软件工程项目的最佳实践

主要内容

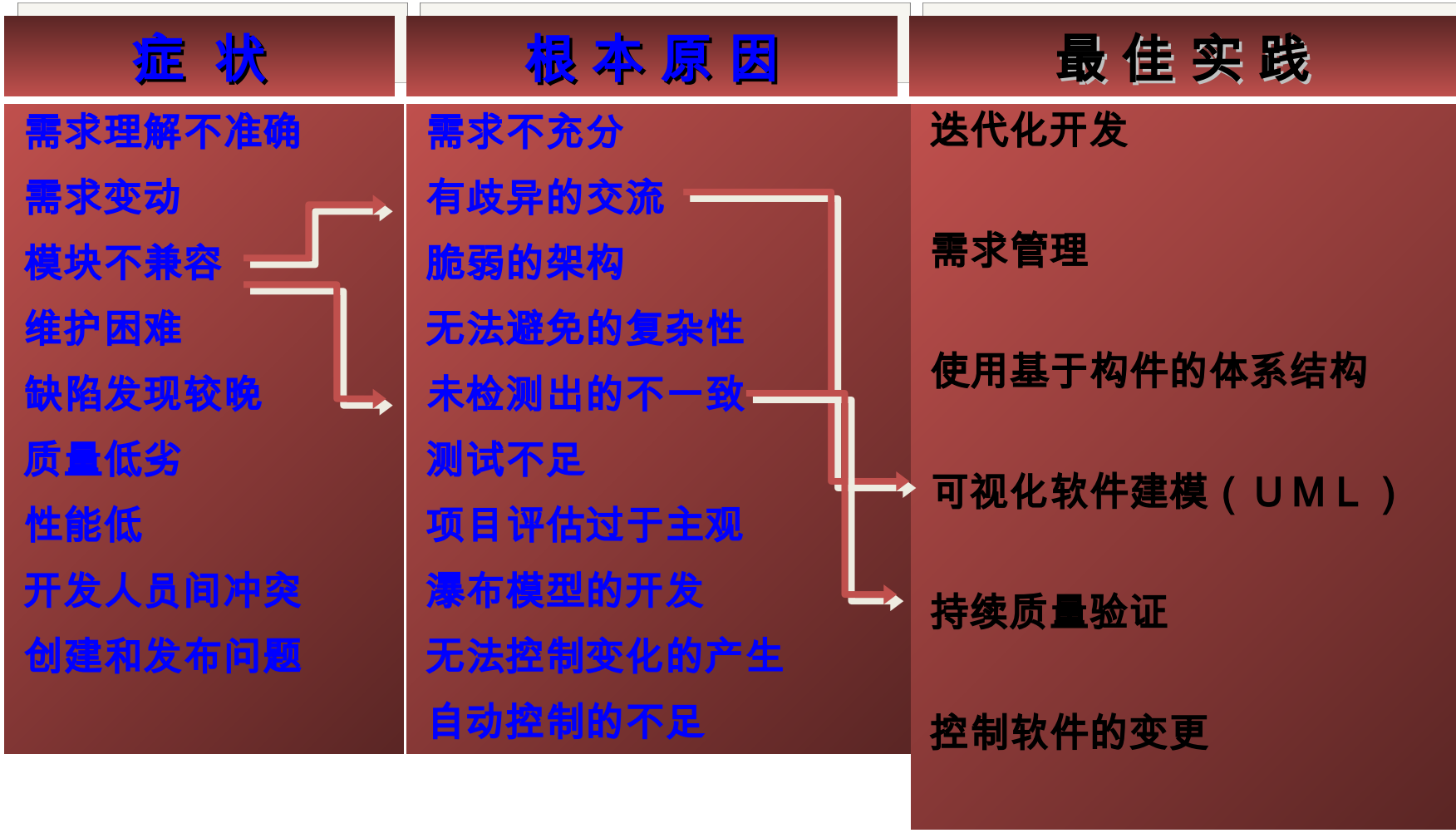
软件工程的最佳实践

面向对象的基本特征

软件开发常见问题

- 对用户的需求理解不准确
- 对需求的改变束手无策
- 模块不兼容
- 软件维护困难
- 项目的严重缺陷发现较晚
- 软件质量低劣或用户缺少经验
- 高负荷下性能低
- 不配合的团队力量
- 不可靠创建和发布过程

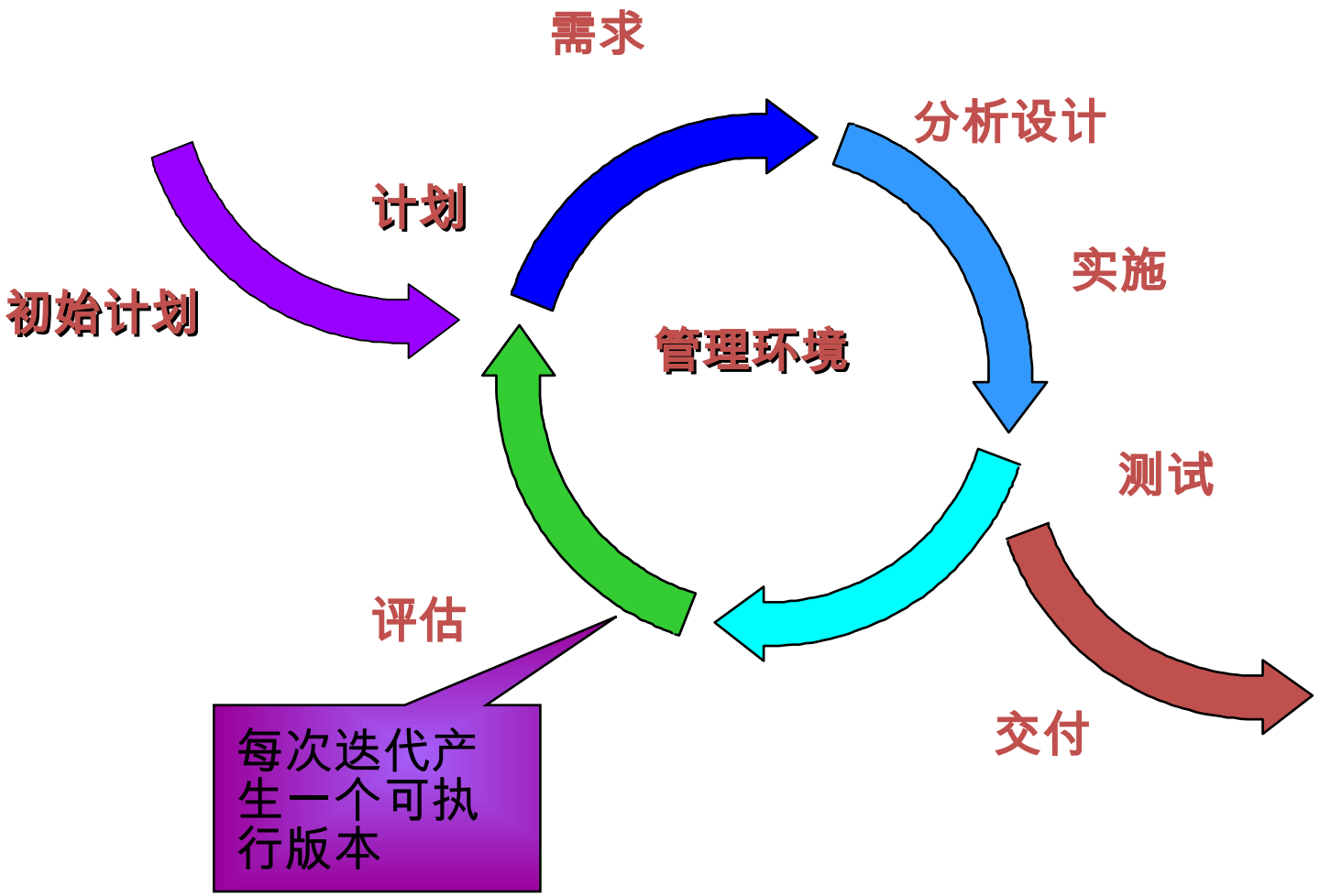
追踪问题的根源


症 状
根 本 原 因
最 佳 实 践
需求理解不准确
需求变动
模块不兼容
维护困难
缺陷发现较晚
质量低劣
性能低
开发人员间冲突
创建和发布问题
需求不充分
有歧异的交流
脆弱的架构
无法避免的复杂性
未检测出的不一致
测试不足
项目评估过于主观
瀑布模型的开发
无法控制变化的产生
自动控制的不足
迭代化开发
需求管理
使用基于构件的体系结构
可视化软件建模（UML）
持续质量验证
控制软件的变更

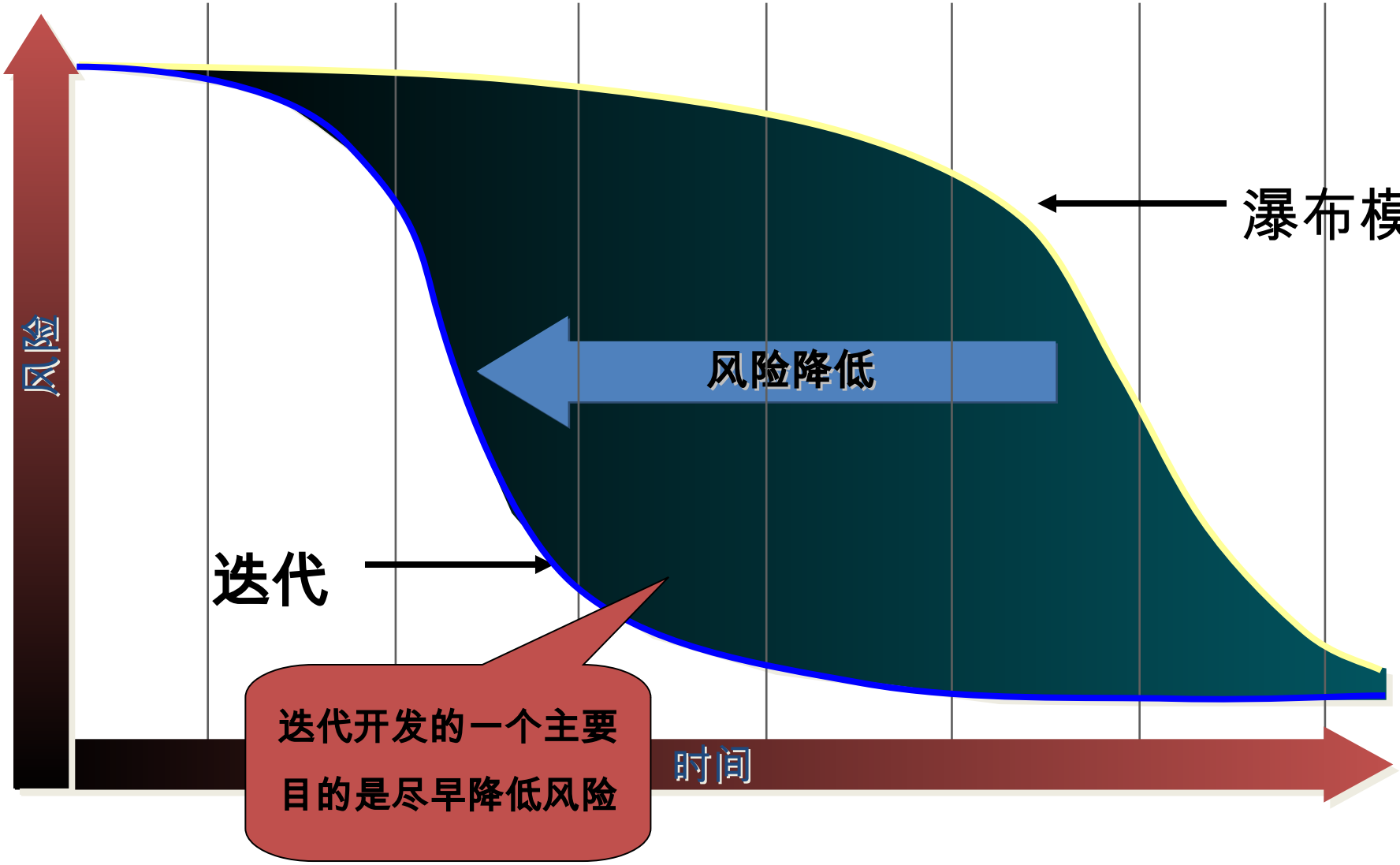
六个最佳实践

- 迭代化开发
- 需求管理
- 使用基于构件的体系结构
- 可视化软件建模
- 持续质量验证
- 控制软件变更

迭代化开发



风险比较



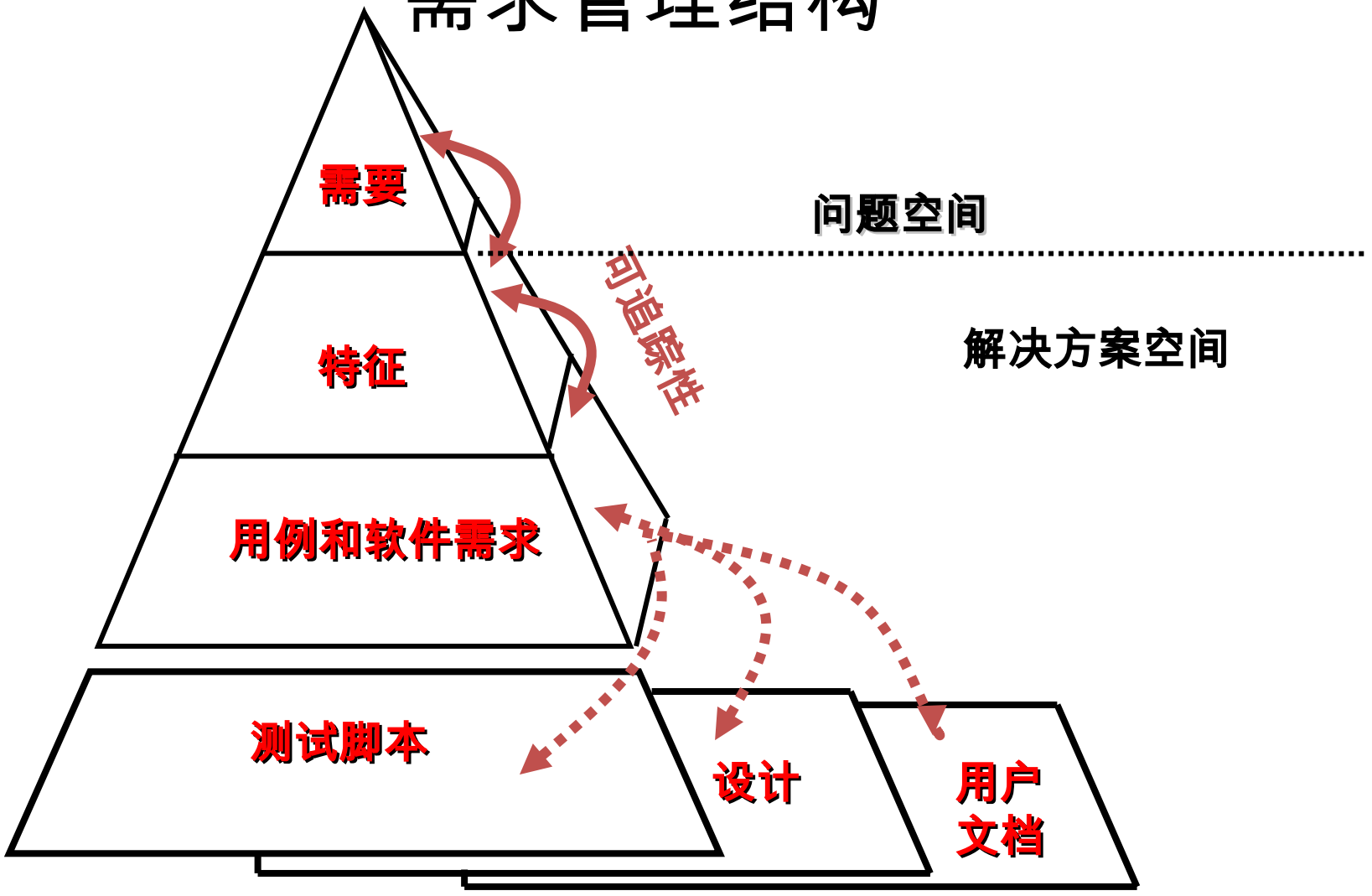
需求管理

- 建立体系化的方法来提取，组织，记载和管理变化的需求，确保
  - 解决了正确的问题
  - 构建了正确的系统

需求管理的内容

- 分析问题
- 理解涉众需要
- 定义系统
- 管理项目规模
- 改进系统定义
- 管理需求变更

需求管理结构



基于构件的体系结构

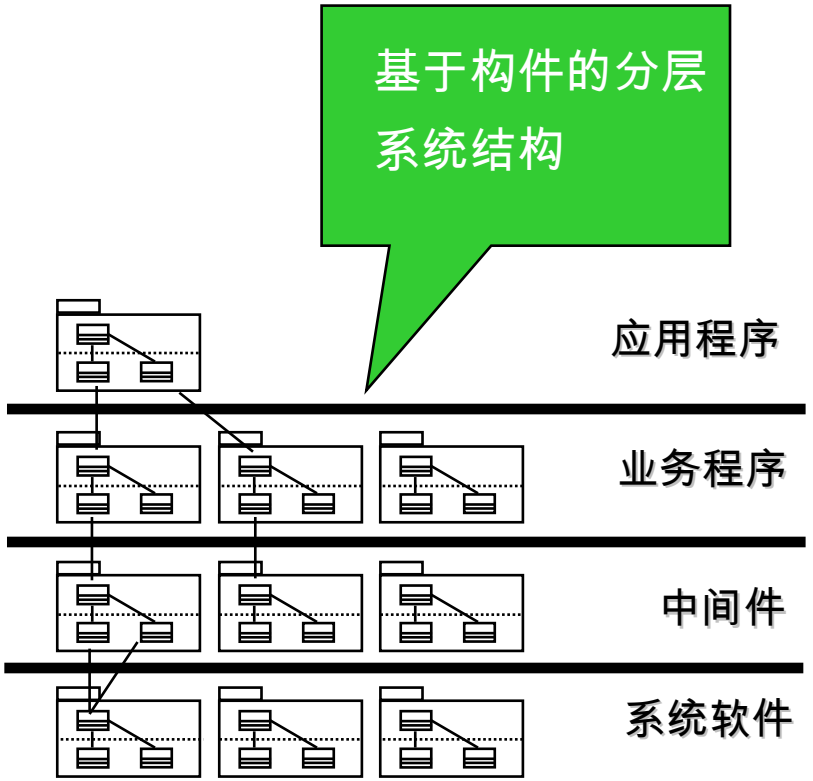
- 描述了如何设计灵活的，可容纳修改的，直观便于理解的，并且促进有效软件重用的弹性结构

有弹性的基于构件的架构

- 弹性架构
  - 满足当前和未来的需求
  - 改进可扩展性
  - 支持复用
  - 系统依赖的封装
- 基于构件
  - 复用或定制构件
  - 选择可利用的商务构件
  - 现有软件的增值式发展

基于构件的体系结构的目标

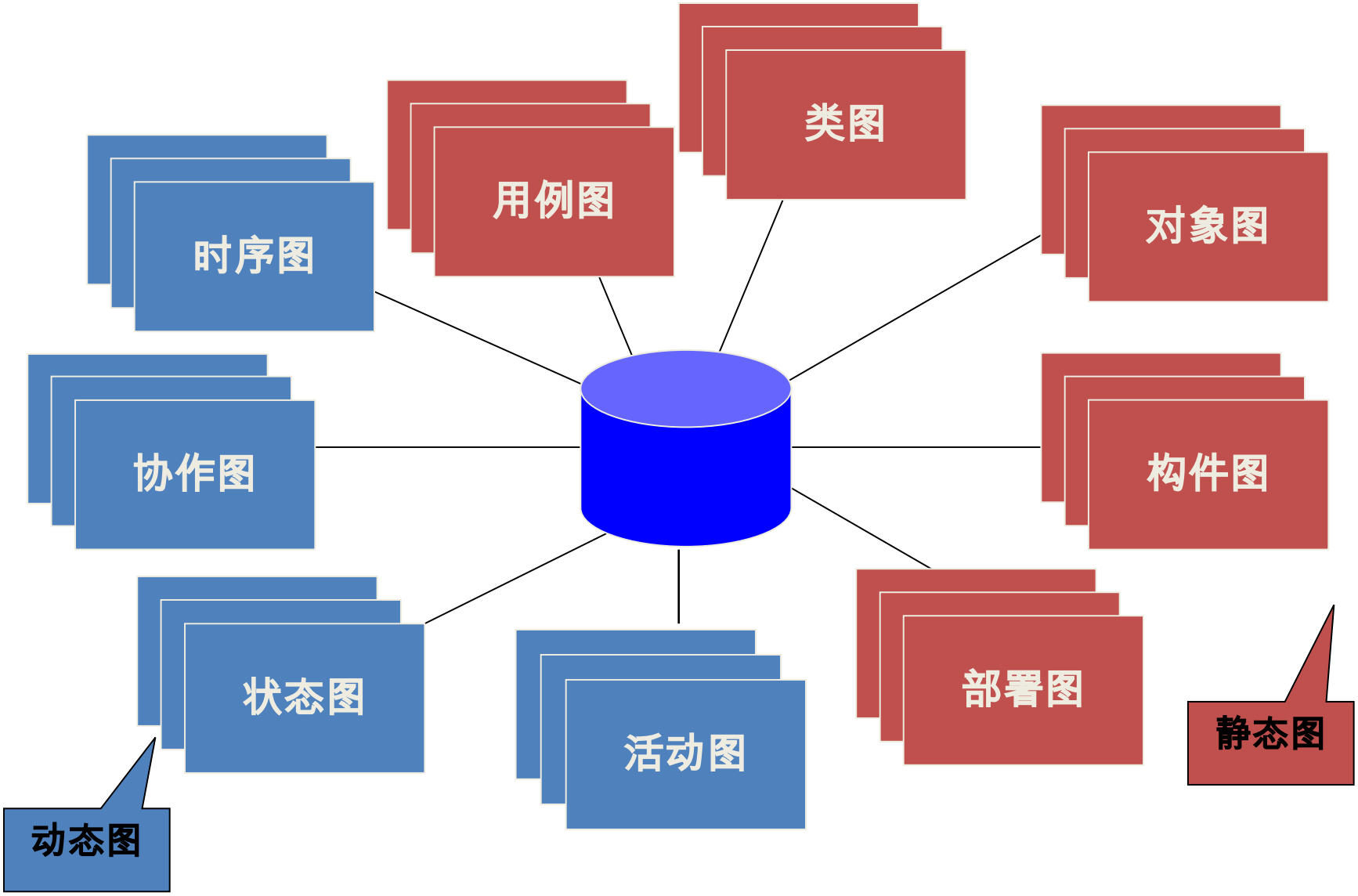
- 复用基础
  - 构件复用
  - 架构复用
- 项目管理基础
  - 计划
  - 人员分配
  - 交付
- 智能控制
  - 管理复杂性
  - 维护完整性



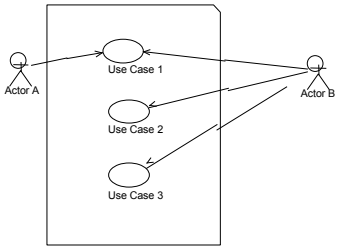
可视化软件建模

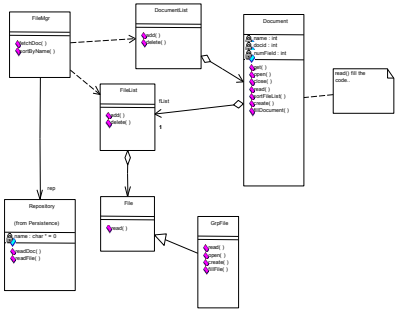
- 为什么需要可视化建模
  - 捕获系统的静态结构和动态行为
  - 显示系统各部件如何配合
  - 保持设计和实现的一致性
  - 根据需要适当隐藏或显示细节
  - 促进无歧异的沟通
    - UML ：适用于所有工作人员的语言

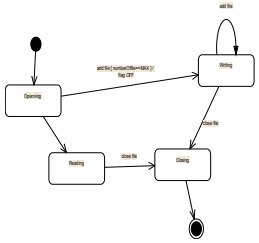
UML 可视化建模



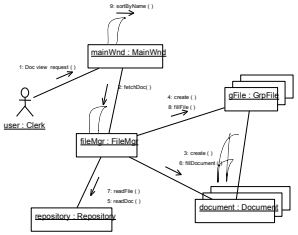
可视化模型

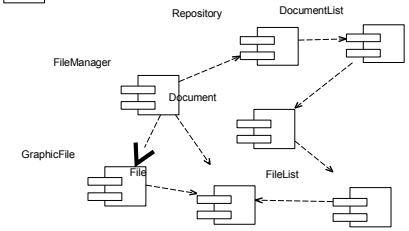




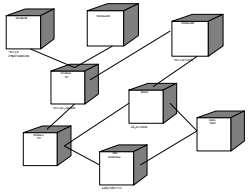


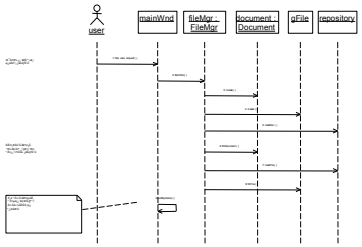
协作图



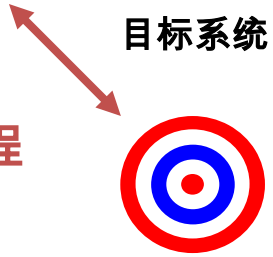


构件图





时序图



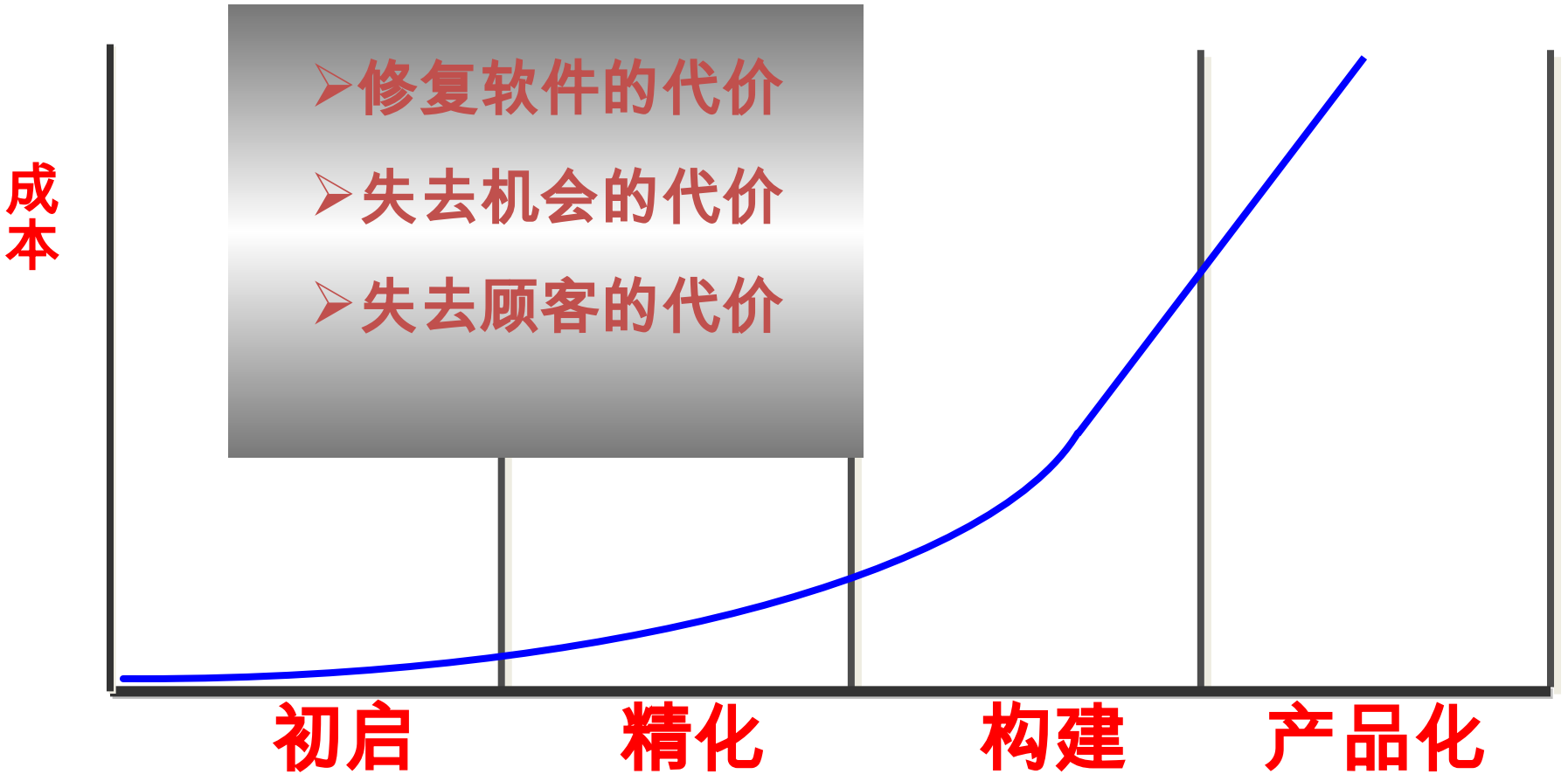
目标系统

双向工程

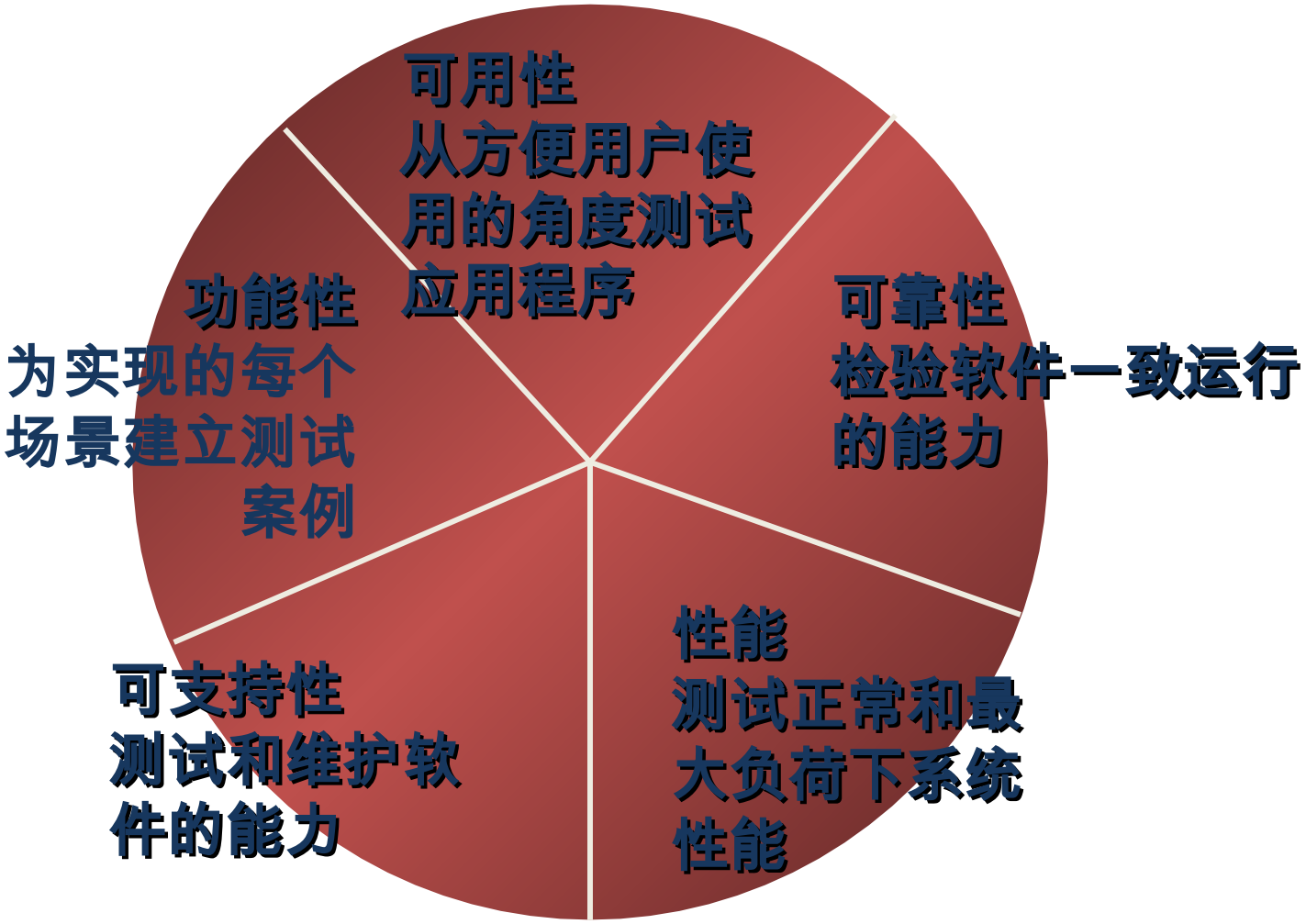
持续质量验证

- 质量评估被内建于过程、所有的活动，包括全体成员，使用客观的度量和标准，并且不是事后型的或单独小组进行的分离活动

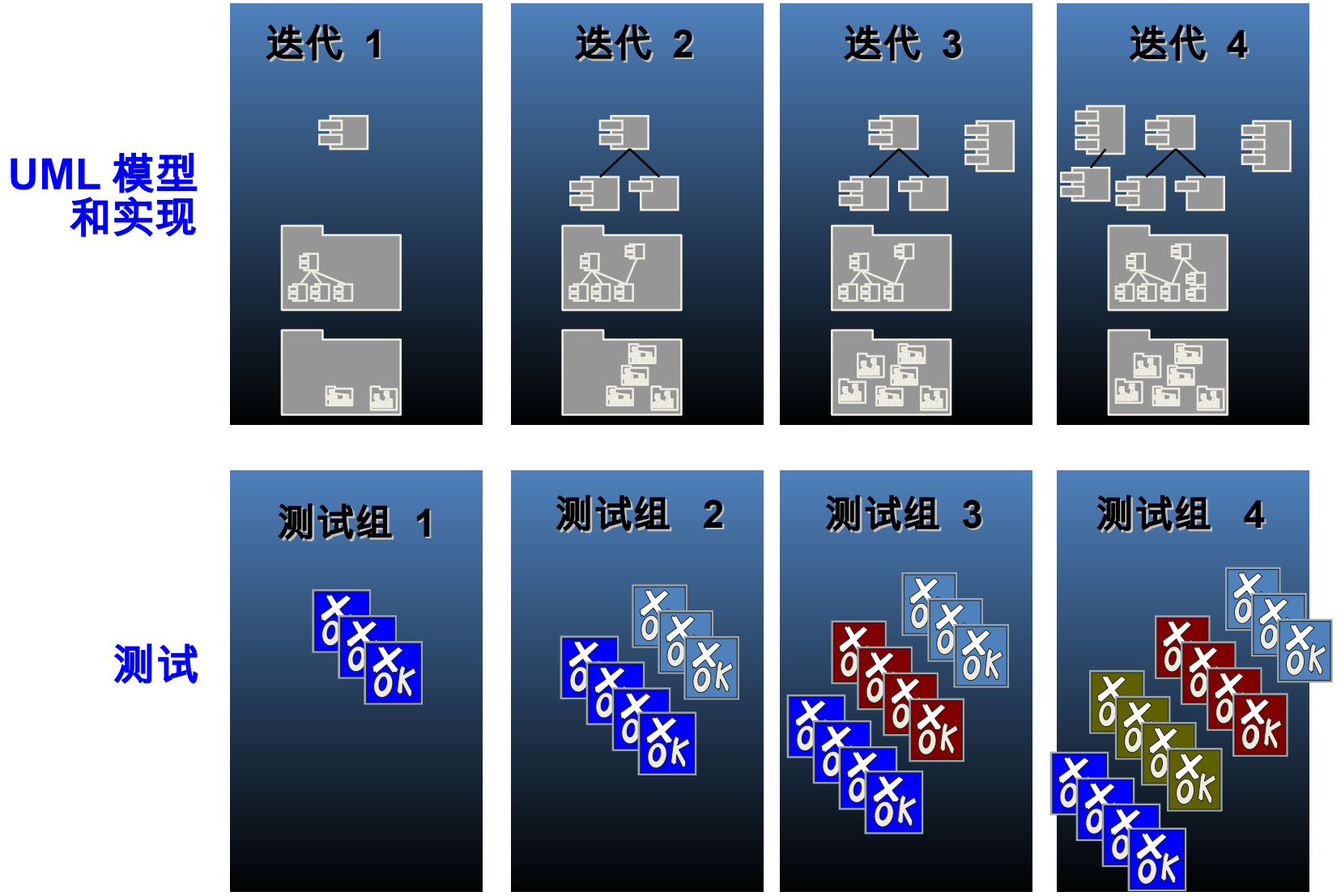
产品化阶段，发现和修复软件问题需要成百上千倍的成本



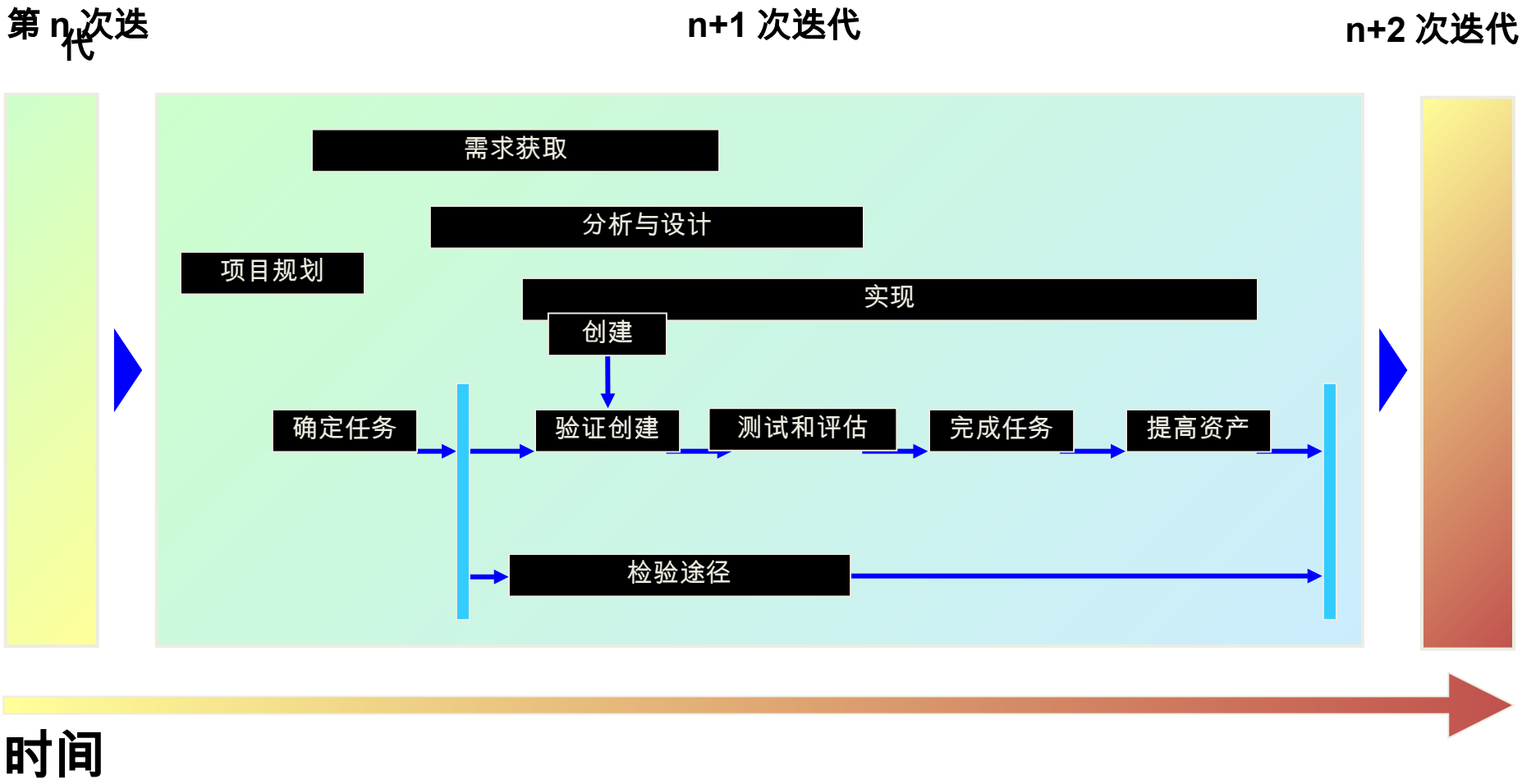
从多方面测试软件质量


可用性
从方便用户使用的角度测试应用程序
可靠性
检验软件一致运行的能力
性能
测试正常和最大负荷下系统性能
可支持性
测试和维护软件的能力
功能性
为实现的每个场景建立测试案例

测试每个迭代

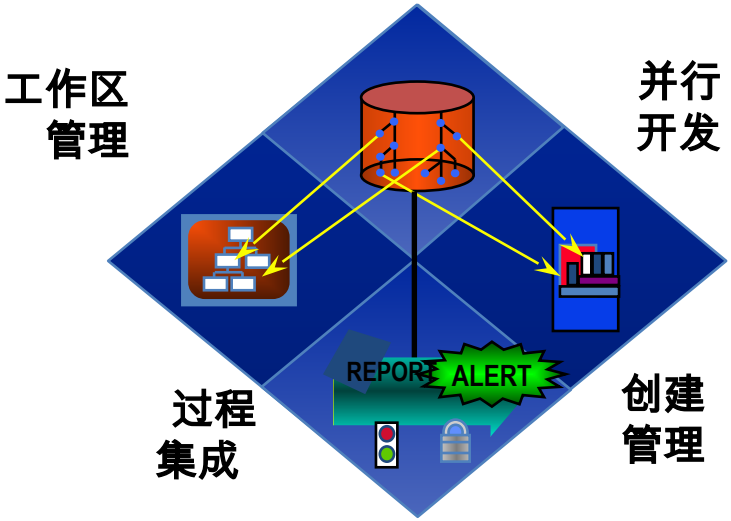


软件开发生命期中的测试



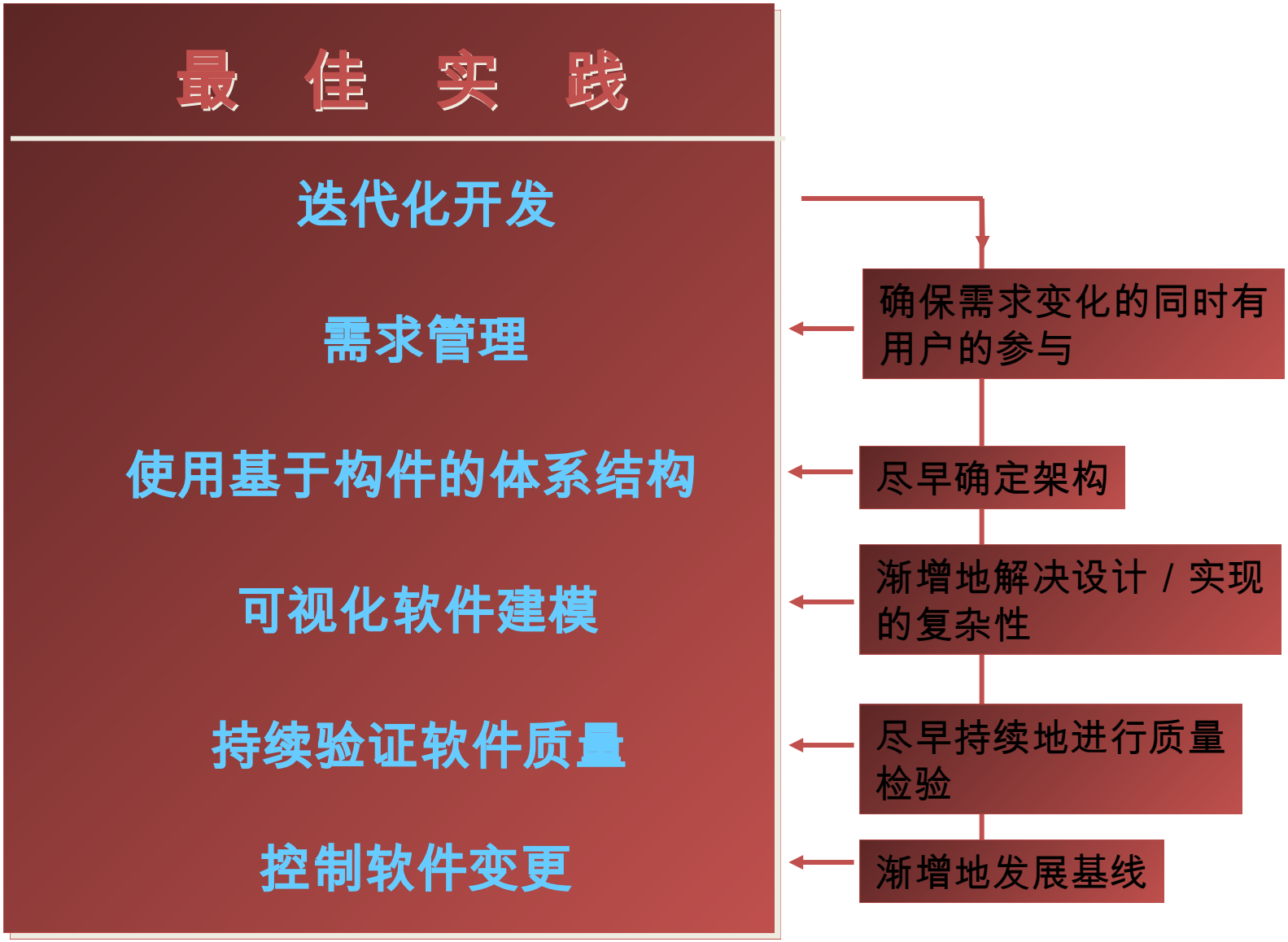
控制软件变更

- 需要控制的是
  - 每个开发者安全的工作区
  - 自动化集成管理
  - 并行开发



配置管理不仅仅是检入与检出

最佳实践间的相互增强作用



最佳实践的实现

- 面向对象技术促进了最佳实践的实现
  - 迭代化开发：能适应变化的需求，渐增地集成元素，更易于软件复用
  - 使用基于构件的体系结构：结构上强调基于构件的开发
  - 可视化建模：便于理解，容易修改

第二章
软件工程的最佳实践

主要内容

软件工程的最佳实践

面向对象的基本特征

什么是面向对象技术

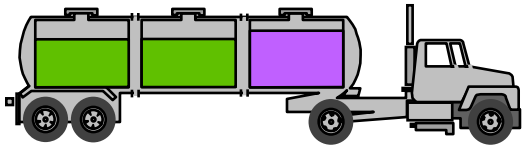
- 面向对象技术是一个综合概念，既包括指导开发软件的一系列原则，也包括支撑这些原则的程序设计语言，数据库及其它工具。

面向对象方法的特点

- 提供了单一的范例
- 便于架构和代码的复用
- 模型更接近地反映客观世界
- 更好的稳定性
- 能更好地适应变更

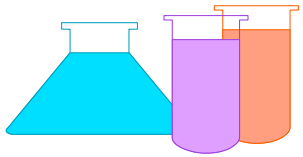
什么是对象

- 非正式的，一个对象代表了一个实体
- 可以是实际的，概念上的，软件
  - 实际的实体



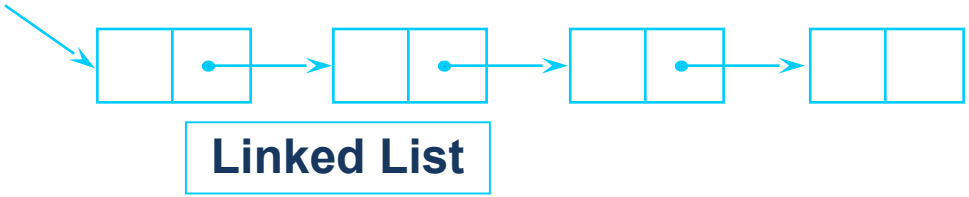
Truck

  - 概念上的实体



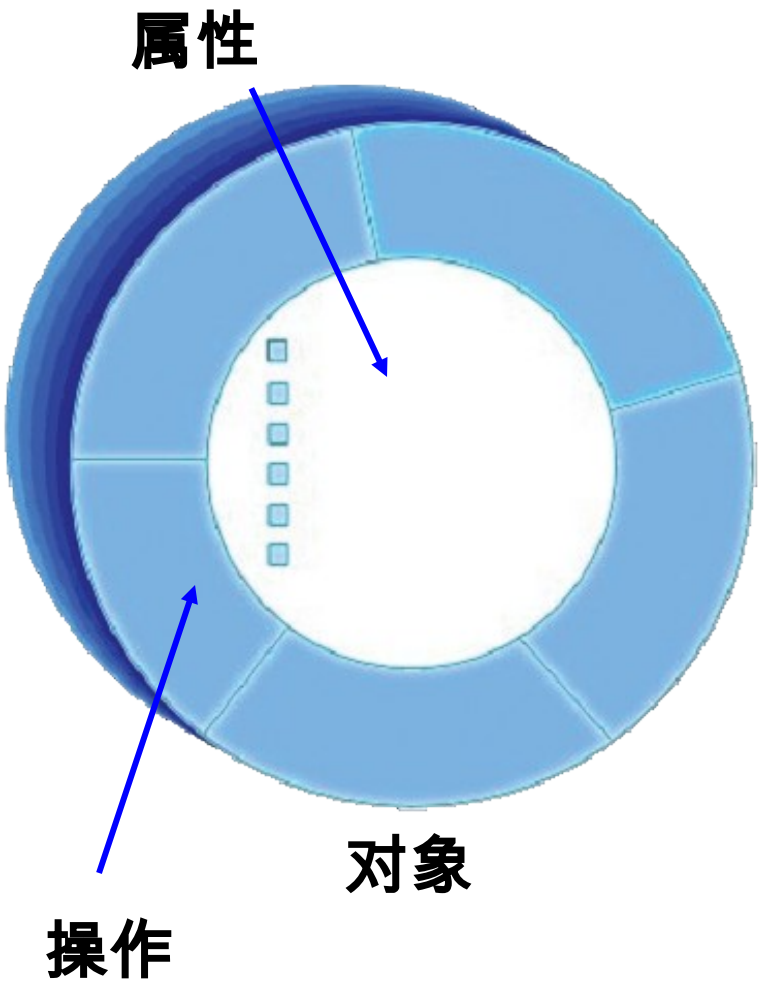
Chemical Process

  - 软件实体



对象的正式定义

- 对象是有明确定义的边界和封装了状态和行为的实体
  - 状态由属性和关系表示
  - 行为由操作，方法和状态机表示



对象具有状态

- 对象的状态指对象可能出现的情形
- 对象的状态通常会随时间而改变

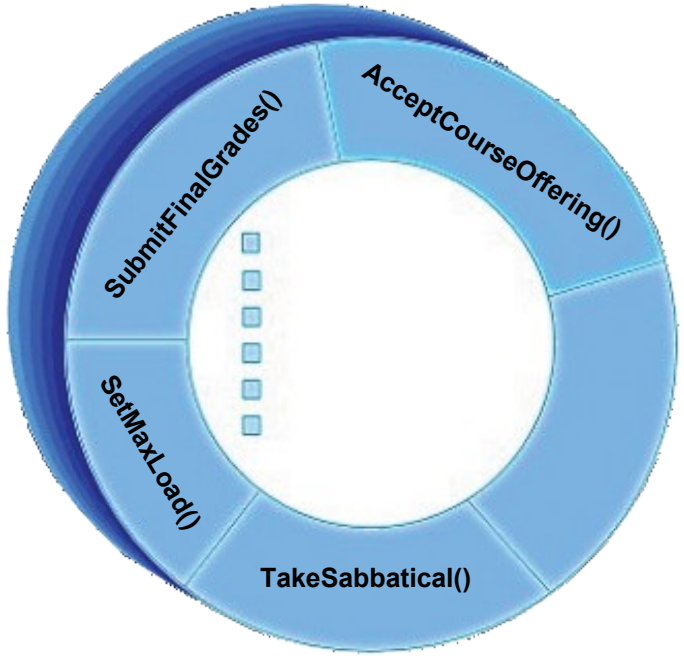
Name: J Clark
Employee ID: 567138
Date Hired: July 25, 1991
Status: Tenured
Discipline: Finance
Maximum Course Load: 3 classes

Professor Clark

对象具有行为

- 行为决定了对象如何行动和对其它对象发来的要求所做的反应
- 对象可见的行为表示为它能响应的消息的集合（能执行的操作的集合）

Professor Clark's behavior
Submit Final Grades
Accept Course Offering
Take Sabbatical
Maximum Course Load: 3 classes



Professor Clark

对象可唯一识别

- 即使对象有相同的特性，还是能识别每个不同的对象

Professor “J Clark”
teaches Biology

Professor “J Clark”
teaches Biology

什么是类

- 类是对一组有着相同属性，操作，关系和语义的对象的描述
  - 对象是类的实例
- 类是一种抽象
  - 强调相关的特征
  - 抑制其它的特征

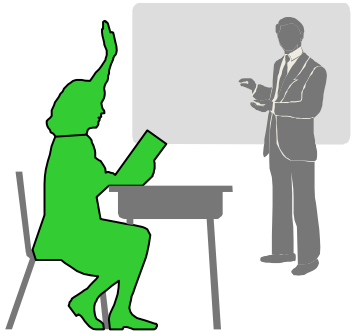
面向对象的主要技术

- 抽象
- 封装
- 模块化
- 继承
- 多态

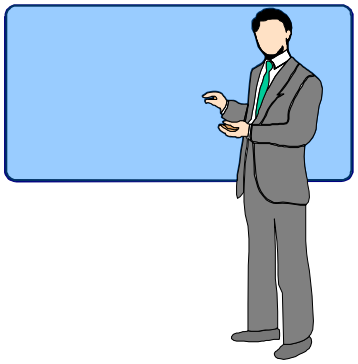
抽象（ Abstraction ）

- 指着重于某重要的一面或想关注的一面，来表示某个物体或概念
- 面向对象技术通过抽象化现实世界中的物体，来描述一个系统

抽象示例

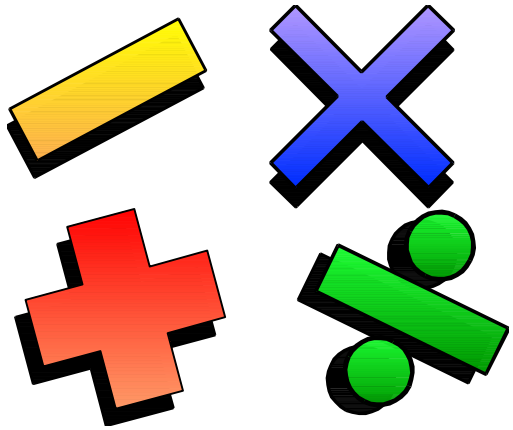


学生



教授

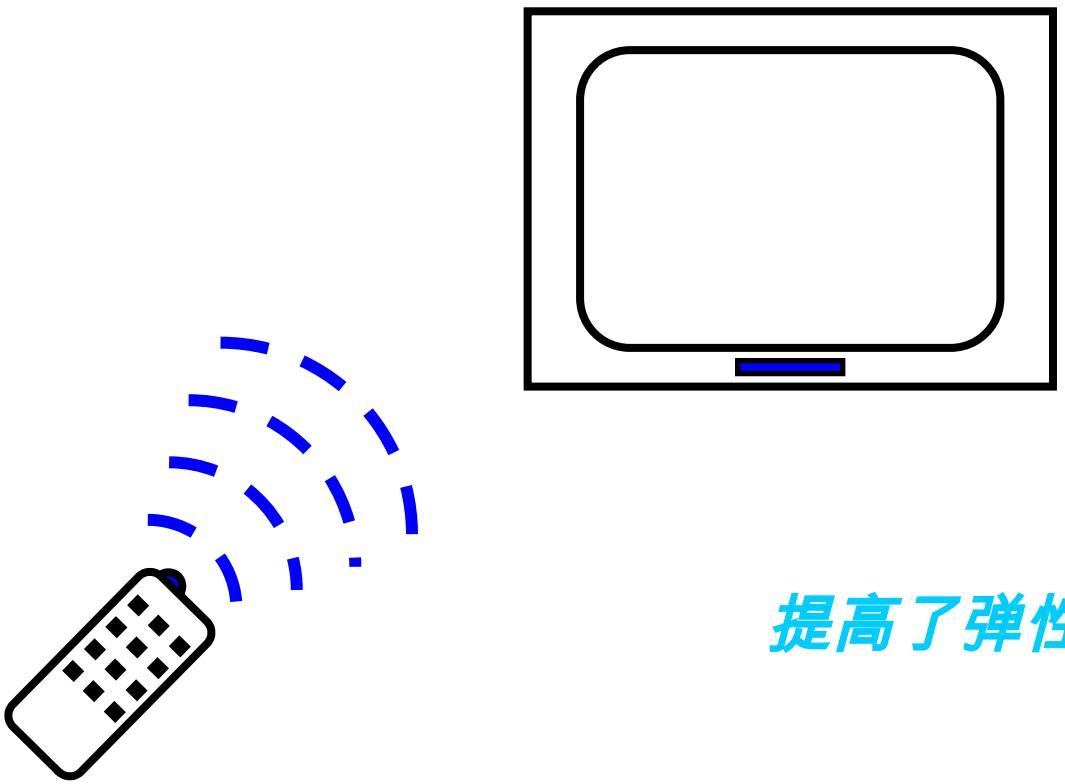
课程提供 (9:00 AM，周一，周三，周五)



课程（例如，代数学）

封装（ Encapsulation ）

- 对客户隐藏实现
  - 客户依赖于接口



提高了弹性

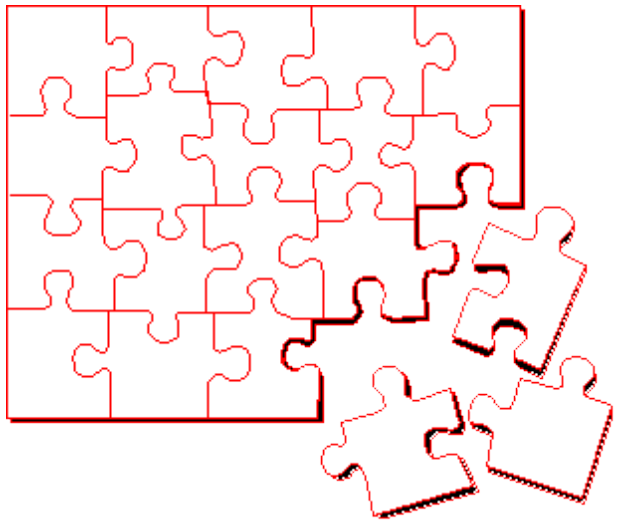
封装示例

- Professor Clark 下学期教授四门课程

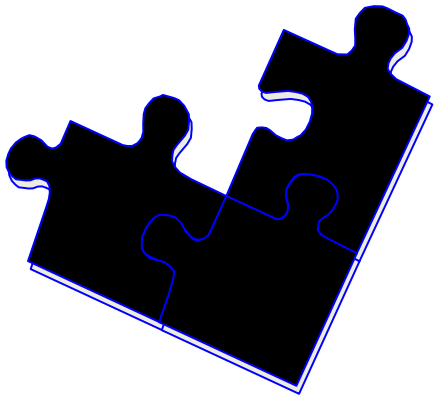
SetMaxLoad(4)

模块化（ Modularity ）

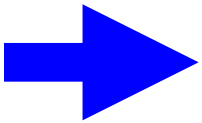
- 模块化体现在将复杂问题分解为可处理的小部分
- 模块化帮助人们理解复杂的系统



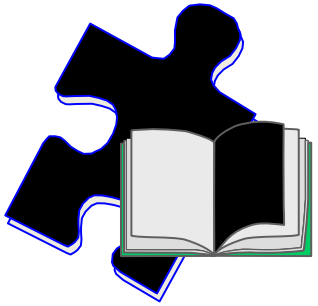
模块化示例



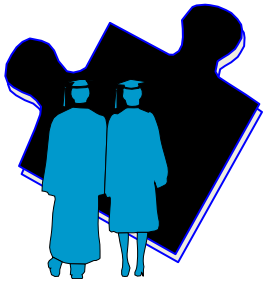
课程注册系统



付费系统



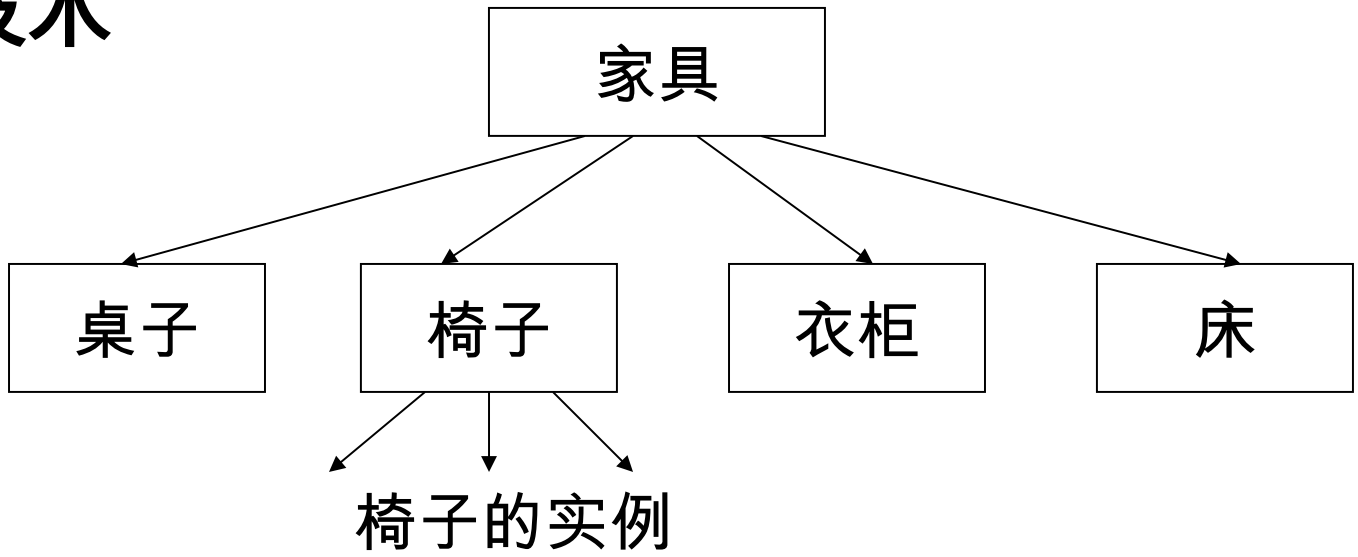
课程目录系统



学生管理系统

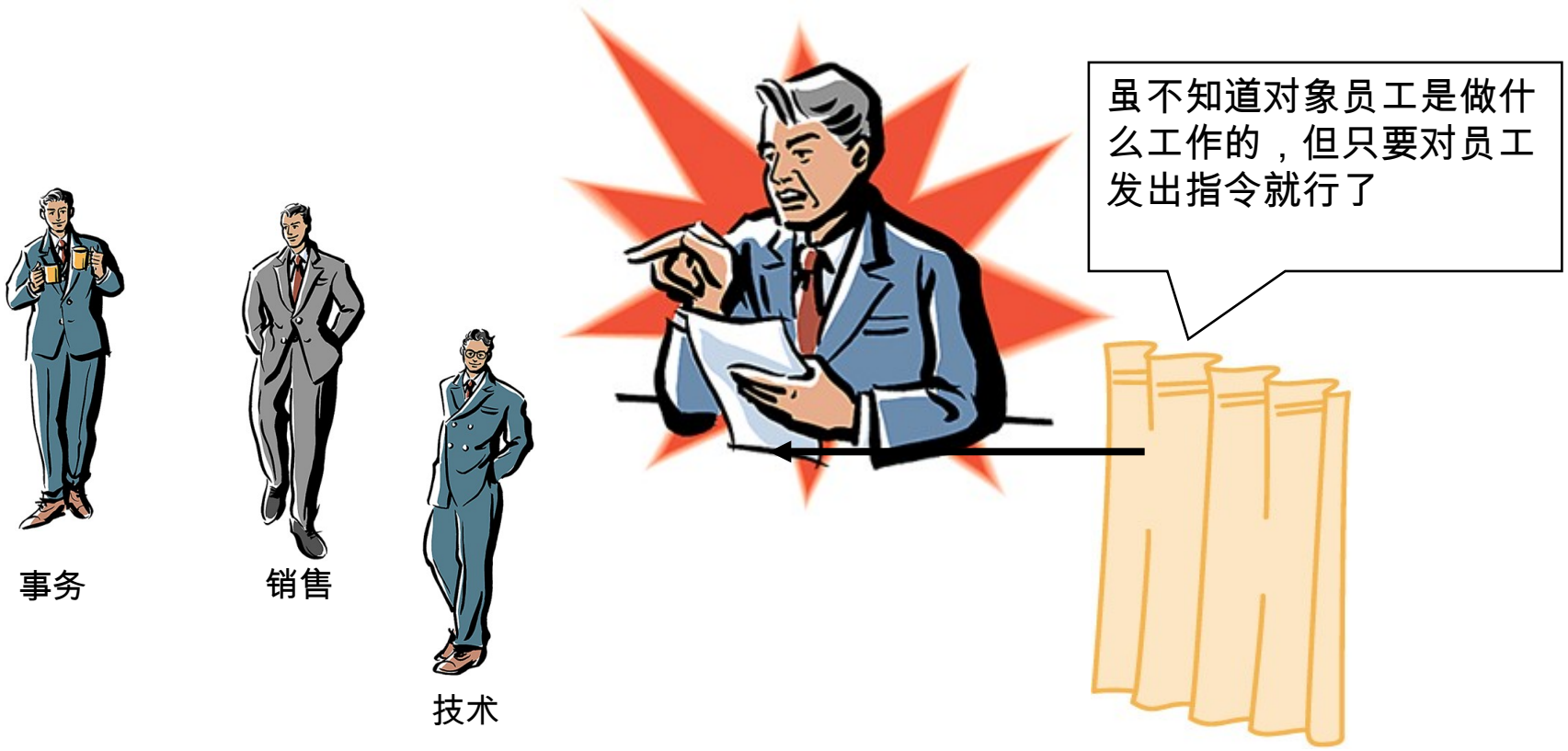
继承（ Inheritance ）

- 使用已存在的定义做为基础建立新定义的技术



多态（ Polymorphism ）

- 对于相同的消息，让各个对象产生不同的行为



多态性示例

计算各员工工资的程序

不用多态性时

```
if ( 员工对象 =“ 销售” ) {
    payment = 计算销售人员工资 ();
} else if ( 员工对象 = “ 技术” ) {
    payment = 计算技术人员工资 ();
} else if ( 员工对象 = “ 事务” ) {
    payment = 计算事务人员工资 ();
}
```

工作分工一增加，程序就必须修改

使用多态性时

```
payment = 员工对象 . 工资计算 ();
```

即使工作分工增加，程序也不需要修改

小结

- 软件工程的六个最佳实践
- 面向对象方法论

作业

- 简述软件工程的 6 个最佳实践
- 分别简述软件工程 6 个最佳实践应对的是什么挑战
- 解释对象、类、继承、聚合、多态、封装
- 论述面向对象的优势（特点）