

软件工程概论复习课



2024年春

宋娟



关于考试

- I. 简答题 (**5*8分=40分**)
- II. 分析题 (**3*20分=60分**)
- III. 平时成绩 **30%** 考勤和大作业
- IV. 考试成绩**70%**

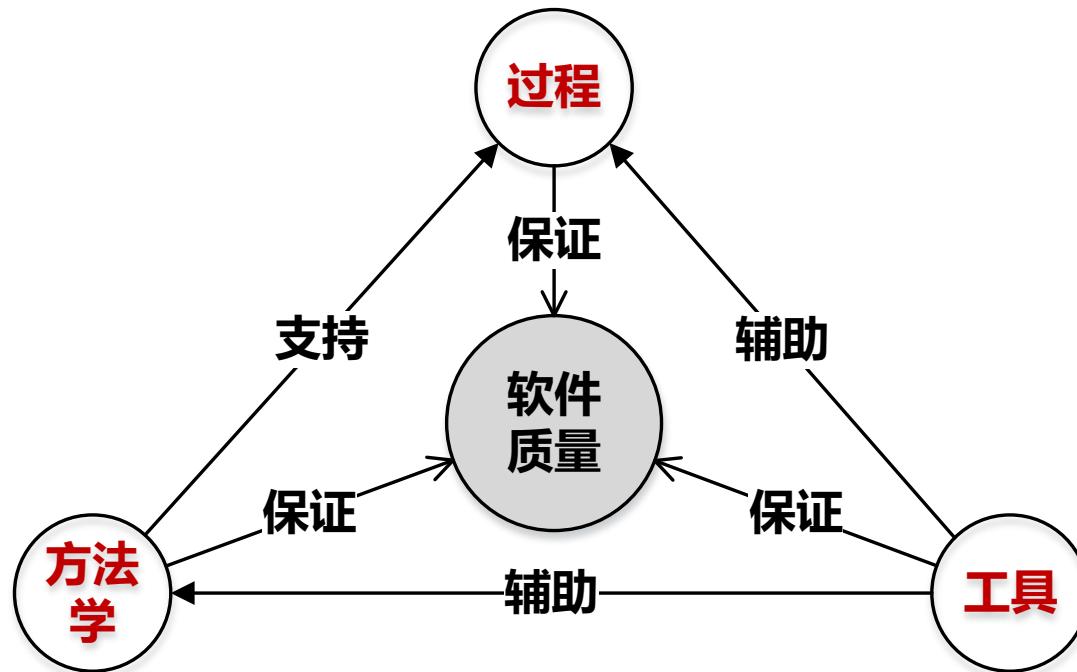


何为软件工程

■ 何为软件工程

- 软件危机
- 将**系统的、规范的、可量化**的方法应用于软件的开发、运行和维护的过程；以及上述方法的研究 – [IEEE 93]

■ 软件工程的三要素



2. 3. 1 过程(Process)

- 从管理的视角，回答软件开发、运行和维护需要开展哪些工作、按照什么样的步骤和次序来开展工作
- 对软件开发过程所涉及的人、制品、质量、成本、计划等进行有效和可量化的管理
- 典型成果
 - ✓ 过程模型，如瀑布模型、增量模型、原型模型、迭代模型、螺旋模型等等
 - ✓ 方法，如敏捷开发方法、群体化开发方法、DevOps方法
 - ✓ 管理，如配置管理、质量管理、团队组织等



2.3.2 方法学(Methodology)

- 从技术的视角，回答软件开发、运行和维护如何做 的问题
- 为软件开发过程中的各项开发和维护活动提供系统性、规范性的技术支持

- ✓ 如何理解和认识软件模型是什么
- ✓ 如何用不同抽象层次的模型来描述软件制品
- ✓ 采用什么样的建模语言来描述软件模型等等

□ 典型成果

- ✓ 结构化软件开发方法学
- ✓ 面向对象软件开发方法学
- ✓ 基于构件的软件开发方法学



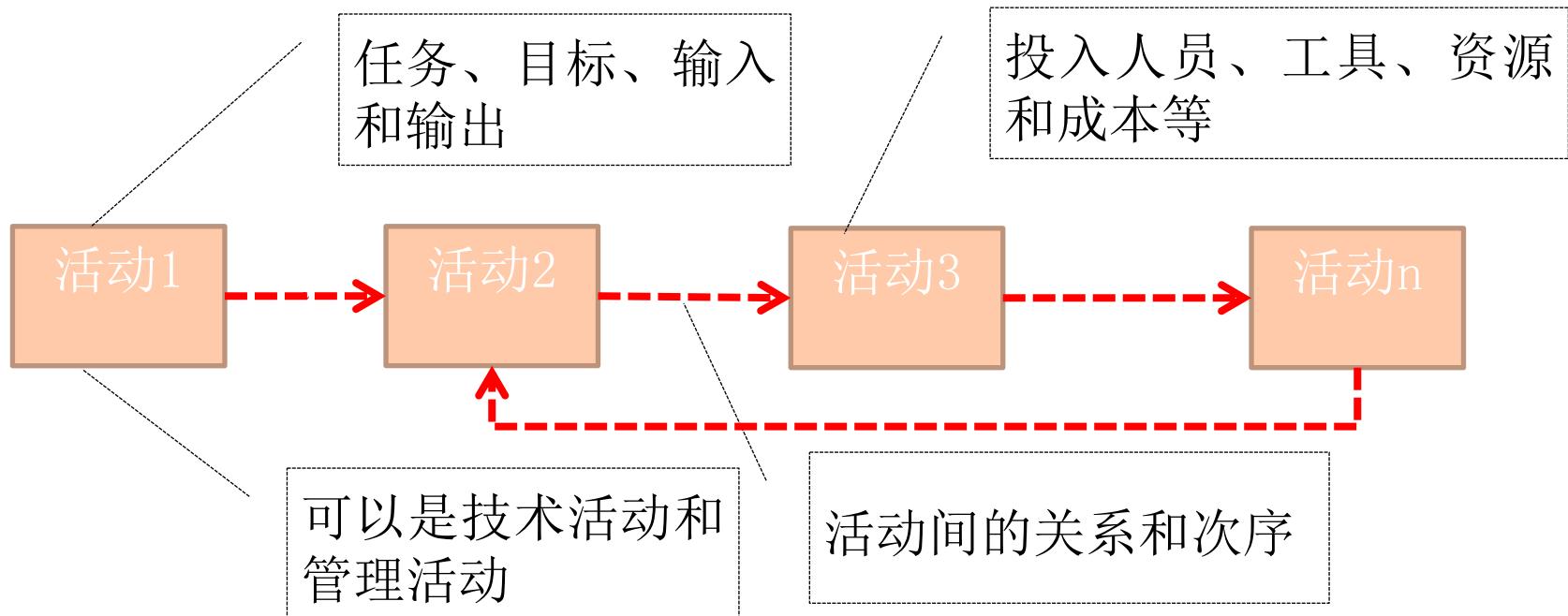
2. 3. 2 工具(Tool)

- 从工具辅助的视角，主要回答如何借助工具来辅助软件开发、运行和维护的问题
- 帮助软件开发人员更为高效地运用软件开发方法学来完成软件开发过程中的各项工作，提高软件开发效率和质量，加快软件交付进度。
 - ✓ 如需求分析、软件设计、编码实现、软件测试、部署运行、软件维护、项目管理、质量保证等，简化软件开发任务，
- 典型成果
 - ✓ SonarQube、Eclipse、Visual Studio等

1. 3 软件过程模型

□ 软件过程模型 (Software Process Model)

- ✓ 定义了软件开发的具体活动以及活动间的逻辑关系



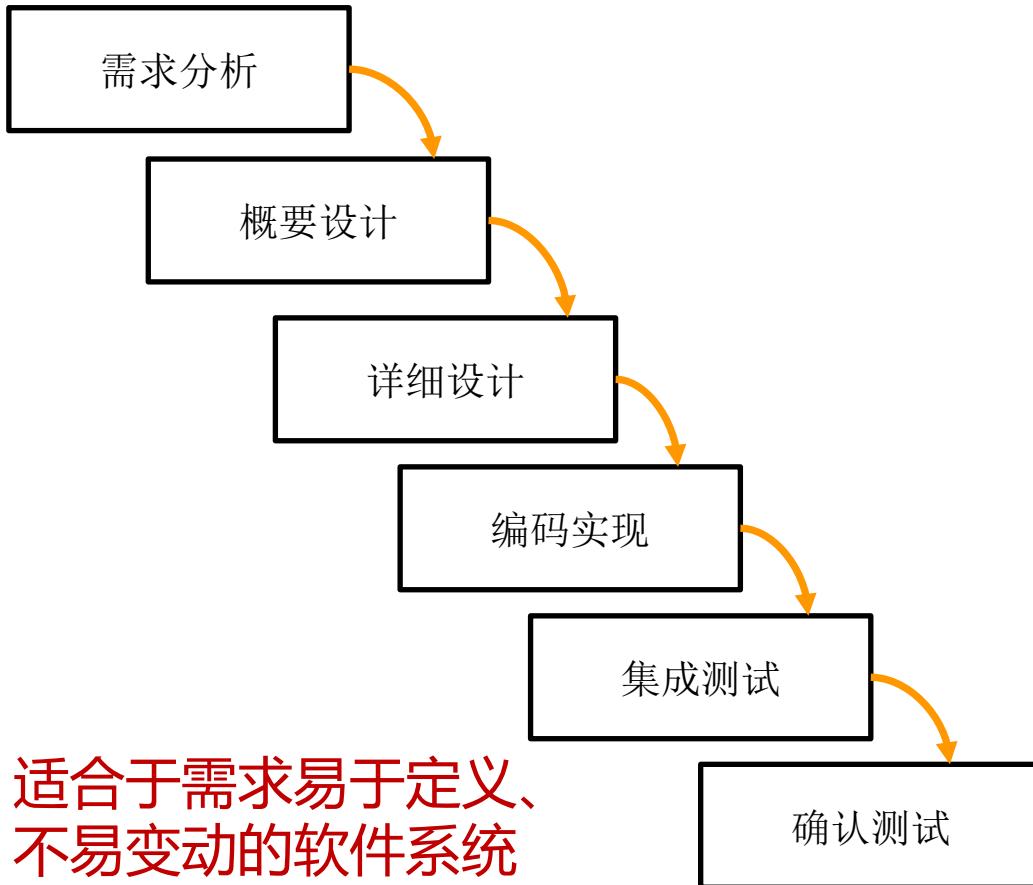


典型的软件过程模型

- 瀑布模型
- 增量模型
- 迭代模型
- 原型模型
- 螺旋模型

- 需要系统、规范性的软件过程模型的指导
- 每种软件过程模型有其各自的特点和适用的场所

2. 2 瀑布模型 (Waterfall Model)



1970提出的第一个软件过程模型

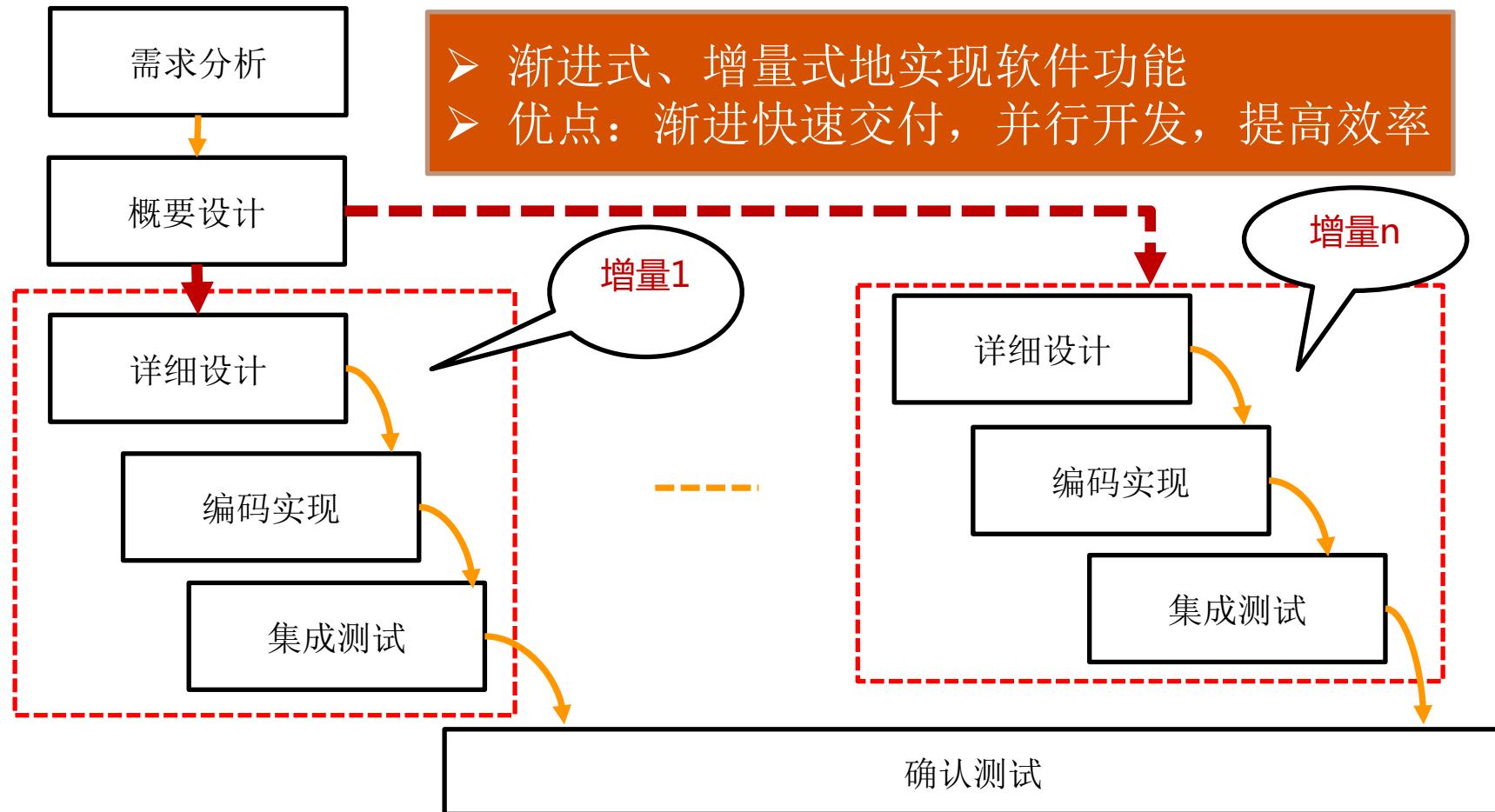
特点

- 与软件生命周期相互一致
- 步骤严格按照先后次序和逻辑关系来组织实施
- 上一步骤的输出是下一步骤的输入，下一步在需等到前一步骤完成后才能实施
- 每个活动结束后需要评审

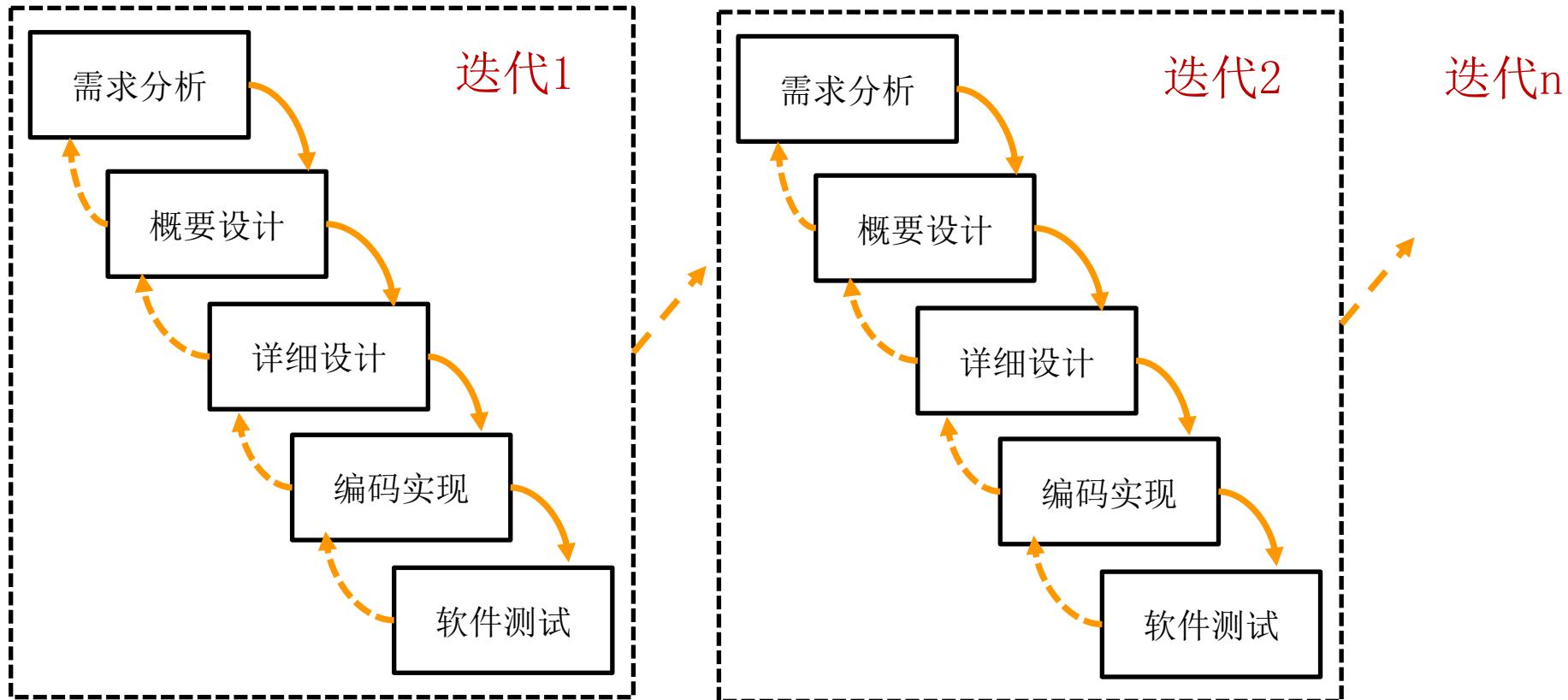
优点

- 简单，一目了然，易理解、掌握、应用和管理

2.4 增量模型(Incremental Model)



2.5 迭代模型(Iterative Model)



每次迭代只针对部分需求，完成部分可确定的软件需求



Why Software Engineering?

■ Who does software engineering?

- Customer, user, and developer

■ A system approach

- What is a system?
- The elements of a system: activity and object, relationship and boundary

■ An engineering approach

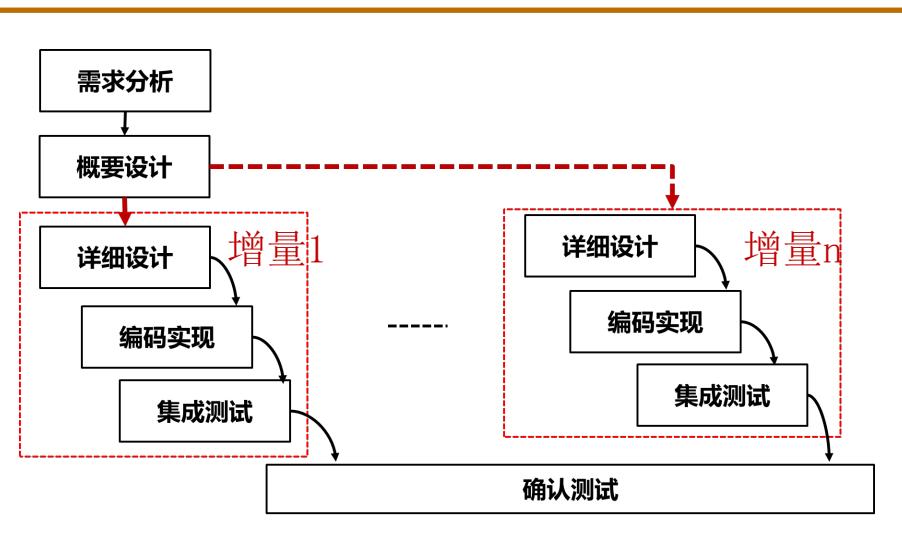
- Building a system

■ Members of the development team

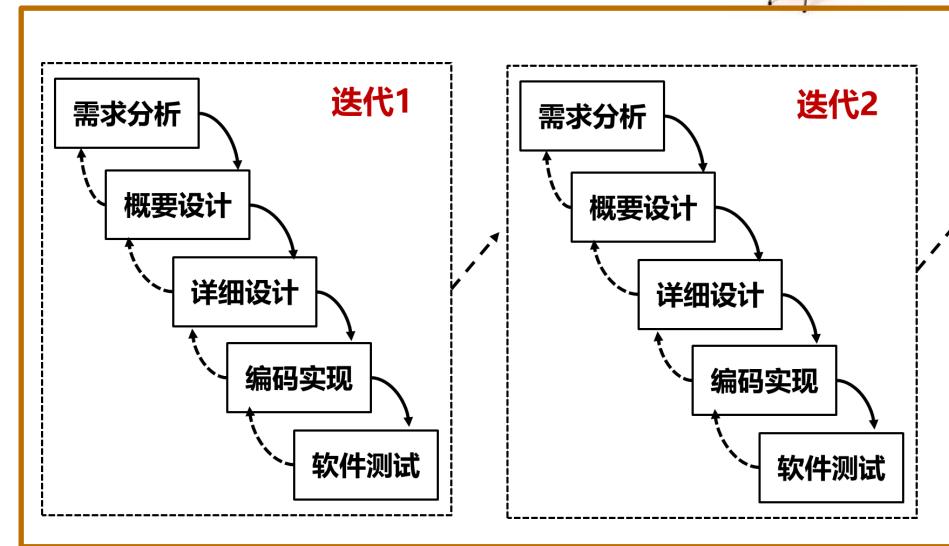
- The roles of the development team

思考和讨论

■ 增量过程模型与迭代过程模型有何区别？

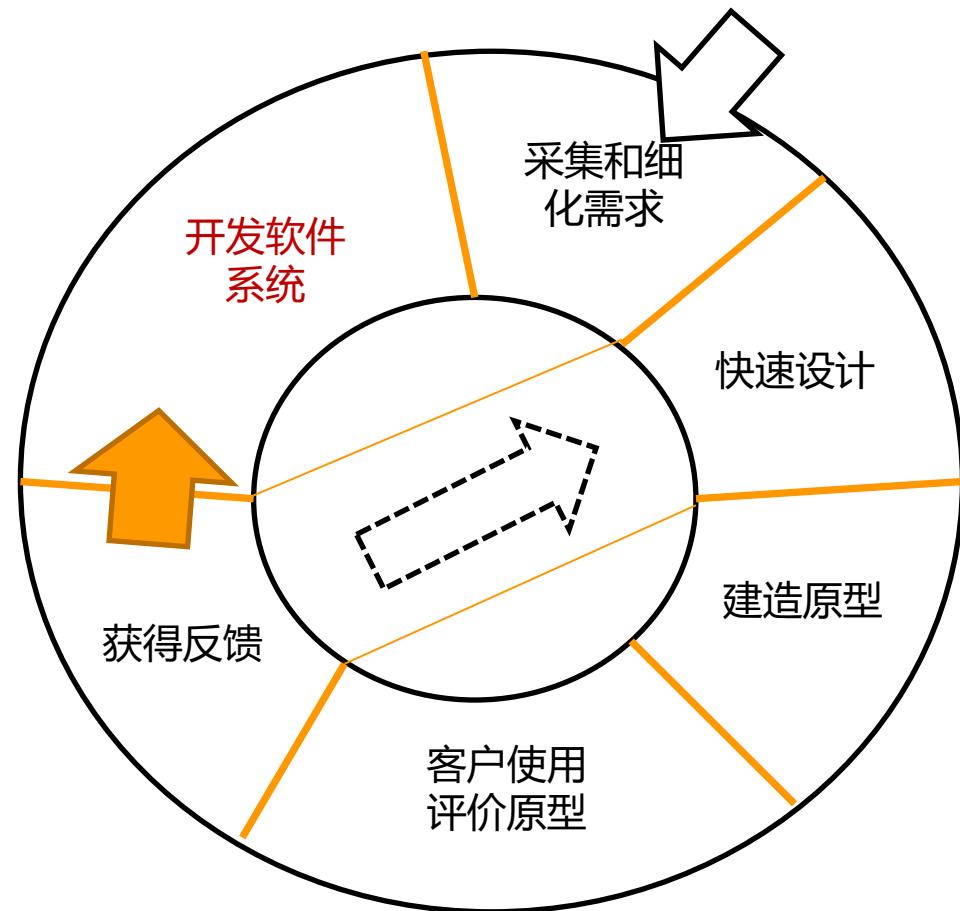


增量模型



迭代模型

2. 6 原型模型 (Prototype Model)



□ 何为软件原型?

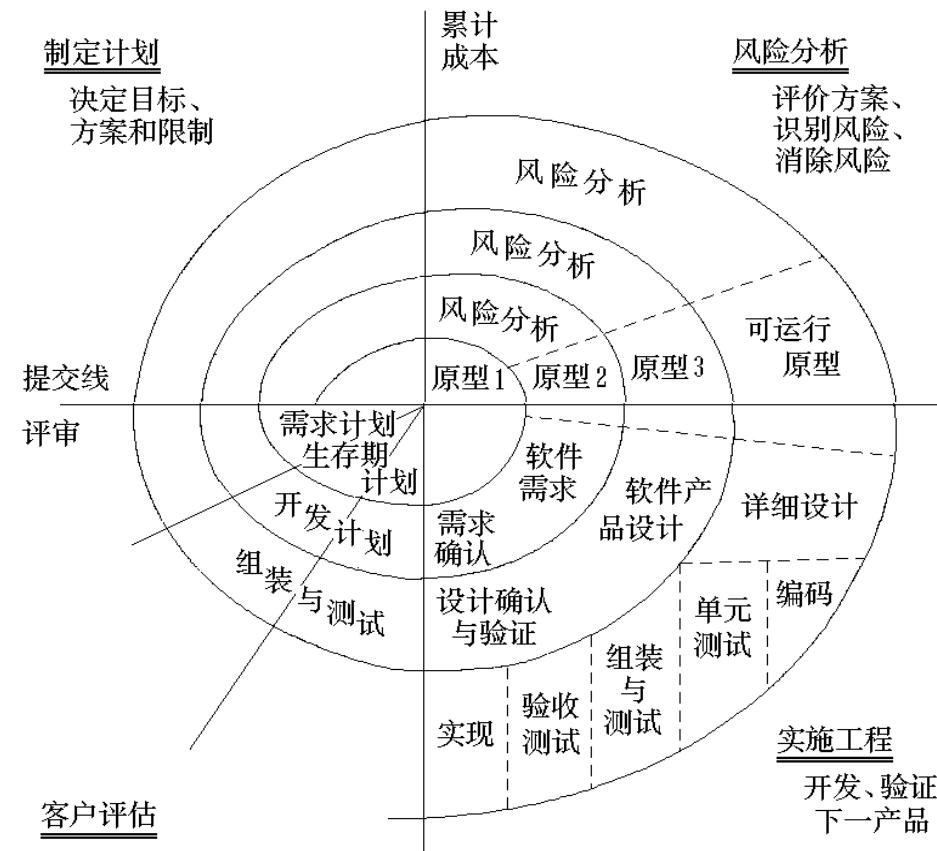
- ✓ 用户界面
- ✓ 执行流程
- ✓ 抛弃型原型
- ✓ 开发型原型



➤ 特点

- ✓ 软件原型作为交流载体和媒介
 - ✓ 支持用户参与到软件开发中
 - ✓ 持续、渐进地导出用户要求
- 适合于需求难导出、不易确定且持续变动的软件

2.7 螺旋模型 (Spiral Model)



口 软件风险

- ✓ 使软件开发受到影响和损失、甚至导致失败的、可能会发生的事件

- 集成迭代模型和原型模型
- 引入风险分析，风险驱动的过程模型
- 每个迭代四个阶段，若干活动
- 适合于需求不明确、开发风险高、开发过程中需求变更大的软件项目
- 不足：管理复杂



不同软件过程模型的特点

| 模型名称 | 指导思想 | 关注点 | 适合软件 | 管理难度 |
|------|--------------------|-----------------------------|-----------------------|------|
| 瀑布模型 | 提供系统性指导 | 与软件生命周期相一致 | 需求变动不大、较为明确、可预先定义的应用 | 易 |
| 原型模型 | 以原型为媒介指导用户的需求导出和评价 | 需求获取、导出和确认 | 理解需求难以表述清楚、不易导出和获取的应用 | 易 |
| 增量模型 | 快速交付和并行开发 | 软件详细设计、编码和测试的增量式完成 | 需求变动不大、较为明确、可预先定义的应用 | 易 |
| 迭代模型 | 多次迭代，每次仅针对部分明确软件需求 | 分多次迭代来开发软件，每次仅关注部分需求 | 需求变动大、难以一次性说清楚的应用 | 中等 |
| 螺旋模型 | 集成迭代模型和原型模型，引入风险分析 | 软件计划制定和实施，软件风险管理，基于原型的迭代式开发 | 开发风险大，需求难以确定的应用 | 难 |



软件需求(Software Requirement)分析

何为软件需求

- 从软件本身的角度，软件需求是指软件用于解决现实世界问题时所表现出的功能和性能等方面的要求
- 从软件利益相关方的角度，软件需求是指软件系统的利益相关方对软件系统的功能和质量，以及软件运行环境、交付进度等方面提出的期望和要求
- 软件需求刻画了软件系统能做什么（**What to do**），应表现出怎样的行为，需满足哪些方面的条件和约束等要求



软件需求的类别

□ 软件功能性需求(Functional)

- ✓ 功能需求描述了软件能做什么、具有什么功能、可提供什么样的服务
- ✓ 功能需求大多来自软件用户、客户以及开发者群体

□ 软件质量方面的需求(Quality)

- ✓ **外部质量属性**, 外部可展现的, 用户、客户等会非常关心, 如运行性能、可靠性、易用性等
- ✓ **内部质量属性**, 隐藏在内部的, 软件开发工程师会非常关心, 如可扩展性、可维护性、可理解性

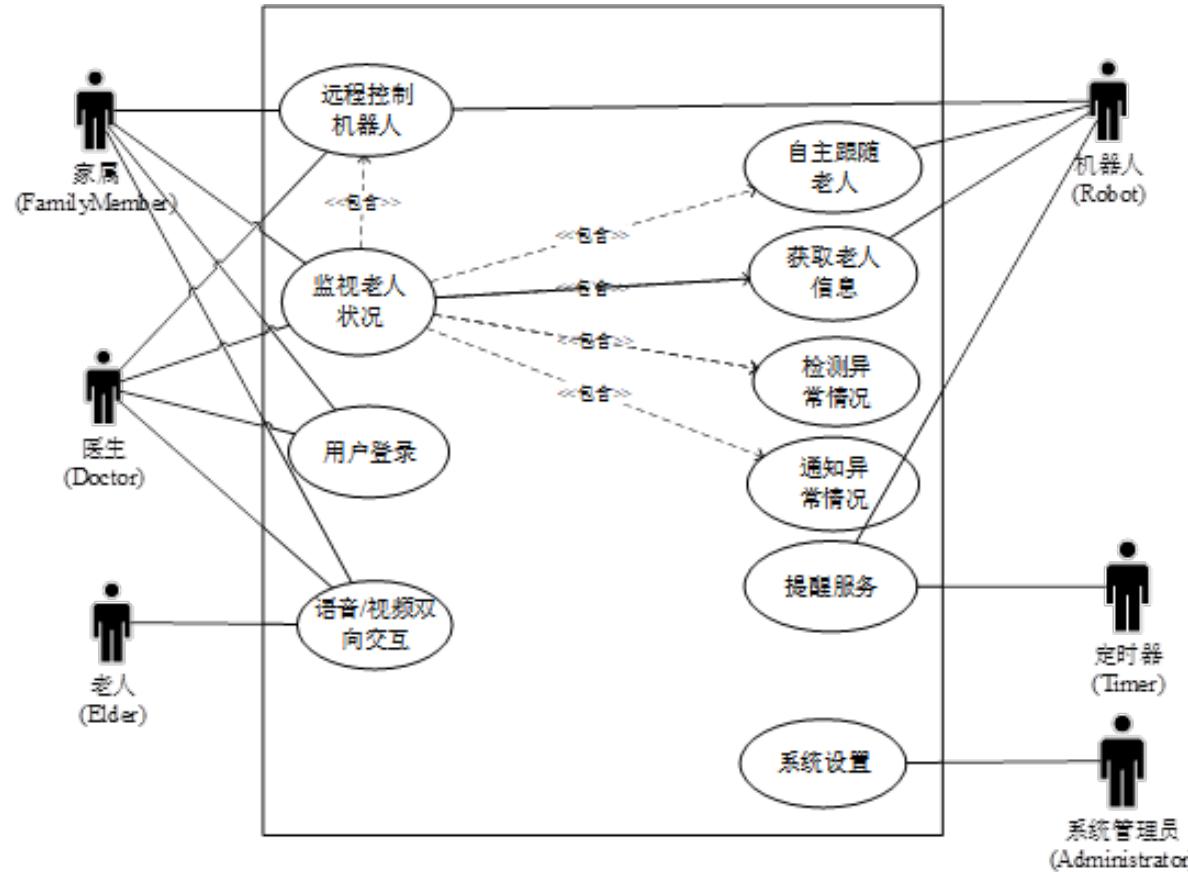
□ 软件开发约束性需求(Constraint)

- ✓ 开发成本、交付进度、技术选型、遵循标准等方面提出的要求

支持需求建模和分析的UML图

| 视点 | 图 (diagram) | 说明 |
|----|----------------------------|----------------------|
| 结构 | 包图 (package diagram) | 从包层面描述系统的静态结构 |
| | 类图 (class diagram) ✓ | 从类层面描述系统的静态结构 |
| | 对象图 (object diagram) | 从对象层面描述系统的静态结构 |
| | 构件图(component diagram) | 描述系统中构件及其依赖关系 |
| 行为 | 状态图(statechart diagram) | 描述状态的变迁 |
| | 活动图(activity diagram) | 描述系统活动的实施 |
| | 通信图(communication diagram) | 描述对象间的消息传递与协作 |
| | 顺序图(sequence diagram)✓ | 描述对象间的消息传递与协作 |
| 部署 | 部署图 (deployment diagram) | 描述系统中工件在物理运行环境中的部署情况 |
| 用例 | 用例图 (use case diagram) ✓ | 从外部用户角度描述系统功能 |

示例：初步软件需求

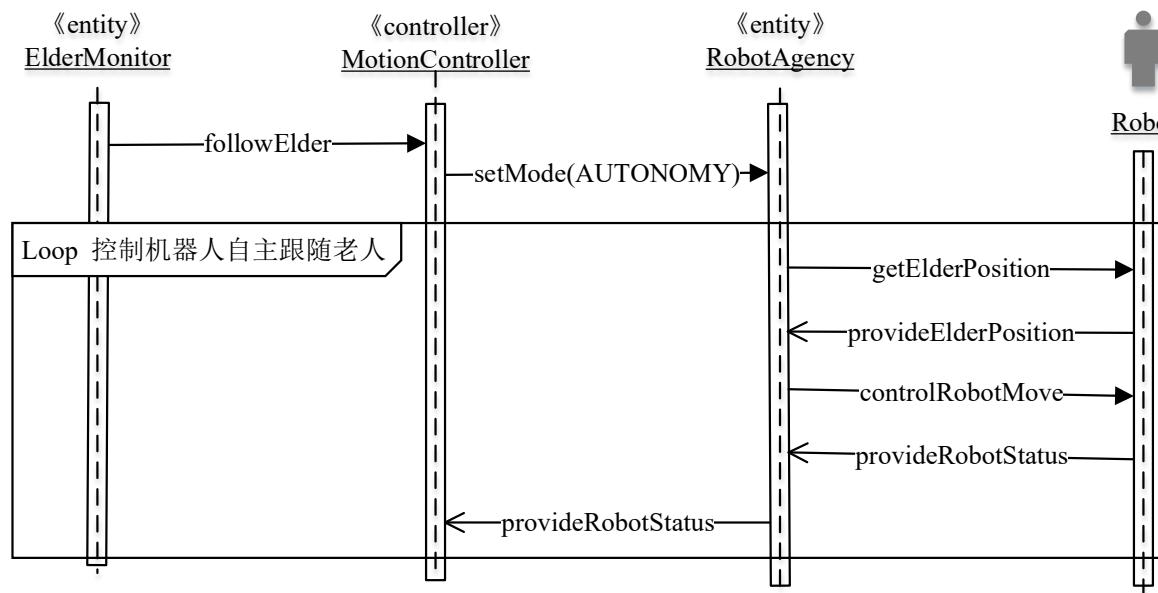


这张图说清楚软件需求了吗？

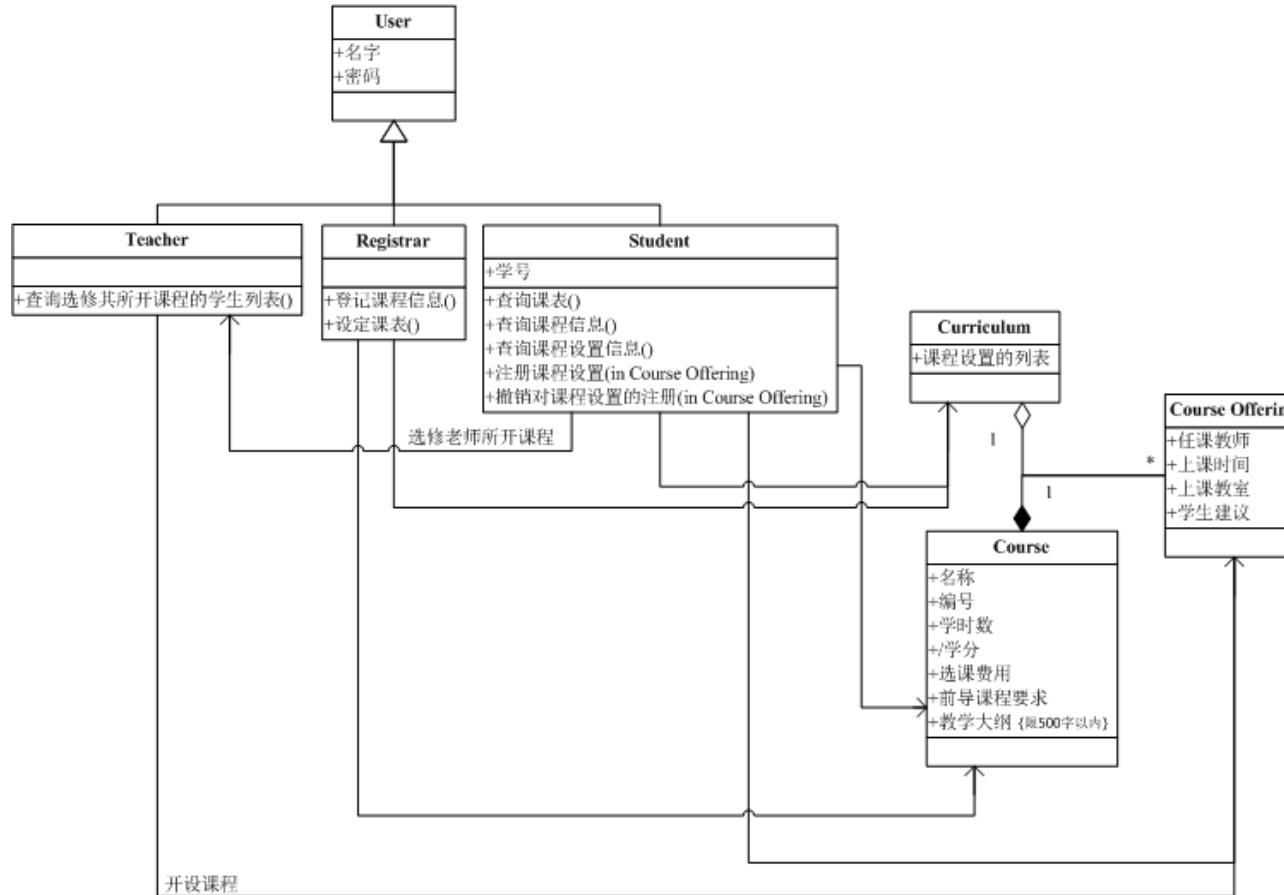
顺序图

口描述对象间的消息交互序列

- ✓ 纵向：时间轴，对象及其生命线(虚线)，活跃期(长条矩形)
- ✓ 横向：对象间的消息传递



示例：类图





概要设计

① 2.2 软件设计基本原则

- ① 抽象与逐步求精
- ② 模块化，高内聚度、低耦合度
- ③ 信息隐藏
- ④ 多视点和关注点分离
- ⑤ 软件重用
- ⑥ 迭代设计
- ⑦ 可追踪性



高内聚度原则

□ 何为模块的内聚度?

✓ 指该模块内各成分间彼此结合的紧密程度，越高越好，高内聚

□ 内聚度分类

✓ **偶然性内聚**: 模块内各成分为完成一组功能而结合在一起，关系松散

✓ **逻辑性内聚**: 模块完成的诸任务逻辑上相关

✓ **时间性内聚**: 模块内诸任务必须在同一时间段内执行

✓ **过程性内聚**: 模块内各成分相关且必须按特定次序执行

✓ **通讯性内聚**: 模块内各成分对数据结构的同一区域操作

✓ **顺序性内聚**: 模块内各成分与同一功能相关且顺序执行

✓ **功能性内聚**: 模块内各成分是一整体，完成单个功能



低耦合度原则

□ 何为模块间的耦合度?

- ✓ 模块间的相关程度，越低越好，低耦合

□ 耦合度分类

✓ 非直接耦合: 二个模块都不依赖对方而独立存在

✓ 数据耦合: 二个模块通过参数交换信息且仅限于数据

✓ 控制耦合: 二个模块通过参数交换信息包含控制信息

✓ 特征耦合: 介于数据耦合和控制耦合之间

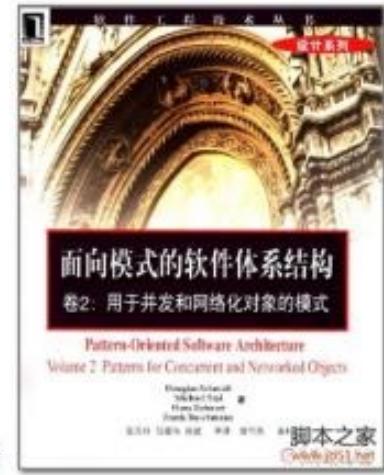
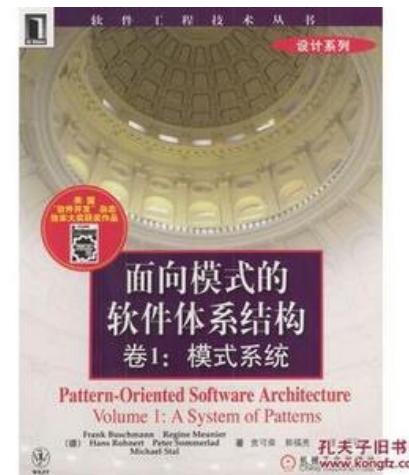
✓ 外部耦合: 二个模块与同一外部环境相关联(文件等)

✓ 公共耦合: 模块间通过全局数据环境相互作用

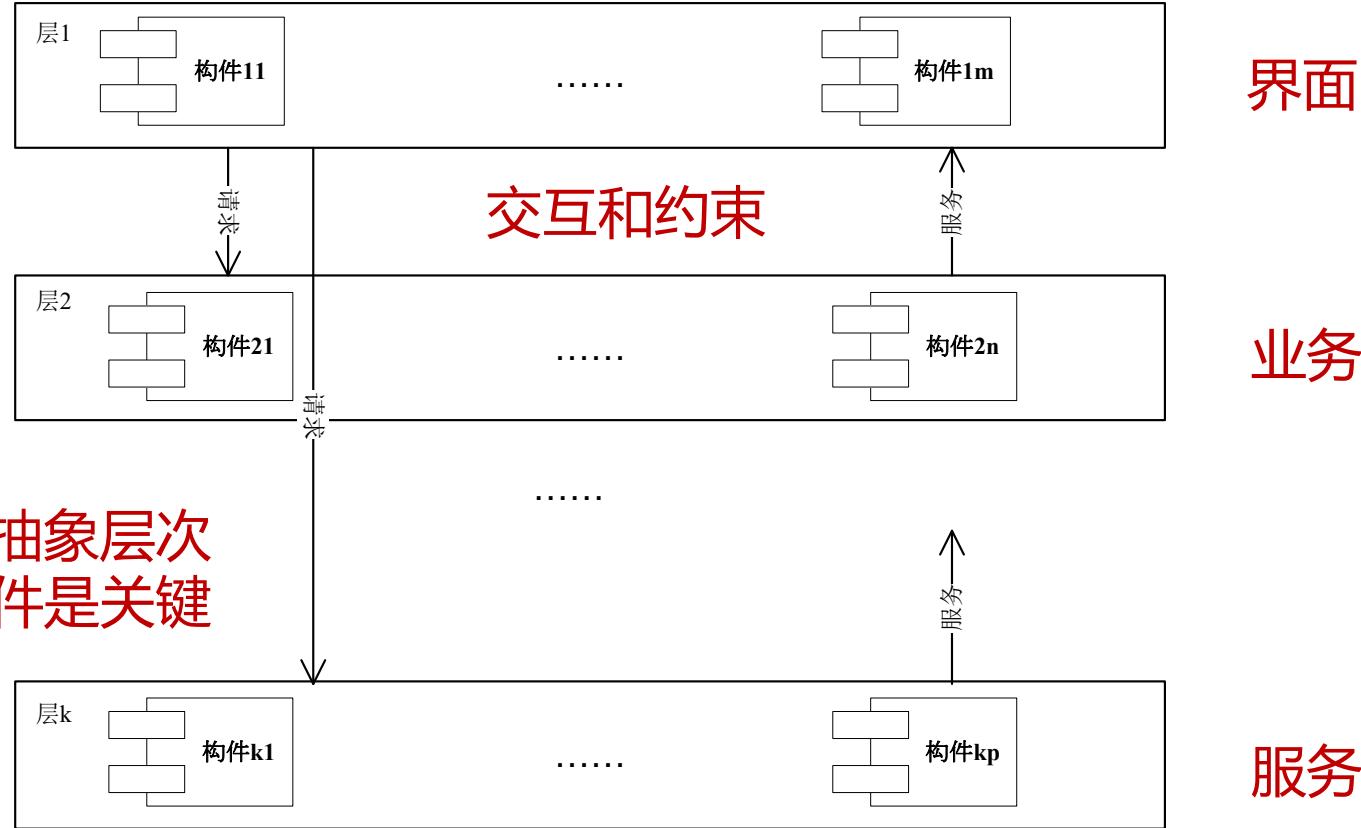
✓ 内容耦合: 一个模块使用另一模块内的数据和控制信息，或者直接转移到另一模块内执行

常用软件体系结构风格

- 分层风格
- 管道与过滤器风格
- 黑板风格
- MVC风格
- SOA风格
- 总线风格
-



示例：分层体系结构风格





1. 5. 2 管道与过滤器风格

□ 构件

- ✓ 将软件功能实现为一系列处理步骤，每个步骤封装在一个过滤器构件中

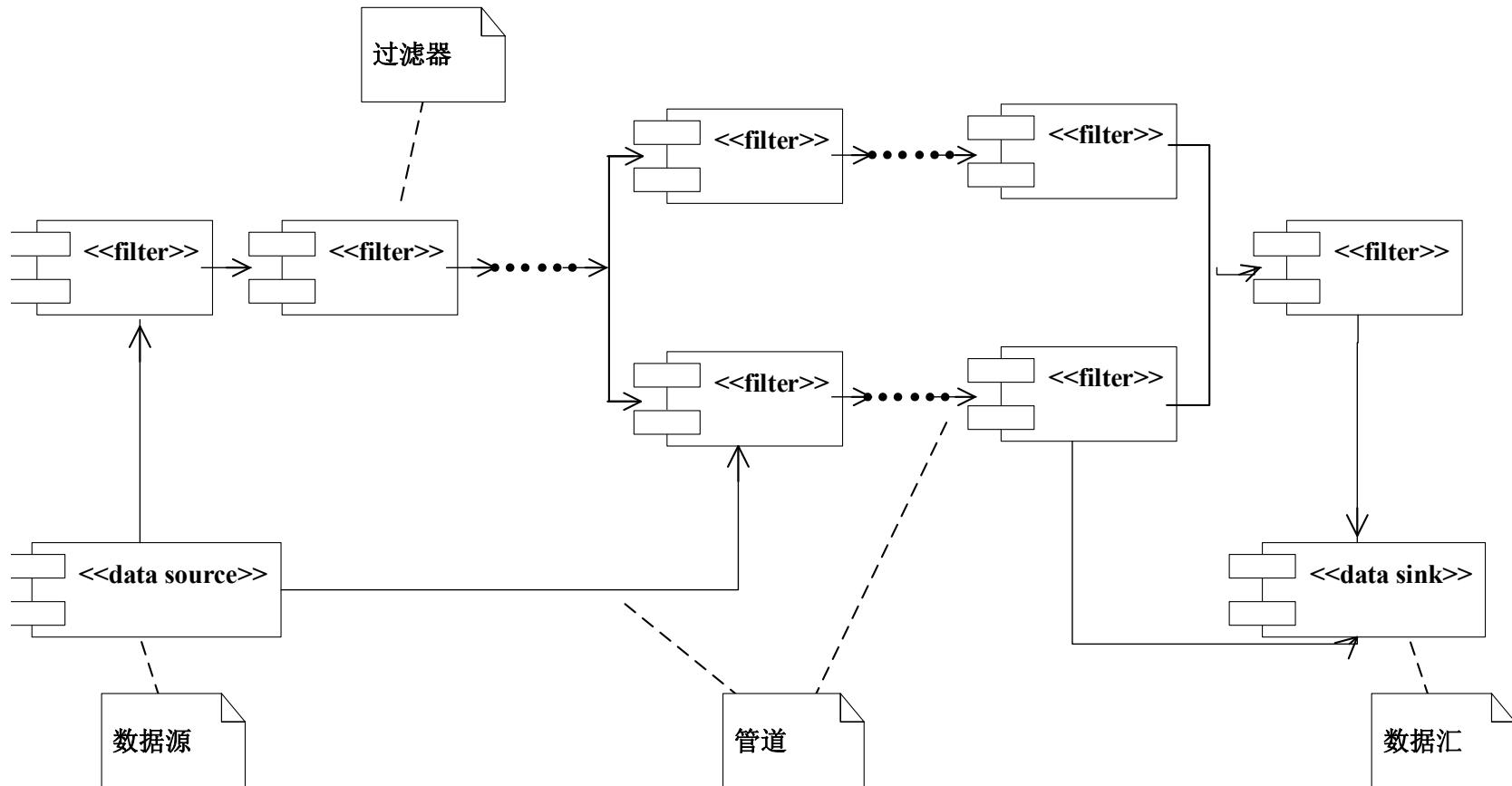
□ 连接子

- ✓ 相邻过滤器间以管道连接，一个过滤器的输出数据借助管道流向后续过滤器，作为其输入数据

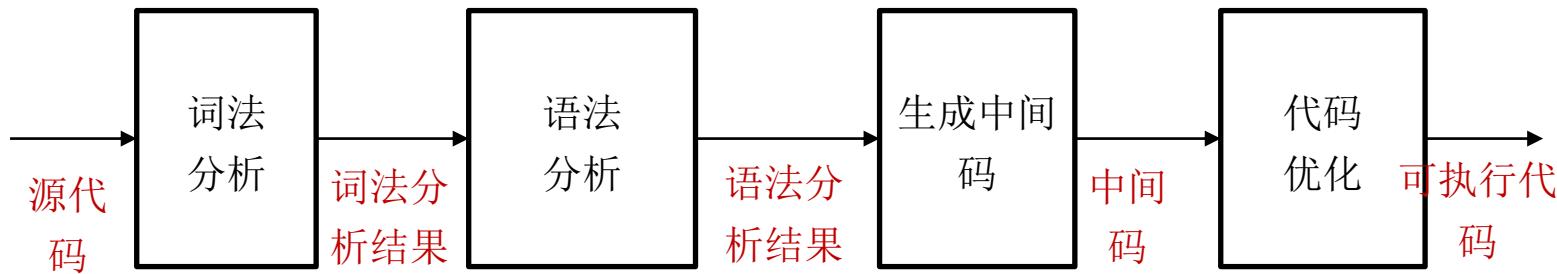
□ 数据

- ✓ 软件系统的输入由数据源（data source）提供
- ✓ 软件最终输出由源自某个过滤器的管道流向数据汇（data sink）
- ✓ 典型数据源和数据汇包括数据库、文件、其他软件系统、物理设备等

管道与过滤器模式的示例



示例：管道与过滤器风格



编译器采用的就是一个典型的管道/过滤器风格

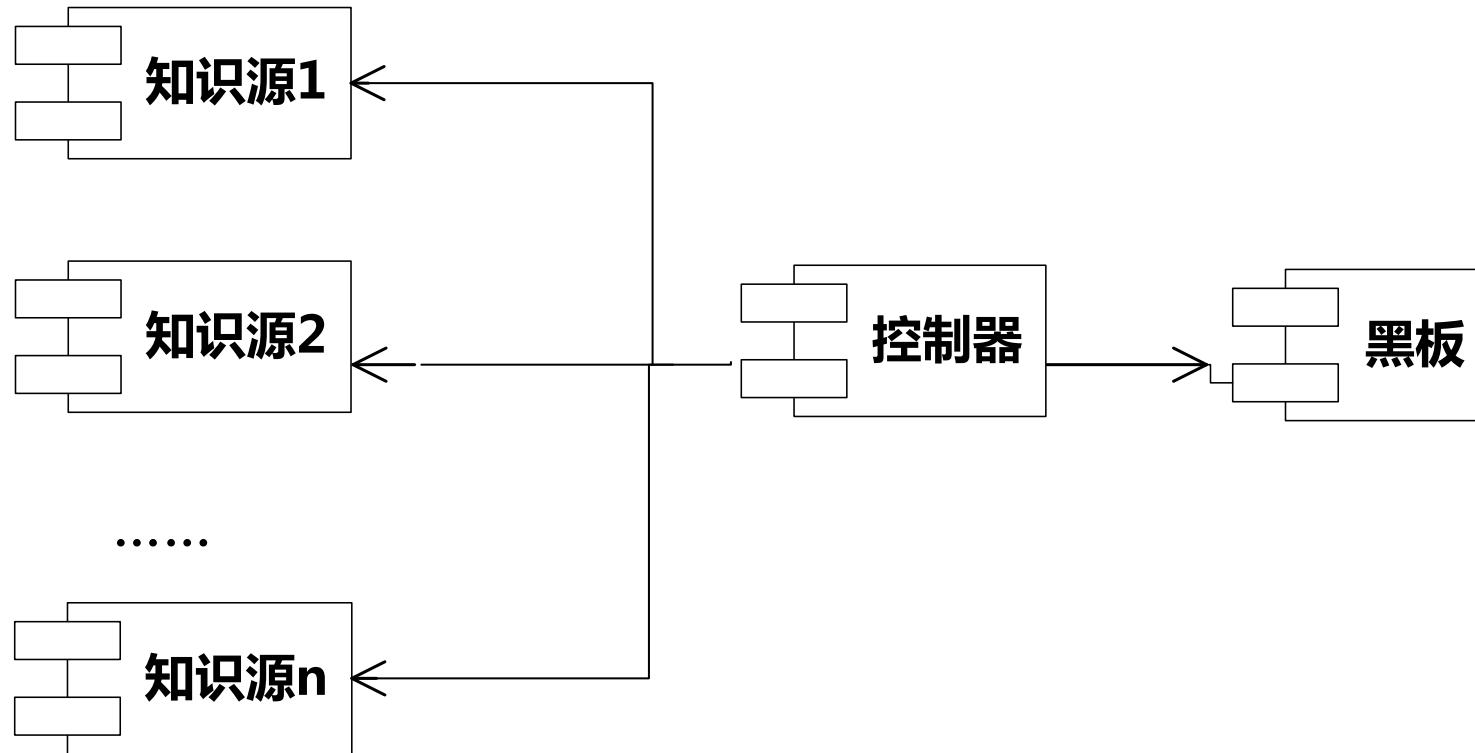


1. 5. 3 黑板风格

- 将软件系统划分为黑板、知识源和控制器三类构件
 - ✓ **黑板**: 负责保存问题求解过程中的状态数据，并提供这些数据的读写服务
 - ✓ **知识源**: 负责根据黑板中存储的问题求解状态评价其自身的可应用性，进行部分问题求解工作，并将此工作的结果数据写入黑板
 - ✓ **控制器**: 负责监视黑板中不断更新的状态数据，安排（多个）知识源的活动。



黑板风格示意图





1. 5. 4 MVC风格

□ 模型构件

- ✓ 负责存储业务数据并提供业务逻辑处理功能

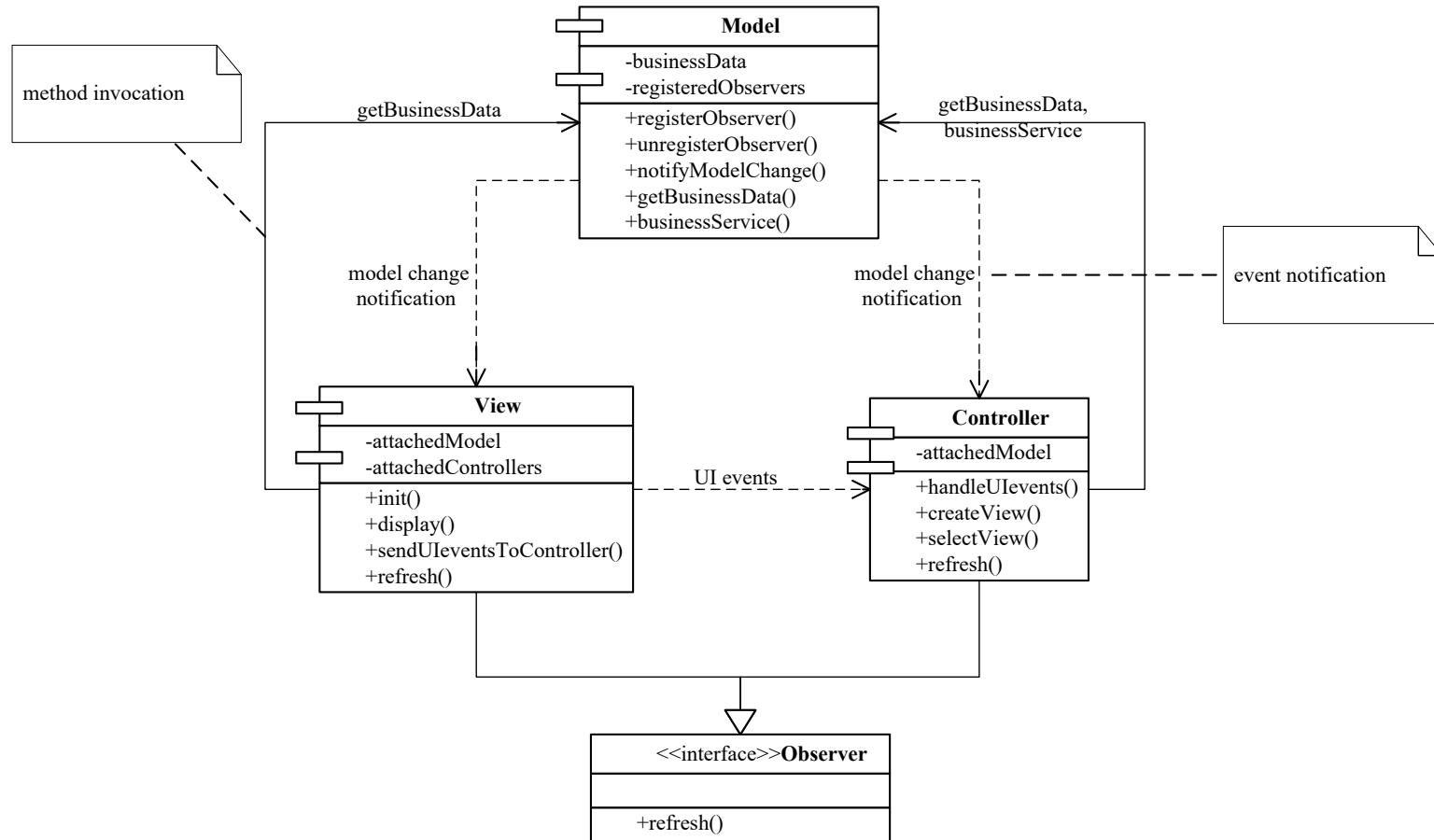
□ 视图构件

- ✓ 负责向用户呈现模型中的数据

□ 控制器

- ✓ 在接获模型的业务逻辑处理结果后，负责选择适当的视图作为软件系统对用户的界面动作的响应

MVC体系结构示意图





不同体系结构风格适合的应用

| 类别 | 特点 | 典型应用 |
|----------|------------------------------------|---------------------------|
| 管道/过滤器风格 | 数据驱动的分级处理，处理流程可灵活重组，过滤器可重用 | 数据驱动的事务处理软件，如编译器、Web服务请求等 |
| 层次风格 | 分层抽象、层次间耦合度低、层次的功能可重用和可替换 | 绝大部分的应用软件 |
| MVC风格 | 模型、处理和显示的职责明确，构件间的关系局部化，各个软构件可重用 | 单机软件系统，Web应用软件系统 |
| SOA风格 | 以服务作为基本的构件，支持异构构件之间的互操作，服务的灵活重用和组装 | 部署和运行在云平台上的软件系统 |
| 消息总线风格 | 提供统一的消息总线，支持异构构件之间的消息传递和处理 | 异构构件之间消息通信密集型的软件系统 |



软件测试

□ 测试、调试和排错

□ 目的

- ✓ 测试发现缺陷
- ✓ 调试定位缺陷
- ✓ 排错纠正错误

□ 独立性不同

- ✓ 测试由独立的测试小组进行
- ✓ 调试和排错由开发人员完成

2. 4. 1 单元测试

□ 测试对象

- ✓ 对软件基本模块单元进行测试
- ✓ 过程、函数、方法、类

□ 测试方法

- ✓ 大多采用白盒测试技术

□ 测试的内容

- ✓ 模块接口测试
- ✓ 模块局部数据结构测试
- ✓ 模块独立执行路径测试
- ✓ 模块错误处理通道测试
- ✓ 模块边界条件测试



在详细设计阶段就可以设计单元测试用例及计划

2. 4. 2 集成测试

□ 测试对象

- ✓ 对软件模块之间的接口进行测试
- ✓ 过程调用、函数调用、消息传递、远程过程调用

□ 测试技术

- ✓ 采用黑盒测试技术

□ 集成测试的内容

- ✓ 过程调用
- ✓ 函数调用
- ✓ 消息传递
- ✓ 远程过程调用
- ✓ 网络消息



2. 4. 3 确认测试

□ 测试对象

- ✓ 对软件的功能和性能进行测试
- ✓ 判断目标软件系统是否满足用户需求

□ 依据和标准

- ✓ 软件需求规格说明书

□ 测试技术

- ✓ 采用黑盒测试技术



在需求分析阶段就可以设计确认测试用例及计划

白盒测试用例设计的指导原则

□ 如何设计测试用例？

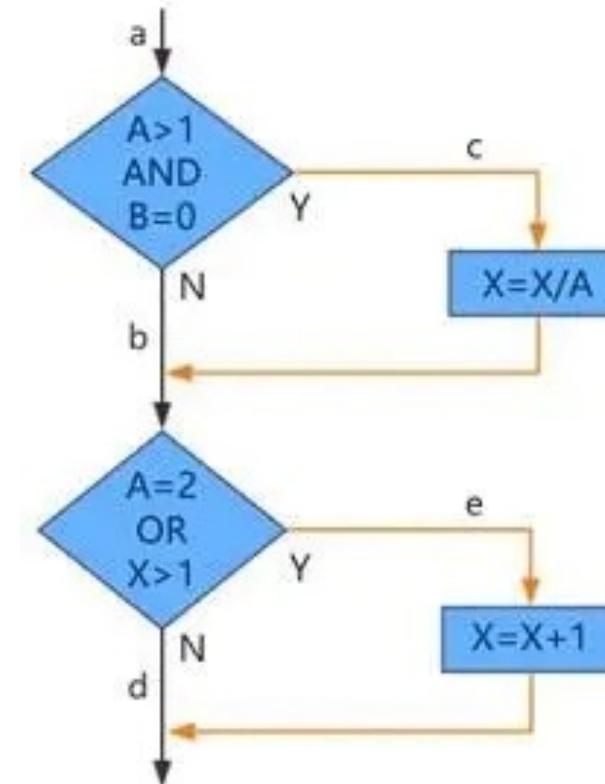
- ✓ 内部执行流程
- ✓ 生成测试数据

□ 设计多少测试用例？

- ✓ 遵循覆盖原则

□ 测试用例覆盖准则

- ✓ 语句覆盖
- ✓ 分支覆盖
- ✓ 路径覆盖
- ✓ 基本路径覆盖



- ✓ 语句覆盖
- ✓ 分支覆盖
- ✓ 路径覆盖
- ✓ 基本路径覆盖



写出各种覆盖的路径和测试用例

语句覆盖

a-c-e

A=2, B=0, X=2

分支覆盖

T1T2 a-c-e, A=2, B=0, X=2^d

F1F2 a-b-d, A=0, B=2, X=1

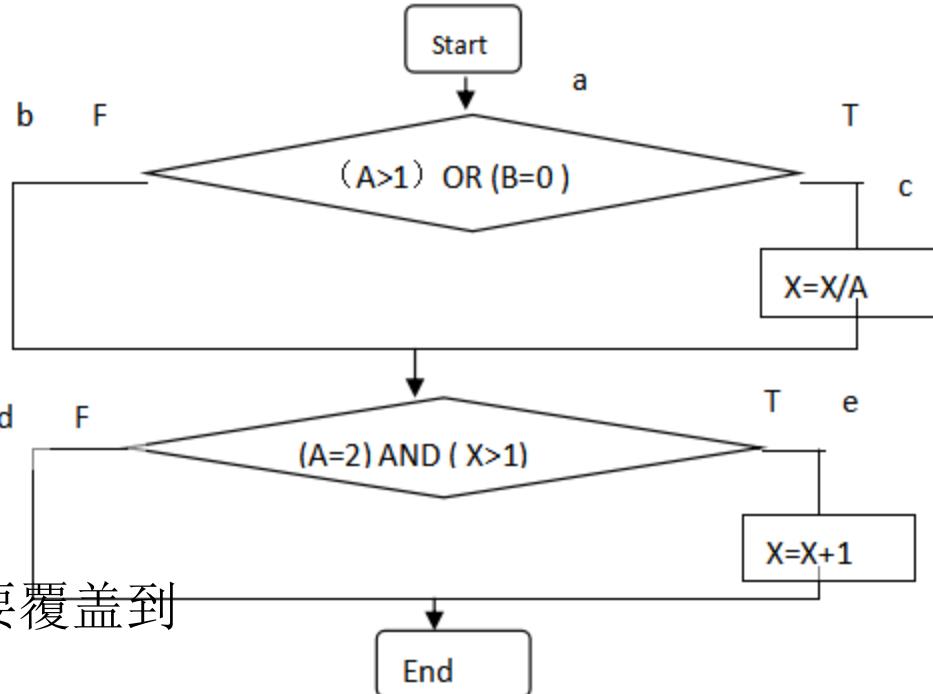
条件覆盖，每个条件的真和假都要覆盖到

A>1 T ;A<=1 F

B=0 T;B!=0 F

A=2 T; A!=2 F

X>1; T;X<=1 F



两个测试用例 TTTT; FFFF

路径覆盖

a-c-e, a-b-d, a-b-e

a-c-d

黑盒测试-等价分类法

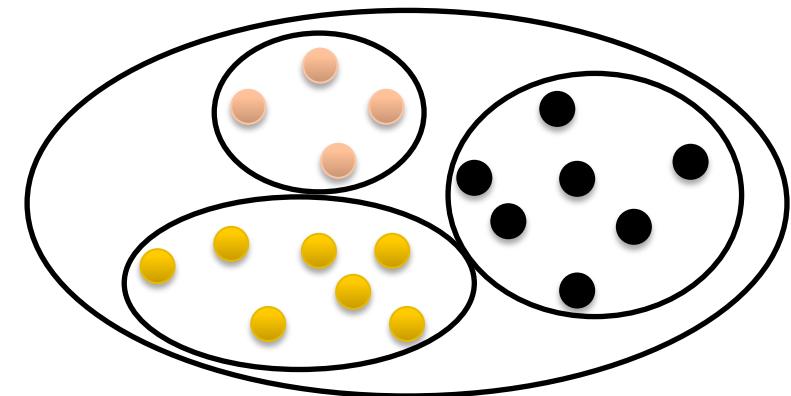
口思想

- ✓ 把程序的输入数据集合按输入条件划分为若干个**等价类**
- ✓ 每一个等价类对于输入条件而言为一组有效或无效的输入
- ✓ 为每一个等价类设计一个测试用例

口优点

- ✓ 减少测试次数，不丢失发现错误的机会

每个等价类中
的数据具有相
同的测试特征





等价分类法的基本原则

□ 输入条件为一范围

- ✓ 划分出三个等价类
- ✓ (1) 有效等价类(在范围内), (2) 大于输入最大值, (3) 小于输入最少值

□ 输入条件为一值

- ✓ 划分为三个等价类
- ✓ (1) 有效, (2) 大于, (3) 小于

□ 输入条件为集合

- ✓ 划分二个等价类
- ✓ (1) 有效(在集合内), (2) 无效(在集合外)

□ 输入条件为一个布尔量

- ✓ 划分二个等价类
- ✓ (1) 有效(此布尔量), (2) 无效(布尔量的非)

等价分类法示例

□ $z = \text{func}(x, y)$:

- ✓ 当 $0 < x < 1024$ 且 $y = 0$, $z = -1$
- ✓ 否则, $z = x * \lg(y)$

测试用例9个

□ 关于x的等价类

- ✓ (0, 1024)
- ✓ (-#, 0]
- ✓ [1024, +#)

□ 关于y的等价类

- ✓ {0}
- ✓ (-#, 0)
- ✓ (0, +#)

- | |
|------------------------|
| ① X=1,y=0, z=-1 |
| ② X=1,y=-1, z=** |
| ③ X=1,y=1 , z=** |
| ④ X=0,y=1 , z=** |
| ⑤ X=0,y=-1 , z=** |
| ⑥ X=0,y=1 , z=** |
| ⑦ X=2000,y=0 , z=** |
| ⑧ X=2000,y=-100 , z=** |
| ⑨ X=2000,y=200 , z=** |



1. 3 软件维护的形式

- 纠正性维护
- 完善性维护
- 适应性维护
- 预防性维护

纠正性维护

□ 何为纠正性维护

- ✓ 纠正软件中的缺陷和错误

□ 起因

- ✓ 用户在使用软件过程中一旦发现缺陷，他们会向开发人员提出纠正性维护的请求

□ 目的

- ✓ 诊断和改正软件系统中潜藏的缺陷

为什么软件在使用时还有缺陷存在？



适应性维护

□ 何为适应性维护?

- ✓ 对软件进行改造以便适应新的运行环境和平台

□ 起因

- ✓ 软件运行于一定的环境(硬件、OS、网络等)之上，运行环境发展很快，出现了变化

□ 目的

- ✓ 适应环境变化和发展而对软件进行维护

为什么软件需要适应环境的变化?



改善性维护

□ 何为改善性维护?

- ✓ 对软件进行改造以增加新的功能、修改已有的功能

□ 起因

- ✓ 在软件系统运行期间，用户可能要求增加新的功能、建议修改已有功能或提出其他改进意见

□ 目的

- ✓ 满足用户日益增长的各种需求而对软件系统进行的改善和补充

小米便签维护中是否涉及完善性维护？做了哪些完善性工作



预防性维护

□ 何为预防性维护？

- ✓ 对软件结构进行改造以便提高软件的可靠性和可维护性等

□ 起因

- ✓ 为进一步改善软件系统的可维护性和可靠性，为以后的软件改进奠定基础的维护活动

□ 目的

- ✓ 获取软件结构，重新改善软件结构

小米便签需要做预防性维护吗？



1. 4 软件维护的特点

□ 同步性

- ✓ 软件维护需要与软件使用同步进行

□ 周期长

- ✓ 软件维护周期会更长，一些软件会服役十几年甚至几十年的时间

□ 费用高

- ✓ 维护成本高达总成本80%以上，维护费用是开发费用的3倍以上

□ 难度大

- ✓ 充分理解待维护软件的架构、设计和代码，这极困难。尤其是在软件设计文档缺失的情况下，这一问题更为突出

- ✓ 50%-90%的时间被消耗在理解程序上

Question?

