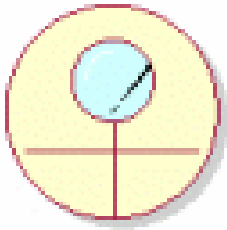


软件工程实践

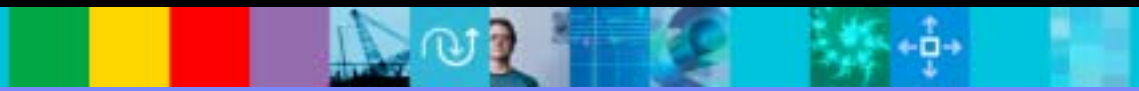
Rational 用户组

uml.org.cn

UML软件工程组织

基于用例的需求管理
RMUC

UML软件工程组织



Evolve by case

瞿广峰

课程大纲

- 需求管理简介
- 用例建模方法
- 问题分析
- 定义系统
- 管理系统规模
- 改进系统定义
- 变更管理
- 产品生命周期中的需求管理

引言1: 需求管理简介

定义：

- 需求（Requirement）
  - 系统必须满足的条件或必须具备的能力
- 需求管理
  - 一系列步骤用以：
    - 寻找、提取、组织、记录和管理需求
    - 建立并维护客户/用户与项目团队间就变更的需求达成的共识

软件需求的指标

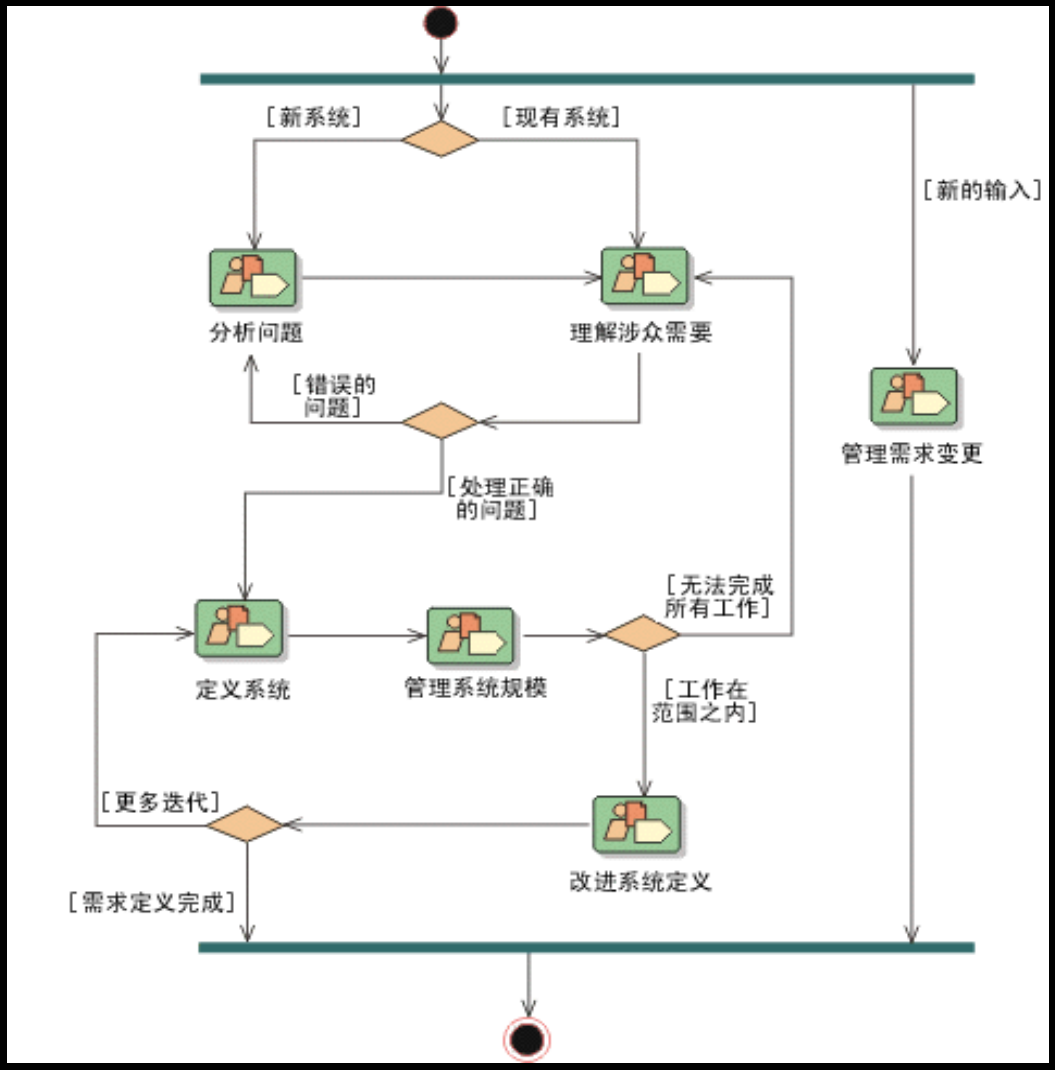
- 可验证
- 按重要性、稳定性排序
- 可修改
- 可追踪
- 可理解
- 正确
- 完备
- 统一
- 无歧义

- 需求中不包括的部分：
  - 设计
  - 验证
  - 项目数据

需求管理的目的

- 与客户和其他涉众在系统的工作内容方面达成并保持一致。
- 使系统开发人员能够更清楚地了解系统需求。
- 定义/限定系统边界。
- 为计划迭代的技术内容提供基础。
- 为估算开发系统所需成本和时间提供基础。
- 定义系统的用户界面，重点是用户的需要和目标。

需求管理工作流程

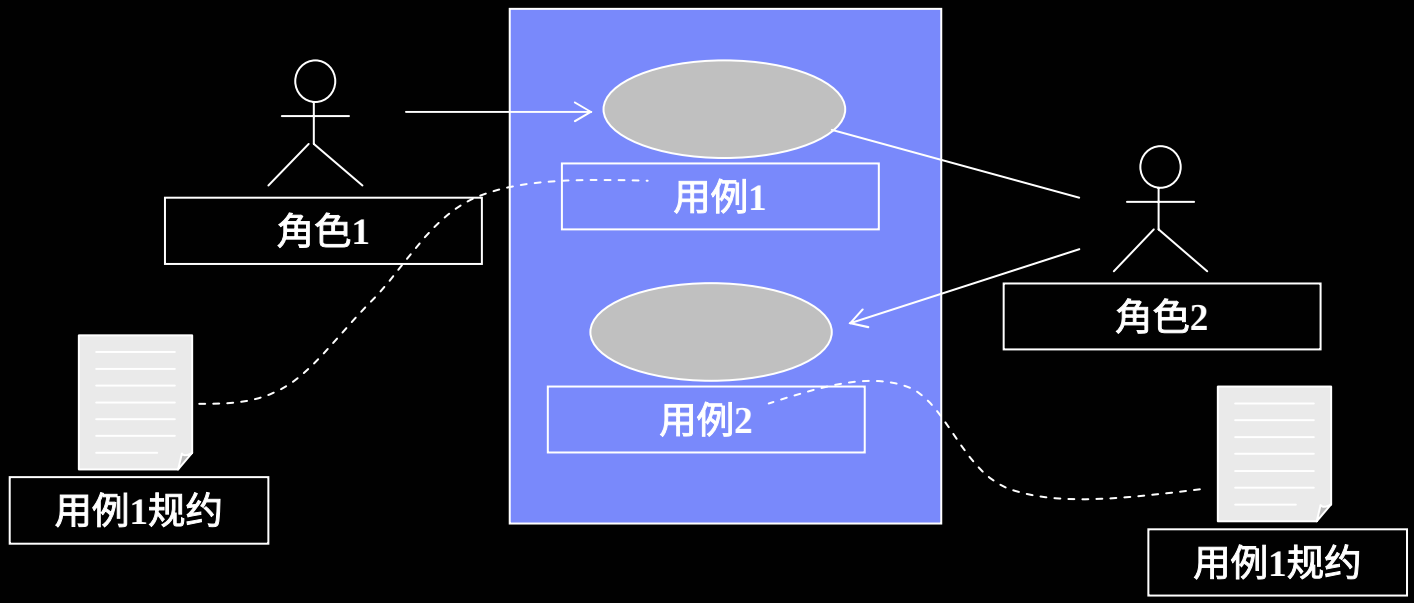

[新系统]
[现有系统]
[新的输入]
分析问题
理解涉众需要
[错误的问题]
管理需求变更
[处理正确的问题]
[无法完成所有工作]
定义系统
管理系统规模
[工作在范围之内]
[更多迭代]
改进系统定义
[需求定义完成]

引言2: 用例建模方法

- 什么是用例建模？
  - 捕捉待开发系统行为的方法
  - 表述系统行为的方法
  - 识别谁或什么与系统交互，和系统应该做什么
  - 验证所有的需求都已经捕捉到
  - 做计划的工具

角色与用例

- 角色：在系统之外与系统交互的某人或某事物
- 用例：系统执行的外部可见的产生对角色有价值结果的动作序列
- 用例的命名：能够说明目标的动名词组



描述用例的要点

- 外部可见：用户行为、系统响应
- 有价值
- 详细程度：到能让所有人对需求达成共识为止
- 用户界面草图
- 使用无歧义的词汇
- 语言精确

找角色的常用问题

- 谁/什么使用系统？
- 谁/什么从系统获得信息？
- 谁/什么向系统提供信息？
- 谁/什么支持、维护系统？
- 哪些其它系统使用此系统？
- 公司的哪个部门使用系统？
- …

找用例的常用问题

- 每个角色的目标是什么？
  - 为什么该角色想要使用此系统？
  - 该角色是否要创建、保存、更改、移动或读取系统的数据？如果是，为什么？
  - 该角色是否要通知系统外部事件或变化？
  - 该角色是否需要知道系统内部的特定事件？
- 系统是否向业务提供了所有的正确的行为？

用例检查点

- 用例模型的简介部分简明清晰地概述此系统的目的和功能。
- 所有的用例已确定；这些用例共同说明所有的必要行为。
- 所有的功能性需求都至少映射到一个用例。
- 该用例模型不包含多余的行为；所有的用例都可回溯到某个功能性需求来证明其合理性。

角色检查点

- 是否您已找到所有的角色？也就是说，是否您已经对系统环境中的所有角色都进行了说明和建模？
- 每个角色是否至少涉及到一个用例？
- 您能否列出至少两名可以作为特定角色的人员？
- 是否有角色担任与系统相关的相似角色？如果有，您应该将他们合并到一个角色中。

练习：找角色和用例

- 找出与课程注册系统交互的所有角色
- 找出系统的用例
- 画出用例图
  - 参照用例、角色的检查点

第一章：分析问题

什么是“问题”？

开发者的理解与客户的期望之间的差别。

分析问题的重要性：

- 避免出现“是的...但是...”
- 减少额外的工作
- 理解需求
- 找到正确的解决方法

分析问题——活动

- 就问题定义取得一致
- 理解根本原因
- 定义业务需求（可选）
- 确定正确解决方案
- 确定涉众
- 定义解决系统的边界
- 确定系统或项目的约束

取得一致

- 问题是什么？
  - 理解用户的观点
  - 记录用户的观点
  - 就其达成共识
- 真正的问题是什么？
  - 找到根本原因
  - 强调根本原因
- 方法：
  - 问题背后的问题：鱼刺图
  - 关注关键问题： 20－80原则，Pareto图

确定项目涉众

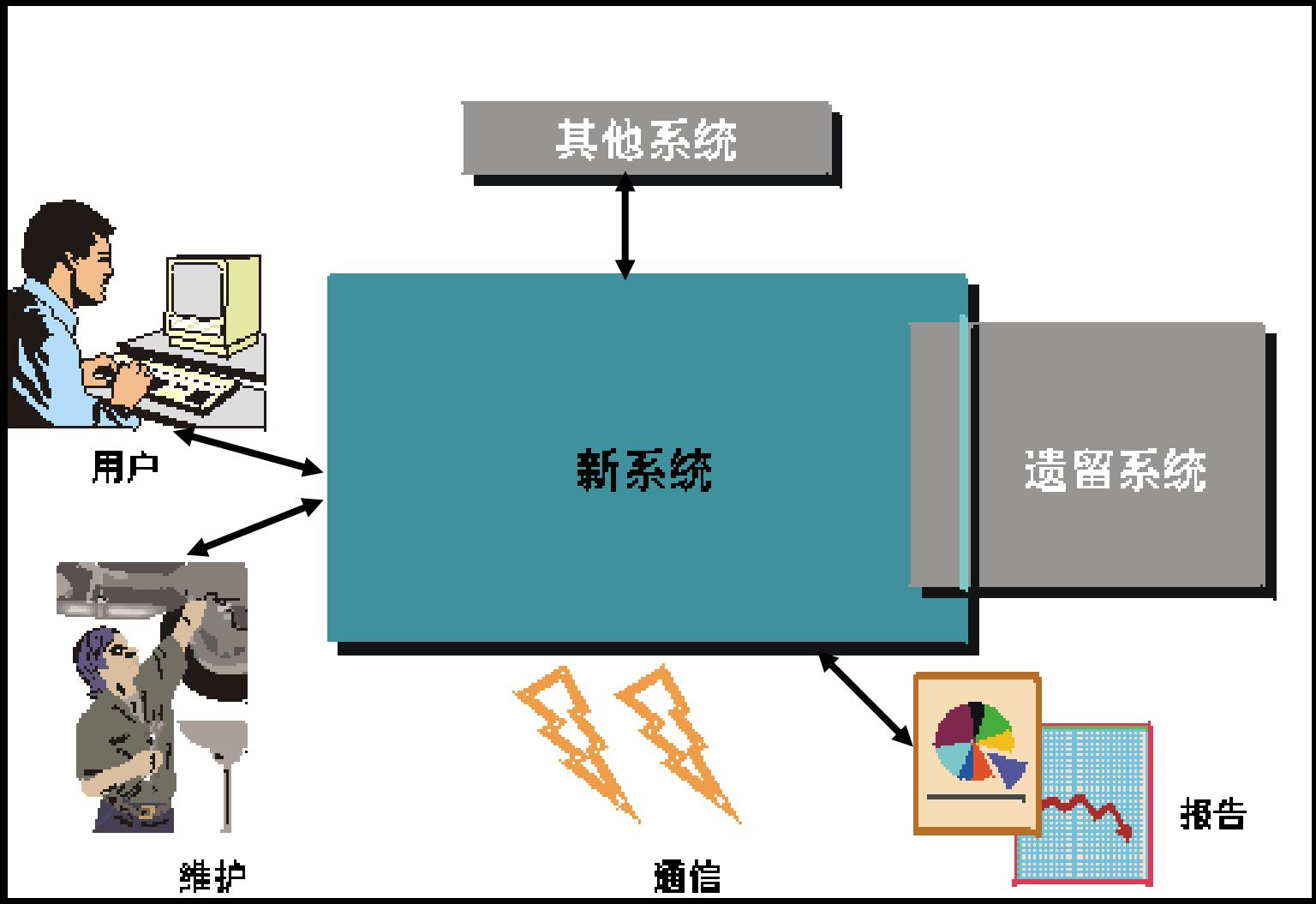
- 涉众是所有会受到项目结果重大影响的人。
- 示例：
  - 客户（或客户代表）
  - 用户（或用户代表）
  - 投资者
  - 股东
  - 生产经理
  - 买方
  - 设计员
  - 测试员
  - 文档编写员 等等

确定对系统强加的约束

约束：对提供解决方法的自由度的限制。

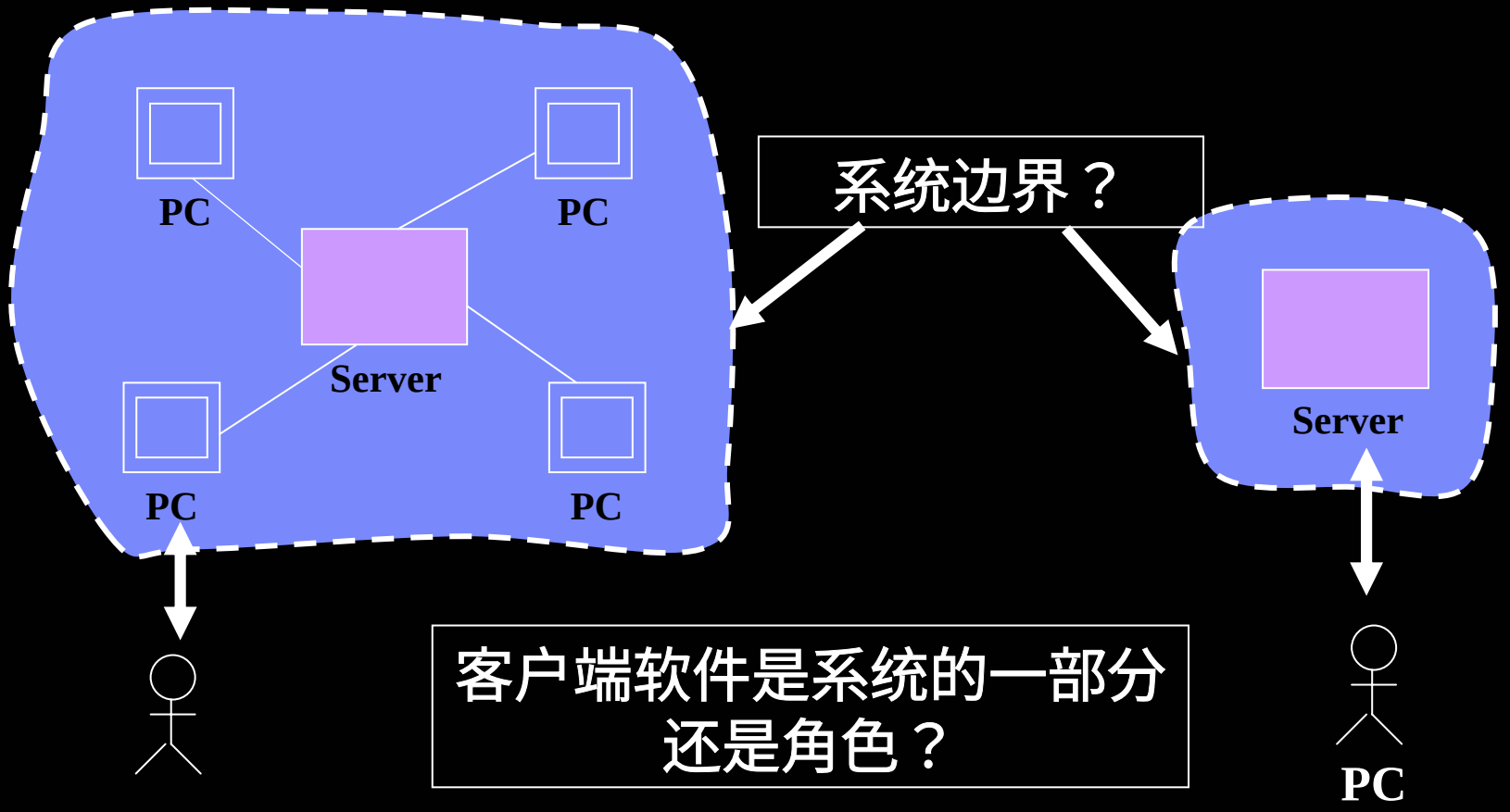
- 经费
- 技术
- 可行性
- 系统
- 环境
- 政策
- …

定义解决系统边界



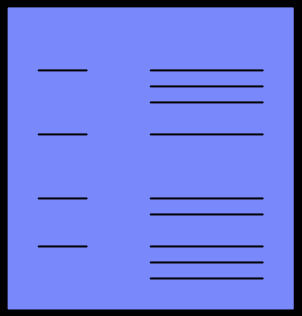
确定系统边界

- 用角色帮助确定系统边界



获取常用词汇

- 定义项目中使用的词
- 用于防止误解



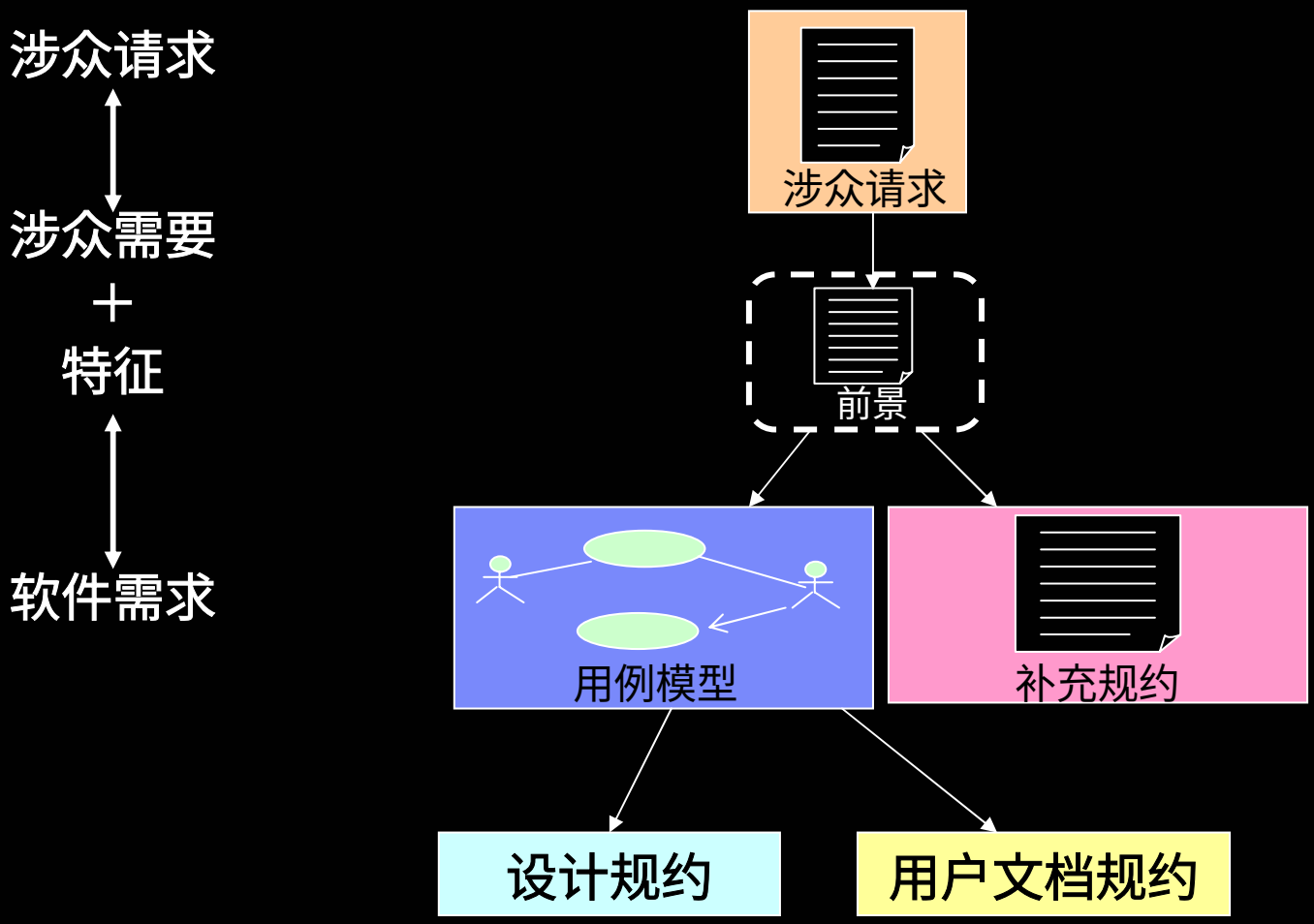
词汇表

获取常用词汇

- 尽可能早开始
- 在整个项目中不断更新

在某些项目中，词汇表将是用于记录项目的业务领域的唯一工件。如果既不执行业务建模，也不执行领域建模，就将出现这种情况。

前景文档



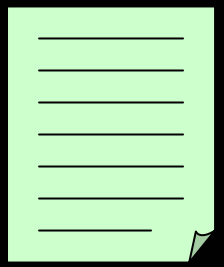
前景文档

- 管理、市场、项目组间交流
- 给用户的初始反馈
- 开始对产品的通用理解
- 建立高层次特征的范围和优先级

所有部门工作的基础

前景文档

- 描述与项目有关的最根本的“什么”和“为什么”的系统级文档，将来确认所有决策时都应以此为标准。
- 核心内容：
  - 用户需要
  - 目的、目标
  - 市场定位
  - 使用环境和平台
  - 产品特征
- 参见RUP中“前景”模板
  - 关注：问题说明、产品说明



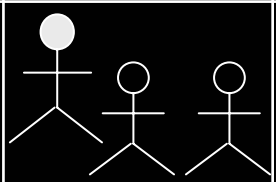
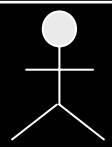
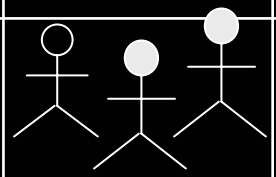
前景

需求管理计划

- 需求管理计划的主要内容：
  - 需要收集的需求类型
  - 需要跟踪的需求类型
  - 需要跟踪的属性类型
  - 需要产生的文档类型
  - 管理指南

需求属性

用属性建立项目元素间的关系

	状态	风险	优先级	工作量	预算
特征需求 10	证实	高	高		￥￥￥
特征需求 13	预想	中	低		￥￥
特征需求 40	证实	高	强制		￥

回顾：问题分析

1. 问题分析的主要活动有哪些？
2. 如何就问题取得一致？
3. 什么因素决定系统边界？
4. 角色如何帮助决定系统边界？
5. 为什么要建立词汇表？
6. 需求管理计划中有什么内容？

第二章：理解涉众需求

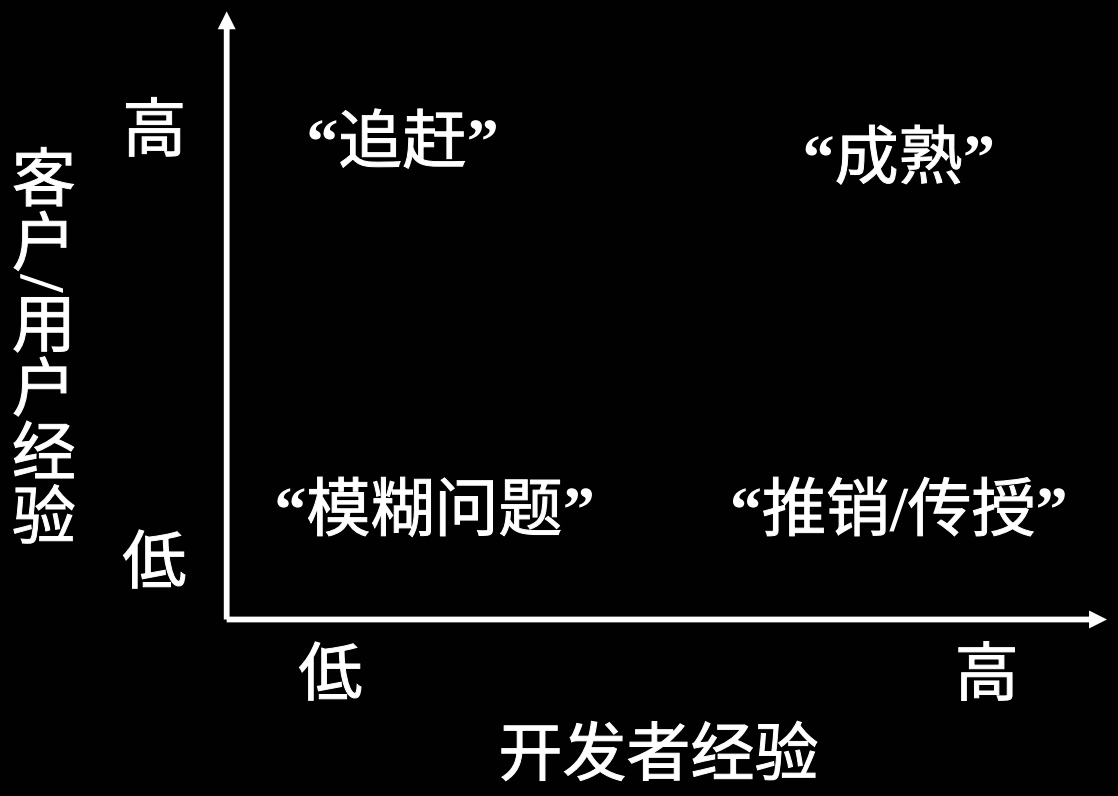
我们会碰到什么样的问题？

- 涉众
  - 不知道自己需要什么。
  - 知道自己需要什么，但无法明确地表达出来。
  - 认为自己知道自己的要求，但直到交付的时候才知道不是。
- 系统分析员
  - 认为自己比用户更明白用户的问题。
- 每个人
  - 认为其他人都是受小团体的利益推动的。

收集用户需求的方法

- 需求研讨会
- 头脑风暴与意见裁减
- 用例研讨会
- 故事板
- 访谈
- 问卷调查
- 角色扮演
- 原型
- 复审客户需求规约

收集需求：应使用哪种方法？



你会为各个域选择哪种工具？

需求研讨会

- 头脑风暴
- 访谈
- 问卷调查
- 角色扮演
- 业务建模
- 故事板
- 原型
- 复审需求

BACK

第三章：定义系统

- 产品定位说明

对于	*** *[目标客户]*
他们	希望能够 *[陈述需要或机会]*
该**（本产品名）	属于*[产品类别]*
具有	N大优点 *[陈述重要优点，即促使人们购买的原因]*
不同于	*[主要的竞争产品]*
我们的产品	可以 *[陈述主要的区别]*

软件需求的质量

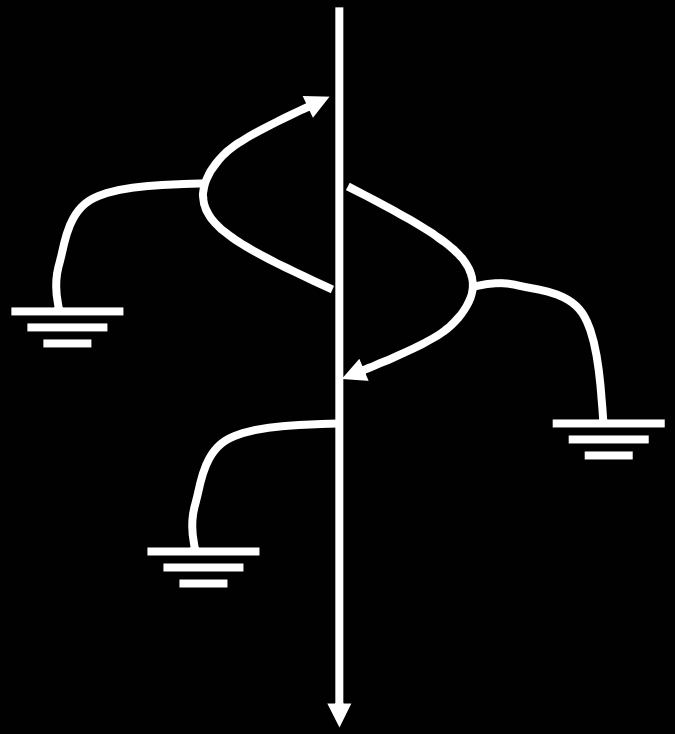
- 正确
- 完备
- 一致
- 无歧义
- 按照重要性和稳定性排序
- 可证实
- 可修改
- 可追踪
- 可理解

产品特征

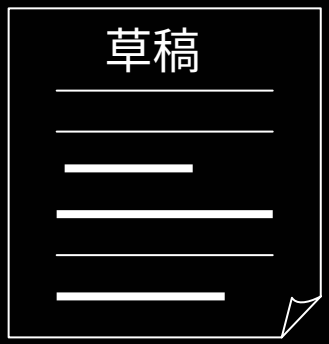
- 外部可见的、系统直接满足涉众需要的服务
- 示例：
  - “缺陷追踪系统”应提供趋势信息以帮助项目经理评估项目状态
  - ATM必须允许客户在账户建转账
  - 图形用户界面应提供文字敏感帮助

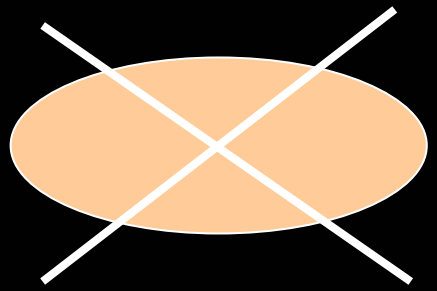
事件流（基本和可选）

- 一个基本流
  - Happy day场景
  - 成功场景从开始到结束
- 许多可选流
  - 常见的变化（新手上路）
  - 特例
  - 意外（错误）流



为什么写事件概述？





不是用例？

用例规模

太大？

太小？

太多？

帮助你找到所有
的可选流

用例检查点

- 各用例之间是否相互独立？
- 不同的用例间是否有非常相似的行为或事件流？
- 是否事件流的一部分已经在别的用例中表述过了？
- 如果模型很大或模型的各部分间的责任分散，应创建用例包
  - 包使模型易于理解

回顾：定义系统

1. 产品定位声明中有哪些内容？
2. 用例模型中描述角色和用例的哪些性质？
3. 如何整理用例模型工件？

第四章：管理系统规模

- 定义系统规模：
  资源+时间 → 我们能作多少工作？
- 建立需求基线：
  - 特征1：系统必须…
  - 特征2：系统必须…
  - 特征3：系统必须…
  - 特征n：系统必须…

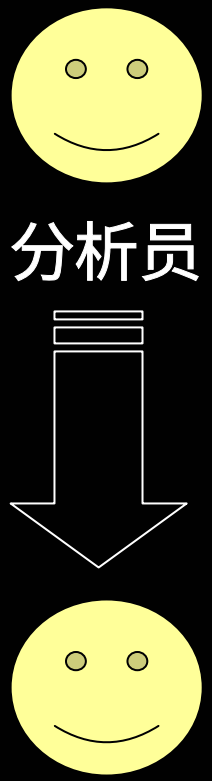
我们如何获知需要？

我们如何确定优先级？

我们在哪设立基线？

原则：少承诺，多交付

用例的优先级



- 看看项目中的用例
- 你在首轮迭代中选择那些用例？为什么？
- 你还要考虑什么？
- 定义系统规模

1. 考虑与在基线范围内的特征相关的用例
2. 为架构迭代选择用例
   - 代表重要、核心功能
   - 具有基础的架构覆盖
   - 处于架构的一个特定、微妙的位置
   - 被标志为高风险
3. 为将来的迭代对用例/场景定优先级

如何管理规模

- 控制需求
  - 输入
  - 输出
  - 处理

控制需求的输入

正式来源

- 需求规约
- 客户请求
- 市场部
- Bug报告和改进请求
- 对竞争对手的分析

非正式来源（泄漏）

- 谈话中获得的改进请求
- 客户直接对编程的要求
- 程序员“仔细考虑，对客户有利的”增改
- 硬件故障或特性限制
- 对竞争者的“膝跳反射”
- 程序员的“后门”

控制需求的输出

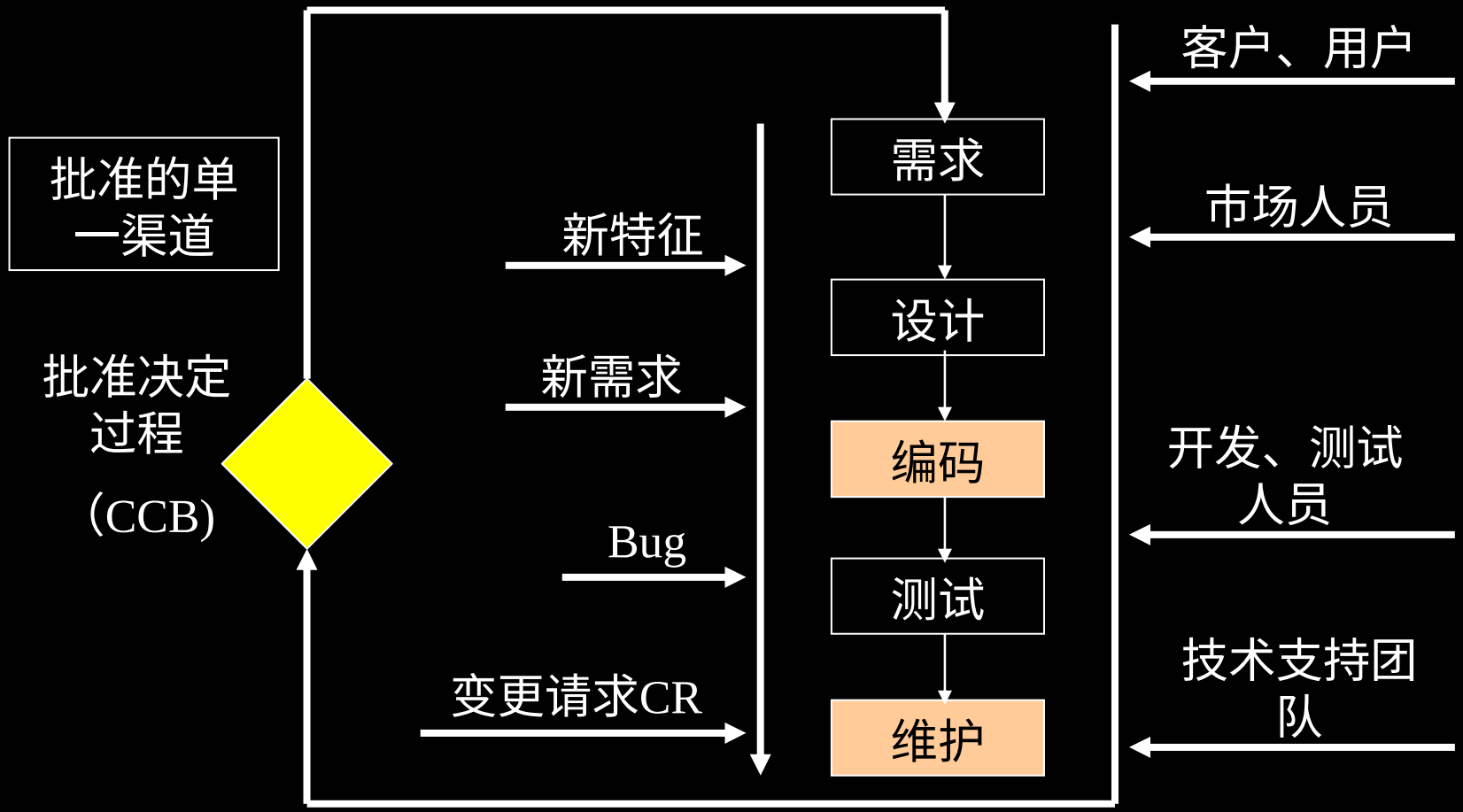
- 记录需求
  - 写下来
  - 将需求放在一个中央仓库里
  - 使所有涉众都能访问到需求
  - 保持在一个合理的稳定状态
  - 控制变化（防止“ 规模扩散 ”）
- 使用需求属性和追踪性
  - 从正确的人那里得到有用的信息
  - 为需求在仓库中建立属性/追踪

控制需求的处理

- 过程支持
  - 迭代开发
  - 复审需求
  - 变更管理
- 资源支持
  - 人
    - 扩员
    - 培训
  - 工具
    - 仓库
    - 网络
    - E-mail
    - 文档处理器

所有变更的单一渠道

需求变更来自产品生命周期中每一次迭代中的多种来源



产品推介人（产品经理）

- 帮助管理项目规模
- 为产品说话
- 与管理层、用户和开发团队交流
- 阻止特征扩散
- 在用户需要和及时交付之间做平衡
- 代表客户与开发团队间的官方渠道

管理意外

- 为什么会有意外？
  - 人们的预期不同
  - 分析员也是人
  - 有事发生
- 如何管理意外：
  - 理解客户期待
  - 使期待限制在正确的范围
  - 说明限制的原因
  - 少承诺，多交付

保证可能性的存在

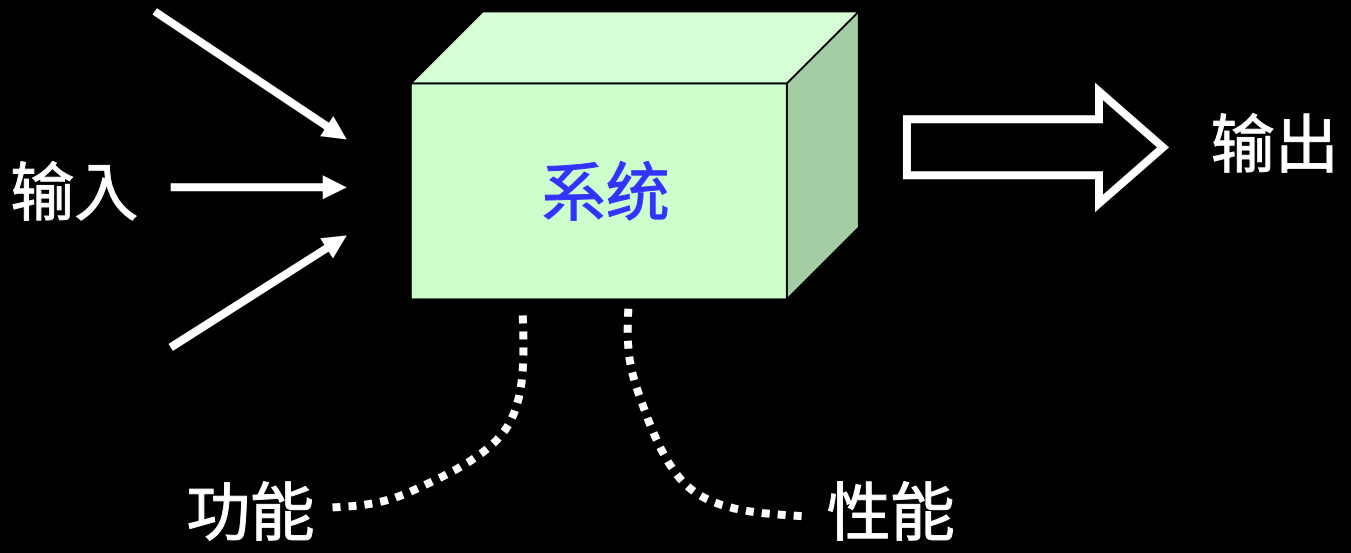
- 改进沟通技巧
  - 对事不对人，关注利益而不是位置
  - 为无法达成共识准备一个最佳备选方案（BATNA）
  - 找到双赢选择
  - 运用外交手腕

回顾：管理系统规模

1. 属性在管理规模中有什么用处？
2. 管理规模的3个重要步骤是什么？
3. 产品经理的作用是什么？
4. 为什么管理意外重要？如何管理？

第五章：改进系统定义

软件需求规定什么？



需求是系统必须具备的能力或必须满足的条件。

软件需求是系统的外部黑盒定义，定义外部可见的“什么”；

“如何”通常做为设计约束定义。

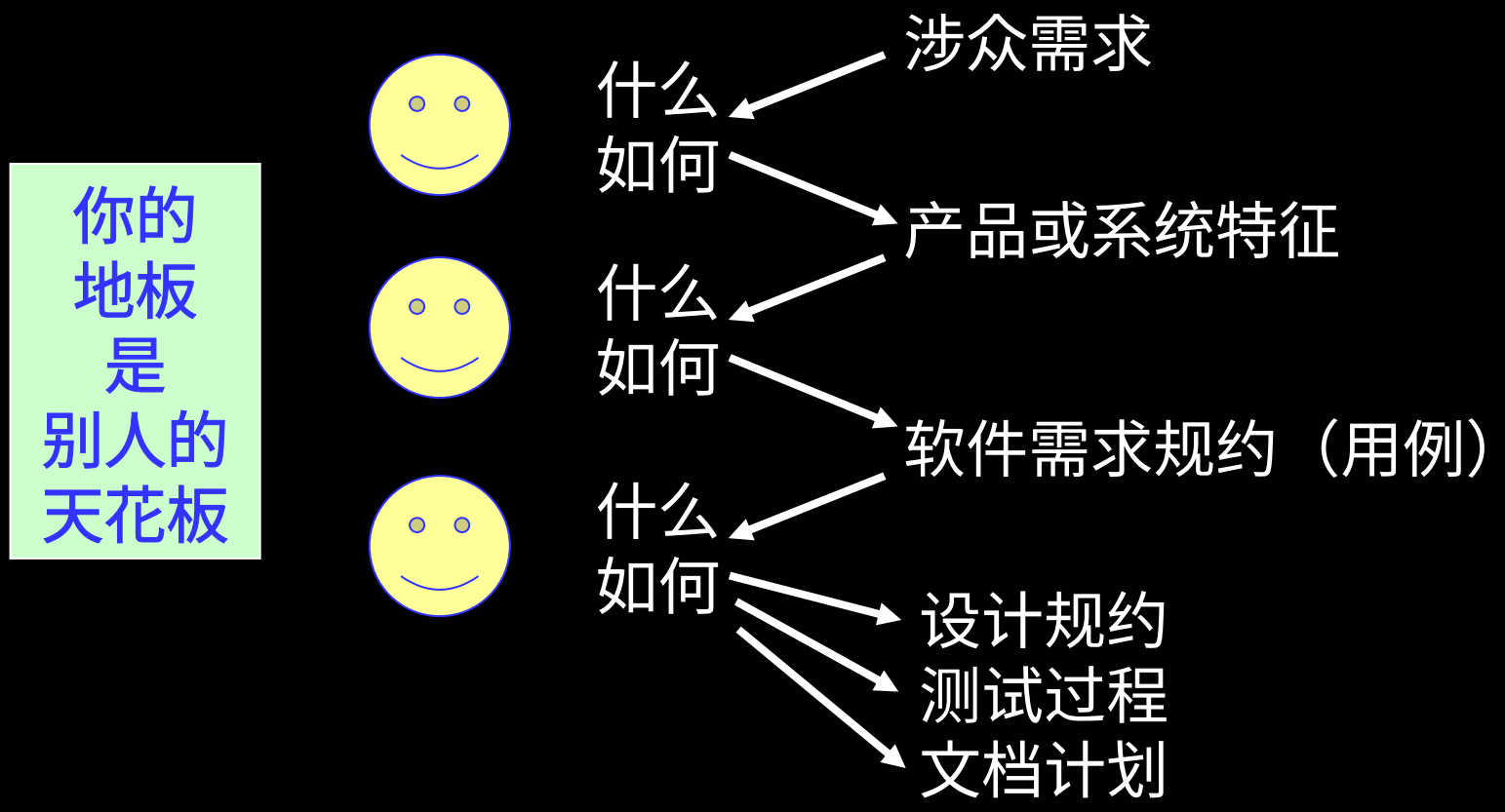
设计约束

- 需求允许多种设计选择
  - 设计是选择的一种
- 没的选择的需求就是设计约束
  - 将它与其它需求分开
  - 放在软件需求的特殊段中（用例规约、补充规约）
  - 标明来源
  - 记录理由
- 例子
  - 被要求使用的算法
  - 被要求使用的数据库

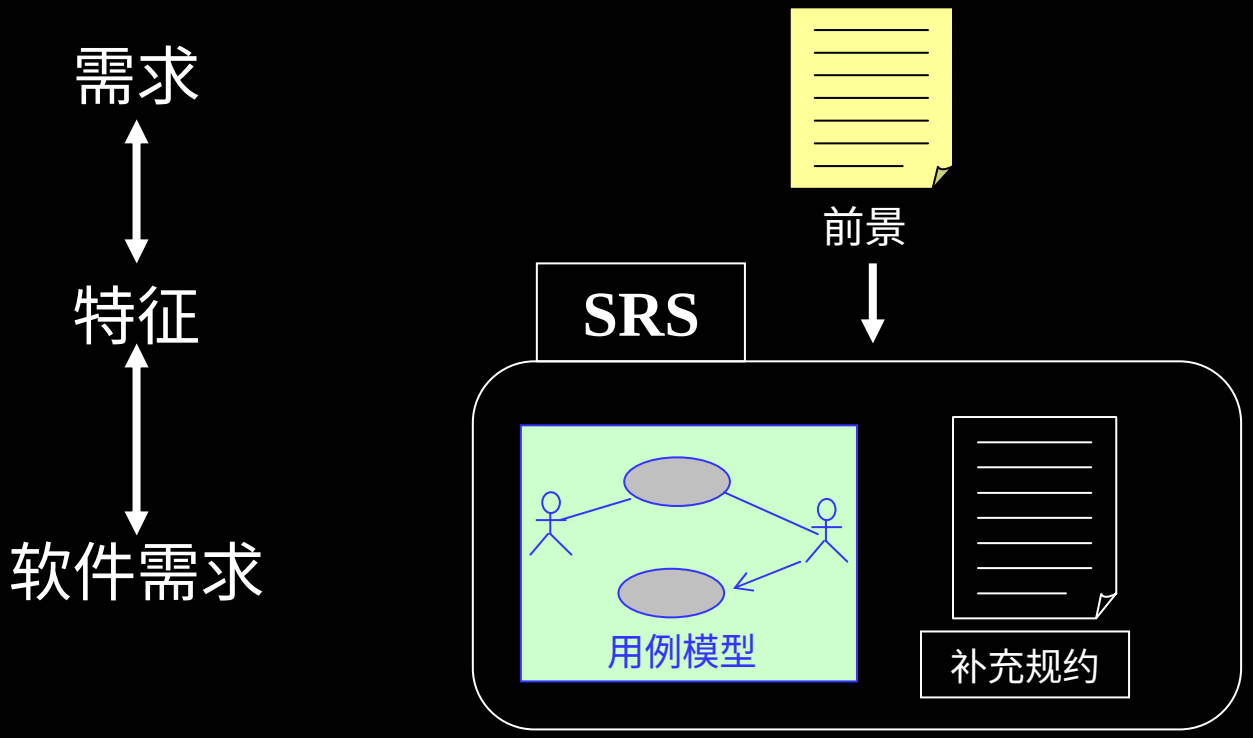
What对How的难题

问题：我该如何写需求告诉听众做什么，而不是如何做？

答案：这取决于你的视角。



软件需求规约（SRS）



SRS定义系统的所有外部行为和特征。

作用：
- 各团体的交流基础
- 各团体的合同协议
- 开发的基础：设计、实现、测试

SRS包的内容

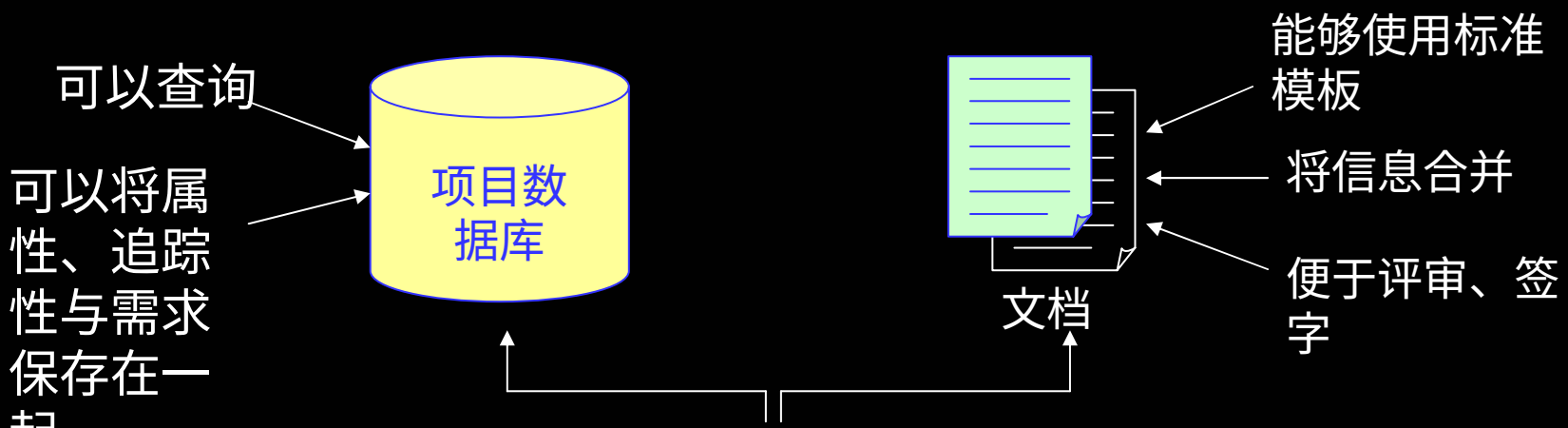
1. 简介
   - 1.1 目的
   - 1.2 范围
   - 1.3 定义、首字母缩写词和缩略语
   - 1.4 参考资料
   - 1.5 概述
2. 整体说明
   - 2.1 用例模型调查
   - 2.2 假设与依赖关系
3. 具体需求
   - 3.1 用例报告
   - 3.2 补充规约
4. 支持信息

附录

目录

SRS包中的内容保存在哪？

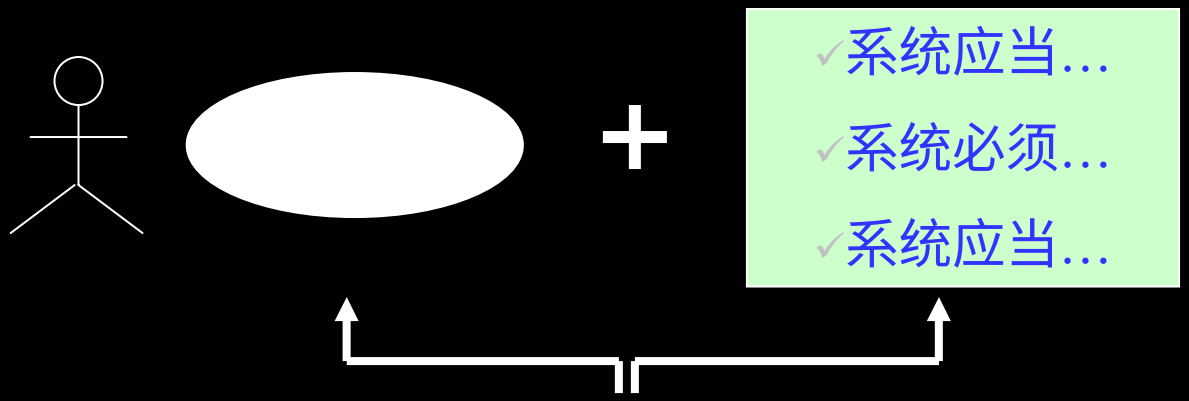
用例图、文本文件、用户界面原型分别用不同工具创建、保存在不同位置、有不同的访问路径，该如何保存和访问？



- 使用哪种信息保存方式取决于：
  - 可用的工具
  - 开发环境和习惯
  - 客户环境和习惯
  - 项目的规模

如何表述功能性需求规约？

- 用例图和陈述句
  - 对于理解任何复杂的系统都是必要的
- 可以的话，请选用例图



- 选择使用哪种表述方式时，应考虑：
  - 应用与项目的环境
  - 开发团队的技术与舒适程度
  - 客户环境和习惯
  - 项目的风险，和如何能最好地表述风险

不在用例中的需求怎么办？

- 用陈述句描述软件需求
  - 给每一条需求编号并加标题
  - 将相关的需求分组，以提供上下文
- 使用团队宜于理解的语言
  - 简单句
    - 名词＋动词
  - 考虑使用关键词，例如“ 应当 ”
  - 简短
    - 用1到3句说明一条需求
  - 使用词汇表中的词
- 与用例相关的放在用例规约的“ 特殊需求 ”段
- 与整个系统相关的放在“ 补充规约 ”中

非功能性需求

- FURPS中的URPS
  - 可用性
  - 可靠性
  - 性能
  - 可支持性
- 法律、规章
  - ISO
  - FCC
  - DOD
  - FDA
- 设计约束
  - 操作系统
  - 环境
  - 兼容性
  - 应用标准

还需考虑什么？

通讯协议、界面

- 如果角色是另一个系统或外部硬件，就需要一个通讯协议
  - 用例规约中要说明是否使用了已有的协议（TCP/IP…）
  - 如果是新协议，在对象建模阶段要完整描述（在用例规约中引用）
- 在用例描述中加入预期屏幕显示的草图
- 注意不要在用例文档中放太多的细节

细化用例

- 基本事件流：每一步的双向描述
- 可选事件流：每一步的开始、条件、动作、恢复
- 前置条件：不是用例的触发条件
- 事后条件：应对备选流也为真，可包含变量、条件（“如果…”）。

回顾：改进系统定义

1. 软件需求规约（SRS）的内容？
2. 如何描述功能性需求？
3. 何时使用用例描述需求？
4. 何时用陈述句？
5. 有几种非功能性需求？
6. 什么是设计约束？记录在哪？

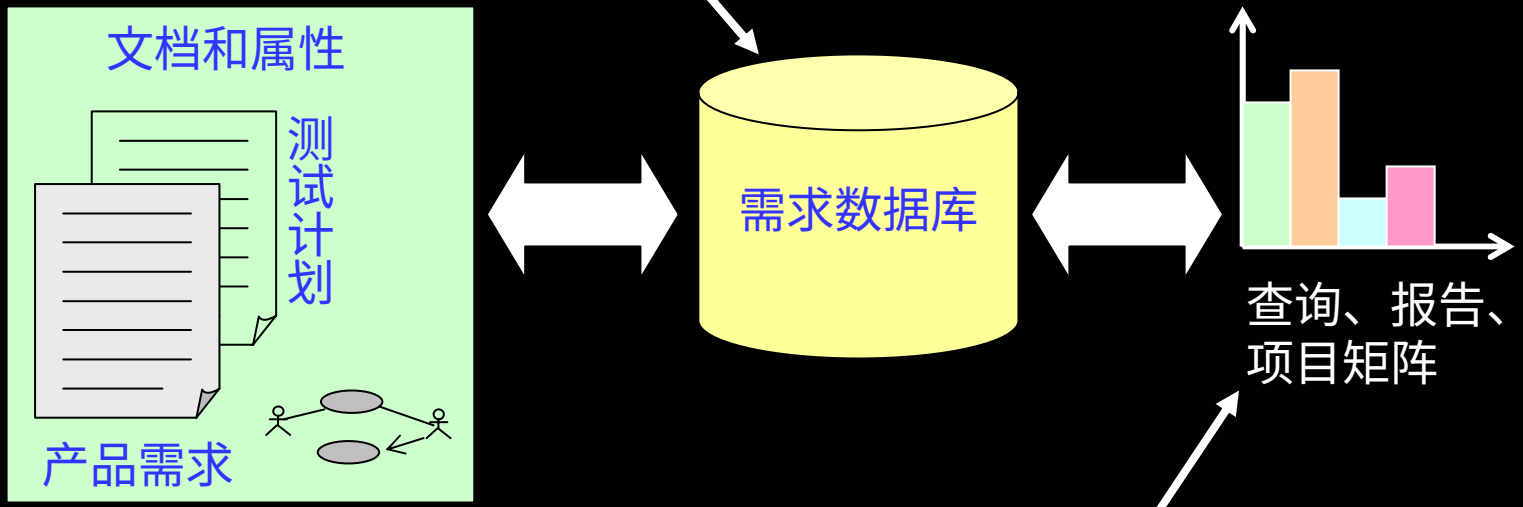
第六章：管理需求变更

需求为什么会改变？

- 我们没有在正确的时间问正确的人正确的问题
- 已经解决的问题又改变了
- 用户改变主意或理解
- 外部环境改变
- 我们没能创建或采用一个流程管理变更
- 我们对问题的理解加深了

用数据库帮助管理变更

数据库保存项目需求、需求属性、联系、版本
数据库提供自动追踪－自动维护属性间的联系
数据库易于查询、生成报表



项目矩阵用于帮助项目经理、系统分析员评估项目状态、进度、预期性能、优先级、工作量、新增或变化或不稳定的需求

追踪性

业务需求

驱动

客户需求，

后者驱动

用户需求，

后者要求

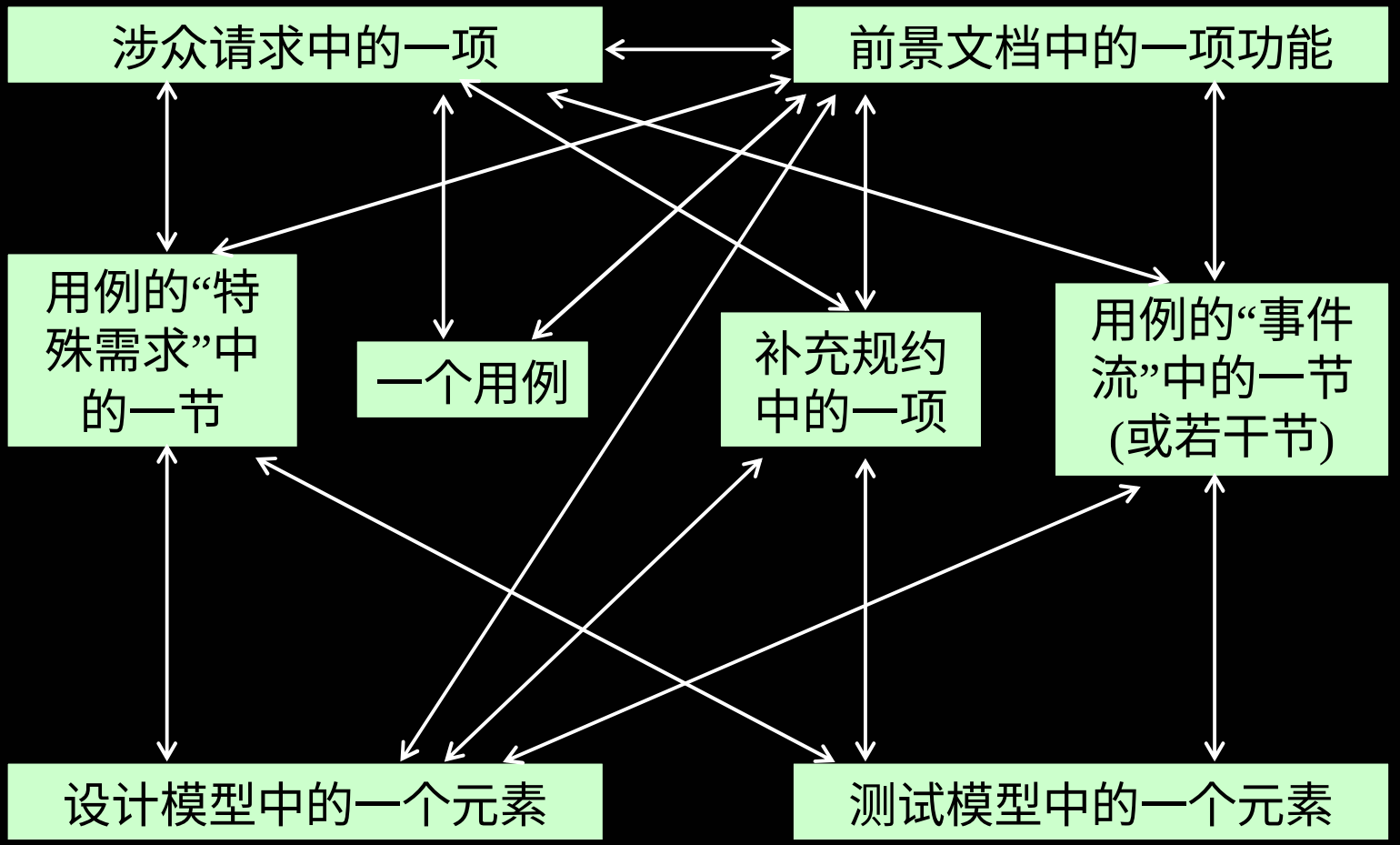
产品特征，

后者需要开发者

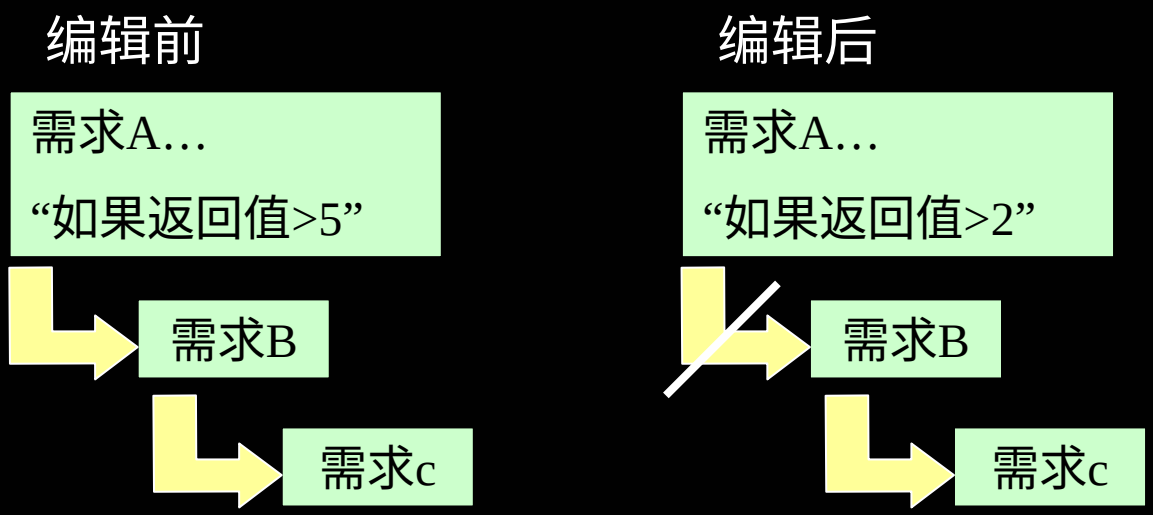
实现、测试、记录

- 保证质量
  - 证实所有需求都得到实现
  - 证实实现只做了应该做的。
- 分析变更的影响
  - 查找相关需求
  - 检查相关需求

建立你自己的追踪策略


涉众请求中的一项
前景文档中的一项功能
用例的“特殊需求”中的一节
一个用例
补充规约中的一项
用例的“事件流”中的一节(或若干节)
设计模型中的一个元素
测试模型中的一个元素

用追踪性分析影响



- 变更的需求的所有链接都被标记为“可疑”
- 可疑链接必须由用户解除

管理需求变更的一个核心概念是“可疑”的可追踪性链接

参见RUP中“概念：可追踪性”

追踪性对项目经理的帮助

项目经理可以通过追踪性查询：

- “没有链接到产品特征的用户需求有哪些？”
- “正在进行迭代#2的那些用例的测试的状态如何？”
- “请将测试失败的结果按照严重程度排序。”
- “这次发布的版本计划了哪些特征？是要满足哪些用户？状态如何了？”
- …

需求的配置管理

- 防止未授权的变更
- 防止修订需求文档
- 得到以前版本文档
- 允许一种可管理基线的发布策略
- 防止同时更新文档

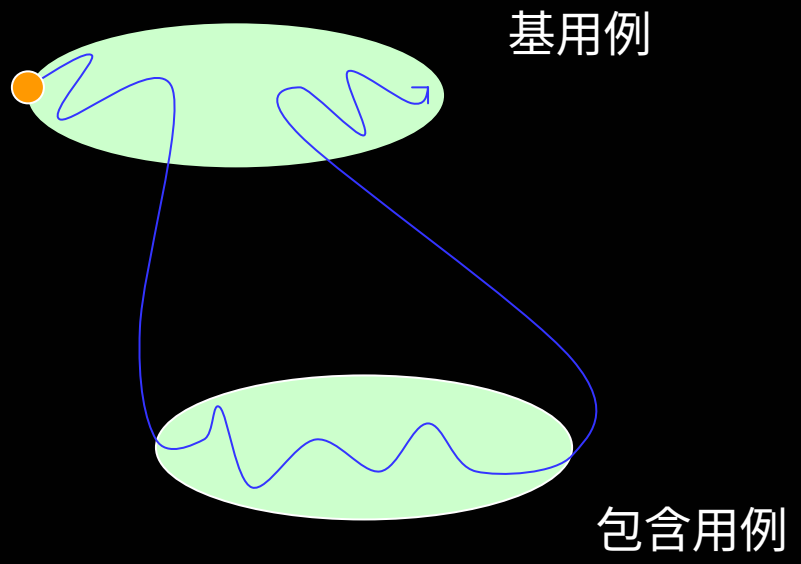
管理变更基线：

何时冻结规约？

用例的包含关系

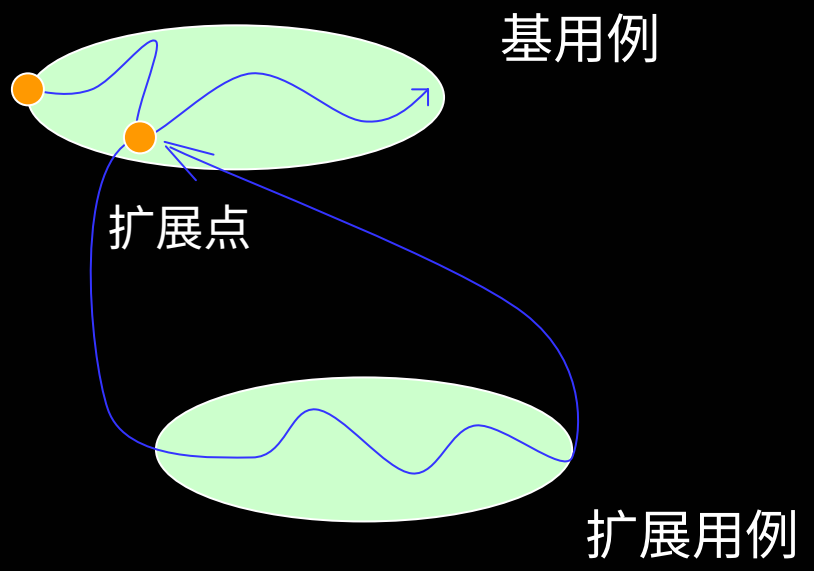
包含关系

- 找用例间相同的行为
  - 避免重复描述
  - 确保行为一致
- 从基用例中分解、封装出一部分行为
  - 简化事件流
  - 分解出不是主要目的的行为



用例的扩展关系

- 分解出可选或特例行为
  - 只在特定条件下执行
  - 分解简化基用例
  - 例子：触发告警
- 添加扩展行为
  - 独立开发的功能，可能在后续版本中
  - 例子：查看新闻

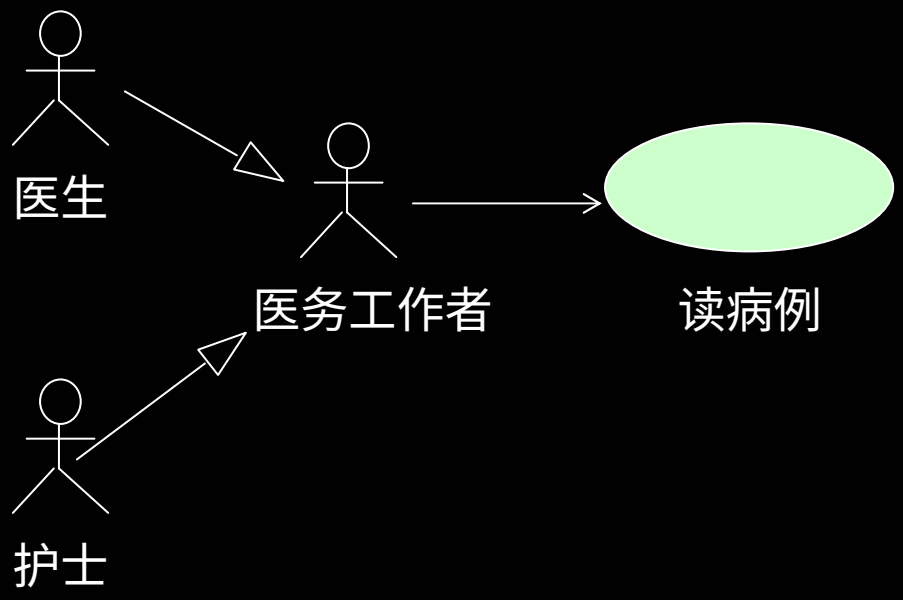


用例检查点

- 被包含用例：是否对包含它的用例做了假设？应避免这种假设，以使被包含用例不受基用例的改动的影响。
- 是否一个用例的事件流应该插到别的用例到事件流中？如果是，建立扩展关系。

角色的泛化关系

- 简化多个角色与一个用例的关系
- 说明子角色可以执行父角色的所有行为

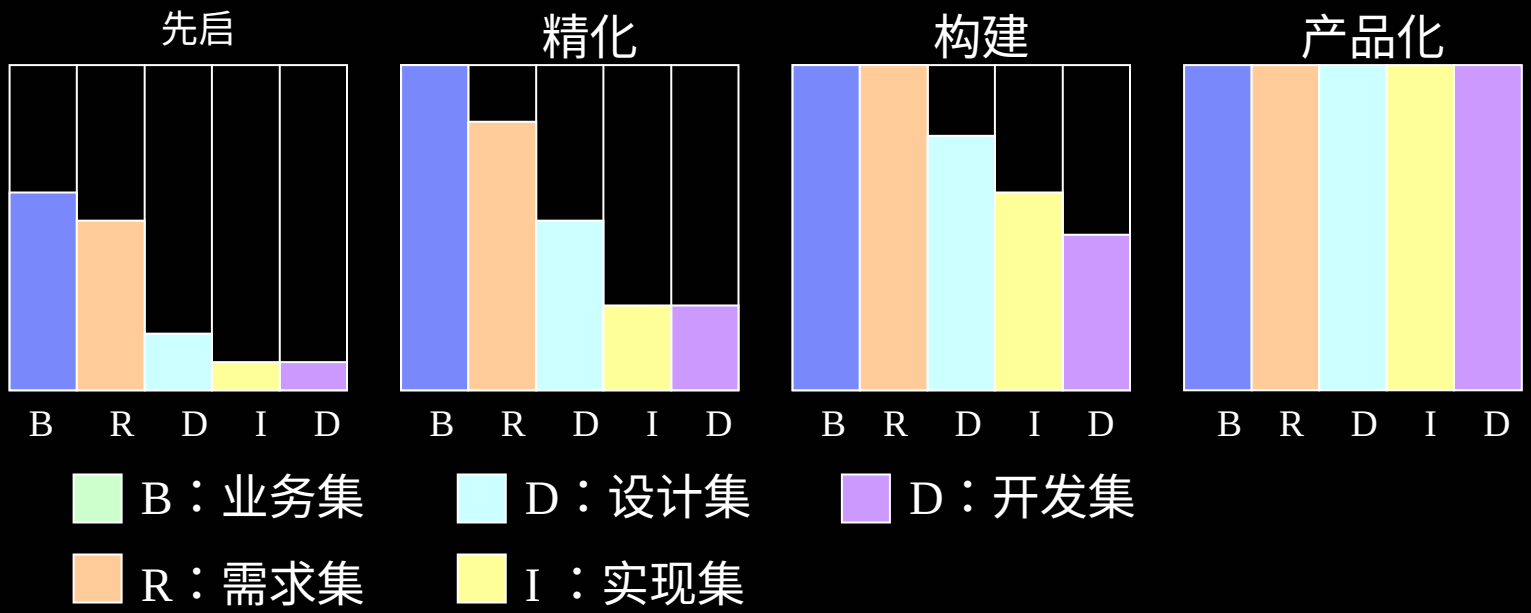


回顾：管理需求变更

1. 需求为什么会改变？
2. 变更控制委员会（CCB）如何防止规模扩散？
3. 需求仓库带来哪些好处？
4. 需求追踪如何帮助管理变更？
5. 将需求纳入配置管理有什么好处?

后记：产品生命周期中的需求

- 4阶段中的典型需求结果



4阶段中的典型需求结果

- 先启阶段：搜集信息，创建业务用例
  - 用例模型概要的草稿
  - 初始的词汇表
  - 一些用例事件流（找需求）
  - 用户界面草图
  - 原型（可选）
- 构建阶段：创建完整系统
  - 与用例事件流有关的需求变更
  - 更新用例事件流
  - 分析、设计、实现、测试
- 精化阶段：改进需求，创建架构
  - 更新词汇表
  - 搜集更多的软件需求
    - 用例、补充规约
  - 改进初期迭代中的用例
  - 定义架构的用例图
- 产品化阶段：
  - 需求不应变更
  - 如加入新特征，则通过与构建阶段同样的迭代过程

迭代的评估

- 应用评估标准
  - 功能性
  - 性能
  - 容量
  - 质量
- 考虑外部变更
  - 例如：用户需求变更、竞争对手的计划
- 决定需要返工的工作
- 将需要返工的工作纳入迭代

需求评估方法

	目的	参与者	过程
走查	▪早期查错 ▪是否违反风格、技术、标准 ▪提示	几个项目成员	▪结果概览 ▪开发者讲解，其他成员评价 ▪记录错误
检查	▪部门间分享观点 ▪早期查错 ▪支持、改进、撤销决定	协调员、记录员、作者、复审员	▪会前组织 ▪围绕主题（协调员） ▪记录观点（记录员） ▪找、讨论错误（复审员）
复审	▪确保结果完备、稳定 ▪决定是否继续项目	管理层、项目领导、过程所有人、分析员	▪检查文档状态（评估结果） ▪回顾项目输出 ▪授权下一阶段开始

THE END.