# 复习

第1章　软件工程学概述

第3章　需求分析（包含第2章的数据流和数据字典）

第5章　总体设计

第6章　详细设计

第7章　实现

第8章　维护

第9章　面向对象方法学引论

第10章　面向对象分析

第11章　面向对象设计

# 软件危机

软件危机是指在软件开发和维护过程中所遇到的一系列严重问题

- 成本/进度估计不准确
- 闭门造车，用户不满意
- 不可维护
- 软件成本的比例逐年上升
- 软件产率低
- 软件质量得不到保证
- 。。。。

# 软件危机

## 产生软件危机的原因

- 软件规模加大、复杂性提高
- 软件是逻辑产品，缺乏“可见性”
- 缺乏有效的、系统的技术手段和管理方法
- 用户和软件开发人员的理解鸿沟（Gap）
- 错误的认识和做法
  - 忽视软件需求分析的重要性
  - 认为软件开发就是写程序并设法使之运行
  - 轻视软件测试和软件维护

# 软件危机

## 消除软件危机的途径

- **消除 “软件就是程序”的错误观念**

  软件是程序、数据及相关文档的完整集合。文档是开发、使用和维护程序所需要的图文资料。

- **成功的软件开发技术和方法**
- **软件工具和软件工程支撑环境**

# 第1章　软件工程学概述

- 1968年 NATO 计算机科学会议
- 软件危机→解决途径→软件工程

## ◆ 软件工程

软件工程是指导计算机软件开发和维护的一门工程学科。采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，以经济地开发出高质量的软件并有效地维护它。

# 第1章　软件工程学概述

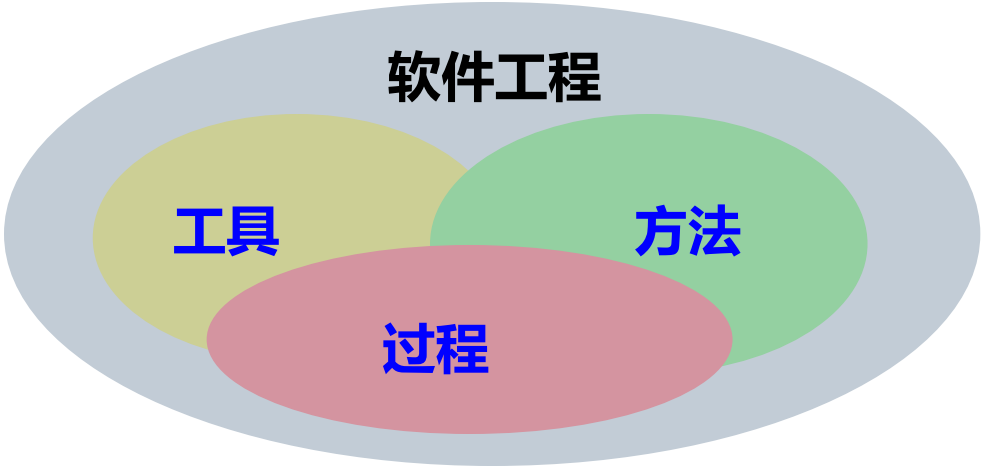
## 软件工程的本质特性

- 关注大型程序的构造
- 中心课题是控制复杂性
- 软件经常变化
- 开发效率非常重要
- 和谐合作：纪律是软件开发项目的一个关键
- 软件必须有效地支持它的用户（V & V）
- 一种文化背景的人替另一种文化背景的人创造产品

# 第1章　软件工程学概述

## 软件工程方法学

- 包括技术和管理两方面
- 3要素



# 第1章　软件工程学概述

- **软件生命周期的3个大阶段**

1. **软件定义：确定总目标、工程的可行性；导出实现策略及系统功能；估计资源和成本，并且制定工程进度表。**
   - 问题定义、可行性研究、需求分析
2. **软件开发：具体设计和实现在前一个时期定义的软件**
   - 总体设计、详细设计、编码和单元测试、综合测试
3. **软件维护：使软件持久地满足用户的需要**

# 第1章　软件工程学概述

- 软件生命周期的8阶段

1. 问题定义：　要解决的问题是什么?
2. 可行性研究：　问题是否有可行的解决办法?
3. 需求分析：　目标系统必须做什么
4. 总体设计（概要设计）：　概括地说，应该怎样实现目标系统?
5. 详细设计：　应该怎样具体地实现系统?
6. 编码和单元测试：　编写代码、测试模块
7. 综合测试：　集成测试、验收测试、现场测试或平行运行
8. 软件维护：　使系统持久地满足用户的需要

# 第1章　软件工程学概述

- 过程模型

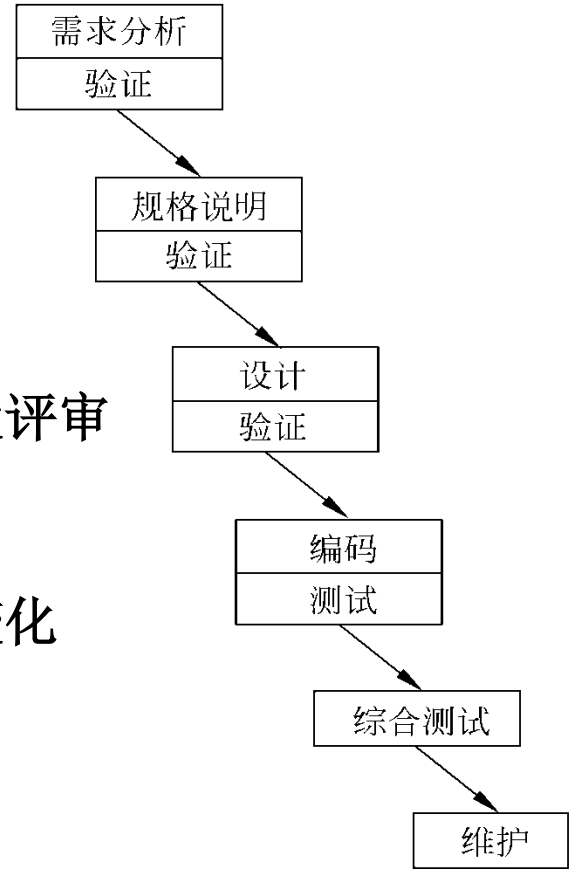
  将生命周期划分成阶段及各阶段的执行顺序

- 典型的过程模型

  1. 瀑布模型(Waterfall model)：文档驱动
  2. 快速原型开发模型(Rapid Prototyping model)
  3. 增量模型(Incremental model)
  4. 螺旋模型(Spiral model)：风险驱动
  5. 喷泉模型
  6. 敏捷开发（如：XP，eXtreme Programming, 极限编程）
  7. RUP

# 第1章　软件工程学概述

- 瀑布模型




特点

1. 阶段的顺序性和依赖性
2. 推迟实现：区分逻辑与物理设计
3. 文档驱动（质量保证）：阶段文档及阶段评审

缺点

1. 开发过程不能逆转，不适合需求的动态变化
2. 开发后期客户才能看到软件
3. 文档驱动

采用瀑布模型要求：较稳定的需求

# 第1章　软件工程学概述

## 快速原型模型

### ◆ 特点

✓ 快速原型可以取代规格说明阶段，但不是设计阶段

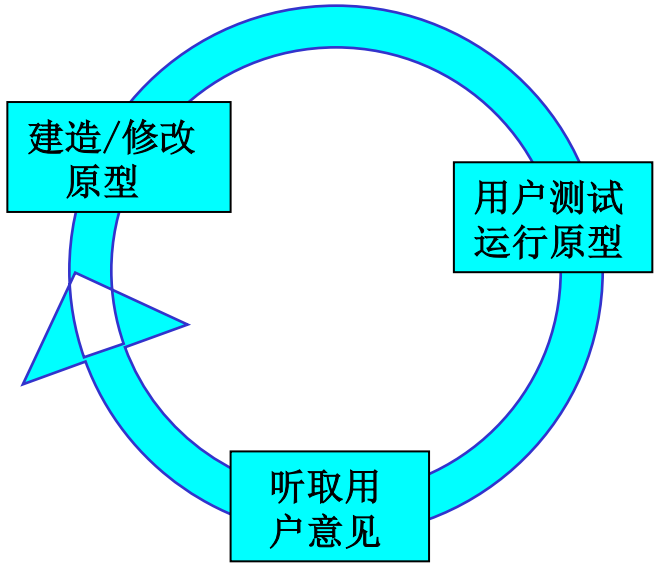
✓ 易适应需求变化

✓ 开发与培训的同步

### ◆ 应用范围

✓ 用户需求不完全或不确定

✓ 项目招投标



# 第1章　软件工程学概述

## ◆ 增量模型

把软件作为一系列增量构件来开发

- 优点
  - 分阶段交付产品，早期回报
  - 适应需求变化
- 缺点
  - 需要一个开放结构，方便构件加入

# 软件过程：敏捷开发

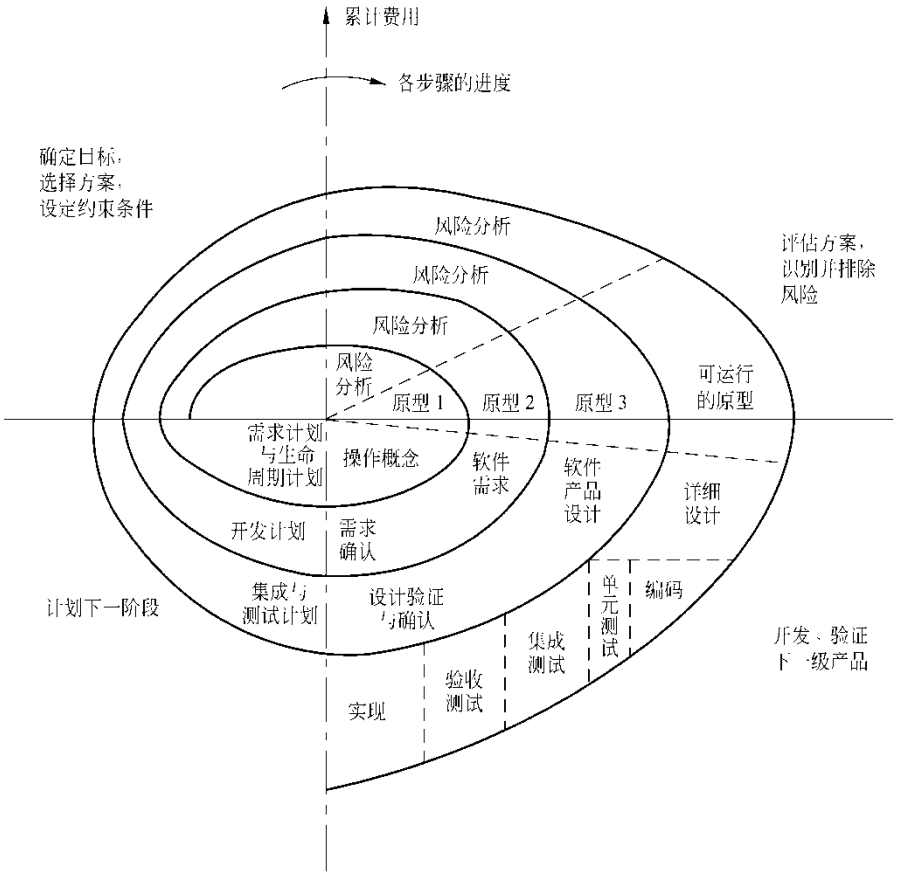
## 极限编程（XP）的有效实践

- 客户作为开发团队成员
- 成对编程(pair programming)
- 短交付周期
- 测试驱动开发
- ……

# 软件过程：螺旋模型

- 结合瀑布模型和快速原型模型，强调“风险分析”
- 可看成每阶段之前增加了风险分析过程的快速原型模型

# 软件过程：螺旋模型



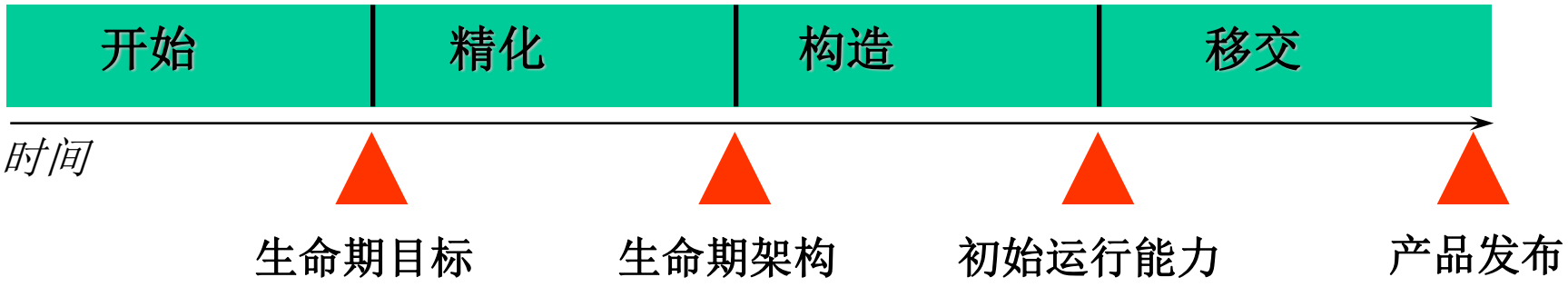

**完整的螺旋模型**

# 软件过程：喷泉模型

- 迭代：软件开发过程的一种内在属性
  - 阶段间的迭代
  - 阶段内工作步骤的迭代
- 工作重点：分析阶段
  - 定义面向问题的对象，不断充实和扩充对象
  - 无缝连接：分析得到的对象模型也适用设计和实现

喷泉模型体现了面向对象开发过程迭代和无缝连接的特性

# 软件过程： RUP过程

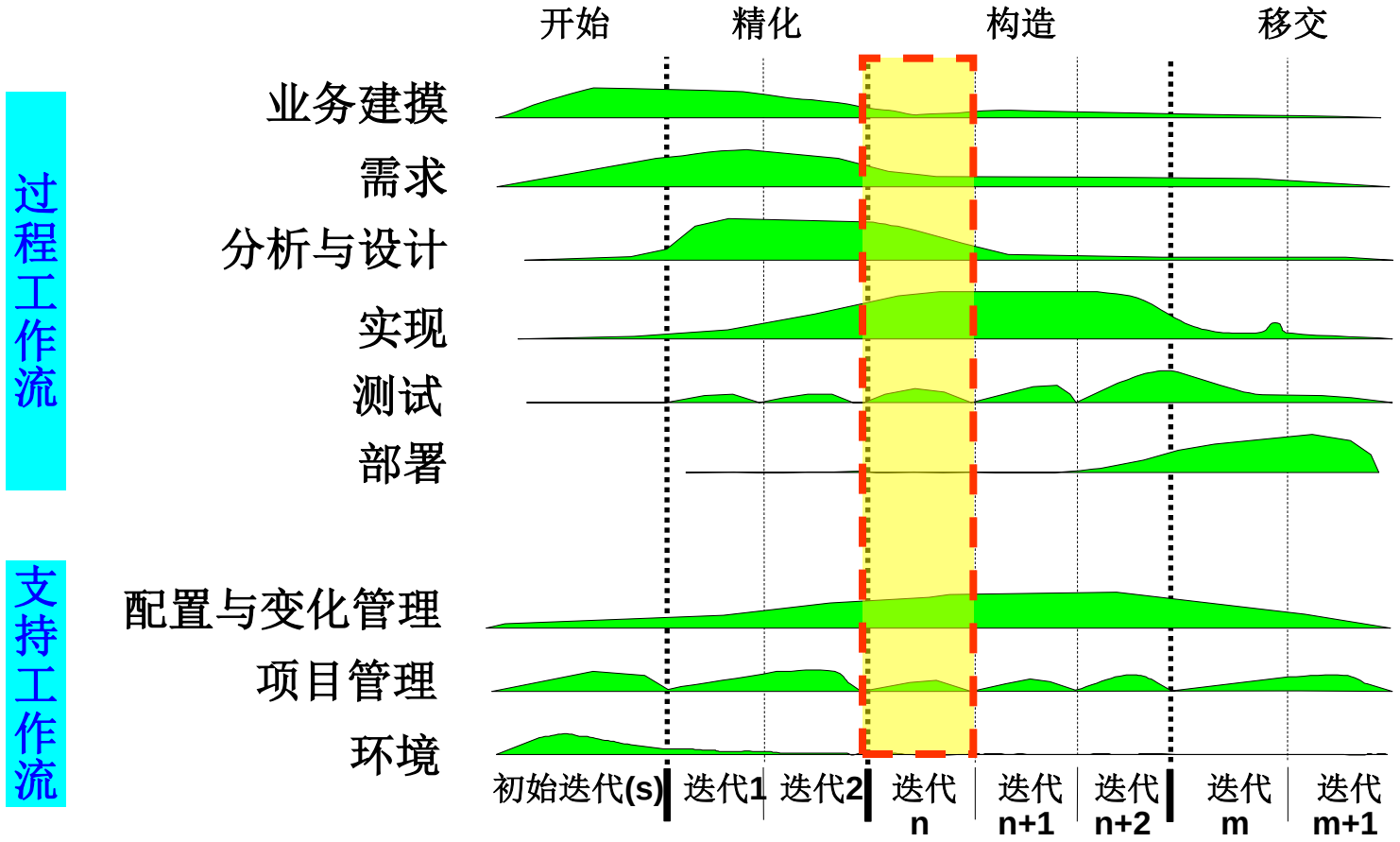



## 四个阶段

- 开始：建立业务模型，定义项目范围
- 精化：系统的体系结构、项目计划、资源需求、
- 构造：构件开发，软件产品
- 移交：软件产品移交给用户

# 软件过程： RUP过程

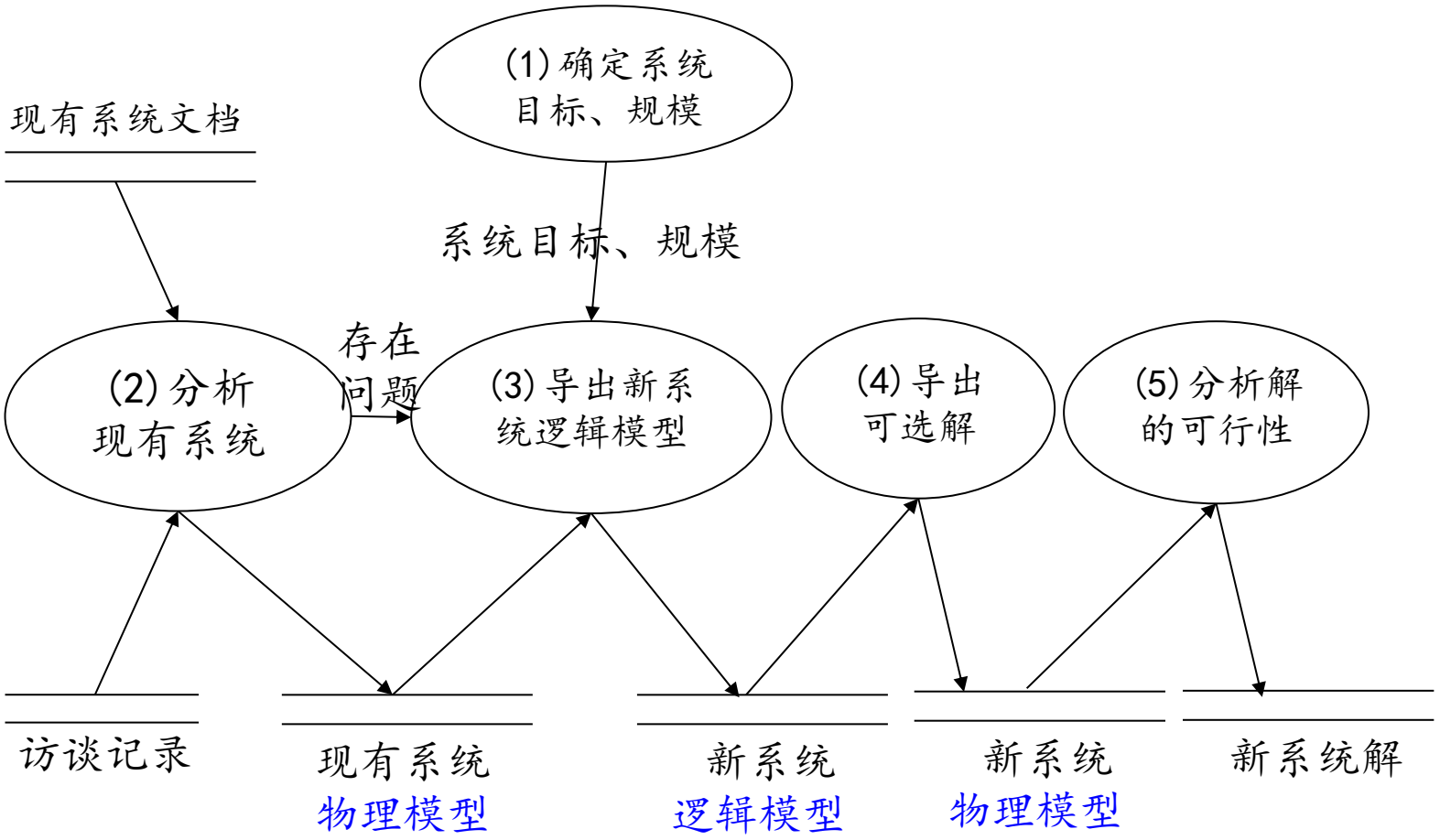
## 迭代与增量



# 可行性研究的任务

- 用最小代价在尽可能短的时间内确定问题是否能解决
- 不是解决问题，而是确定问题是否值得去解决
- 分析几种可能解法，每种解法的可行性
  - (1) 技术可行性：现有技术能实现系统吗?
  - (2) 经济可行性：经济效益能超过开发成本吗?
  - (3) 操作可行性：操作方式在组织内行得通吗?

# 可行性研究过程




物理模型（系统数据流图）：物理数据流，描述信息在物理元素之间的流动
逻辑模型（数据流图）：描述数据在软件中流动和被处理的逻辑

# 第3章　需求分析

- 需求分析的任务
  1. 回答“系统必须做什么?”。定义需求
  2. 结果：软件需求规格说明书。建立模型
- 软件需求
  - 用户解决问题或达到目标所需要的条件或能力
  - 需求层次：业务需求→用户需求→功能与非功能需求
- 具体任务
  - 确定综合要求

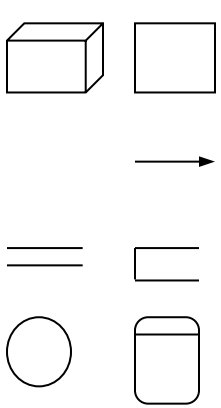
    功能需求、非功能需求（性能、可靠性、可用性、出错处理、接口、约束等）
  - 分析数据要求
  - 导出逻辑模型
  - 修正开发计划

# 第3章　需求分析

## ◆ 数据流图（DFD）

描绘数据在软件中流动和被处理的逻辑, 不涉及具体的物理部件

- 数据流图符号




- 根据描述画数据流图

# 第3章　需求分析

## ◆ 数据字典

- 对数据流图中的所有元素的定义
- 由数据元素组成数据的方式

（1）3种基本类型：顺序、选择、重复

（2）重复的特殊情况：可选

单位名称

| 商品名 | 数量 | 单价 | 金额 |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| 总金额 | | | |

日期　　营业员

发票＝单位名称＋{商品名＋数量＋单价＋金额}$_1^5$＋总金额＋日期＋营业员

# 第3章　需求分析

◆需求验证的四个方面

1. 一致性：　需求不互相矛盾
2. 完整性：　包括用户需要的每一个功能或性能
3. 现实性：　现有技术可以实现
4. 有效性：　确实能解决用户面对的问题

# 总体设计的任务

◆“概括地说，系统应该如何实现？”

- 系统设计
  - 划分系统：程序、文件、数据库、人工过程和文档等
- 结构设计
  - 划分程序：模块以及模块间的关系，建立软件结构

# 第5章 总体设计

## ◆ 设计原理

1. 模块化

   C（P1＋P2）＞ C（P1）＋ C（P2）　　C为复杂度

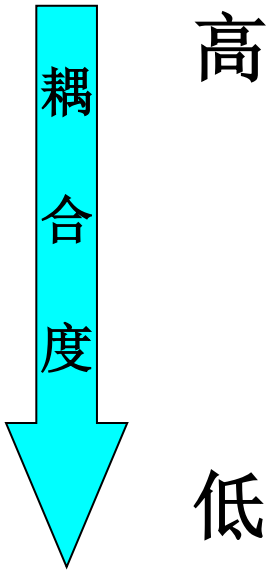
   E（P1＋P2）＞ E（P1）＋ E（P2)　　E为工作量

2. 抽象：关注事物的本质特性，暂不考虑细节
3. 逐步求精：推迟对问题细节的考虑，逐步增加细节
4. 信息隐藏：内部信息(过程和数据)对外不可访问
5. 局部化：把关系密切的元素放在一起
6. 模块独立程度的2个定性度量标准
   - 内聚：模块内各元素的相关程度
   - 耦合：模块间的交互程度

# 第5章 总体设计

- 耦合（含义）

1. 内容耦合
2. 共用（公共环境）耦合
3. 控制耦合
4. 印记（特征）耦合
5. 数据耦合



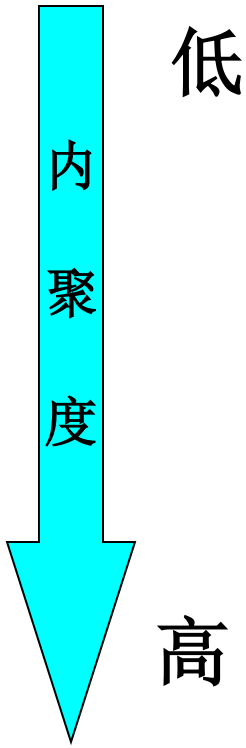

设计原则

多用数据耦合，少用控制耦合和特征耦合

限制公用耦合，不用内容耦合

# 第5章 总体设计

- 内聚（含义）
  1. 偶然性内聚
  2. 逻辑性内聚
  3. 时间性内聚
  4. 过程性内聚
  5. 通信性内聚
  6. 顺序内聚
  7. 功能性内聚
  8. 信息性内聚




低

高

设计原则

力争高内聚

识别低内聚，改进以提高内聚度

# 第5章 总体设计

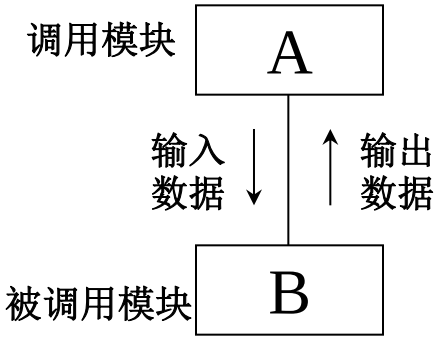
## ◆ 启发规则

1. 提高模块独立性
2. 模块规模适中
3. 深度、宽度、扇出和扇入都应适当
4. 模块的作用域应该在控制域之内
   - ➢ 模块的作用域：受一个判定影响的所有模块集合
   - ➢ 模块的控制域：模块及其直接或间接的从属模块集
5. 降低模块接口的复杂度
6. 设计单入口单出口的模块
7. 模块功能可以预测

# 第5章 总体设计

## ◆结构图

描述软件的模块层次，反映各模块之间的调用和数据传递




## ◆面向数据流的设计

- ✓ **DFD → 结构图**
- ✓ 两种信息流：变换流和事务流
- ✓ 变换流到软件结构图的转换

# 第6章 详细设计

- 任务
  - ✓怎样具体地实现这个系统?
  - ✓确定模块的处理过程:算法、数据结构
  - ✓详细设计的结果基本决定了最终程序的质量

# 结构程序设计

## ◆ 特点

1. 自顶向下逐步求精
2. 单入、单出的控制结构（正确使用goto）
3. 结构化定理

任何单入口单出口的程序都可以由“顺序”、“选择”和“循环”3种基本结构实现

# 第6章 详细设计

## ◆工具（会做）

### ●程序流程图

箭头的自由性，不能保证程序设计的结构化

### ●盒图（N-S图）

符合结构化原则，不可能任意转移控制

易表现嵌套关系以及模块的层次结构

### ●PAD图

符合结构化原则，不可能任意转移控制

### ●判定表

# 第7章 实现

## ◆ 编码风格

- ✓ 编码风格的作用就是使代码容易读
- ✓ 风格良好的代码更容易阅读和理解，错误更少

1. 使用一致和有意义的标识符名
2. 用缩进显示程序结构
3. 用加括号的方式排除二义性
4. 尽量不要进行浮点数的相等比较
5. 避免大量使用循环嵌套和条件嵌套
6. 利用sizeof()计算对象的大小
7. 程序的注释
   - 序言性注释和功能性注释
   - 对一段程序注释，而不是每一个语句
8. 用数据结束标记（EOF、BOF），而非数据数目来判断文件结束

# 第7章 实现

## ◆ 测试

- 测试对软件规格说明、设计和编码的最后复审
- 开发总工作量的40%以上
- 测试目的：发现软件中的错误
- 调试：诊断并改正错误
- 为提高测试效率，选择发现错误几率大的数据作为测试用例
- 好的测试方案是极可能发现迄今尚未发现的错误
- 成功的测试是发现了至今尚未发现的错误的测试
- 测试不能证明程序中没有错误
- Pareto原理：80%的错误很可能是20%的模块造成的
- 从“小规模”测试逐步到“大规模”测试
- 穷举测试不可能
- 应该由独立的第三方从事测试工作

# 第7章 实现

- 测试方法
  - 黑盒测试：又称功能测试或数据驱动测试
  - 白盒测试：又称结构测试或逻辑驱动测试
- 测试步骤
  - 模块测试（单元测试）：主要发现编码和详细设计的错误
  - 子系统测试
    - 模块放在一起形成一个子系统来测试
    - 着重测试模块的接口
  - 系统测试
    - 将子系统装配成一个完整的系统
    - 主要发现软件设计中的错误，也可能发现需求说明中的错误

集成测试

  - 验收测试（确认测试）
    - 验证软件的有效性（是否与用户合理期待相符）
    - 用户参与，使用实际数据进行测试
    - 主要发现系统需求说明书中的错误
  - 平行运行

# 第7章 实现

- 回归测试
  - 重新执行已经做过的测试的某个子集，以保证变化（程序改错、新模块加入等）没有带来非预期的副作用
- (α) Alpha测试

  在开发者的指导下，用户在开发现场进行
- (β) Beta测试

  用户在实际使用环境下进行测试

# 调试的困难

1） 症状和原因可能相距甚远

2） 当改正一个错误后，症状可能暂时消失

3） 症状可能不是由错误引起的（如舍入误差）

4） 症状可能是由不易跟踪的人为错误引起的

5） 症状可能是由定时问题引起的

6） 可能很难重新产生完全一样的输入条件

7） 症状可能时有时无

8） 症状可能是由分布在许多任务中的原因引起的

# 调试途径

- 蛮干法

  到处都写上WRITE（输出）语句，在信息海洋中发现错误线索

- 回溯法

  从症状开始，往回追踪分析源程序代码，直到找出错误原因为止

- 原因排除法

  - 对分查找法
    - ✓ 在程序中点附近“注入”变量的正确值，运行程序并检查所得到的输出
    - ✓ 根据输出判别错误是在上半部分还是下半部分
  - 归纳法
    - ✓ 分析和错误有关的数据，发现可能的错误原因
    - ✓ 导出对错误原因的假设，利用已有的数据来证明或排除这些假设
  - 演绎法
    - ✓ 设想出所有可能的出错原因，然后试图用测试来排除每一个假设的原因

# 单元测试

- ◆检测最小单元—模块
- ◆重点

  模块接口、局部数据结构、重要的执行通路、出错处理、边界条件
- ◆代码审查

  审查小组对代码进行复审，可发现30%～70%的错误
- ◆计算机测试
  - 驱动程序

    模拟被测模块的调用模块，接收测试数据，打印测试结果
  - 存根程序

    模拟被测模块的子模块，印出入口检验或操作结果

# 第7章 实现

## ◆集成测试方法

### ●非渐增式集成

- ➢先进行单元测试，所有模块按设计进行一次性集成

### ●渐增式集成

- ➢将单元测试与集成测试结合在一起
- ➢逐个将待集成的模块同已集成的那些模块结合起来进行测试
  - ✓自顶向下集成：不需驱动模块，需设计存根模块
  - ✓自底往上集成：不需存根模块，需设计驱动模块
  - ✓三明治式（混合式）集成

# 第7章 实现

## ◆几种集成测试方法的优缺点

| 方法 | 优点 | 不足 |
| --- | --- | --- |
| 非渐增式集成 | | 没有错误隔离手段<br>主要设计错误发现迟 |
| 自顶向下集成 | 错误隔离，不需驱动模块<br>较早发现主要设计错误 | 可复用模块得不到充分测试 |
| 自底往上集成 | 错误隔离，不需存根模块<br>可复用模块得到充分测试 | 主要设计错误发现迟 |
| 三明治式集成 | 错误隔离<br>较早发现主要设计错误<br>可复用模块得到充分测试 | |

# 第7章 实现

- 测试用例：测试数据+预期输出
- 白盒测试技术：逻辑覆盖

以程序内部逻辑结构为基础的设计测试用例的技术

1. **语句覆盖**：每个语句至少执行一次
2. **判定覆盖（分支覆盖）**：每个判定的每个分支至少执行一次
3. **条件覆盖**：每个判定中的每个条件都取到可能的值
4. **判定／条件覆盖**：每判定中的每个条件都取到可能值，且每个判定的每个分支都至少执行一次
5. **条件组合覆盖**：每个判定中的各种条件组合都至少执行一次

- 设计测试用例（会做）

# 第7章 实现

## ◆ 黑盒测试技术

### 1. 等价划分

- 把输入域划分成数据类，取每类中的一个典型值进行测试
- 设计测试用例

  (1) 新增测试用例尽可能多地覆盖尚未被覆盖的有效等价类

  (2) 新增测试用例覆盖一个尚未被覆盖的无效等价类

### 2. 边界值分析

- 着重测试输入等价类和输出等价类的边界，选取的测试数据应该刚好等于、刚刚小于和刚刚大于边界值。

● **会用等价划分和边界值分析法设计测试用例**

# 第8章 维护

使软件持久地满足用户的需要。

软件维护的本质是修改和压缩了的软件定义和开发过程。

- 占生命周期的60%以上

软件工程的一个重要目标：提高软件的可维护性，减少软件维护的代价。

- 4类维护
  - 改正性维护：诊断和改正错误。17%～21%
  - 适应性维护：适应环境的变化。 18%～25%
  - 完善性维护：增加新功能或修改已有功能。 50%～66%
  - 预防性维护：改进未来的可维护性或可靠性。4%左右

# 可维护性

- 理解、改正、改进软件的难易程度
- 提高可维护性是支配软件工程方法学的关键目标
- 决定软件可维护性的因素
  1）可理解性
  2）可测试性
  3）可修改性
  4）可移植性
  5）可重用性

# 第9章　面向对象方法学引论

## ◆面向对象方法学的4个要点

1) 软件系统是由对象（Object）组成

2) 对象组成对象类(Class)

3) 类可以组成一个类层次，子类继承(inheritance)父类

4) 对象间仅能通过传递消息(Message)互相联系

**OO=objects+classes+inheritance+communication with messages**
**面向对象 = 对象 + 类 + 继承 + 消息通信**

# 第9章　面向对象方法学引论

- 简化软件的开发与维护，提高软件的可重用性
- 一些概念
  - 对象：数据结构及这些数据结构上的操作的封装体
  - 类：具有相同数据和相同操作的一组相似对象的集合
  - 消息：要求某个对象执行某个操作的规格说明
  - 方法：对象所能执行的操作（服务）
  - 属性：类中所定义的数据
  - 封装：对象的状态数据、代码和局部数据，外面不能直接访问
  - 继承：子类共享基类中定义的数据和方法的机制
  - 多态性：同一方法，不同的子类有不同的实现。
  - 函数重载： 同一作用域内，相同函数名不同参数

# 第9，10章　面向对象方法学

## ◆面向对象建模

### ●对象模型

模拟客观世界实体的对象以及对象彼此间的关系的映射，描述系统的数据结构。类图，最基本的、核心模型

### ●动态模型：描述系统的控制结构

### ●功能模型：描述系统的功能

# 第9，10章　面向对象方法学

- 类图

描述类及类与类之间的静态关系（关联、聚集、泛化等）

A关联B：A和B的对象之间的某种语义上的联系

A是B聚集：B组成A，B是A的一部分

A是B的泛化：B是一种A

# 第9，10章　面向对象方法学

- 状态图

  描绘对象状态及引起状态转换的事件

- 用例图

  描述外部行为者所理解的系统功能

- 事件跟踪图/顺序图/序列图

  描述一组交互对象间的动态协作关系，表示完成某项行为的对象和对象间传递消息的时间顺序

# 第11章　面向对象设计

◆任务（概要设计、详细设计）

- 系统设计：确定实现系统的策略和目标系统的高层结构
- 对象设计：确定解空间中的类、关联、接口形式及实现服务的算法

# 第11章　面向对象设计

## ◆面向对象设计的准则

- 模块化、抽象、信息隐藏
- 耦合

(1) 交互耦合：对象之间的耦合通过消息连接来实现

减少消息中包含的参数个数，降低参数的复杂程度

减少对象发送(或接收)的消息数

(2) 继承耦合：一般化类与特殊类之间耦合

使特殊类尽量多继承并使用其一般化类的属性和服务

- 内聚

（1） 服务内聚

一个服务应该完成一个且仅完成一个功能

（2） 类内聚

类的属性和服务应该全都是完成该类对象的任务所必需的

（3） 一般-特殊内聚

一般-特殊结构应该是对相应的领域知识的正确抽取

# 第11章　面向对象设计

- 面向对象设计的启发规则
  - 设计清晰易懂
  - 一般-特殊结构深度适当
  - 设计简单的类
    - 任务明确简单
    - 属性和服务不要过多，简化对象间协作
  - 使用简单的协议，消息参数少
  - 使用简单的服务

# 第11章　面向对象设计

- 重用（再用或复用）

  同一事物不修改或稍加改动后可重复使用

- 软件重用层次
  - 知识重用。如，软件工程知识的重用
  - 方法和标准的重用。如，开发方法或国标
  - 软件成分的重用
    - ✓代码重用：剪贴、包含、继承等；
    - ✓设计结果重用：实现变化（如移植），设计基本不变
    - ✓分析结果重用：体系结构变化，需求基本不变

# 第11章　面向对象设计

- 类构件的重用方式

(1) 实例重用

- 按需创建类实例
- 用对象成员创建出更复杂的类

(2) 继承重用

- 为提高继承重用的效果，关键是设计一个合理的、具有一定深度的类构件继承层次结构。

(3) 多态重用

- **转换接口**：重用必须重新定义的服务的集合
- **扩充接口**：如派生类没给出接口的新算法，则继承父类算法