



# 系统架构设计师

DESIGNER:王川林  
软件工程



## 课程内容提要

系统规划

软件开发方法

面向对象基础

需求工程

软件系统建模

软件架构设计

系统设计

测试与评审

软件开发环境与工具

系统运行与评价

软件开发方法 ★★★  
软件开发模型 ★★★★★

基础概念 ★★  
UML4+1视图 ★★★★★  
UML图 ★★★★★★  
UML关系 ★★★★★

需求获取 ★★★★★  
需求分析 ★★★★★

架构风格 ★★  
架构评估 ★★

界面设计 ★★  
业务流程设计 ★★★★★  
面向对象设计原则 ★★★★★★  
设计模式 ★★★★★★

可维护性因素 ★★★★★

- ➡ 软件开发生命周期
- ➡ 软件开发模型
- ➡ 构件与软件重用
- ➡ 逆向工程
- ➡ 净室软件工程

。

瀑布模型

演化模型

增量模型

螺旋模型

快速原型模型

喷泉模型

V模型

迭代模型/迭代开发方法

快速应用开发

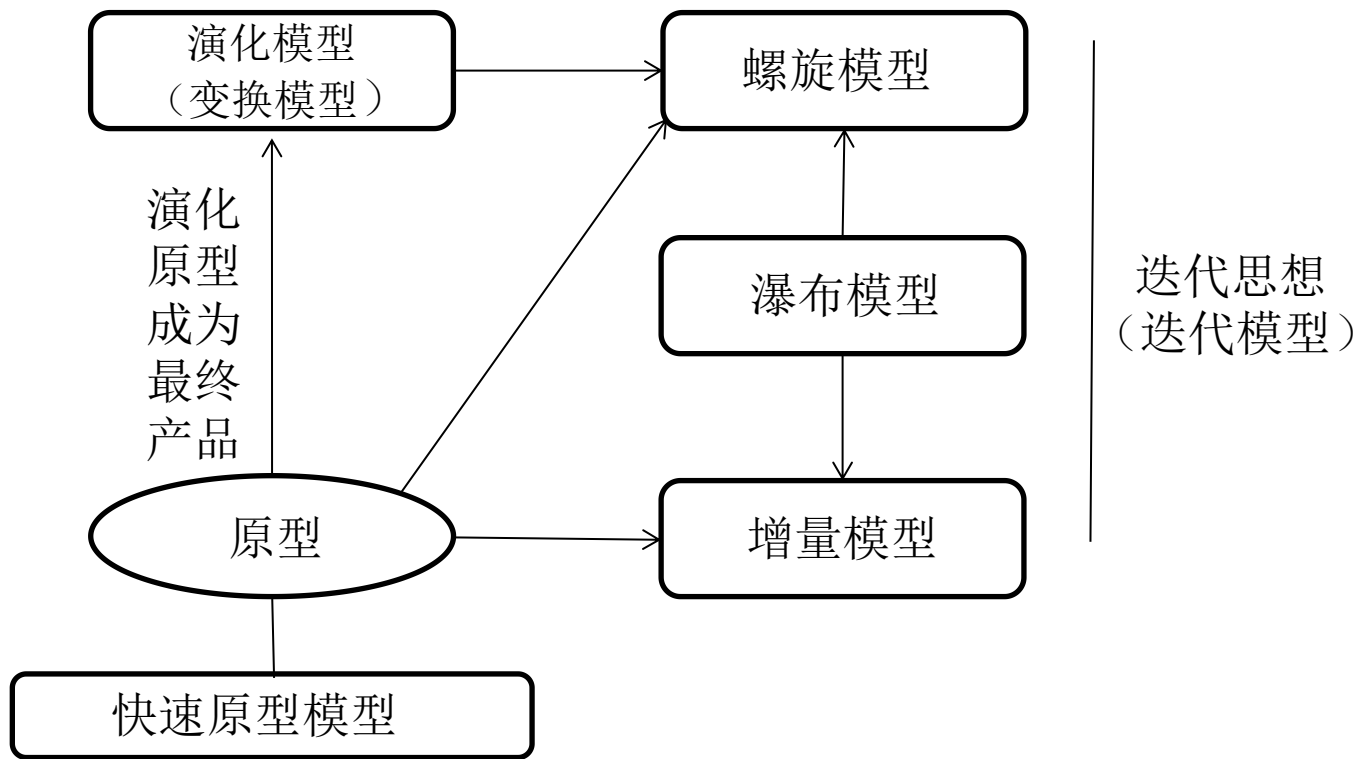
构件组装模型/基于构件的开发方法

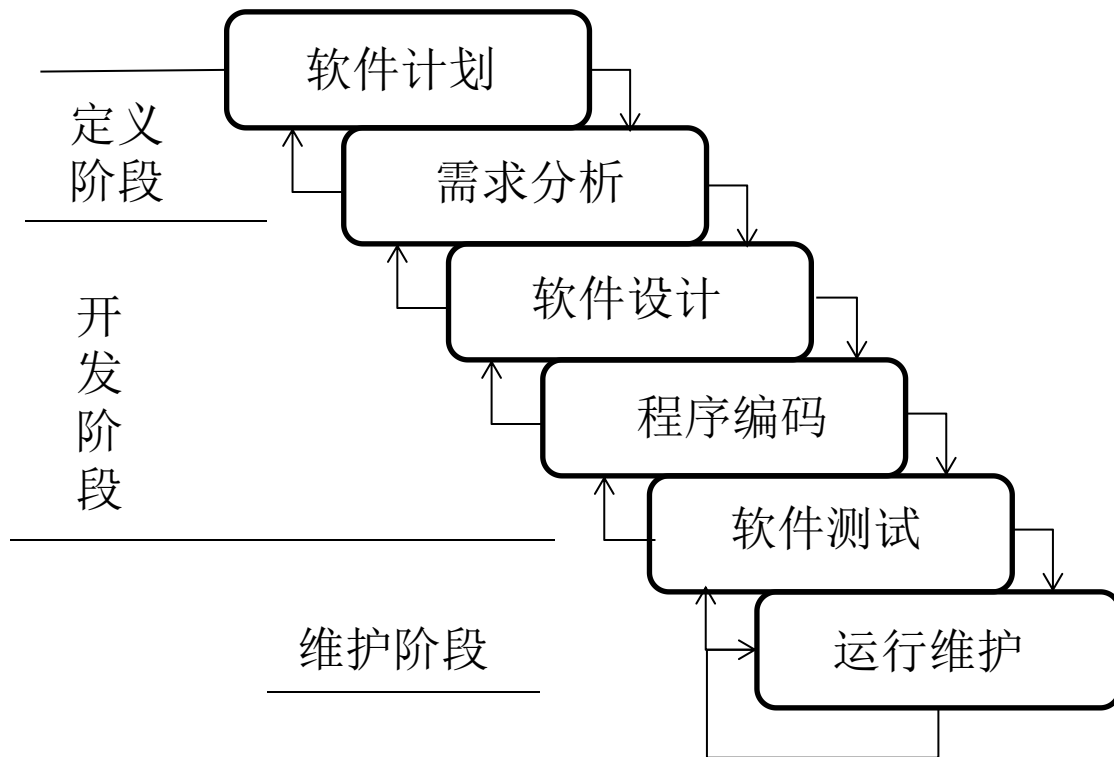
统一过程/统一开发方法

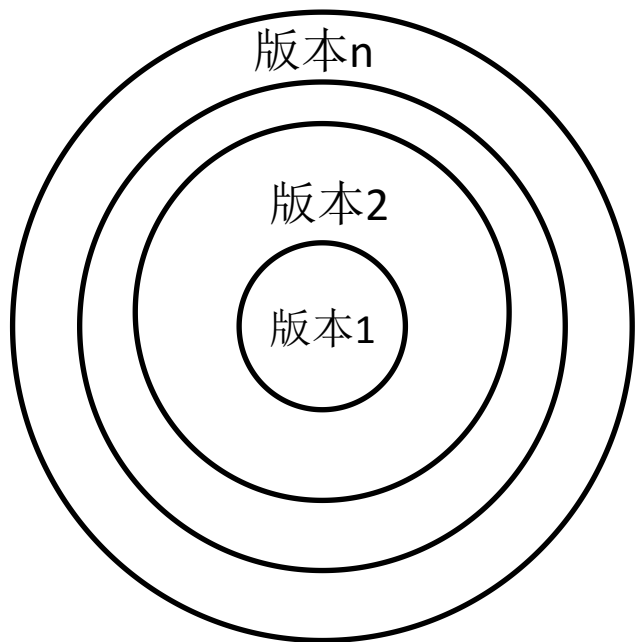
敏捷开发方法

模型驱动的开发方法

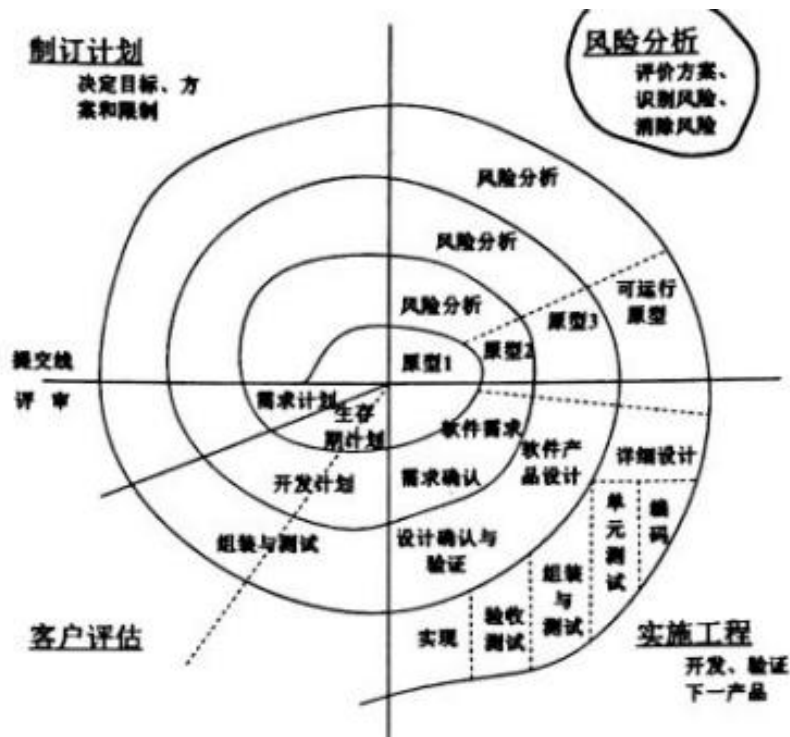
基于架构的开发方法

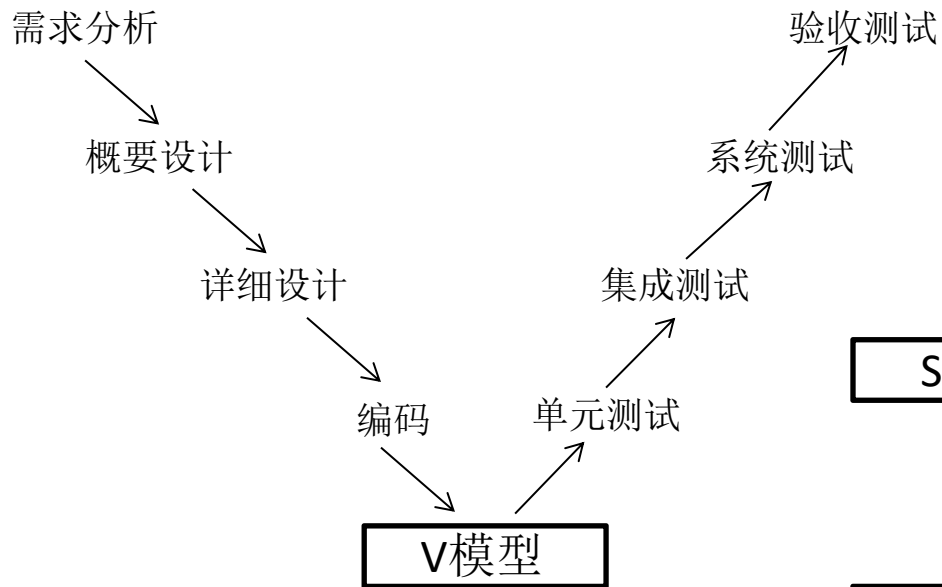






每一个增量均发布  
一个可操作的产品





喷泉模型

迭代  
无间隙

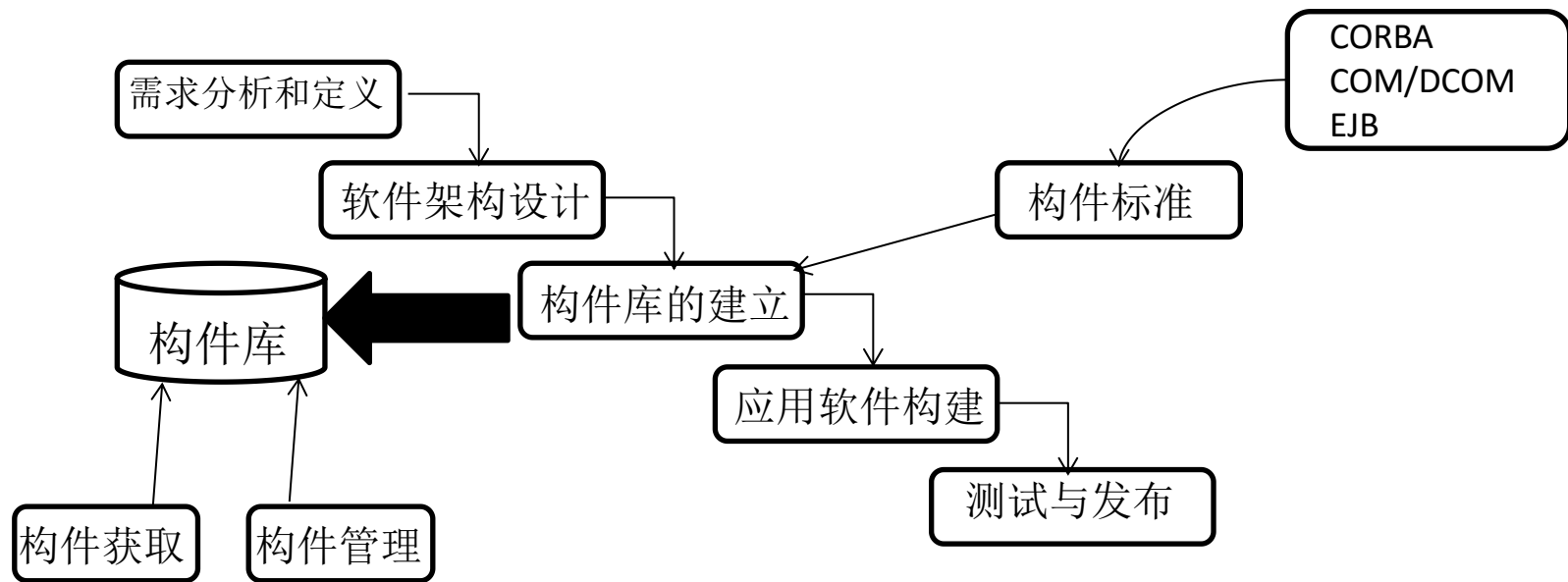
SDLC

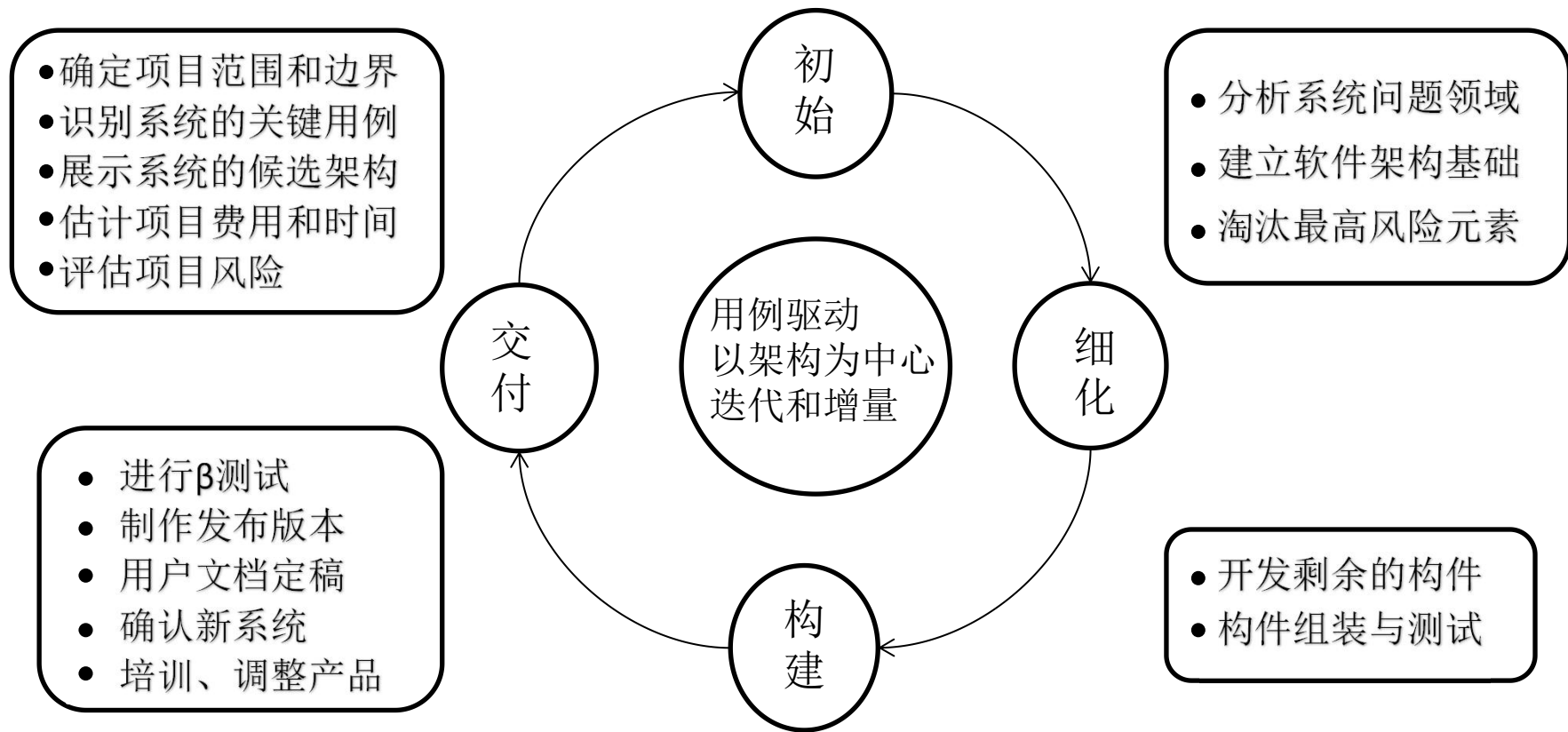
CBSD

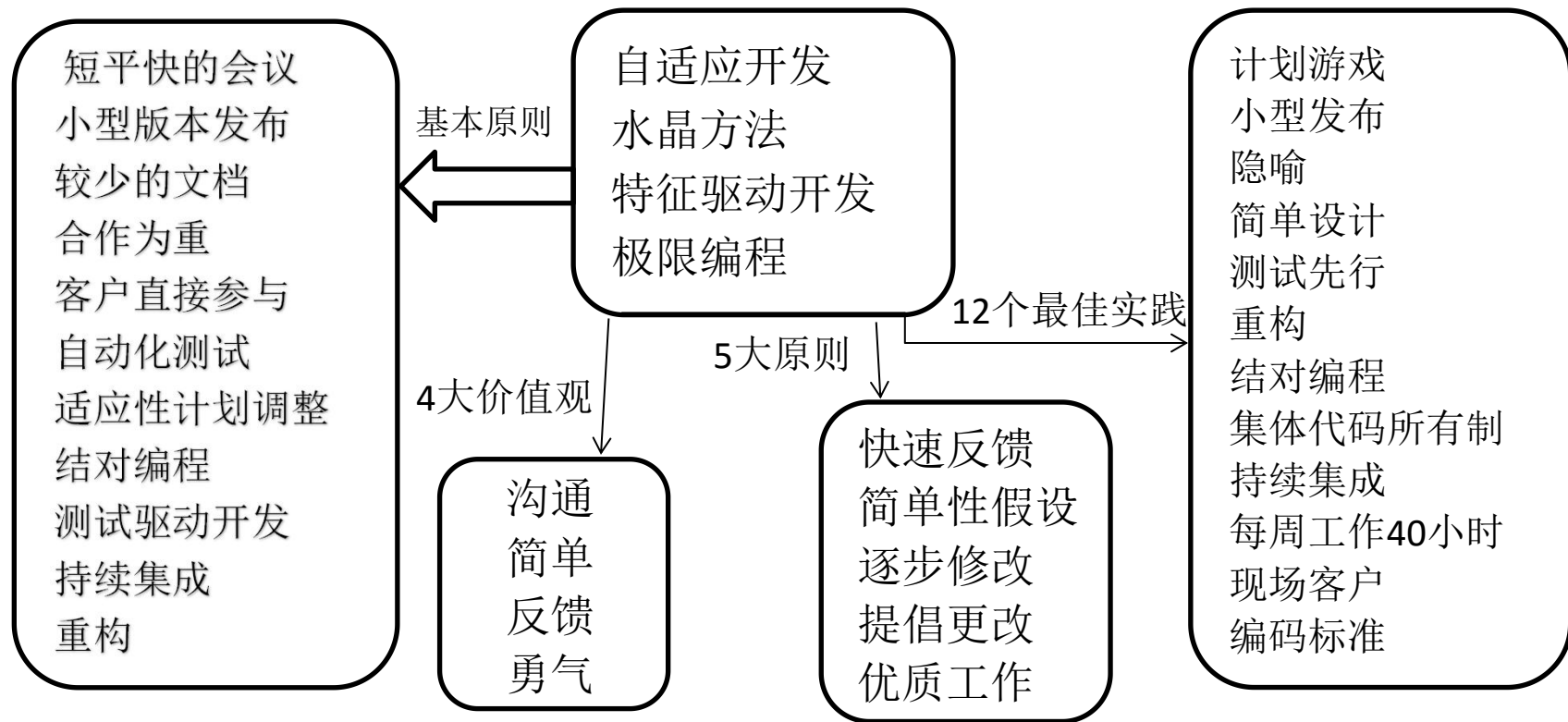
RAD

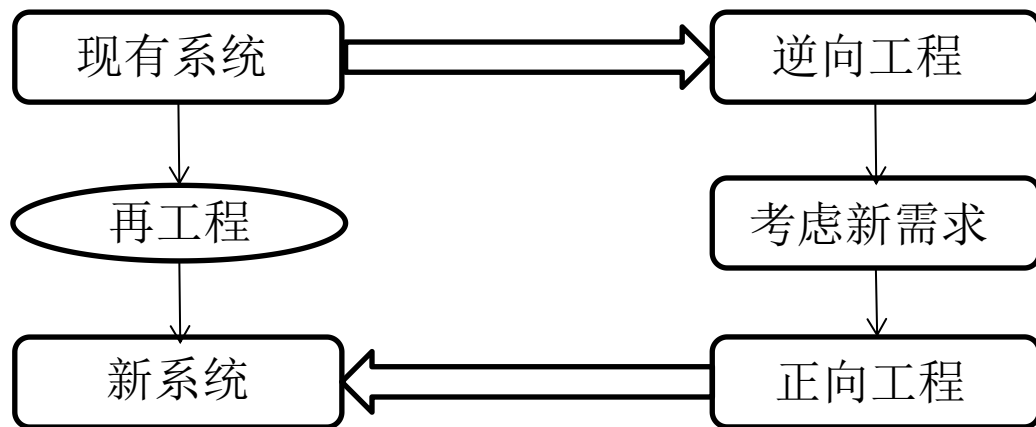
业绩建模  
数据建模  
过程建模  
应用生成  
测试与交付











**实现级：**包括程序的抽象语法树、符号表、过程的设计表示

**结构级：**包括反映程序分量之间相互依赖关系的信息，例如调用图、结构图、程序和数据结构

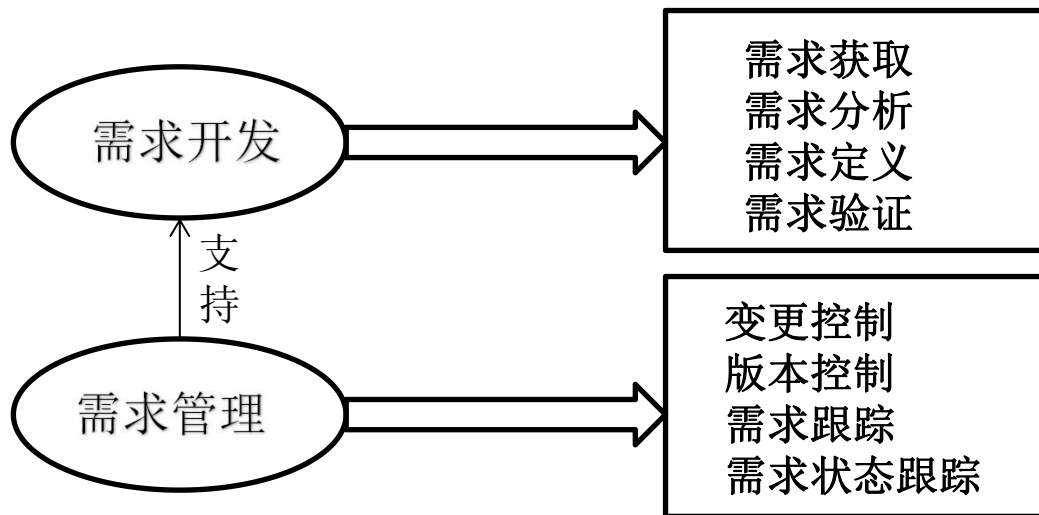
**功能级：**包括反映程序段功能及程序段之间的关系的信息，例如数据和控制流模型

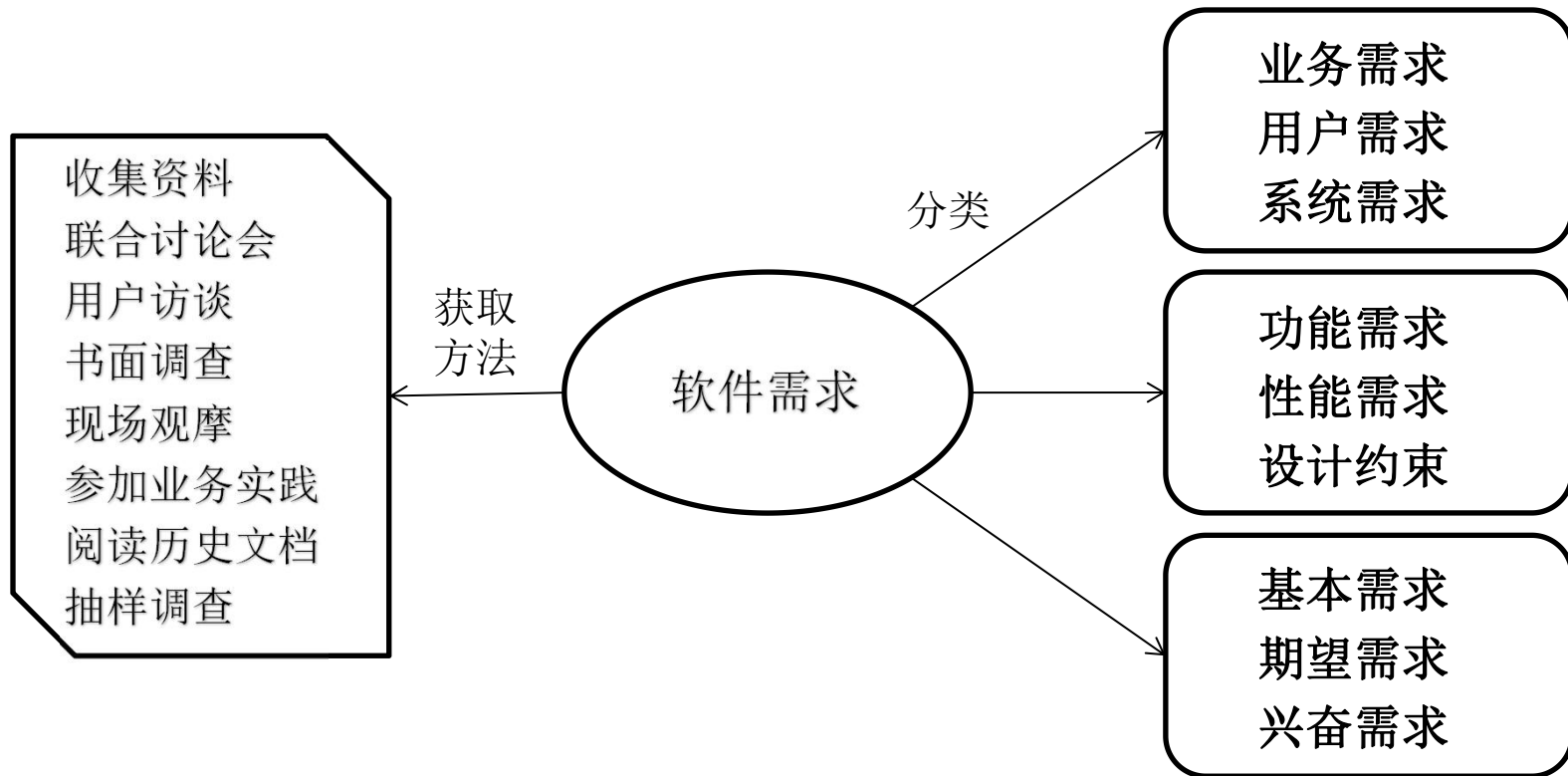
**领域级：**包括反映程序分量或程序诸实体与应用领域概念之间对应关系的信息，例如实体关系模型

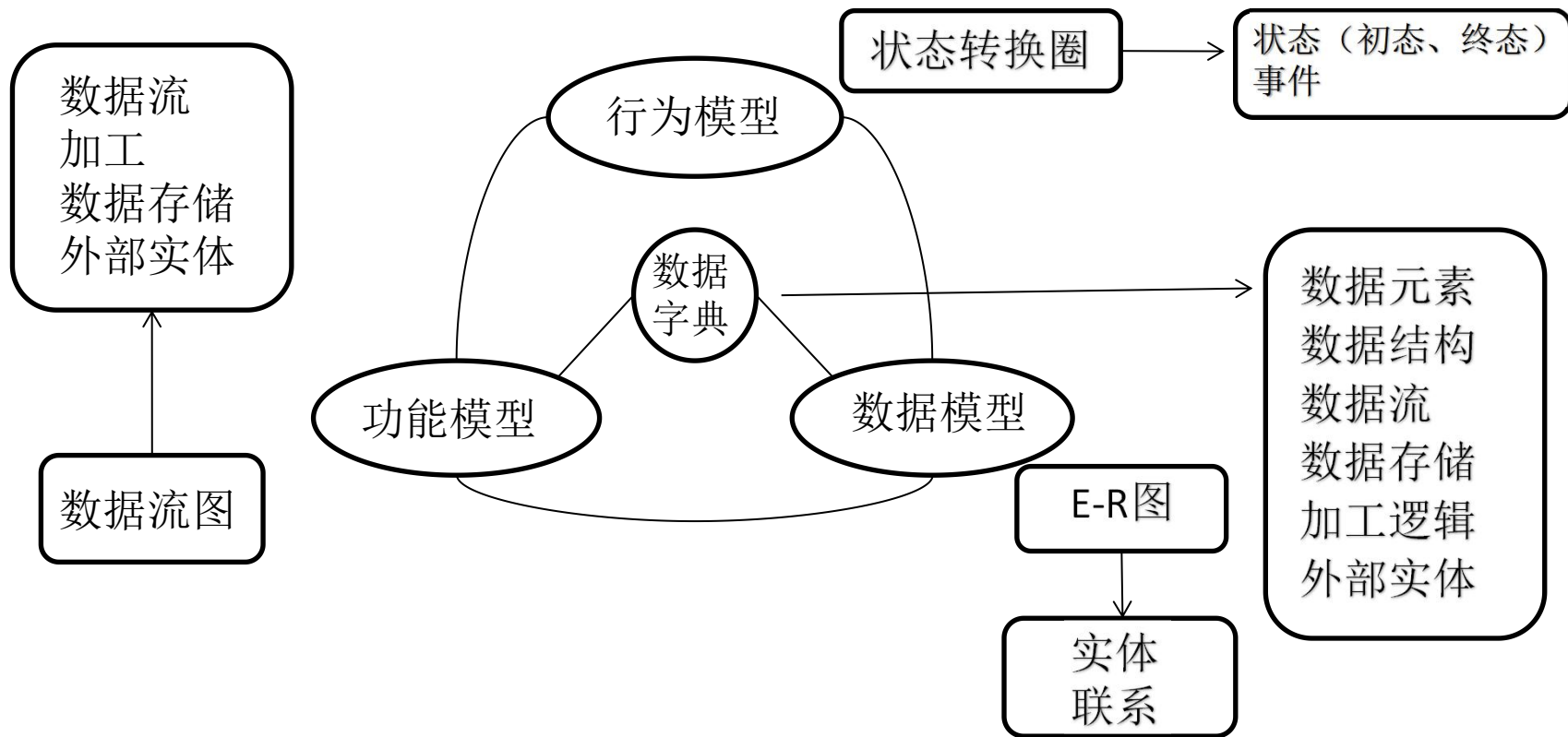
- 净室即无尘室、洁净室，也就是一个受控污染级别的环境
- 使用盒结构规约（或形式化方法）进行分析和设计建模，而且强调将正确性验证，而不是测试，作为发现和消除错误的主要机制
- 使用统计的测试来获取认证被交付的软件的可靠性所必需的的出错率信息

软件需求是指用户对系统在功能、行为、性能、设计约束等方面的期望

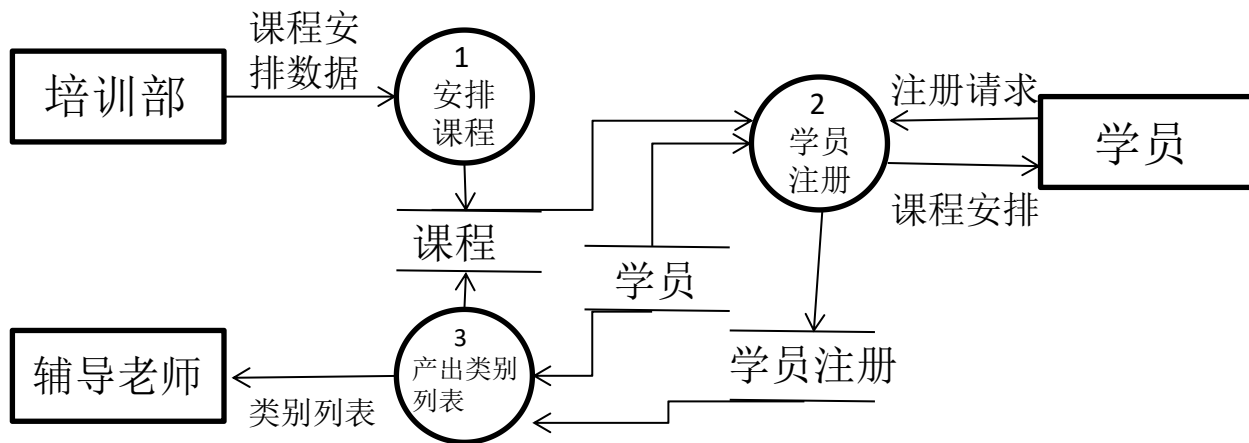
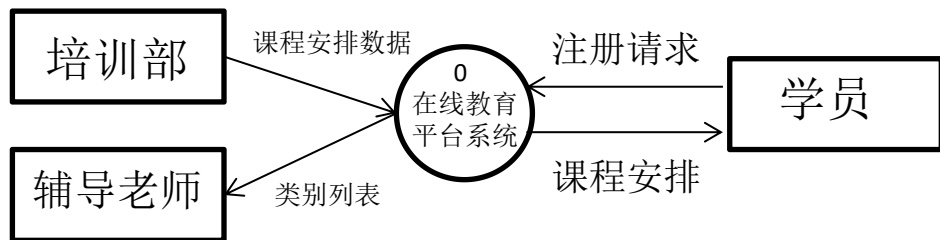
软件需求是指用户解决问题或达到目标所需的条件或能力，是系统或系统部件要满足合同、标准、规范或其他正式规定文档所需具有的条件或能力，以及反映这些条件或能力的文档说明。



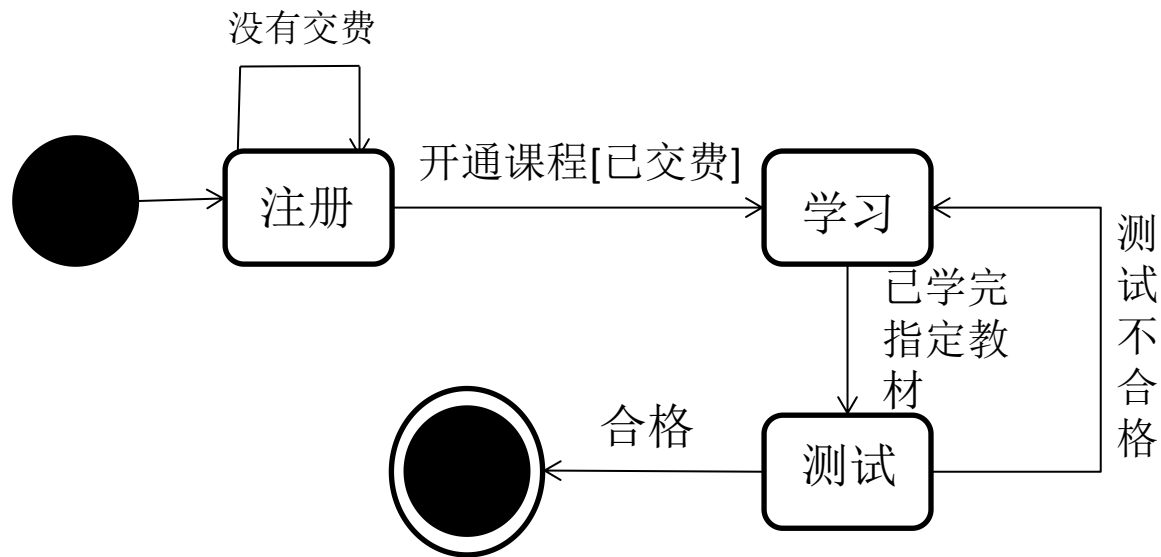


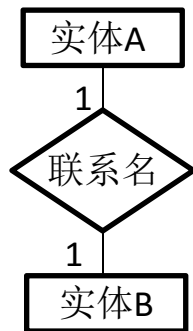




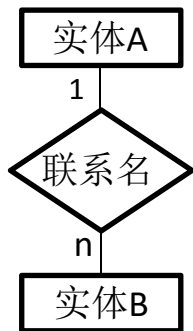


数据流  
加工  
数据存储  
外部实体

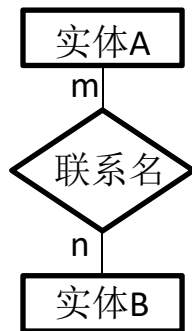




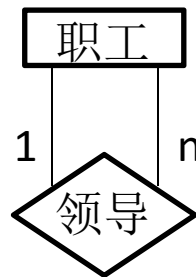
(a) 1:1的联系



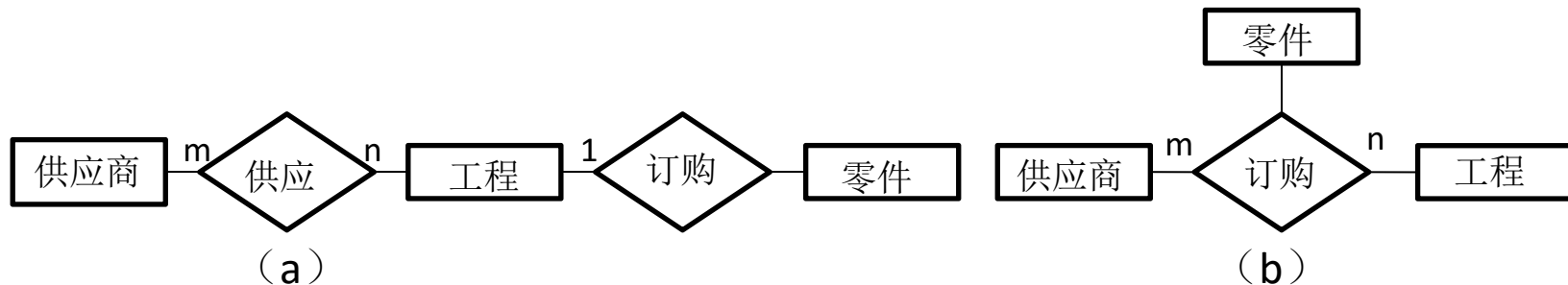
(b) 1:n的联系



(c) m:n的联系



(d) 同一实体集内一对多的联系



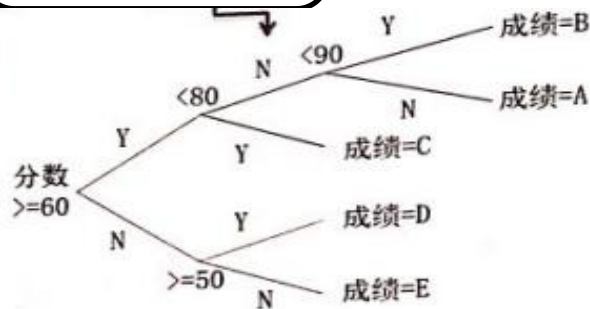
(a)

(b)

数据元素  
数据结构  
数据流  
数据存储  
加工逻辑  
外部实体

分数 ≥60	是		否		
分数	≥80		<80	≥30	小于50
分数	≥90	>90			
成绩	A	B	C	D	E

结构化语言  
判定表  
判定树



```

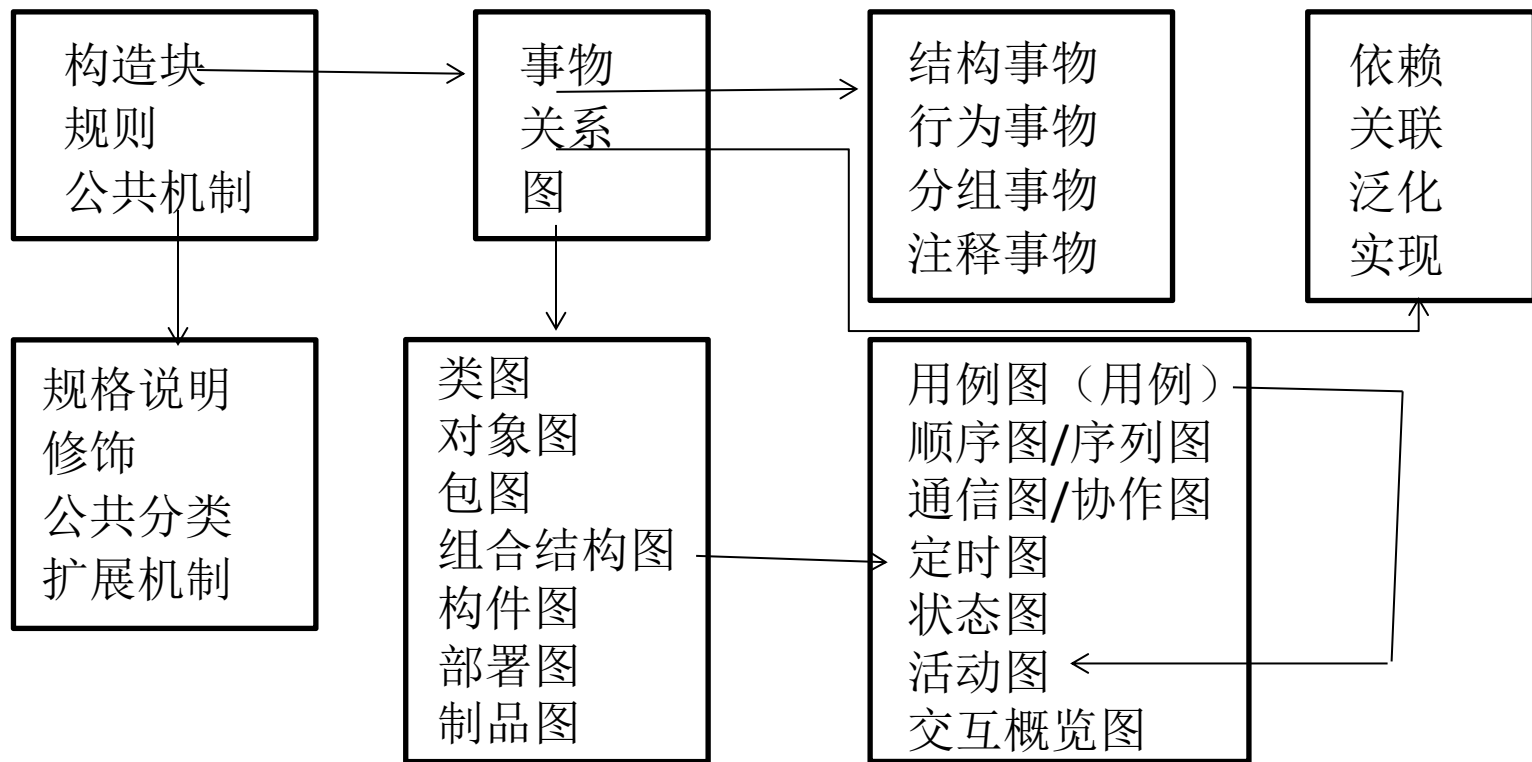
IF 分数 ≥ 60 Then
  IF 分数 < 80 Then
    成绩 = C
  ELSE
    IF 分数 < 90 Then
      成绩 = B
    Else
      成绩 = A
    EndIf
  EndIf
ELSE
  IF 分数 ≥ 50 Then
    成绩 = D
  ELSE
    成绩 = E
  EndIf
EndIf
  
```

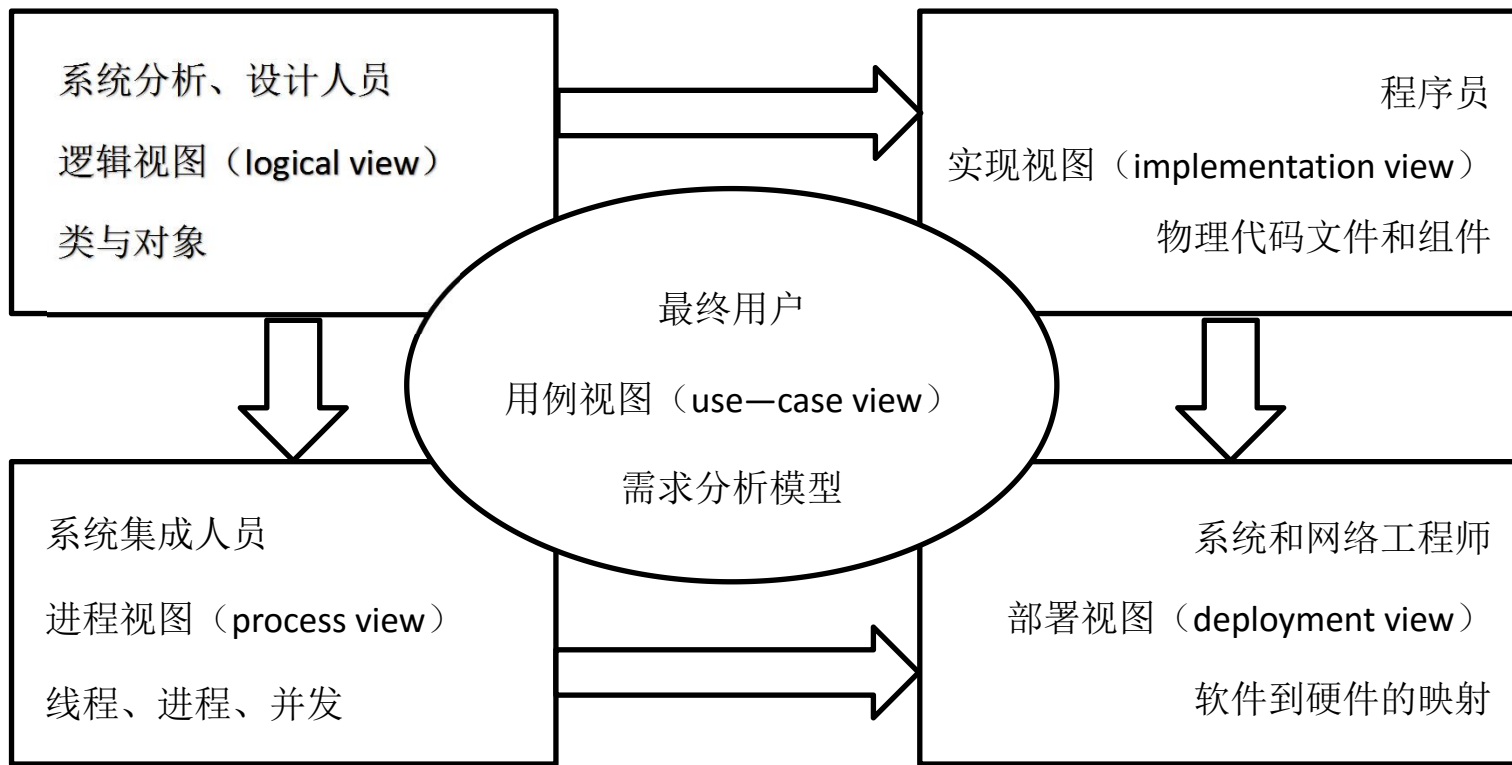
对象  
类（实体类、边界类、控制类）  
抽象  
封装  
继承与泛化  
多态  
接口  
消息  
组件  
模式和复用

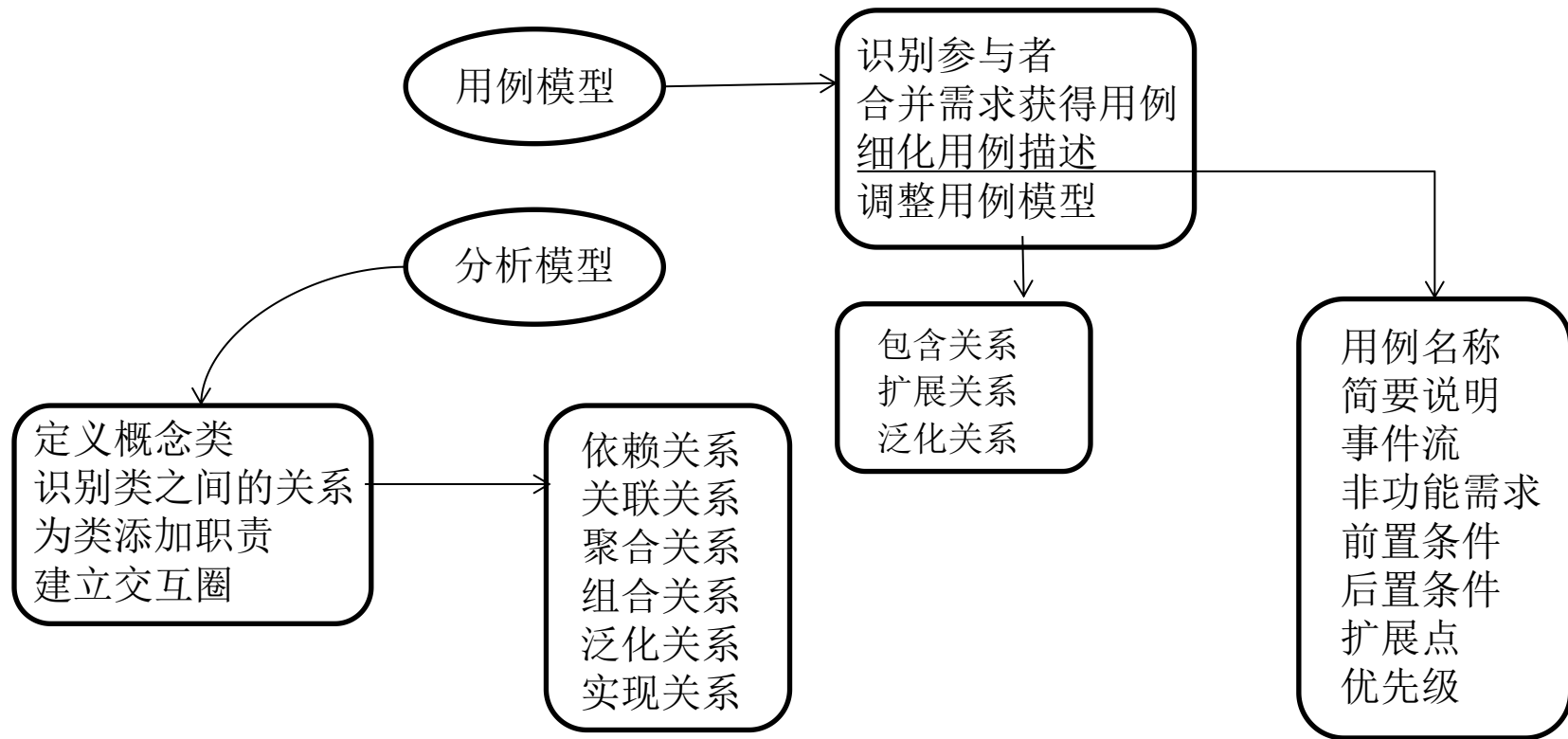
实体类映射需求中的每个实体，实体类保存需要存储在永久存储体中的信息,例如,在线教育平台系统可以提取出学员类和课程类,它们都属于实体类。

控制类是用于控制用例工作的类,一般是由动宾结构的短语(“动词+名词”或“名词+动词)转化来的名词,例如,用例"身份验证可以对应于一个控制类“身份验证器”,它提供了与身份验证相关的所有操作。

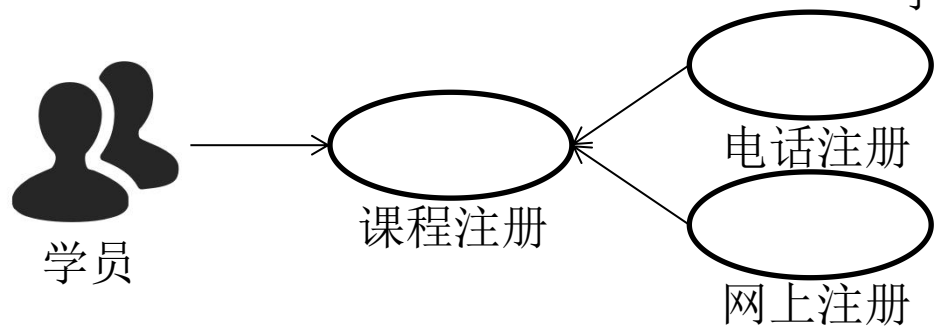
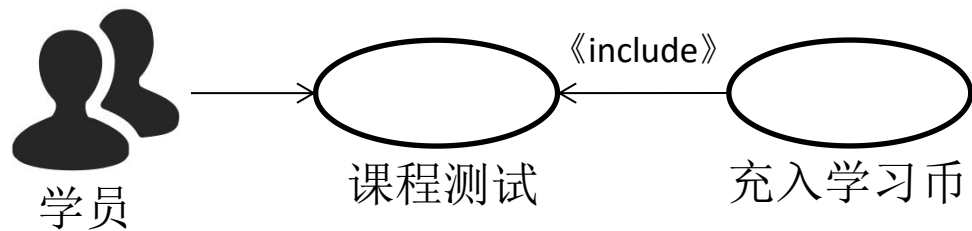
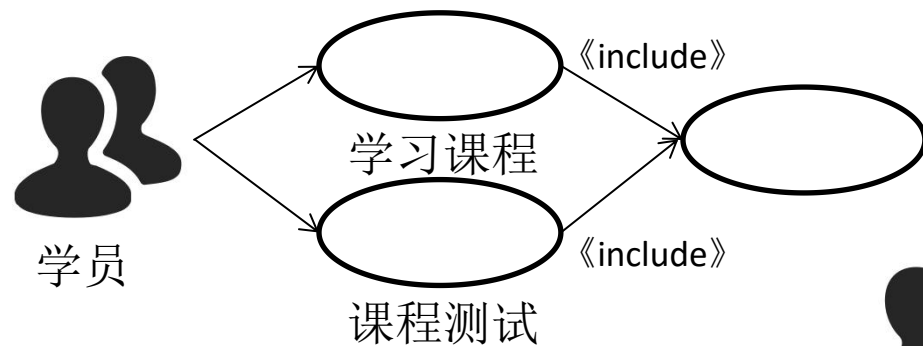
边界类用于封装在用例内、外流动的信息或数据流。边界类位于系统与外界的交接处,包括所有窗体、报表、打印机和扫描仪等硬件的接口,以及与其他系统的接口

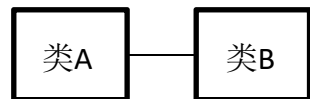




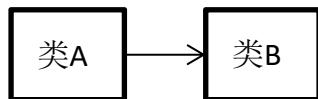




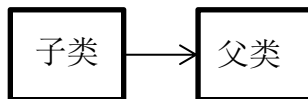




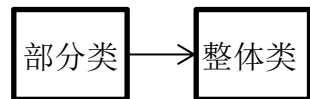
(a) 关联关系



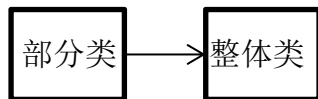
(b) 依赖关系



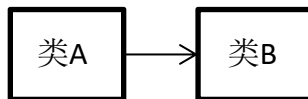
(c) 泛化关系



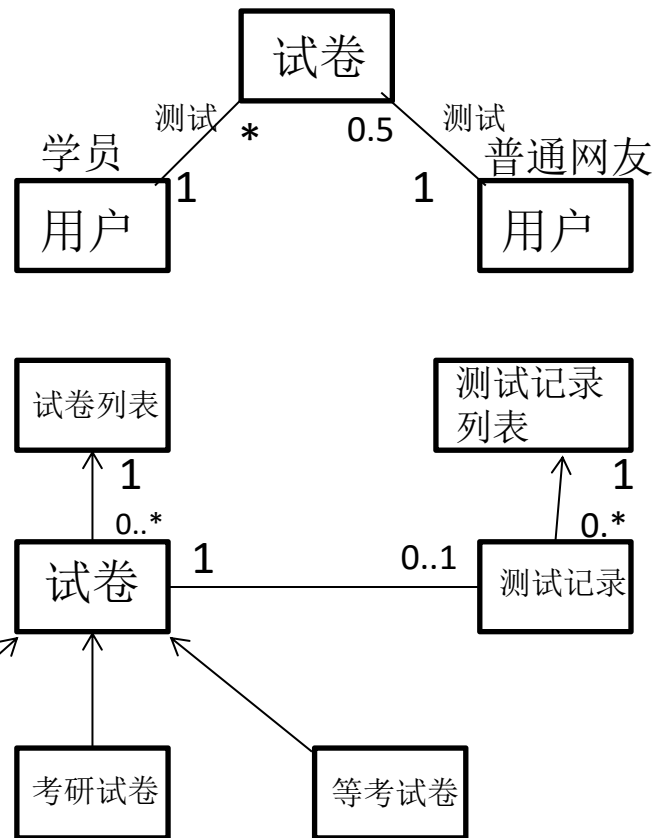
(d) 聚合关系



(e) 组合关系



(f) 实现关系



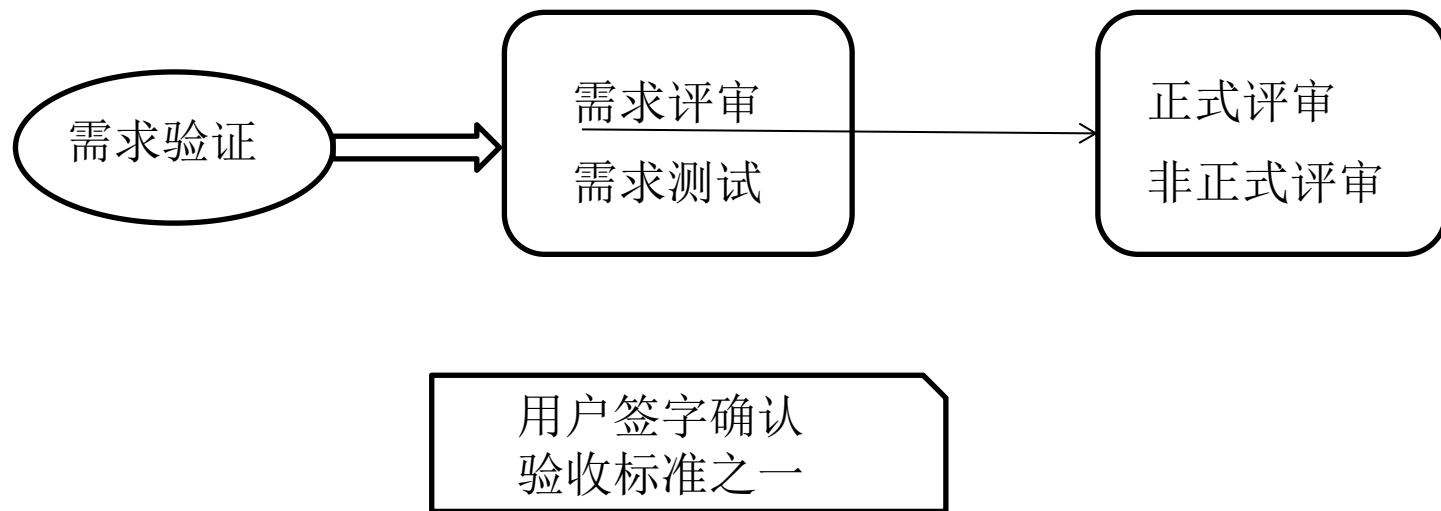
## 严格定义法

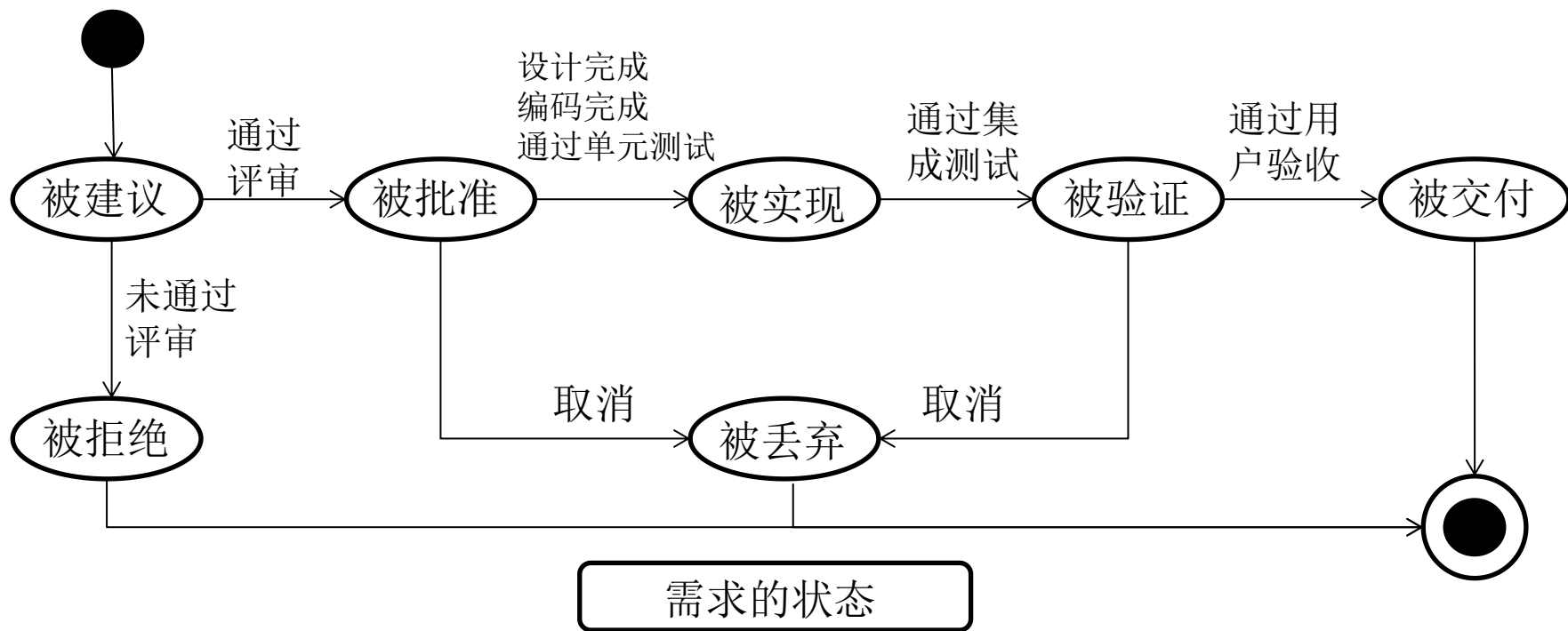
- 所有需求都能够被预先定义
- 开发人员与用户之间能够准确而清晰地交流
- 采用图形/文字可以充分体现最终系统

- 用结构和自然语言编写文本型文档
- 建立图形化模型
- 编写形式化规格说明

## 原型法

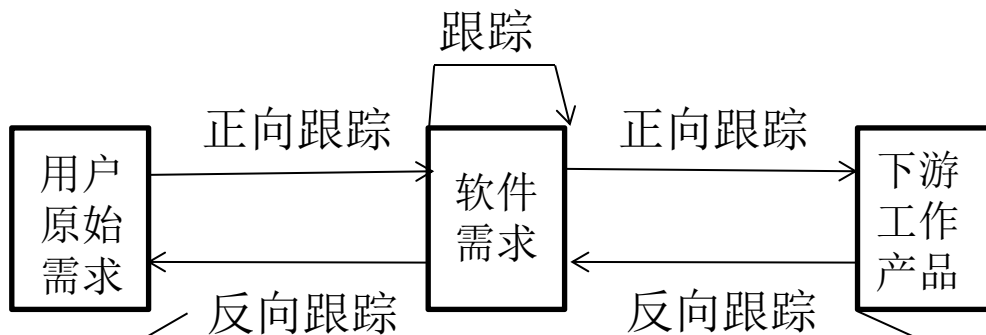
- 并非所有的需求都能在开发前被准确地说明
- 项目参加者之间通常都存在交流上的困难
- 需要实际的，可供用户参与的系统模型
- 有合适的系统开发环境
- 反复是完全需要和值得提倡的，需求一旦确定，就应遵从严格的方法





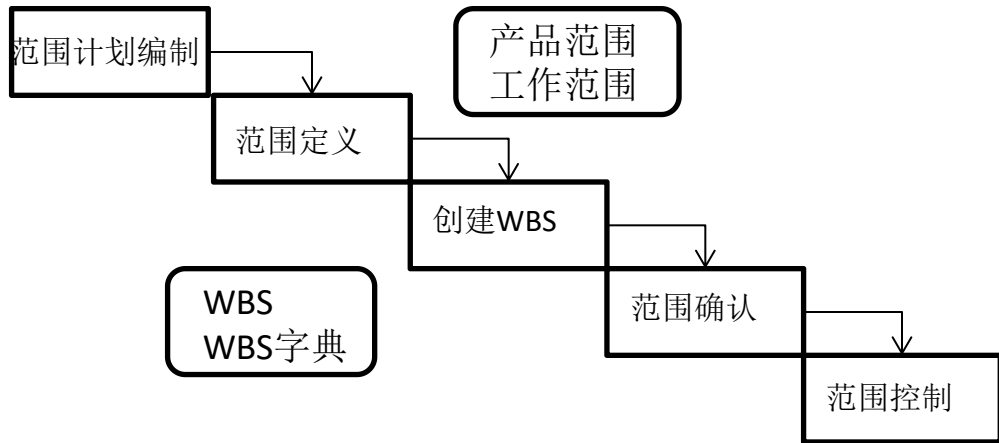
无足够用户参与  
忽略了用户分类  
用户需求的不断增加  
模棱两可的需求  
不必要的特性  
过于精简的SRS  
不准确的估算

带有风险的做法



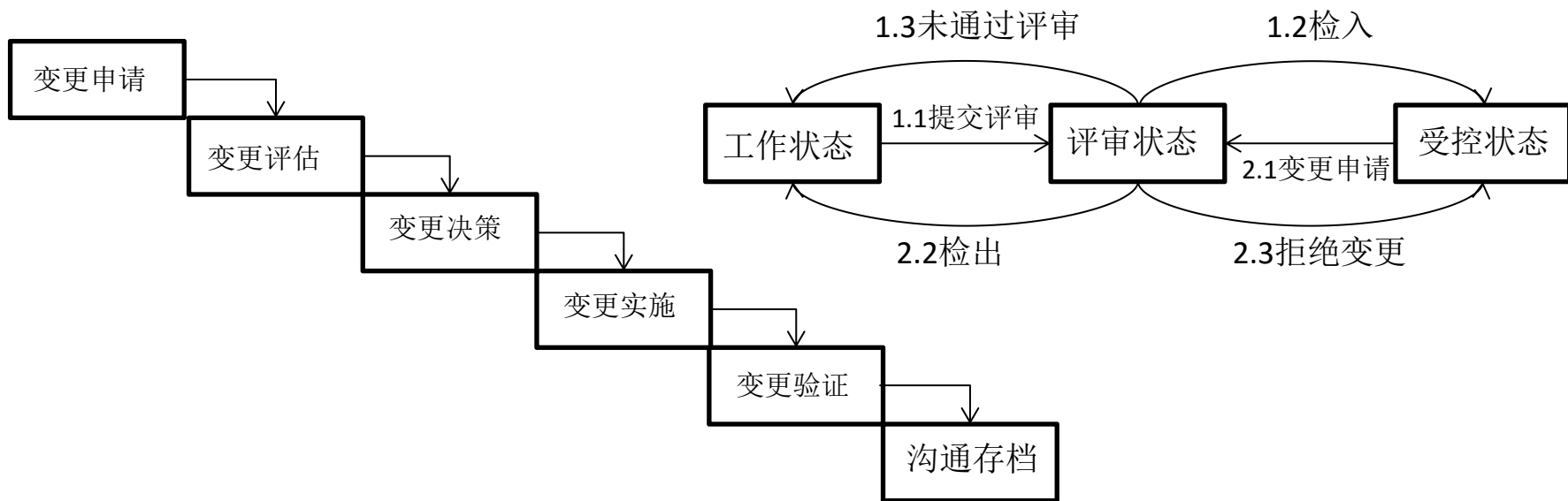
用例 原始需求	UC-1	UC-2	UC-3	.....	UC-n
FR-1					
FR-2					
.....					
FR-m					

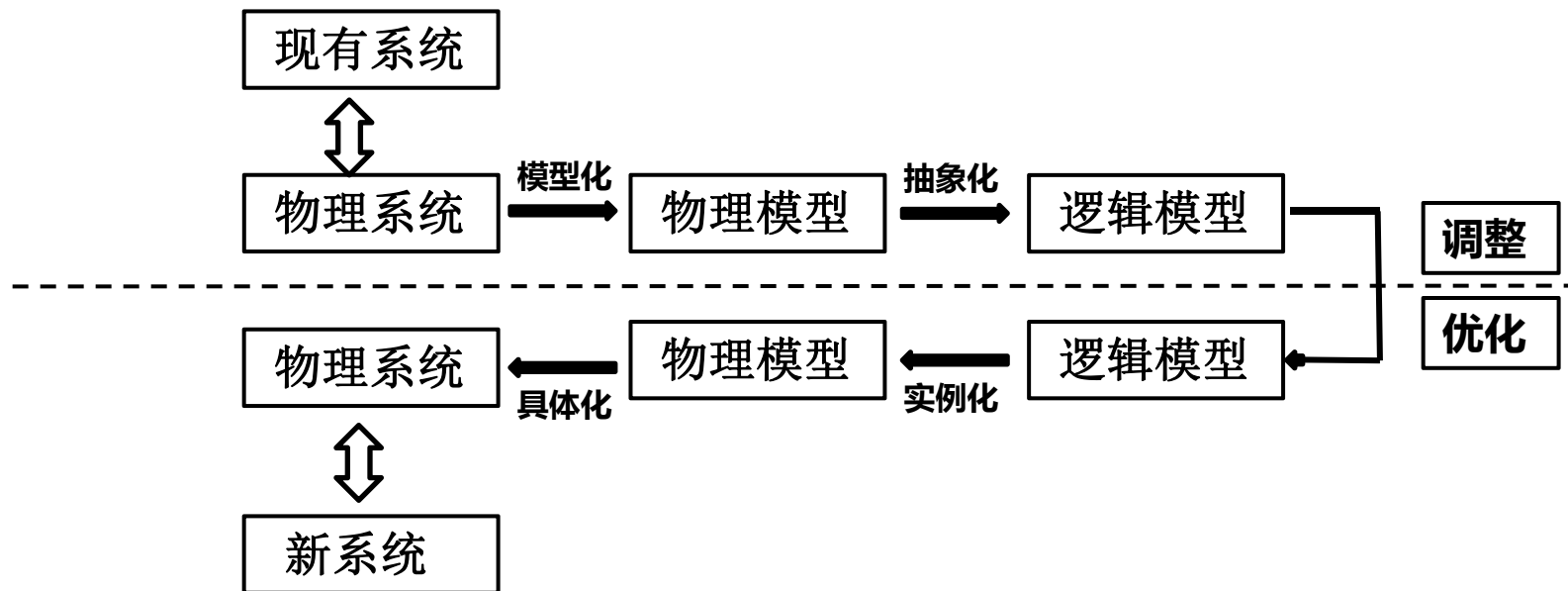
用例 元素	功能点	设计元素	代码模块	测试用例
UC-1				
UC-2				
.....				
UC-n				



层次名称	层次编号	层次描述	层次目的
决策层	1	总项目	工作授权和解除
管理层	2	项目	预算编制
	3	任务	进度计划编制
技术层	4	子任务	内部控制
	5	工作包	
操作层	6	努力水平	





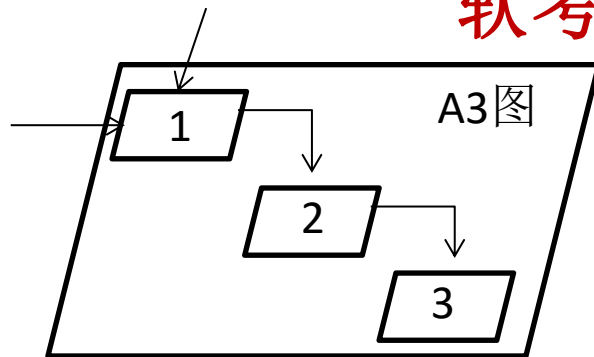
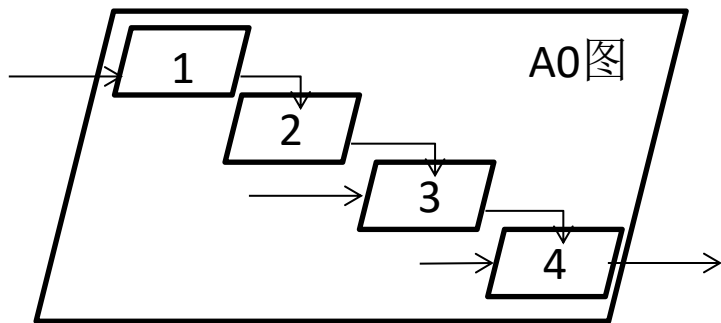
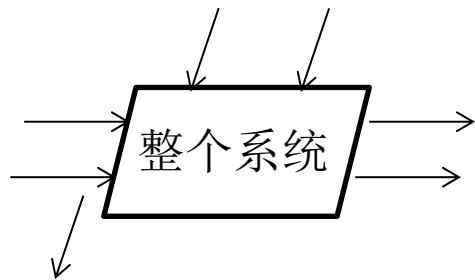


- 处理流程设计
- 人机界面设计
- 结构化设计
- 面向对象设计

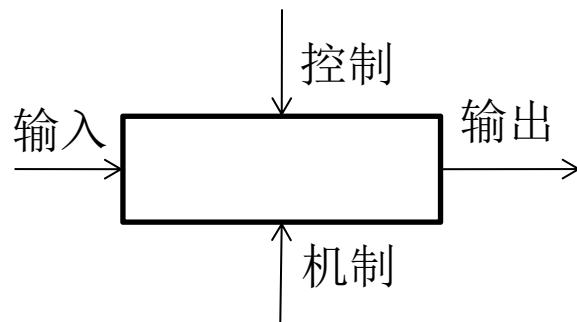
- 价值链分析法
- 客户关系分析法
- 供应链分析法
- 基于ERP的分析法
- 业务流程重组

- 标杆瞄准
- IDEF（集成定义方法）
- DEMO（组织动态本质建模法）
- Petri网
- 业务流程建模语言
- 基于服务的BPM

- 确定需要进行标杆研究的流程和影响流程成败的关键因素
- 确定瞄准目标的标杆企业、组织及其流程
- 通过走访、调研、会谈广告等采集数据，并进行分析
- 从众多标杆数据中，选定最佳改进标准
- 根据标杆指标，评估企业的既有流程，并确立改进目标



(a) 整个系统分解图



(b) 活动的表示方法

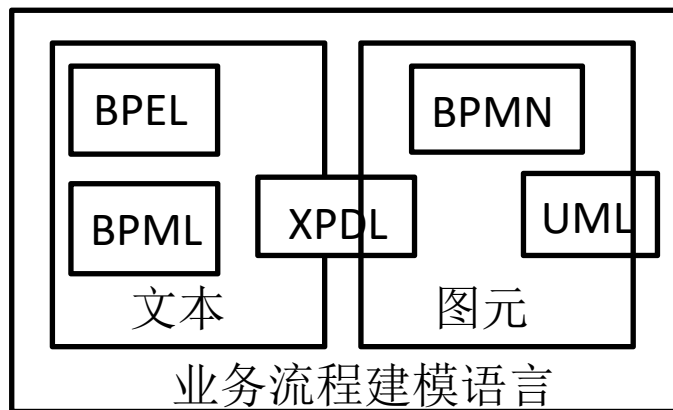
- 交互模型
- 业务流程模型
- 事务模型
- 行为模型
- 事实模型
- 互约束模型

- 描述企业事务各个阶段的角色
- 确定事务阶段之间的因果和条件关系
- 在流程表中描述因果和条件关系
- 检查所有事务阶段的角色（发起者或执行者）

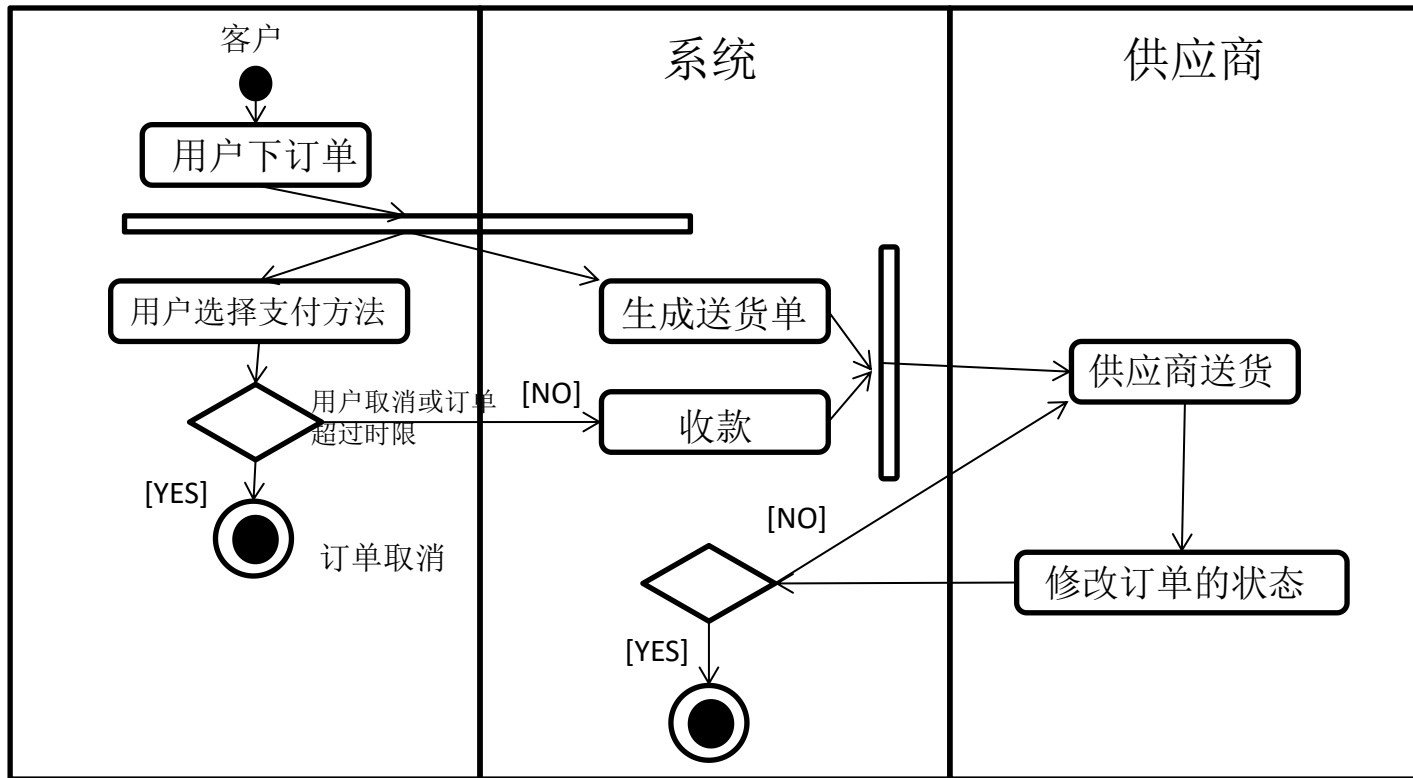
因果关系：A事务的执行促使B事务的开始

条件关系：A事务的完成形成B事物开始或完成的条件

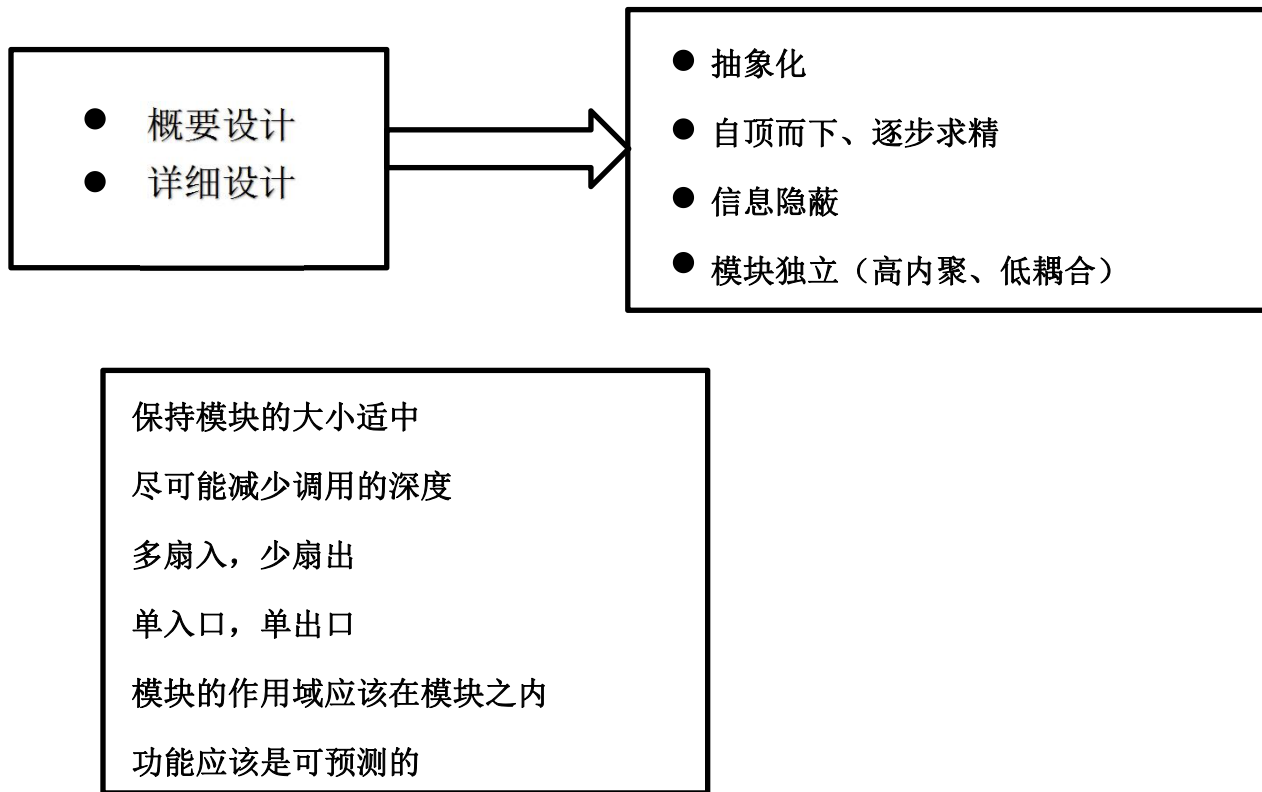




BPEL:业务流程执行语言  
BMPL:业务流程建模语言  
BPMN:业务流程建模标注  
XPDL:XML流程定义语言



- 置于用户控制之下
- 减少用户的记忆负担
- 保持界面的一致性



耦合类型	
非直接耦合	两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的
数据耦合	一组模块借助参数表传递简单数据
标记耦合	一组模块借助参数表传递记录信息（数据结构）
控制耦合	模块之间传递的信息中包含用于控制模块内部逻辑的信息
外部耦合	一组模块都访问同一全局简单变量，而且不是通过参数表传递该全局变量的信息
公共耦合	多个模块都访问同一个公共数据环境
内容耦合	一个模块直接访问另一个模块的内部数据；一个模块不通过正常入口转到另一个模块的内部；两个模块有一部分程序代码重叠；一个模块有多个入口



## 分析模型

用例模型

分析模型  
(领域模型)

## 设计师

设计用例实现方案

设计技术支撑实施

设计用户界面

细化设计模型

## 设计模型

架构图（用包图表示）

用例实现图（用交互图表示）

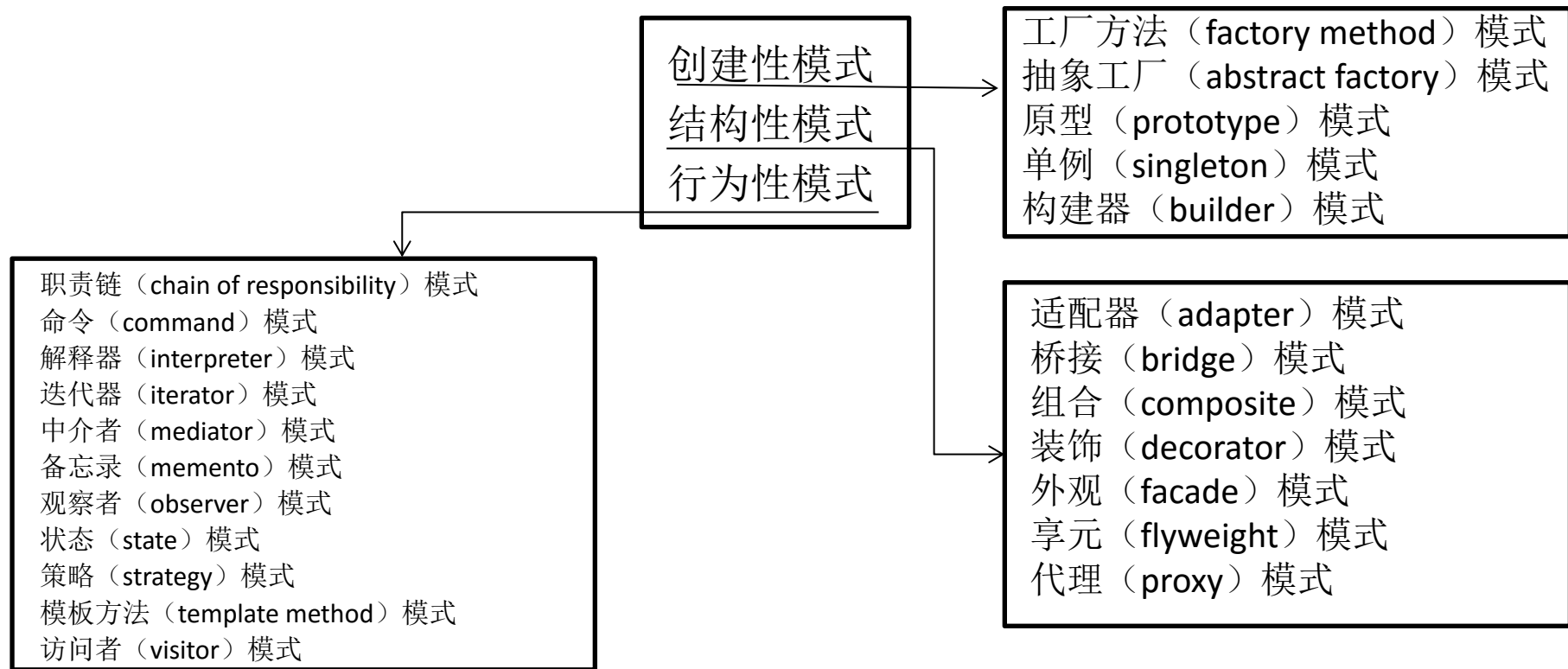
类图（完整、精确）

其他（状态图、活动图等）

- 单一职责原则：设计目的单一的类
- 开放-封闭原则：对扩展开放，对修改封闭
- 李氏（**Liskov**）替换原则：子类可以替换父类
- 依赖倒置原则：要依赖于抽象，而不是具体实现；针对接口编程，不要针对实现编程
- 接口隔离原则：使用多个专门的接口比使用单一的总接口要好
- 组合重用原则：要尽量使用组合，而不是继承关系达到重用目的
- 迪米特（**Demeter**）原则（最少知识法则）：一个对象应当对其他对象有尽可能少的了解

- 架构模式：软件设计中的高层决策，例如C/S结构就属于架构模式，架构模式反映了开发软件系统过程中所作的基本设计决策
- 设计模式：主要关注软件系统的设计,与具体的实现语言无关
- 惯用法:是最低层的模式,关注软件系统的设计与实现,实现时通过某种特定的程序设计语言来描述构件与构件之间的关系。每种编程语言都有它自己特定的模式，即语言的惯用法。例如引用-计数就是C++语言的一种用惯用法





设计模式名称	简要说明
Abstract Factory 抽象工厂模式	提供一个接口，可以创建一系列相关或相互依赖的对象，而无需指定它们具体的类
Builder 构建器模式	将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示
Factory Method 工厂方法模式	定义一个创建对象的接口，但由子类决定需要实例化哪一个类。工厂方法使得子类实例化的过程推迟
Prototype 原型模式	用原型实例指定创建对象的类型，并且通过拷贝这个原型来创建新的对象
Singleton 单例模式	保证一个类只有一个实例，并提供一个访问它的全局访问点

设计模式名称	简要说明	速记关键字
Adapter 适配器模式	将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作	转换接口
Bridge 桥接模式	将类的抽象部分和它的实现部分分离开来，使它们可以独立地变化	继承树拆分
Composite 组合模式	将对象组合成树型结构以表示“整体-部分”的层次结构，使得用户对单个对象和组合对象的使用具有一致性	树形目录结构
Decorator 装饰模式	动态地给一个对象添加一些额外的职责。它提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活	附加职责
Facade 外观模式	定义一个高层接口，为子系统的一组接口提供一个一致的外观，从而简化了该子系统的使用	对外统一接口
Flyweight 享元模式	提供支持大量细粒度对象共享的有效方法	
Proxy 代理模式	为其他对象提供一种代理以控制这个对象的访问	

设计模式名称	简要说明	速记关键字
Chain of Responsibility 职责链模式	通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合，将接受对象链接起来，在链中传递请求，直到有一个对象处理这个请求	传递职责
Command 命令模式	将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，将请求排队或记录请求日志，支持可撤销的操作	日志记录，可撤销
Interpreter 解释器模式	给定一种语言，定义它的文法表示，并定义一个解释器该解释器用来根据文法表示来解释语言中的句子	
Iterator 迭代器模式	提供一种方法来顺序访问一个聚合对象中的各个元素，而不需要暴露该对象的内部表示	
Mediator 中介者模式	用一个中介对象来封装一系列的对象交互。它使各对象不需要显式地相互调用，从而达到低耦合，还可以独立地改变对象间的交互	不直接引用

设计模式名称	简要说明	速记关键字
Memento 备忘录模式	在不破坏封装性的的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，从而可以在以后将该对象恢复到原先保存的状态	
Observer 观察者模式	定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新	
State 状态模式	允许一个对象在其内部状态改变时改变它的行为	状态变成类
Strategy 策略模式	定义一系列算法，把它们一个个封装起来，并且使它们之间课互相替换，从而让算法可以独立于使用它的用户而变化	多方案切换
Template Method 模块方法模式	定义一个操作中的算法骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可重新定义算法的某些特定步骤	
Visitor 访问者模式	表示一个作用于某对象结构中的各元素的操作，使得在不改变各元素的类的前提下定义作用于这些元素的新操作	

## 1.系统设计

(1) 的选择是开发一个软件系统时的基本设计决策;(2) 是最低层的模式,关注软件系统的设计与实现,描述了如何实现构件及构件之间的关系。引用一计数是C++管理动态资源时常用的一种 (3)

(1)A.架构模式

C.设计模式

(2)A.架构模式

C.设计模式

(3)A.架构模式

C.设计模式

B.惯用法

D.分析模式

B.惯用法

D.分析模式

B.惯用法

D.分析模式

测试评审方法

验证与确认

测试自动化

软件调试

测试设计和管理方法



尽早、不断的进行测试

程序员避免测试自己设计的程序

既要选择有效、合理的数据，也要选择无效、不合理的数据

修改后应进行回归测试

尚未发现的错误数量与该程序已发现错误数成正比

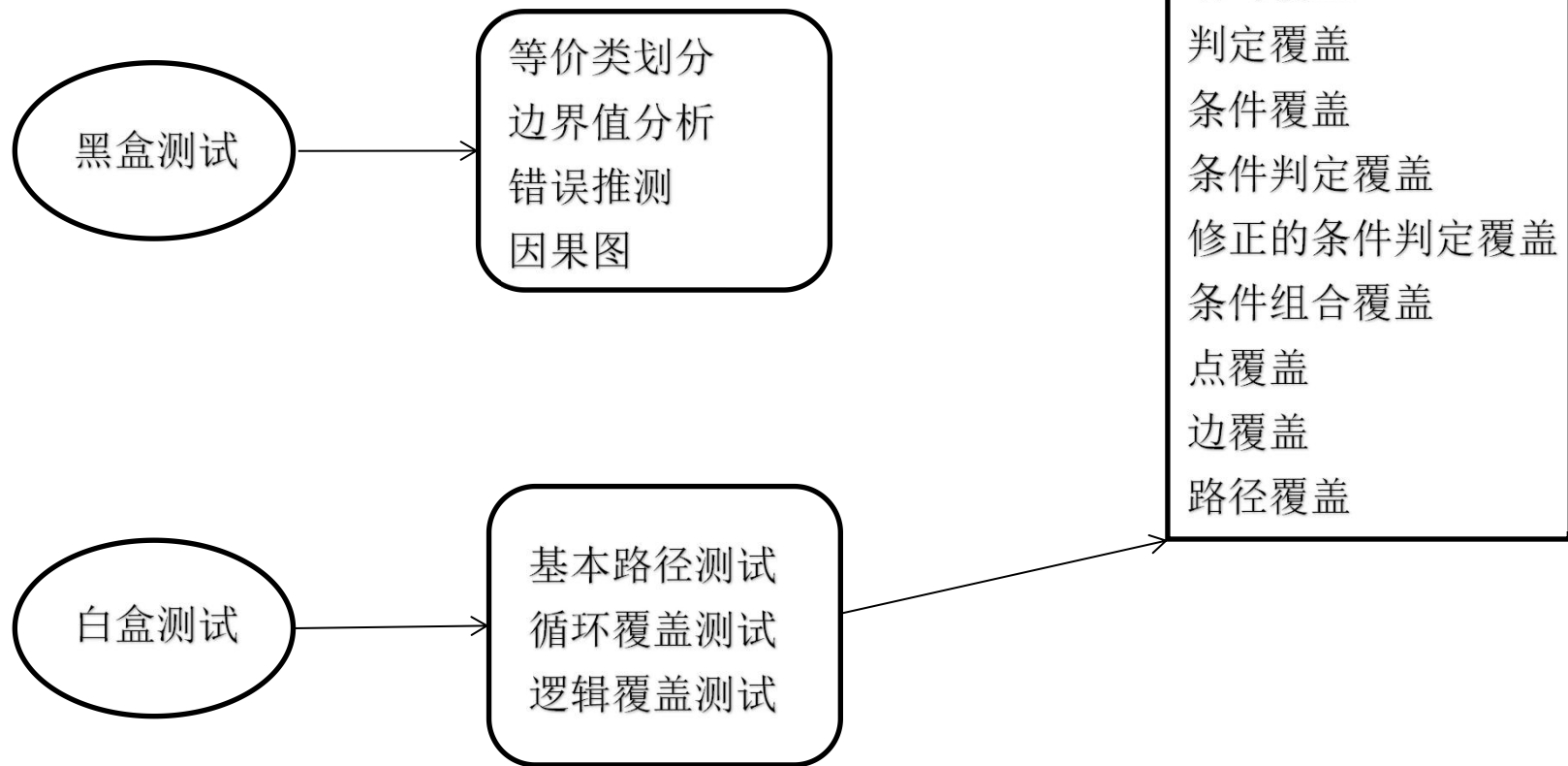
动态测试

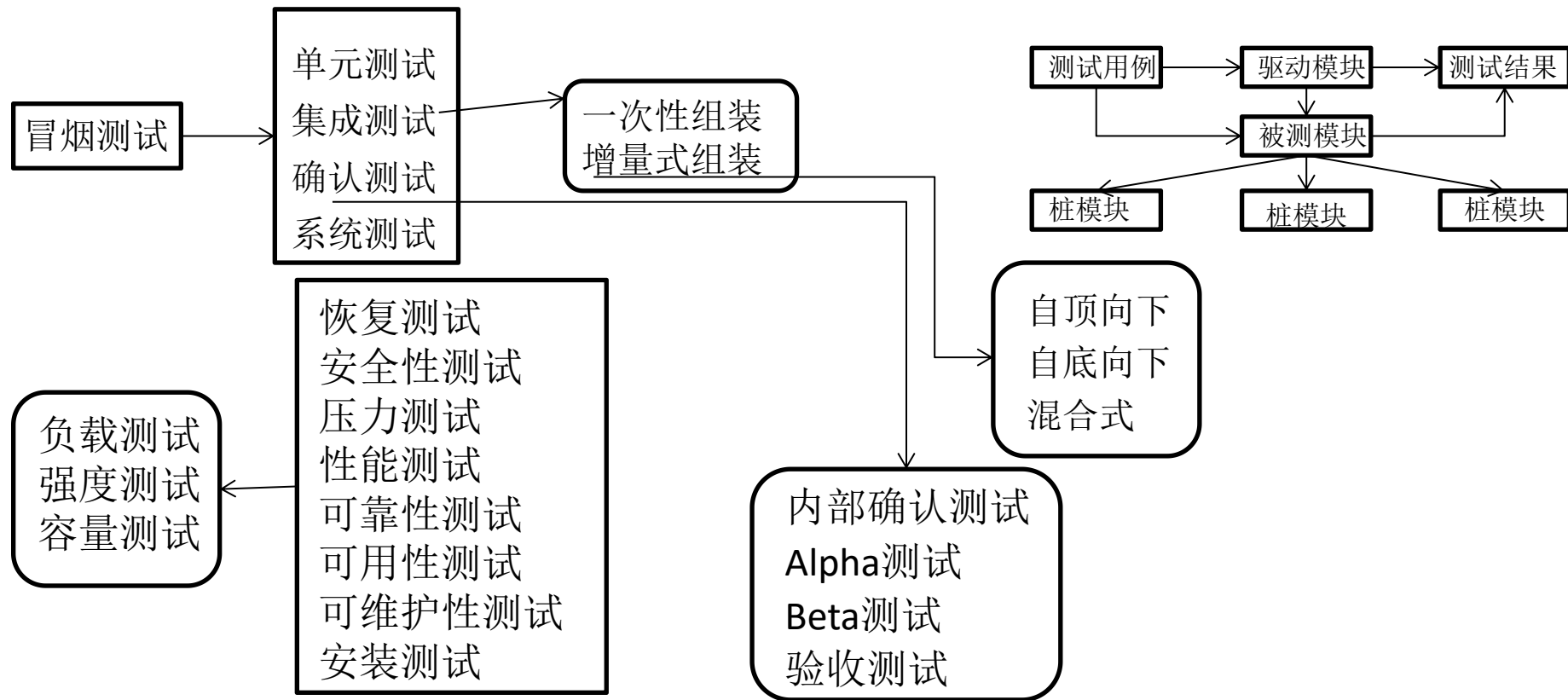
黑盒测试法  
白盒测试法  
灰盒测试法

静态测试

桌前检查  
代码走查  
代码审查







- 算法层（单元测试）：包括等价类划分测试、组合功能测试（基于判定表的测试）、递归函数测试和多态消息测试
- 类层（模块测试）：包括不变式边界测试、模态类测试
- 模板层/类树层（集成测试）：包括多态服务测试和展平测试
- 系统层（系统测试）

- √ 提高测试执行的速度
- √ 提高运行效率
- √ 保证测试结果的准确性
- √ 连续运行测试脚本
- √ 模拟现实环境下受约束的情况

- × 所以测试活动都可以自动完成
- × 减少人力成本
- × 可以免费获得
- × 降低测试工作量

### 软件调试方法

- 蛮力法：主要思想是“通过计算机找错”，低效，耗时
- 回溯法：从出错处人工沿控制流程往回追踪，直至发现出错的根源。复杂程序由于回溯路径多，难以实施
- 原因排除法：主要思想是演绎和归纳，用二分法实现

### 软件调试与测试的区别

- 测试的目的是找出存在的错误，而调试的目的是定位错误并修改以修正错误
- 调试是测试之后的活动,测试和调试在目标、方法和思路上都不同
- 测试从一个已知的条件开始,使用预先定义的过程,有预知的结果;调试从一个未知的条件开始,结束的过程不可预计
- 测试过程可以事先设计,进度可以事先确定;调试不能描述过程或持续时间

- 验证是指在软件开发周期中的一个给定阶段的产品是否达到在上一阶段确立的需求过程
- 确认是指在软件开发过程结束时对软件进行评价以确定它是否和软件需求相一致的过程
- 测试是指通过执行程序来有意识地发现程序中的设计错误和编码错误的过程。测试是验证和确认的手段之一

- 质量保证一般是每隔一定时间（例如，每个阶段末）进行的，主要通过系统的质量审计和过程分析来保证项目的质量
- 质量控制是实时监控项目的具体结果,以判断它们是否符合相关质量标准,制订有效方案,以消除产生质量问题的原因
- 一定时间内质量控制的结果也是质量保证的质量审计对象.质量保证的成果又可以指导下一阶段的质量工作,包括质量控制和质量改进

阶段式



组织能力成熟度

成熟度等级	过程域
已管理级	需求管理、项目计划、配置管理、项目监督与控制、供应商合同管理、度量和分析、过程和产品质量保证
已定义级	需求开发、技术解决方案。产品集成、验证、确认、组织级过程焦点、组织级过程定义、组织级培训、集成项目管理、风险管理、集成化的团队、决策分析和解决方案、组织级集成环境
定量管理级	组织级过程性能、定量项目管理
优化级	组织级改革与实施、因果分析和解决方案





连续式



软件过程能力

连续式分组

过程域

过程管理

组织级过程焦点、组织级过程定义、组织级培训、组织级过程性能。组织级改革与实施

项目管理

项目计划、项目监督与控制、供应商合同管理、集成项目管理、风险管理、集成化的团队、定量项目管理

工程

需求管理、需求开发、技术解决方案、产品集成、验证、确认

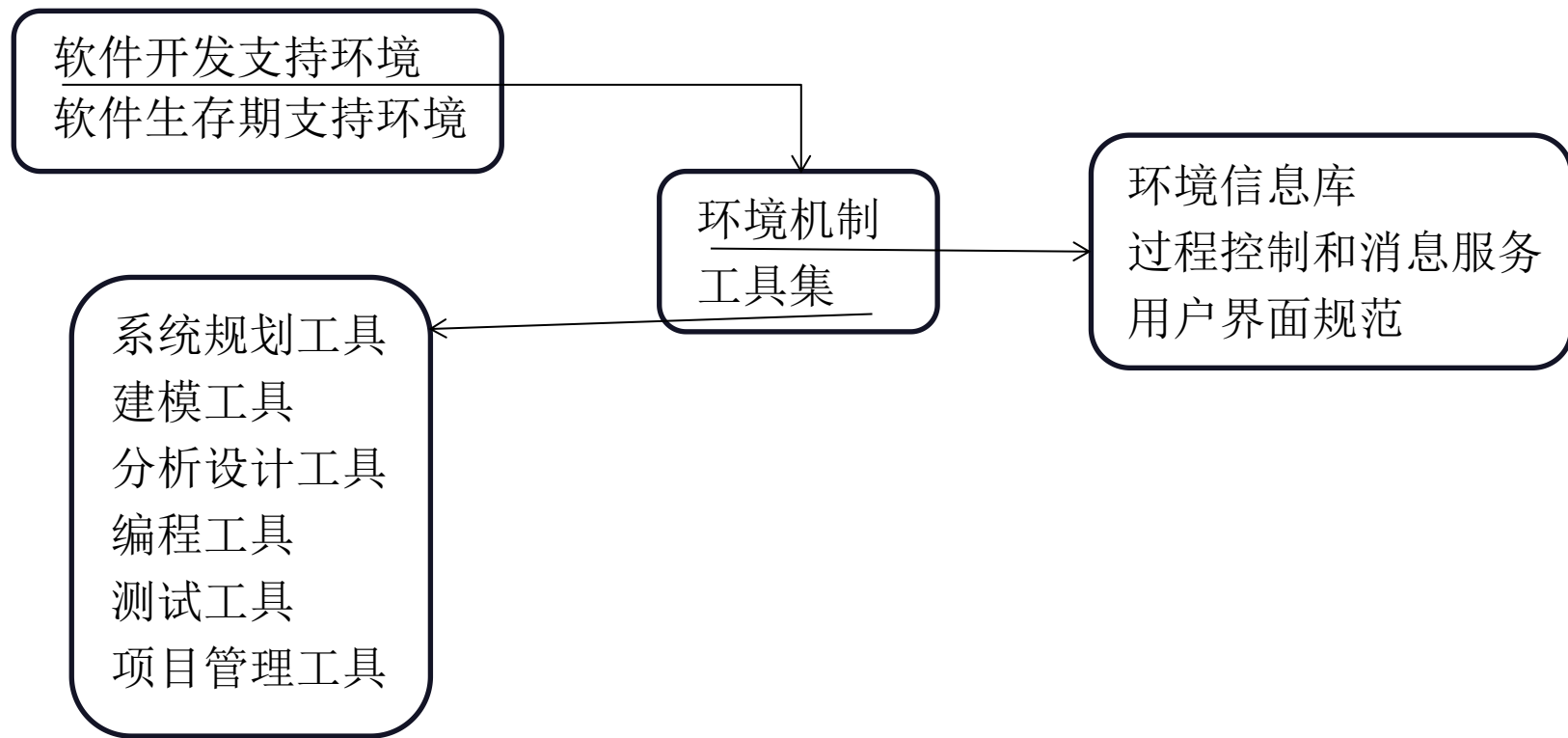
支持

配置管理、度量和分析、过程和产品质量保证、决策分析和解决方案、组织级集成环境、因果分析和解决方案



- IEEE对配置项的定义为硬件、软件或二者兼有的集合,为配置管理指定的,在配置管理过程中作为一个单独的实体对待,可作为配置项管理的有:外部交付的软件产品和数据、指定的内部软件工作产品和数据、指定的用于创建或支持软件产品的支持工具、供方/供应商提供的软件和客户提供的设备/软件。
- 典型配置项包括项目计划书、需求文档、设计文档、源代码、可执行代码、测试用例运行件所需的各种数据,它们经评审和检查通过后进入软件配置管理

- 开发库（动态库、程序员库、工作库；动态系统、开发者系统、开发系统、工作空间）
- 受控库（主库、系统库；主系统、受控系统）
- 产品库（备份库、静态库、软件仓库；静态系统）

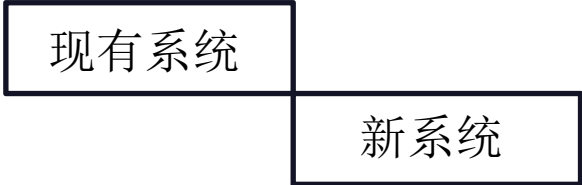


系统转换计划

软件维护

系统审计

系统评价

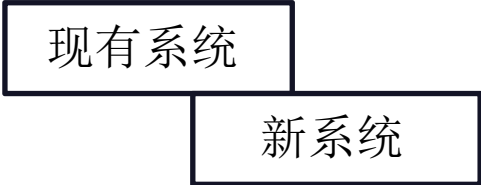


Existing System

The diagram shows two overlapping boxes. The top-left box is labeled '现有系统' (Existing System). The bottom-right box is labeled '新系统' (New System). The boxes overlap in the middle, representing a direct transition where the new system replaces the old one.

New System

直接转换策略

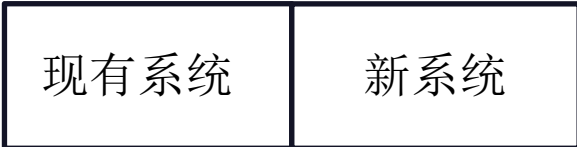


Existing System

The diagram shows two separate, side-by-side boxes. The top-left box is labeled '现有系统' (Existing System). The top-right box is labeled '新系统' (New System). This represents a parallel conversion strategy where both systems run simultaneously.

New System

并行转换策略

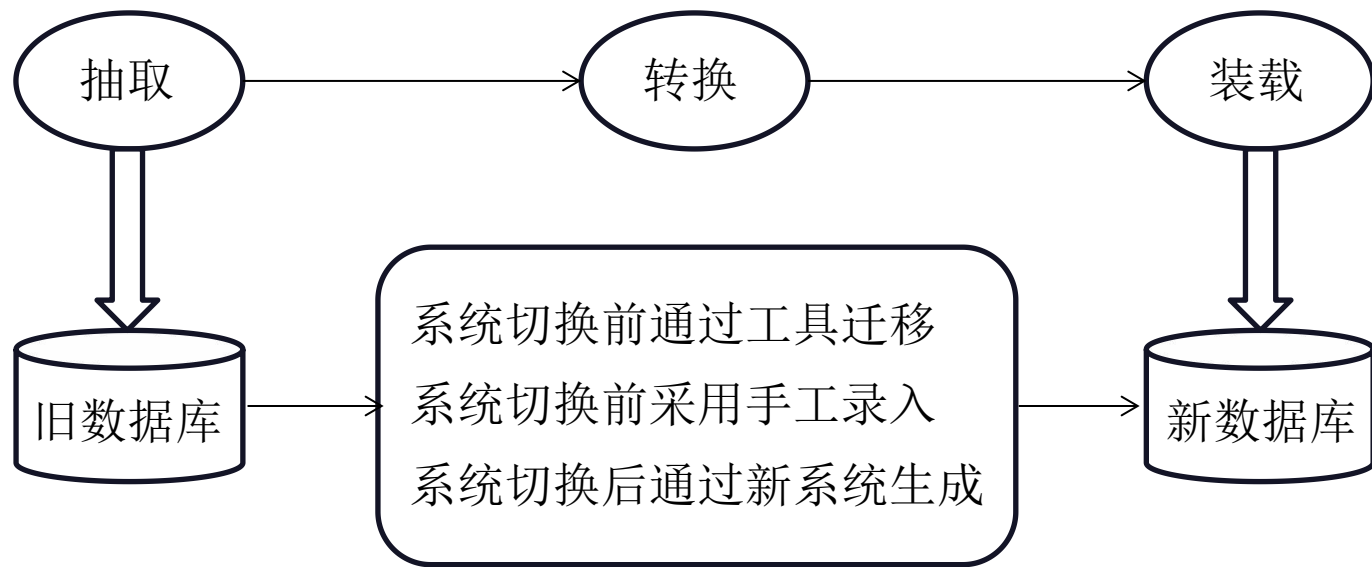


Existing System

The diagram shows a single large rectangle divided into two equal horizontal sections. The top section is labeled '现有系统' (Existing System) and the bottom section is labeled '新系统' (New System). This represents a phased conversion strategy where the new system is introduced in stages.

New System

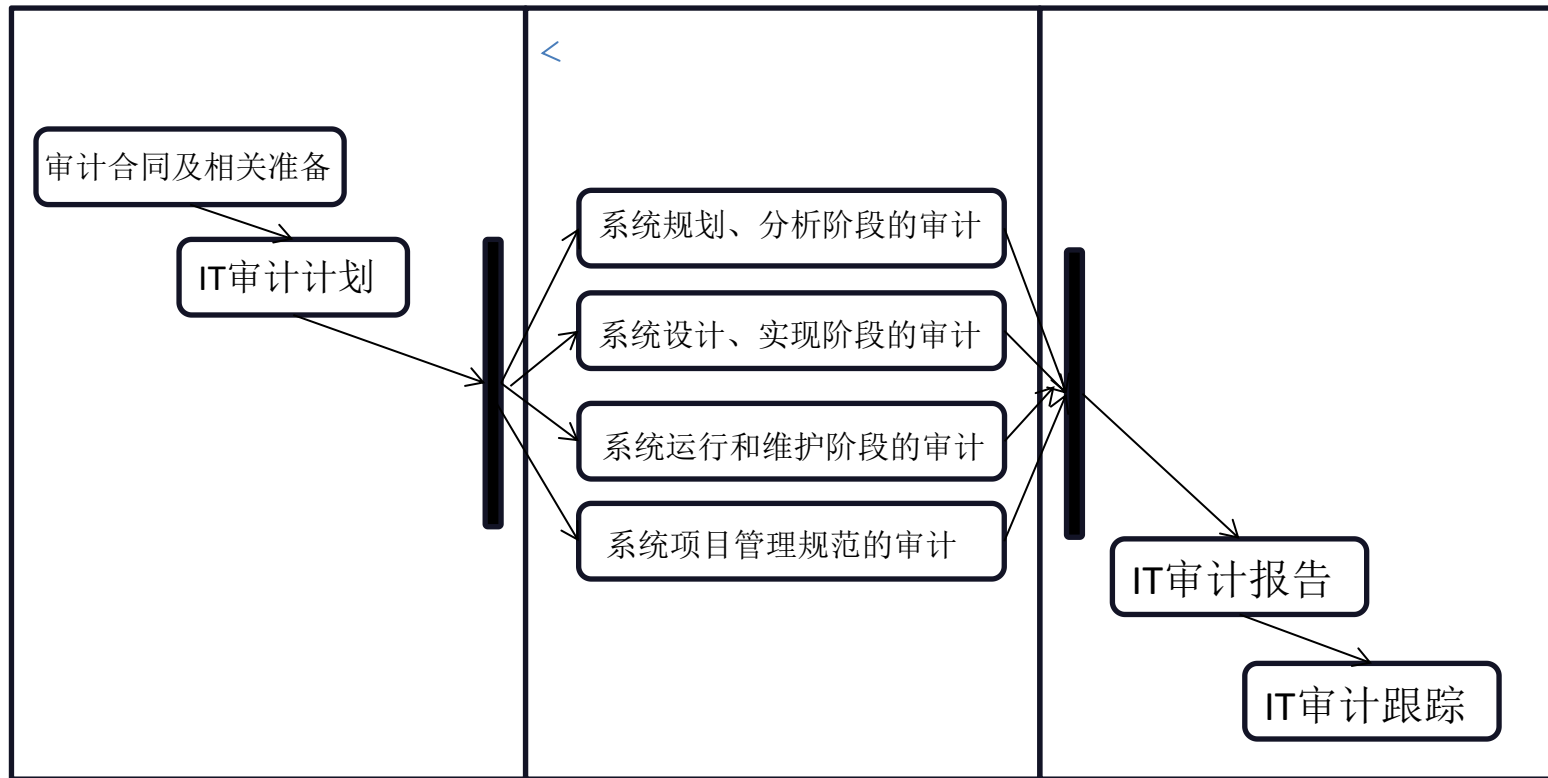
分段转换策略



软件维护是生命周期的个完整部分,可以将软件维护定义为需要提供软件支持的全部活动,这些活动包括在交付前完成的活动,以及交付后完成的活动。交付前完成的活动包括交付后运行的计划和维护计划等;交付后的活动包括软件更改,培训、帮助资料等

可	易分析性
维	易改变性
护	稳定性
性	易测试性

维	改正性维护 (25%)
护	适应性维护 (20%)
类	完善性维护 (50%)
型	预防性维护 (5%)





系统性能评价  
系统效益评价  
系统建设评价

- 确定评价对象，下达评价通知书，组织成立评价工作组合专家咨询组
- 拟定评价工作方案，收集基础资料
- 评价工作组实施评价，征求专家意见和反馈给企业，撰写评价报告
- 评价工作组将评价报告报送专家咨询组复核，向委托人送达评价报告和选择公布评价结果建立评价项目档案



DESIGNER:王川林  
软件工程



# THANK YOU