**1知道软件工程的概念**

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

**2软件工程的本质特征**

关注于大型程序的构造

中心课题是控制复杂性

软件经常变化

开发软件的效率非常重要

和谐地合作是开发软件的关键

软件必须有效地支持它的用户

软件工程领域中是由具有一种文化背景的人替具有另一种文化背景的人

**3软件工程的基本原理**

(1)用分阶段的生命周期计划严格管理

(2)坚持进行阶段评审

尽早发现在软件开发过程中所犯的错误

(3)实行严格的产品控制--基准配置管理

一切有关修改软件的建议，都必须按照严格的规程进行评审，获得批准以后才能实施修改

(4)采用现代程序设计技术

提高软件开发和维护的效率，提高软件产品的质量

(5)结果应能清楚地审查

根据总目标及完成期限，规定开发组织的责任和产品标准

(6)开发小组的人员应该少而精

当开发小组人员数为N时，可能的通信路径有N(N-1)/2条，可见随着人数N的增大，通信开销将急剧增加

(7)承认不断改进软件工程实践的必要性

不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验。

**4瀑布模型**

**过程**：需求分析与规约、设计与规约、编码与单元测试、集成测试系统测试、运行与维护

**特征：**

接受上一阶段的结果作为本阶段的输入

利用这一输入实施本阶段应完成的活动

对本阶段的工作进行评审

将本阶段的结果作为输出，传递给下一阶段

**缺点**

缺乏灵活性，难以适应需求不明确或需求经常变化的软件开发

线性开发，开发过程末期见到成果，风险大

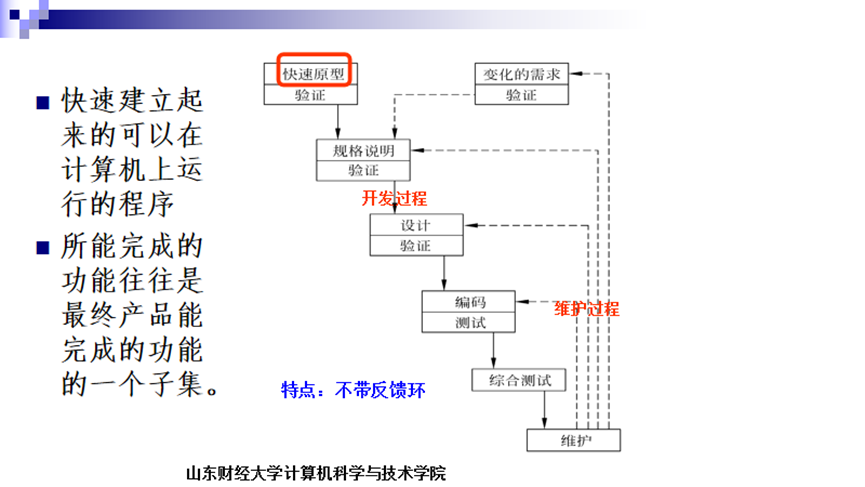
开发早期存在的问题往往要到交付使用时才发现，维护代价大（一个制造汽车的故事）

**5快速原型模型（重点掌握）带图**，

（1）基本思想：

通过开发系统原型和用户反复交互，以明确需求，使系统在不断调整与修改中得以进化成熟。

（2）流程图



**6 两种基本模型**

**Throwaway prototyping(抛弃式原型)**

最初的原型在完成并得到用户认可之后，将不会作为交付给用户的最终系统的一部分，而是被抛弃，其目的只是为了收集与验证需求；

– 该类原型可能是不可执行的(例如，只包含用户界面)；

**Evolutionary prototyping(演化式原型)**

最初构造的原型将具备较高的质量，包含了系统的核心功能，然后通过收集需求对其进行不断的改善和精化；

该类原型是可执行的，将成为最终系统的一部分；

**问题**

缺乏过程可见性;

系统结构通常会很差;

需要一些特别的技术（如原型快速开发技术），通常与主流技术不兼容.

**适用情况**

适合中小规模的交互系统;

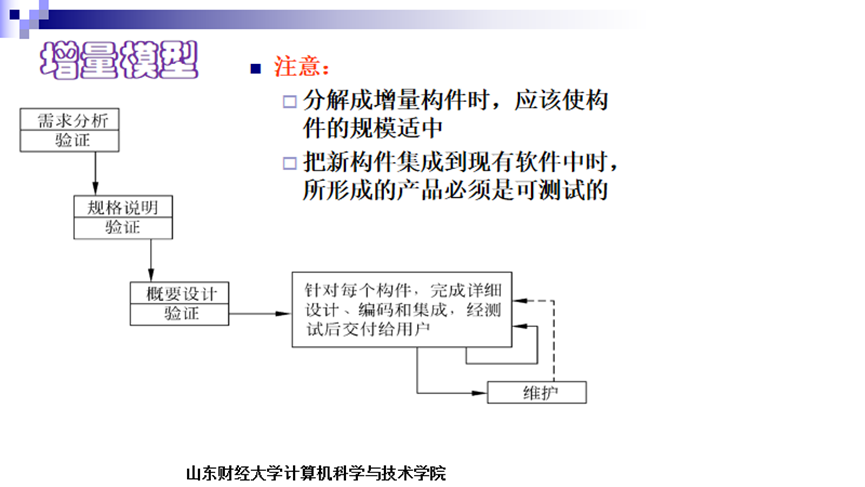
可用于大型系统的局部开发（如系统界面），可以和瀑布模型混合使用；

生命周期较短的系统。

**7增量模型（重点掌握+图+优点+问题）**

（1）增量模型也称为渐增模型，是Mills等人1980年提出来的。

使用增量模型开发软件时，把软件产品作为一系列的增量构件来设计、编码、集成和测试，每个构件由多个相互作用的模块构成，并且能够完成特定的功能。



（3）优点

能在较短时间内向用户提交可完成一些有用的工作产品

用户有较充裕的时间学习和适应新产品

项目失败的风险较低

最重要的系统服务将接受最多的测试。

优先级最高的服务首先交付，然后再将其他增量构件逐次集成进来

（4）问题

在把每个新的增量构件集成到现有软件体系结构中时，必须不破坏原来已经开发出的产品。

软件体系结构必须是开放的，即向现有产品中加入新构件的过程必须简单、方便。

因此，采用增量模型比采用瀑布模型和快速原型模型更需要精心的设计。

**8喷泉模型［变量对象了解就行］**

喷泉模型是典型的面向对象生命周期模型。

“喷泉”一词体现了迭代和无间隙特性。图中代表不同阶段的圆圈相互重叠，这明确表示两个活动之间存在重叠。

**9快速应用模型［多个团队……，开发快］**

多个团队并行进行开发，是瀑布模型的高速变体，通过基于构件的构建方法实现快速开发

**10形式化方法模型［第一页+第三页］形式化……求阶，净堂……是一个团队流水线开发**，

（1）概念

式化系统开发模型是一种基于形式化数学变换的软件开发方法

凡是采用严格的数学语言，具有精确的数学语义的方法，都称为形式化方法。

它可将系统规格说明转换为可执行的程序

（2）形式化方法的主要特点

用数学记号表达的详细的形式化规格说明。设计、实现和单元测试等开发过程由变换开发过程代替。通过一系列变换将形式化规格说明细化成为程序(形式化程序求精)。

（3）净室过程模型:1987年IBM开发，是一个团队流水线开发，

**11动态视角（四个过程）（1）四个过程**

初始Inception

细化Elaboration

构造Construction

移交Transition

**12实践观点一整页。**

（1）迭代地开发软件(每个构件的开发迭代，每个阶段内的多次迭代)

（2）管理需求

（3）使用基于构件的体系结构

（4）建立可视化模型（UML）

（5）不断地验证软件质量(贯穿整个开发)

（6）控制对软件的变更（使每个修改需求可接受和可跟踪）

**13极限编程：适用于……场合，有效实践（选择题）**

（1）概念

敏捷过程中最重要的部分，广泛应用于需求模糊且经常改变的场合。

（2）有效实践【选择题】

客户是开发团队的成员

短交付周期（每两周完成一次迭代，交付用户）

验收测试

结对编程

测试驱动开发

集体所有（小组每个成员都有更改代码的权利并负责）

持续集成（一天之内多次集成系统）

可持续的开发速度（每周<=40小时，加班<=2周）

及时调整计划

简单的设计（只考虑当前）

重构（调整和优化系统的内部结构）

使用隐喻（全局视图）

**1可行性研究（三个可行性的含义）**

技术可行性、经济可行性、操作可行性

**2系统流程图**

描绘物理系统的工具，其基本思想是用图形符号以黑盒子形式描绘系统里面的每个部件

反映信息在各部分间流动的情况 ≠控制、处理信息（不同于程序流程图）

**数据流图用途**

基本目的是利用它作为交流信息的工具

作为分析和设计的工具

可以逻辑模型考虑系统的物理实现

**3数据字典**

是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义的集合

数据字典（Data Dictionary,DD）是“关于系统数据的数据库”。

建立数据字典是为了对数据流程图上各个元素作出详细的定义和说明。

**1需求分析的三个任务**

(1) 必须定义软件应完成的功能，要求建立功能模型

(2) 必须理解并描述问题的信息域，应该建立数据模型

(3) 必须描述作为外部事件结果的软件行为，要求建立行为模型

**2需求获取**

**访谈**：分发调查表、情景分析

**简易的应用规格说明技术**：面向团队，提倡用户与开发者密切合作，共同标识问题，提出方案要素，商讨不同方案并指定需求

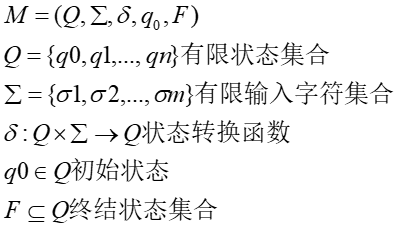
**面向数据流自顶向下求精**

**快速原型**：快速建立起来的旨在演示目标系统主要功能的可运行的程序

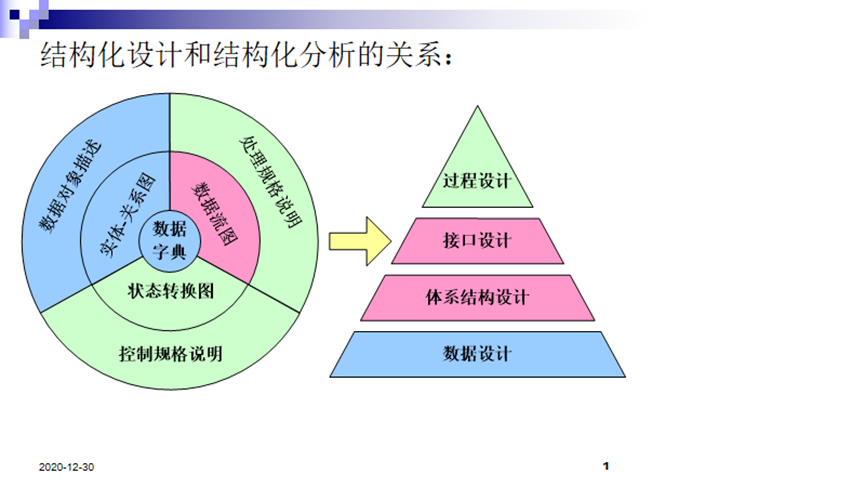
要点：原型应该实现用户看得见的功能，例屏幕显示或打印报表，省略 “隐含”功能，例修改文件

特点：快速和容易修改

2有限状态机的表示



**1结构化设计和结构化分析的关系**



**2详细设计设计：对每个屋子（每个模块）**

总体设计结果是模块设计图

详细设计是对每个模块的设计

**3设计原理**

（1）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。

（2）抽象

过程抽象

在软件设计中将处理过程的实现细节隐藏在数据抽象中，可以直接通过模块接口使用这些处理操作

数据抽象

采用抽象数据类型表示数据，实现数据封装，使得使用者可通过接口使用数据而不必关心数据结构的实现。

（3）自顶向下、逐步求精

将软件体系结构自顶向下，对过程细节和数据细节从抽象到具体，逐层细化，直到用编程语言的语句能够实现为止

（4）信息隐藏和局部化

信息隐藏：应该这样设计和确定模块，使得一个模块内包含的信息(过程和数据)对于不需要这些信息的模块来说，是不能访问的。

局部化：局部化的概念和信息隐藏概念是密切相关的。所谓局部化是指把一些关系密切的软件元素物理地放得彼此靠近。显然，局部化有助于实现信息隐藏。

（5）模块独立性

每个模块完成一个相对独立的子功能，并且与其它模块间的接口简单。

**4耦合，常见的耦合，强弱，排列，这是什么耦合**

（）软件结构中各个模块之间相互关联程度的度量。

（）常见的耦合（独立性由弱到强）

由弱到强：内容耦合、公共环境耦合、特征耦合、控制耦合、数据耦合

内容耦合

公共环境耦合：模块之间通过一个公共数据环境相互作用

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

控制耦合：传递的信息中有控制信息(尽管有时这种控制信息以数据的形式出现)

数据耦合：如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据

**5内聚，分辨各种内聚，过程内聚和顺序内聚要会区分**

**内聚**：模块内部各个元素彼此结合的紧密程度的度量，模块内聚性越强，功能独立性越好，对于形成的模块结构有比较好的作用。要求模块结构达到高内聚，低耦合。

**低内聚、巧合内聚**：当几个模块内正好有一段代码是相同的，将它们抽取出来形成单独的模块，即巧合内聚模块。这种模块没有独立功能，各部分之间没有联系，或联系很松散。

评价：

模块内各元素之间没有实质性联系，很可能在一种应用场合需要修改这个模块，在另一种应用场合又不允许这种修改，从而陷入困境；

可理解性差，可维护性产生退化；

模块是不可重用的。

解决方案：

将模块分成更小的模块，每个小模块执行一个操作。

**逻辑内聚**：这种模块(例如，读一个记录，写一个记录)把几种相关的功能组合在一起

每次被调用时，由传送给模块的判定参数来确定该模块应执行哪一种功能。

**中内聚：过程内聚**

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

使用流程图做为工具设计程序时，如果流程图太大，可以把流程图中的某一部分划出组成模块，就得到过程内聚模块。

例如，把流程图中的循环部分、判定部分、计算部分分成三个模块，这三个模块都是过程内聚模块。

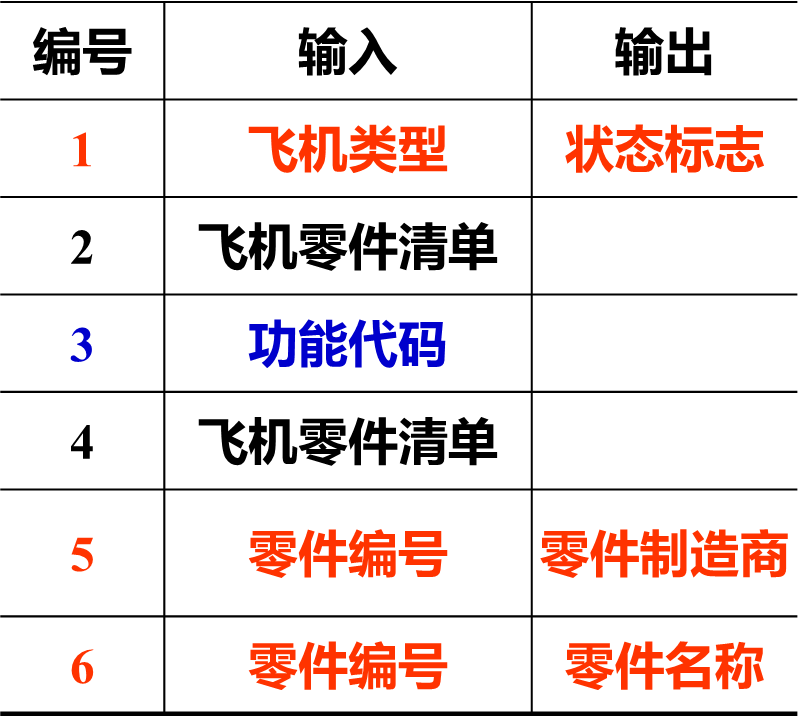
**通信内聚：**如果一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据

通常，通信内聚模块是通过数据流图来定义的。

高内聚：

**顺序内聚**：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(通常一个处理元素的输出数据作为下一个处理元素的输入数据)

**功能内聚**：模块内所有处理元素属于一个整体，完成一个单一的功能.



接口1,5,6都是数据耦合，传递的都是简单变量

接口3传递一个功能代码，属于控制耦合

当具有逻辑内聚的模块传递功能代码时就传递了控制元素。此时属于控制耦合

图中p、t和u具有公共环境耦合

更新同一个数据库

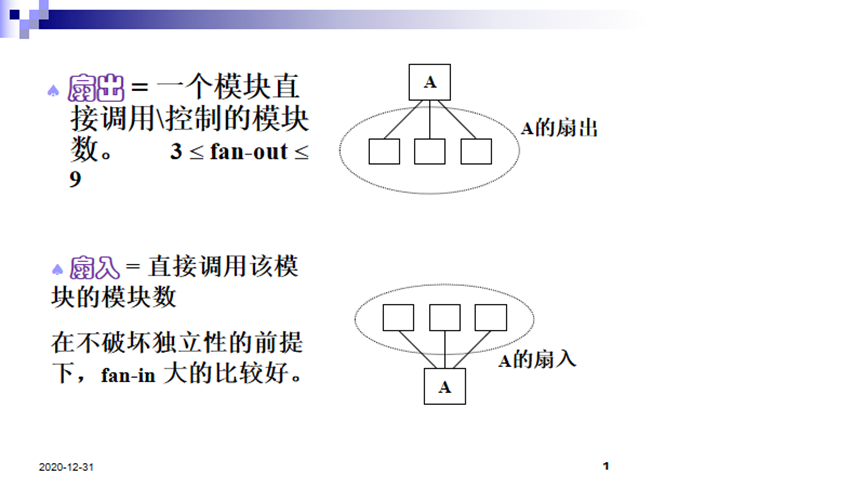
接口2中，如果s使用或更新p传递给它的零件清单中的所有元素，则属于数据耦合；

如果s只访问该清单的一部分元素，则属于特征耦合

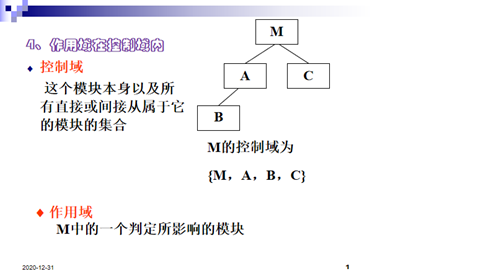
接口4与接口2类似

**6启发性规则：**争取低耦合、高内聚（增加内聚 > 减少耦合）

**7扇入 扇出**



8 作用域在控制域内



**1详细设计的任务：设计每个模块的算法**

**结构化程序设计中的三种结构**

结构化程序设计思想的提出

概念最早由Dijkstra提出，他是鉴于高级语言Fortran中的”GOTO语句有害”

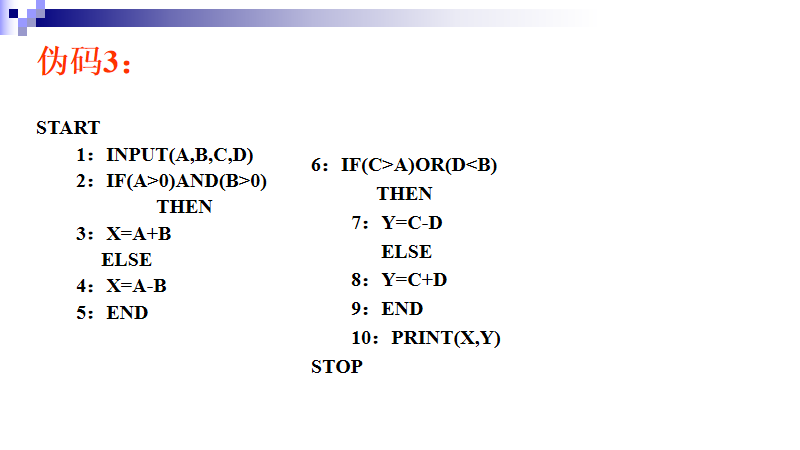
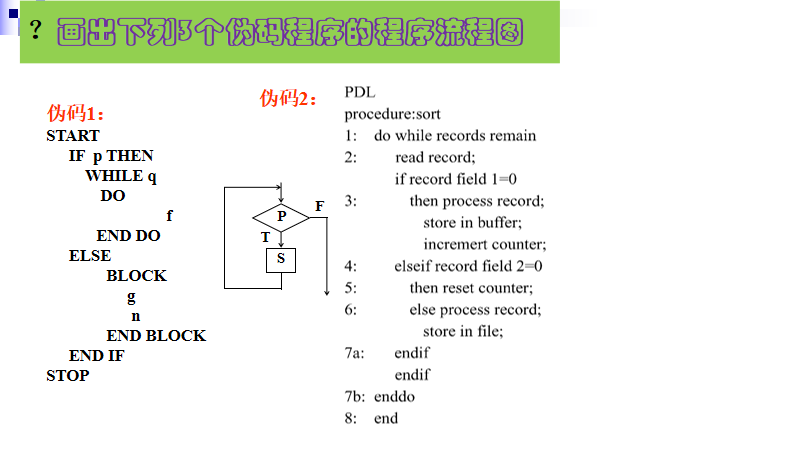
顺序（Sequence）;条件（Condition）;重复（Repetition）

**设计表示法**

图形设计表达式

表格设计表达式

过程设计语言表达式



**面向数据结构的设计：因为的下一句**

Jackson认为，流程图不是最好的设计工具:

因为设计首先考虑的应该是问题的结构，而流程图却让设计者首先考虑的是程序在计算机内的执行顺序(算法)。这样，就掩盖了有问题的结构。

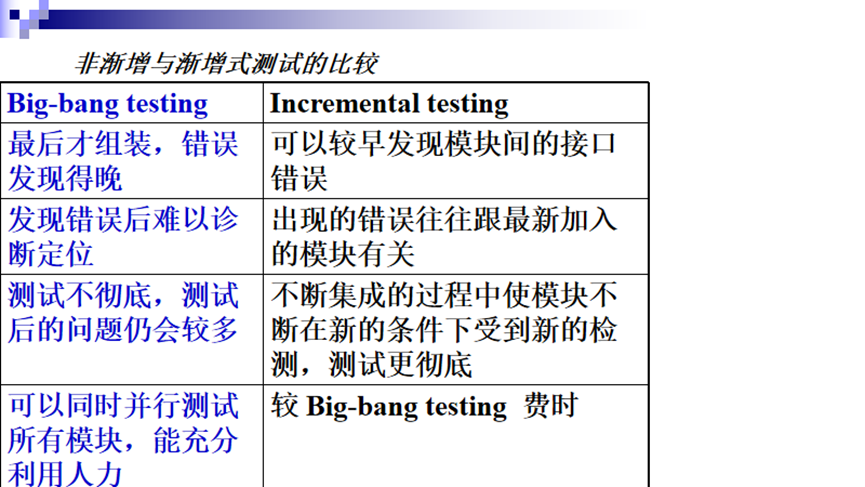
**1软件设计的基本概念**

为了发现程序中的错误而执行程序的过程，软件测试贯穿于软件定义和开发的整个期间

**2单元测试用白盒测试法**

(1)白盒测试：上边驱动，下边存根（也叫桩）

(2)集成测试：渐增测试：两个方法，都要看



**4黑盒测试：**等价划分，ppt48，仔细看、ppt51仔细看看、边界值分析，例如需要看看，0-99的边界、四种维护：重点

（1）黑盒测试技术----等价划分

问题：如何选择适当的子集，使其发现最多的错误

解决上述问题的办法：把所有可能的输入数据划分为若干等价类，这样就有可能得到一种假设，即测试某个等价类的代表值就等价于对这一类其他值的测试

等价划分技术是用来设计发现错误种类的测试用例，以减少必须设计的测试用例总数

（2）等价类划分的启发式规则：

如果规定了输入值的范围，则可划分出一个有效的等价类(输入值在此范围内)，两个无效的等价类(输入值小于最小值或大于最大值)

如果规定了输入数据的个数，则类似地也可划分出一个有效的等价类和两个无效的等价类

如果规定了输入数据的一组值，而且程序对不同输入值做不同处理，则每个允许的输入值是一个有效的等价类，此外还有一个无效的等价类(任一个不允许的输入值)

如果规定了输入数据必须遵循的规则，则可以划分出一个有效的等价类(符合规则)和若干个无效的等价类(从各种不同角度违反规则)；

如果规定了输入数据为整型，则可以划分出正整数、零和负整数等3个有效类；

如果程序的处理对象是表格，则应该使用空表，以及含一项或多项的表。

（3）设计测试方案时两个步骤

设计一个新的测试方案以尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步骤直到所有有效等价类都被覆盖为止；

设计一个新的测试方案，使它覆盖一个而且只覆盖一个尚未被覆盖的无效等价类，重复这一步骤直到所有无效等价类都被覆盖为止。

（4）边界值分析

注意：① 程序最容易在边界发生错误；

② 通常与等价划分结合进行。

错误推测(Failure Prediction)

思路：① 列出可能有的错误；

② 列出容易发生错误的特殊情况。 以此为基础设计测试方案。

（5）软件维护

软件维护——在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件的过程。

A:改正性维护(corrective maintenance )——诊断和改正错误,约占全部维护活动的 17~20%；

B：适应性维护(adaptive maintenance) ——为了和变化了的环境（如软\硬件升级、新数据库等）适当地配合而修改软件 ，约占全部维护活动的18~25%；

C：完善性维护（perfective maintenance) ——为了增加新功能，修改已有功能，改造界面，增加HELP等，而修改软件,约占全部维护活动的50~66% ；

D：预防性维护(preventive maintenance) ——为了改进未来的可维护性或可靠性，或为了给未来的改进奠定更好的基础而修改软件，与其它维护活动共占总维护的4%左右。

注：① 一般维护的工作量占生存周期70%以上，维护成本约为开发成本的4倍(80 - 20 Rule)；

② 文档维护与代码维护同样重要。

**1多态 两页**

（1）多态概念

根据为请求提供服务的对象不同可以得到不同的行为

（2）动态绑定、后期绑定或运行时绑定

在运行时对类进行实例化，并调用与实例化对象相应的方法

静态绑定、前期绑定或编译时绑定

如果方法的调用是在编译时确定的

（3）多态实现

方法重载

是Java 实现多态性的一种方式，指在同一个类中的两个或两个以上的方法可以有同一个名字，只要它们的参数声明不同即可

方法覆盖

覆盖体现了子类补充或改变父类方法的能力，通过覆盖使一个方法在不同子类间表现出不同的行为

**2面向对象建模 三个模型**

（1）对象模型（类图、对象图、包图）：

表示静态的、结构化的系统的“数据”性质，它是对模拟客观世界实体对象以及对象间的关系的映射，描述了系统的静态结构。

三个模型中最重要、最基本、最核心的。

（2）动态模型（状态图，第3.6节）

表示瞬时的、行为化的系统的“控制”性质，它规定了对象模型中的对象的合法变化序列。

（3）功能模型（用例图）

表示变化的系统的“功能”性质，它指明了系统应该“做什么”。

**3接口的概念：**接口（interface）── 描述了一个类或构件的一组外部可用的服务（操作）集。接口定义的是一组操作的描述，而不是操作的实现**。**接口体现了使用与实现分离的原则

**4什么是用例，要画用例图**（16分）

用例（use case）── 也称用况，用于表示系统想要实现的行为，即描述一组动作序列（即场景）。而系统执行这组动作后将产生一个对特定参与者有价值的结果。

**5状态图：动态图，一个对象的……变化**

状态图（state machine）── 描述了一个对象或一个交互在生存周期内响应事件所经历的状态序列，单个类或者一组类之间协作的行为都可以用状态机来描述。

**6 关联关系**

关联(association)是一种结构关系，它描述了两个或多个类的实例之间的连接关系，是一种特殊的依赖。

关联分为普通关联、限定关联、关联类，以及聚合与复合。

**普通关联**

普通关联是最常见的关联关系，只要类与类之间存在连接关系就可以用普通关联表示。普通关联又分为二元关联和多元关联。

二元关联描述两个类之间的关联，用两个类之间的一条直线来表示，直线上可写上关联名。

多重性（multiplicity）：多重性表明在一个关联的两端连接的类实例个数的对应关系，即一端的类的多少个实例对象可以与另一端的类的一个实例相关。

**限定关联**

限定关联通常用在一对多或多对多的关联关系中，可以把模型中的多重性从一对多变成一对一，或将多对多简化成多对一。

在类图中把限定词（qualifier）放在关联关系末端的一个小方框内。

**关联类**

关联关系的名字

来概括关联关系的语义

关联类

对关联关系的语义做详细的定义、存储和访问

用来描述关联的属性。

**聚合**

聚合（Aggregation）也称为聚集，是一种特殊的关联。它描述了整体和部分之间的结构关系。

共享聚合：如果在聚合关系中处于部分方的实例可同时参与多个处于整体方实例的构成，则该聚合称为共享聚合。

复合聚合：如果部分类完全隶属于整体类，部分类需要与整体类共存，一旦整体类不存在了，则部分类也会随之消失，或失去存在价值，则这种聚合称为复合聚合。

**导航**（navigability）是关联关系的一种特性，它通过在关联的一个端点上加箭头来表示导航的方向。

**7泛化关系**

泛化(generalization)关系就是一般类和特殊类之间的继承关系。

在UML中，一般类亦称泛化类，特殊类亦称特化类

泛化针对类型而不针对实例，因为一个类可以继承另一个类，但一个对象不能继承另一个对象。

泛化可进一步划分成**普通泛化和受限泛化两类。**

**普通泛化**

在泛化关系中常遇到抽象类。

一般称没有具体对象的类为抽象类。抽象类通常作为父类，用于描述其他类（子类）的公共属性和行为。普通泛化可以分为多重继承和单继承。多重继承是指一个子类可同时继承多个上层父类。

**受限泛化**

受限泛化关系是指泛化具有约束条件。

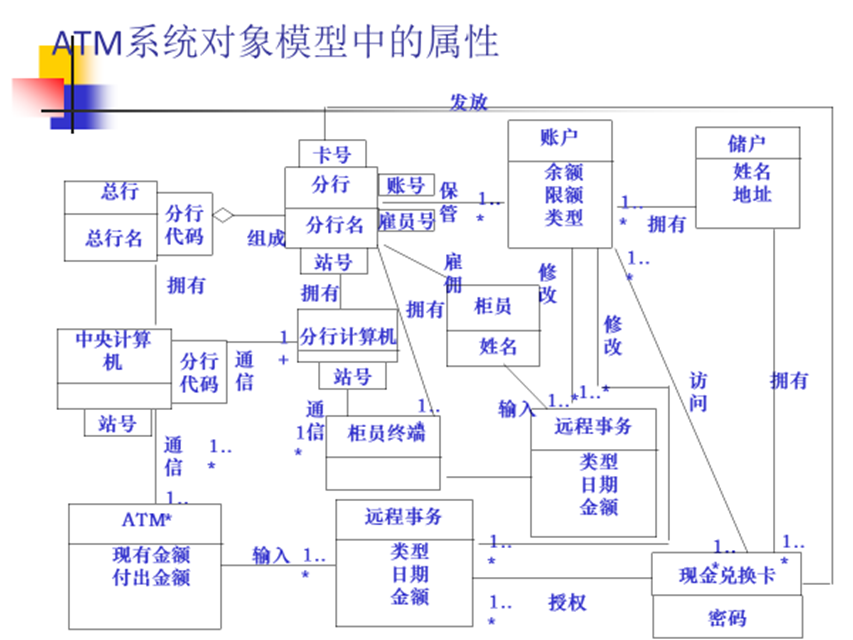
一般有4种约束：交叠（overlapping）、不相交（disjoint）、完全（complete）和不完全（incomplate）。

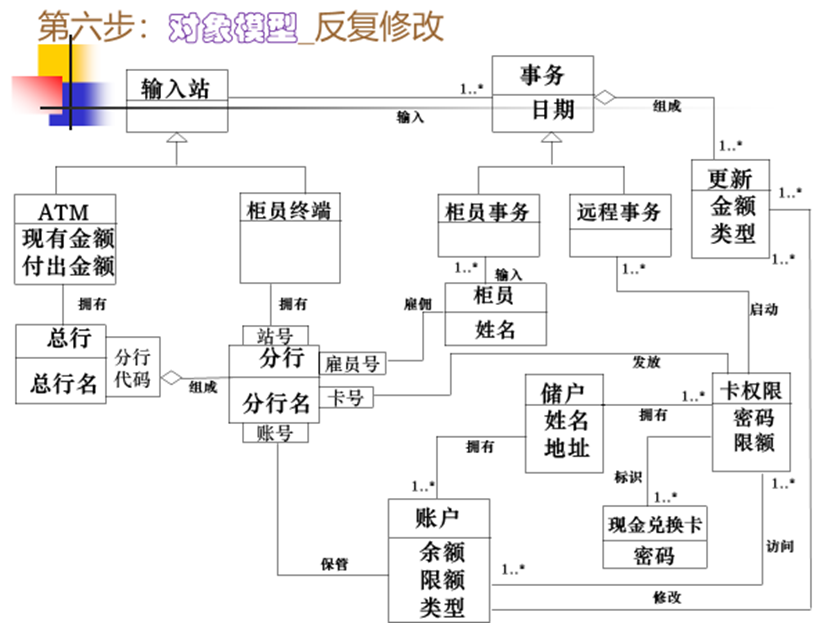
**8实现关系**

实现(implement)是泛化关系和依赖关系的结合，也是类之间的语义关系，通常在以下两种情况出现实现关系：

(1)接口和实现它们的类或构件之间；

(2)用例和实现它们的协作之间。





**面向对象设计模型**

从面向对象分析到面向对象设计(OOD)，是一个逐渐扩充模型的过程

分析是提取和整理用户需求，并建立问题域精确模型的过程

设计则是把分析阶段得到的需求转变成符合成本和质量要求的、抽象的系统实现方案的过程。

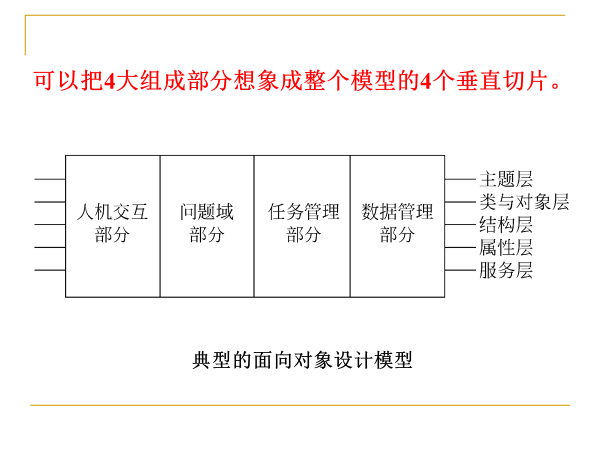
在实际的软件开发过程中分析和设计的界限是模糊的

分析和设计活动是一个多次反复迭代的过程

与面向对象分析模型一样，也由主题、类与对象、结构、属性、服务等5个层次组成。

大多数系统的面向对象设计模型，在逻辑上都由4大部分组成。

问题域子系统、人机交互子系统、任务管理子系统、数据管理子系统

****

**类构件**

面向对象技术中的“类”，是比较理想的可重用软构件，称之为类构件。

**1. 可重用软构件应具备的特点**

模块独立性强

具有高度可塑性

接口清晰、简明、可靠

精心设计的“类”基本上能满足上述要求，可以认为它是可重用软构件的雏形

**2. 类构件的重用方式**

实例重用

由于类的封装性，使用者无须了解实现细节就可以使用适当的构造函数，按照需要创建类的实例。这是最基本的重用方式。

继承重用

继承重用提供了一种安全地修改已有类构件，以便在当前系统中重用的手段。

多态重用

利用多态性不仅可以使对象的对外接口更加一般化，从而降低了消息连接的复杂程度。

**设计问题域子系统**

面向对象设计

仅需从实现角度对问题域模型做一些补充或修改，主要是增添、合并或分解类与对象、属性及服务，调整继承关系等等。

当问题域子系统过分复杂庞大时，应该把它进一步分解成若干个更小的子系统。

**设计人机交互子系统**

在面向对象设计过程中，则

应该对系统的人机交互子系统进行详细设计，以确定人机交互的细节。

使用由原型支持的系统化的设计策略，是成功地设计人机交互子系统的关键。

**设计任务管理子系统**

1. 分析并发性

通过面向对象分析建立起来的动态模型，是分析并发性的主要依据。

如果两个对象彼此间不存在交互，或者它们同时接受事件，则这两个对象在本质上是并发的。

并发事件处理：多处理器或单处理器采用时间分片策略仿真多处理器环境。

2. 设计任务管理子系统

确定事件驱动型任务

这类任务可能主要完成通信工作。事件到达之前处于睡眠状态（不消耗处理器时间）

确定时钟驱动型任务

每隔一定时间间隔就被触发以执行某些处理。触发之前处于睡眠状态。

确定优先任务

高优先级，低优先级

确定关键任务

确定协调任务：3个以上任务，应增加一个协调任务

尽量减少任务数

确定资源需求：软硬件需求

**设计数据管理子系统**

1. 选择数据管理存储模式

选择数据存储管理模式

文件管理系统

关系数据库管理系统

面向对象数据库管理系统

1. 设计数据管理子系统

1. 设计数据格式：与数据存储管理模式有关

关系数据库管理系统（基于第三范式）：列出每一类的属性表。

面向对象数据库管理系统

扩展的关系数据库途径：使用与关系数据库管理系统相同的方法。

扩展的面向对象程序设计语言途径：不需要规范化属性的步骤，因为数据库管理系统本身具有把对象值映射成存储值的功能。

2. 设计相应的服务设计相应的服务：

关系数据库管理系统

定义一个objectServer类，它提供下列服务：通知对象保存自身，检索已存储的对象（查找、读值、创建并初始化对象）。

面向对象数据库管理系统

扩展的关系数据库途径：与关系数据库管理系统方法相同。如果某个类的对象需要存储起来，则在这个类中增加一个属性和服务，用于完成存储对象自身的工作

扩展的面向对象程序设计语言途径：无须增加服务，已经给每个对象提供了“存储自己”的行为。

**四个模式**

State模式：目的：允许一个对象在其内部状态改变时改变其行为

Decorator模式：目的：动态地给一个对象添加一些额外的职责（就增加功能来说，Decorator 模式相比生成子类更为灵活）

Adapter模式：目的：将一个类的接口转换成客户希望的另外一个接口；Adapter 模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作

Observer模式：观察者模式（别名：依赖，发布-订阅）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变化时，所有依赖于它的对象都得到通知并被自动更新。

第十三章

软件项目管理：所谓管理就是通过计划、组织和控制等一系列活动，合理地配置和使用各种资源，以达到既定目标的过程。软件项目管理先于任何技术活动之前开始，并且贯穿于软件的整个生命周期之中。软件项目管理过程从一组项目计划活动开始，而制定计划的基础是工作量估算和完成期限估算。