分析题四个

简答题七个，题量不大

第一章

- **什么是软件工程、软件工程研究什么**、什么是软件
  - 软件工程

    软件工程是一门研究用工程化方法构建和维护有效的、实用的和高质量的软件的学科。把工程学的基本原理和方法引进到软件的设计和生产中，研制软件设计和生产的方法和工具
  - 什么是软件
    - 运行时能提供所要求功能和性能的指令或计算机程序集合
    - 程序能满意地处理信息的数据结构
    - 描述程序功能需求及程序如何操作和使用所要求的文档
  - 软件工程研究什么

    工序、规范、质量、工具、人
- **软件生命周期有哪些阶段，每个阶段做什么**
  - **问题定义**：要解决的问题是什么，给出 **问题性质报告、工程目标和规模报告、访问调查**
  - **可行性研究：** 对上一个阶段所定义的问题找到行得通的解决办法，导出系统的高层次抽象模型，得到更具体的工程规模和目标，估计成本和效益
  - **需求分析：** 为了解决问题，目标系统必须做什么，
  - **总体设计：** 如何解决这个问题
  - **详细设计：** 解法具体化，
  - **编码和单元测试：** 书写程序，测试编写的每个模块
  - **综合测试：** 集成测试、验收测试
  - **软件维护：** 改正性维护、适应性维护、完善性维护、预防性维护
- **软件工程诞生的原因：软件危机**
  - 什么是软件危机，为什么会爆发软件危机
    - 软件危机是计算机软件开发和维护过程中遇到的一系列严重问题
      - 怎样满足对软件日益增长的需求
      - 怎样维护数量不断膨胀的已有软件
    - **软件危机爆发的原因**
      - 软件本身的特点：逻辑部件规模庞大
      - 不正确的开发和维护方法：忽视需求分析、软件开发定义为程序编写，轻视软件维护
- 软件工程的两个主线
  - 实用化，两种有些差别
    - 面向对象
    - 结构化
      - 需求分析：数据流图
      - 怎么样转变为结构模型

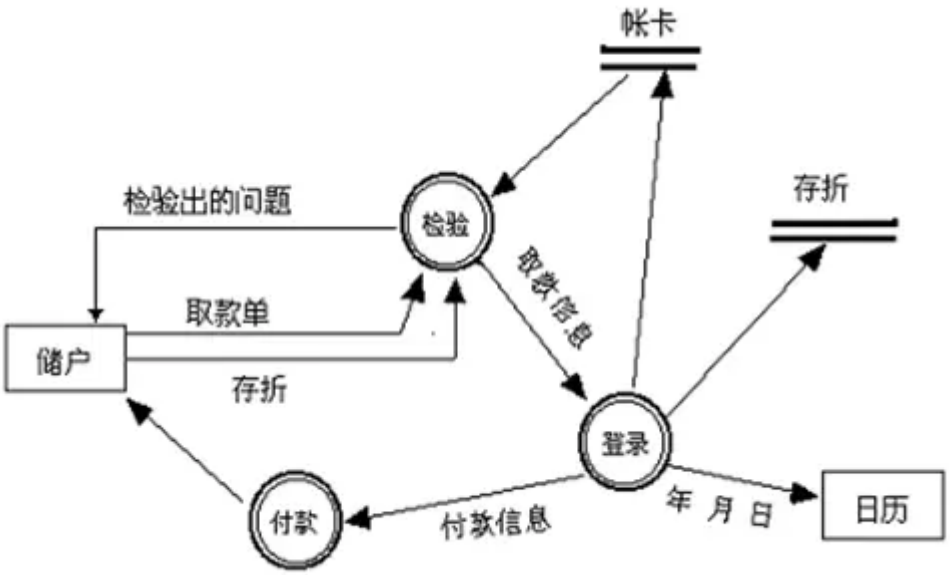
    - 形式化开发方法：采用严格的数学语言，具有精确的数学语义的方法

- 软件模型，
    - **瀑布模型**：合格进入下一阶段，否则进入前一阶段，有什么问题，解决了什么问题？
    - **增量模型**：构造一系列的可执行中间版本
    - **快速原型模型**：快速构建可以运行的原型系统
    - 螺旋模型
    - **喷泉模型**：多次重复演进，个阶段之间没有明显的界限
    - **智能模型**：基于知识的软件开发模型，建立知识库、专家系统

  要有所了解

第二章 可行性分析

- 可行性方法是从哪四个方面进行分析的
    - 技术可行性
    - 经济可行性
    - 操作可行性
    - 法律可行性
- 描述底层软件结构的 **数据流图**



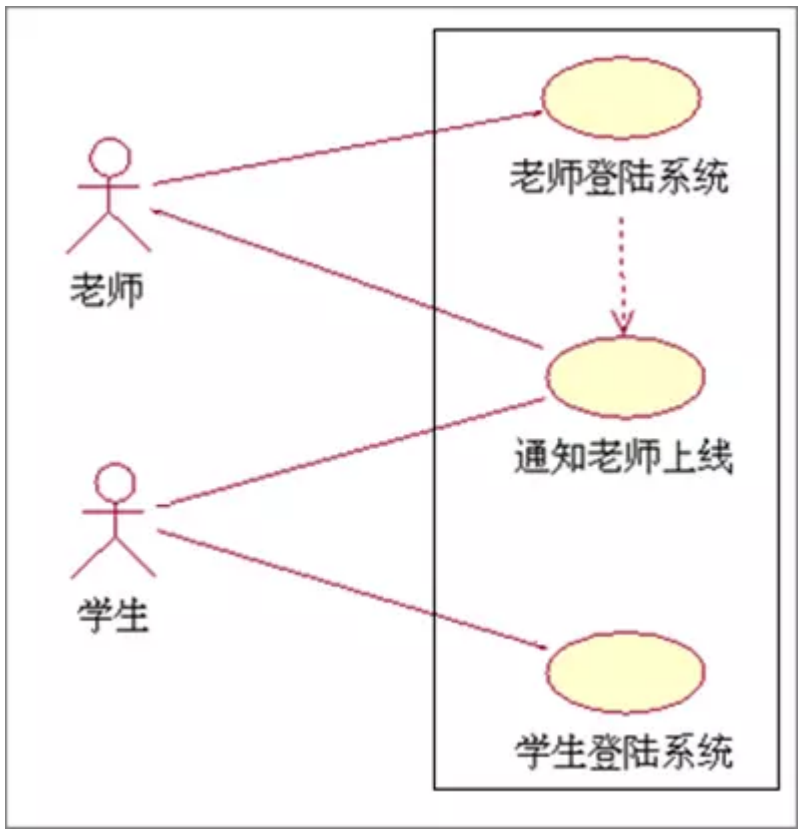
    - 圆圈：加工
    - 方框：数据输入或输出
    - 箭头：数据流
    - 双横线：数据存储文件

第三章 需求分析

需求分析是软件定义时期的最后一个阶段，他的基本任务是确定系统必须完成那些工作，对目标系统提出完整、准确、清晰、具体的要求，并在需求分析结束之前，写出软件需求规格说明书。

- 面向对象的方法
  - **用例图**

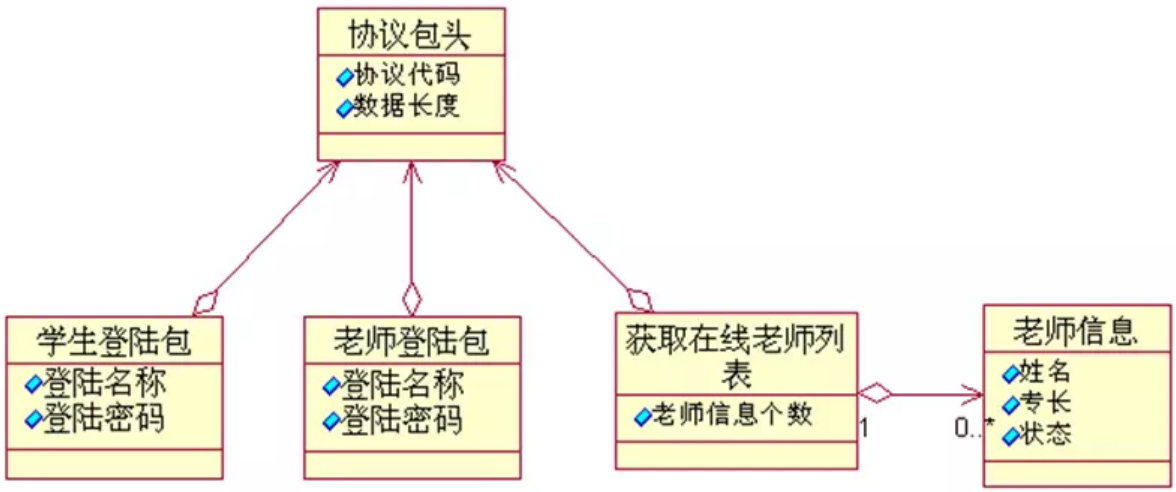
    用例图描述系统提供的功能单元、



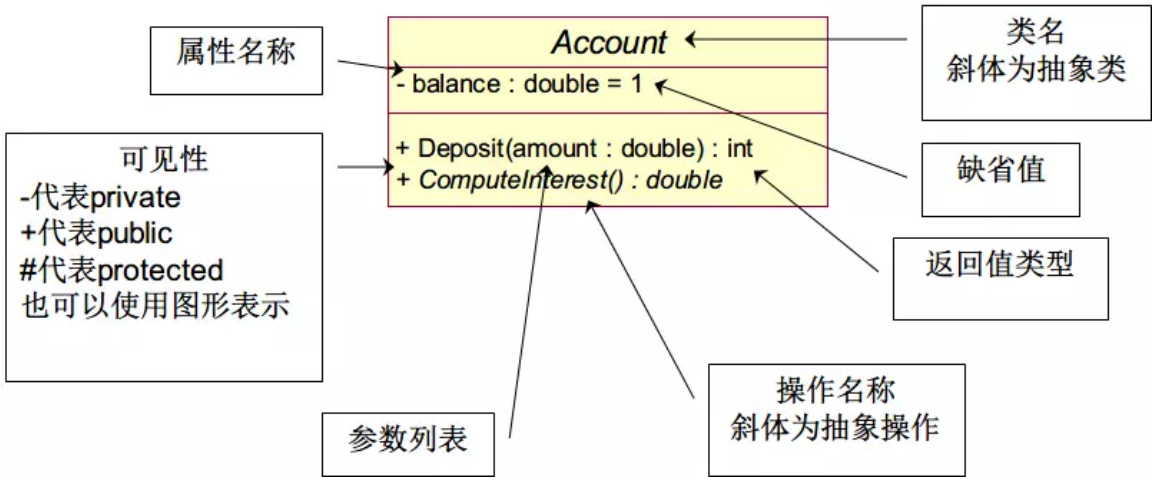
    使用有系统名称标签的方框表示系统的边界，在系统边界外部的表示参与者，内部为组成系统行为的用例，参与者和用例之间的关系使用 **实线** 表示

  - **类图**

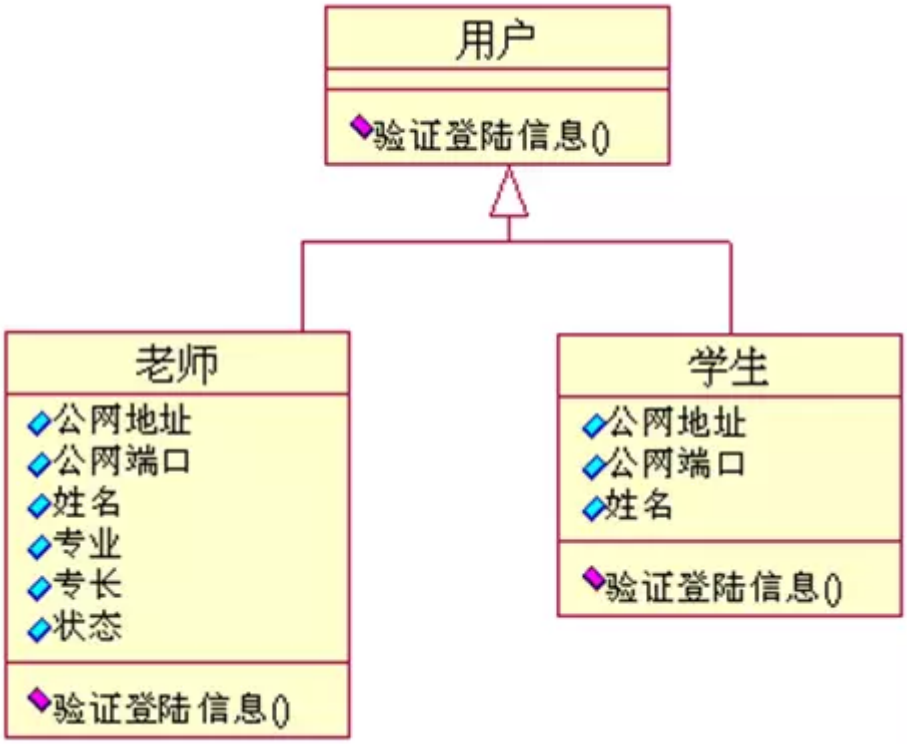
    表示不同的实体：人，事物，数据之间的关系，显示了系统的静态结构



    - 类：三段式方框表示

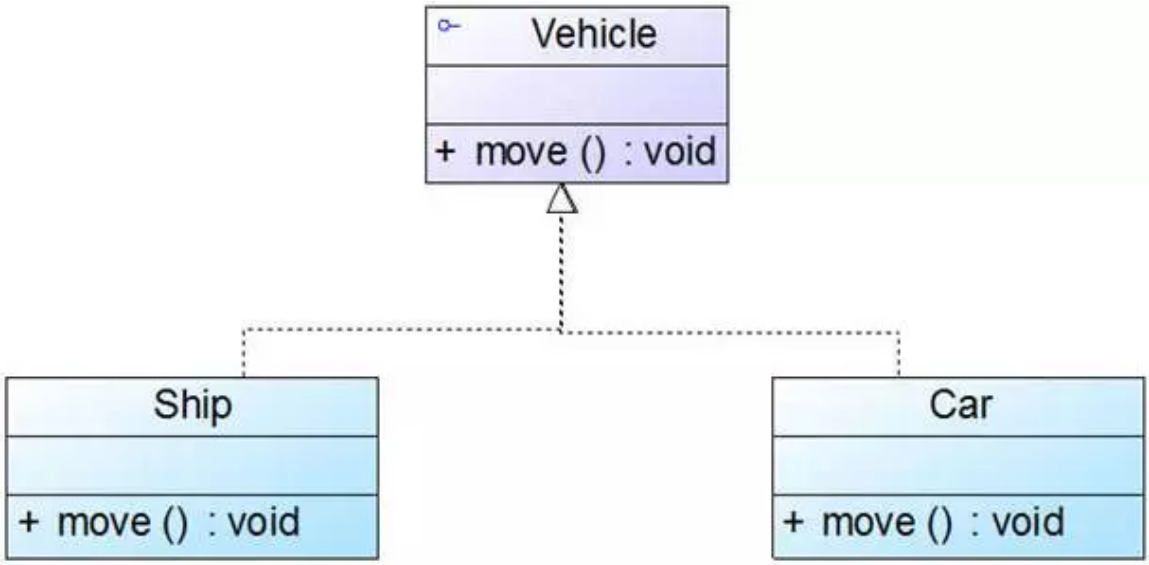


- 泛化关系：

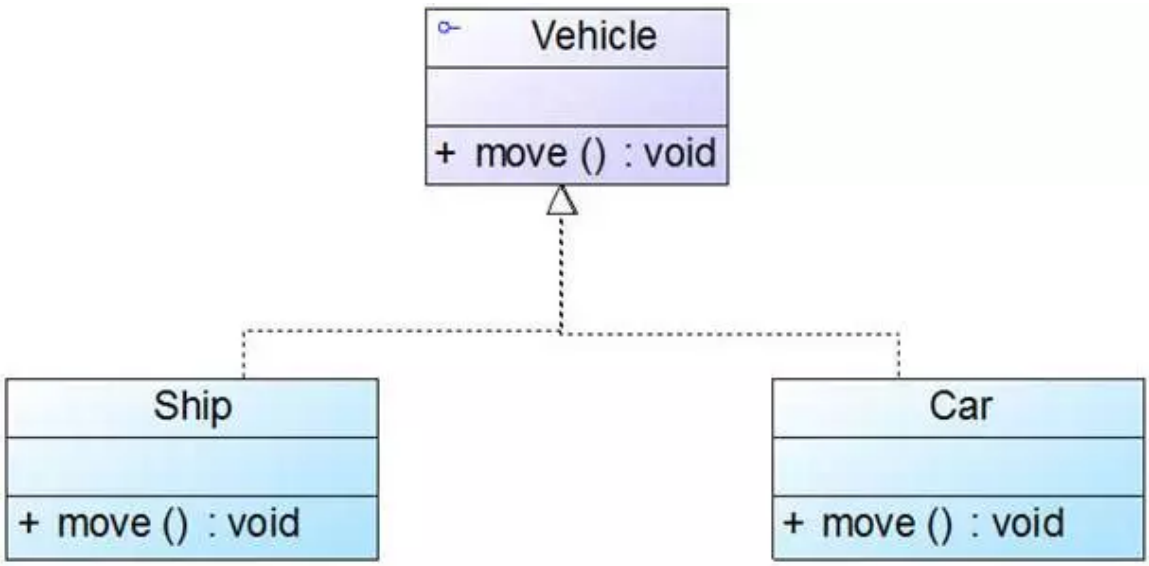


空心三角箭头要向上指

- 实现关系



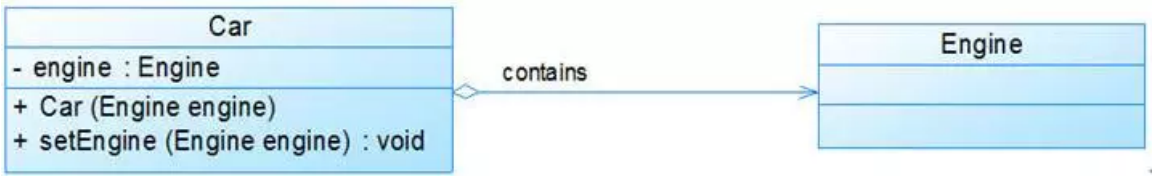
- 依赖关系



> 特定事物的改变有可能会影响到使用该事物的其他事物，在需要表示一个事物使用另一个事物时使用依赖关系。大多数情况下，依赖关系体现在某个类的方法使用另一个类的对象作为参数。在UML中，依赖关系用带箭头的虚线表示，由依赖的一方指向被依赖的一方。例如：驾驶员开车，在Driver类的drive()方法中将Car类型的对象car作为一个参数传递，以便在drive()方法中能够调用car的move()方法，且驾驶员的drive()方法依赖车的move()方法，因此类Driver依赖类Car，如图1所示：

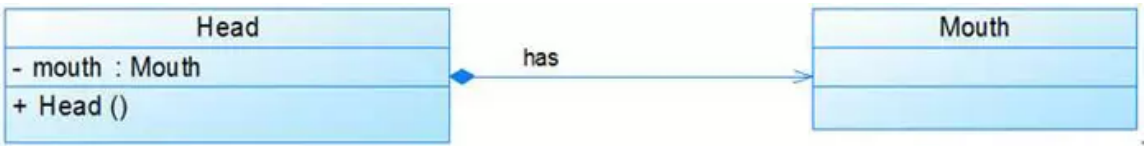
- 聚合关系

> 聚合(Aggregation)关系表示整体与部分的关系。在聚合关系中，成员对象是整体对象的一部分，但是成员对象可以脱离整体对象独立存在。在UML中，聚合关系用带空心菱形的直线表示。例如：汽车发动机(Engine)是汽车(Car)的组成部分，但是汽车发动机可以独立存在，因此，汽车和发动机是聚合关系，如图6所示：



- 组合关系

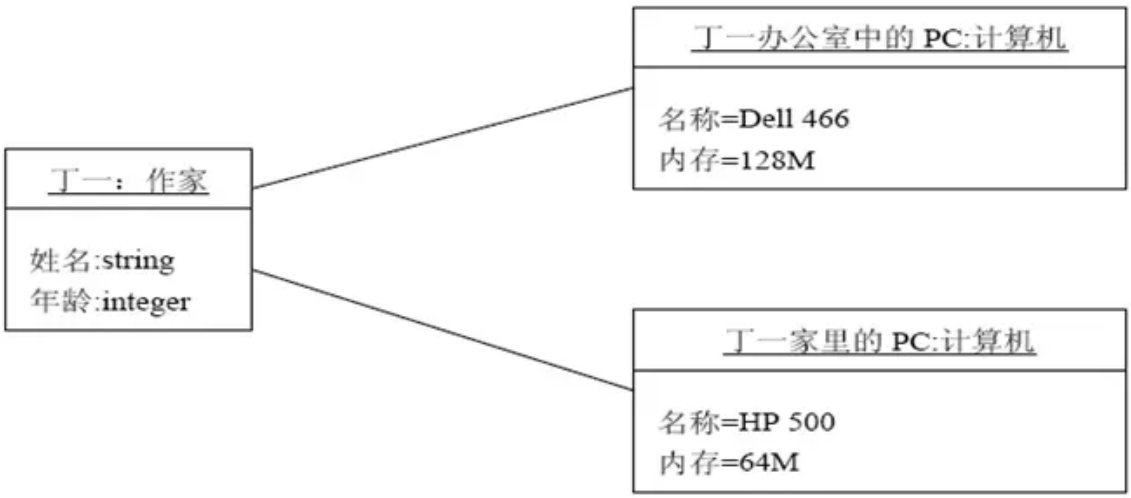
  > 组合(Composition)关系也表示类之间整体和部分的关系，但是在组合关系中整体对象可以控制成员对象的生命周期，一旦整体对象不存在，成员对象也将不存在，成员对象与整体对象之间具有同生共死的关系。在UML中，组合关系用带实心菱形的直线表示。例如：人的头(Head)与嘴巴(Mouth)，嘴巴是头的组成部分之一，而且如果头没了，嘴巴也就没了，因此头和嘴巴是组合关系，如图7所示：



- 对象图

  对象图和类图具有相同的表示形式，对象图是类图的实例，差别在于对象的名字下面要加下划线

(a) 类图



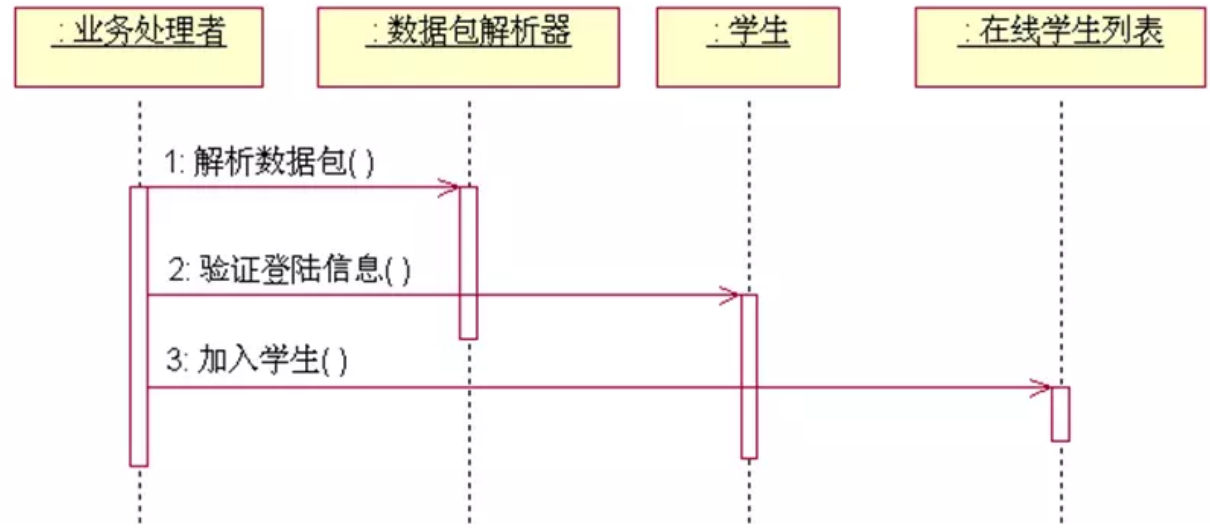
(b) 对象图

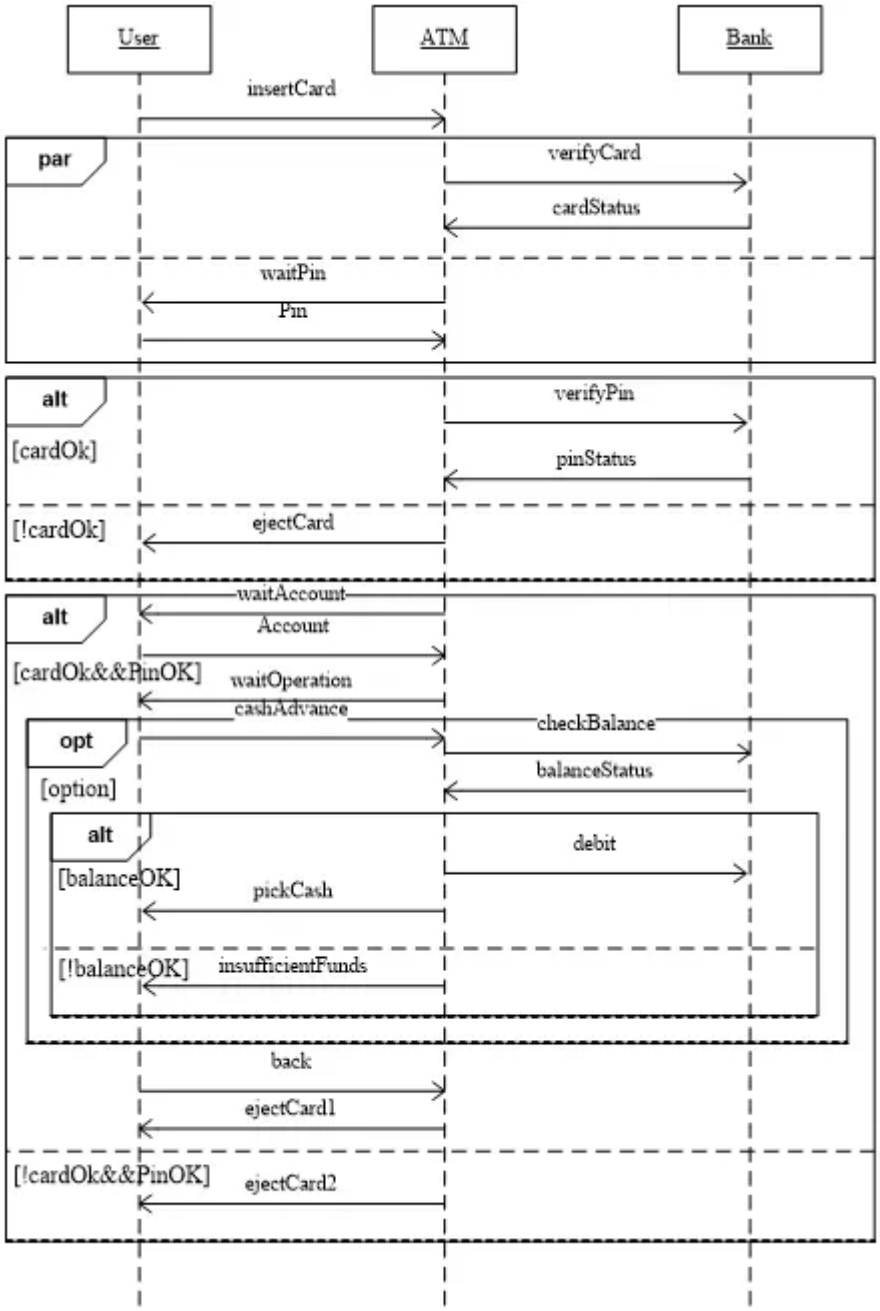
- 结构化方法
  - 延续使用数据流图
- **整个需求分析的过程分三个方面**
  - 获取和理解用户需求
  - 描述分析用户需求
  - 对用户需求进行评审
- 要理解怎么做好需求分析

第四章 统一建模语言UML

- **描述交互过程的模型**
  - 序列图 (顺序图)

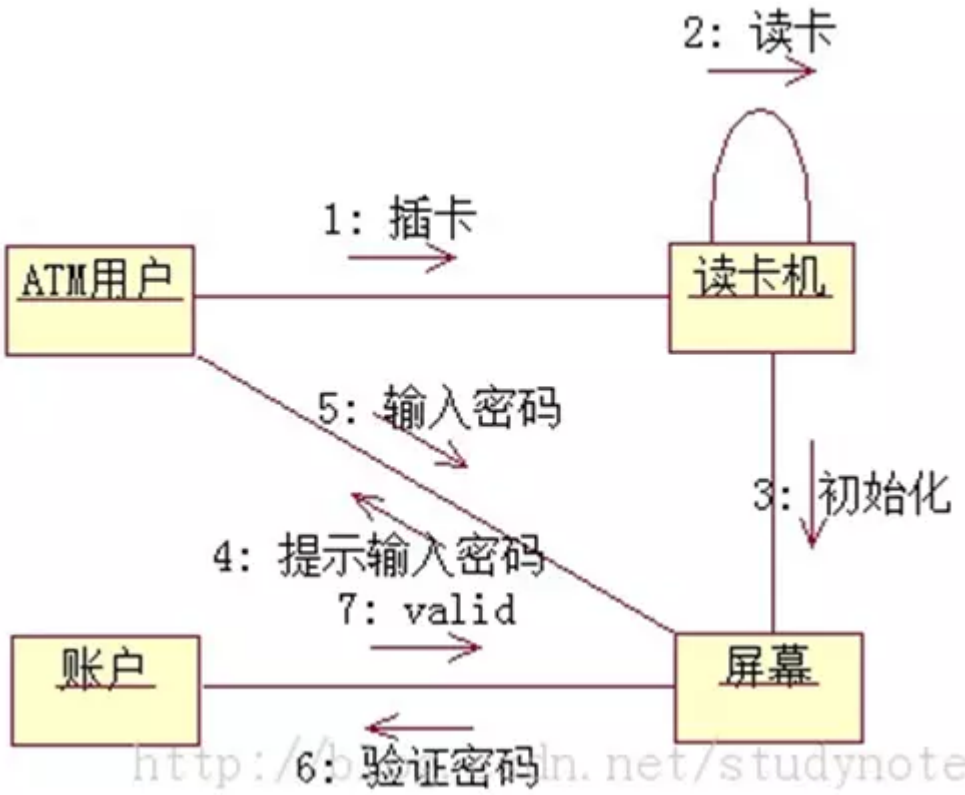
    顺序图用来描述对象之间动态的交互关系，着重体现**对象间消息传递的时间顺序。**自上而下





- 协作图

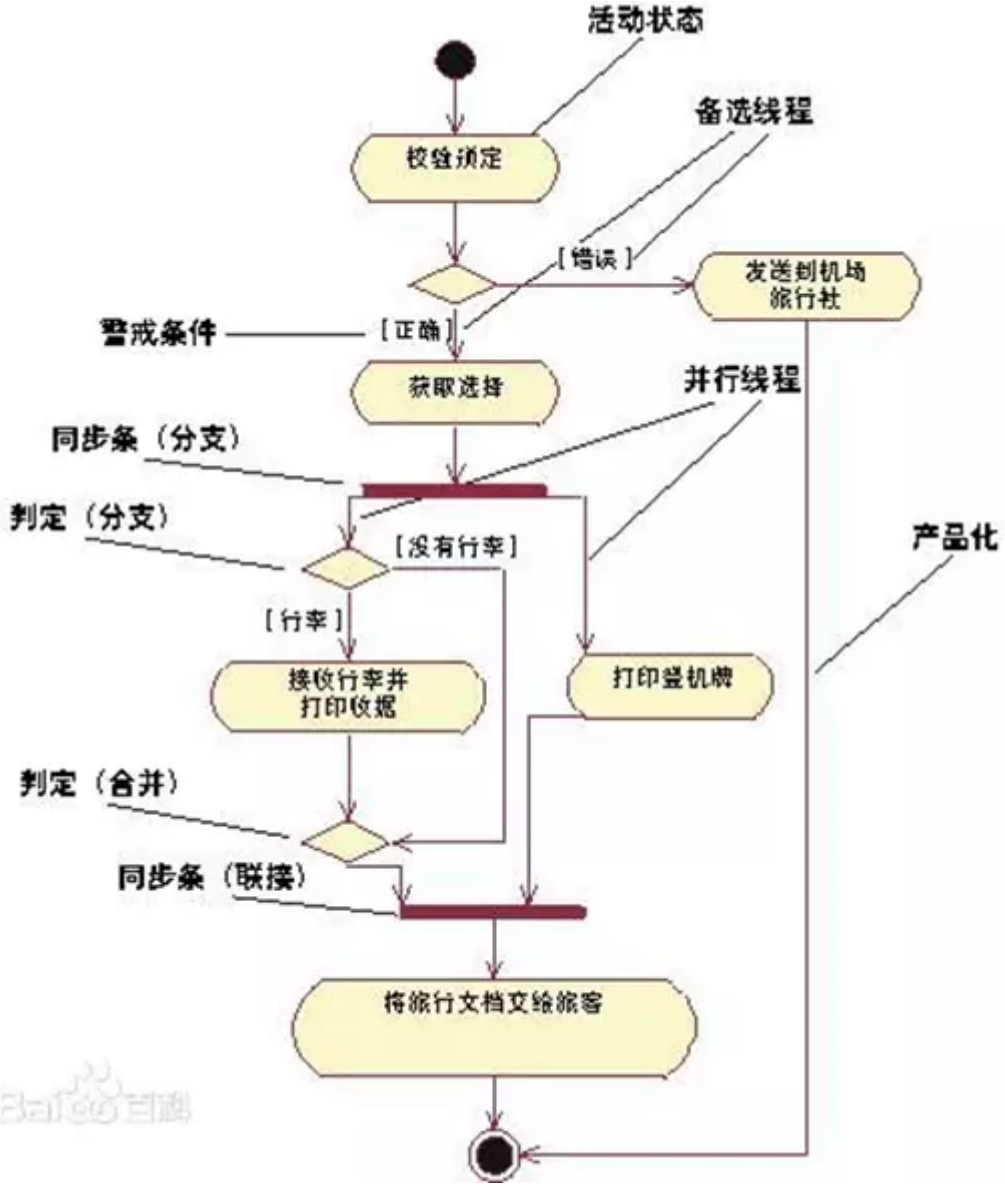
  描述相互合作的对象间的交互关系和链接关系，协作图着重体现交互对象间的静态链接关系



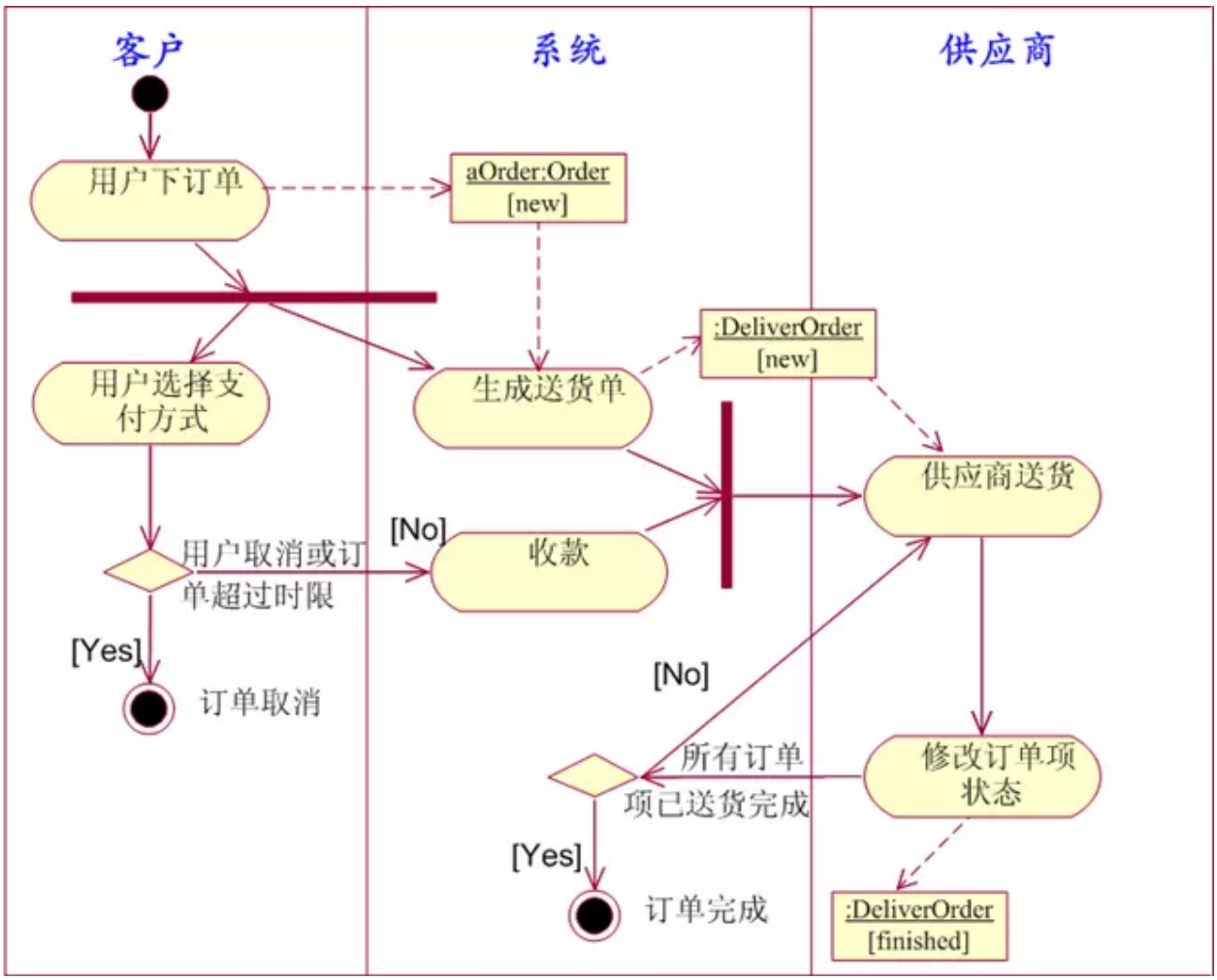
他两之间的区别：顺序：交互的顺序，协作：对象的交互，如果要表示执行顺序，需要用数字标识

- 活动图、

  活动图描述工作的流程，对并行的工作流能很好的支持



- **用泳道来区分分支和循环**



- 多个用户来交互的时候，一定是带泳道的

第五章 总体设计

- 启发式规则
  - 模块之间的耦合性要低
  - 内聚性要高

第六章 详细设计

- 要会画程序流程图 非常重要
  - 程序流程图的作用
    - 可以转换为盒图和 PAT图 **必须要会**
    - 退化为流图，计算环形复杂度，确定程序按路径测试的路径数
- 判定树和判定表有所了解，要看一下
- 伪码：
  - 如果害怕看不懂，要知道描述的规则，要能看懂伪码

第七章 面向对象设计

- 要会使用 UML
- 活动、顺序、协作、用例图、类图
- **面向对象设计的六条准则**
- 了解相关的概念，核心要掌握怎么使用面向对象的方法构建模型

第八章 软件实现

- 重要的是测试
  - 测试的环节
    - 单元、集成、系统、确认 逐层递进
    - **每个环节测试的是什么，用什么方法 （白盒黑盒）**
    - 看懂ppt里面的例子
  - **alpha、beta 测试**
- 要会白盒测试
  - 覆盖能力的五种
  - 设计测试用例去测试
- 黑盒测试
  - 等价类划分
    - 有效等价类、无效等价类
    - 选取测试数据时要在边界值上选取有特征的
  - 路径测试
    - 流图计算环形复杂度，三种方法
      - 区域数，不要漏掉外面的区域
    - 判断有多少条通路，针对每条通路测试

最后两章

- 软件维护有哪些类别
  - 三种被动型，一种主动型
- 软件配置管理管什么，什么能构成基线
  - 管理计算机程序、描述计算机程序的文档、数据
  - 基线是通过了正式复审的软件配置项

总体设计

总体设计要确定软件整体结构，

总体设计有 9 个步骤 （采用结构化方法）

1. 设想供选择的方案

   > 根据需求分析阶段得出的数据流图考虑各种可能的实现方案

2. 选取合理的方案

   > 对每个方案准备这些资料
   >
   > - 流程图
   > - 组成系统的物理元素清单
   > - 成本效益分析
   > - 实现这个系统的进度计划

3. 推荐最佳方案
4. 功能分解

   重要：内聚性和耦合性

   内聚性和耦合性的强弱关系

   结构设计 是总体设计阶段的任务
5. 设计软件结构

   软件结构可以用层次图或结构图来描绘 （考）

   可以直接通过数据流图映射出软件结构，这是 *面向数据流的设计方法*
6. 设计数据库
7. 制定测试计划
8. 书写文档 ： 用户手册测试计划、实现计划
9. 审查和复审

设计原理

1. 模块化

模块：边界元素限定的相邻程序元素的序列，有一个总体标识符表示他

模块化：把程序划分成独立命名且可独立访问的模块，每个模块完成一个子功能，把这些模块集成起来构成一个整体，可以完成指定的功能，满足用户的需求

- 使得一个复杂的大型程序能被人的智力所管理

对于每个程序都有一个最适应的模块数，使得系统的开发成本最小

2. 抽象

现实世界中的事物，状态，过程中间有共性，

- 一般抽象过程
  - 用层次的方法构造和分析复杂系统
  - 一个复杂的动态系统首先可以用高级的抽象概念构造和理解，高级概念可以种低级的概念构造
- 软件工程的抽象过程
  - 抽象到精化

3. 逐步求精

- **尽量推迟对问题细节的考虑**
- 忽略目前还不需要考虑的细节，每个问题在适当的时候被解决

4. 信息隐藏和局部化

- **信息隐藏：** 模块内包含的信息，对于不需要这些信息的模块来说，是不能访问的
- **局部化：** 把一些关系密切的软件元素，物理地放的更近

5. 模块独立

- 模块独立是 模块化、抽象、信息隐藏和局部化 概念的直接结果
- 模块独立程度的两个度量：

  耦合、内聚

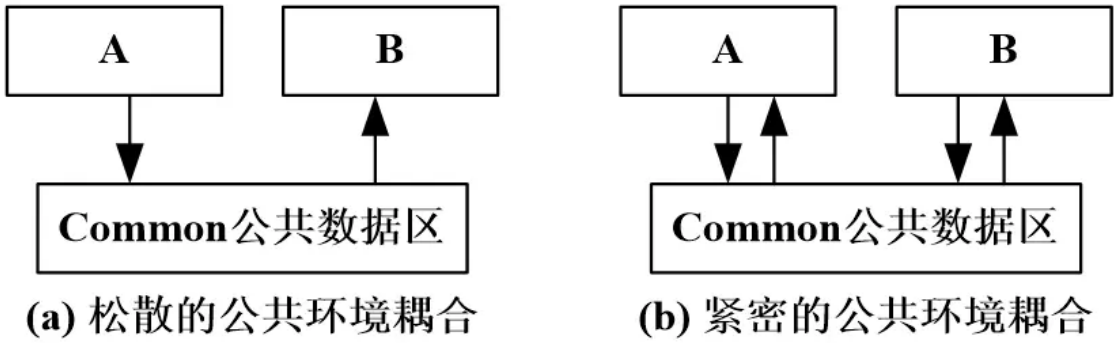
  耦合是什么、内聚是什么，有哪几种关系

  两个概念都有六个评价的不同类型
  - 我们期望模块和模块之间的关系越简单越好
  - 我们期望耦合低，内聚高，每个模块完成一个相对独立的子功能

  > 耦合：是对一个软件结构内不同模块之间互连程度的度量。 要求：在软件设计中应该追求尽可能松散耦合的系统。 可以研究、测试或维护任何一个模块，而不需要对系统的其他模块有很多了解； 模块间联系简单，发生在一处的错误传播到整个系统的可能性就很小； 模块间的耦合程度强烈影响系统的可理解性、可测试性、可靠性和可维护性。

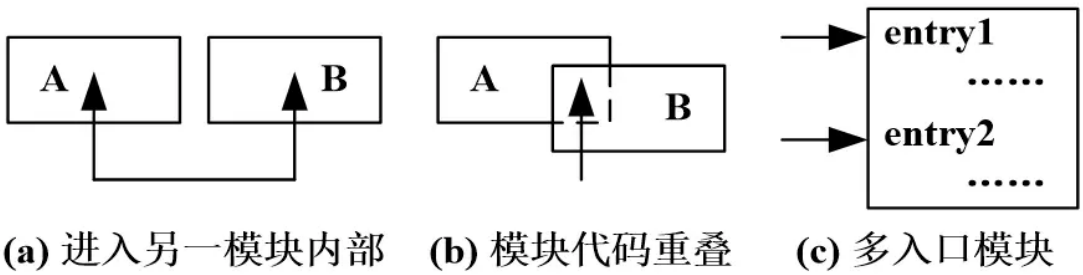
  - 耦合程度的度量 （六种耦合关系）
    - 非直接耦合，完全独立：两个模块每一个都能独立工作，不需要另一个模块的存在
    - **数据耦合** 两个模块之间通过参数来交换信息，并且交换的信息仅仅是数据

      （系统中至少必须存在这种耦合，数据耦合是理想的目标）
    - **控制耦合** 两个模块彼此传递的信息含有控制信息 (比如传递后会作为 case 条件，执行通路的判断)
    - **特征耦合** 当把整个数据结构作为参数传递而被调用的模块只需要使用其中一部分数据元素时，就出现了特征耦合。
    - **公共环境耦合** 当两个或多个模块通过一个公共数据环境相互作用时，它们之间的耦合称为公共环境耦合。公共环境可以是全程变量、共享的通信区、内存的公共覆盖区、任何存储介质上的文件、物理设备等等。



- **内容耦合** 最高程度的耦合是内容耦合。如果出现下列情况之一，两个模块间就发生了内容耦合：

  一个模块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。



  尽量使用数据耦合，

  少用控制耦合和特征耦合，

  限制公共环境耦合的范围， 完全不用内容耦合。

- 内聚

> 标志一个模块内各个元素彼此结合的紧密程度，它是信息隐藏和局部化概念的自然扩展。简单地说，理想内聚的模块只做一件事情。 要求：设计时应该力求做到高内聚，通常中等程度的内聚也是可以采用的，而且效果和高内聚相差不多；但是，低内聚不要使用。

  - 偶然内聚

    如果一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的，就叫做偶然内聚。

  - 逻辑内聚：

    如果一个模块完成的任务在逻辑上属于相同或相似的一类 *但是这些任务互不影响，各自独立*，则称为逻辑内聚。

  - 时间内聚

    如果一个模块包含的任务必须在同一段时间内执行，就叫时间内聚。

  - 过程内聚

    如果一个模块内的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。

  - 通信内聚

    如果模块中所有元素都使用同一个输入数据和(或)产生同一个输出数据，则称为通信内聚。即在同一个数据结构上操作。

  - 顺序内聚

如果一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行，则称为顺序内聚。

- 功能内聚

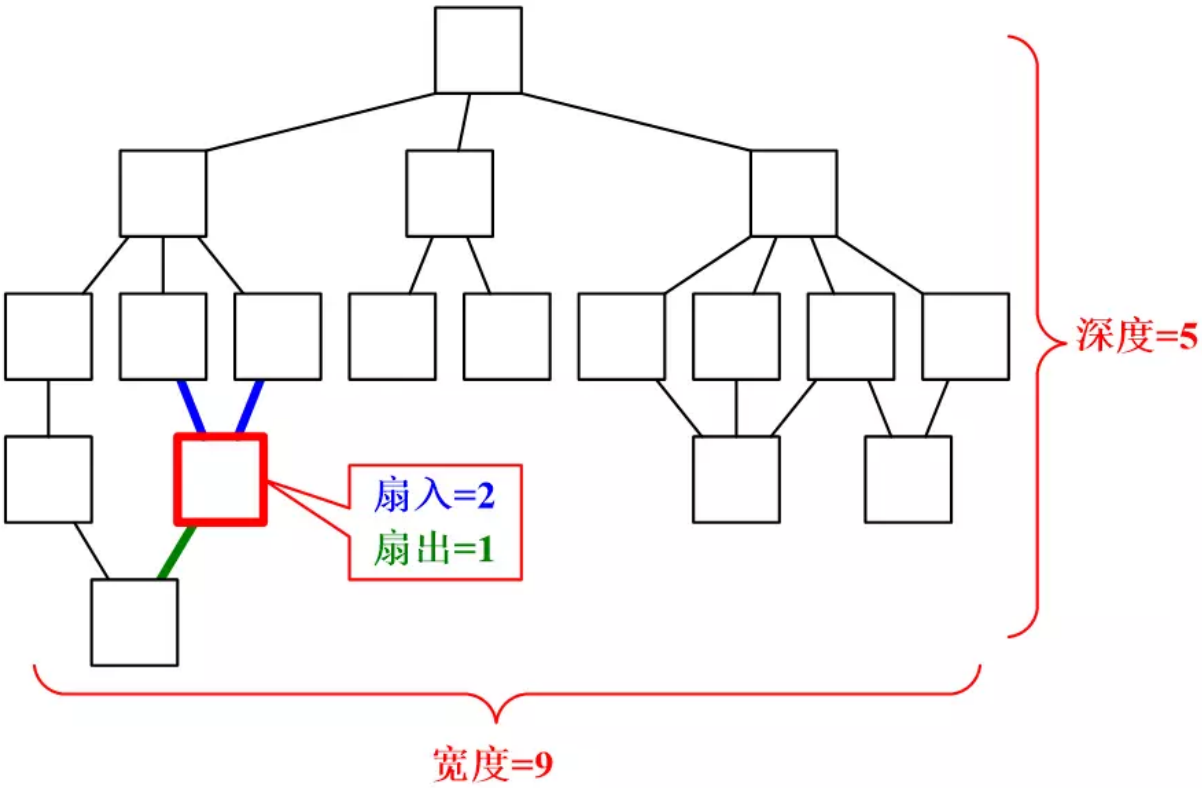
  如果模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。功能内聚是最高程度的内聚。

启发规则

1. 改进软件结构，提高模块独立性

   通过模块分解或合并，降低耦合提高内聚。

   - 模块功能完善化
   - 消除重复功能，改进软件结构
2. 模块规模应该适中
   - 过大的模块分解不充分，分解时要保证模块的独立性
   - 过小的模块开销大于有效 操作，而且模块的数目过多使得系统的接口复杂
3. 深度、宽度、扇出、扇入适当
   - 扇出：模块直接调用的模块数目
   - 扇入：有多少上级模块直接调用它



4. 模块的作用域应该在控制域内

   模块的作用域：定义为受该模块内一个判定影响的所有模块的集合。 模块的控制域：是这个模块本身以及所有直接或间接从属于它的模块的集合

   解决：

   - 上移，提高控制域
   - 下移，进入控制域
5. 降低模块接口的复杂程度

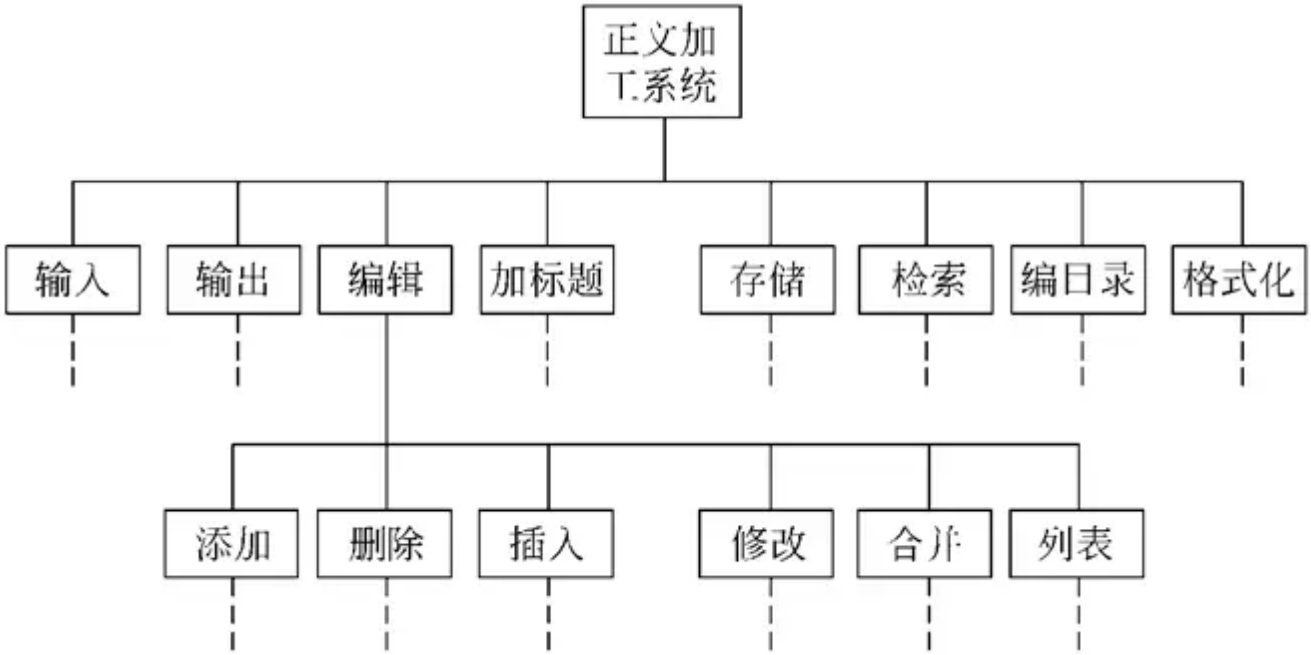
6. 设计单入口单出口的模块
7. 模块的功能应该可以预测 （只要输入的数据相同，就产生同样的输出

描绘软件结构的图形工具

怎么样描绘软件结构，如果面向过程的设计方法，要用层次图和 HIPO 图

层次图

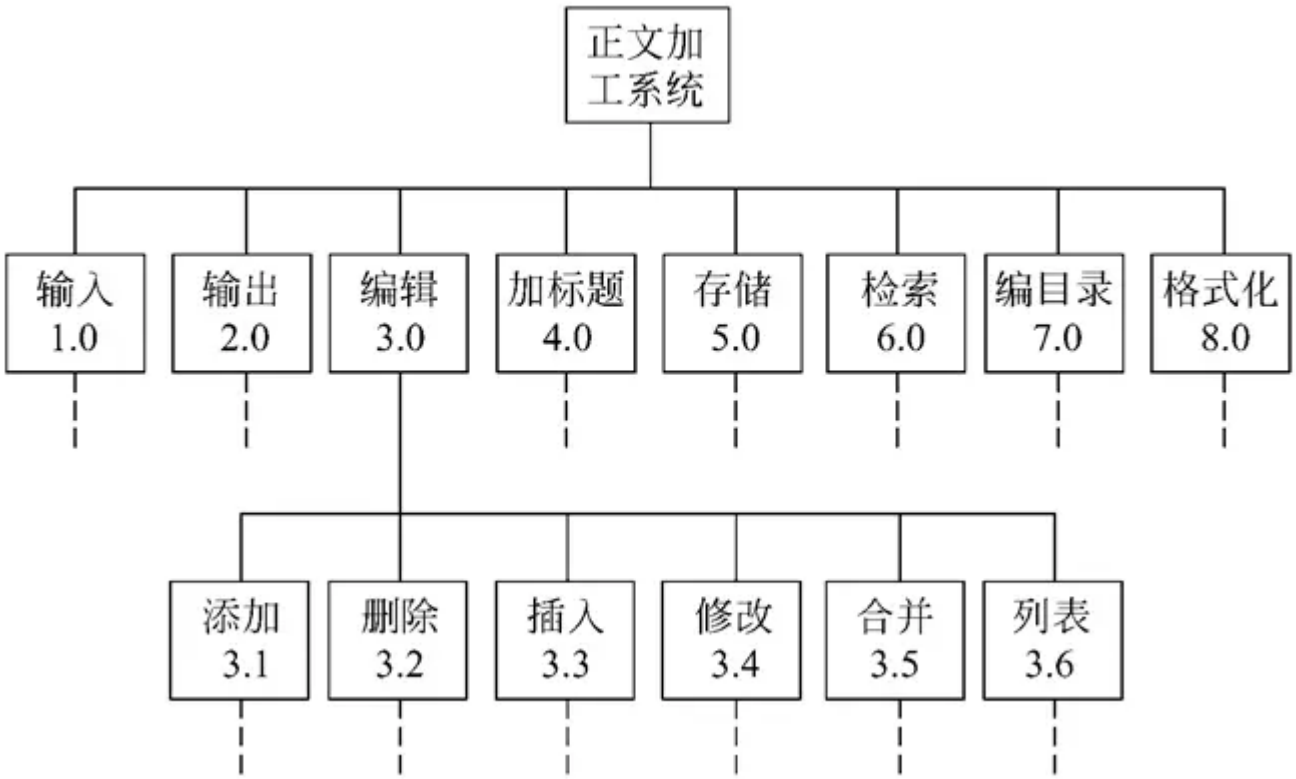
层次图用来描绘软件的层次结构。很适于在自顶向下设计软件的过程中使用。



HIPO

在层次图的基础上，加入了输入/处理/输出

H图：

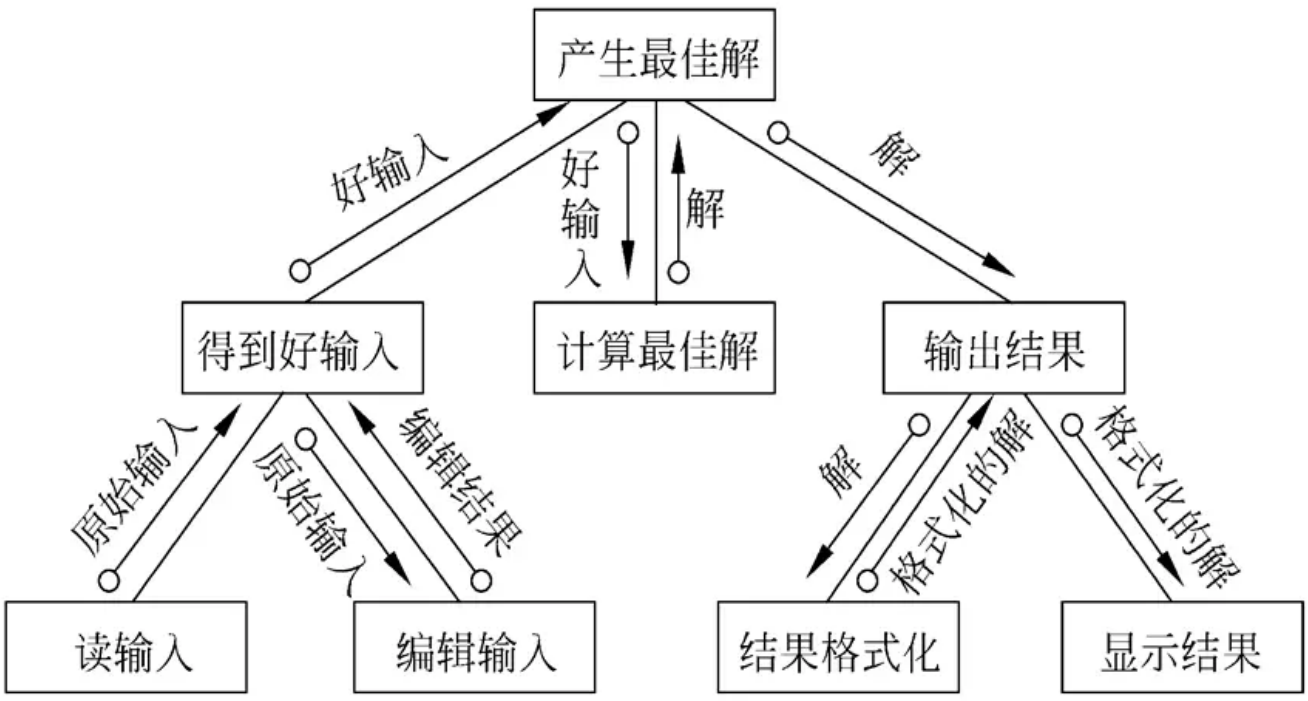


和H图中每个方框相对应，应该有一张IPO图描绘这个方框代表的模块的处理过程。模块在H图中的编号便于追踪了解这个模块在软件结构中的位置。

IPO 图是针对每一个方框的

结构图

方框代表一个模块； 方框之间的直线表示模块的调用关系； 尾部是空心圆箭头表示传递的是数据； 尾部实心圆箭头表示传递的是控制信息。



面向数据流的设计方法（重要）

怎么把数据流图转换为软件结构图

> 在面向对象的方法里面，有模型驱动的思想，结构化设计里，数据流图有规则可以软换成设计模型

两种流的概念

> 面向数据流的设计方法把信息流映射成软件结构，信息流的类型决定了映射的方法。

信息流的两种类型：

描述系统的逻辑结构

- 变换流

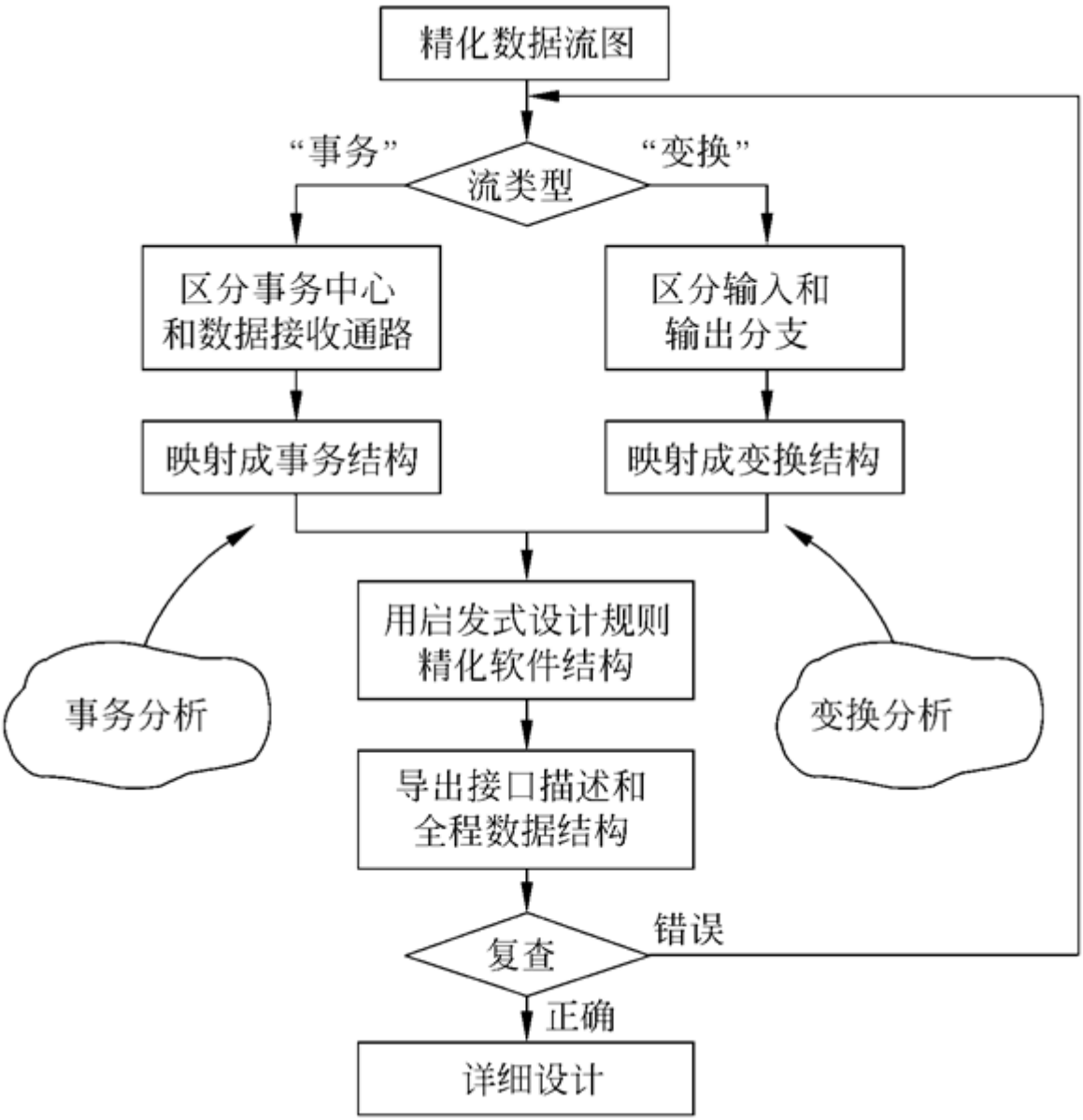
  把输入变换成输出，这样的流程

  输入-> 加工处理 -> 输出

- 事务流

  数据到达处理中心之后，处理中心根据数据的类型，选择一个动作序列来执行

  接收输入数据； 分析每个事务以 确定它的类型； 根据事务类型 选取一条活动通路

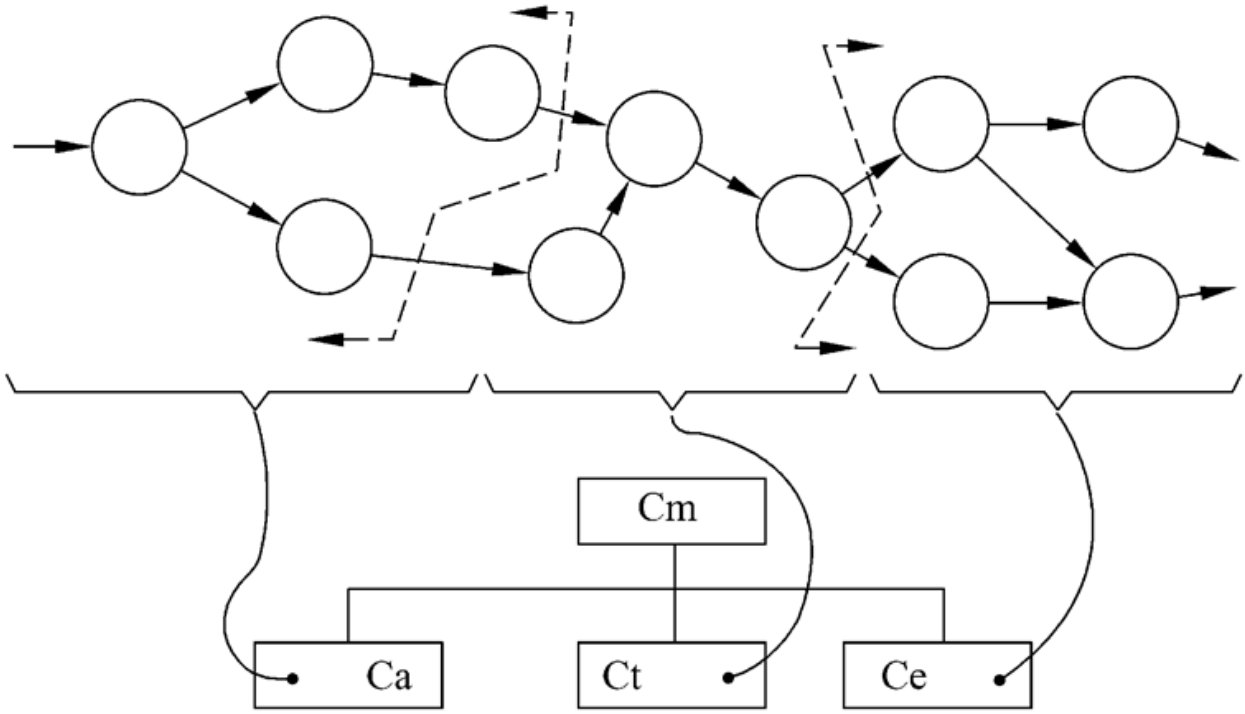


变换分析

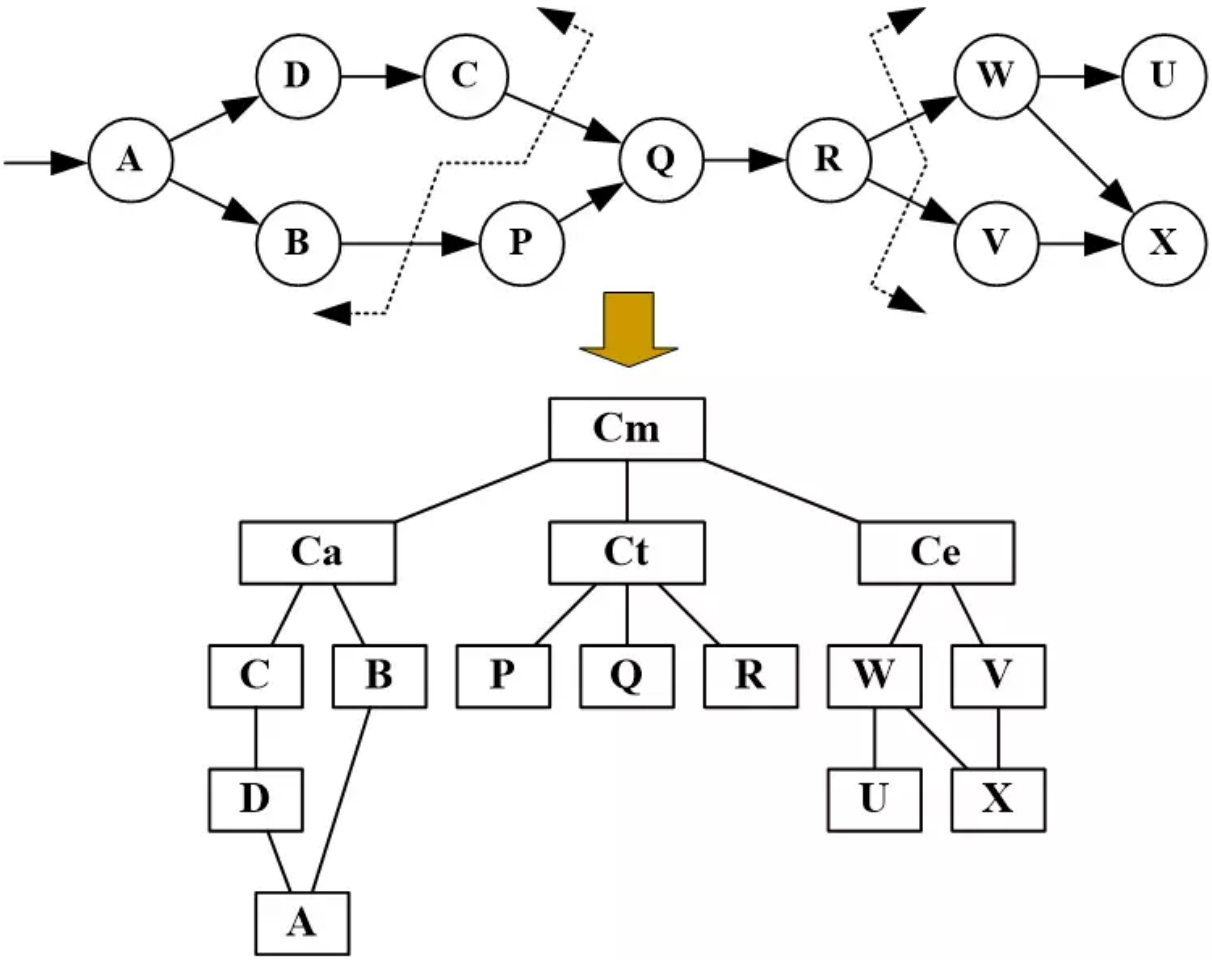
> 变换分析是一系列设计步骤的总称，经过这些步骤把具有变换流特点的数据流图按预先确定的模式映射成软件结构。

1. 第一级分解

   需要划两根虚线，将数据流的输入部分和输出部分划分出来



2. 完成第二级分解

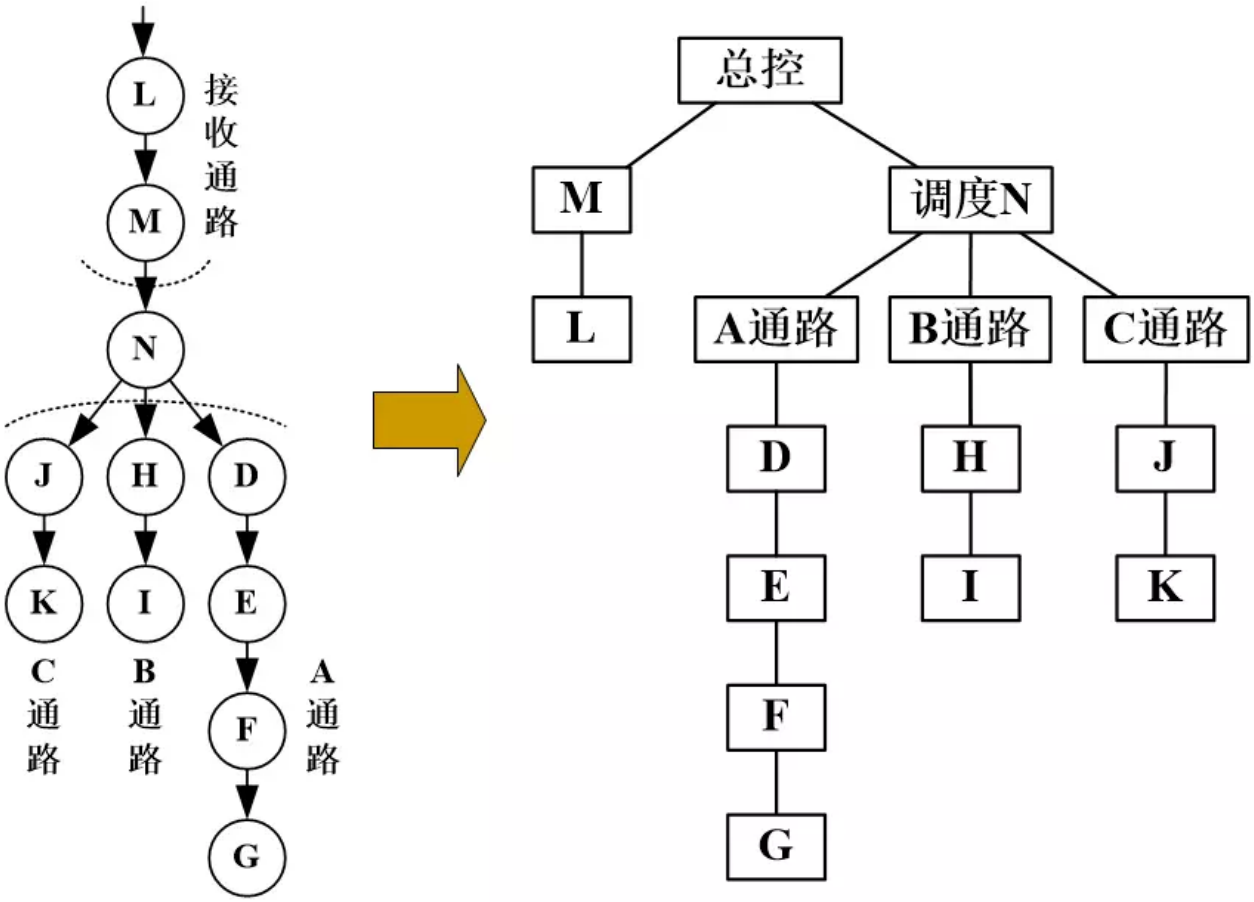


3. 使用启发式规则进行精化

第二步得到的，每个数据加工过程都定义为一个模块，因此在此基础会需要模块的合并

事务分析

1. 把事务流映射为接受分支和发射分支



> 变换流本质上是特殊的事务流，只有唯一的发射通路

设计步骤

1. 复查基本系统模型（数据流图）
2. 精化数据流图
3. 确定数据流图具有变换特性还是事务特性 （要找到事务中心）
4. 确定输入输出流的边界，孤立出变换中心
5. 完成 第一级分解，分解成 Ca Ce Ct
6. 完成第二集分解
7. 使用启发规则进行精化

详细设计

> **详细设计阶段的根本目标：**确定应该怎样具体地实现所要求的系统。经过这个阶段的设计工作，应该得出对目标系统的精确描述，从而在编码阶段可以把这个描述直接翻译成用某种程序设计语言书写的程序。**

结构程序设计

- 最基本的逻辑控制结构：

  顺序、选择、循环，【拓展】case，do-UNTIL，【修正】break

人机界面设计

> 人机界面设计是接口设计的重要组成部分

- 三条黄金规则
  - 置用户于控制之下
  - 减少 用户记忆负担
  - 保持界面一致

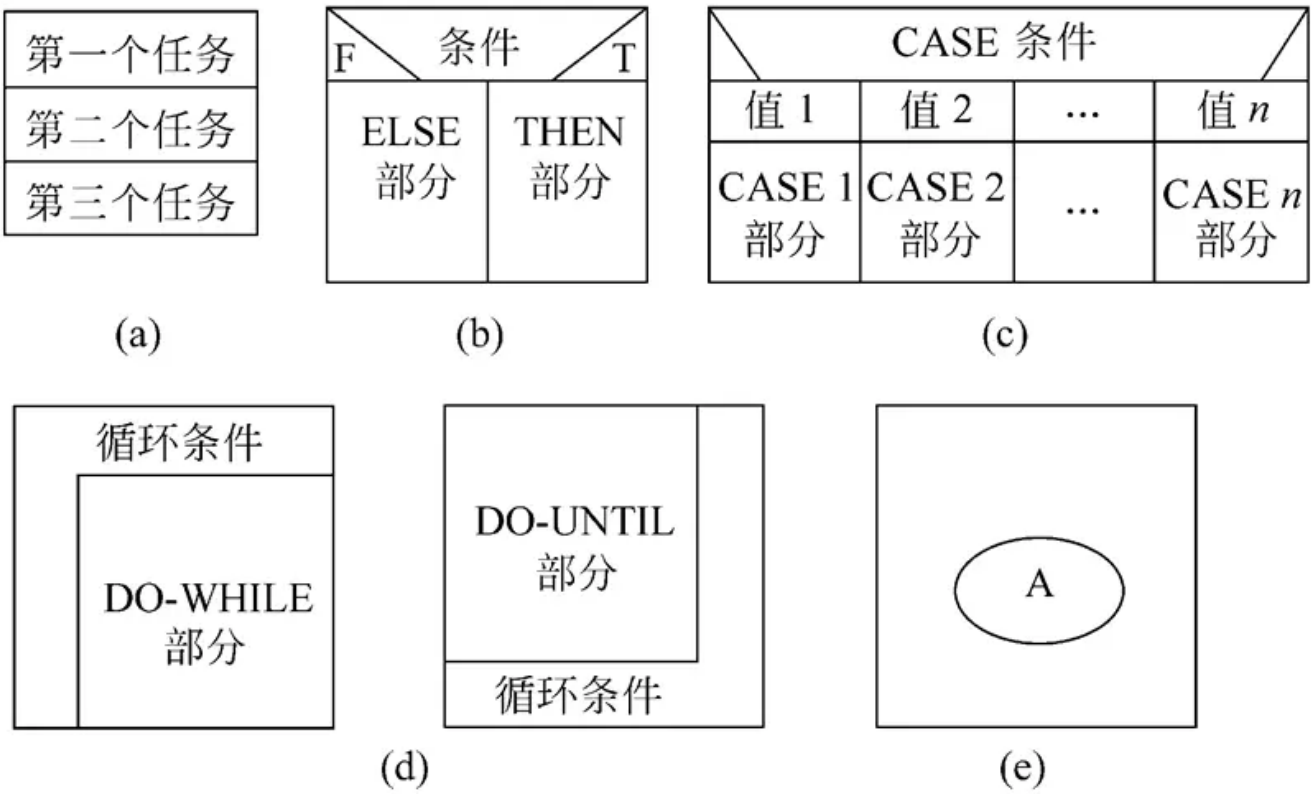
过程设计的工具

程序流程图（程序框图）（要求会画）

只要能看懂带矩形，菱形，箭头，圆形 符号的程序流程图即可

下面的图三年考两次

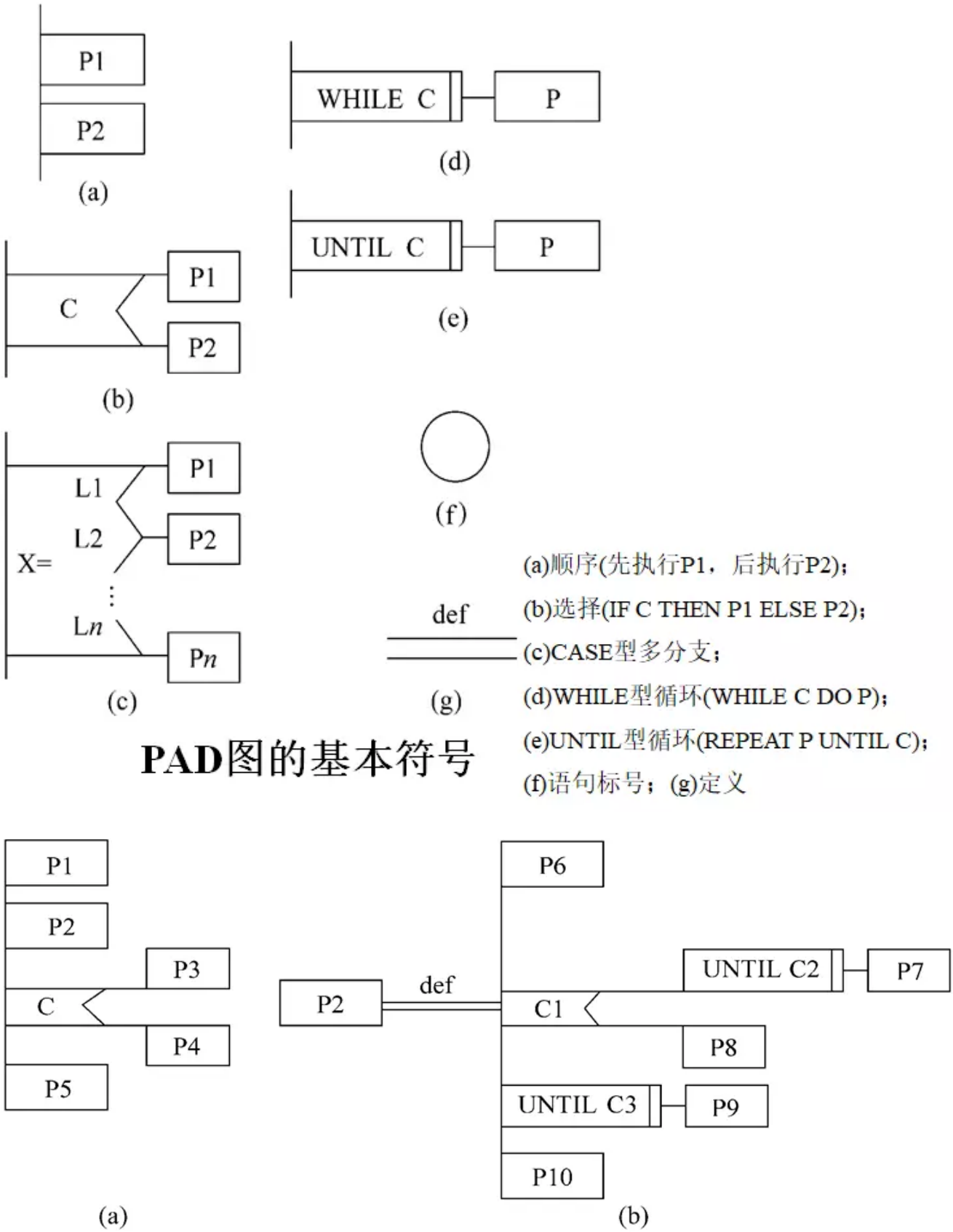
盒图



盒图的基本符号

(a)顺序；(b)IF-THEN-ELSE型分支；(c)CASE型多分支；(d)循环；(e)调用子程序A

PAD图



PAD图的基本符号

要会把程序流程图转化为盒图和 PAD 图（必须掌握）（**这个肯定考**）

先按照分支节点进行大分段，然后再进行逐段细分

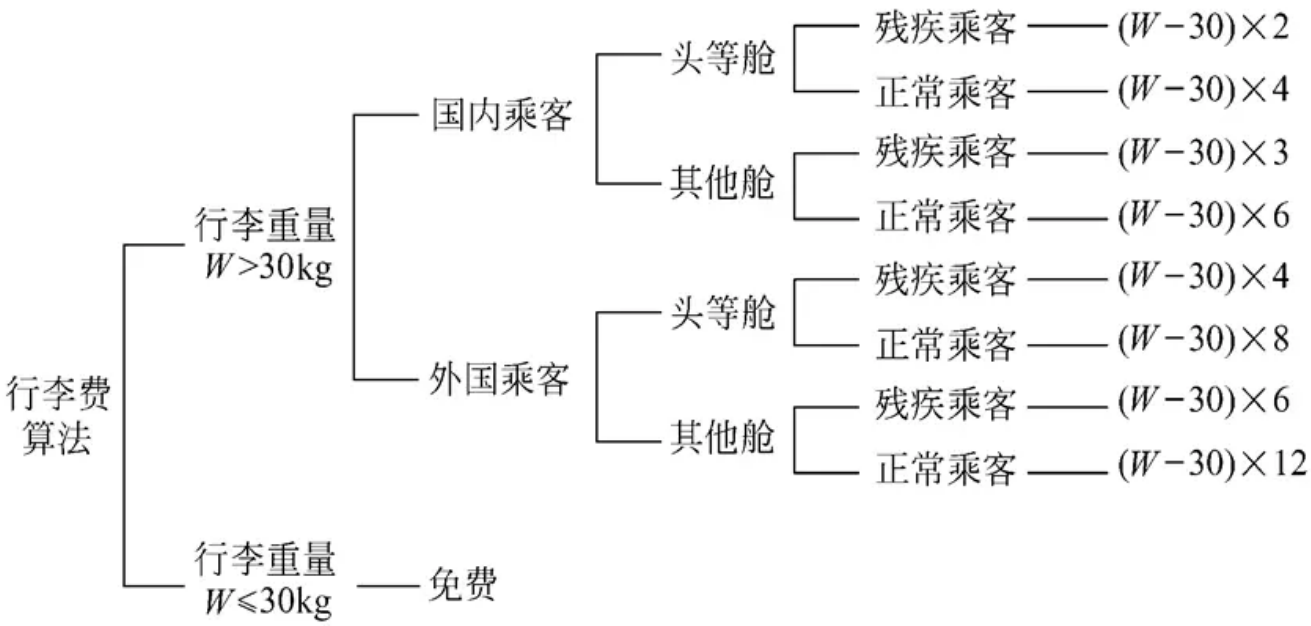
判定表

> 算法中如果包含多重嵌套的条件时，可以用判定表或判定树

一张判定表由4部分组成：

- 左上部列出所有条件；
- 左下部是所有可能做的动作
- 右上部是表示各种条件组合的一个矩阵；
- 右下部是和每种条件组合相对应的动作。

判定树



过程设计语言

- **伪码**
  - 要掌握的
    1. 会写伪码
    2. **会看伪码**

期末题目：通过伪码告诉程序的功能，在理解伪码的基础上，构建流程图，PAD 图，盒图

就两个题型：简答题，一共三十分，一共五道题，五道分析题，一共70分，一道题十分，剩下四个15分

面向数据结构的设计方法

最终目标： 得出对程序处理过程的描述

Jackson 图

使用 Jackson 描绘数据结构，

注意 Jackson 和层次图的区别

Jackson 方法

1. 用 Jackson 描述数据结构
2. 找出输入输出数据结构之间的联系
3. 导出描绘程序结构的 Jackson

   能理解构建过程就可以了，不需要自己构建

程序复杂程度的定量 度量

要考

这是设计的好与不好的评价指标

McCabe 方法

- **流图：**

  退化了的程序流程图：保留程序的控制流程

  流图的元素：

  - 节点
  - 边
  - 区域：边和
  - 节点围成的面积称为区域，包括图外部未被包围起来的开放区域
- 映射方法：
  1. 对于顺序结构和选择结构或循环的开始语句，映射为流程图的一个节点
  2. 选择结构：开始语句一个节点，两条分支各一个节点，结束映射为一个节点
  3. 循环结构：开始和结束语句各自映射为一个节点
- **环形复杂度**
  1. 直接等于区域数
  2. 边数 - 节点数 + 2
  3. 判定节点数 + 1
- 用途

  度量程序内分支数、循环个数，程序复杂度

Halstead 方法 不考

协作顺序（描述对象间的交互）

- 顺序图：着重展示消息的执行顺序
- **协作图：** 只强调了对象间的消息，消息的先后顺序需要靠标号指定

活动状态

- 状态图：描述对象的状态变化
- 活动图：描述执行过程中对象的行为变化

这四个图必须掌握

面向对象设计的准则

考！基本的六条准则

- 模块化：对象就是模块，把数据结构和操作这些数据的方法结合在一起
- 抽象：面向对象的方法不仅支持过程抽象，而且支持数据抽象
- 信息隐藏：
- 弱耦合：不同对象之间关联的紧密程度

  对象之间耦合的分类
  - 交互耦合
    - 对象之间的耦合通过消息连接来实现
    - 交互耦合应该尽可能松散
  - 继承耦合
    - 继承耦合越紧密越好
- 强内聚
  - 设计中使用的构件内的各个元素对完成目标的贡献程度

    三种内聚
- 可重用

软件重用

1. 知识重用
2. 方法和标准的重用
3. 软件成分的重用

- **软件成分重用级别**
  - 代码重用
  - 设计结构重用
  - 分析结果重用

第八章 实现

这一部分占考试的三分之一

这一章主要是测试，白盒测试和黑盒测试这两次考试考一个，大的分析题

- **编码和测试统称为实现**

编码不考

强调一点：编码风格 + 注释

软件测试

1. 什么是测试

   为了发现程序中的错误而执行过程的过程

2. 什么是好的测试

   是能发现尚未发现的错误的测试方案

软件测试准则

很重要

- 所有测试都能追溯到用户的需求
- 应该远在测试开始之前就指定出测试计划 （在需求分析阶段）
- 应用Pareto 原则 （二八原理）*已经找到错误的地方制定2更多的测试方案*
- 应该从小规模测试开始，逐步进行大规模测试
- 穷举测试不可能
- 应该由独立的第三方从事测试工作

	黑盒测试	白盒测试
优点	①适用于各阶段测试 ②从产品功能角度测试 ③容易入手生成测试数据	①可构成测试数据使特定程序部分得到测试 ②有一定的充分性度量手段 ③可获较多工具支持
缺点	①某些代码得不到测试 ②如果规格说明有误，则无法发现 ③不易进行充分性测试	①通常不易生成测试数据 ②无法对未实现规格说明的部分进行测试 ③工作量大，通常只用于单元测试，有应用局限
性质	一种确认技术，回答“我们在构造一个正确的系统吗？”	一种验证技术，回答“我们在正确地构造一个系统吗？”

软件测试的阶段

重要

测试阶段	主要依据	测试人员	测试方式	主要测试内容
单元测试	系统设计文档	开发小组	白盒测试	接口测试 路径测试
子系统测试	系统设计文档 需求文档	独立测试小组	白盒测试 黑盒测试	接口测试 路径测试 功能测试 性能测试
系统测试	需求文档	独立测试小组	黑盒测试	功能测试 健壮性测试 性能测试 用户界面测试 安全性测试 压力测试 可靠性测试 安装/反安装测试
确认测试 验收测试	需求文档	用户	黑盒测试	

- 单元测试：
  - 使用白盒测试
  - 测试每一条语句，依照系统设计文档
- 子系统测试（集成测试）
  - 交给独立的第三方测试人员
- 系统测试
  - 整个系统的功能完备
- 确认测试（验收测试）
  - 用户进行测试

Alpha 和 beta 测试

- alpha：用户在开发者的场所进行，Alpha 在受控的环境中进行
- Beta测试：开发者不再 Beta 测试的现场，是软件在开发者不能控制的环境中的真实应用

白盒测试和黑盒测试方法

两个里面考一个

白盒测试

五中逻辑覆盖的测试用例

流图得到环形复杂度、得到测试路径 路径数和环形复杂度相同 退化到数据流图

- 测试目的、应该输入的测试数据、预期的结果
- 白盒测试测试的一条条语句
- 我们期望在每一种测试的方案里面，把所有的语句覆盖到

  五类覆盖准则

  设计满足覆盖条件的测试样例
  - 语句覆盖

    选择足够多的测试数据，使得被测程序中的每个语句至少被执行一次
    - 只是覆盖了语句，没有覆盖所有的判定通路
    - 对程序的逻辑覆盖很少，
    - 最弱的覆盖标志
  - 判定覆盖

    每个判定的每种可能的结果都应该至少执行一次
    - 没有考虑到判定中的每一个条件
  - 条件覆盖
    - 满足所有的语句覆盖，每个语句至少执行一次，判定表达式中的每个条件都取到各种可能的结果
    - 探究每个判定取得所有结果的情况
    - 条件覆盖通常比判定覆盖强，但也有反例，一些测试用例并不能把每个判定覆盖到
  - 判定/条件覆盖

        - 使得判定表达式中的每个条件都取到各种可能的值，每个判定表达式也都取到各种可能的结果
    - 条件组合覆盖
        - 要求选取足够多的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。

控制结构测试

- 基本路径测试

  环形复杂度

  首先计算程序的环形复杂度； 以该复杂度为指南定义执行路径的基本集合； 从该基本集合导出的测试用例可保证程序中的每条语句至少执行一次，而且每个条件在执行时都将分别取真、假两种值。

  一定要看

  程序得到流图、流图得到环形复杂度、环形复杂度得到测试样例

- 循环测试
    - 简单循环

      使用以下测试集来测试简单循环

黑盒测试

- 两大原则 必须掌握
    - 测试用例越少越好
    - 一个测试用例能指出一类错误 （提高测试用例的质量）（归类的思想）

等价划分

- 有效等价类 *输入值在规定的范围内*
- 无效等价类
    - 输入值小于最小值
    - 输入值大于最大值

  不一定一定有两个无效等价类

给一段程序的功能描述，程序判断输入的这一串字符，判断他是否满足条件，现在要测试这个程序，等价划分把有限的输入域划分为几类，在每一类中选择样例，进行测试，使用有效、无效等价类进行归类设计，根据有效无效等价类设计测试用例

- **设计测试方案的步骤**
    1. 尽可能多的覆盖尚未被覆盖的有效等价类，制导所有有效等价类都被覆盖为止
    2. 设计测试方案，使他覆盖且只覆盖一个尚未被覆盖的无效等价类，制导无效等价类都被覆盖为止

  一定要会

边界值分析

有了等价类之后，选取测试数据时，要选刚刚达到阈值的数据

- 软件生命周期到底有哪些阶段
- 什么是软件： 程序+文档+数据结构

维护

软件工程的目的是要提高软件的可维护性，减少软件维护所需要的工作量

软件维护的定义

四项维护

- 改正性维护
- 适应性维护
- 完善性维护
- 预防性维护 （主动性维护，其余都是被动型维护）

软件项目管理

- 如何估算软件规模和工作量
    - 人月
    - 相同规模软件的代码行数

软件质量保证

软件质量保证 （SQA）

- 基于非执行的测试 （复审或评审）
- 基于执行的测试（软件测试）
- 程序正确性证明

什么是软件配置

在软件的整个生命周期内管理变化的一组活动

1. 软件配置项
    - 计算机程序
    - 描述计算机程序的文档
    - 数据
2. 基线
    - 已经通过了正式复审的规格说明或中间产品，作为进一步开发的基础，只有正式的变化控制过程才能改变他
    - **基线是通过了正式复审的软件配置项**

能力成熟度模型

- 初始级
  - 软件过程的特征是无序 的
- 可重复级
  - 软件机构建立了基本的项目管理过程，建立起了必要的过程规范
- 已定义级
  - 定义了一套完成的软件过程，软件过程已经文档化和标准化
- 已管理级
  - 对软件过程和软件产品定义了定量的质量目标
- 优化级
  - 集中精力持续不断的改进软件过程，以防止出现缺陷为目标