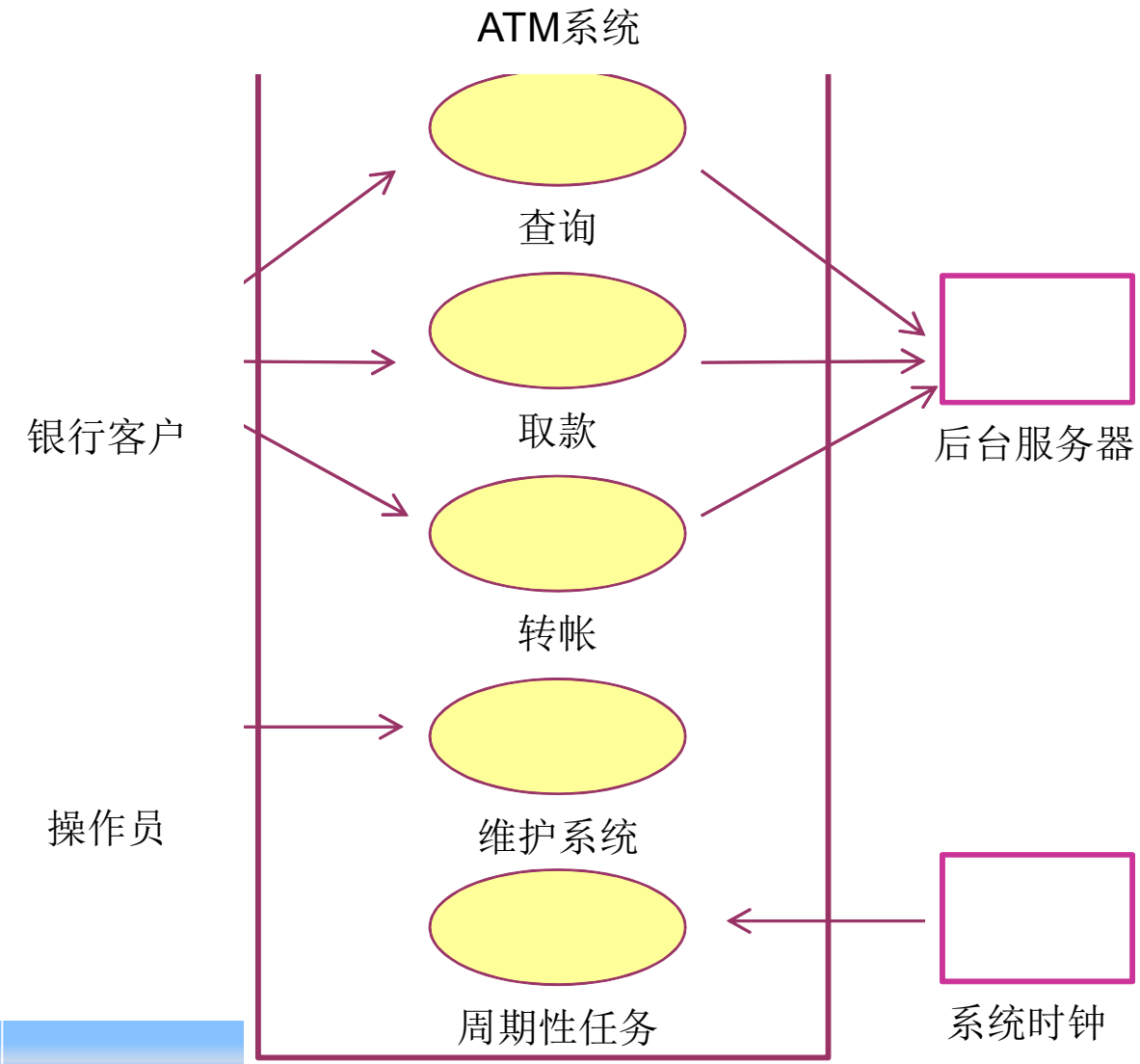
# 识别用例的几点注意事项

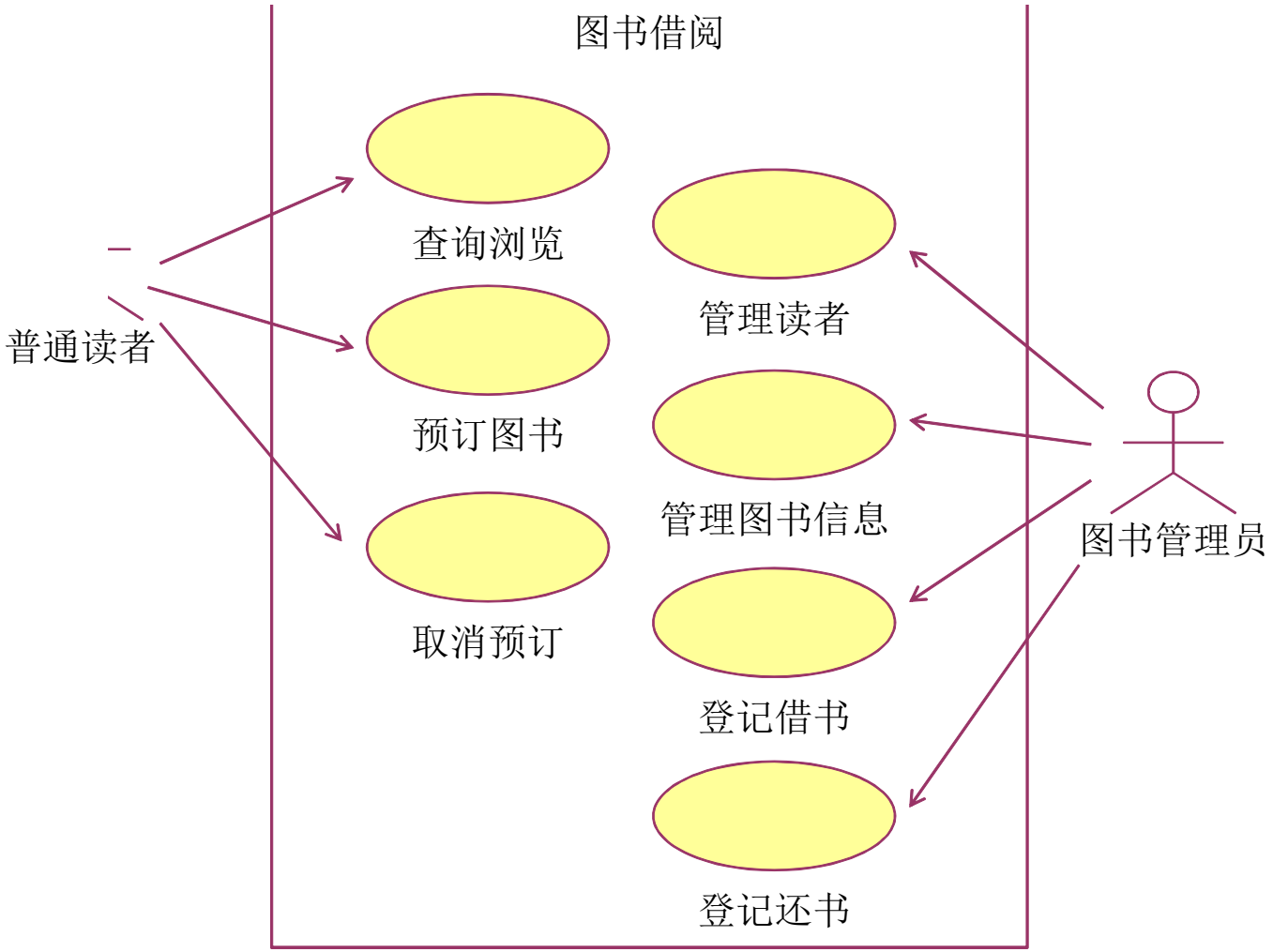
- 用例必须是由某一个**actor**触发而产生的活动，即每个用例至少应该涉及一个**actor**。
- 如果存在与**actor**不进行交互的用例，需要将其并入其他用例，或者是检查该用例相对应的参与者是否被遗漏。
- 反之，每个参与者也必须至少涉及到一个用例，如果发现有不与任何用例相关联的参与者存在：
  - 仔细考虑该参与者是如何与系统发生对话的；
  - 由参与者确定一个新的用例；
  - 该参与者是一个多余的模型元素，应该将其删除。

# 发现有用的用例

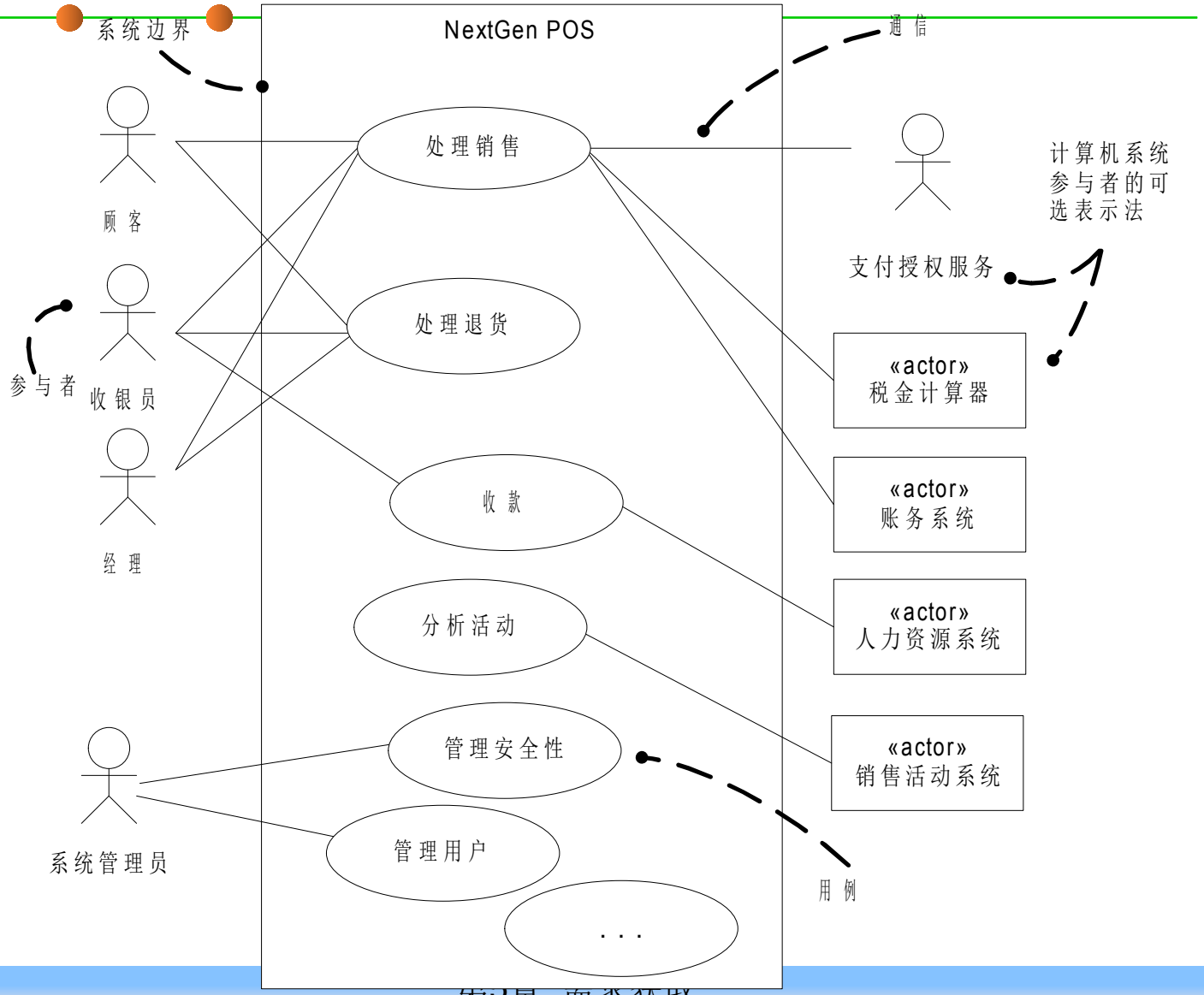
- 下面那个是有效用例？
  - 就供应者合同进行协商
  - 处理退货
  - 登录
  - 将商品进行条码扫描
- 老板测试
- 基本业务过程测试
  - 基本业务过程：一个人于某个时刻在一个地点所执行的任务，用以响应业务事件。该任务能够增加可量化的业务价值，并且以持久状态留下数据。
- 规模测试
  - 用例由一系列相关联的步骤组成，不能是单独的活动！

# Step 4：识别参与者与用例之间的通讯关联



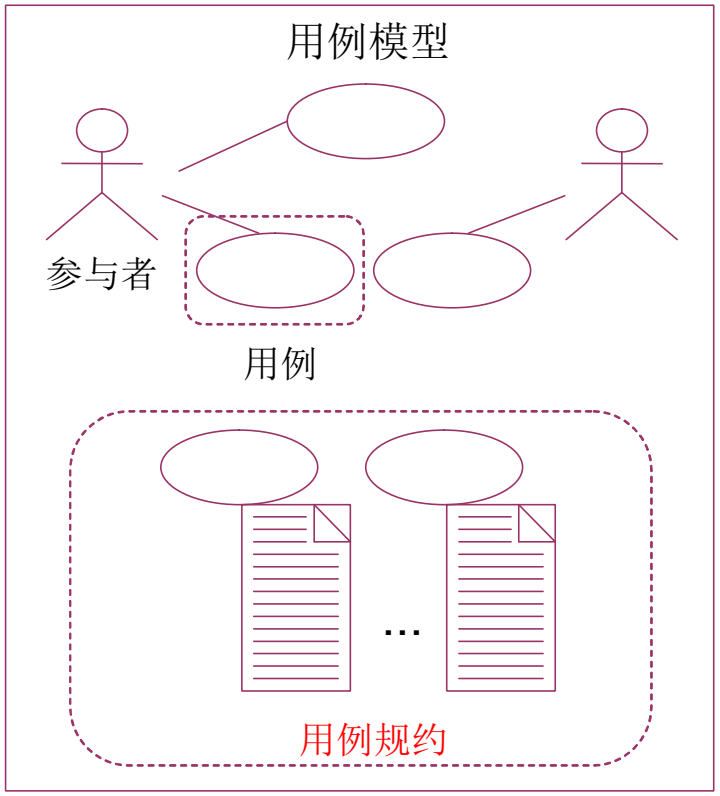

图书借阅
查询浏览
预订图书
取消预订
管理读者
管理图书信息
登记借书
登记还书
普通读者
图书管理员

# 用例图示例


系统边界
NextGen POS
通信
顾客
处理销售
计算机系统参与者的可选表示法
支付授权服务
处理退货
参与者
收银员
«actor»
税金计算器
«actor»
账务系统
收款
经理
分析活动
«actor»
人力资源系统
«actor»
销售活动系统
管理安全性
系统管理员
管理用户
用例
. . .

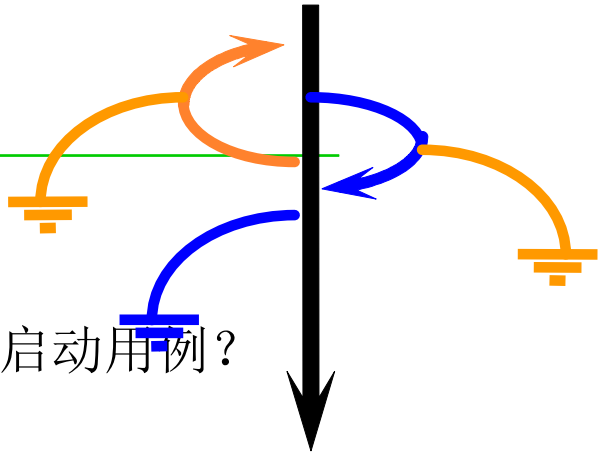
# Step 5：给出用例的详细描述

- 单纯的用例图并不能描述完整的信息，需要用文字描述不能反映在图形上的信息。




用例名：
参与者及关注点：
主成功场景：
　事件1
　事件2
　……
扩展
前置条件：
后置条件：
……

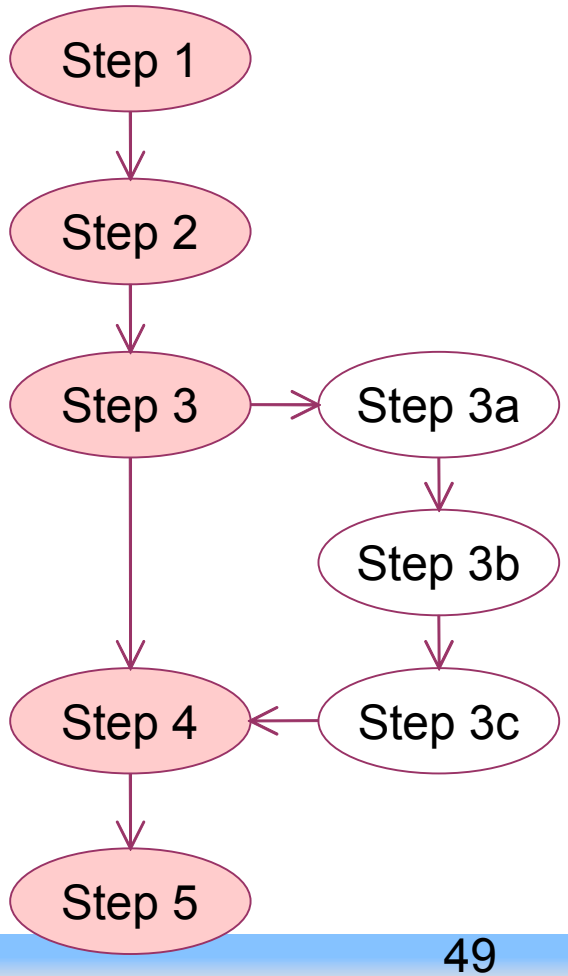
# 事件流



- **用例的事件流：**
  - 说明用例如何启动，即哪些参与者在何种情况下启动用例？
  - 说明参与者与用例之间的信息处理过程；
  - 说明用例在不同条件下可以选择执行的多种方案；
  - 说明用例在什么情况下才能被视作完成；
- **分为常规流和扩展流两类：**
  - 常规流：描述该用例最正常的一种场景，系统执行一系列活动步骤来响应参与者提出的服务请求；
  - 扩展流：负责描述用例执行过程中异常的或偶尔发生的一些情况。

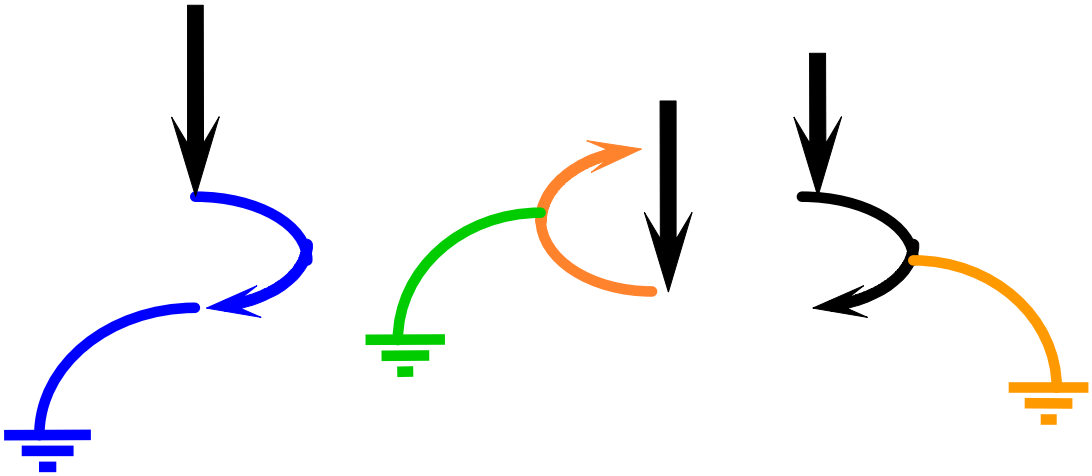
# 常规事件流

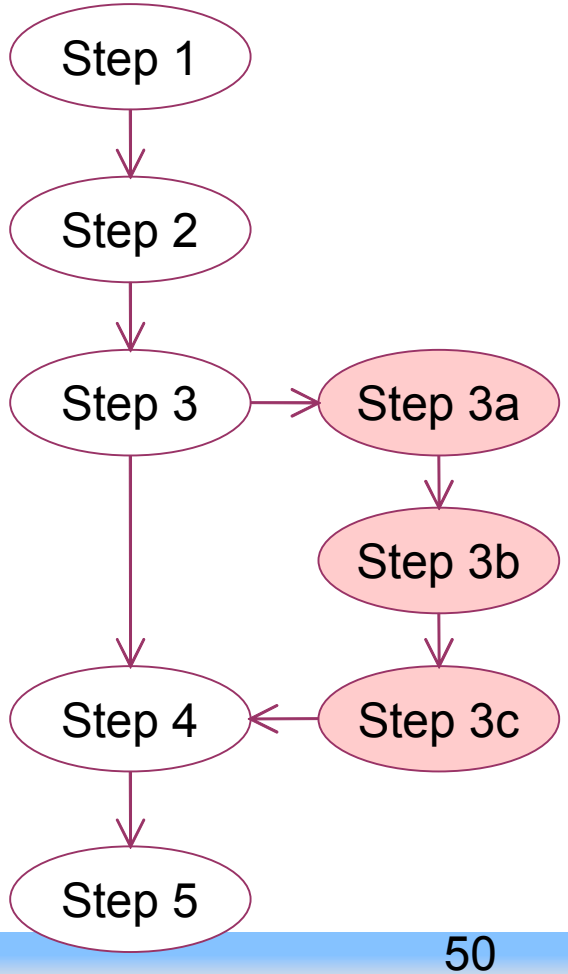
- 每一个步骤都需要用数字编号以清楚地标明步骤的先后顺序
- 用一句简短的标题来概括每一步骤的主要内容
- 对每一步骤，从正反两个方面来描述
  - 参与者向系统提交了什么信息
  - 对此系统有什么样的响应



# 扩展事件流

- **扩展流的描述格式可以与基本流的格式一致，也需要编号并以标题概述其内容。**
  - 起点：该扩展流从事件流的哪一步开始；
  - 条件：在什么条件下会触发该扩展流；
  - 动作：系统在该扩展流下会采取哪些动作；
  - 恢复：该扩展流结束之后，该用例应如何继续执行。





# 编写用例文本的准则

- 以无用户界面约束的本质风格编写用例
- 编写简洁的用例
- 编写黑盒用例
- 采用参与者和参与者目标的视角

# [案例]用例描述1

## 用例：登记借书

**1.** 目标：

本用例允许图书管理员登记普通读者的借书记录

**2** 事件流：

**2.1** 常规流程

当读者希望借书、图书管理员准备登记有关的借书记录时，本用例开始执行。

(1) 系统要求管理员输入读者的注册号和所借图书号；
(2) 图书管理员输入信息后，系统产生一个唯一的借书记录号；
(3) 系统显示新生成的借书记录；
(4) 图书管理员确认后，系统增加一个新的借书记录

**2.2** 扩展流程

**(1)** 读者没有注册

在主流程中，如果系统没有读者的注册信息，系统将显示错误信息，用例结束；

**(2)** 所借图书不存在

在主流程中，如果所借图书已被借出或者系统中无该图书，系统将显示错误信息，用例结束。

**3** 前提条件：用例开始前，图书管理员必须在系统登录成功；

**4** 后置条件：如果用例执行成功，该读者的借书记录被更新，否则，系统状态不变。

# [案例]用例描述2

**用例名称：**处理销售

**参与者与关注点：**

●收银员：希望准确、快速地输入，而且没有支付错误，因为如果少收货款，将从其工资中扣除。

●… …

**前置条件：**收银员必须经过确认和认证

**成功保证（或后置条件）：**存储销售信息。准确计算税金。更新账务和库存信息。

**主成功场景（或基本流程）：**

1.顾客携带所购商品或服务到收银台通过POS机付款。

2.收银员开始一次新的销售交易。

3.收银员输入商品条码

**扩展（或替代流程）：**

3a.无效商品ID（在系统中未发现）：

系统提示错误并拒绝输入该ID。

收银员响应该错误

**特殊需求：**

●使用大尺寸平面显示器触摸屏，文本信息可见距离为1米

●。。。

**发生频率：**可能会不断地发生

**未解决问题：**

提成处理规则不确定

收银员换班时如何处理

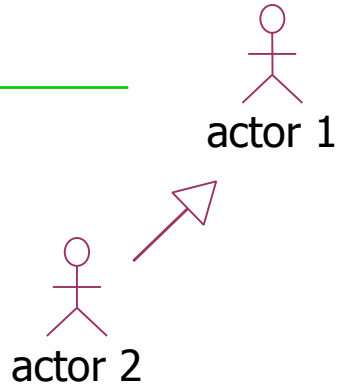
… …

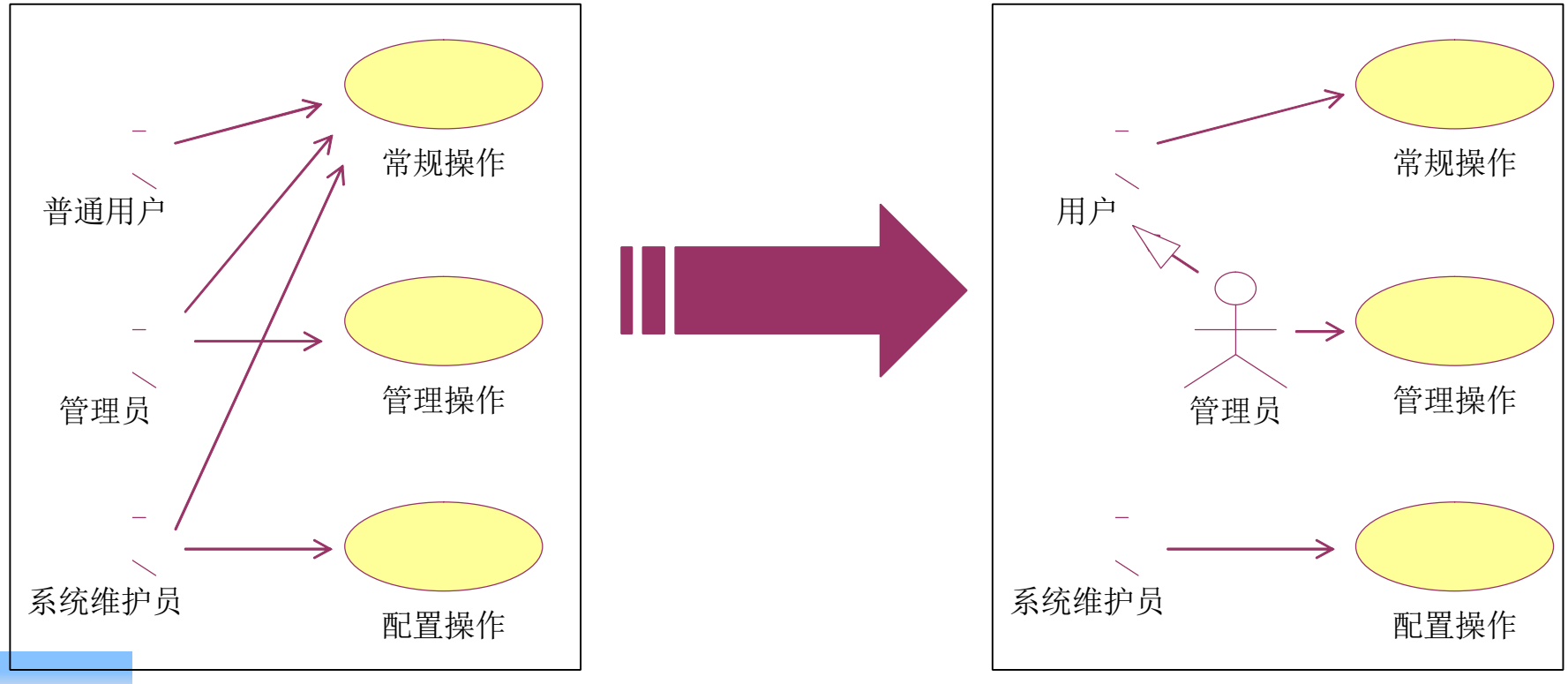
# Step 6：细化用例模型

1. 在一般的用例图中，只需表述参与者和用例之间的通讯关联，除此之外，还可以描述：
   - 参与者与参与者之间的泛化(generalization)
   - 用例和用例之间的包含(include)
   - 用例和用例之间的扩展(extend)
   - 用例和用例之间的泛化(generalization)关系
2. 根据用例描述绘制活动图
3. 补充非功能性需求

# 参与者之间的关系

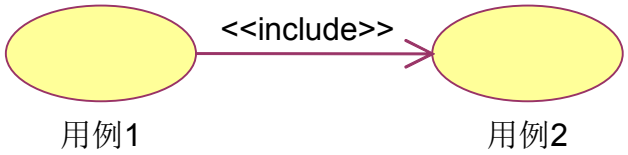
- 参与者之间可以有泛化**(Generalization)**关系。

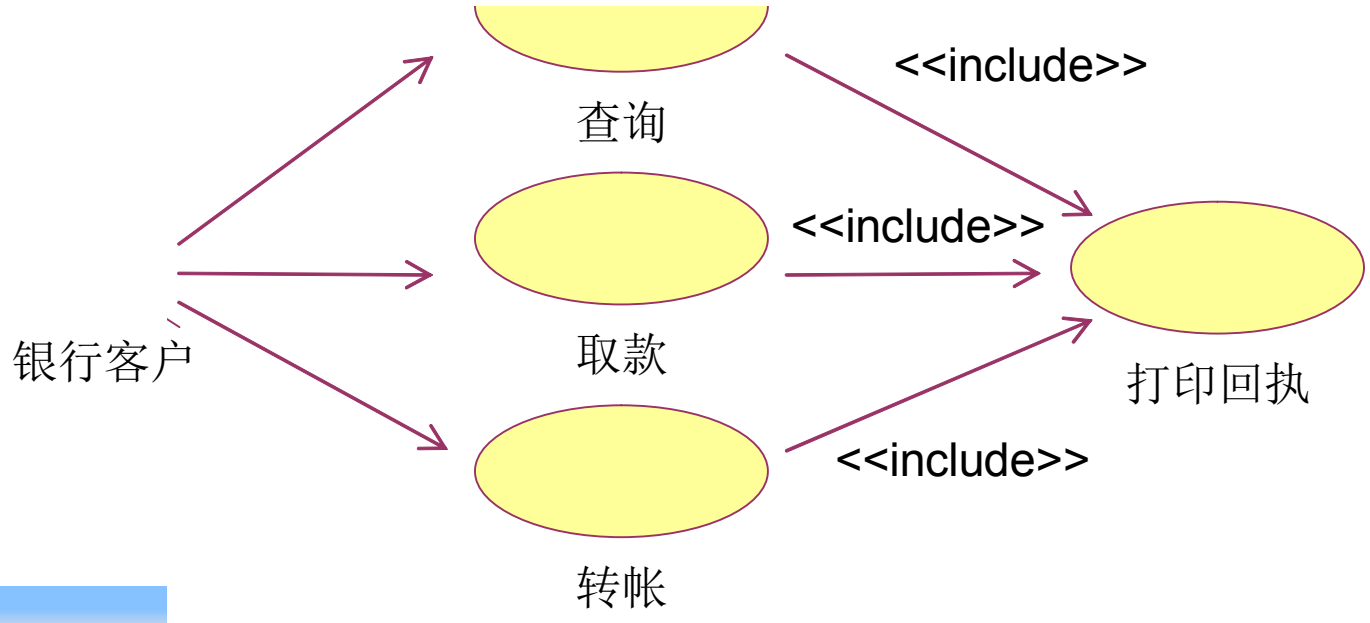


# 用例之间的关系：包含(include)

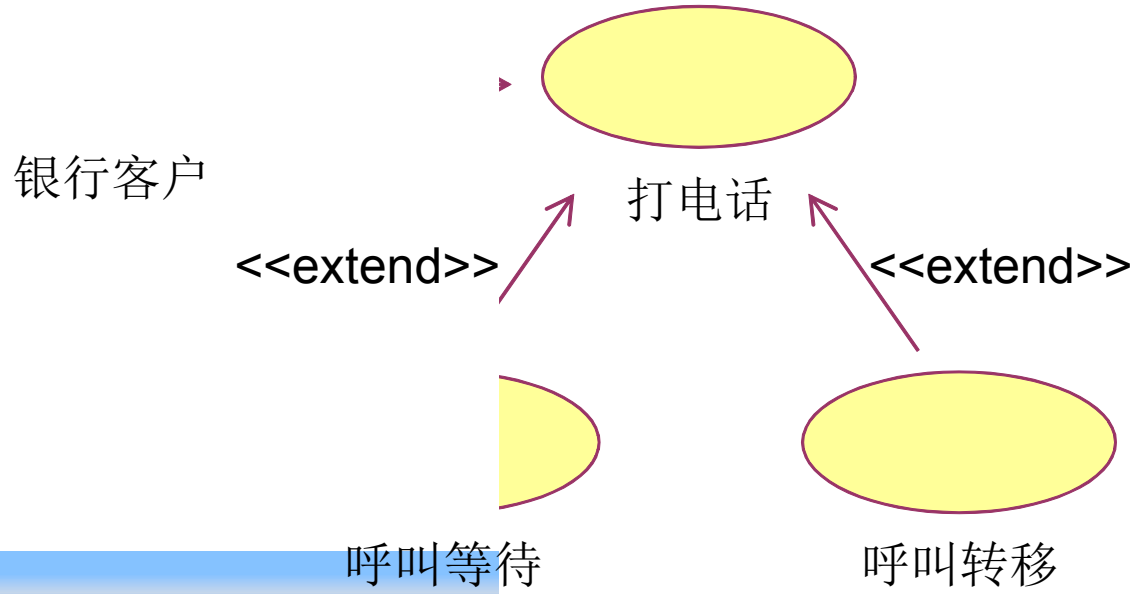
- “包含关系”是通过在关联关系上加入**<<include>>**标记来表示；
- 语义：用例**1**会用到用例**2**，用例**2**的事件流将被插入到用例**1**的事件流中
- 一般表示为公共功能



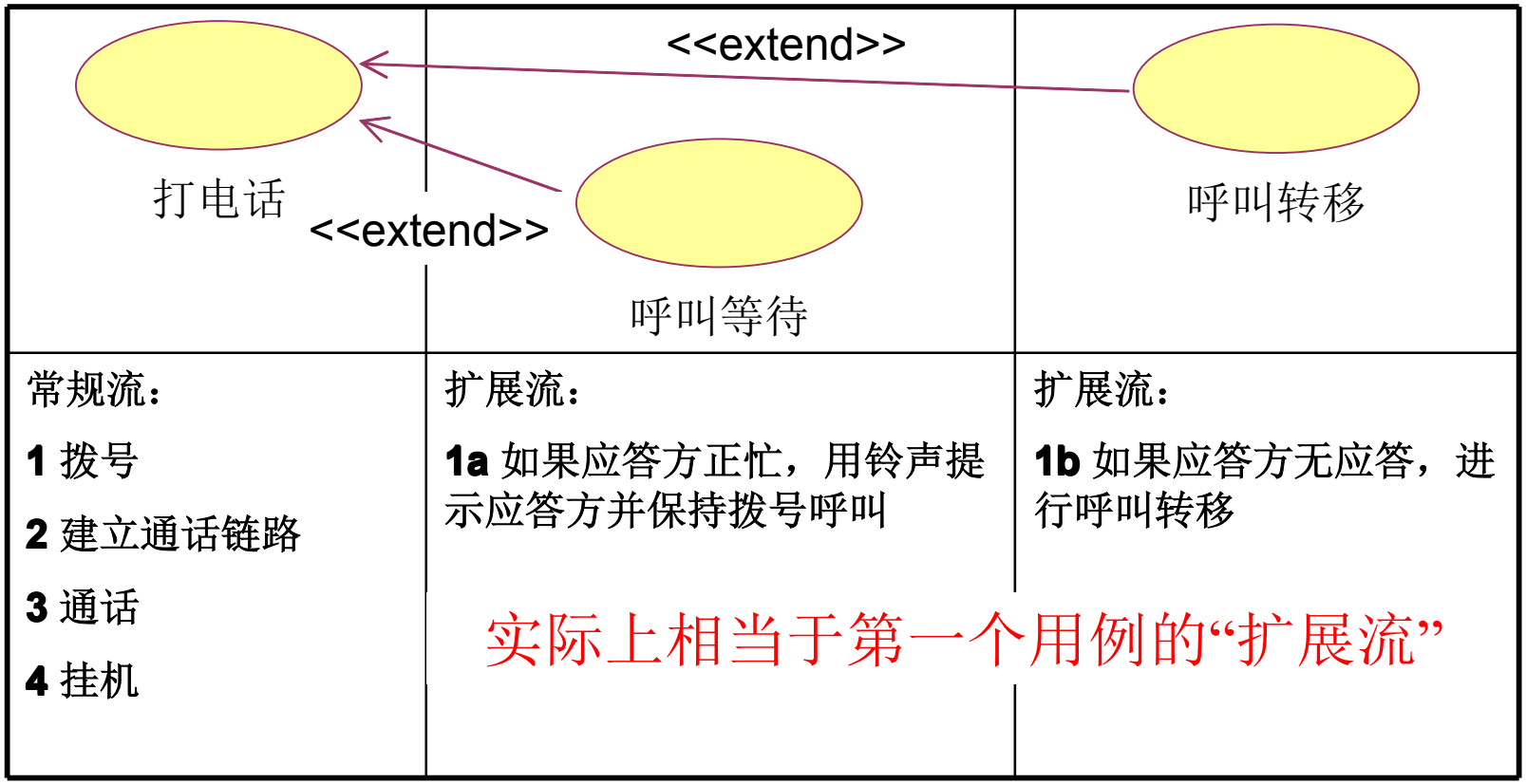



# 用例之间的关系：扩展(extend)

- “扩展关系”是通过在关联关
- 语义：用例2在某些特定情
  被插入到用例2的事件流中。
- 一般表示为异常功能

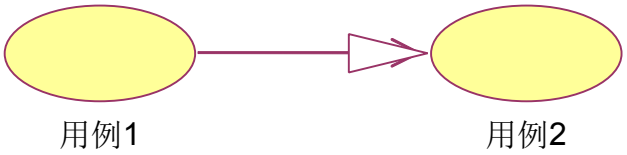


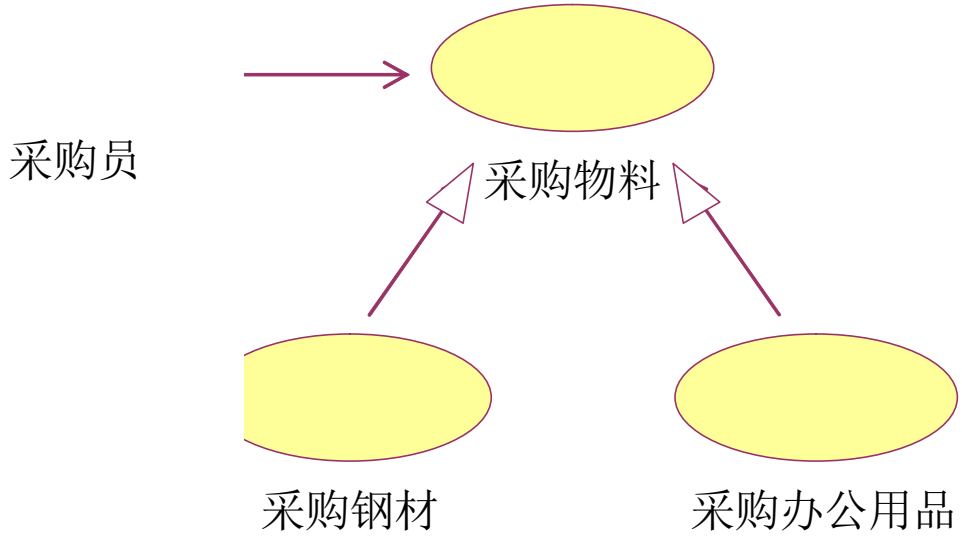
# 用例之间的关系：扩展(extend)



# 用例之间的关系：泛化(generalization)

- 当多个用例共同
  抽象成为父用例
- 子用例继承了父



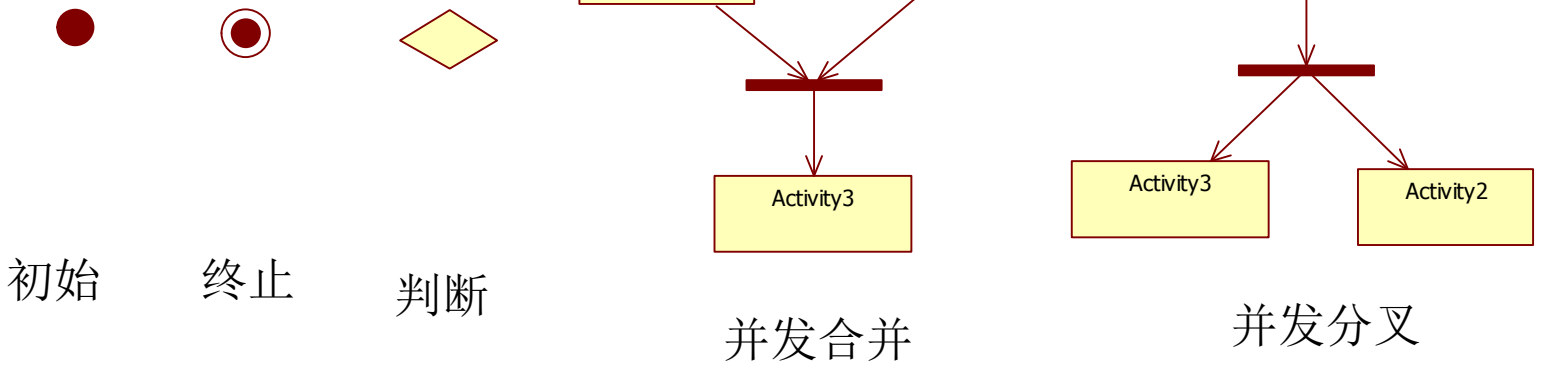
# “用例的关系”

- 参与者与参与者之间的泛化**(generalization)**
- 用例和用例之间的包含**(include)**
- 用例和用例之间的扩展**(extend)**
- 用例和用例之间的泛化**(generalization)**关系

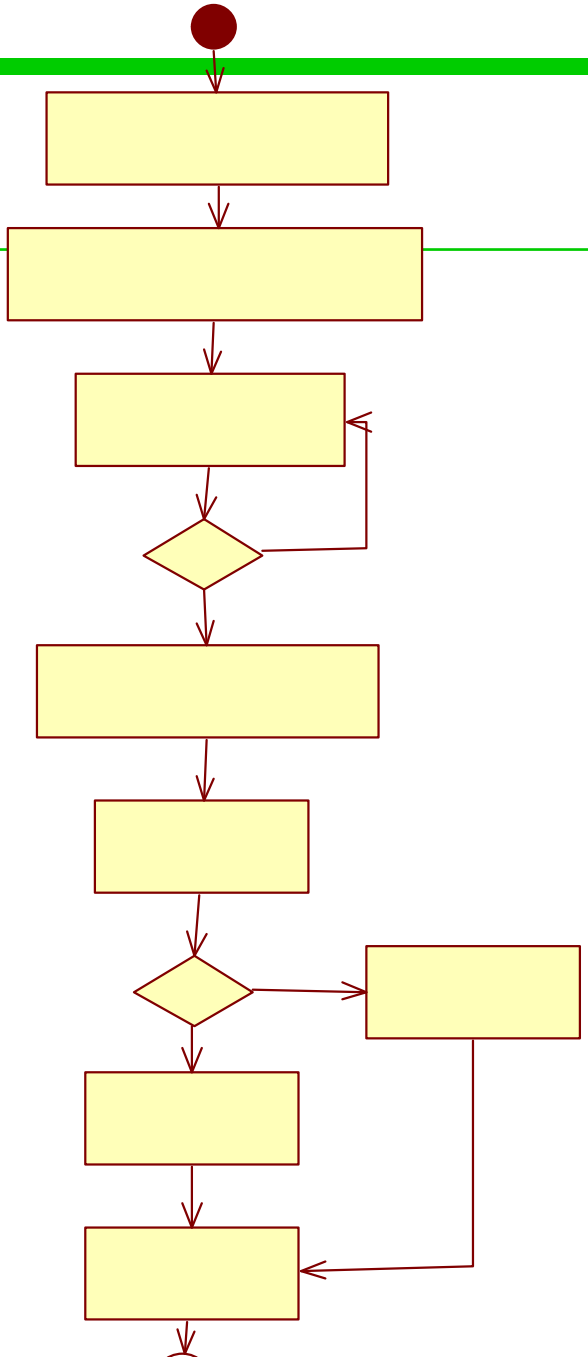
- 为什么要引入上述关系？有什么优越性？

# 活动图

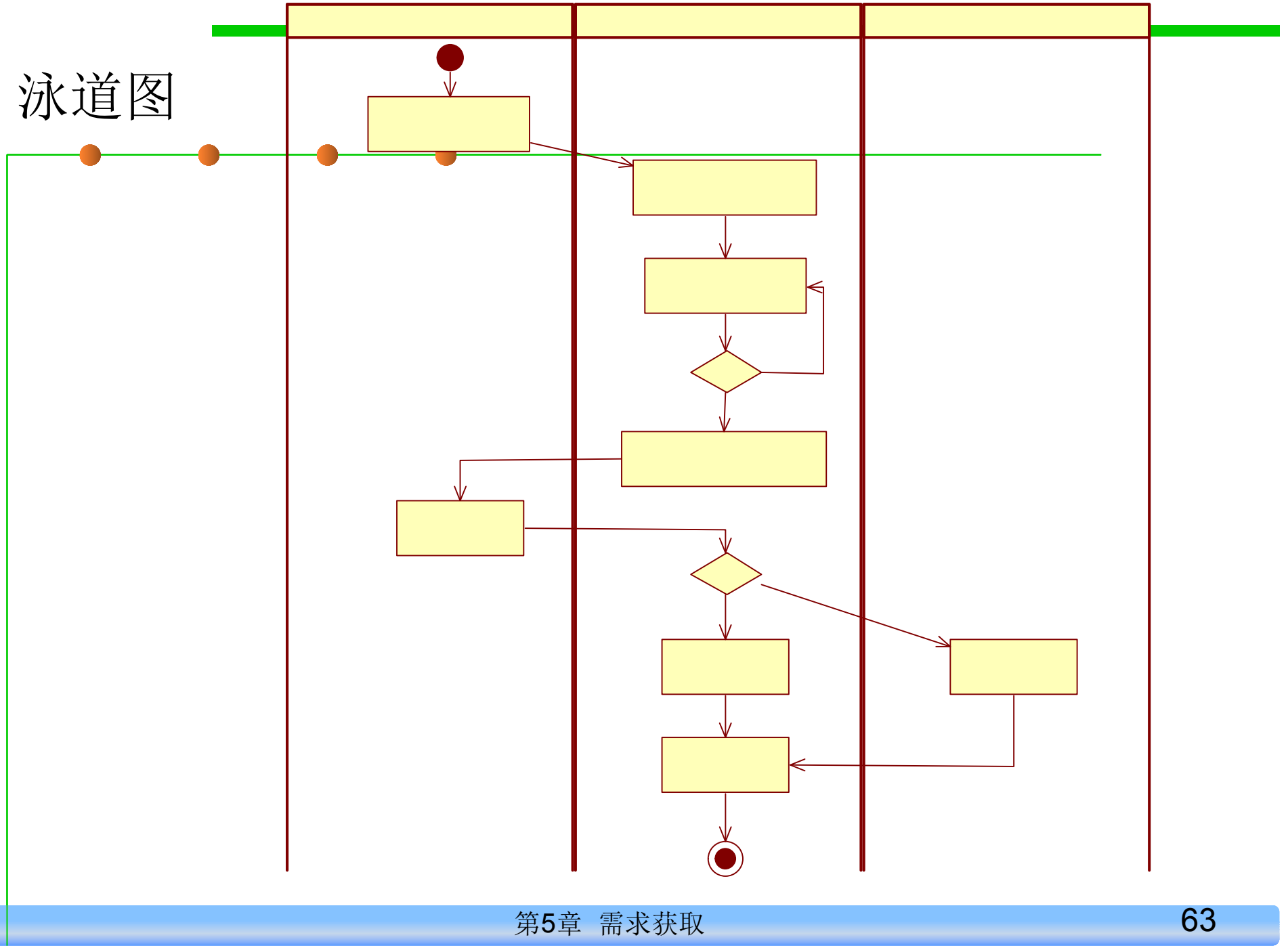
- 显示了组成复杂过程的步骤序列，例如算法或工作流
- 活动图用于详细描述用例



# 活动图描述



# 泳道图



# 标识非功能性需求（URPS+）

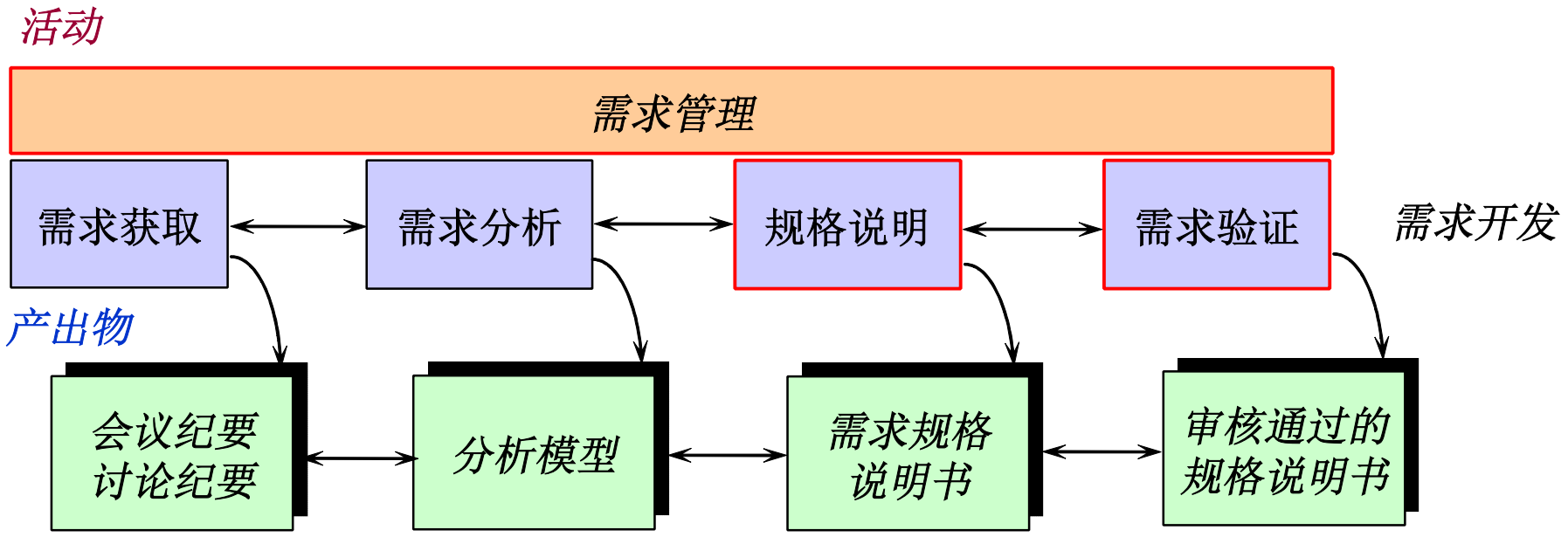
- 可用性
- 可靠性
- 性能
- 可支持性
- 实现
- 接口

（参见示例）

# 本章内容

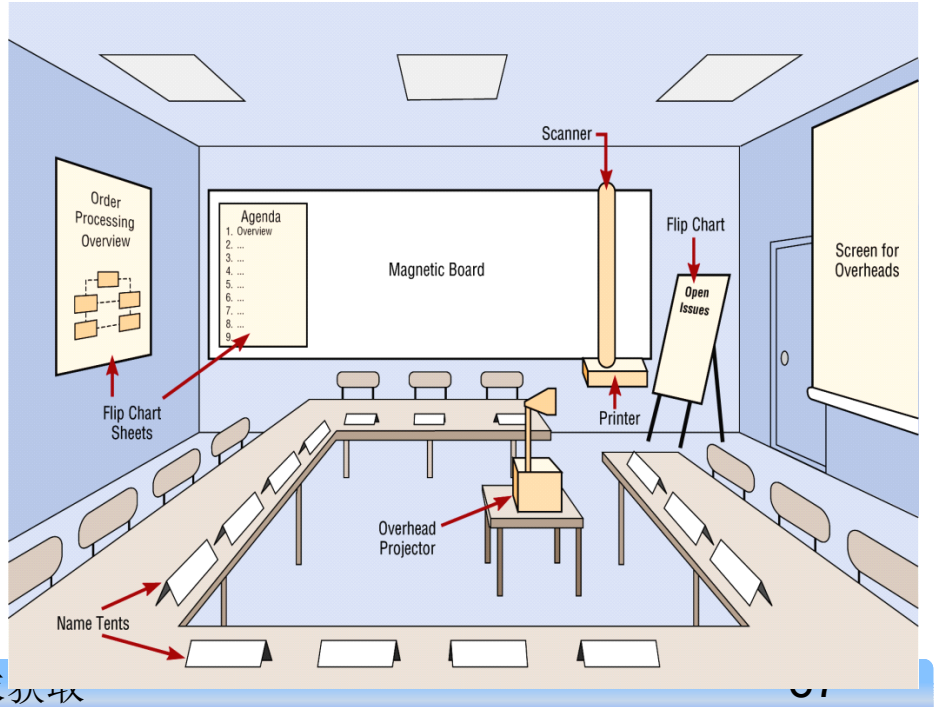
- **5.1** 需求获取的挑战
- **5.2** 需求获取的途径
- **5.3** 需求获取技术
- **5.4** 需求获取管理
  - 5.4.1 需求规格说明
  - 5.4.2 需求验证
  - 5.4.3 需求变更

# 需求工程的总体流程


活动
需求管理
需求获取
需求分析
规格说明
需求验证
需求开发
产出物
会议纪要
讨论纪要
分析模型
需求规格
说明书
审核通过的
规格说明书

# 与客户协商规格说明
# --联合应用设计(Joint Application Design, JAD)

- 通过让所有相关人员一起参加某个单一会议来定义需求或设计系统。
- 系统相关者在短暂而紧凑的时间段内集中在一起，一般为**1**至**2**天，与会者可以在应用需求上达成共识、对操作过程尽快取得统一意见。
- 协助建立一支高效团队，围绕一个目的：项目的成功；
- 所有人员都畅所欲言；
- 促进用户与开发团队之间达成共识；
- 能够揭露和解决那些妨碍项目成功的行政问题；
- 最终很快产生初步的系统定义。



# 需求规格说明书(SRS)

- **软件需求规格说明书SRS (Software Requirements Specification)：**
  - 需求开发的结果，精确的阐述一个软件系统必须提供的功能和性能，以及它所要考虑的限制条件。
  - 为了使用户和软件开发者双方对该软件的初始规定有一个共同的理解，使之成为整个开发工作的基础。

# 需求规格说明书(SRS)

- **SRS作为软件开发各类人员之间进行理解和交流的手段：**
  - 用户：通过SRS指定需求，检查需求描述是否满足期望；
  - 设计人员：了解软件需要开发的内容，将其作为软件设计的基本出发点；
  - 测试人员：指定测试计划、测试用例和测试过程；
  - 产品发布人员：编写用户手册和帮助信息；
  - 项目管理人员：规划软件开发过程、准确估计开发进度和成本、控制需求变更过程。

# SRS应包含的内容

- **功能(Functionality)：**
  - 该软件系统能够向用户提供何种服务？
- **非功能属性(Non-Functional Attributes)：**
  - 可用性
  - 可靠性
  - 性能
  - 可支持性
  - 实现
  - 接口
  - …

# SRS不应包含的内容

- 项目开发计划
  - 诸如成本、人员、进度、工具、方法等
- 产品保证计划
  - 诸如配置管理、验证与测试、质量保证等
- 软件设计细节
  - 需求通常用于表达“做什么”，而不描述“如何做”

# 功能性描述

- **[例1]** 如果可能的话，应当根据图书编号的列表在线确认所输入的图书编号。

- 修改之后：
  - 系统必须根据在线的图书编号列表确认所输入的图书编号。如果在图书编号列表中查不到该图书的编号，或者当进行图书编号确认时图书编号列表不可访问，系统必须显示一个出错信息并且拒绝预订。

# 非功能性描述

- **[例2]** 产品必须在固定的时间间隔内提供状态信息，并且每次时间间隔不得小于**60** 秒。
- 修改之后：
  - 后台任务管理器应该在用户界面的指定区域显示状态信息。
    1. 在后台任务进程启动之后，消息必须每隔**60±10** 秒更新一次，并且保持连续的可见性。
    2. 如果正在正常处理后台任务进程，那么后台任务管理器必须显示后台任务进程已完成的百分比。
    3. 当完成后台任务时，后台任务管理器必须显示一个“已完成”的信息。
    4. 如果后台任务中止执行，那么后台任务管理器必须显示一个出错信息。

# 编写需求规格说明的原则

- 原则1：只描述“做什么”而无须描述“怎么做”
- 原则2：必须说明运行环境
- 原则3：考虑用户、分析员和实现者的交流
  - 对形式化和自然语言之间作出恰当的选择
  - 明确的理解最重要，不存在十全十美的软件规格说明书
- 原则4：力求寻找到恰如其分的需求详细程度
  - 一个有益的原则就是编写单个的可测试需求文档
  - 建议将可测试的需求作为衡量软件产品规模大小的尺度

# 编写需求规格说明的原则

- 原则5：文档段落不宜太长
  - 简短
  - 记住：不要在需求说明中使用“和/或”、“等等”之类的词
- 原则6：避免使用模糊的、主观的术语
  - 如用户友好、容易、简单、迅速、有效、许多、最新技术、优越的、可接受的、最大化、最小化、提高等
  - 不可验证
- 建议：采用一种标准的SRS 模板

# SRS的三大部分：“1. 引言”

1. 引言
   - 1.1 系统目标
   - 1.2 系统范围
   - 1.3 术语表
   - 1.4 参考资料
   - 1.5 总结
2. 现状描述
3. 建议的系统

# SRS的三大部分：“2. 现状描述”

1. 引言
2. 现状描述

*如开发的是支持业务系统软件（如销售业务），详细描述业务现状。如业务概况、与业务相关的组织机构设置、人员职责、与其他相关业务联系等。*

*如开发的是支持产品系统的软件（多为嵌入式软件），描述市场上与之相关的产品功能，并分析其特点*

3. 建议的系统

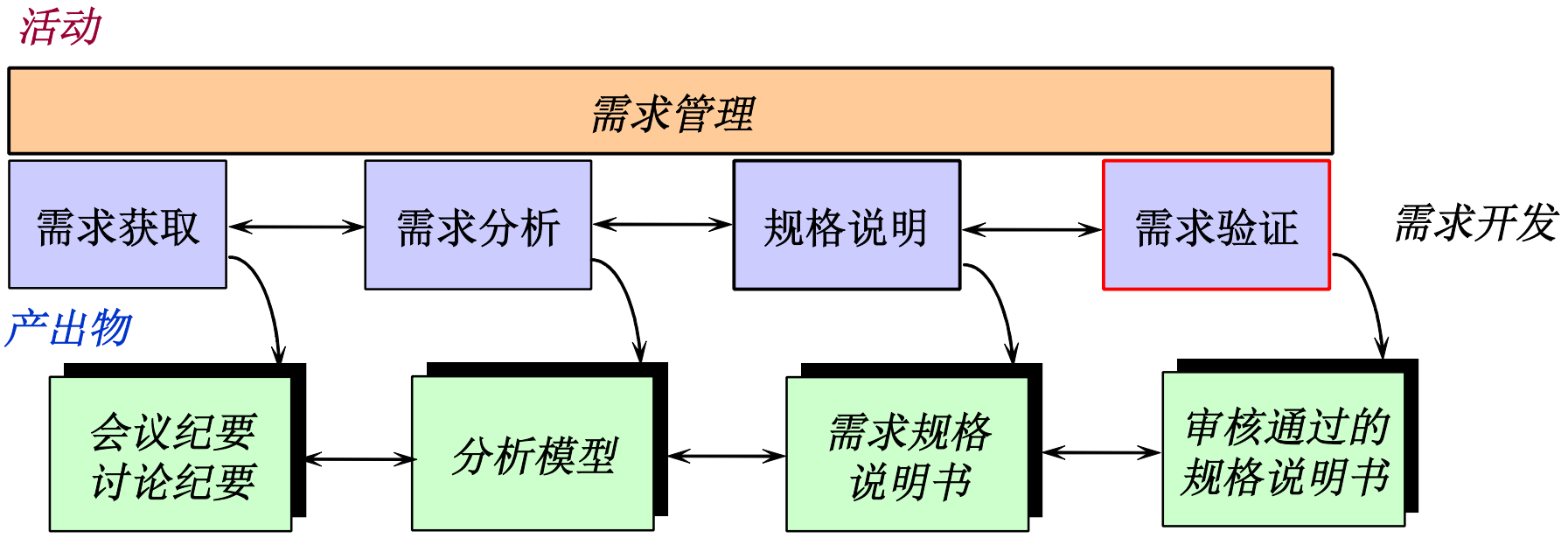
# SRS的三大部分：“3. 建议的系统”

1. 引言
2. 现状描述
3. 建议的系统
   - **3.1** 概述
   - **3.2** 功能性需求
   - **3.3** 非功能性需求
   - **3.4** 系统模型
     - **3.4.1** 用例模型
     - **3.4.2** 对象模型
     - **3.4.3** 动态模型
     - **3.4.4** 用户接口

# 本章内容

- **5.1** 需求获取的挑战
- **5.2** 需求获取的途径
- **5.3** 需求获取技术
- **5.4** 需求获取管理
  - 5.4.1 需求规格说明
  - 5.4.2 需求验证
  - 5.4.3 需求变更

# 需求验证


活动
需求管理
需求获取
需求分析
规格说明
需求验证
需求开发
产出物
会议纪要
讨论纪要
分析模型
需求规格
说明书
审核通过的
规格说明书

# 需求验证

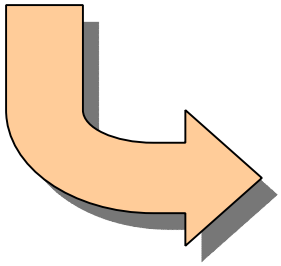
- **需求验证：通过确定以下几方面的内容来检验需求能否满足客户的意愿**
  - 软件需求规格说明正确描述了预期的系统行为和特征
  - 从系统需求或其它来源中得到软件需求
  - 需求是完整的和高质量的
  - 所有对需求的看法是一致的
  - 需求为继续进行产品设计、构造和测试提供了足够的基础

# 需求验证

- **需求验证的技术**
  - 需求评审：由不同代表(如分析员、客户、设计人员、测试人员)组成的评审小组以会议形式对需求进行系统性分析。
  - 原型评价：客户和用户在一个可运行的原型系统上实际检验系统是否符合他们的真正需要。
  - 测试用例生成：通过设计具体的测试方法，发现需求中的问题。
- **需求验证主要围绕SRS的质量特性展开**
  - 正确性
  - 无二义性
  - 完整性
  - 一致性
  - 按重要度排序
  - 可验证性
  - 可修改性
  - 可跟踪性

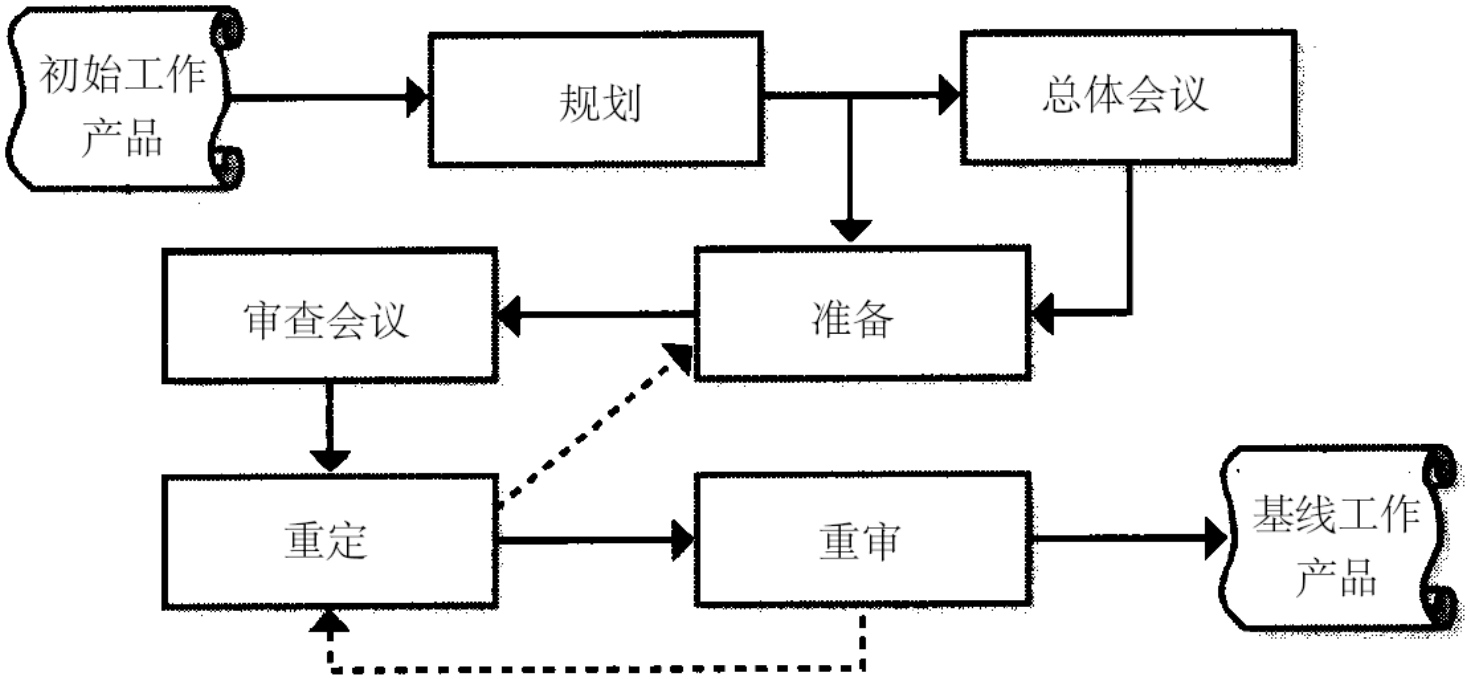
# 需求评审

- “在审查需求文档或其它软件产品上花费一个小时，可节省将来十个小时的工作时间。”
- 需求评审所需的参与角色：
  - 编写需求文档的分析员
  - 未来的开发人员
  - 未来的测试人员
  - 项目经理
  - 客户/用户代表



- 作者：创建或维护正在被审查的产品；
- 调解者：与作者一起为审查制订计划，协调各种活动，并且推进审查会的进行；
- 读者(审查员)：每次审查规格说明中的一块内容，并做出解释，而且允许其它审查员在审查时提出问题。对于一份需求规格说明，审查员每次必须对需求给出注解或一个简短评论；
- 记录员：用标准化的形式记录在审查会中提出的问题和缺陷。

# 需求评审的过程



# 需求评审的退出条件

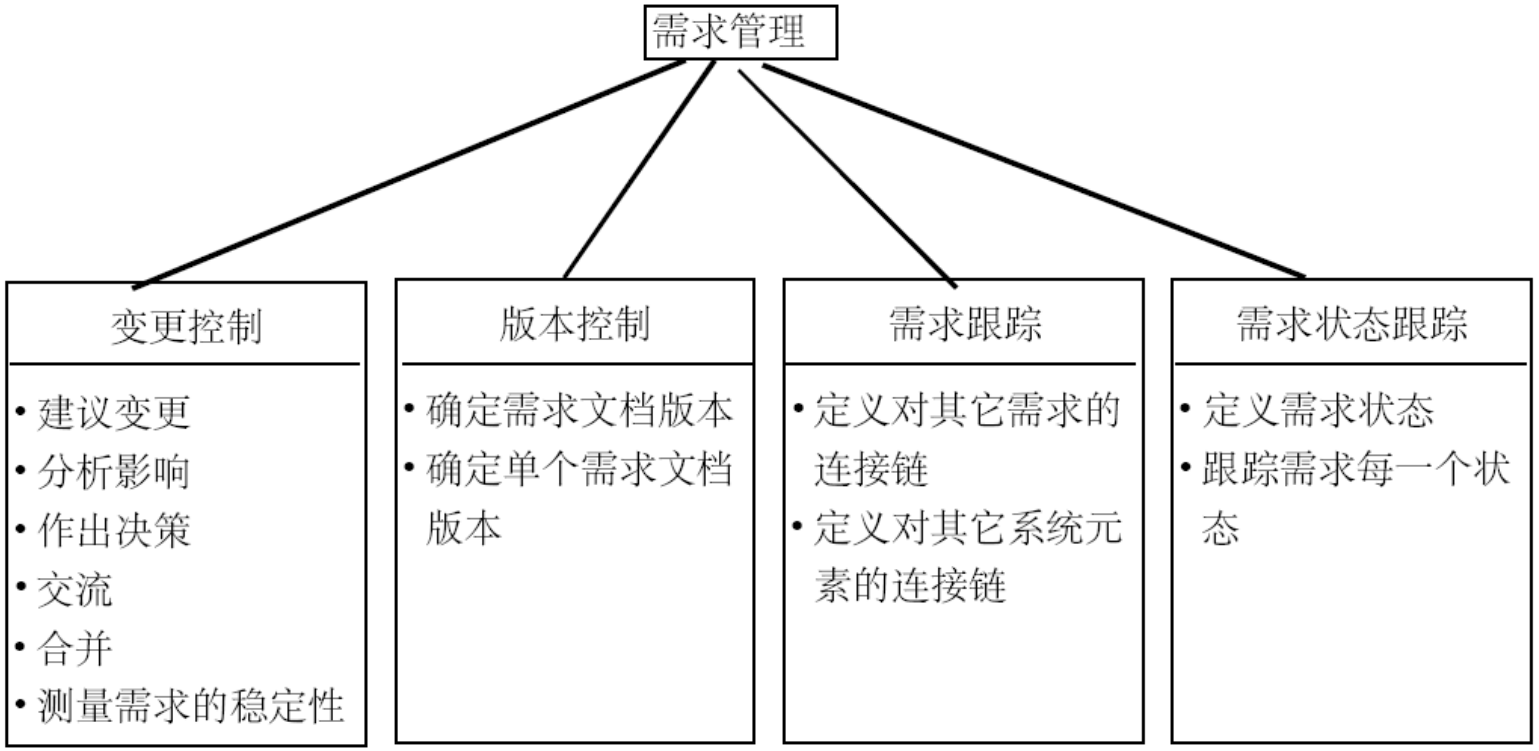
- 已经明确阐述了审查员提出的所有问题
- 已经正确修改了文档
- 修订过的文档已经进行了拼写检查和语法检查
- 所有**TBD**的问题已经全部解决，或者已经记录下每个待确定问题的解决过程、目标日期和提出问题的人
- 文档已经登记入项目的配置管理系统
- 检查是否已将审查过的资料送到有关收集处

# 本章内容

- **5.1** 需求获取的挑战
- **5.2** 需求获取的途径
- **5.3** 需求获取技术
- **5.4** 需求获取管理
  - 5.4.1 需求规格说明
  - 5.4.2 需求验证
  - 5.4.3 需求变更

# 需求变更

- 需求管理是分析变更影响并控制变更的过程，主要包括变更控制、版本控制和需求跟踪等活动。



# 需求变更管理

- “无论何时客户对我们的分析人员提出变更要求，他们总是说同意，我们就只好努力去做出来。”

  ——这是软件设计与开发人员的经常抱怨

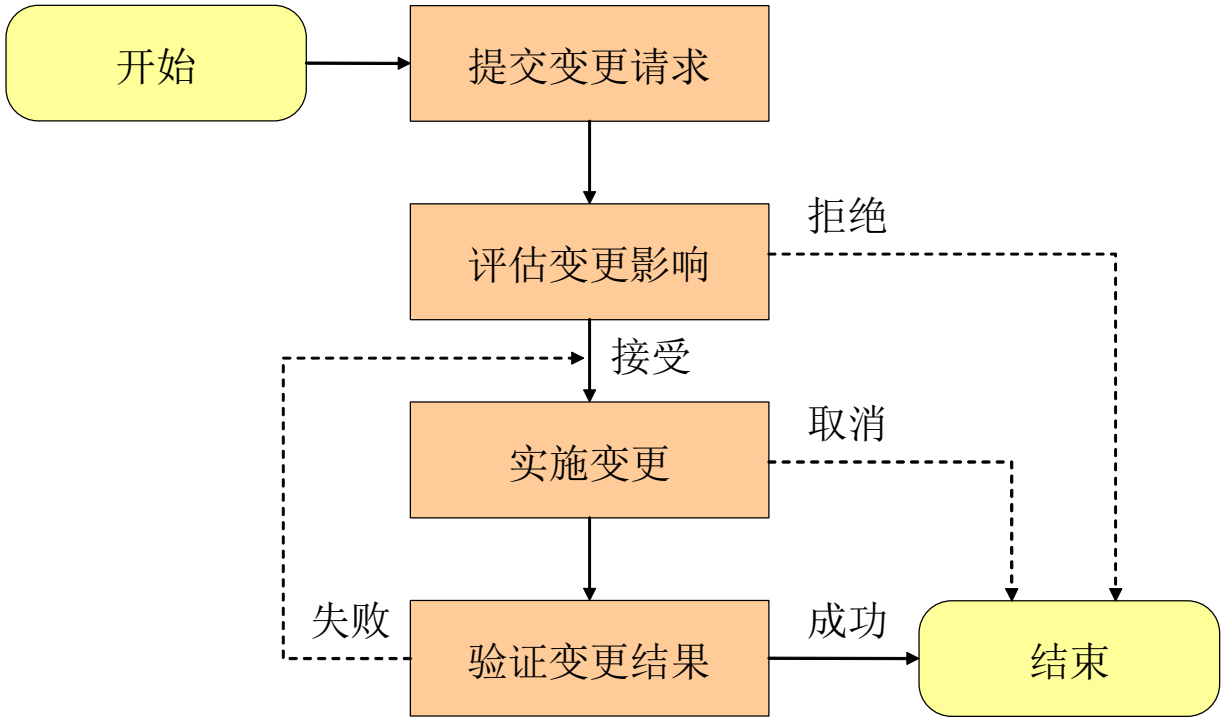
- 不被控制的变更是项目陷入混乱、不能按进度执行或软件质量低劣的共同原因
  - 应仔细评估已建议的变更
  - 挑选合适的人选对变更做出决定
  - 变更应及时通知所有涉及的人员
  - 项目要按一定的程序来采纳需求变更

# 需求变更管理

- **理想情况：**
  - 在开始构造前应该收集到所有新系统的需求，在通过需求验证之后应该冻结，而且在开发中基本上不变更。
- **现实情况：**
  - 很早确定需求却忽视了“有时候客户并不知道需要什么”的现实，开发人员应该对用户这些需求变更作出响应。
- **因此，需要采纳需求变更过程来控制用户需求的频繁变化。**

# 需求变更控制

- 变更控制过程并不是“拖延时间以敷衍用户”，它是一个渠道和过滤器，通过它可以确保采纳最合适的变更，使变更产生的负面影响减少到最小。



# 私自变更造成的后果…

——“我正在应客户的要求添加一个销售分类查询的功能，本以为很快就可以完成，但实际上比我原先预计的工作量超期多了。”

——“本该通过正式渠道进行变更，但这个功能看上去较简单，所以当时我就答应他了。”

——“这个功能其实并不简单，每次当我认为该完工了，但总能意识到在另一个文件中漏了一个变更，所以不得不修改它、再测试一遍。”

——“原以为花4个小时就可以了，实际上花了4天时间，造成我没能按计划完成任务。”

# 原因是什么？

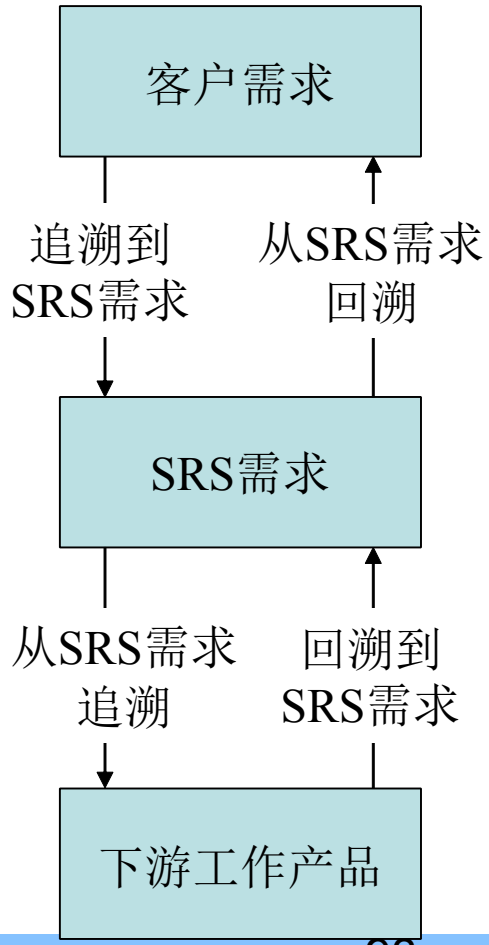
- 各项需求之间存在着关联关系，因此一个看似很小的改变可能会影响很大的范围。
- 在同意接受建议的变更之前，要确信弄清楚此次变更到底会影响到什么。

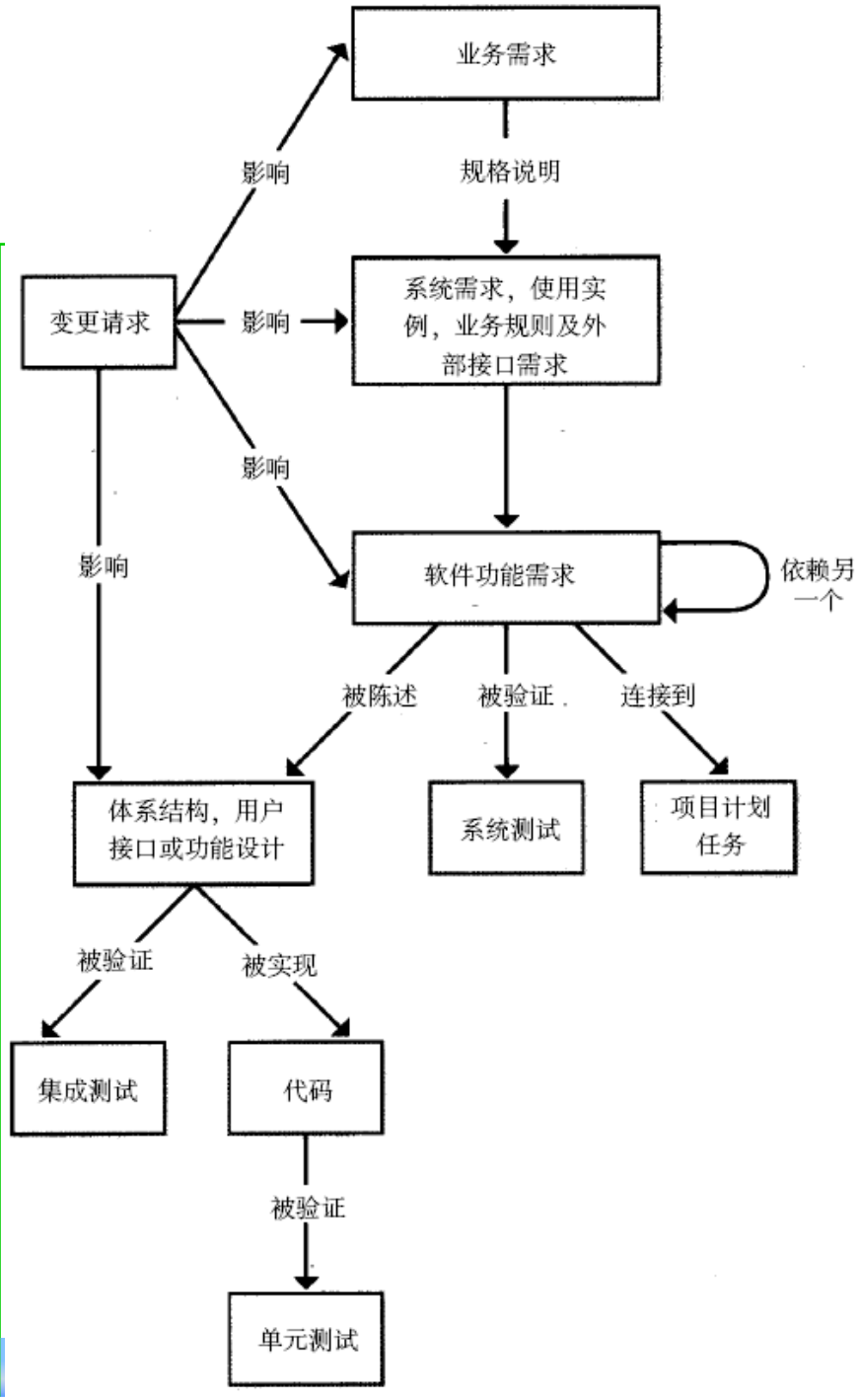
波纹效应

**Ripple effect**

# 追踪性维护

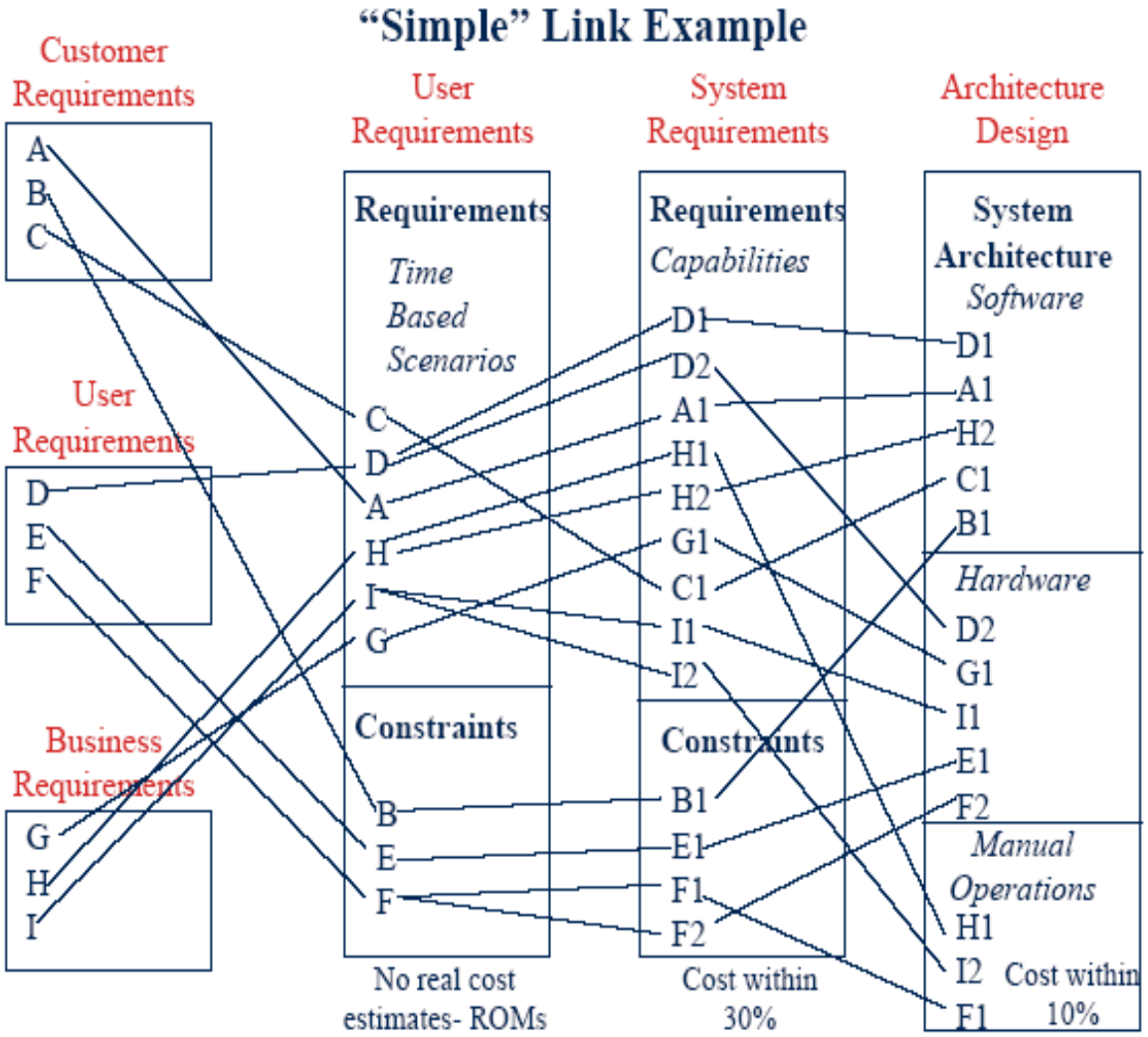
- 需求追溯链(Traceability Link)的四种类型
  - 客户需求→SRS需求
    - 当客户需求发生变更时，可找到受影响的SRS需求，并确认SRS中包含了所有客户需求；
  - SRS需求→客户需求
    - 确认每一项SRS需求的源头；
  - SRS需求→下游产品
    - 确认每项需求对应哪些具体的软件设计与实现，并确认需求的可满足性；
  - 下游产品→SRS需求
    - 确认每个具体软件模块“存在的原因”






# 一些可能的需求跟踪能力联系链

# [例]需求跟踪能力联系链


"Simple" Link Example
Customer Requirements
A
B
C
User Requirements
D
E
F
Business Requirements
G
H
I
User Requirements
Requirements
Time Based Scenarios
C
D
A
H
I
G
Constraints
B
E
F
No real cost estimates- ROMs
System Requirements
Requirements
Capabilities
D1
D2
A1
H1
H2
G1
C1
I1
I2
Constraints
B1
E1
F1
F2
Cost within 30%
Architecture Design
System Architecture
Software
D1
A1
H2
C1
B1
Hardware
D2
G1
I1
E1
F2
Manual Operations
H1
I2
F1
Cost within 10%

# 需求跟踪管理

- 需求跟踪
  - 维护需求与软件制品之间的映射(例如设计对象、用例、测试用例、已实现的软件模块等)
  - 全面分析变更所带来的影响，以便作出正确的变更决策。
- 建立需求跟踪的过程
  - 识别并唯一地标识**SRS**中的每一个需求
  - 建立和更新**SRS**中的跟踪矩阵
  - 工作制品的创建者负责增加该制品与需求的跟踪信息
  - 跟踪矩阵应该作为工作制品的一部分进行审查

# 需求跟踪能力矩阵

- 需求跟踪能力矩阵：表示某项需求和别的系统元素之间的联系链

| 用例 | 功能需求 | 设计元素 | 代码 | 测试实例 |
|---|---|---|---|---|
| UC-28 | catalog.query.sort | Class catalog | catalog.sort() | test 2<br>test 3 |
| UC-29 | catalog.query.import | Class catalog | catalog.import()<br>catalog.validate() | test 10<br>test 11<br>test 12 |

# 变更需求代价：影响分析

- 影响分析**(impact analysis)**：定量分析一次需求变更可能造成什么样的影响

- 方法：
  - 先找到直接受影响的因素；
  - 再根据“波纹效应”，找到间接受影响的因素

变更请求ID_____
标题_____
描述_____
分析者_____
日期_____
优先权评估：
相关收益_____（1-9）
相关代价_____（1-9）
相关成本_____（1-9）
相关风险_____（1-9）
最终优先级_____
预计总耗时_____劳动时数
预计损时_____劳动时数
预计对进度的影响_____天数
额外的成本影响 _____金额
质量影响_____
_____
被影响的其他需求_____
被影响的其他任务_____
要更新的计划 _____
综合的事项 _____
生存期成本事项 _____
可能的变更所需检查的其他部件 _____
_____

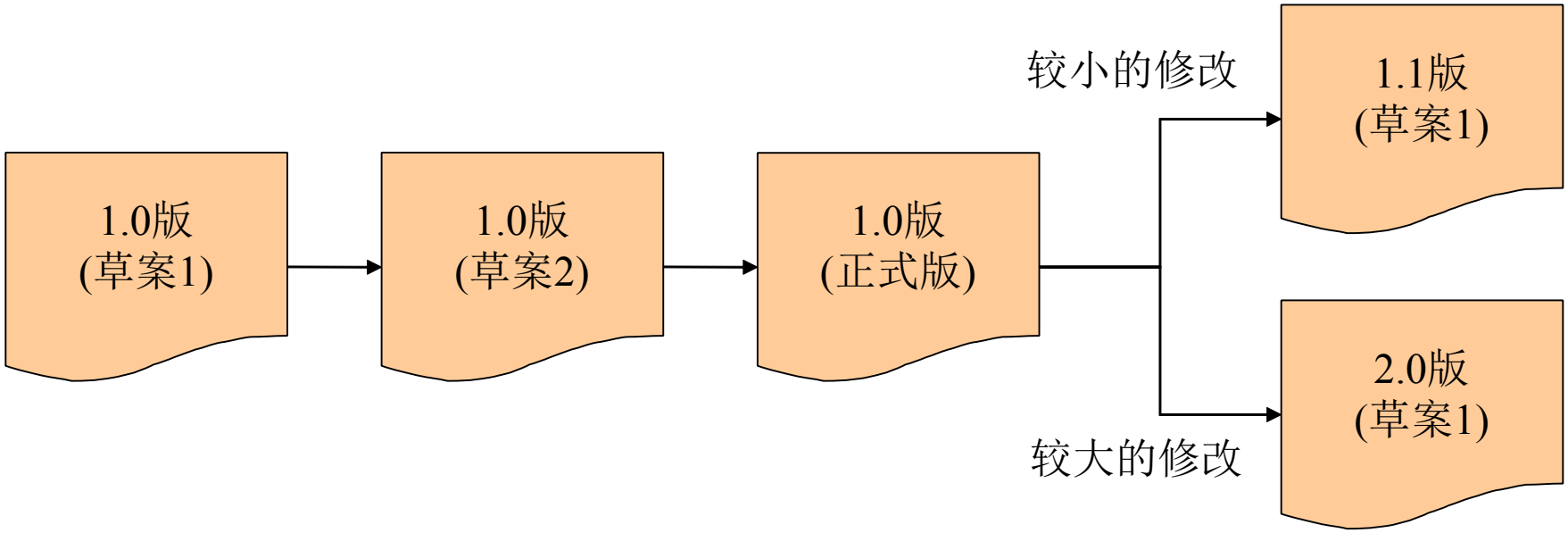
# 需求版本控制

——“我终于实现了库存报告中重排序的功能。”小张在项目的每周例会上说。

——“噢，用户在两周前就取消这个功能了。”项目经理说，“你没看改过的软件需求规格说明吗？”

- 版本控制是管理需求的一个必要方面。
- 需求文档的每一个版本必须被统一确定。
- 组内每个成员必须能够得到需求的当前版本，必须清楚地将变更写成文档，并及时通知到项目开发所涉及的人员。
- 为防止混乱，应仅允许指定的人来更新需求。

# 需求版本控制



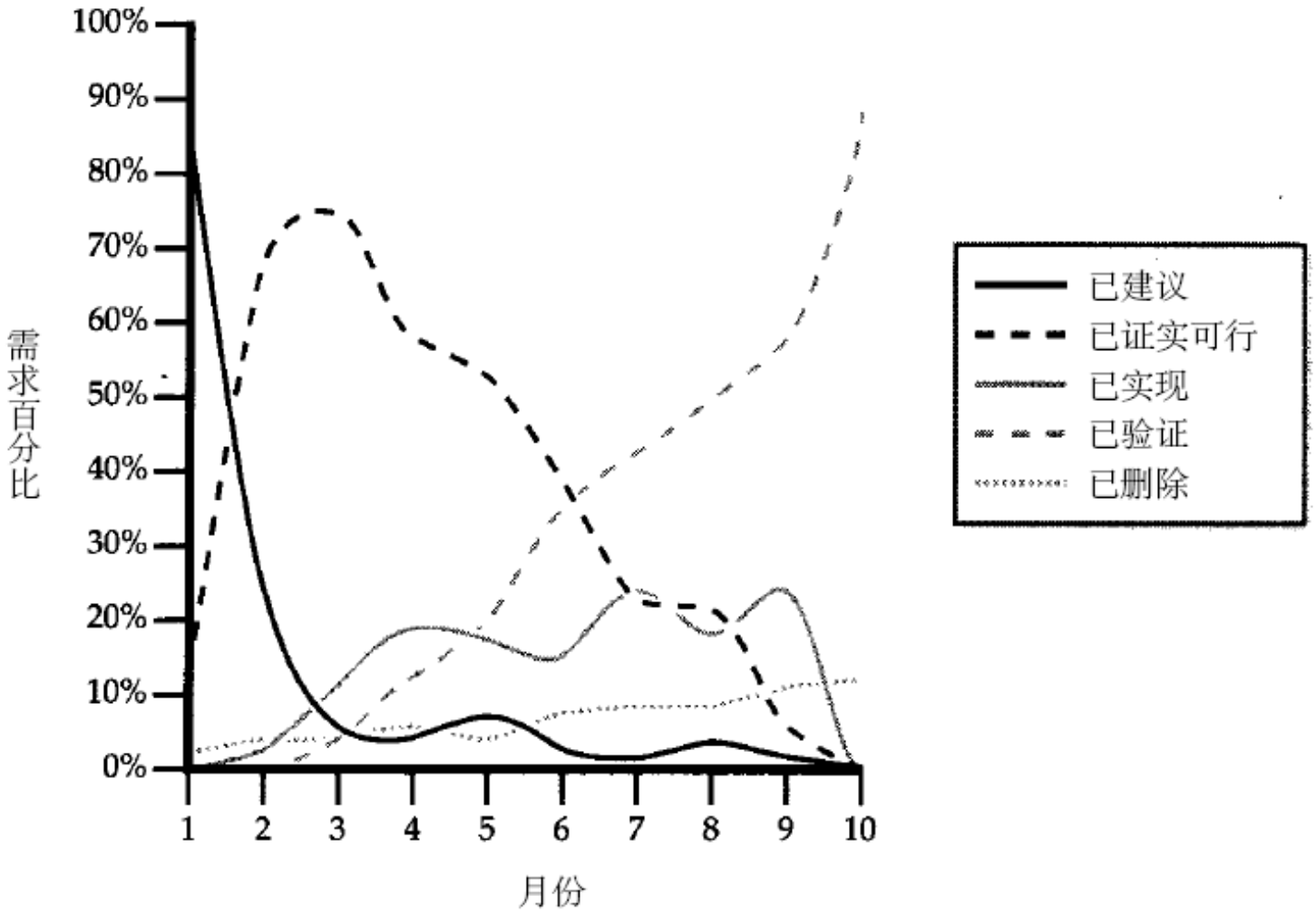
# 每一项需求所需记录的属性

- 创建需求的时间
- 需求的版本号
- 创建需求的作者
- 负责认可该需求的人员
- 需求状态
- 需求的原因或根据(或信息的出处)
- 需求涉及的子系统
- 需求涉及的产品版本号
- 使用的验证方法或接受的测试标准
- 产品的优先级或重要程度
- 需求的稳定性

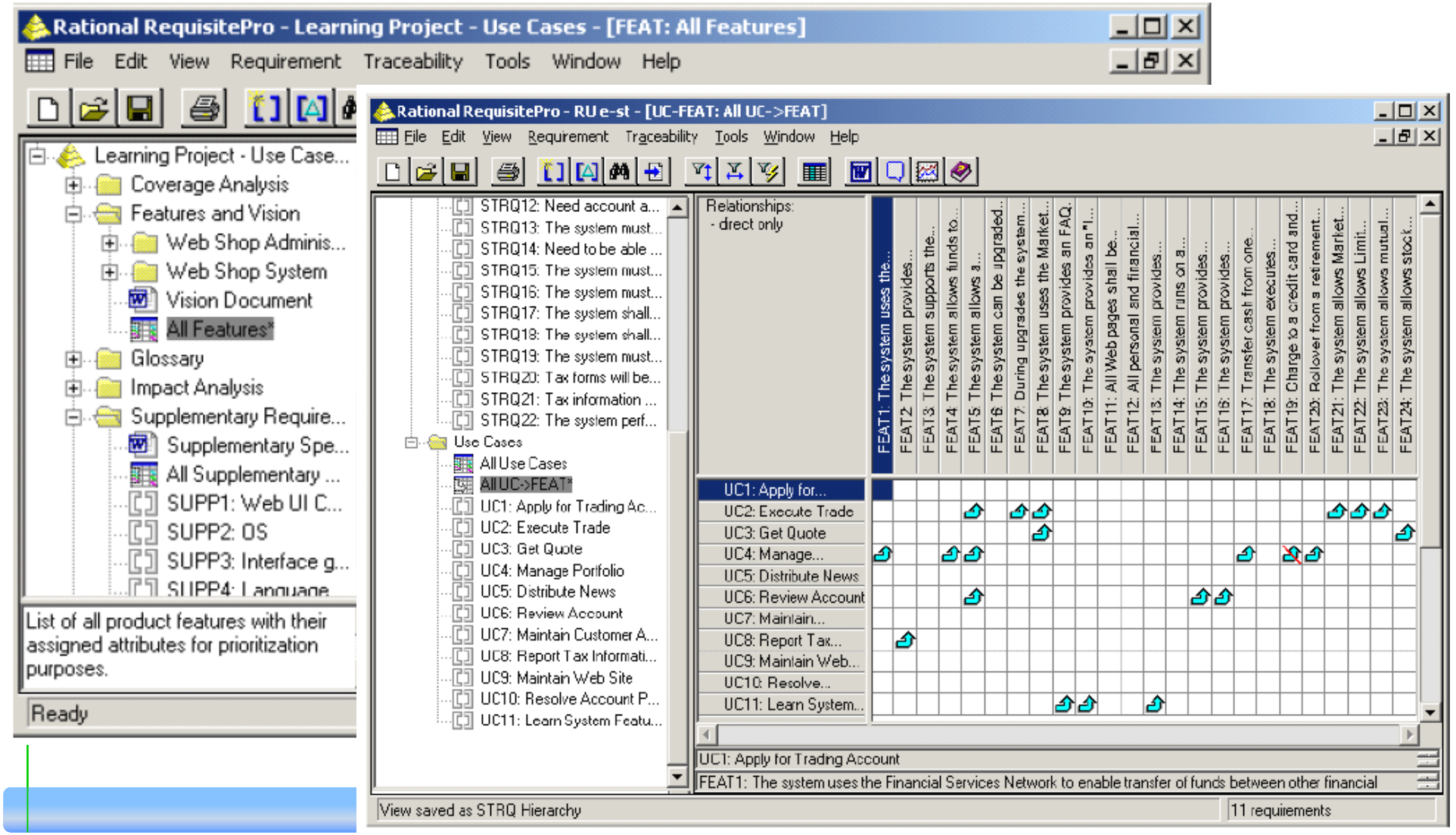
# 需求的状态

| 状态值 | 定义 |
| --- | --- |
| 已建议 | 该该需求已被有权提出需求的人建议 |
| 已批准 | 该需求已被分析，估计了其对项目余下部分的影响（包括成本和对项目其余部分的干扰），已用一个确定的产品版本号或创建编号分配到相关的基线中，软件开发团队已同意实现该项需求 |
| 已实现 | 已实现需求代码的设计、编写和单元测试 |
| 已验证 | 该使用所选择的方法已验证了实现的需求，例如测试和检测，审查该需求跟踪与测试用例相符。该需求现在被认为完成 |
| 已删除 | 该计划的需求已从基线中删除，但包括一个原因说明和做出删除决定的人员 |

# 在整个项目开发周期中跟踪需求状态的分布



# 需求管理工具：RequisitePro



# 结束

**2011年4月20日**