

Chapter 4 Capturing the Requirement

Contents:

- A: eliciting requirements
- B: modeling requirements
- C: reviewing requirements
- D: documenting requirements

需求似乎耳熟能详,但做起来还真不简单,比如” 如何保持需求的完整性和一致性?”“如何满足需求的扩充?

Why requirements are so important ?

----sidebar 4.1 (Standish Group的调查结果)

What is the requirements?

----Understanding what the customers and users expect the system to do.

Chapter 4 Capturing the Requirement

接本章第一页文本框内问题的讨论：

- 1.需求如何保持一致？这是一件容易的事情吗？
- 2.需求的扩充/变更有界限吗？用户怎么看待？

见下面开始的两页内容：

第一页问题：需求中的“加工”部分的进一步描述
将如何保持一致？

第二页问题：需求扩充/变更的用户签字确认问题！

功能描述	实现对现有图档资料的借阅/复制功能，其中借阅是对非电子文档而言的，复制是对电子文档而言的。
支持文件	
输入	借阅/复制申请信息，包括申请人、申请借阅的资料编号（图号或文件编号）、申请时间、申请原因、备注等 注：可参考附表四.数据规格描述表D110004项目资料信息表、D110005申请表
加工 (事件流)	1.提出图档的借阅/复制申请信息。 2.对申请信息进行 审批 ，如果审批通过，则进行相应的图档借阅/复制操作，否则返回，重新申请。 3.对已借阅的图档还有相应的归还功能，记录归还人及归还时间。 4.借阅/复制操作有日志记录。 注：可参考附表一.业务规程描述表B110402图档借阅/复制
输出	借阅/复制操作日志 注：可参考附表四.数据规格描述表D110004项目资料信息表、D110005申请表。
DFD图 或活动图	业务数据流程：

设计师很可能就按照自己的理解做下去了，若是多角色合审，而审核前有需要不同准备的要求呢？

谈谈需求变更的用户签字确认问题

在软件项目执行过程中，需求的变更在所难免。让用户对需求变更签字确认往往是项目组比较头疼的事情之一。一般来说，正常的需求变更还是需要用户签字确认的，当然要做通客户的工作，让客户理解：签字是为了更好的保障开发人员理解真正需求。

一般来说，用户不签字有两种原因：

- 一. 是用户认为不是变更，是应该做的工作。
- 二. 是担心承担责任，不愿意签字。

如何解决？

（是不是变更必须要进行充分讨论。对甲乙双方而言，该承担的责任一定要承担。特别是有时需求变更意味着工作量变化）

（提示：需求变更问题要在合同中写明流程与责任等。）

Chapter 4 Capturing the Requirement

需求的重要性:

A: requirements process is critical to good software development （需求的困难在于要准确的决定构建什么系统；包括人机界面及其蕴含的功能、对机器接口、对其他软件系统接口、性能的匡算等等。错误的需求在项目后期难以调整。）

B: the top 8 factors that cause the failure of projects (siderbar4.1)

----不完整的需求；缺少用户参与；缺乏资源；不切实际的期望；缺乏行政及政策支持；需求和规格说明的变更；缺乏计划；因市场或政策原因不再需要该系统。

----几乎所有这前8个因素都直接或间接涉及需求。

Chapter 4 Capturing the Requirement

需求的重要性----看修复需求错误的代价比:

----假设: 在需求定义过程中找出并修复一个基于需求的问题只需花费: \$ 1.00

----那么: 在设计过程中修复这个问题花费: \$ 5.00

----那么: 在编码过程中修复这个问题花费: \$ 10.00

----那么: 在单元测试过程中修复这个问题花费: \$ 20.00

----。 。 。 。 。

----那么: 在系统交付后进入维护阶段后, 修复这个问题花费: \$ 200.00

---- “死亡行军 (Death March) 从需求开始”: 用来描述其进度表几乎不可能完成的项目。“两难境地”的加班人经常半夜回家。

Chapter 4 Capturing the Requirement

4.1 The Requirements Process(需求的过程)

1. Introduction:

① new system

A: the new system may replace an existing system or way of doing things (replacement)

B: the new system is an enhancement or extension of a current manual or automatic system (enhancement/extension)

note: frequently the new system is planned for doing things that have never been done before

(new functions) 无论上述哪种情况,都会有新功能出现,这是由于软件需求的特性使然

Chapter 4 Capturing the Requirement

② requirement(P143)

A: definition: it is an comprehensive expression of desired behavior which dealing with objects, states, functions. (是对来自用户的关于软件系统的期望行为的综合描述, 涉及系统的对象、状态、约束, 功能等)

B: task: understand customer's problems and need

----构建工资单生成系统: 每两星期发放一次工资, 具有一定薪水级别的雇员的工资发放方法(存入账户或发放现金), 客户对工资单的访问要求(是门户级别还是端口级别的认证)等等。(原始需求)

----寻找需求的内容: 雇员(标识关键实体), 雇员的每星期的薪水计算不得超过40小时(限定或约束), 雇员Y是雇员X的领导时则有权改变雇员X的薪水(标识实体间的关系) (较正式需求)

2.3节静态建模: 工件, 活动, 规则/状态等等

Chapter 4 Capturing the Requirement

C: focus on: the customer and the problem , not on the solution or the implementation

- 不予考虑的事项：系统如何实现的过程。
- 不提及采用什么数据库系统；
- 不提及软件服务的体系结构；
- 不提及计算机内存分配策略；
- 不提及语言级别的实现等等。

③ determining requirement (确定需求的过程) (Fig4.1)

A: elicitation (原始需求获取)

(问题定义) (what the system should do) (来自客户)

B: analysis(问题分析) (better understand the required behavior, capture requirements in a model or sometime in a prototypes, usually raises additional questions , etc.)

(讨论需求时会引发一系列问题：薪水支付期间离岗，机动车服务系统中的库存控制的具体实施可能引发软件职能重组等等。)

Chapter 4 Capturing the Requirement

C: Specification(规格说明草稿)

(rewrite the definition formally by notations)

D: Validation (需求核准) (check by developers and customer, sometime test the prototypes)

E: final output: formal document of reruirements
(正式的 <SRS> (after validation))

④ importance (P142, siderbar4.1)

---- requirements process is critical to good software development

A: the top 8 factors that cause the failure of projects

B: Incomplete requirements(no.1) and changing of requirements(no.6) account for 13.1% + 8.7%

REQUIREMENTS ELICITATION AND ANALYSIS

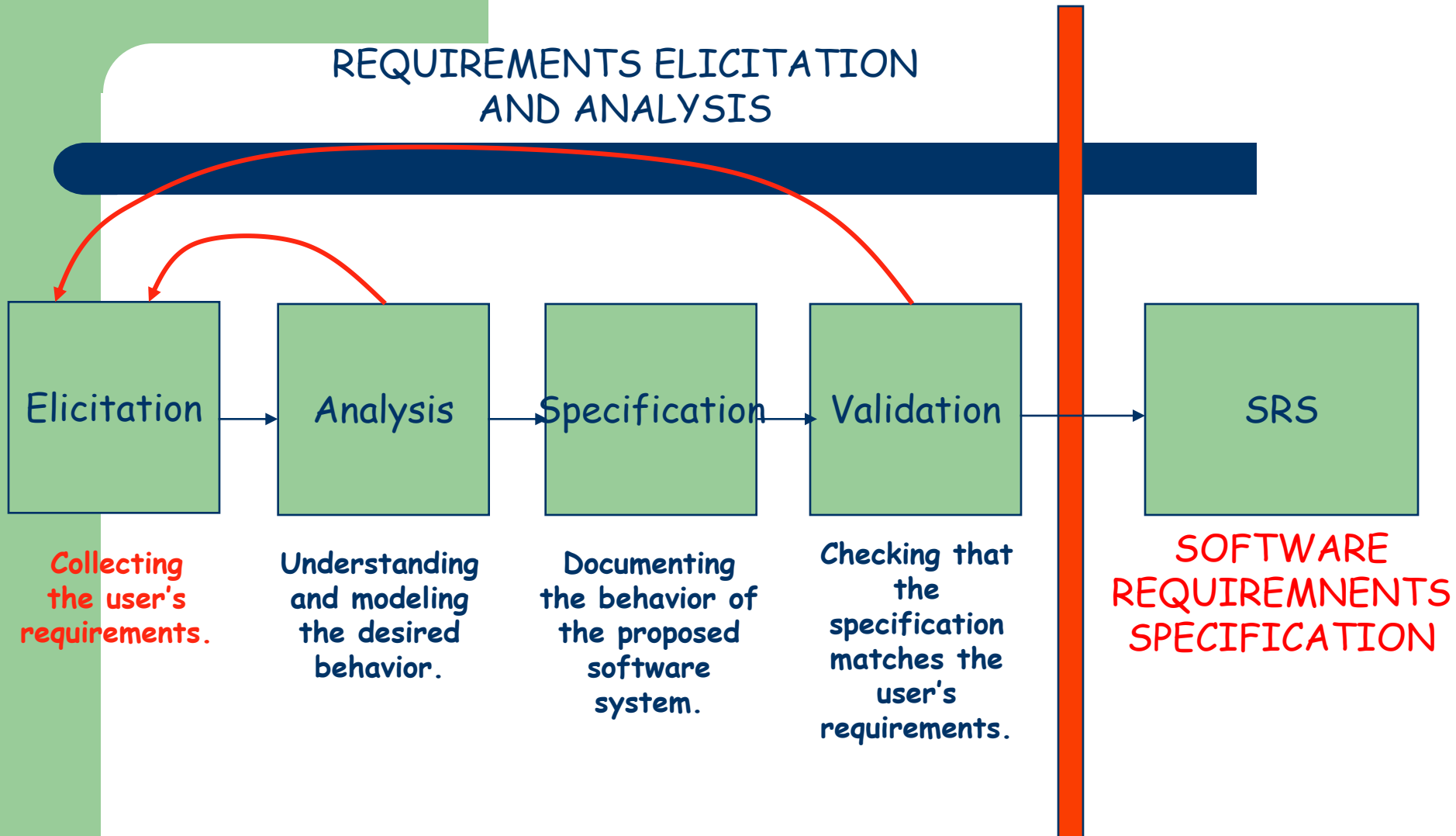


Fig 4.1 The process for capturing the requirements.

Chapter 4 Capturing the Requirement

⑤ Agile Requirements Modeling (siderbar4.2)

（敏捷开发方法的需求建模）

A: agile method appropriate for small or medium team work, and unstable requirement

B: requirement---implements source codes directly

C: agile method gather and implement the requirement in increments

D: 当前的较为主流的团队开发模式: 增量式或迭代式开发

----每次实现小部分细化了的需求.

----多次迭代才能实现全部的需求.

----讲究大型软件设计的艺术和方法.

----局部采用敏捷开发方法.（敏捷方法也在吸纳迭代思想）

Chapter 4 Capturing the Requirement

Feature of agile method and formal approach :

A: focus: heavy process is most appropriate for large-team development and stable requirement .

B: feature: agile---- after essential requirements, then coding directly. ensure implementation correctly by test cases.

C: problem: it is difficult to ensure the quality of the main steps and other elements, for example 1, new project design's solution.

(可能设计不够成熟)

example 2, the test cases's quality.

(测试用例的质量不高)

关于需求的说明：

- 需求不涉及设计问题和系统实现-----软件解决方案与实现方案不被提及。
- 需求最好不要直接指导设计-----为了给予设计人员最大的灵活性。
- 需求过程中的问题分析是一个广泛性问题。
 - 稳定性需求与不稳定性需求，软件规模的巨大差异，都给需求制作带来了巨大挑战。
 - （因为涉及许多参与者，角度不同，意见不一，需求很难满足各方观点，见教材下节：）

用户画像分析软件_精准洞察用户需求

- 捕捉正确的“场景”，找到对的“人”。精确进行需求切片。
- 对用户线上和线下行为深度洞察，构建全面、精准、多维的用户画像体系，为APP提供丰富的用户画像数据以及实时的场景识别能力，帮助APP全方位了解用户。

行为标签
近期活跃的应用
近期去过的场景

场景标签：机场，
商圈，电影院，
景区，自定义场
景等



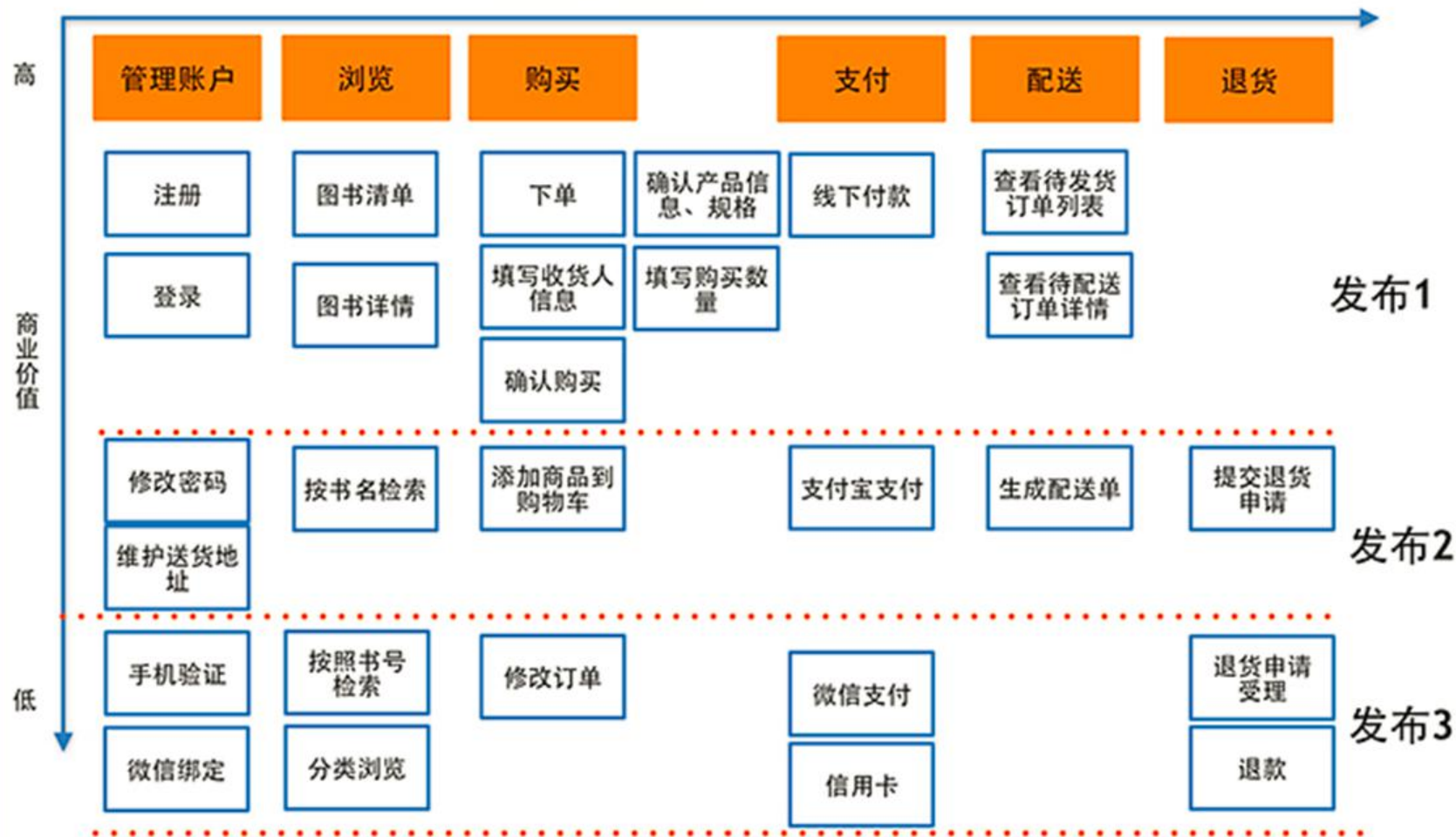
属性标签：性别，
年龄层次，消费
水平，职业等

兴趣标签：购物，
教育，影音，游
戏，金融理财等

定制化标签：
定制化标签 A，
B，C 等。

故事地图示例 - 在线购书网站

业务流程 (时间线)



Chapter 4 Capturing the Requirement

4.2 Requirement elicitation (需求的引出) (P144)

----requirements elicitation is an especially critical part of the process. We must use a variety of techniques to determine what the users and customers really want (problem breakdown, identify people, coming to an agreement about every problem, etc.) (至少要弄清业务含义吧)

---- requirements elicitation is not easy.

(Customer and developer has different viewpoint and background, all stakeholders should come to an agreement about the requirements are)

(siderbar4.3:早期的需求有时会容忍观点的不一致性)

Chapter 4 Capturing the Requirement

- ① **Stakeholders: clients, customers, users, domain experts, market experts, lawyers, software engineers.**
- ② **explain about cooperation between stakeholders**
 - A: each stakeholder has a particular view of the system
 - B: different participants may expect differing levels of detail in requirements
 - C: user and developers may have preconception with each other (**table 4.1**)
- ③ **conclusion : requirements analyst should have the ability to understand each view and capture the requirements in a way that reflects the concerns of each participant.**

How developers see users

How users see developers

Users don't know what they want.

Users can't articulate what they want.

Users have too many needs that are politically motivated.

Users want everything right now.

Users can't prioritize needs.

Users refuse to take responsibility for the system.

Users are unable to provide a usable statement of needs

Users are not committed to system development projects.

Users are unwilling to compromise.

Users can't remain on schedule.

Developers don't understand operational needs

Developers place too much emphasis on technicalities.

Developers try to tell us how to do our jobs.

Developers can't translate clearly-stated needs into a successful system

Developers say no all the time.

Developers are always over budget.

Developers are always late.

Developers ask users for time and effort, even to the detriment of the users' important primary duties.

Developers set unrealistic standards for requirements definition.

Developers are unable to respond quickly to legitimately changing needs

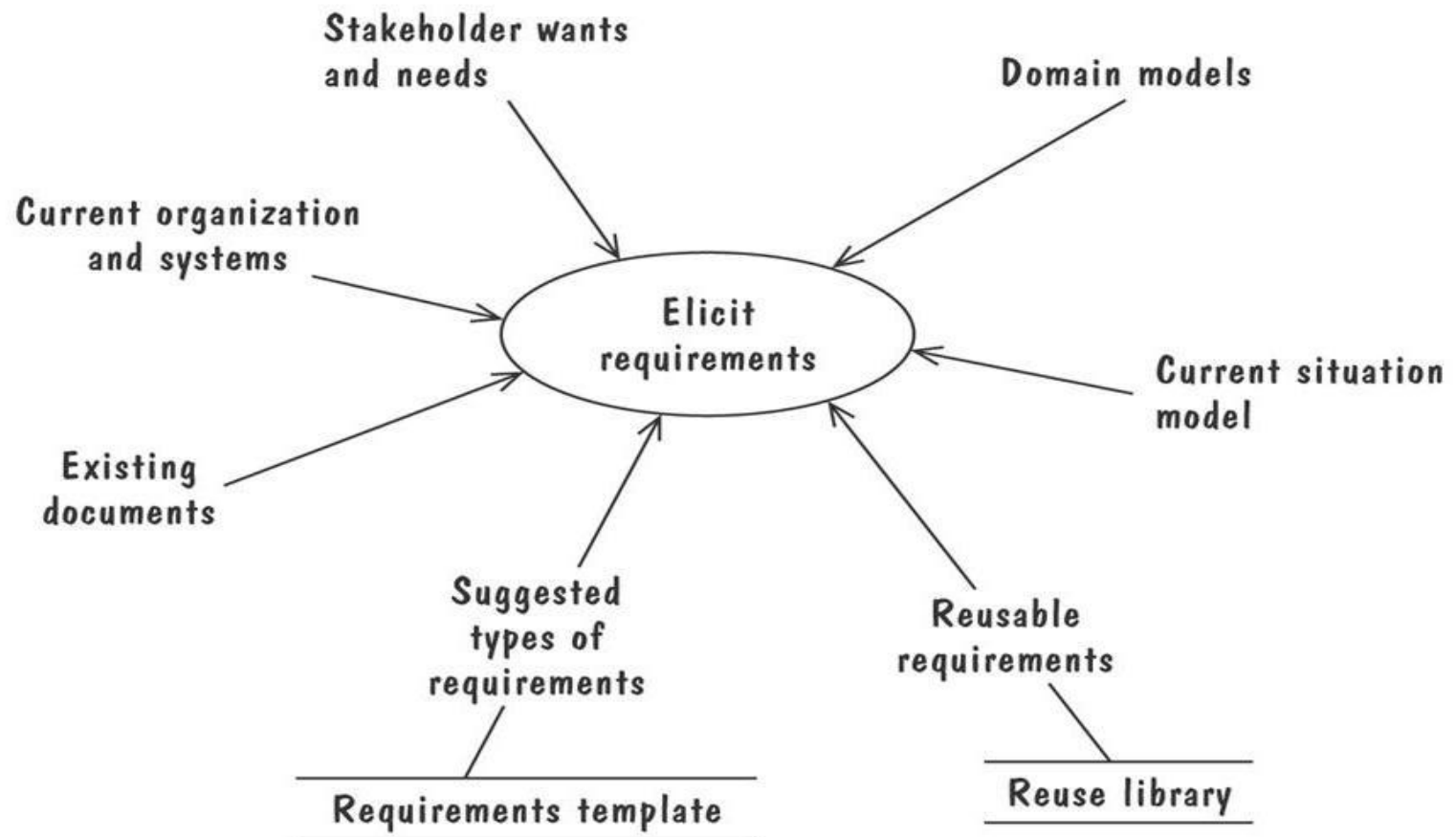
Chapter 4 Capturing the Requirement

④ other means of eliciting requirements(P147-148)

- A: 评审可用文档。手工记录，原来用户手册等。
- B: 观察当前系统（如果存在）。或观察未来的应用场景。
- C: 做用户的学徒。详细参观学习等。
- D: 小组专访。团队与用户或利益共担方交流。
- E: 聚焦特定需求。系统扩展与特殊处理等。

⑤ Volere需求过程模型（图4-2）-----额外的需求资源

- The Volere requirements process model suggests some additional sources for requirements



Chapter 4 Capturing the Requirement

4.3 Types of Requirements

1. Functional vs. non-functional requirements

① Functional requirement (功能需求) :

---describes internal functions and an interaction between the system and its environment

(描述系统内部功能或系统与外部环境的交互作用,涉及系统输入应对,实体状态变化,输出结果,设计约束与过程约束等.)

- **Examples: (practical functions)**
 - **System shall communicate with external system X.** (例如: 一般通讯功能----链接: 完成一次信息交换)
 - **Printing weekly paycheck :** (一般功能)

Chapter 4 Capturing the Requirement

- When paychecks are issued? (某种条件下的功能)
- What input is necessary for a paycheck to be printed?

(为完成某种功能而提出的附加功能—完成一次输入)

② **design constraints(设计约束)** (see table 4.2)

物理环境：对环境或设备的限制等（安装及环境要求等）

接口：涉及输入输出的限制或约束条件. (输入格式预定等)

用户：使用者的基本情况（限定几种类型的用户）

③ **process constraints(过程约束)** (see table 4.2)

资源：材料、人员技能或其它。

文档：类型、数量或其它。（涉及其针对性及要求等）

标准：比如阅读文档时的用户指派标准。

其他：什么原因会导致从工资单列表中删除某雇员？

Chapter 4 Capturing the Requirement

④ Quality /nonfunctional requirement (质量需求)

---- describes some quality characteristic that the software solution must possess

(描述软件方案必须具备的某些质量特征)

- **Examples:**

- Paychecks distributed no more than 4 hours after initial data are read.(系统某些性能约束性)
- System limits access to senior managers(安全性)
- System will respond to query in not more than 3 seconds (反应时间)

Note: two requirements are elicited in formal way

Table 4.2---questions to get requirements (设计师考试有内容)

Chapter 4 Capturing the Requirement

2. Resolving Conflicts

We are bound to encounter conflicting ideas of what the requirements ought to be , so we and customer should prioritize requirements.

three categories of requirement prioritizations:

(需求优先级划分方案一般大致分为3类:)

A: those that absolutely must be met

B: those that are highly desirable but not necessary

C: those that are possible but could be eliminated

----- (之所以有上述分类的原因: 有的软件开发项目受到了技术、开发时间、合作路径和资源等诸多因素的限制, 有的需要二次资金投入等)

Chapter 4 Capturing the Requirement

making requirement testable (P151, sidebar4.4)

(使需求变得可测试)

----example: “*water quality information must be accessible immediately.*”

“*water quality records must be retrieved within 5 seconds of a request.*”

----three methods :

A: 针对需求确定一种量化的描述方法，避免模糊的表达方式（尽量以形式化、数字或数学模型刻画需求）

B: 将各种指代用词替换为实体的正式名称（需求不是写小说--重名、化名不可以，“这些、那些”不可以）

C: 每个名词或事项应在需求文档中给出唯一定义。

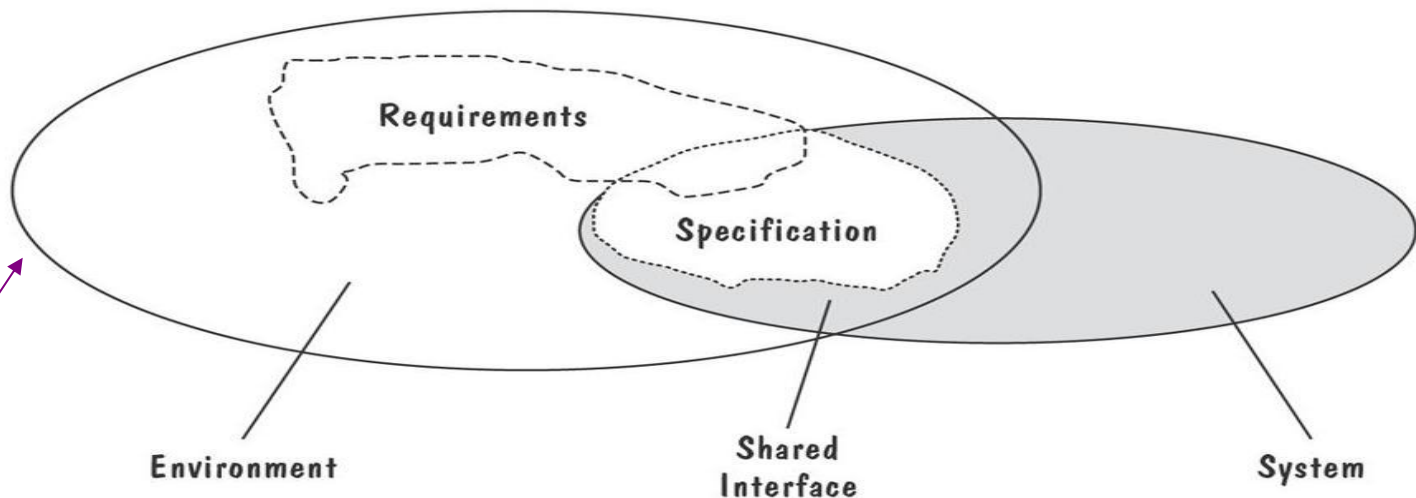
（很多名词、事项等要讨论才能确定）

Chapter 4 Capturing the Requirement

3. Two Kinds of Requirements Documents

- ① Requirements definition: complete listing(完整的罗列) of what the customer expects the system to do
- ② Requirements specification (SRS): restates(重述) the definition in technical terms or notations so that the designer can start on the design
- ③ Configuration management: supports direct correspondence between the two documents
(a set of procedures that track several stages)
(配置管理：使软件过程各阶段文档保持一致的系列过程)
(it provides the threads that tie the system parts together, unifying components that have been developed separately)

- Requirements defined anywhere within the environment's domain, including the system's interface
- Specification restricted only to the intersection between environment and system domain



需求不能解决问题, 特别是环境方面

- 旋转门: 人是属于环境的, 系统域只能检测是否有人投硬币或刷卡, 以及门的开启和旋转角度及时间等, 一般不管人是否走入等等。否则, 要是全部场景考虑在内, 则系统代价极大。

Chapter 4 Capturing the Requirement

软件配置管理的进一步说明

- W. Babich 的解释性定义：

软件配置管理是一种标识、组织和控制修改的技术，目的是最有效的提高生产率。软件配置管理能协调软件开发，使混乱减少到最小。

- 软件配置管理的任务(大型项目开发时, 下面每一项都可能展开:)
 - 制定软件配置管理计划
 - 确定配置标识规则
 - 实施变更控制
 - 报告配置状态
 - 进行配置审核
 - 进行版本管理和发行管理
- 配置管理工具: 早期CVS, 。开源系统 + 二次开发。

Chapter 4 Capturing the Requirement

- 软件配置管理与软件开发过程的区别
 - 两类不同的变更：
 - 开发阶段内部发生的变更：
 - 开发过程解决不了的变更：
 - 变更的评估和批准以及变更实施都要由软件配置管理人员去做。
 - 开发过程应纳入配置管理过程的控制之下。
- 忽视软件配置管理可能导致的混乱现象
 - 发错了版本
 - 安装后不工作
 - 异地不能正常工作
 - 已经解决的缺陷过后又出现错误
 - 开发人员把产品拿出去出售赢利
 - 找不到最新修改了的源程序
 - 找不到编程序的人

Chapter 4 Capturing the Requirement

4.4 Characteristics of Requirements(需求的特征)

① Types of Requirements Document（需求文档的类型）

- Physical environment（谈对物理环境之要求，不谈拓扑结构。）
- Interfaces（只谈人机交互的目标性问题）
- Users and human factors
- Functionality
- Documentation
- Data
- Resources
- Security
- Quality
- Quality assurance

设计师不认为分析师的
拓扑结构就是合适的

Chapter 4 Capturing the Requirement

①Note:

A: requirement: flow of information+constraints.

(Includes: source +transformation +destination +constrains)

B: three purposes of requirements:

X: understand (the customers's need)

Y: function+characteristic (will be realized by developer)

Z: verifying the rightness(of the requirements)

② characteristics(of high quality requirement):

A:correctness B:consistency C: accuracy

D: completeness H: (see P155-156)

③ example:155

前四节总结：需求不容易，制定大型需求要做耐心细致工作。

Chapter 4 Capturing the Requirement

4.5 Modeling Notation (建模符号描述系统 / 建模原语)

1. Reasons for requirements modeling:

A: two traits (特征) of engineering disciplines :

需求中的分解是
为了简化问题，
而设计中的分解
是为提高系统的
模块化、可维护
性、性能等质量
属性

repeatable processes .

standard notations for modeling .

importance of requirement modeling:

translating the requirements in a completely
different form from the customer's original requests,
allowing the customer to examine our models
in order to validate the model's accuracy .

Some notations of requirements modeling have different
goals from that of notations of design modeling when
decompose the two specifications

Chapter 4 Capturing the Requirement

2.面向对象建模

- 哲学观点----世界是由物质构成的，物质之间有联系。
- 程序思想----模拟客观世界，才能解决客观世界的问题。
----软件应该由若干程序实体(对象)组成，实体之间有必然联系与交互。

面向对象思想比较自然地模拟了人类认识客观世界的方式和法则，面向对象的分析和设计应该从建模开始。构造模型通常出于以下几个目的：

- 在着手解决一个复杂问题之前，对解决方案进行检测；
- OO建模接近自然,方便同客户或其他相关人员进行交流；
- 加强视觉效果；（可视化需求是SE多年奋斗目标。）
- 对复杂问题进行量化和简化。

Chapter 4 Capturing the Requirement

面向对象建模

软件工程：
从解决
一个特定
的案例到
解决类问
题的方法
论

- 模型是对事物的一种抽象，人们常常在正式建造实物之前，首先建立一个简化的模型，以便更透彻地了解它的本质，抓住问题的要害；
在模型中，人们总是剔除那些与问题无关的、非本质的东西，从而使模型与真实的实体相比更加简单、易于把握；
在建造一个复杂系统时，开发者必须从多种不同的角度来抽象系统，使用准确的符号来构造模型，然后检查这些模型是否符合系统的需求，并逐步添加细节，从而将这些模型转化成实现方案。
- 建模语言 / 原语是面向对象建模中的一个非常关键的因素。

Chapter 4 Capturing the Requirement

UML的设计目标:

- 运用面向对象概念来构造系统模型
- 建立起从概念模型直至可执行体之间明显的对应关系
- 着眼于那些有重大影响的问题
- 创建一种对人和机器都适用的建模语言

UML概要

- **UML**由**OMG**(对象管理组----一个国际性的、开放会员的、非盈利性的技术标准联盟)于**1997年11月**批准为标准建模语言。
- **UML**建立在当今国际上最有代表性的三种面向对象方法(**Booch**方法, **OMT**方法, **OOSE**方法)的基础之上。
- **UML**是一种建模语言而不是一种方法, **UML**本身是独立于过程的。

Chapter 4 Capturing the Requirement

UML系列模型图

UML为人们提供了从不同的角度去观察和展示系统的各种特征的一种标准表达方式。在UML中，从任何一个角度对系统所作的抽象都可能需要用几种模型图来描述，而这些来自不同角度的模型图最终组成了系统的完整模型。

一般而言，我们可以从以下几种常用的视角来描述一个系统：

- 系统的使用实例：从系统外部的操作者的角度描述系统的功能。
- 系统的逻辑结构：描述系统内部的静态结构和动态行为，即从内部描述如何设计实现系统功能。
- 系统的构成：描述系统由哪些程序构件所组成。
- 系统的并发性：描述系统的并发性，强调并发系统中存在的各种通信和同步问题。
- 系统的配置：描述系统的软件和各种硬件设备之间的配置关系。

Chapter 4 Capturing the Requirement

UML模型图（5类，10种）：

- 用例图
- 静态图（类图，对象图，包图）
- 行为图（状态图，活动图）
- 交互图（顺序图，合作图）
- 实现图（构件图，配置图）

问题：小型团队的一般软件开发最少
可能使用哪几种图？

Chapter 4 Capturing the Requirement

UML主要文件:

- UML概要 (UML Summary)
- UML语义 (UML Semantics)
- UML表示法指南 (UML Notation Guide)
- 对象约束语言规约 (Object Constraint language Specification): 该文件定义并介绍了一种对象约束语言 (OCL), 其用途是用来说明在图形化的系统模型中不能充分表达的建模信息。它是一种形式化语言。

<http://www.rational.com/uml/index.jtmpl>

备注: **UML**作为一种通用语言, 只支持任何**OO**方法, 也并不主张特定的体系结构框架 (只包含少量的体系结构构造)。

Chapter 4 Capturing the Requirement

(1). UML use-case diagrams （用例图）

从本质上讲，一个用例是用户（或角色）与计算机之间为达到某个目的的一次典型交互作用 (功能实现)(对话模型)

- 用例描述了用户提出的一些可见的需求；
- 用例可大可小；
- 用例对应一个具体的用户目标
- 用例也必须描述用户没有直接提出的一些需求；
 - 举例：正在制作一个关于快递业务的网站。用户要求能显示快递编号就行了。比如显示ems快递编号为：**1000000000001**。但是我想在后面加一个连接：“点击查询快递动态”，有没有必要做这样小的更改？
 - 或者：不单独做个按钮，而是把快递单号做成可点击的焦点。
 - 与上面改动设想相对照，这里就出现了一个“用户体验”的问题。

Chapter 4 Capturing the Requirement

- 合同不会覆盖这些内容，此时需求不能受合同所限制。
- 客户之所以找我们来做设计，就是希望借助专业的设计力量来弥补他们自身的短板，所以在项目推进的过程里，要注重沟通。基于我们的专业性，提出一些对方没想到的好点子，考虑得深一些，任何一个想把事情做好的客户都是欢迎这样的需求或设计的。开发方和用户的良好信任关系也是这样通过技术能力、合作诚意慢慢建立起来的。
- 分析师要引导项目需求朝正确的方向前进，而不是像小资般“YY”。
- 软件工程与其说是一门课程，不如说是一门技术思想汇集，其难点在于从需求开始，在每一个环节都要体现正确的分析和处理问题的过程思想，其范畴已经远远不只局限于该门课程。





用例图描述系统外部的执行者与系统的用例之间的某种联系。

- 所谓用例是指对系统提供的功能（或称系统的用途）的一种描述；
- 执行者是那些可能使用这些用例的人或外部系统；
- 用例和执行者之间的联系描述了“谁使用哪个用例”。

Chapter 4 Capturing the Requirement





- 用例图着重于从系统外部执行者的角度来描述系统需要提供哪些功能，并且指明了这些功能的执行者是谁；
- 用例图在UML方法中占有十分重要的地位，人们甚至称UML是一种用例图驱动（见下页解释）的开发方法。

用例图中的图符：

-  用例
-  执行者
-  系统：用于界定系统功能范围(边界)，描述该系统功能的用例都置于其中，而描述外部实体的执行者都置于其外。
-  关联：连接执行者和用例，表示执行者所代表的系统外部实体与该用例所描述的系统需求有关。

Chapter 4 Capturing the Requirement

用例图中的其他图符：

-  使用：由用例A连向用例B, 表示用例A中使用了用例B中的行为或功能。（use/include）
-  扩展：由用例A连向用例B, 表示用例B描述了一项基本需求，而用例A则描述了该基本需求的特殊情况。（extend, 部分相同，不是包含）
-  注释体：对UML实体进行文字描述
-  注释连接：将注释体与要描述的实体连接，说明该注释体是针对该实体所进行的描述。
- 注：上述图符中的“使用”和“扩展”的概念在不同的文献有着一定的差别，解释有所不同，各个机构的实际使用方式也有所不同。
- 其他图符：refine（精炼；引导实现），继承，依赖等等；有的还是自定义式的，例如：可以添加《love》关系，让两个用例之间产生爱情，而且还可以是单相思。

Chapter 4 Capturing the Requirement

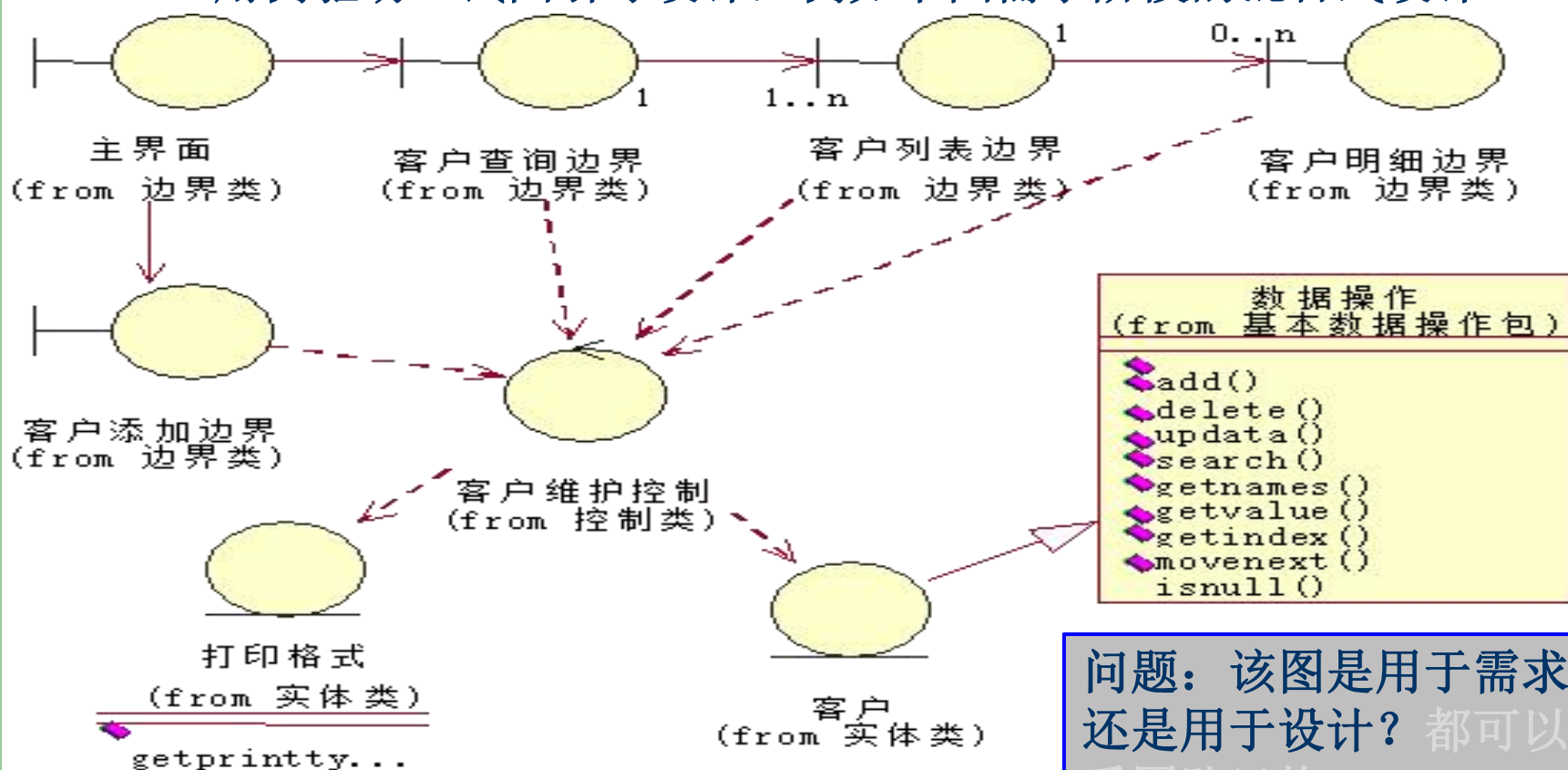
关于“用例驱动”的若干解释：

- 用例驱动战略可以通过在该问题中添加几个字来描述：需要该系统为每个用户做什么？这几个字有着重大意义。它们迫使我们从用户的利益角度出发进行考虑，而不仅仅是考虑系统应当具有哪些良好功能。
- 用例并不仅仅是定义一个系统的需求的一个工具，它们还驱动系统的设计、实现和测试。
 - 基于用例模型，软件开发人员创建一系列的设计和实现模型来实现各种用例。开发人员审查每个后续模型，以确保它们符合用例模型。测试人员将测试软件系统的实现效果，以确保实现模型中的组件正确实现了用例。这样，用例不仅启动了开发过程，而且与开发过程密切结合在了一起。（用例驱动实际是关注用户的利益驱动）
 - “用例驱动”意指开发过程将遵循一个流程：它将按照一系列由用例驱动的工作流程来进行。首先是定义用例，然后是设计用例，最后，用例是测试人员构建测试案例的部分来源。（不少测试用例专门基于用例，比如一些特殊的互动场景是用例所表现出来的）

Chapter 4 Capturing the Requirement

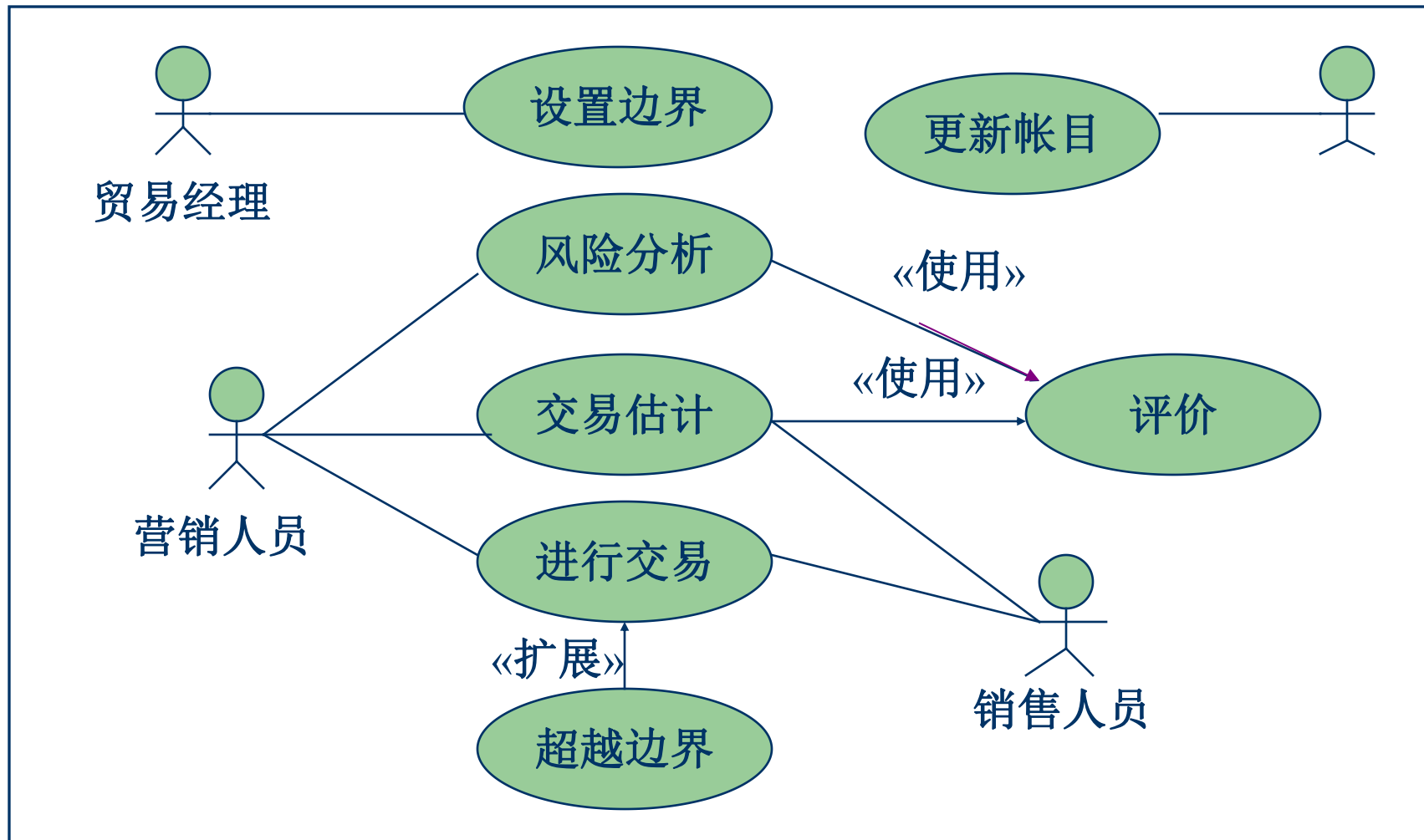
关于“用例驱动”的若干解释（观摩）

■ “用例驱动”试图引导设计，例如下图需求阶段的混合式设计：



问题：该图是用于需求还是用于设计？都可以，看团队风格

Chapter 4 Capturing the Requirement



Chapter 4 Capturing the Requirement

用例模型的获取：

- 获取执行者
- 获取用例

获取执行者：

- 谁使用系统的主要功能（主要使用者）？
- 谁需要系统支持他们的日常工作？
- 谁来维护、管理系统使其能正常工作（辅助使用者）？
 - 例如：教材购销系统的“各班学生用书表”需要定期维护，于是得有“系统维护”用例以及与其打交道的“执行者”。
- 系统需要控制哪些硬件？
- 系统需要与其他哪些系统交互？
- 对系统产生的结果感兴趣的是哪些人？（哪怕只能浏览）

Chapter 4 Capturing the Requirement

获取用例：

- 执行者要求系统提供哪些功能？
- 执行者需要读取、产生、删除信息有哪些类型？
- 必须提醒执行者的系统事件有哪些？
- 执行者必须提醒系统事件有哪些？成为用例中的功能？

关键问题：

刻画用例图详细到什么
时候为止？

答：一直画到接近系统内部实现就可以停止分解

注意事项：

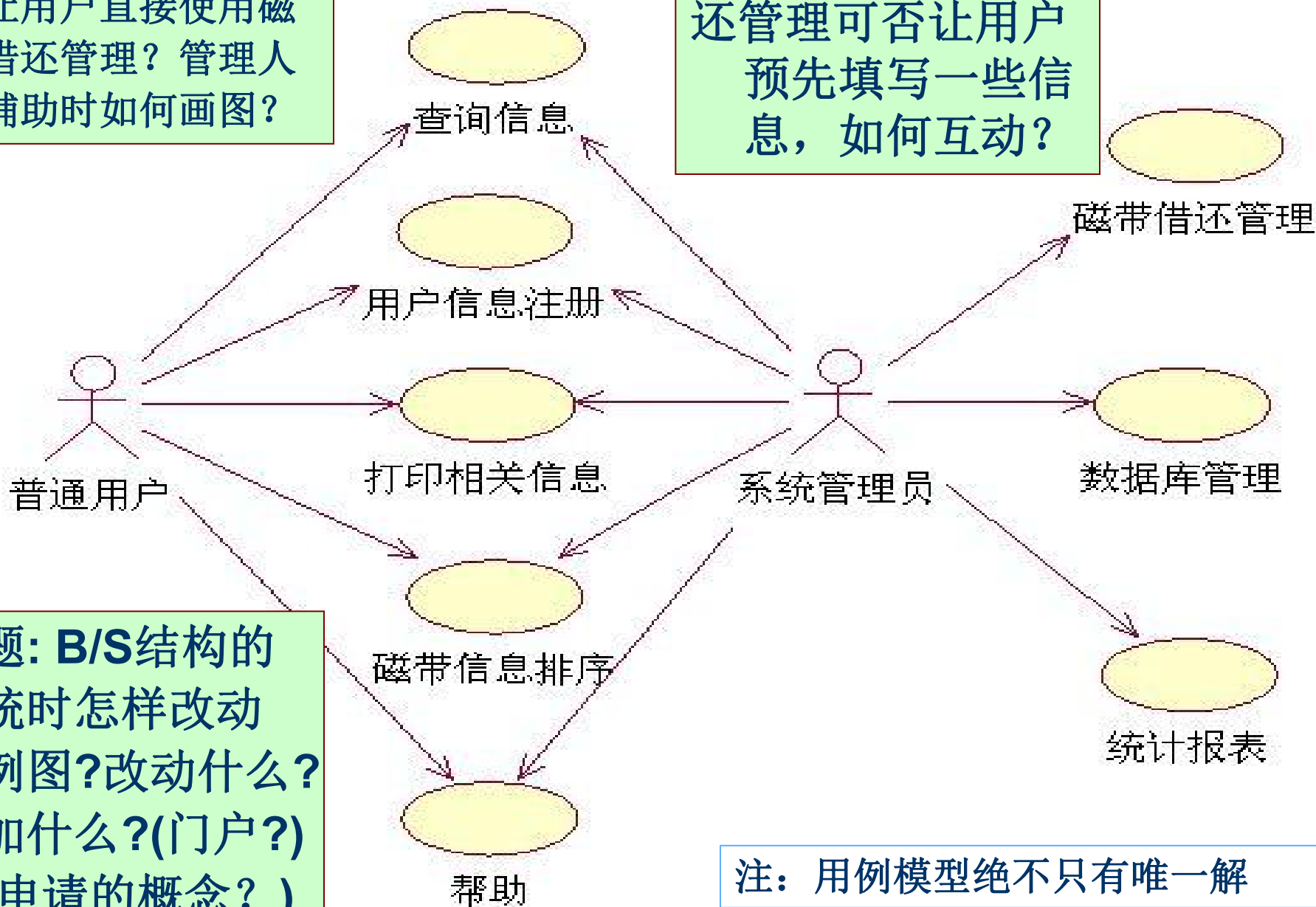
- 用例只跟参与者打交道，不能把功能分解成大堆用例。
- 用例不能是内部实现，也不能没有结果。

例模型

问题3: 鉴于手续严格, 磁带借还管理可否让用户预先填写一些信息, 如何互动?

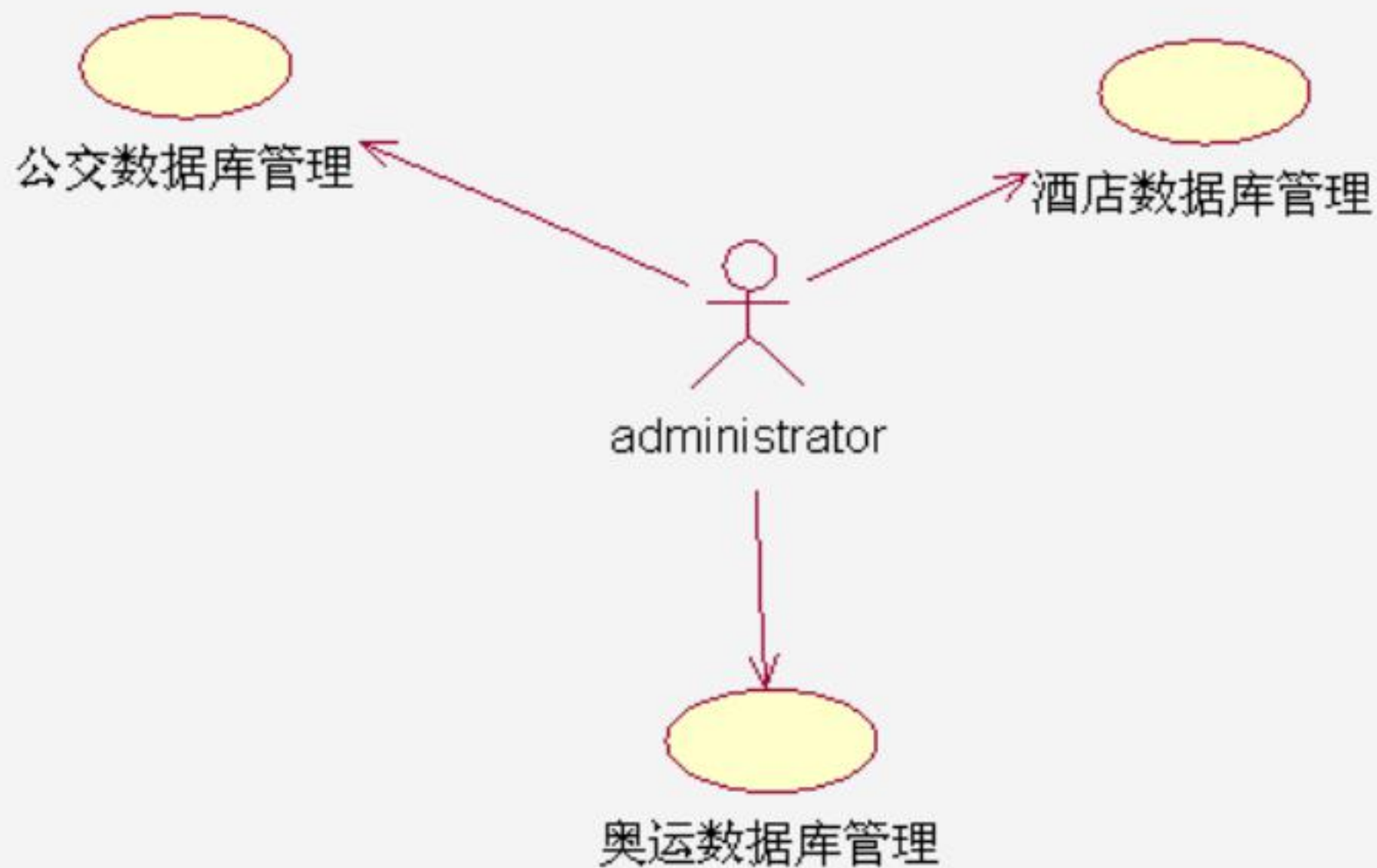
磁

问题: 有没有别的系统设想? 例如: 能否可以让用户直接使用磁带借还管理? 管理人员辅助时如何画图?



问题: B/S结构的系统时怎样改动用例图? 改动什么? 增加什么?(门户?) (预申请的概念?)

注: 用例模型绝不只有唯一解



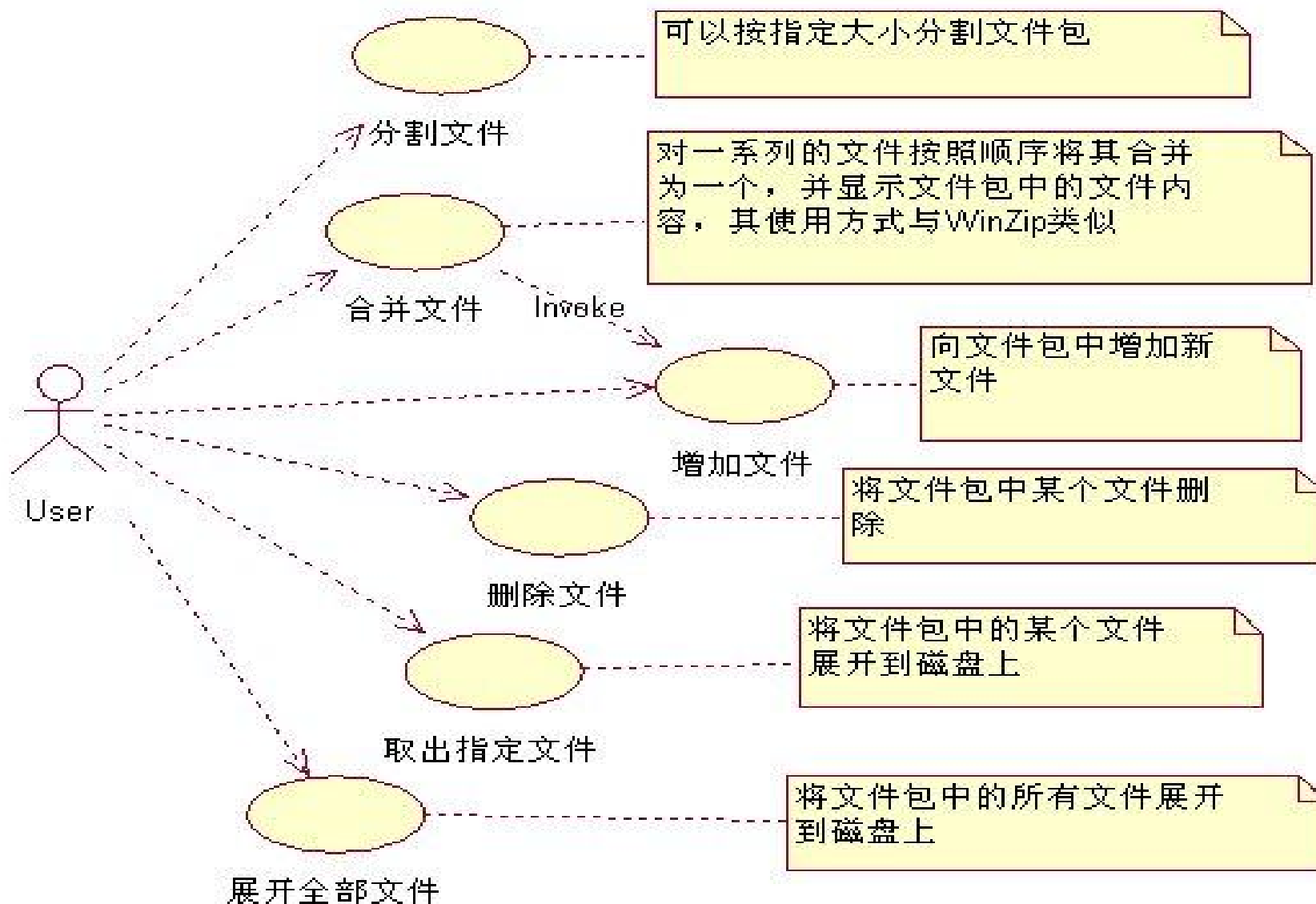
OLYMPIC03-3用例规约：

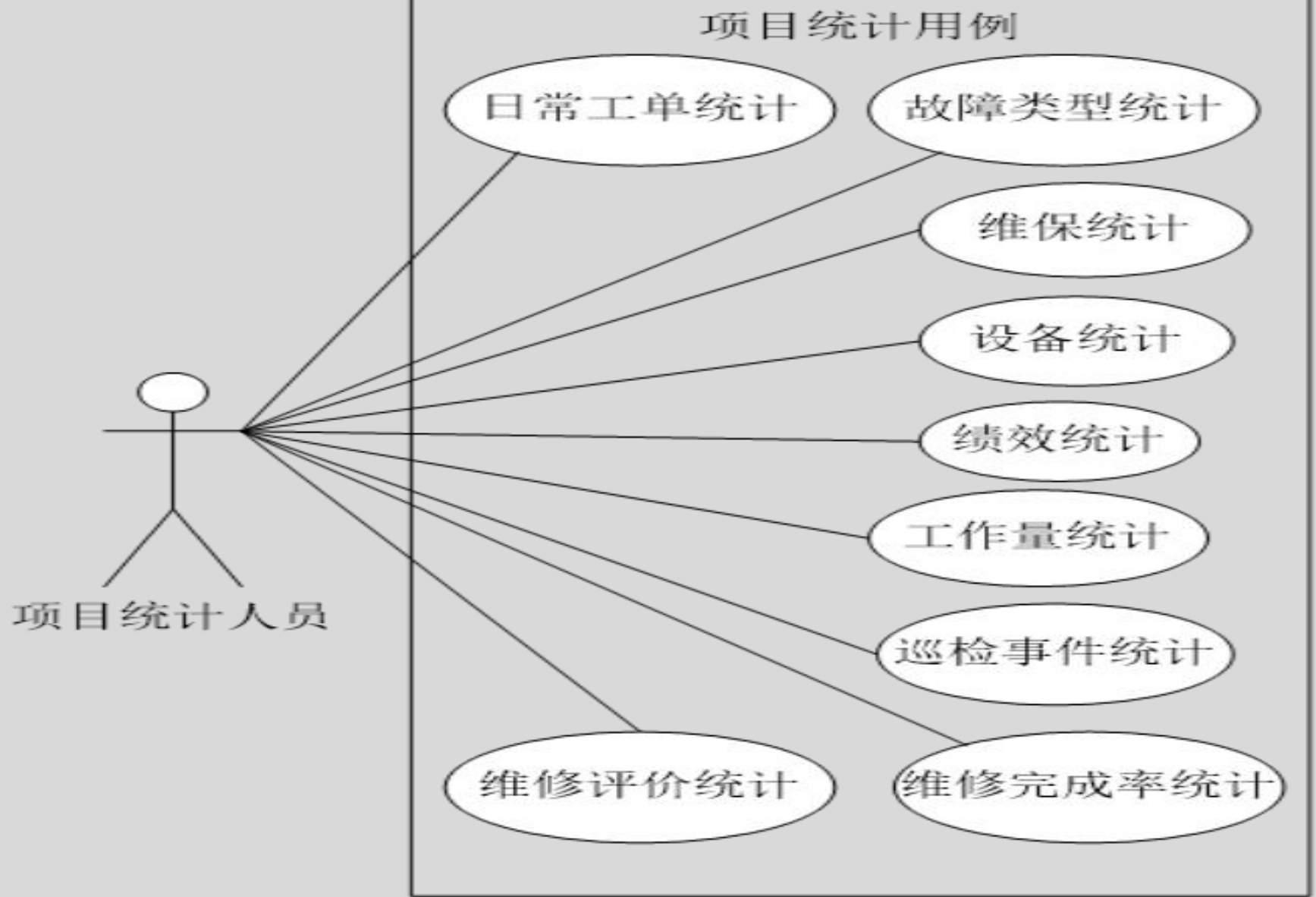
用例名称：	酒店数据库管理
角色：	管理员
用例ID：	OLYMPIC03—3
说明：	管理员进入数据库进行更新并传到客户端
输入：	需要修改的酒店数据库信息
加工：	无
输出：	无
前置条件：	管理员进入服务器端
基本事件流：	1. 管理员根据实际情况更新酒店数据库， 2. 将更新后的数据库传到客户端。
可选事件流：	管理员输入的数据与数据类型不匹配，系统提示错误并重新输入。
后置条件：	数据库得到更新并传到客户端

文件分割合并器系统用例图

程序设计：金旭亮

最新修改时间：2001年9月29日





用例：	项目统计
参与角色：	项目统计人员
前置流：	角色登录
事件流：	<ol style="list-style-type: none"> 1. 登录Web端后，根据时间范围统计各状态下的日常工单信息。 2. 对故障类型进行统计，查看故障发生频率。 3. 执行维修完成率统计，结果包括全年维保数、已完成、进行中等状态下的数量。 4. 依据截止日期条件间进行设备统计，结果包括设备总数、运行数量、故障数量、停运、报废等状态下的数量。 5. 进行绩效统计，查看总积分、基础积分、抢单积分等统计结果。 6. 依据时间周期、起止时间等统计工作量信息。 7. 登录APP端，进行全部工单统计，查看共单项情内容，包括工单的编号、报单位置、故障类型、报单状态、故障图片等。 8. 进行维修完成率统计，查看报修总数、已完成、特殊工单等状态的统计结果。 9. 进行专业分组统计，查看环状图等统计结果。 10. 统计故障发生率，显示指定时间范围内故障发生情况，包括故障分类、故障类型、报修数量、比例等。 11. 查看维保计划内容，包括维修人、编号、维保项目等。 12. 统计维保完成率情况，包括已完成、进行中、未完成等状态。 13. 统计设备完好率情况，根据设备的编号、名称等确定设备。 14. 指定工作员基础上进行员工工作量统计，以分组为单位，查看柱状图等统计结果。 15. 进行维修评价统计，查看分组统计结果。
异常流：	APP 端的统计结果中环状图无法加载。
结果流：	<ol style="list-style-type: none"> 1. Web端的日常工单、维保、设备、绩效等统计结果可返回。 2. APP端中项目工单、维修完成率、专业分组、故障发生率的统计结果可返回。 3. APP端中维保完成率、设备完好率、维修员工作量、维修评价、维保计划的统计结果可加载。

Chapter 4 Capturing the Requirement

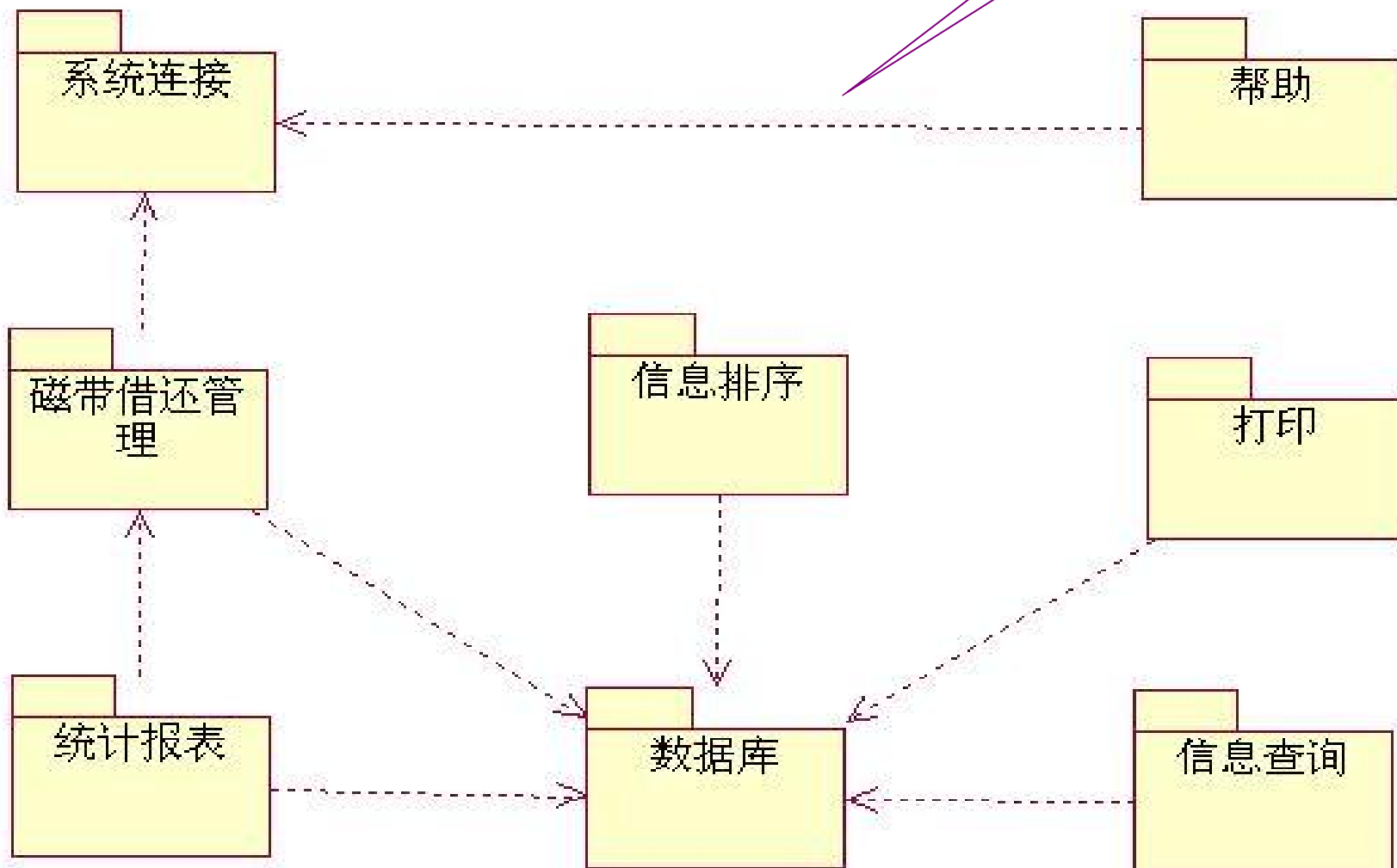
(2). UML Class Diagrams (类图)

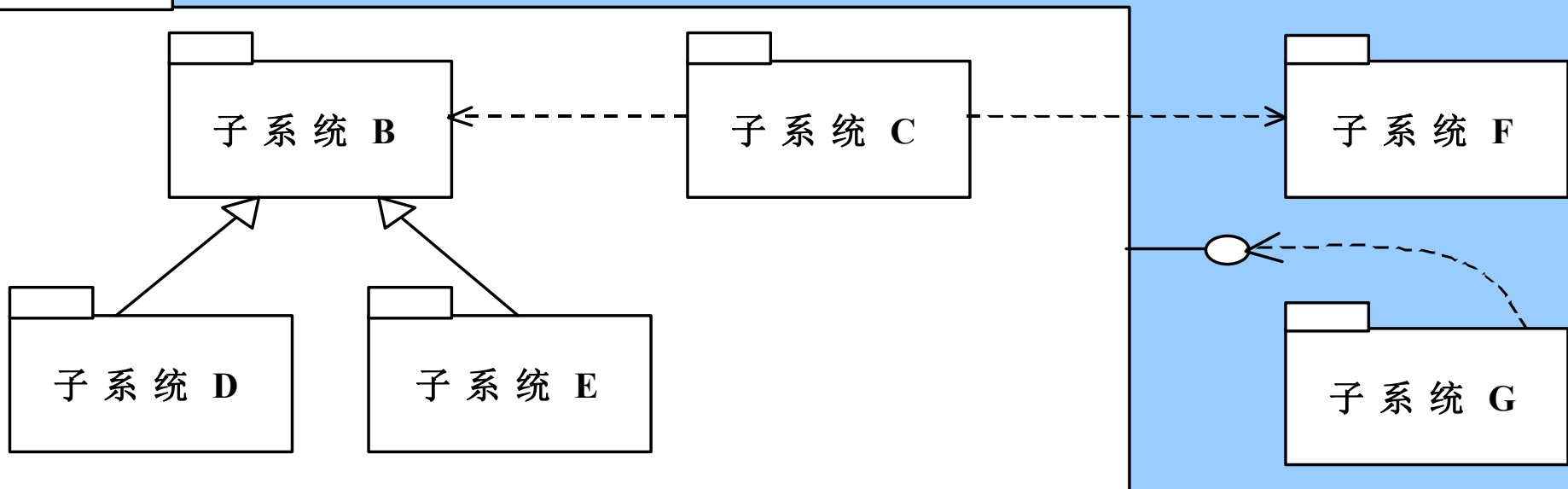
- 在面向对象的建模技术中，类、对象和它们之间的关系是最基本的建模元素。对于一个想要描述的系统，其类模型、对象模型以及它们之间的关系揭示了系统的结构。
- 类图描述了系统中的类及其相互之间的各种关系，其本质反映了系统中包含的各种对象的类型以及对象间的各种静态关系（关联、子类型等关系）。
- 分析阶段类的种类：边界类，实体类，控制类（查阅相关文档，参看其具体画法）
- 详见第六章----需求及设计阶段类图的生成演化过程

(3) Other UML diagrams

包图

A: 包图的结构

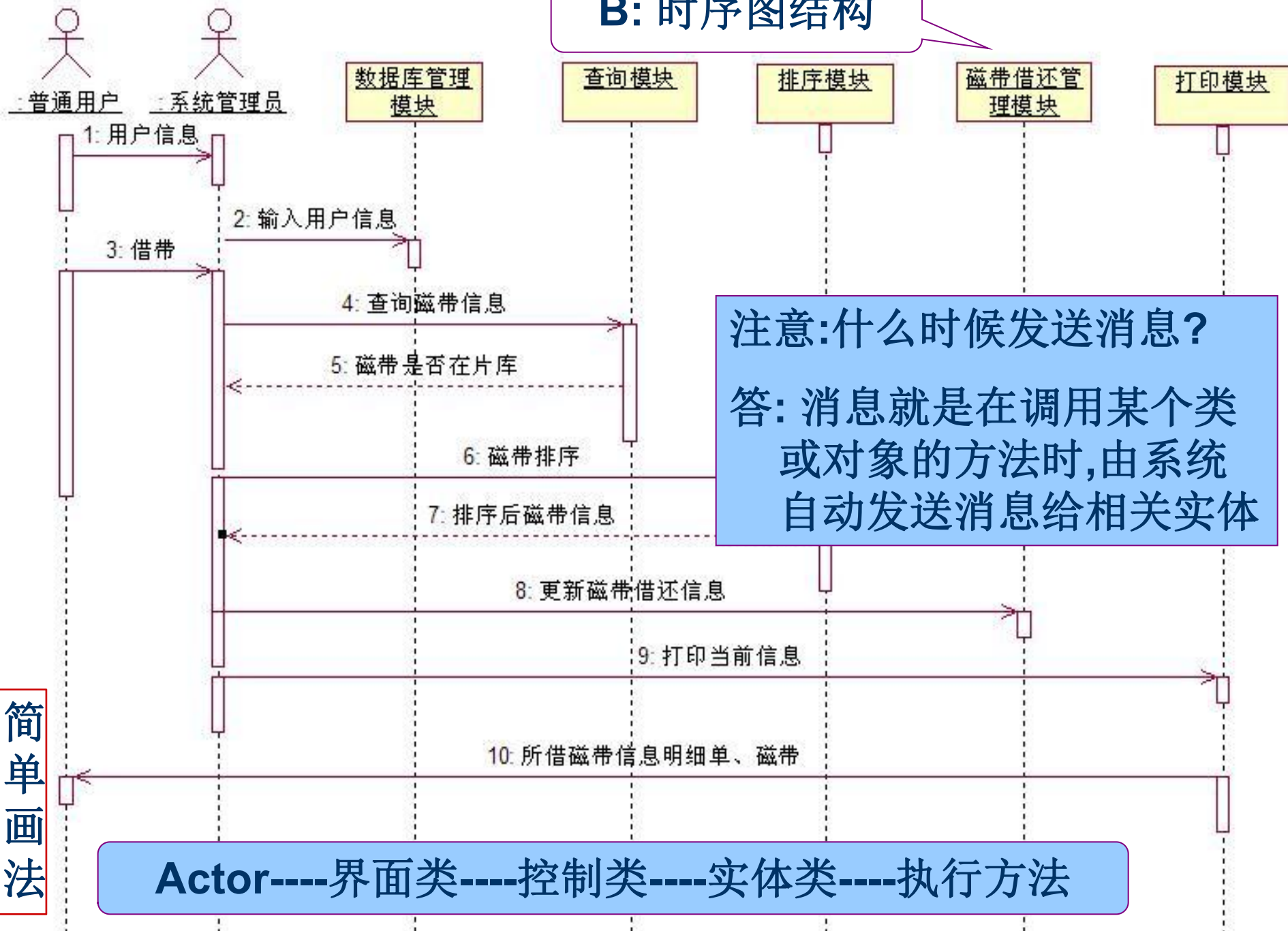


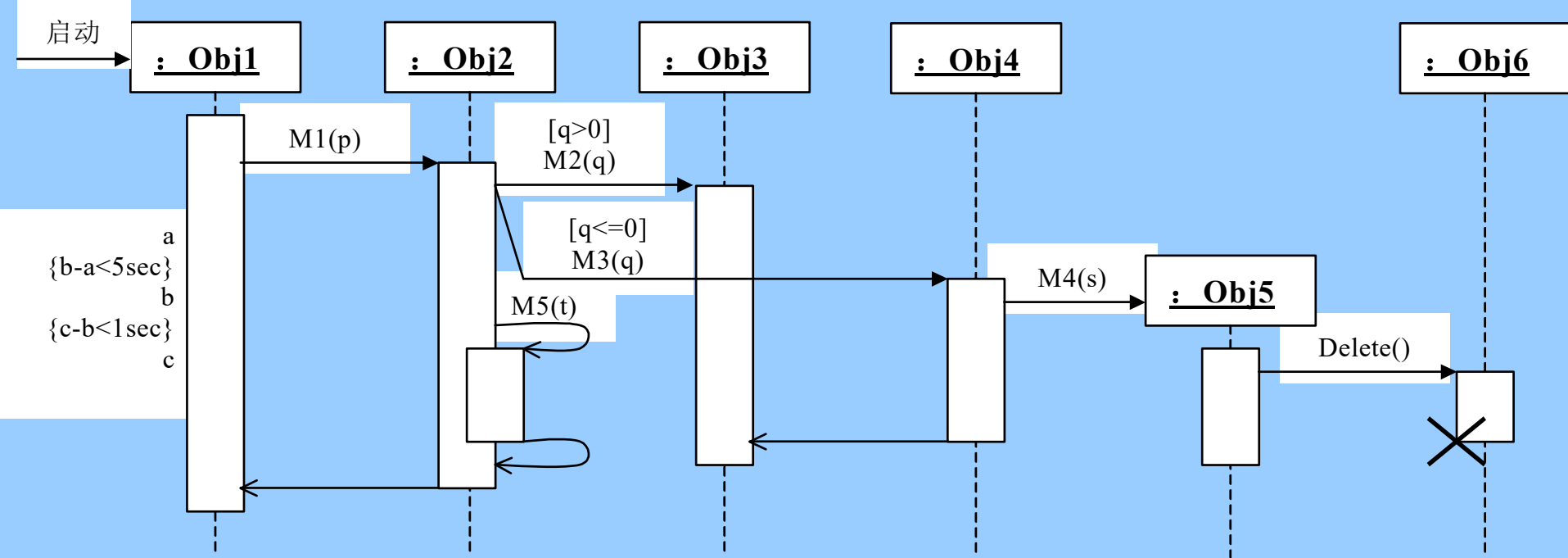


图##.31：包的图示和包之间的关系

- 图##.31说明了包的图示表达和包之间的关系。由图可见，包之间也存在类似类之间的继承、引用等依赖关系；包也可以有接口，接口与包之间用带小圆圈的实线相连。图中，**B、C、D、E**在**A**中，**C**依赖于**B**，**C**也依赖于外部的**F**，而**D、E**分别继承了**B**，而**G**使用了软件系统包**A**的接口。
- 包也有可见性，例如私有、公有、保护，用以控制外部包对包中内容的访问。

B: 时序图结构





图##.35：时序图的图示表示举例

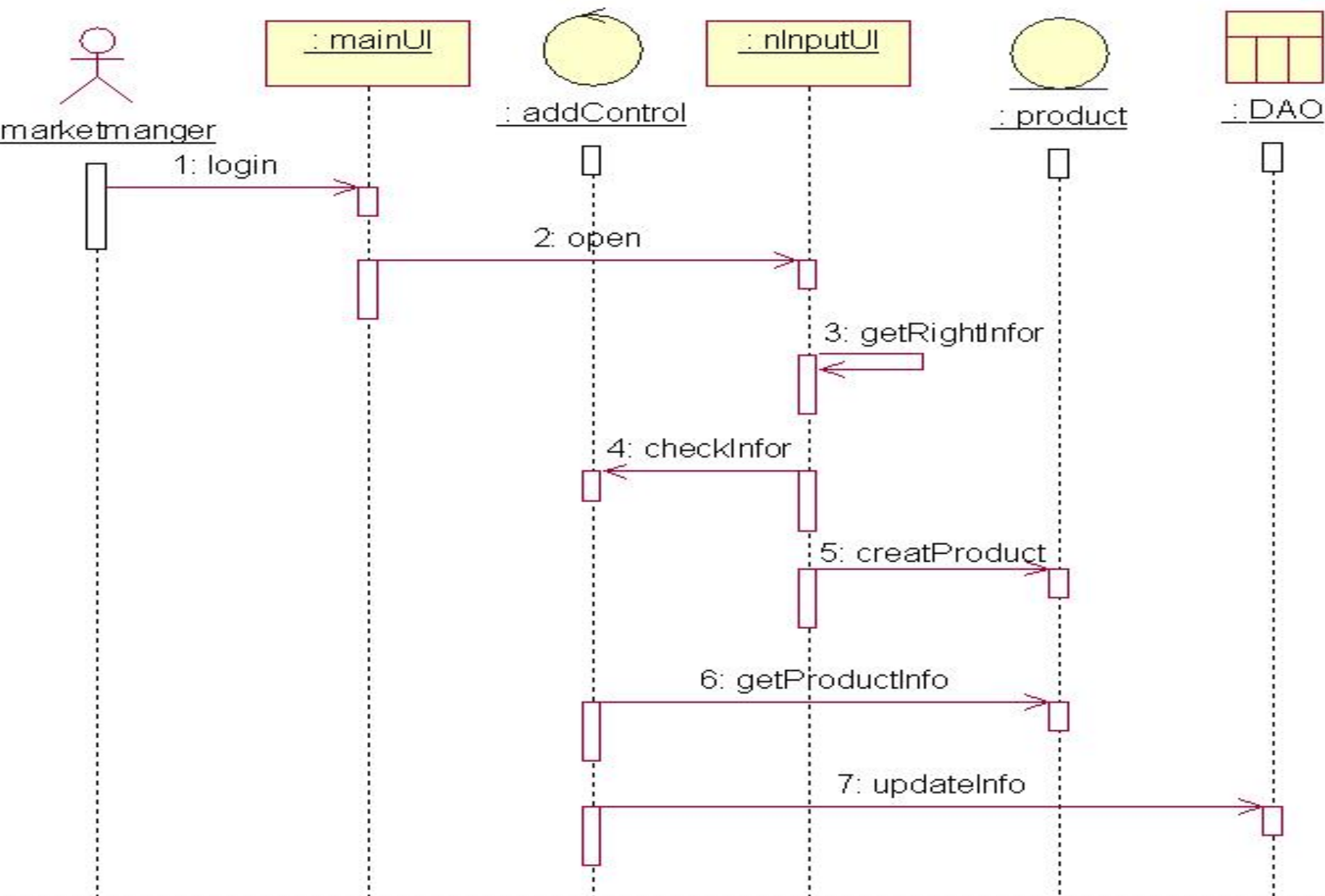
复杂画法

B: 序列图：图##.35给出了图示，其中，**M2**和**M3**是有条件的互斥消息，**M4**消息创建了对象**Obj5**，它的**delete**消息则撤销了对象**Obj6**，**M5**消息进行了递归调用，**Obj1**左面的标签和约束指明了**a**、**b**、**c**三点之间的时间约束要求。

序列图中的对象可以是并发执行的，每一个对象有自己运行的线程控制着。这时，需要通过激活、异步消息、同步控制和活动对象来表示。

序列图有两种，一种是描述特定对象之间生存期中消息通信的所有情节，称作一般序列图；一种是描述消息通信的个别情节的实例序列图，如果需要描述所有的情节，则需要多个实例序列图。

雅芳电子商务系统---添加产品



Chapter 4 Capturing the Requirement

c: 部署图（构件图/组件图）

----部署图描述了系统在运行时的物理结构、配置和关系，涉及处理器、设备、通讯等硬件单元和软件部件。部署图的描述是基于代表硬件单元节点之上的。

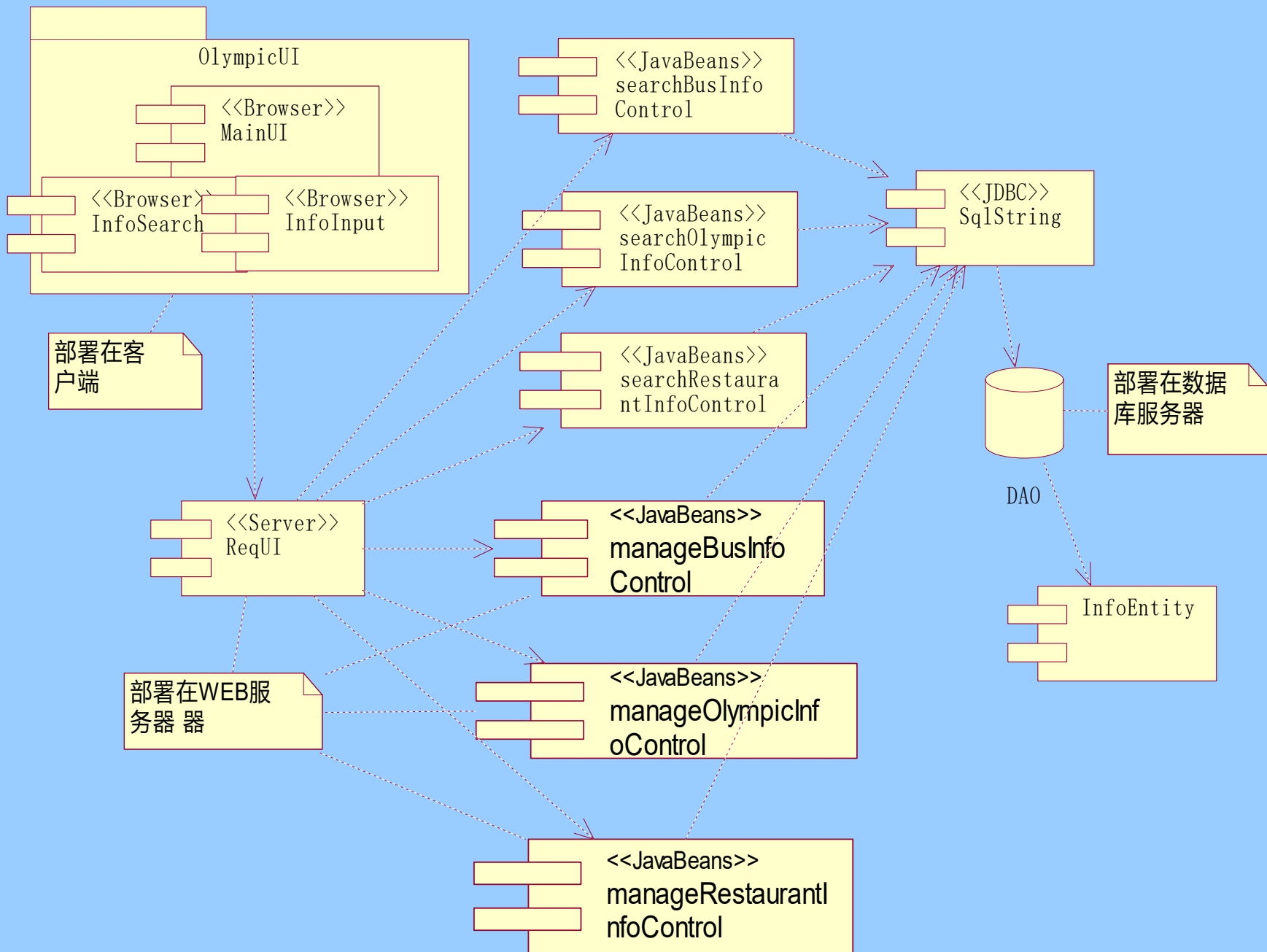
----节点是系统中计算资源的代表，每个节点是一个计算机设备及受其管理支配的、处理器、打印机、通讯设备等，以及在其上运行的类、对象等部件。通过这些部件的识别，可以跟踪确认系统的组成以及组成间的关系和在硬件上的配置关系。

----节点按照硬件资源标识。节点的描述包括三方面：位置（标识）、构成（包含的硬件设备和软件部件）、能力（计算能力、计算功能等）。

----可以把节点作为类或实例加以描述。节点类描述了具有相同特性的等待实例化的节点，实例代表了实际出现的节点。（有时候把某个客户端描述成一个综合节点类）（此设计方法为的是支持大规模部署。）

Chapter 4 Capturing the Requirement

- 在把部件分配给节点时，需要考虑许多因素。例如：资源和能力的利用、地理位置对功能实现和系统性能的影响、资源配置与效率的发挥、保密和安全性的实现、系统综合性能的影响、可扩展和可移植等特性的影响。
- 复杂的系统需要在全面分析的基础上建立布署图，并在深入和详细研究后对布署图进行扩展和完善，必要时需要综合考察不同设计方案。
- 奥运福娃架构图及部署图（如下页图所示：）
采用**MVC**模型，实现表现层和控制层的分离，提高可重用性、可移植性。
注意：在客户端、**web**服务器、数据库服务器等的具体部署情况。

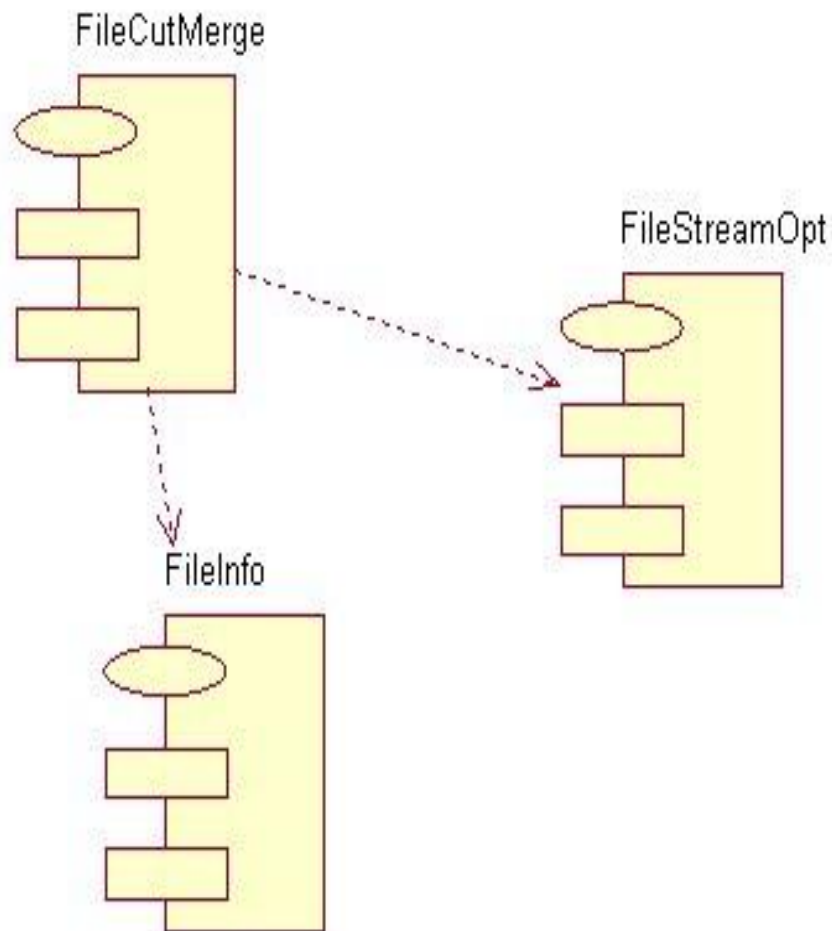


➡ 一般用组件图来生成代码

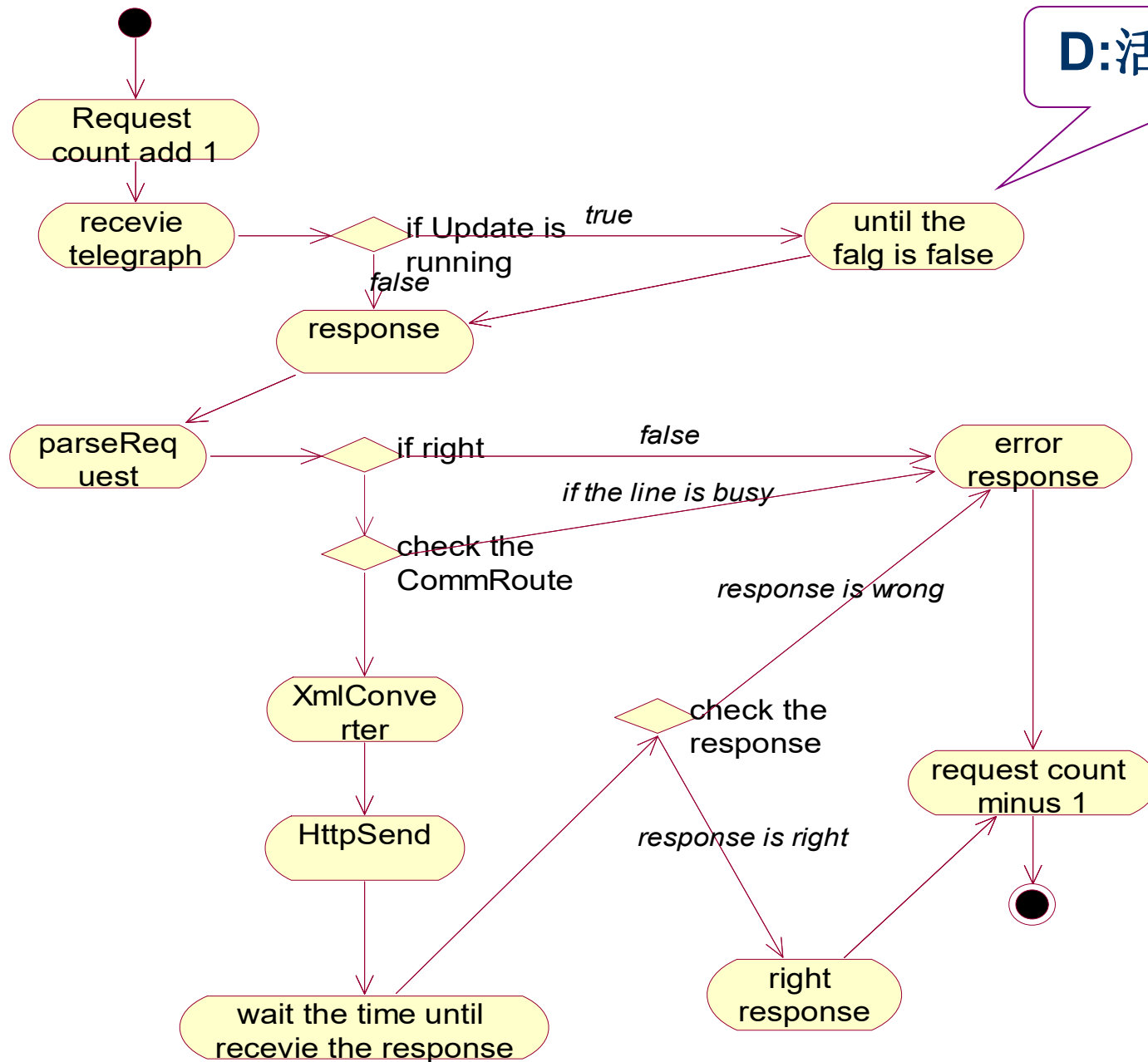
使用**ROSE**进行正向代码生成：

- 1：设计类接口
- 2：新建一个组件
- 3：将类赋与组件
- 4：生成代码

ROSE的上述介绍仅为参考。
但有一些类似的开发环境，使用类似的手段开发软件。



D:活动图结构



Chapter 4 Capturing the Requirement

2+. 类图的设计-----与软件设计的模式有关

- 不会设计模式就不是真正的编程高手, 更不是高水平的系统分析员和优秀的软件设计师!
- 数十种软件设计模式, 数十个软件设计水平成长的故事!(号称!)
- 设计模式浅谈-----薪资计算实例 (附加java程序及类图)
 - 简单工厂模式
 - 工厂模式
 - 抽象工厂模式
 - 其他若干模式等。
- 设计模式虽在别的课程里面有所介绍, 但这里将从工程项目需求变更、质量与软件效率的角度介绍其部分思路。

Chapter 4 Capturing the Requirement

3. How to Express Requirements (需求的表达方式)

Note: A: drawback in using natural language

(其一: 所有使用者必须用同样方式解释含义,但这很难
其二:用自然语言写需求,则不易识别系统的各种元素)

B: benefit defining requirement in formal notation

(以严格可控制的方式定义需求,将使其易追踪和管理)

(1). Static descriptions(静态描述) : (only define the entities/objects, their attributes and the relationships, not describe how relationships change with time)

① **Indirect reference (间接引用)**

- Example: k equations in n variables

② **Recurrence relations (递归关系)**

- Example: $F(0)=1; F(1)=1; F(n+1)=F(n)+F(n-1)$

③ **Axiomatic definition (公理式定义)**

Chapter 4 Capturing the Requirement

④ Expression as a language (形式化语言表达式)

- Example: Backus-Naur form---将需求描述成正则语言的“串”

⑤ Data Abstraction (数据抽象)

----is a technique for describing what data are for,
rather than how they look or what they are called

Note: data-type dictionary(only list data type)

data modeling: data-type+relationships

A: data-type/class

- Example : student record
----define the data type by text style

实际上也是
对象图

B: relationship: -----the relation between objects

include the relation between data-types

Chapter 4 Capturing the Requirement

Semester record

Semester type

Semester date

Grade-point average

Completed hours

Semester type

(Fall, Spring, Summer)

Address information

Telephone number

Street address

City

State

Postal code

Student record

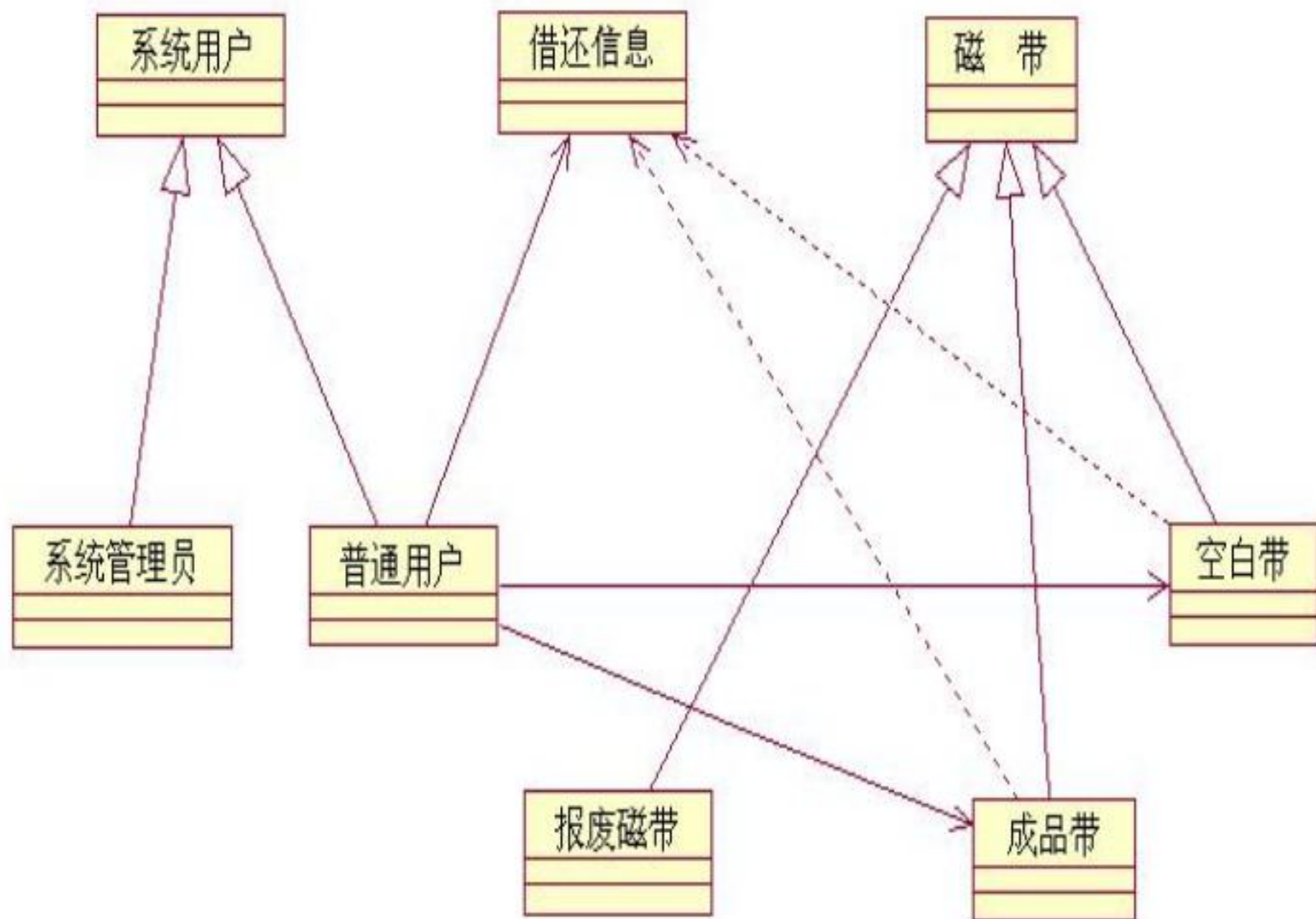
Name

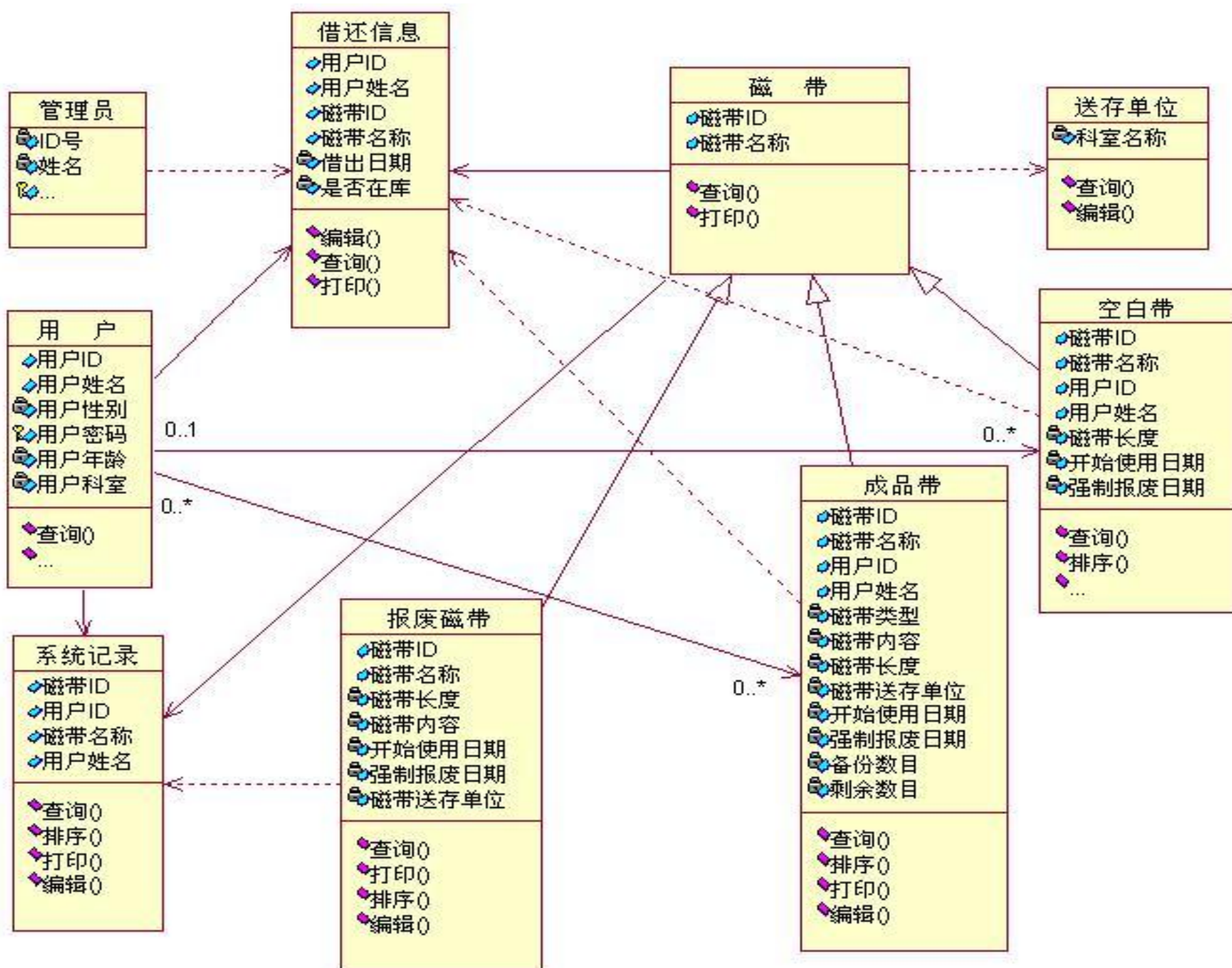
Student number

Address information

Number of semesters

{Semester record}





Chapter 4 Capturing the Requirement

问题讨论:

- (1): 由描述数据类型极其处理.可以导出类图.(假设此时不考虑存储效率的话。)
- (2): 由描述功能类型极其涉及的数据.也可以导出类图.
(以上是两种设计理念)
- (3): 若类图等还不足于描述数据关系,那只好再用别的非UML描述手段甚至文字等予以补充了.(例如ER图等等很多类型)

Chapter 4 Capturing the Requirement

(2). Dynamic descriptions (动态描述)

system states $\xrightarrow{\text{stimulus(激励)}}$ change over time \rightarrow general description
(dynamic description)

① Decision tables (判定表)

conditions \rightarrow rule \rightarrow action

Table 4.1. Decision table.

	<i>Rule 1</i>	<i>Rule 2</i>	<i>Rule 3</i>	<i>Rule 4</i>	<i>Rule 5</i>
High standardized exam scores	T	F	F	F	F
High grades	-	T	F	F	F
Outside activities	-	-	T	F	F
Good recommendations	-	-	-	T	F
Send rejection letter			X	X	X
Send admission forms	X	X			

Chapter 4 Capturing the Requirement

- ② **Functional descriptions and transition diagrams**
(功能性描述和变迁图)

$$f(S_i, C_j) = S_k$$

(3). **Object-oriented specifications** (面向对象的规格说明)

- ① **note: A: functional approach**

----describes how inputs are transformed to outputs

- B: object-oriented approach**

----focuses on the entities enacting

X: what do entities look like?

Y: what can we do with the entity?

Z: what aspects of entities and processes are
persistent over time?

Chapter 4 Capturing the Requirement

②several notions

- A: object: Each entity in the system is an object
- B: method or operation: is an action that can be performed directly by the object or can happen to the object.
- C: Encapsulation: the methods form a protective boundary around an object
- D: Class hierarchies of objects---- (encourage inheritance)
- E: Multiple inheritance:
- F: Polymorphism: same method for different objects, each with different behavior

Chapter 4 Capturing the Requirement

- ③advantages: is that the object and method descriptions are closely associated with the application domain for which the software is to be built
(more practical than traditional approach)
- ④three models of OMT(Object Modeling Technique)
 - A: object model ----对象的继承关系等等(P160)
 - B: dynamic model ----状态图等(P166)
 - C: functional model: like DFD (in P172), use case , etc.

Chapter 4 Capturing the Requirement

4. Additional Requirements Notations

（其他的需求描述系统）

Data Flow Diagrams（**DFD**数据流图）(P172)

①purpose: describe how the data flows into the system, how they are transformed, and how they leave the system. (the emphasis is always on the flow of the data, not on the flow of control)

②organization:



加工



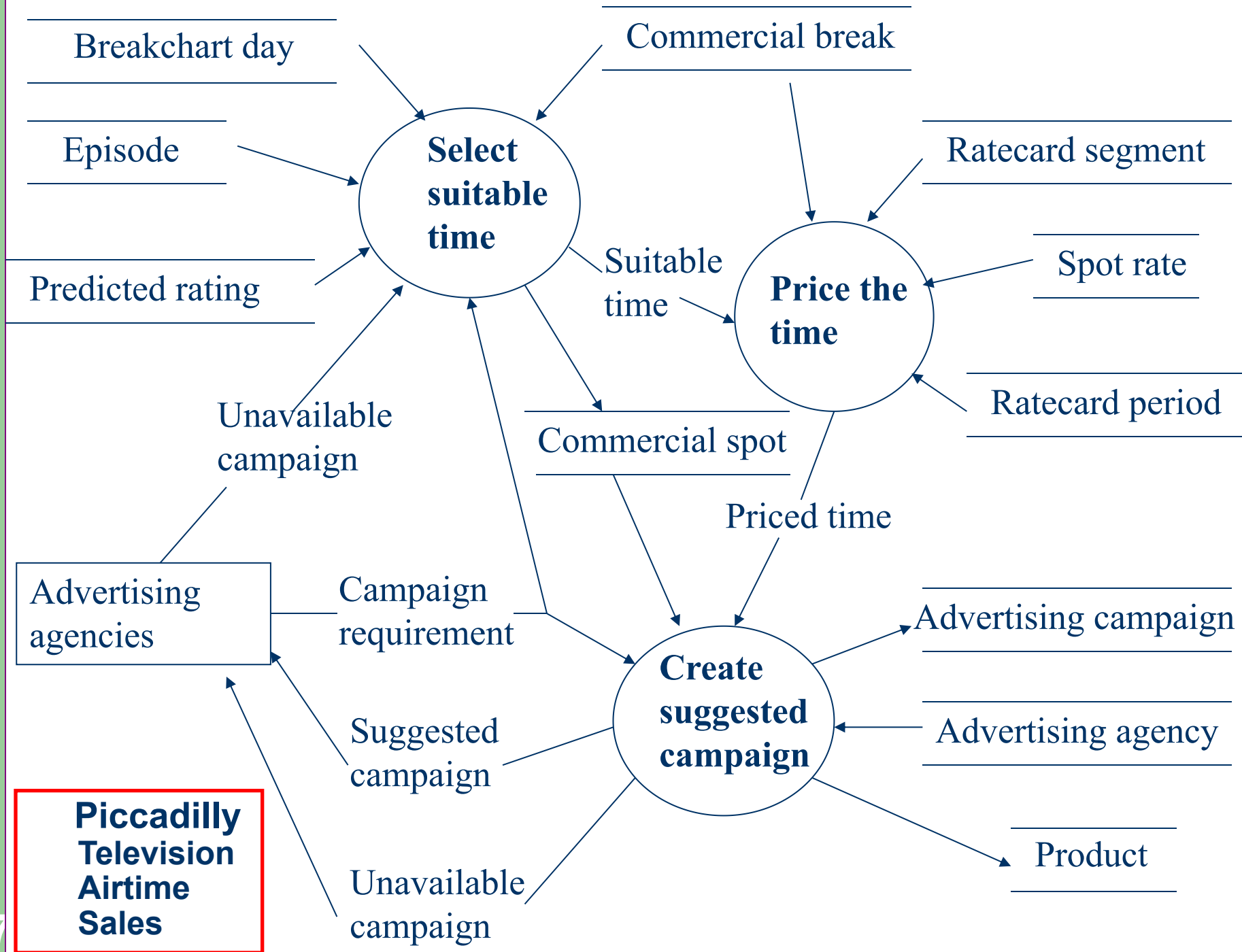
数据流向



数据集合



外部项



Chapter 4 Capturing the Requirement

5. Object Constraint Language(OCL) (大多用于设计)

----is specially designed for **expressing constraints on object models**(对象的约束说明), and introduces language constructs for navigating from one object to another via association paths, for dealing with collections of objects, and for expressing queries on object type . (可追踪状态、属性变化, 但很少用。)

----**example: figure 4.18** 。 (部分图书馆类的模型)

----最左边约束: 规定主顾的罚金不能是负数。(Fines \geq 0)

----最上面约束: 通过对某些实例和对象中属性和操作的一阶运算性说明, 指出了其图书编目号码的唯一性。

----最下面约束: 方法borrow()操作的前置条件和后置条件。

----**对象约束语言(OCL)** , 是常态化的类图说明和注释。

Publication.allinstances->forall(p1, p2 |
p1 <> p2 implies p1.callnumber <> p2.callnumber)

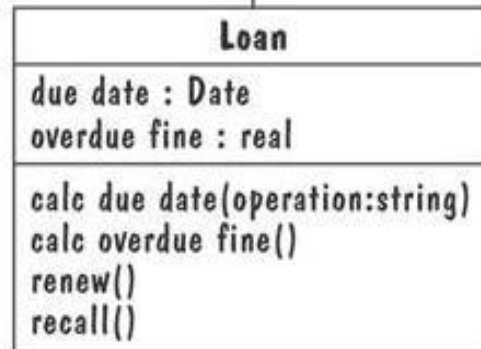


fines ≥ 0

0..1
borrower

borrows

0..*



«precondition»
borrower.fines = 0 and loan state = inlibrary
«postcondition»
post: loan state = onloan

Chapter 4 Capturing the Requirement

4.6 Requirements and Specification Languages

1. UML (OMG2003) (统一建模语言)

1+9 graphical modeling notations

+ OCL constraint language

- 用例图（一种高层的DFD图）
- 类图（一种ER图）
- 时序图与协作图（一种事件流踪迹）
- 状态图（一种状态机模型）
- OCL特性（逻辑描述）

Chapter 4 Capturing the Requirement

2. SDL(Specification and Description Language) (ITU2002) (规格说明和描述语言) (略)

three graphical diagrams

+ algebraic specifications for defining
complex data types

Chapter 4 Capturing the Requirement

4.7 Prototyping requirements(原型化需求)(P191)

cause:A: 用户有时不能严格确定自己的详细/完整的需求

B: 用户有确定的需求,但开发者自己无法肯定方案是否真实可行

C: 建立原型可帮助获取需求的细节(P191-s2-L8)---需改进的地方、多余的特征、缺失的功能等。以帮助用户确定最终方案。

1. **Throw-away prototypes** (抛弃式原型)

----仅用于了解问题、探索可行性，并不打算用来作为将来实际提交系统的一部分，而是用完扔掉。

2. **Evolutionary prototypes** (进化式原型)

----用于了解问题，并作为将来准备提交的系统的一部分。

 **Rapid prototyping (both upper techniques)**

 **feature:** ①prototyping before designing

②help to understand requirement to decide final design.

详细
评论
一个
现有
产品
比详
细想
象一
个新
产品
要容
易得
多!

Chapter 4 Capturing the Requirement

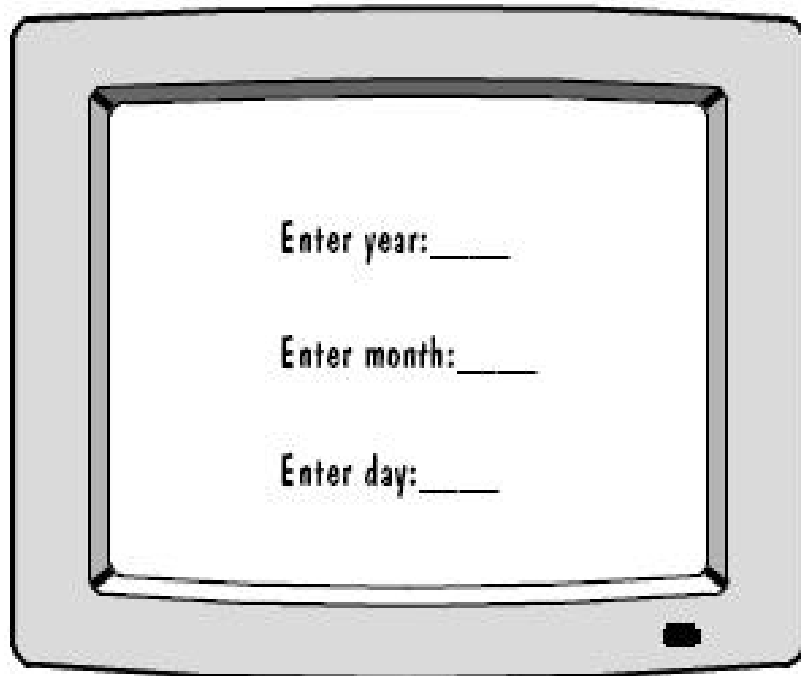
3. Example ----**prototyping use interface** (界面原型) in the stage of requirement capturing. (教练员笔记)
(fig4.23, 4.24, 4.25)

4. **Purpose:** (of prototyping requirements)

A: 有的需求难以用文字和符号说明,而原型可以帮助我们解决这个问题---- find “good look and feel” (原型化的过程可帮助我们找到 “好的视觉和感觉”)。

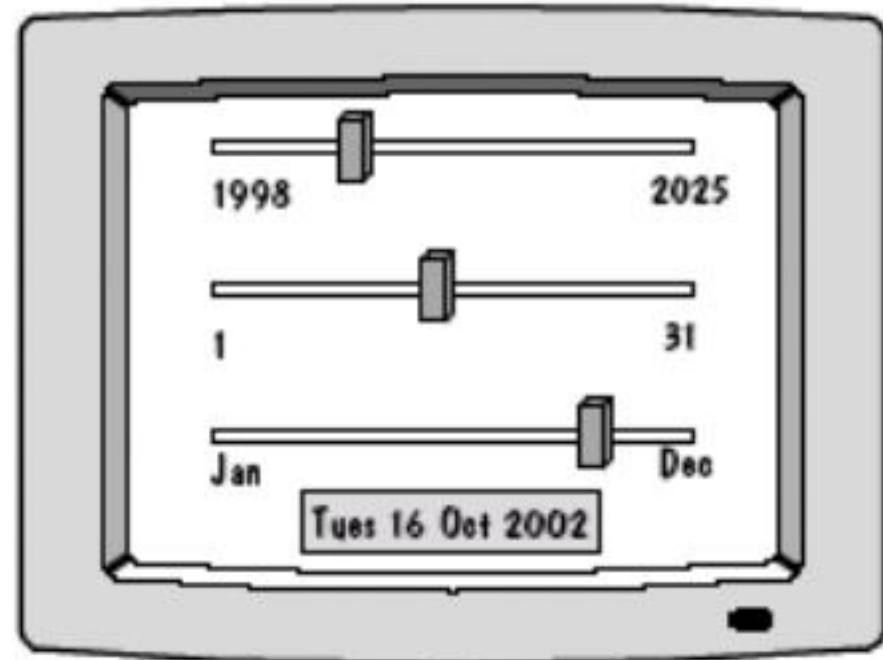
B: **evaluate efficiency and performance** (about nonfunctional requirements)。

(对非功能性需求, 可以评价性能和效率)。



Several prototypes

确定了进化式原型，才能较准确描述事件流



Chapter 4 Capturing the Requirement

4.8 Requirements documentation（需求文档化）

普遍问题：需求说明的层次的均衡性问题 (补充材料4-6)

1. **Necessity:** ① digital requirement is important
for cross-reference in every stage
② for configuration management

2. Requirements definition document (需求定义)

---- what the customer wants

- general purpose
- background and objectives of system
- description of customer-suggested approach
- detailed characteristics
- operational environment

Chapter 4 Capturing the Requirement

3. Requirements specification document (SRS: 需求规格说明) (Fig4-26参考)

----what the designers need to know

- Document system's interface
- Restate functionality formally
- Restate quality requirement

- ① explain: (mathematical models or formal terms)
- ② example: ----IEEE template of <SRS>下页
- ③ example: next page (其实就是形式化描述)

Example

4..1.3.1 INITIATE TRACK ON IMAGE. Logical processing shall be done to INITIATE TRACK ON IMAGE. This shall have as input HANDOVER DATA. This shall have as output HOIQ, STATE DATA, and IMAGE ID. This logical processing shall, when appropriate, identify a new instance of IMAGE. This logical processing, when appropriate, shall identify the type of entity instance as being IMAGE ON TRACK. NOTE: A request for pulses is made by entering a formal record into the HOIQ which feeds the pulse-send procedures.

ALPHA: INITIATE_TRACK_ON_IMAGE.

INPUTS: HANDOVER_DATA.

OUTPUTS: HOIQ, STATE_DATA, IMAGE_ID.

CREATES: IMAGE.

SETS: IMAGE_ON TRACK.

DESCRIPTION: “(4.1.3.1)A REQUEST FOR PULSE IS MADE BY ENTERING A FORMAL RECORD REQUEST INTO THE HOIQ WHICH FEEDS THE PULSE SENDING PROCEDURES.”

Chapter 4 Capturing the Requirement

4.8++ Participants in requirements process

1. Participants:

- Contract monitors
- Customers and users
- Business managers
- Designers
- Testers
- Requirements analysts

倾听、沟通、妥协
-----心量与素养

2. The ability about analysts:(“**sorting ability**”)

---- people skill + technical skill (社交能力加技术能力)

---- **table 4.1 (P148/next page)**

How developers see users

How users see developers

Users don't know what they want.

Users can't articulate what they want.

Users have too many needs that are politically motivated.

Users want everything right now.

Users can't prioritize needs.

Users refuse to take responsibility for the system.

Users are unable to provide a usable statement of needs

Users are not committed to system development projects.

Users are unwilling to compromise.

Users can't remain on schedule.

Developers don't understand operational needs

Developers place too much emphasis on technicalities.

Developers try to tell us how to do our jobs.

Developers can't translate clearly-stated needs into a successful system

Developers say no all the time.

Developers are always over budget.

Developers are always late.

Developers ask users for time and effort, even to the detriment of the users' important primary duties.

Developers set unrealistic standards for requirements definition.

Developers are unable to respond quickly to legitimately changing needs

Chapter 4 Capturing the Requirement

4.9 Requirements validation

1. Requirements validation

检查需求规格文档
与定义的对应性

复审在团队软件工程中是常规性方法

(1) 我院某学生在外资公司实习的失败经历(十六年前的幼稚: 我的程序凭什么得让别人检查啊?).

(2) 一个专科生在微软的成功(word文本)

determining that the
consistent with the
(definition)

definition document
be traced specification

③ techniques: table

一种常用的
validate方法

2. Requirements review

① Review goals and objections of system.

Compare requirements with goals and objectives.

② Describe operational environment.

Chapter 4 Capturing the Requirement

③ Examine

- interfaces
- information flow
- functions

④ Check for omissions,incompleteness,inconsistency

⑤ Document risk

⑥ Discuss how system will be tested.(test plan)

3.Example

Chapter 4 Capturing the Requirement

4.10 measuring requirements

1. Focus on: { product (definition and specification)
process
resource

2. Measurement

①product: A: number of requirement(size)

B: evaluating the requirement document

②process: the number of changes to requirement and the causes which led to change .

③resource(X)

Chapter 4 Capturing the Requirement

4.11 Choosing a Requirements Specification Technique

1. Evaluation for specification methods

- no technique that is best for all projects
- we must get a set of criteria to determine which technique is most suitable

2. A set of criteria

Chapter 4 Capturing the Requirement

2. A set of criteria

- **Applicability**
- **Implementability**
- **Testability/simulation**
- **Checkability**
- **Maintainability**
- **Modularity**
- **Level of abstraction/expressibility**
- **Soundness**
- **Verifiability**
- **Run-time safety**
- **Tools maturity**
- **Looseness**
- **Learning curve**
- **Technique maturity**
- **Data modeling**
- **Discipline**

Chapter 4 Capturing the Requirement

3. Example: table 4.4

Focus:

A: We must realize that no requirements specification technique is complete

B: We must consider “bounding” the characteristics of the individual project and the preferences of developers or customers

Value-Up Software Development



- **Moving from Work-Down to Value-Up Practices**
 - The Driving Forces for Change
 - The Business Context for Software Development
 - A Value-Up Approach to Software Development
 - Comparing Value-Up and Work-Down Practices
 - The Core Principles of Value-Up
 - The Importance of Project “Flow”
 - Measuring Flow
 - Exercise: Work-Down vs. Value-Up

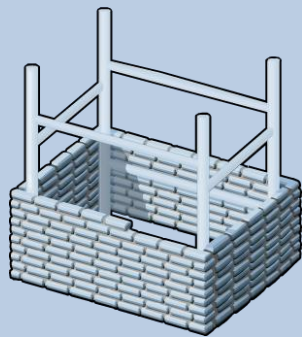
The Driving Forces for Change



The Business Context for Software Development

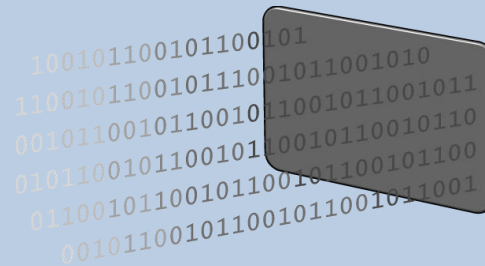
Microsoft

Civil Engineering



- Design risks are low
- Design cost is small relative to build cost
- Opportunity to deliver incremental value is rare

Software Engineering



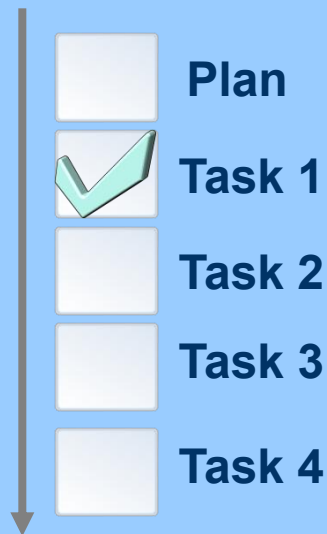
- Design can evolve as lessons are learned
- Design and build costs are relatively equal
- Delivering incremental value is beneficial

- Supporting Value-Up Practices with Visual Studio Team System (VSTS)

A Value-Up Approach to Software Development

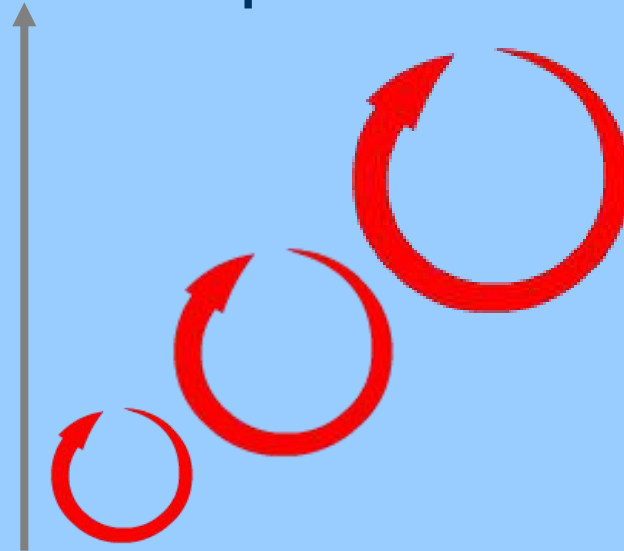
Microsoft

Work-down



- In traditional project management tasks are decomposed into items and checked off as completed

Value-up



- Value-up methods deliver incremental value for the customer to evaluate each iteration

Comparing Value-Up and Work-Down Practices

Microsoft

	Work-down	Value-up
Planning and change process	Get planning and design right up front	Change happens, embrace it
Primary measurement	Task completion	Only deliverables that the customer count
Definition of quality	Conformance to specification	Value to the customer
Acceptance of variance	Tasks can be identified and estimated deterministically	Variance is part of all process flows
Intermediate work products	Documents, models, and other artifacts are necessary	Just enough to minimize the uncertainty
Troubleshooting approach	Time, resource, functionality and quality	Find and remove the bottlenecks
Approach to Trust	Monitor and measure performance relative to plan	Pride of workmanship and teamwork

The Core Principles of Value-Up

- Increase ROI by making continuous flow of value the focus
- Reliable results by engaging customers at frequent intervals (shared ownership)
- Expect uncertainty and manage for it (through iteration, anticipation and adaptation)
- Unleash creativity and innovation by recognizing that the individuals are the source of value
- Boost performance through group accountability
- Improve effectiveness and reliability through specific strategies, processes and practices

The Importance of Project “Flow”

- Flow is central to value-up thinking and practices
 - Working software and completed documentation are the work products that count
 - Software to the customer is measured regularly
- Example, Extreme Programming and the flow of “customer value”

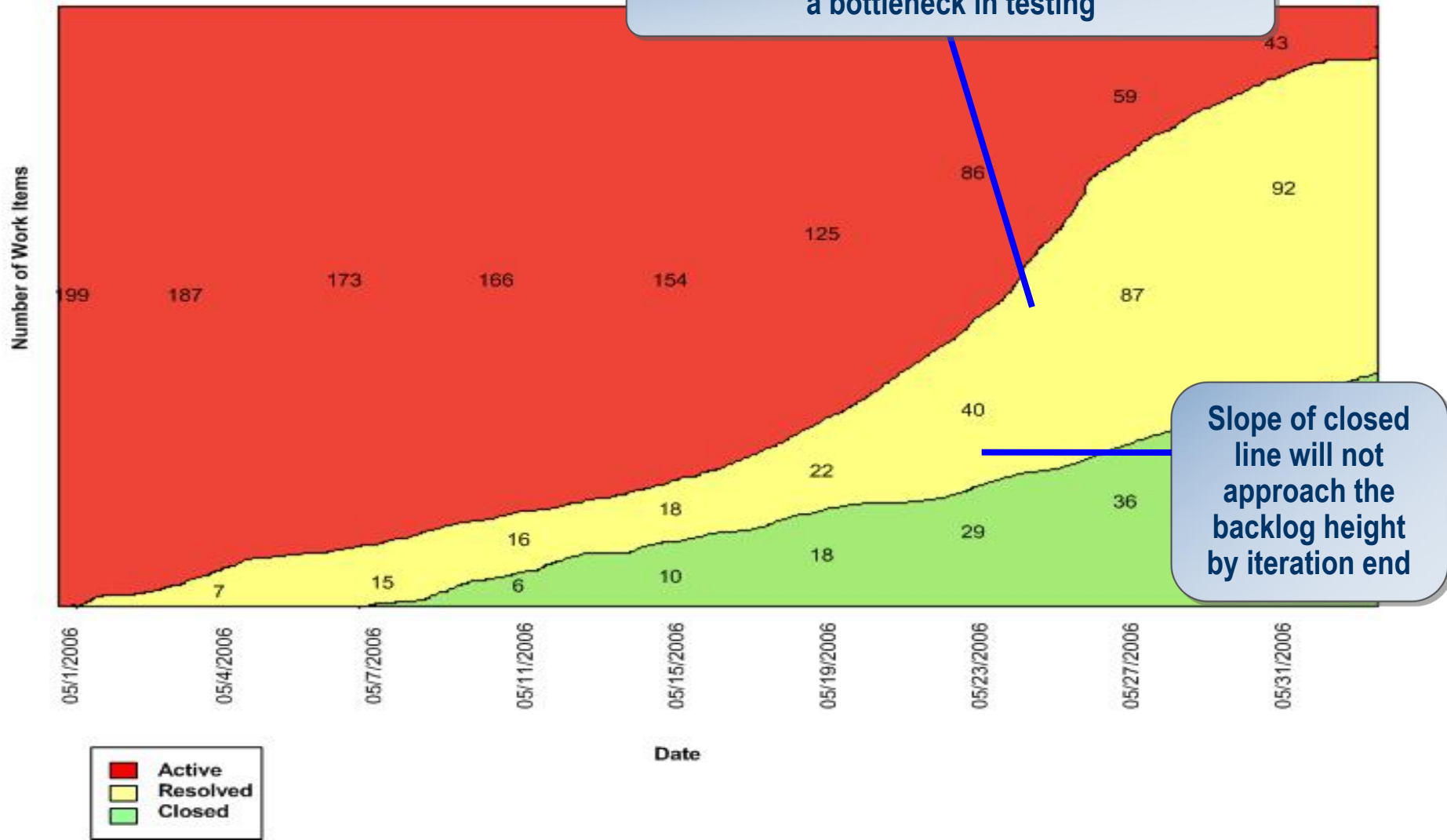


Working software delivered
to the customer at
regular intervals

Customer satisfaction is
measured at regular
intervals

Measuring Flow

Remaining Work
How much work is left and when will it be done?



Work-Down vs. Value-Up

- **How do you handle planning and process change?**
- **What is your primary measurement of progress?**
- **How do you define quality?**
- **How do you view variance?**
- **What are your intermediate work products?**
- **How do you troubleshoot a project?**
- **Who do you trust?**

Supporting Value Up Practices with Visual Studio Team System

- **Why Value-Up Requires Tooling**
- **Benefits of a Common Product Backlog**
- **Benefits of Instrumenting Daily Activities**
- **Assessing Quality**
- **Assessing Quality (Continued)**
- **Benefits of Iterative Development**

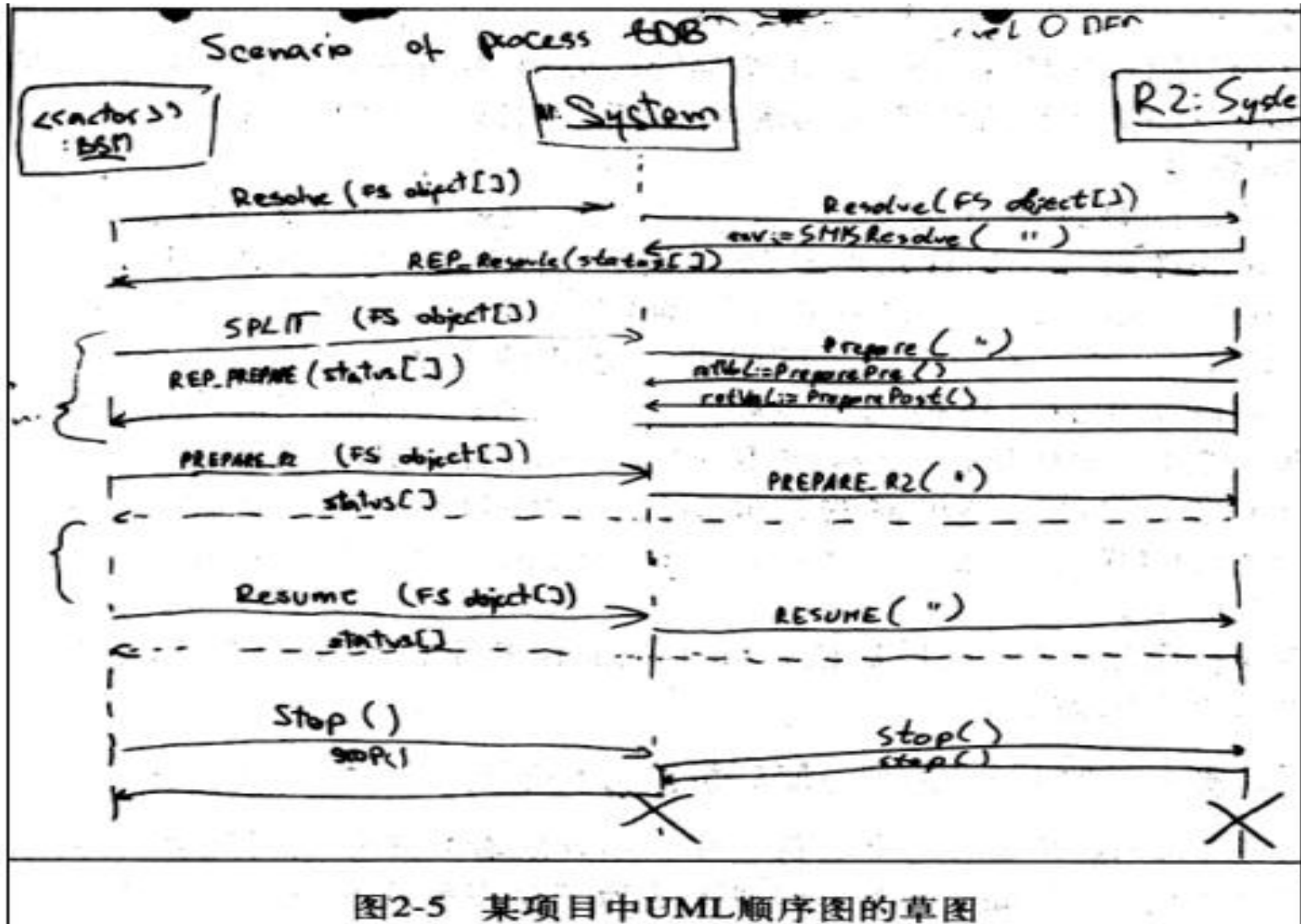
Why Value-Up Requires Tooling

- The manual tracking of project progress is inefficient
 - Manual processes, and disparate tools lead to inaccurate information
 - Much time and effort is wasted
- Tooling is required to
 - Reduce the overhead and costs
 - To provide automated instrumentation of processes
 - To provide change management and auditing (particularly for regulatory compliance)



① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

敏捷建模草图示例



- 课堂练习题：

- 如何拒绝需求分析过程中某些不合理的用户需求？
- 分析阶段类的大致种类有边界类、实体类、控制类。请简述其具体含义（包括画法）。
- 课后练习题：习题12或习题17.（任选，**UML**描述格式）
- 网上实例1：小米校招产品作业解读：设计一款网络日记**APP**。（画像课堂作业）

- 信息系统需求分析考试题目