# 软件工程导论

## 软件工程学概述

### 软件危机

#### 软件危机介绍

在计算机软件开发和维护过程中所遇到的一系列严重问题。主要包含两个方面如何开发软件，以满足日益增长的软件需求如何维护数量不断膨胀的已有软件

#### 产生软件危机原因

##### 主观原因：

忽视需求分析轻视软件维护没有认识到程序只是软件的一部分（很多人的共性问题）没有认识到软件开发只是软件漫长生命周期中一个比较次要的阶段越到后期如果引入变动则代价越高

##### 客观原因：

软件是逻辑实体，具有不可见性，所以管理和控制较为困难软件不会磨损，维护意味着需要修改原来的设计，维护困难软件规模庞大，程序复杂性随规模增加而增加

#### 消除危机途径

①对计算机软件应该有正确的认识应该彻底消除在计算机系统早期发展阶段形成的“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。其中，程序是能够完成预定功能和性能的可执行的指令序列;数据是使程序能够适当地处理信息的数据结构;文档是开发、使用和维护程序所需要的图文资料

②要吸取和借鉴人类长期从事各种工程项目积累的原理、概念、技术和方法

③积极开发和使用计算机辅助开发软件

④探索更好更有效的管理措施和手段对开发过程进行控制和管理

### 软件工程

#### 介绍

指导计算机软件开发开发和维护的一门学科。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，经济的开发出高质量的软件并维护它。

##### 软件工程的本质特征：

①关注大型程序的构造

②中心课题是控制复杂性

③软件经常变化

④开发效率非常重要

⑤开发人员和谐合作是关键

⑥软件需要有效支持用户

⑦软件开发者替代其他领域人员创造产品

#### 方法学

##### 概念：

把在软件生命周期全过程中使用的一整套技术方法的集合称之为方法学，也称为泛型。软件工程方法学包含三个要素：方法、工具、过程

##### 分类：

传统方法学（生命周期方法学）

面向对象方法学

### 软件生命周期

#### 软件定义

##### 问题定义：

弄清用户要解决什么问题通过对客户的访问调查，系统分析员扼要地写出关于 问题性质、工程目标和工程规模的书面报告, 经过讨论和必要的修改之后这份报告应该得到客户的确认

##### 可行性研究：

确定问题是否可行为了回答这个问题，系统分析员需要进行一次大大压缩和简化了的系统分析和设计过程。可行性研究阶段的任务是研究问题的范围，探索这个问题是否值得去解，是否有可行的解决办法。可行性研究的结果是客户作出是否继续进行这项工程的决定的重要依据，只有投资可能取得较大效益的那些工程项目才值得继续进行下去。

##### 需求分析：

为了解决这个问题，系统需要具备怎样的功能系统分析员必须和用户密切配合，充分交流信息，以得出经过用户确认的系统逻辑模型。用数据流图、数据字典和简要的算法表示系统的逻辑模型。这个阶段的一项重要任务，是用正式文档准确地记录对目标系统的需求，这份文档通常称为软件需求规格说明书(SRS)。

#### 软件开发

##### 总体设计：

设计软件结构，确定程序由哪些模块组成以及模块间的关系软件工程师应该用适当的表达工具描述每种方案,分析每种方案,推荐一个最佳方案并制定出详细计划。另一项主要任务就是设计程序的体系结构，即确定程序由哪些模块组成以及模块间的关系。

##### 详细设计：

针对每个模块，设计详细规格说明，确定算法和数据结构详细设计阶段的任务就是把设计方案具体化，也就是回答:“应该怎样具体地实现这个系统呢?”这个阶段的任务是设计出程序的详细规格说明。在这个阶段将详细地设计每个模块，确定实现模块功能所需要的具体算法和数据结构

##### 编码和单元测试：

将详细设计内容用语言实现，并测试每个模块编码和单元测试阶段的关键任务是写出正确的、容易理解的、容易维护的程序模块。程序员把详细设计的结果翻译成用选定的高级编程语言书写的程序，编写出的每-一个模块，并对编写好的各个模块进行测试

##### 综合测试：

通过各种类型测试使软件达到预定要求，最基本的测试是集成测试和验收测试

#### 软件维护

1. 改正性维护，即诊断和改正在使用过程中发现的软件错误;

2. 适应性维护，即修改软件以适应环境的变化;

3. 完善性维护，即根据用户的要求改进或扩充软件使它更完善;

4. 预防性维护，即修改软件，为将来的维护活动预先做准备；

### 软件过程

##### 定义：

是为了获得高质量软件所需要完成的一系列任务框架，它规定了完成任务的工作步骤。通常用软件生命周期模型来描述软件过程。

##### 瀑布模型

将软件生命周期的各项活动规定为依照固定顺序连接的若干阶段工作，最终得到软件产品。

##### 快速原型模型

快速建立可运行的程序，它完成的功能往往是最终产品功能的一个子集。

##### 迭代模型

先完成一个系统子集的开发，再按同样的开发步骤增加功能，如此递增下去直至满足全部系统需求。

##### 增量模型

先完成一个系统子集的开发，再按同样的开发步骤增加功能，如此递增下去直至满足全部系统需求。

##### 螺旋模型

在每个阶段之前都增加了风险分析过程的快速原型模型。

##### 喷泉模型

典型的面向对象软件过程模型。体现了迭代和无缝的特性。

### 习题

1. 面向对象方法学中的是由描述该（ 对象 ）属性的数据以及可以对这些数据施加的所有操作封装在一起构成的统一体

2. 软件维护困难的主要原因是（ 开发方法的缺陷 ）

3. 软件工程是采用（ 工程 ）的概念、原理、技术方法指导计算机程序设计的工程学科

4. 下列属于维护阶段的文档是（ 软件问题报告 ）。

5. 随着开发小组人数的（ 增加 ），因交流开发进展情况和讨论遇到的问题而造成的通信开销也急剧增加

6. 下列不属于软件开发时期具体设计和实现的步骤是（ 问题定义 ）。

7. 因计算机硬件和软件环境的变化而作出的修改软件的过程称为（ 适应性维护 ）。

8. 软件工程的基本要素包括方法、工具和（ 过程 ）。

9. 具有风险分析的软件生命周期模型是（ 螺旋模型 ）。

10. 包含风险分析的软件工程模型是（ 螺旋模型 ）。

11. “软件危机”是指（ 软件开发和维护中出现的一系列问题 ）。

12. 下列不属于软件定义时期的步骤是（ 总体设计 ）。

13. 支持面向对象技术的软件生存周期模型是（ 喷泉模型 ）。

14. 软件质量必须在（ 分析、设计与实现过程中 ）加以保证。

15. 为了提高软件的可维护性或可靠性而对软件进行的修改称为（   预防性维护      ）

16. 软件生命周期一般可分为\_**问题定 义**\_\_\_、**可行性研究**、\_\_**需求分析**\_、**总体设计**、**详细设计**、\_**编码和单元测试**\_\_\_、**综合测试**、\_\_**软件维护**\_\_。

17. RUP软件生命周期划分的四个阶段:\_\_**初始阶段**\_\_、**精化阶段**、\_**构建阶段**\_\_\_、\_\_**移交阶段**\_

18.软件生命周期由\_\_**软件定义**\_\_、\_\_ **软件开发**\_\_和 **软件维护**3个时期组成,每个时期又进一步划分成若干个阶段。

19.软件开发时期具体设计和实现在前一个时期定义的软件,它通常由下述4个阶段组成:\_\_**总体设计、详细设计、 编码和单元测试、综合测试**。

20.与传统方法相反,面向对象方法把\_**\_数据**\_\_和\_\_**行为**\_\_看成是同等重要的,它是一种以数据为主线,把数据和对数据的操作紧密地结合起来的方法。

21.列举四种软件开发模型\_**\_瀑布模型、 快速原型模型、 增量模型、 喷泉模型、螺旋模型**等。

22.软件定义时期通常进一步划分成3个阶段,即\_**问题定义**、**可行性研究** 和 **需求分析**

**23.**软件工程基本要素:\_\_**方法**\_\_、\_\_**工具**\_\_和\_**\_过程**\_\_。

24.\_\_**软件维护**\_过程本质上是修改和压缩了的软件定义和开发过程。

## 可行性研究

### 可行性研究的任务

##### 最根本任务

对以后的行动方针提出建议

##### 具体任务

1.分析和澄清问题的定义

2.导出系统的逻辑模型：

数据流图+数据字典

3.根据逻辑模型探索若干种可供选择的解法

4.研究每种解法的可行性

经济可行性：经济效益是否大于开发成本
 技术可行性：现有技术能够实现
 操作可行性：系统操作方式是否可行

### 数据流图

##### 定义（DFD）

描述信息流和数据从输入到输出所经受的变换。没有任何具体物理部件，只是描绘数据在软件中流动和被处理的逻辑过程

##### 特点

数据流图中没有具体的物理部件，只是描绘数据在软件中流动和被处理的逻辑过程

数据流图是系统逻辑功能的图形表示，是分析员与用户之间极好的通信工程

设计时只需考虑系统必须完成的基本逻辑功能，不考虑怎样具体地实现这些功能

##### 符号



### 习题

* 1. (单选题)下列哪种数据流的符号代表的是变换数据的处理（       圆角矩形    ）。

2. (单选题)数据字典是关于数据的信息的集合,它所定义的对象都包含于（     数据流图     ）。

3. (单选题)系统流程图是描述（     物理系统       ）的工具。

4. (单选题)在系统流程图中, 平行四边形符号代表的含义是（      表示输入或输出    ）。

5. (单选题)数据字典是定义哪一种系统描述工具的数据的工具（    数据流程图       ）。

6. (单选题)程序流图中的“→”代表（       控制流 ）。

7. (单选题)以下关于数据流图的说法错误的是：（C）。
* A. 数据流图舍去了具体的物质,只剩下数据的流动、加工处理和存储
* B. 数据流图是用作结构化分析的一种工具
* ~~C. 传统的的数据流图中主要由加工、数据源点/终点、数据流、控制流、数据存储组成~~
* D. 数据流图的绘制采用自上向下、逐层分解的方法

8. (单选题)在软件开发过程中,DFD 是面向（     数据流     ）分析方法的描述工具。

9. (填空题)\_\_**系统流程图**\_\_是概括地描绘物理系统的传统工具。

10. (填空题)分层次的描绘方法便于阅读者按从\_\_**抽象**\_\_到\_\_**具体**\_\_的过程逐步深入地了解一个复杂的系统。

11. (填空题)\_\_**数据流图\_**\_是一种图形化技术,它描绘信息流和数据从输入移动到输出的过程中所经受的变换。

12. (填空题)成本/效益分析方法主要从四个方面考虑\_\_**货币的时间价值**\_\_、\_ **投资回收期**\_\_\_、**纯收入**、\_\_**投资回收率**\_\_。

13. (填空题)数据流图有4种成分：**源点或终点**、**处理**、**数据存储**  和  **数据流**。

14. (填空题)画数据流图的基本目的是利用它作为 **信息交流** 的工具。

15. (填空题)一般说来，数据字典应该由对 **数据流 、  数据元素 、  数据存储** 和**处理**4类元素的定义组成。

16. (填空题)数据字典最重要的用途是作为  **分析阶段** 的工具。

## 需求分析

### 需求分析

#### 准则

必须理解并描述问题的信息域，根据这条准则应该建立数据模型
必须定义软件应完成的功能，这条准则要求建立功能模型
必须描述作为外部事件结果的软件行为，这条准则要求建立行为模型
必须对描述信息、功能和行为的模型进行分解，用层次的方式展示细节

#### 概念

需求分析是软件定义时期的最后一个阶段,它的基本任务是准确地回答“系统必须做什么”这个问题，目标系统提出完整、准确、清晰、具体的要求。在需求分析阶段结束之前，系统分析员应该写出软件需求规明书，以书面形式准确地描述软件需求

### 需求分析任务

1）确定对系统的综合要求。

2）分析系统的数据要求。

##### 工具：常用的方法：

 建立数据模型

##### 常用图形工具：

层次方框图
 warnier图

3）导出系统的逻辑模型。

4）修正系统开发计划。

### 与用户沟通获取需求的方法

##### 访谈

 基本形式：

 正式会谈

 非正式会谈

##### 面向数据流自顶向下求精

##### 简易的应用规格说明技术

##### 快速建立软件原型

快速建立软件原型：旨在演示目标系统主要功能的可运行的程序。构建原型的要点是，它应该实现用户看得见的功能，省略目标系统的“隐含”功能。快速建立软件原型是最准确、最有效、最强大的的需求分析技术

### 分析建模与规格说明

#### 模型分类：

**数据模型（实体-联系图）**：描绘数据对象及数据对象之间的关系

**功能模型（数据流图）**：描绘数据在系统中流动时被处理的逻辑过程，指明系统具有的变换数据的功能

**行为模型（状态转换图）**：描绘系统的各种行为模式在不同状态间转换的方式

### 实体-联系图（ER图）

#### 属性

属性定义了数据对象的性质，必须把一个或多个属性定义为“标识符”，即当希望找到数据对象的一个实例时，用标识符属性作为“关键字”(“键”)。应该根据对所要解决的问题的理解，来确定特定数据对象一组合适的属性

#### 联系

数据对象彼此之间相互连接的方式称为联系，也称为关系。联系也可能有属性，分为以下3种类型：

一对一（1:1）

一对多（1:N）

多对多（M:N）

#### 实体联系图（E-R图）

定义：使用实体联系图可以建立数据模型，利用E-R图描绘的数据模型称之为E-R模型

符号表示：



### 数据规范化

第（一、二、三）范式

### 状态转换图（状态图）

##### 定义

通过描绘系统的状态及引起系统状态转换的事件来表示系统的行为。状态图还提供了 行为建模机制，指明了作为特定事件的结果系统将做哪些动作。

##### 状态

* 1. 状态是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式
	2. 状态有初态、终态和中间状态
	3. 一张状态图只能有一个初态，而终态可以没有也可以有多个

##### 事件

在某个特定时刻发生的事情，它是对引起系统做动作或(和)从一个状态转换到另一个状态的外界事件的抽象。简而言之，事件就是引起系统做动作或(和)转换状态的控制息

##### 符号





对于圆角矩形，可以将其分为上、中、下三部分：
 上部分是状态的名称；

中部分是状态变量的名字和值；

下部分是活动表

### 其他图形工具

**层次方框图**：用树形结构的一系列矩形框描绘数据的层次结构；随着结构的逐步精细，对数据结构的描绘也越来越详细。

**Warnier图**：用树形结构描绘信息的层次结构；可以表明信息的逻辑组织；可以表明某类信息出现的条件或者是否重复出现。

**IP图**：IP是输入、处理、输出图的简称，能够方便地描绘输入数据、对数据的处理和输出数据之间的关系。



### 验证软件需求正确性的四个方面

一致性：所有需求必须是一致的，任何一条需求不能和其他需求互相矛盾。
 完整性：需求必须是完整的，规格说明书应该包括用户需要的每一个功能或性能。
 现实性：指定的需求应该能用现有的硬件和软件技术可以实现。
 有效性：必须证明需求是正确有效的，确实能解决用户面对的问题。

### 习题：

需求分析的任务不包括一下那个选项(       面向数据流自顶向下求精     )。

下列不属于快速建立软件原型常用的方法和工具的是(      结构化语言       )。

在结构化分析方法中,(      数据流图      )表达系统内部数据运动的图形化技术

(      功能需求      )是对软件系统的一项基本需求。

在状态图中,终态用(      同心圆      )表示

(     实体联系图       )描绘数据对象及数据对象之间的关系,是用于建立数据模型的图形。

需求分析阶段的研究对象是(     用户要求       )。

数据字典是软件需求分析阶段的最重要工具之一,其最基本的功能是(       数据定义     )

以下哪个选项不属于可重用软构件应具备的特点(     模块独立性弱       )。

需求分析的产品是(    需求规格说明书        )。

需求分析是软件定义时期的最后一个阶段,它的基本任务是准确地回答(      系统必须做什么       )问题

实体联系图(ER图)中包括那三个元素:  **数据对象、属性、联系。**

在访问用户的过程中使用\_\_**情景分析技术\_**\_往往非常有效

需求分析阶段得出的\_\_\_***数据流图***\_是总体设计的极好的出发点

快速原型应该具备的特性：**快速、 容易修改**

需求分析的目标之一就是把**数据流**和**数据存储**定义到**元素级**。

访谈有两种基本形式,分别是\_\_**正式访谈**\_\_和\_\_**非正式的访谈**\_\_。

软件需求规格说明书 是**需求分析阶段**得出的最主要的文档.

在状态图中,初态用\_\_**实心圆**\_\_表示,终态用一对\_\_**同心圆**\_\_表示

结构化分析方法就是面向数据流  **自顶向下**    进行需求分析的方法。

## 总体设计

### 总体设计概念

#### 定义

总体设计的基本目的就是回答“系统应该如何实现”这个问题，又称为概要设计或初步设计

#### 主要任务

###### 划分出组成系统的物理元素程序、文件、数据库、人工过程和文档等，但是每个物理元素仍然处于黑盒子级，这些黑盒子里的具体内容将在以后仔细设计

###### 设计软件的结构，也就是要确定系统中每个程序是由哪些模块组成的，以及这些模块相互间的关系

### 设计过程（包括系统设计阶段和结构设计阶段）

##### 系统设计阶段：确定系统具体实现方案

###### 设想供选择的方案

###### 选取合理的方案

###### 推荐最佳方案

##### 结构设计阶段：确定软件结构

###### 功能分解

###### 设计软件结构

###### 设计数据库

###### 制定测试计划

###### 书写文档

主要有：系统说明、用户手册、测试计划、详细实现计划、数据库设计结果

###### 审查和复查

### 设计原理

#### 模块化

#### 抽象

#### 逐步求精

#### 信息隐藏和局部化

#### 模块独立（高内聚、低耦合）

##### “模块独立的概念是模块化、抽象、信息隐藏和局部化概念的直接结果。其质量标准是耦合和内聚”

##### 模块耦合及其分类：

###### **耦合度从低到高**：完全独立<数据耦合<特征耦合<控制耦合<外部耦合<公共耦合<内容耦合

###### **要求**：低耦合、使用数据耦合、不用内容耦合

###### 完全独立：如果两个模块中的每一个都能独立地工作而不需要另一个模块的存在，则称它们彼此完全独立，耦合程度最低。

###### 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，则称它们是数据耦合。

###### 特征耦合： 如果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,则称它们是特征耦合。

###### 控制耦合： 如果两个模块彼此间通过参数交换信息，并且传递的信息中包含控制信息(这种控制信息可以以数据的形式出现)，则称它们是控制耦合。

###### 外部耦合： 一组模块都访问同一全局简单变量，而且不通过参数表传递该全局变量的信息，则称之为外部耦合。

###### 公共耦合： 如果两个或多个模块通过一个公共数据环境相互作用,则称它们是公共环境耦合。

###### 内容耦合： 内容耦合是最高程度的耦合，一个模块直接访问另一模块的内容，则称这两个模块为内容耦合。

##### 模块内聚及其分类：

###### 内聚度从低到高： 偶然内聚、逻辑内聚、时间内聚——低内聚 过程内聚、通信内聚——中内聚 顺序内聚、功能内聚——高内聚

### 启发规则

改进软件结构提高模块独立性。

模块规模应该适中。

深度、宽度、扇入和扇出应适当。

模块的作用域应该在控制域之内。

力争降低模块接口的复杂程度。

设计单入口单出口的模块。

模块功能应该可以预测但要防止过分局限。

### 描述软件结构的图形工具

#### 层次图和HIP图

层次图用于**描绘软件的层次结构**，其中一个矩形框代表一个模块，方框间的连线表示调用关系而不像层次方框图那样表示组成关系。

#### 结构图

##### 结构图不仅描述调用关系，还描述传递的信息和调用方式

##### 符号

###### 方框代表模块、框内注明模块的名字或主要功能。

###### 箭头或直线表示调用关系

###### 尾部是空心圆表示传递的是数据；若是实心圆则表示传递的是控制信息

### 面向数据流的设计方法

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

#### 信息流两种类型

##### 变换流

##### 事务流，以事务为中心

##### 设计过程

#### 变换分析



### 习题

软件设计中划分模块的一个准则是（      高内聚低耦合      ）。

为了提高模块的独立性，模块内部最好是（     功能内聚      ）。

试判断下列叙述中,哪个(些)是正确的(     事务分析和变换分析的设计步骤是基本相似的        )。

结构化方法的基本原则是（  抽象与分解         ）。

模块的内聚性最高的是（     功能内聚       ）。

模块独立性是软件模块化所提出的要求，衡量模块独立性的度量标准则是模块的（     内聚性和耦合性      ）。

模块的独立程度可以由两个定性标准度量,这两个标准分别称为\_\_**内聚**\_\_和\_\_**耦合**\_\_。

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。

信息流有下述两种类型:\_**变换流\_**\_\_和\_**事务流**\_\_\_。

\_**\_抽象**\_与求精是一对互补的概念。

\_**层次图\_**\_用来描绘软件的层次结构。

模块耦合分为 **数据耦合、 控制耦合、 特征耦合、 公共环境耦合、内容耦合** ，应该坚决避免使用\_\_**\_内容耦合**\_。

 **逐步求精**\_定义为为了能集中精力解决主要问题而尽量推迟对问题细节的考虑。

模块内的\_**高内聚**\_往往意味着模块间的松耦合。

总体设计阶段主要由\_\_**系统设计**\_\_和\_**\_结构设计**\_\_两阶段组成。

\_\_**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。

\_\_**模块化\_**\_是为了使一个复杂的大型程序能被人的智力所管理,是软件应该具备的唯一属性

\_**\_模块\_**是构成程序的基本构件。

**\_内聚**\_衡量一个模块内部各个元素彼此结合的紧密程度。

耦合强弱取决于\_**\_模块间接口的复杂程度**\_\_。

## 详细设计

###### 目标：确定怎样具体地实现所要求地系统以及确定每个模块的算法。

### 结构程序设计

### 人机界面设计考虑的四个问题

##### 系统响应时间

##### 用户帮助设施

##### 出错信息处理

##### 命令交互

### 过程设计的工具

#### 程序流程图

##### 符号



##### 优点：对控制流程的描绘很直观，便于初学者掌握。

##### 缺点：

程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构，程序流程图中用箭头代表控制流，因此程序员不受任何约束，可以完全不顾结构程序设计的精神，随意转移控制程序流程图不易表示数据结构。

#### 盒图（N-S）

出于要有一种不允许违背结构程序设计精神的图形工具的考虑，提出了盒图

##### 符号：



#### PAD图

##### 符号



#### 判定表

#### 判定树

#### 过程设计语言（PDL）

**PDL也即伪代码**，是用正文形式表示数据和处理过程的设计工具。

### 面向数据结构的设计方法

#### Jacksn图

数据元素之间的逻辑关系只有：**顺序、选择、重复**

#### 改进的Jacksn图

#### Jacksn方法

“Jacksn 设计方法是由英国的 M. Jacksn 提出的,它是一种面向数据结构的软件设计方法”

### 程序复杂程度的定量度量

#### McCabe方法

##### 流图

###### “流图实质上是“退化了的”程序流程图，它仅仅描绘程序的控制流程，完全不表现对数据具体操作以及分支或循环的具体条件，流图通常被用来突出表示程序的控制流”

###### “把程序流程图映射为流图”

##### 环形复杂度

###### 定义：

“McCabe方法根据程序控制流的复杂程度定量度量程序的复杂程度，这样度量出的结果称为程序的环形复杂度”

###### 计算方法

环形复杂度定量度量程序的逻辑复杂度，可以用3种方法（P137）中的任何一种来计算环形复杂度:

流图中的区域数等于环形复杂度

流图G的环形复杂度V(G)=E-N+2,E是流图中边的条数，N是结点数

流图G的环形复杂度V(G)=P+1，其中，P是流图中判定结点的数目。

#### Halstead方法

##### 定义

Halstead方法是根据程序中运算符和操作数的总数来度量程序的复杂程度。

##### 方法

P139下方

### 习题：

程序的三种基本控制结构是（     顺序、选择和重复      ）。

人机界面设计指南不包括（   数据输出指南     ），包括（一般交互指南、信息显示指南、数据输入指南）

PDL 是（    伪码式    ）语言。

用户帮助设施中组织帮助信息的方法不包括（   帮助菜单     ），包括（平面结构、信息的层次结构、超文本结构）

Jacksn 设计方法是由英国的 M. Jacksn 提出的,它是一种面向（   数据结构     ）的软件设计方法。

详细设计的任务是确定每个模块的（   算法     ）。

软件复杂性度量的参数包括（    规模    ）、难度、结构、智能度。

系统响应时间有两个重要属性,分别是\_\_**长度**\_\_和\_\_**易变性**\_\_。

常见的用户帮助设施可分为\_\_**集成的**\_\_和\_**附加的\_**\_\_两类。

使用最广泛的描述过程设计的方法是\_**程序流程图**\_\_\_。

\_流图\_实质上是“退化了的”程序流程图,描绘程序的\_\_**控制流程**\_\_,不表现对\_\_**数据**\_\_的具体操作以及\_\_**分支或循环**\_\_的具体条件

\_\_**结构程序设计**\_\_技术是进行详细设计的逻辑基础。

\_\_**过程设计**\_\_是详细设计阶段完成的主要工作。

在开发有清楚的层次结构时可采用\_\_**面向数据结构\_**\_的设计方法完成设计过程设计。

使用\_\_**环形复杂度\_**\_可以定量度量程序的复杂程度

\_\_\_**黑盒测试**\_把程序看作一个黑盒子,完全不考虑程序的内部结构和处理过程。

## 实现、测试

### 编码

#### 定义：

“编码是把软件设计结果翻译成用程序设计语言书写的程序,是对设计的进一步具体化，因此程序的质量主要取决于软件设计的质量。”

#### 程序设计语言的选择

#### 编码风格（P148）

##### 程序内部的文档

###### 定义：程序内部的文档包括恰当的标识符、适当的注解和程序的视觉组织

###### 命名规则

###### 注释规则

###### 程序清单布局规则

##### 数据说明

###### 数据说明的次序应该标准化。有次序就容易查阅，因此能够加速测试、调试和维护的过程

###### 多个变量名在一个语句中说明时，应该按字母顺序排列这些变量

###### 使用复杂的数据结构，应用注解说明用设计语言实现这个数据结构的方法和特点

##### 语句构造

###### 原则：每个语句都应该简单而直接，不能为了提高效率而使程序变得过分复杂

###### 使语句简单明了的规则

##### 输入输出

##### 效率

### 软件测试基础

#### 软件测试目标

#### 软件测试准则

#### 软件测试方法

（1）黑盒测试（功能测试）

“把软件看成一个黑盒子，不考虑其内部结构和处理过程，只按照规格说明书的规定，测试软件是否能够正确接收输入数据，并产生正确的输出数据。也即测试程序是否正确实现了其功能”

（2）白盒测试

“把软件看作一个透明的盒子，完全知道程序内部结构和处理算法，根据程序内部的逻辑结构测试程序内部的主要执行通路是否能够按照预定的要求正确工作”

#### 测试方法

（1）单元测试（模块测试）

”单元测试是把每个模块作为一个单独的实体来测试，检验其正确性。目的在于保证每个模块作为一个单元能够正确运行模块测试所发现的是编译和详细设计的错误“

（2）子系统测试

“子系统测试是把经过单元测试的模块放在一起形成一个子系统来测试。模块相互间协调和通信是此测试的主要问题，也即子系统测试着重测试模块的接口”

（3）系统测试

“系统测试是把经过测试的子系统装配成一个完整的系统来测试。在这个过程中不仅应该发现设计和编码的错误，还应该验证系统确实能够提供需求说明书中指定的功能，而且系统动态特性也符合预定要求系统测试发现的往往是软件设计中的错误，也可能发现需求说明书中的错误”

（4）验收测试（确认测试）

“验收测试是把软件系统作为单一的实体进行测试，它是在用户积极参与下进行的，而且主要使用实际数据进行测试，验收测试的目的是验证系统确实能够满足用户的需要”
验收测试发现的是系统需求说明书中的错误

（5）平行运行

“平行运行就是同时运行新开发出来的系统和将被它取代的旧系统，以便比较新旧两个系统的处理结果”

### 单元测试

#### 概念

单元测试集中检测软件设计的最小单元一模块，它和编码属于软件过程的同一个阶段。在编写出源程序代码并通过了编译程序的语法检查之后，就可以用详细设计描述作指南，对重要的执行通路进行测试，以便发现模块内部的错误。单元测试主要使用白盒测试技术，而且对多个模块的测试可以并行地进行，包括人工测试和计算机测试两种

#### 测试重点

（1）模块结构

（2）局部数据结构

（3）重要的执行通路

（4）出错处理通路

（5）边界条件

#### 测试方法

（1）代码审查

（2）计算机测试

### 集成测试

#### 概念

（1）方法分类

非渐增测试： 先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序

渐增测试： 把下一个要测试的模块同已经测试好的那些模块结合起来进行测试,测试完以后再把下一个应该测试的模块结合进来测试，每次增加一个模块。渐增式测试同时完成单元测试和集成测试

#### 渐增式测试策略（P157）

（1）自顶向下集成：从主控制模块开始，沿着程序的控制层次向下移动，逐渐把各个模块结合起来。在把附属于主控制模块的模块组装到程序结构中时，使用深度优先的策略或宽度优先的策略

①：深度优先

深度优先的结合方法先组装在软件结构的一条主控制通路上的所有模块

②：宽度优先

宽度优先的结合方法是沿软件结构水平地移动,把处于同一个控制层次上的所有模块组装起来

（2）自底向上集成：自底向上测试从软件结构最低层的模块开始组装和测试。因为是从底部向上结合模块，总能得到所需的下层模块处理功能，所以不需要存根程序

#### 回归测试

定义：回归测试是指重新执行已经做过的测试的某个子集，以保证上述这些变化没有带来非预期的副作用。它可以用于保证由于调试或其他原因引起的变化，不会导致非预期的软件行为或额外错误的测试活动

### 确认测试

#### 概念："确认测试也称为验收测试，它的目标是验证软件的有效性"

验证： 为了保证软件正确的实现某个特定要求而进行的一系列活动
确认：为了保证软件确实满足了用户需求而进行的一系列活动
软件有效性：如果软件的功能和性能如同用户所合理期待的那样，软件就是有效的

#### 确认测试的范围

#### 软件配置复查

#### Alpha测试Beta测试

###### "Alpha测试由用户在开发者的场所进行，并且在开发者对用户的“指导”下进行测试，且开发者负责记录发现的错误和遇到的问题。即Alpha测试是在受控的环境中进行的"

###### "Beta测试由软件的最终用户们在一个或多个客户场所进行。开发者通常不在Beta测试的现场，即Beta测试是软件在开发者不能控制的环境中的“真实”应用"

### 白盒测试技术

#### 逻辑覆盖

##### 定义：“逻辑覆盖是对一系列测试过程的总称， 这组测试过程逐渐进行越来越完整的通路测试“

##### 分类：

###### 语句覆盖：选择足够多的测试数据，被测试程序中的每条语句至少执行一次

###### 判定覆盖：不仅每个语句至少执行一次，而且每个判定的每种可能的结果都应该至少执行一次

###### 条件覆盖：不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果

###### 判定/条件覆盖：选择足够多的测试数据，使判定表达式中的每个条件都取到各种可能的结果，而且每个判定表达式也都取到各种可能的结果。它同时满足判断覆盖和条件覆盖

###### 条件组合覆盖：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。满足条件组合覆盖，也一定满足判定覆盖、条件覆盖和判断/条件覆盖

###### 点覆盖：连通图G的子图G’是连通的，而且包含G的所有结点，则称G’是G的点覆盖。满足点覆盖标准要求选取足够多的测试数据，使得程序执行路径至少经过流图的每个结点一次，也即点覆盖标准和语句覆盖标准是相同的

###### 边覆盖：连通图G的子图G'’是连通的，而且包含G的所有边，则称G'’是G的边覆盖。为满足边覆盖的测试标准，要求选取足够多的测试数据，使程序执行路径至少经过流图每条边一次，也即边覆盖与判定覆盖是相同的

###### 路径覆盖：选取足够多的测试数据，使程序的每条可能路径都至少执行一次，如果程序图中有环，则要求每个环至少经过一次

#### 控制结果测试

###### 基本路径测试:"基本路径测试是TmMcCabe提出的一种白盒测试技术。使用这种技术设计测试用例时，首先计算程序的环形复杂度，并用该复杂度为指南，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以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"

###### 循环测试:"循环是绝大多数软件算法的基础，但是,在测试软件时却往往未对循环结构进行足够的测试。循环测试是一种白盒测试技术，它专注于测试循环结构的有效性"

1. 循环测试分类:

通常只有3种循环，即简单循环、串接循环和嵌套循环

### 黑盒测试技术

#### 概念

##### 目的

###### 功能不正确或遗漏了功能

###### 界面错误

###### 数据结构错误或外部数据库访问错误

###### 性能错误

###### 初始化和终止错误

##### 适用性

白盒测试在测试过程的早期阶段进行，黑盒测试主要用于测试过程的后期

#### 技术方法：

##### 等价划分

###### 定义：这种技术把程序的输入域划分成若干个数据类，据此导出测试用例，一个理想的测试用例能独自发现一类错误

###### 目的：等价划分法力图设计出能发现若干类程序错误的测试用例，从而减少必须设计的测试用例的数目

###### 流程：

划分数据的等价类

根据等价类设计测试方案

##### 边界值分析

##### 错误推测

### 调试

#### 定义

调试是在测试发现错误之后排除错误的过程

#### 调试过程

#### 调试途径

##### 蛮干法

##### 回溯法

##### 原因排除法

### 软件可靠性

### 习题：

测试单个类的方法不包括（     轮流测试      ）

\_**等价划分**\_\_把程序的输入域划分成若干个数据类,据此导出测试用例。

\_\_**白盒测试**\_在测试过程的早期阶段进行,而\_**黑盒测试**\_\_\_主要用于测试过程的后期。

由模块组装成程序时有两种方法,分别是\_\_**非渐增式测试**\_\_方法和\_\_**渐增式测试**\_\_方法。

模块的环形复杂度越\_**高**\_,可执行的路径就越多,因此,全面测试它的难度就越\_\_**大**\_\_。

单元测试主要使用\_\_白盒测试\_\_技术,而且对多个模块的测试可以并行地进行。

大型软件系统的测试过程基本上由\_**模块测试**\_\_\_、\_\_**子系统测试**\_\_、\_\_**系统测试**\_\_、\_\_**验收测试**\_\_和\_\_**平行运行**\_\_等五个步骤组成。

通常设计测试方案时总是联合使用\_\_**等价划分**\_\_和\_\_**边界值分析**\_\_两种技术。

 白盒 测试方法按照程序内部的逻辑测试程序,检测程序中的主要执行通路是否都能按预定要求正确工作。

通常把测试数据和预期的输出结果称为\_\_**测试用例**\_\_。

子系统测试和系统测试,都兼有检测和组装两重含义,通常称为\_\_**集成测试**\_\_。

## 维护

### 软件维护定义

#### 定义

软件维护是在软件已经交付使用后，为了改正错误或满足新的需要而修改软件的过程，是软件生命周期的最后一个阶段，其基本任务是保证软件在一个相当长的时期能够正常运行

#### 分类

改正性维护：诊断和改正错误的过程（17%~21%）

适应性维护：为了和变化了的环境适当地配合而进行的修改软件的活动（18%~25%）

完善性维护：为了满足用户提出的增加新功能或修改已有功能的要求和一般性改进要求（50%~66%）

预防性维护：当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（4%）

### 软件维护特点

#### 结构化维护和非结构化维护差别巨大

非结构化维护：唯一成分是程序代码，维护活动从艰苦地评价程序代码开始，需要付出很大代价

结构化维护：有完整的软件配置存在，维护工作从评价设计文档开始

#### 维护的代价高昂

#### 维护存在很多问题

### 软件的可维护性

#### 概念

可维护性指的是维护人员理解、改正、改动或改进这个软件的难易程度。提高可维护性是支配软件工程方法学所有步骤的关键目标

#### 决定软件可维护性的因素

（1）可理解性

“软件可理解性表现为外来读者理解软件的结构、功能、接口和内部处理过程的难易程度”

（2）可测试性

“诊断和测试的容易程度取决于软件容易理解的程度”

要求：维护人员需要得到在开发阶段用过的测试方案，以便进行回归测试。在设计阶段应该尽力把软件设计成容易测试和容易诊断的

衡量标准：对于程序模块来说，可以用程序复杂度来度量它的可测试性。模块的环形复杂度越大，可执行的路径就越多，全面测试它的难度就越高

（3）可修改性

“耦合、内聚、信息隐藏、局部化、控制域与作用域的关系等，都影响软件的可修改性”

（4）可移植性

“软件可移植性是指把程序从一种计算环境(硬件配置和操作系统)转移到另一种计算环境的难易程度”

（5）可重用性

“重用是指同一事物不做修改或稍加改动就在不同环境中多次重复使用”

### 可预防性维护

### 软件再工程过程

#### 该模型对应的6类活动

###### 库存目录分析

###### 文档重构

###### 逆向工程

###### 代码重构

###### 数据重构

###### 正向工程

### 习题

软件维护产生的副作用,是指（因修改软件而造成的错误）

维护中因删除一个标识符而引起的错识是（      编码    ）副作用。

面向维护的技术涉及软件开发的（     所有   ）阶段。

可维护性的特性中,相互矛盾的是（     效率与可修改性     ）。

软件维护的副作用主要有哪几种？（     编码副作用、数据副作用、文档副作用    ）

文档可以分为\_\_**用户文档**\_\_和\_\_**系统文档**\_\_两类。

\_**文档\_**\_是影响软件可维护性的决定因素。

\_**改正性维护\_**\_必须预先进行调试以确定错误的具体位置。

## 面向对象方法学引论

### 面向对象方法学概述

#### 面向对象学的要点

##### 基本原则

面向对象方法学的出发点和基本原则，是尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界解决问题的方法与过程，使描述问题的问题空间(问题域)与实现解法的解空间(求解域)在结构上尽可能一致。

##### 定义

面向对象方法是一种以数据或信息为主线，把数据和处理相结合的方法，即把对象作为由数据及可以施加在这些数据上的操作所构成的统一体。

 = bject（对象） + classes（类） + inheritance（继承） + cmmunicatin with messages（封装）

##### 要点

###### **对象**：面向对象的软件系统是由对象组成的，软件中的任何元素都是对象，复杂的软件对象由比较简单的对象组合而成。用对象分解取代了传统方法的功能分解，对象是从客观世界中的实体抽象而来的，是不固定的。

###### **类**：把所有对象都划分成各种对象类,每个对象类都定义了一组数据和一组方法。数据用于表示对象的静态属性,是对象的状态信息。类中定义的方法，是允许施加于该类对象上的操作，是该类所有对象共享的，并不需要为每个对象都复制操作的代码

###### **继承性**：按子类与父类的关系，把若干个对象类组成一个层次结构的系统。子类自动具有和上层的父类相同的数据和方法，而且低层的特性将屏蔽高层的同名特性

###### **封装性**：对象彼此之间仅能通过传递消息互相联系。对象是进行处理的主体，必须发消息请求它执行它的某个操作，处理它的私有数据，而不能从外界直接对它的私有数据进行操作。一切局部于该对象的私有信息，都被封装在该对象类的定义中，就好像装在一个不透明的黑盒子中一样，在外界是看不见的，更不能直接使用,

#### 面向对象学的优点

1.与人类习惯的思维方法一致

2.稳定性好

3.可重用性

4.较易开发大型软件产品

5.可维护性好

### 面向对象概念

#### 对象

##### 对象的定义

###### 对象是具有相同状态的一组操作的集合

###### 对象是对问题域中某个东西的抽象，这种抽象反映了系统保存有关这个东西的信息或与它交互的能力

###### 对象：：=<ID,MS,DS,MI>

##### 对象特点

###### 以数据为中心

###### 对象是主动的

###### 实现了数据封装

###### 具有并行性

###### 模块独立性好

#### 其他概念

##### 类

““类”是对具有相同数据和相同操作的一组相似对象的定义,即类是对具有相同属性和行为的一个或多个对象的描述，包括对怎样创建该类的新对象的说明。类是支持继承的抽象数据类型，而对象就是类的实例”

##### 实例

“实例就是由某个特定的类所描述的一个具体的对象。类是对具有相同属性和行为的一组相似的对象的抽象，类在现实世界中并不能真正存在。实际上类是建立对象时使用的“样板”，按照这个样板所建立的一个个具体的对象，就是类的实际例子，通常称为实例”

##### 消息

消息就是要求某个对象执行在定义它的那个类中所定义的某个操作的规格说明。

##### 方法

方法就是对象所能执行的操作，也就是类中所定义的服务。

##### 属性

属性就是类中所定义的数据，它是对客观世界实体所具有的性质的抽象

##### 封装

###### “封装是把数据和实现操作的代码集中起来放在对象内部。封装也就是信息隐藏，通过封装对外界隐藏了对象的实现细节”

###### 特点

1. 有一个清晰的边界
2. 有确定的接口
3. 受保护的内部实现

##### 继承

广义地说，继承是指能够直接获得已有的性质和特征，而不必重复定义它们。在面向对象的软件技术中，继承是子类自动地共享父类中定义的数据和方法的机制

##### 多态性

多态性是指子类对象可以像父类对象那样使用，同样的消息既可以发送给父类对象也可以发送给子类对象。即在类等级的不同层次中可以共享一个方法的名字， 不同层次中的每个类各自按自己的需要来实现这个行为

##### 重载

1. 函数重载：在同一作用域，若干个参数特征不同的函数可以使用相同的函数名
2. 运算符重载：同一个运算符可以施加于不同类型的操作数上面

### 面向对象建模

#### 对象模型：描述系统数据结构

#### 功能模型：描述系统功能

#### 动态模型：描述系统控制结构

### 对象模型

#### 概念

##### 定义

对象模型表示静态的、结构化的系统的数据性质。它是对模拟客观世界实体的对象以及对象彼此间的关系的映射，描述了系统的静态结构。对象模型为建立动态模型和功能模型，提供了实质性的框架。

##### 工具

使用UML（统一建模语言）提供的类图来建立对象模型。在UML中，类的实际含义是一个类及属于该类的对象。

1. 用例图：从用户角度描述系统功能。
2. 类图：描述系统中类的静态结构。
3. 对象图：系统中的多个对象在某一时刻的状态。
4. 状态图：是描述状态到状态控制流，常用于动态特性建模 。
5. 活动图：描述了业务实现用例的工作流程 。
6. 顺序图：对象之间的动态合作关系，强调对象发送消息的顺序，同时显示对象之间的交互 。
7. 协作图：描述对象之间的协助关系 。
8. 构件图：一种特殊的UML图来描述系统的静态实现视图 。
9. 部署图：定义系统中软硬件的物理体系结构 。
10. 包图：对构成系统的模型元素进行分组整理的图 。
11. 组合结构图：表示类或者构建内部结构的图 。
12. 交互概览图：用活动图来表示多个交互之间的控制关系的图。

#### 类图基本符号

##### 定义类

###### 表示

UML中类的图形符号为长方形，用两条横线把长方形分上、中、下3个区域，3个区域分别放类的名字、属性和服务



###### 命名规则

1. 标准术语、确切含义、名词短语

##### 定义属性

###### 格式：



###### 可见性有三种：

1. 公有的（+）、私有的（-）和保护的（#）

###### 类型名：表示该属性的数据类型

##### 定义服务

###### 格式：



###### 可见性：有公有的（+）、私有的（-）和保护的（#）

###### 参数表：用逗号隔开不同参数，每个参数语法为 “参数名:类型名=默认值”

#### 表示关系的符号

##### 关联

###### 定义：

1. “关联表示两个类的对象之间存在某种语义上的联系。”

###### 普通关联

1. 定义：普通关联是最常见的关联关系，只要在类与类之间存在连接关系就可以用普通关联表示。



###### 限定关联

1. 定义：

常用在一对多或多对多的关联关系中，可以把模型中的重数从一对多变成一对一。



###### 关联类

1. 定义

为了说明关联的性质，可能需要一些附加信息。关联类可以用来记录相关信息。



##### 聚集（关联的特例）

定义：“表示类与类之间的关系是整体与部分的关系”

共享聚集

“如果在聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成，则该聚集称为共享聚集”



组合聚集

“如果部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也会随之消失， 则该聚集称为组合聚集”



##### 泛化（本质就是继承）

###### 定义：“它是通用元素和具体元素之间的一种分类关系。具体元素完全拥有通用元素的信息，并且还可以附加一些其他信息。在UML中，用一端为空心三角形的连线表示泛化关系，三角形的顶角紧挨着通用元素”

###### 普通泛化

1. 抽象类：“没有具体对象的类称为抽象类。”
2. 具体类：“具体类有自己的对象，并且该类的操作都有具体的实现方法。”

###### 受限泛化

1. 定义：“可以给泛化关系附加约束条件，以进一步说明该泛化关系的使用方法或扩充方法，这样的泛化关系称为受限泛化。”
2. 约束（预定义的约束有4种）
3. 多重：一个子类可以同时多次继承同一个上层基类
4. 不相交：一个子类不能多次继承同一个基类。-般的继承都是不相交继承
5. 完全：父类的所有子类都已在类图中穷举出来了
6. 不完全：父类的子类并没有都穷举出来，随着对问题理解的深入，可不断补充和维护。是默认的继承关系

##### 依赖和细化

###### 依赖关系

1. “依赖关系描述两个模型元素之间的语义连接关系:其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变了，将影响依赖于它的模型元素。”

###### 细化关系

1. “对同一个事物在不同抽象层次上描述时，这些描述之间具有细化关系。”

### 动态模型

#### 概念

动态模型表示瞬时的、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

#### 建模

用UML提供的状态图来描绘对象的状态、触发状态转换的事件以及对象的行为。每个类的动态行为用一张状态图来描绘，各个类的状态图通过共享事件合并起来，从而构成系统的动态模型，即动态模型是基于事件共享而互相关联的一组状态图的集合

### 功能模型

#### 概念

###### 功能模型表示变化的系统的功能性质，它指明了系统应该做什么，因此更直接地反映了用户对目标系统的需求

###### 功能模型由一组数据流图组成

#### 用例图

##### 在UML中把用用例图建立起来的系统模型称为用例模型，是进行需求分析和建立功能模型的强有力工具

##### 定义：

##### 用例模型描述的是外部行为者所理解的系统功能。用例模型的建立是系统开发者和用户反复讨论的结果，它描述了开发者和用户对需求规格所达成的共识。

##### 表示

###### 系统：“系统被看作是一个提供用例的黑盒子”

###### 用例：“一个用例是可以被行为者感受到的、系统的一个完整的功能”

1. 用例代表某些用户可见的功能，实现一个具体的用户目标
2. 用例总是被行为者启动的，并向行为者提供可识别的值
3. 用例必须是完整的

###### 行为者：“行为者是指与系统交互的人或其他系统，它代表外部实体”

###### 用例间关系

1. 扩展关系：“向一个用例中添加一些动作后构成了另一个用例”
2. 使用关系：“一个用例使用另一个用例时”

### 三种模型之间的关系

###### 类是对具有相同属性和行为的一组相似对象的定义。应该按照子类、父类的关系,把众多的类进一步组织成一个层次系统，这样做了之后,如果不加特殊描述,则处于下一层次上的类可以自动继承位于上一层次的类的属性和行为。

###### 它们分别是描述系统静态结构的对象模型、描述系统控制结构的动态模型。以及描述系统计算结构的功能模型其中对象模型是最基本、最核心、最重要的。

###### 通常,使用 UML 的类图来建立对象模型，使用 UML的状态图来建立动态模型,使用数据流图或 UML 的用例图来建立功能模型。在 UML 中把用用例图建立起来的系统模型称为用例模型

### 习题：

（     聚集      ）也称为聚合,是关联的特例。它表示类与类之间的关系是整体与部分的关系。

用面向对象方法开发软件,通常需要建立3种形式的模型。其中（    对象模型        ）是最基本、最核心、最重要的。

两个或多个对象之间的相互依赖、相互作用的关系就是（      关联      ）。

当对同一个事物在不同抽象层次上描述时,这些描述之间具有（     细化关系       ）。

给泛化关系附加约束条件,以进一步说明该泛化关系的使用方法或扩充方法,这样的泛化关系称为受限泛化。
预定义的约束有4种：多重、（       不相交      ）、完全和不完全。这些约束都是语义约束。

由某个特定的类所描述的一个具体的对象是（        实例     ）。

（         用例图      ）定义了系统的功能需求,它是从系统的外部看系统功能,并不描述系统内部对功能的具体实现。

类与类之间通常有关联、泛化(继承)、依赖和（   细化      ）4种关系。

以下哪种不属于面向对象语言？ （       C语言      ）。—— Smalltalk，C++，EIFFEL是面向对象语言。

(      功能模型      )表明了系统中数据之间的依赖关系,以及有关的数据处理功能,它由一组数据流图组成。

（功能模型）由一组数据流图组成。它的建立有助于软件开发人员更深入地理解问题域,改进和完善自己的设计

在面向对象的程序中,（           封装        ）是指把数据和实现操作的代码集中起来放在对象内部。

(      模块独立性弱   )  不属于可重用软构件应具备的特点。
相反：**高度可塑性**。**接口清晰**、**简明、可靠**。**需求模型和规格说明清晰**。都属于可重用软构件应具备的特点。

（方法）是对象所能执行的操作,也就是类中所定义的服务。

软件\_**\_可移植性\_**\_指的是,把程序从一种计算环境(硬件配置和操作系统)转移到另一种计算环境的难易程度

属性的可见性(即可访问性)通常有下述3种:\_\_**公有的**\_\_、**\_私有的**\_\_\_和\_\_**保护的\_**\_。

类与类之间通常有\_**\_关联**\_\_、\_**泛化\_**\_\_、\_**依赖\_**\_\_和\_**\_细化**\_\_4种关系

对象模型描述了数据流图中的\_\_**\_数据流\_**、\_\_**数据存储**\_\_以及**数据源点/终点**的结构。

UML用例之间主要有\_**\_扩展\_**\_和\_\_**使用**\_\_两种关系,它们是泛化关系的两种不同形式。

系统中每个对象都属于一个特定的对象类。\_**\_类\_**\_是对具有相同属性和行为的一组相似对象的定义。按照**\_子类\_**\_\_和\_\_**父类**\_\_的关系,把众多的类进一步组织成一个层次系统,处于下一层次上的类可以自动继承位于上一层次的类的属性和行为。

\_\_\_**功能模型\_**指明了系统应该“做什么”;\_**\_动态模型**\_\_明确规定了什么时候(即在何种状态下接受了什么事件的触发)做;\_\_**对象模型\_**\_则定义了做事情的实体。

## 面向对象分析

### 面向对象分析的基本过程

#### 概述

就是抽取和整理用户需求并建立问题域精确模型的过程

#### 3个子模型与5 个层次

##### 3个子模型

###### 面向对象建模得到的模型包含系统的三个要素

1. 静态结构（对象模型）：解决任一问题，都需要从客观世界实体及实体间相互关系抽象出极有价值的对象模型
2. 交互次序（动态模型）：当问题涉及交互作用和时序时，动态模型是重要的
3. 数据变换（功能模型）：解决运算量很大的问题，则涉及重要的功能模型

##### 5个层次

###### 定义

1. 复杂问题的对象模型由5个层次组成:主题层、类与对象层、结构层、属性层和服务层

###### 主题

1. 主题是指导读者理解大型、复杂模型的一种机制

###### 7 ± 2原则

1. 心理研究表明，人类的短期记忆能力一般限于一次记忆5~9个对象

###### 面向对象分析顺序

1. 建立对象模型5个主要活动
2. 寻找类与对象
3. 识别结构
4. 识别主题
5. 定义属性
6. 定义服务

###### 建模要点

1. 面向对象分析不可能严格按照顺序线性进行。
2. 必须在反复分析中对初始模型中不准确、不完整和错误的内容加以扩充和更正。
3. 仔细研究类似问题的分析结果，尽可能重用这些结果。

### 需求陈述（P233）

###### 内容

###### 书写要点

### 建立对象模型

#### 基本概念

###### 面向对象分析的首要工作就是建立问题域的对象模型。

###### 对象模型表示静态的、结构化的系统的数据性质。它是对模拟客观世界实体的对象以及对象彼此间的关系的映射，描述了系统的静态结构。对象模型为建立动态模型和功能模型，提供了实质性的框架。

#### 确定类与对象

（1）找出候选的类与对象。

（2）筛选出正确的类与对象。

#### 确定关联

###### 两个或多个对象之间的相互依赖、相互作用的关系就是关联

###### 分析确定关联，能促使分析员考虑问题域的边缘情况，有助于发现尚未被发现的类与对象

###### 步骤

1. 初步确定关联
2. 筛选（筛选标准）：
3. 已删去的类之间的关联
4. 与问题无关的或应在实现阶段考虑的关联
5. 瞬时事件
6. 三元关联
7. 派生关联

###### 改进：完善经筛选后余下的关联

1. 正名
2. 分解
3. 补充
4. 标明重数

#### 划分主题

#### 确定属性

##### 概念

属性是对象的性质，借助于属性人们能对类与对象和结构有更深入更具体的认识

##### 确定属性步骤（P242）

###### 分析

###### 选择

#### 识别继承关系

##### 建立继承关系的方式：

###### **自底向上**：抽象出现有类的共同性质泛化出父类，这个过程实质上模拟了人类归纳思维的过程

###### **自顶向下**：把现有类细化成更具体的子类,这模拟了人类的演绎思维过程。从应用域中常常能明显看出应该做的自顶向下的具体化工作

#### 反复修改

软件开发过程就是一个多次反复修改、逐步完善的过程。仅仅经过一次建模过程很难得到完全正确的对象模型。

### 建立动态模型

#### 概念

##### 适应性

###### 对于仅存储静态数据的系统来说，动态模型并没有什么意义

###### 在开发交互式系统时，动态模型却起着很重要的作用

###### 收集输入信息是系统的主要工作时，则在开发时建立正确的动态模型是至关重要的

##### 步骤

###### 编写典型交互行为的脚本

###### 从脚本中提取出事件，确定触发每个事件的动作对象以及接受事件的目标对象

###### 排列事件发生的次序，确定每个对象的状态及状态间的转换关系，用状态图描绘

###### 比较各个对象的状态图，确保事件之间的匹配

#### 编写脚本

##### 定义

1. 脚本是指系统在某一执行期间内出现的一系列事件。脚本描述用户与目标系统之间的一个或多个典型的交互过程。编写脚本的过程，就是分析用户对系统交互行为的要求的过程

##### 目的

1. 保证不遗漏重要的交互步骤，有助于确保交互过程的正确性、清晰性

##### 内容

1. 脚本描写的范围主要由编写脚本的具体目的决定，既可以包括系统中发生的全部事件，也可以只包括由某些特定对象触发的事件

#### 设想用户界面

##### 概念

1. 大多数交互行为都可以分为应用逻辑和用户界面两部分，通常，系统分析员首先集中精力考虑系统的信息流和控制流，而不是首先考虑用户界面

##### 目的

1. 重要的是在这种界面下的信息交换方式。目的是确保能够完成全部必要的信息交换，而不会丢失重要的信息

##### 方法

1. 快速地建立起用户界面的原型，供用户试用与评价

#### 画事件跟踪图

##### 步骤

###### 确定事件

###### 画出事件跟踪图，实质上事扩充的脚本

#### 画状态图

##### 定义

###### 状态图描绘事件与对象状态的关系。当对象接受了一个事件以后，它的下个状态取决于当前状态及所接受的事件。

###### 由事件引起的改变称为“转换”。

###### 一张状态图描绘了一类对象的行为，它确定了由事件序列引出的状态序列。

##### 方法(P250)

#### 审查动态模型

###### 检查系统级的完整性和一致性。

###### 审查每个事件，跟踪它对系统中各对象所产生的效果，保证与每个脚本都匹配。

### 建立功能模型

#### 定义

###### 功能模型表明了系统中数据之间的依赖关系，以及有关的数据处理功能，它由一组数据流图组成。在建立了对象模型和动态模型之后再建立功能模型

#### 画出基本系统模型图（由两部分组成)

###### 数据源点/终点：数据源点输入的数据和输出到数据终点的数据，是系统与外部世界间交互事件的参数

###### 处理框：代表了系统加工、变换数据的整体功能

#### 画出功能级数据流图

###### 把基本系统模型中单一的处理框分解成若干个处理框，以描述系统加工、变换数据的基本功能，就得到功能级数据流图

#### 描述处理框图

###### 要点

1. 着重描述每个处理框所代表的功能，而不是实现功能的具体算法。

### 定义服务

### 习题：

大型、复杂系统的对象模型通常由下述5个层次组成:主题层、类与对象层、结构层、属性层和（      服务层      ）

(     分析模型    )是系统分析员同用户及领域专家交流时有效的通信手段

两个或多个对象之间的相互依赖、相互作用的关系就是（      关联      ）

面向对象分析首要的工作,是建立问题域的（      对象模型      ）。

大多数交互行为都可以分为应用逻辑和用户界面两部分。通常,系统分析员首先集中精力考虑系统的(    信息流和控制流       )。

编写脚本的目的是保证不遗漏重要的交互步骤,有助于确保整个交互过程的\_\_\_**正确性**\_和 \_\_**清晰性**\_\_。

大型、复杂系统的对象模型通常由下述5个层次组成:\_**主题层\_**\_\_、\_\_**类与对象层**\_\_、\_**\_结构层\_\_**、\_\_**属性层**\_\_和\_\_**服务层**\_\_。

一般说来,可以使用\_\_\_**自底向上**\_、\_\_**自顶向下**\_\_两种方式建立继承(即泛化)关系。

筛选出正确的类与对象时,主要依据下列标准:\_\_\_**冗余**\_、\_**\_无关\_**\_、\_\_**笼统\_**\_、**属性**、**操作、实现**。

属性是对象的性质,藉助于属性人们能对类与对象和结构有更深入更具体的认识。一般说来,确定属性的过程包括\_\_**分析\_**\_和\_**\_选择**\_\_两个步骤。

进一步完善经筛选后余下的关联,通常从下述几个方面进行改进:\_\_**\_正名\_、\_分解\_\_\_、补充、表明重数**。

面向对象分析首要的工作,是建立问题域的\_**\_对象模型\_**\_。动态模型着重表示应用系统的 \_**\_控制逻辑**\_\_。

面向对象的分析方法是利用面向对象的信息建模概念,如实体、关系、属性等,同时运用\_\_**\_封装**\_、\_**\_继承**\_\_、多态等机制来构造模拟现实系统的方法。

## 面向对象设计

### 面向对**象**设计概念

#### 定义

###### 设计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。从面向对象分析到面向对象设计是一个逐渐扩充模型的过程，即面向对象设计就是用面向对象观点建立求解域模型的过程。

#### 分类

###### 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。

### 面向对象设计准则

#### 模块化

#### 抽象

#### 信息隐藏

#### 低耦合

##### 耦合：是指一个软件结构内不同模块之间互联的紧密程度

##### 对象之间耦合分为：

###### 交互耦合：对象间的耦合通过消息连接来实现，则这种耦合是交互耦合。

###### 继承耦合

#### 高内聚（面向对象设计存在的3种内聚）

###### 服务内聚

###### 类内聚

###### 一般一特殊内聚

#### 可重用

### 启发规则

#### 设计结果应该清晰易懂

#### 一般一特殊结构的深度适当

#### 设计简单的类

#### 使用简单的协议

#### 使用简单的服务

#### 把设计变动减至最小

### 软件重用

#### 概述

###### 重用也叫再用或复用，是指同一事物不作修改或稍加改动就多次重复使用；软件重用可分为3个层次：知识重用、方法和标准的重用、软件成分的重用

###### 软件成分重用的级别

1. 代码重用
2. 设计结果重用
3. 分析结果重用：“是一种更高级别的重用，重用某个系统的分析模型。特别适用于用户需求未改变，但系统体系结构发生了根本变化的场合。”

#### 类构件

##### 可重用软件构件特点

###### 模块独立性强。

###### 具有高度可塑性。

###### 接口清晰、简明、可靠。

##### 类构件的重用方式

###### 实例重用

###### 继承重用

###### 多态重用

#### 软件重用的效益

###### 质量

###### 生产率

###### 成本

### 系统分解

#### 分解思想

在设计比较复杂的应用系统时，先把系统分解成若干个较小部分，然后分别设计每个部分。

#### 子系统

##### 定义：

###### 系统的主要组成部分称为子系统，通常根据所提供的功能来划分子系统。

##### 划分原则：

###### 根据所提供的功能来划分子系统，子系统数目应该与系统规模基本匹配。

###### 各个子系统之间应该具有尽可能简单、明确的接口。

###### 应该尽量减少子系统彼此间的依赖性。

#### 分解面向对象设计模型

##### 表示

1. 面向对象设计模型由主题、类与对象、结构、属性、服务5个层次组成。这5个层次一层比一层表示的细节更多，可以把这5个层次想象为整个模型的水平切片
2. 面向对象设计模型在逻辑上都由4大部分组成，分别对应于组成目标系统的4个子系统,即问题域子系统、人机交互子系统、任务管理子系统和数据管理子系统

##### 子系统间交互方式

###### 客户-供应商关系（Client-supplier）

###### 平等伙伴关系

##### 组织系统的方案

###### 层次组织：“软件系统组织成一个层次系统，每层是一个子系统。”

1. 封闭式：每层子系统仅仅使用其直接下层提供的服务。降低了各层次之间的相互依赖性，更容易理解和修改。
2. 开放式：子系统可以使用处于其下面的任何一层子系统所提供的服务。
3. 优点是减少了需要在每层重新定义的服务数目，使系统更高效更紧凑。但其不符合信息隐藏原则。

###### 块状组织

###### 层次和块状的结合

###### 设计系统的拓扑结构

### 设计问题域子系统

#### 概念

###### 面向对象分析所得出的问题域精确模型，为设计问题域子系统建立了完整的框架

###### 保持面向对象分析所建立的问题域结构

###### 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

###### 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

#### 对问题域模型进行的处理

###### 调整需求

###### 重用已有的类

###### 把问题域类组合在一起

###### 增添一般化类以建立协议

###### 调整继承层次

### 设计人机交互子系统

#### 概念

##### 主要**内**容

###### 在面向对象设计过程中,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，其中包括指定窗口和报表的形式、设计命令层次等项内容。

#### 设计策略

##### 分类用户

##### 描述用户

##### 设计命令层次

1. 研究现有的人机交互含义和准则
2. 确定初始的命令层次
3. 精化命令的层次

##### 设计人机交互类

#### 设计步骤

##### 分析并发性

1. 并发性
2. 方法
3. 控制线

##### 设计任务管理子系统

1. 确定事件驱动型任务
2. 确定时钟驱动型任务
3. 确定优先任务
4. 确定关键任务
5. 确定协调任务
6. 尽量减少任务数
7. 确定系统资源需求

### 设计数据管理子系统

#### 概念

 数据库管理子系统：是系统存储或检索对象的基本设施，它建立在某种数据存储管理系统之上，并且隔离了数据存储管理模式的影响

#### 选择数据存储管理模式

##### 文件管理系统

###### 优点：文件管理系统是操作系统的一个组成部分，使用它长期保存数据具有成本低和简单的优点

###### 缺点：文件操作的级别低，为提供适当的抽象级别还必须编写额外的代码，不同操作系统的文件管理系统往往有明显差异

##### 关系数据库管理系统

##### 面向对象数据库管理系统

#### 设计数据管理子系统

##### 设计数据格式

##### 设计相应的服务

### 设计类中的服务

#### 确定类中应有的服务

##### 确定服务的总体思想

###### 对象模型是进行对象设计的基本框架。

###### 必须把动态模型中对象的行为以及功能模型中的数据处理转换成由适当的类所提供的服务。

###### 动态模型中状态图中的状态转换执行对象服务的结果。

###### 功能模型指明了系统必须提供的服务。

##### 确定操作目标对象的启发规则

##### 确定处理归属的启发规则

#### 设计实现服务的方法

##### 设计实现服务的算法

###### 应该考虑以下因素：

1. 算法复杂度：选用复杂度较低(效率较高)的算法，但不能过分追求高效率，应以能满足用户需求为准
2. 容易理解与容易实现：容易理解与容易实现的要求往往与高效率有矛盾，设计者应该对这两个因素适当折衷
3. 易修改：预测将来可能做的修改，并在设计时预先做些准备

##### 选择数据结构

##### 算法与数据结构的关系

##### 定义内部类和内部操作

### 设计关联

#### 关联

##### 定义：

在对象模型中，关联是联结不同对象的纽带，它指定了对象相互间的访问路径。

##### 确定实现关联的策略

选定一个全局性的策略统一实现所有关联。

#### 使用关联的方式

##### 关联的遍历

###### 单向遍历

###### 双向遍历

##### 实现单向遍历

###### 若关联的重数是一元的，则实现关联的指针是一个简单指针

###### 若关联的重数是多元的，则需要用一个指针集合实现关联

##### 实现双向关联

###### 只用属性实现一个方向的关联

###### 两个方向的关联都用属性实现

###### 用独立的关联对象实现双向关联

##### 关联对象的实现

###### 定义

1. 用一个关联类来保存描述关联性质的信息，关联中的每个连接对应关联类的一个对象

###### 方法

1. 对于一对一的关联，关联对象可以与参与关联的任一个对象合并
2. 对于一对多的关联，关联对象可以与多端对象合并
3. 对于多对多的关联，关联链的性质不可能只与一个参与关联的对象有关

### 设计优化

#### 确定优先级

#### 提高效率的几项技术

#### 调整**继**承关系

##### 继承关系

##### 建立类继承

###### 抽象与具体

###### 为提高继承程度而修改类定义

###### 利用委托实现行为共享

### 习题：

## 面向对象实现

### 面向对象实现概述

#### 主要任务

###### 把面向对象设计结果翻译成用某种程序语言书写的面向对象程序。

###### 测试并调试面向对象的程序。

#### 面向对象程序质量的影响因素

###### 面向对象设计的质量。

###### 采用的程序语言的特点。

###### 程序设计风格。

#### 保证软件可靠性的方法

###### 保证软件可靠性的主要措施是软件测试。

###### 面向对象测试的目标是用尽可能低的测试成本发现尽可能多的软件错误。

### 程序设计语言

#### 面向对象语言优点

##### 面向对象设计结果的表达方式

###### 面向对象语言：编译程序可以自动把面向对象概念映射到目标程序中

###### 非面向对象语言：必须由程序员自己把面向对象概念映射到目标程序中

##### 优点

###### **一致的表示方法**：面向对象开发基于不随时间变化的、一致的表示方法。既有利于在软件开发过程中始终使用统一的概念，也有利于维护人员理解软件的各种配置成分。

###### **可重用性**：既可重用面向对象分析结果，也可重用相应的面向对象设计和面向对象程序设计结果。

###### **可维护性**：程序显式地表达问题域语义，对维护人员理解待维护的软件有很大帮助。在选择编程语言时，应该考虑的首要因素是哪个语言能最恰当地表达问题域语义。

#### 面向对象语言技术特点

##### 支持类与对象概念的机制

###### 内容

1. 面向对象语言允许用户动态创建对象，并且可以用指针引用动态创建的对象。需要及时释放不再需要的对象所占用的内存

###### 管理内存的方法

1. 由语言的运行机制自动管理内存，即提供自动回收“垃圾”机制
2. 由程序员编写释放内存的代码

##### 实现聚集结构的机制

###### 使用指针

###### 使用独立的关联对象

##### 实现泛化结构的机制

###### 实现继承的机制

###### 解决名字冲突的机制，即处理在多个基类中可能出现的重名问题

##### 实现属性和服务的机制

###### 实现属性的机制：

1. 支持实例连接的机制
2. 属性的可见性控制
3. 对属性值的约束

###### 实现服务的机制：

1. 支持消息连接(表达对象交互关系)的机制
2. 控制服务可见性的机制
3. 动态联编(在发送消息前，无须知道接受消息的对象属于哪个类)

##### 类型检查

###### 分类

1. 弱类型：语言仅要求每个变量或属性隶属于一个对象。
2. 强类型：语法规定每个变量或属性必须准确地属于某个特定的类。

###### 强类型语言的优点

1. 有利于在编译时发现程序错误，提高软件的可靠性。
2. 增加了优化的可能性，提高软件的运行效率。

###### 适用性

1. 使用强类型编译型语言开发软件产品。
2. 使用弱类型解释型语言快速开发原型。

##### 类库

###### 存在类库，许多软构件就不必由程序员从头编写了，为实现软件重用带来很大方便。

##### 效率

##### 持久保存对象

###### 原因

1. 为实现在不同程序之间传递数据，需要保存数据。
2. 为恢复被中断了的程序的运行，首先需要保存数据。

###### 方法

1. 在类库中增加对象存储管理功能。
2. 使程序设计语言语法与对象存储管理语法无缝集成。

##### 参数化类

###### 定义：参数化类是使用一个或多个类型去参数化一个类的机制，如果程序语言提供一种能抽象出这类共性的机制，则对减少冗余和提高可重用性是大有好处的。

###### 方法

1. 定义一个参数化的类模板
2. 把数据类型作为参数传递进来

##### 开发环境

#### 选择面向对象语言标准

###### 将来能否占主导地位

###### 可重用性

###### 类库和开发环境

###### 其他因素

### 程序设计风格

#### 概念

###### 良好的程序设计风格的重要性。

###### 良好的面向对象程序设计风格内容。

#### 提高可重用性：

##### 代码重用

###### 内部重用：即本项目内的代码重用，主要是找出设计中相同或相似的部分，然后利用继承机制共享它们。

###### 外部重用：即新项目重用旧项目的代码，需要有长远眼光，反复考虑，精心设计。

##### 主要准则

###### 提高方法的内聚。

1. 一个方法应该只完成单个功能。

###### 减小方法的规模。

###### 保持方法的一致性。

###### 把策略与实现分开。

1. 策略方法：负责做出决策，提供变元，管理全局资源，称策略方法。
2. 实现方法：负责完成具体操作，称实现方法。
3. 实现方法相对独立，可在其它系统中重用，将二者分开。

###### 全面覆盖

###### 尽量不使用全局信息

1. 降低方法与外界耦合程度。

###### 利用继承机制

1. 调用子过程：把公共的代码分离出来，构成一个被其他方法调用的公用方法。可以在基类中定义这个公用方法，供派生类中的方法调用。
2. 分解因子：提高相似类代码可重用性的一个有效途径，是从不同类的相似方法中分解出不同的代码，把余下的代码作为公用方法中的公共代码，把分解出的因子作为名字相同算法不同的方法，放在不同类中定义，并被这个公用方法调用。
3. 使用委托：继承关系的存在意味着子类“即是”父类，因此，父类的所有方法和属性都应该适用于子类，仅当确实存在一般一特殊关系时，使用继承才是恰当的，当逻辑上不存在一-般一特殊关系时，为重用已有的代码，可以利用委托机制。
4. 把代码封装在类中：重用通过其他方法编写的、解决同一类应用问题的程序代码的一个比较安全的途径是把被重用的代码封装在类中。

##### 语言、开发环境和类库这3个因素综合起来,共同决定了可重用性。

#### 提高可扩充性

###### 封装实现策略：应该把类的实现策略封装起来，对外只提供公有的接口，否则将降低今后修改数据结构或算法的自由度。

###### 不要用一个方法遍历多条关联链：一个方法应该只包含对象模型中的有限内容。违反这条准则将导致方法过分复杂，既不易理解，也不易修改扩充。

###### 避免使用多分支语句：可以利用D\_CASE语句测试对象的内部状态，而不要用来根据对象类型选择应有的行为，否则在增添新类时将不得不修改原有的代码。

###### 精心确定公有方法。

#### 提高健壮性

1. 预防用户的错误操作
2. 检查参数的合法性
3. 不要预先确定限制条件
4. 先测试后优化

### 测试策略

#### 经典测试

1. 测试软件的经典策略是，从“小型测试”开始，逐步过渡到“大型测试”。

#### 面向对象的单元测试

1. 最小的可测试单元是封装起来的类和对象

#### 面向对象的集成测试

1. 基于线程的测试
2. 基于使用的测试

#### 面向对象的确认测试

### 设计测试用例

#### 测试类的方法

##### 随机测试

##### 划分测试

###### 目的：减少测试类时所需要的测试用例的数量

###### 方法：

1. 基于状态的划分：根据类操作改变类状态的能力来划分类操作
2. 基于属性的划分：根据类操作使用的属性来划分类操作
3. 基于功能的划分：根据类操作所完成的功能来划分类操作

#### 集成测试方法

##### 多类测试

###### 随机测试。

###### 划分测试。

##### 从动态模型导出测试用例

### 习题

为了能带来可观的商业利益,必须在更广泛的范围中运用（ 重用机制 ）。

属于面向对象设计的准则有：（模块化、信息隐藏、强内聚）。

所有面向对象语言都允许用户动态创建\_**对象\_**\_\_,并且可以用指针引用它。

一类方法负责做出决策,提供变元,并且管理全局资源,可称为\_\_**策略方法**\_\_。

\_**参数化类\_**\_\_,就是使用一个或多个类型去参数化一个类的机制。

\_**\_软件工具\_**\_和软件工程环境对软件生产率有很大影响。

存在类库,这为实现\_\_**软件重用**\_\_带来很大方便。

\_**\_健壮性**\_\_就是在硬件故障、输入的数据无效或操作错误等意外环境下,系统能做出适当响应的程度。

\_\_**数据管理子系统**\_\_是系统存储或检索对象的基本设施,它建立在某种数据存储管理系统之上,并且隔离了数据存储管理模式的影响。

在应用系统中,使用关联有两种可能的方式:\_**\_单向遍历**\_\_和 \_\_**双向遍历**\_\_。

设计实现服务的算法应该考虑\_\_**算法复杂度\_**\_、\_\_**容易理解与实现**\_\_、**\_易修改\_**\_\_三个因素。

\_\_**语言\_**\_、**\_开发环境\_\_**\_和 \_\_**类库**\_\_这3个因素综合起来,共同决定了可重用性。

所谓解决名字冲突,指的是处理在多个基类中可能出现的\_**\_重名**\_\_问题。