

理工类课程系列——

《软件工程》作业集

西北工业大学网络教育学院 组编

赵云庆 编

西北工业大学出版社

图书在版编目(CIP)数据

理工类课程系列——《软件工程》作业集/西北工业大学网络教育学院组编. —西安:西北工业大学出版社, 2005. 10

ISBN 7 - 5612 - 1935 - 0

. 理... . 西... . 理工类—高等教育—习题 . TP3 - 44

中国版本图书馆 CIP 数据核字(2005)第 040538 号

出版发行: 西北工业大学出版社

通信地址: 西安市友谊西路 127 号 邮编: 710072

电 话: 029 - 88493844 88491757

网 址: www.nwpup.com

印 刷 者: 陕西向阳印务有限公司

开 本: 787 mm × 1 092 mm 1/ 16

印 张: 5.25

字 数: 120 千字

版 次: 2005 年 10 月第 1 版 2005 年 10 月第 1 次印刷

定 价: 7.00 元

前 言

软件工程是一门实践性很强的学科。学好软件工程不仅要深入理解它的基本概念、原理、技术和方法，更重要的是通过实践学会用软件工程方法分析问题、解决问题。为了配合同学们的学习，我们特编写了这本作业集。作业集中，每道题都有详尽的解答，并附有两套模拟试题及参考答案。本作业集和教材《软件系统开发技术》（修订版）相配套，同时也可以作为其他软件工程教材的作业集使用。

本作业集由赵云庆编写，由西北工业大学网络教育学院组稿和审定。由于编者水平有限，不妥之处在所难免，诚请读者批评指正。

编 者

2005 年 4 月

编 委 会

主 任：魏生民

副 主 任：冷国伟 黄建森

编 委：邓修瑾 田 英 艾 兵

李 琳 李伟华 杨云霞

庞小宁 周 炯 殷俊杰

高宝营 黄 英 赵云庆

目 录

第一章	绪言.....	1
第二章	可行性研究与计划.....	5
第三章	需求分析和规格说明方法.....	7
第四章	设计方法	12
第五章	编程方法	15
第六章	检测和测试方法	19
第七章	维护方法	26
模拟考试题（一）	27
模拟考试题（二）	30
习题与模拟考试题参考答案	32

第一章 绪 言

本章重点与难点：

- (1) 了解软件概念、特点及分类方法。
- (2) 了解软件发展及软件危机的起因。
- (3) 了解软件工程过程及软件生存期的概念。
- (4) 了解软件工程的概念及其要素。
- (5) 了解软件工程的基本目标和原则。

一、多项选择题

1. 软件是计算机系统中与硬件相互依存的另一部分，它是包括 (A)、(B) 及 (C) 的完整集合。其中，(A) 是按事先设计的功能和性能要求执行的指令序列。(B) 是使程序能够正确操纵信息的数据结构。(C) 是与程序开发、维护和使用有关的图文材料。

A ~ C. 软件 程序 代码 硬件 文档 外设 数据 图表

2. 开发软件时对提高软件开发人员工作效率至关重要的一项是 (A)。软件工程中描述生存周期的瀑布模型一般包括计划、(B)、设计、编码、测试、维护等几个阶段，其中设计阶段在管理上又可以依次分成 (C) 和 (D) 两步。

供选择的答案：

A. 程序开发环境 操作系统的资源管理功能
 程序人员数量 计算机的并行处理能力

B. 需求分析 需求调查 可行性分析 问题定义

C, D. 方案设计 代码设计 概要设计 数据设计
 运行设计 详细设计 故障处理设计 软件体系结构设计

3. 从供选择的答案中选出适当字句填入下列关于软件发展过程的叙述中的 () 内。

有人将软件的发展过程划分为 4 个阶段：

第一阶段 (1950 ~ 1950 年代末) 称为“程序设计的原始时期”，这时既没有 (A)，也没有 (B)，程序员只能用机器指令编写程序。

第二阶段 (1950 年代末 ~ 1960 年代末) 称为“基本软件期”。出现了 (A)，并逐渐普及。随着 (B) 的发展，编译技术也有较大的发展。

第三阶段 (1960 年代末 ~ 1970 年代中期) 称为“程序设计方法时代”。这一时期，与硬件费用下降相反，软件开发费急剧上升。人们提出了 (C) 和 (D) 等程序设计方法，设法降低软件的开发费用。

第四阶段 (1970 年代中期 ~ 现在) 称为“软件工程时期”。软件开发技术不再仅仅是程序设计技术，而是包括了与软件开发的各个阶段，如 (E)、(F)、编码、单元测试、综合测试、(G) 及其整体有关的各种管理技术。

供选择的答案：

A ~ D:	汇编语言	操作系统	虚拟存储器概念	高级语言
	结构式程序设计	数据库概念	固件	模块化程序设计
E ~ G:	使用和维护	兼容性的确认	完整性的确认	设计
	需求定义	图像处理		

二、简答题

1. 软件工程过程有哪几个基本过程活动？试说明之。
2. 什么是软件危机？为什么会产生软件危机？
3. 试说明“软件生存周期”的概念。
4. 试论述瀑布模型软件开发方法的基本过程。
5. 软件工程是开发、运行、维护和修复软件的系统化方法，它包含哪些要素？试说明之。
6. 软件工程学的基本原则有哪些？试说明之。
7. 有人说：软件开发时，一个错误发现得越晚，为改正它所付出的代价就越大。对否？请解释你的回答。
8. 什么是软件的定义？

-
9. 软件是如何分类的？
 10. 软件有何特性？
 11. 简述软件的发展阶段？
 12. 什么是软件工程的定义？
 13. 软件工程的目的是什么？
 14. 试说明软件工程的基本原理。
 15. 简述软件工程的研究对象？
 16. 试说明软件工程过程和软件生命周期。

17. 解释软件生命周期模型。

18. 试说明软件开发方法。

19. 什么是软件的生命周期，理解软件的生命周期有什么意义？

20. 什么是线性顺序模型，有什么优缺点？

21. 与其他生命周期模型相比，螺旋模型有何特点？

第二章 可行性研究与计划

本章重点与难点：

- (1) 可行性研究的目的。
- (2) 可行性研究的内容。
- (3) 可行性研究的方法。
- (4) 系统结构的模型化。
- (5) 可行性研究报告的主要内容。

一、填空题

1. 计算机系统的元素可分为过程、_____、_____、_____、_____和文档。
2. 技术可行性研究要考虑的情况包括：_____风险；_____有效性；_____。

二、简答题

1. 系统分析过程，必须考虑哪八个方面的问题？
2. 可行性报告的内容主要包括哪几方面？
3. 系统定义的评审，其目何在？
4. 系统技术评审的评审内容包括哪些问题？
5. 系统管理评审的范围应包括哪些比较关键的问题？

6. 数据流图的作用是什么？它有哪些基本成分？

7. 可行性研究主要研究哪些问题？试说明之。

第三章 需求分析和规格说明方法

本章重点及难点：

- (1) 需求分析的方法。
- (2) 建立系统模型的方法。
- (3) 数据流图的应用。
- (4) 数据字典的实现。

一、填空题

系统分析的目标是识别用户要求、评价_____、进行_____分析和_____分析、把功能分配给_____、_____、_____、数据库和其他系统元素；建立成本和_____限制；生成系统_____说明。

二、单项选择题

1. 软件需求分析阶段的工作，可以分为以下 4 个方面：对问题的识别、分析与综合、编写需求分析文档以及（ ）。

总结 阶段性报告 需求分析评审 以上答案都不正确

2. 各种需求方法都有它们共同适用的（ ）。

说明方法 描述方式 准则 基本原则

3. 在结构化分析方法中，用以表达系统内数据的运动情况的工具有（ ）。

数据流图 数据词典 结构化英语 判定表与判定树

4. 在结构化分析方法中用状态 迁移图表达系统或对象的行为。在状态 迁移图中，由一个状态和一个事件所决定的下一状态可能会有（ ）个。

1 2 多个 不确定

5. 在结构化分析方法中用实体 关系图表达系统中的对象及其关系。在实体 关系图中，表达对象的实例之间的关联有三种类型：一对一联系、（ ）联系、多对多联系。

多对一 一对多 不存在 多种多样的

6. 软件需求分析的任务不应包括（ ）。

问题分析 信息域分析 结构化程序设计 确定逻辑模型

7. 进行需求分析可使用多种工具，但（ ）是不适用的。

数据流图 判定表 PAD 图 数据词典

8. 在需求分析中，分析员要从用户那里解决的最重要的问题是（ ）。

要让软件做什么 要给该软件提供哪些信息
要求软件工作效率如何 要让软件具有什么样的结构

9. 需求规格说明书的内容不应当包括（ ）。

对重要功能的描述 对算法的详细过程性描述

软件确认准则

软件的性能

10. 需求规格说明书文档在软件开发中具有重要的作用，但其作用不应当包括（ ）。

软件设计的依据

用户和开发人员对软件要“做什么”的共同理解

软件验收的依据

软件可行性分析的依据

11. 原型化方法是用户和软件开发人员之间进行的一种交互过程，适用于（ ）系统。

需求不确定性高的

需求确定的

管理信息

决策支持

12. 原型化方法从用户界面的开发入手，首先形成（ ）。

用户界面使用手册

用户界面需求分析说明书

系统界面原型

完善的用户界面

13. 原型化方法是用户和软件开发人员之间进行的一种交互过程，用户（ ），提出意见。

改进用户界面的设计

阅读文档资料

模拟用户界面的运行

运行用户界面原型

14. 原型化方法中用户就（ ）提出意见。

同意什么和不同意什么

使用和不使用哪一种编程语言

程序的结构

执行速度是否满足要求

15. 原型化方法是一种（ ）型的设计过程。

自外向内

自顶向下

自内向外

自底向上

三、分析题

1. 在软件需求分析时，首先建立当前系统的物理模型，再根据物理模型建立当前系统的逻辑模型。试问：什么是当前系统？当前系统的物理模型与逻辑模型有什么差别？

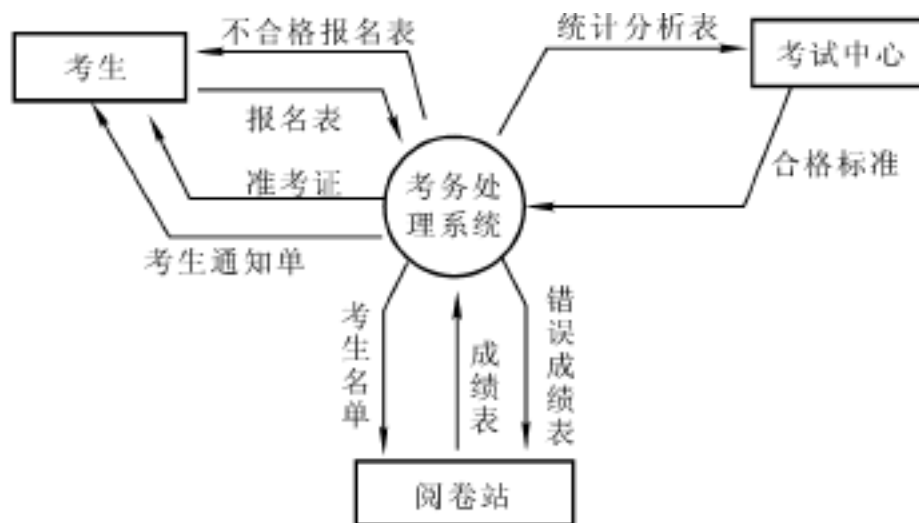
2. 软件需求分析是软件工程过程中交换意见最频繁的步骤。为什么交换意见的途径会经常阻塞？

3. 你认为一个系统分析员的理想训练和基础知识是什么？请说明理由。

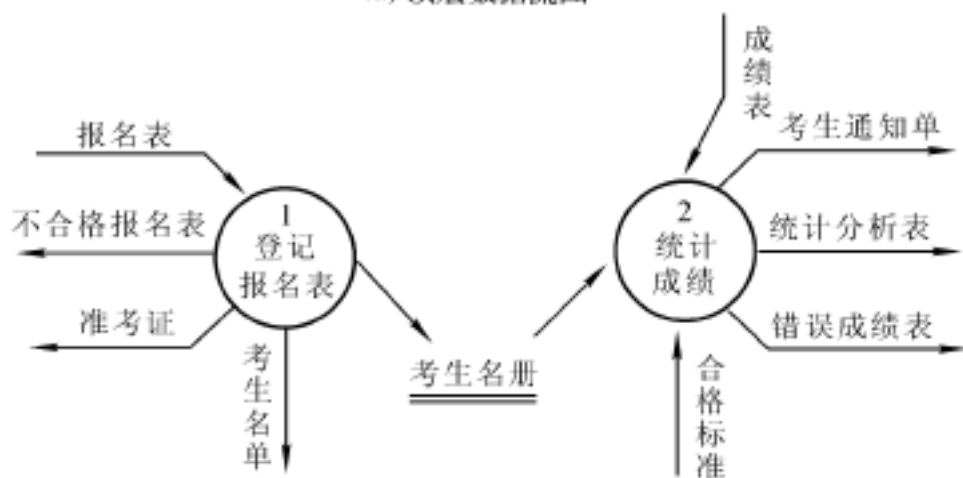
4. 信息和信息结构有什么区别？有没有不存在信息流的系统？有没有不存在信息结构的系统？

5. 软件需求分析的操作性原则和需求工程的指导性原则是什么？

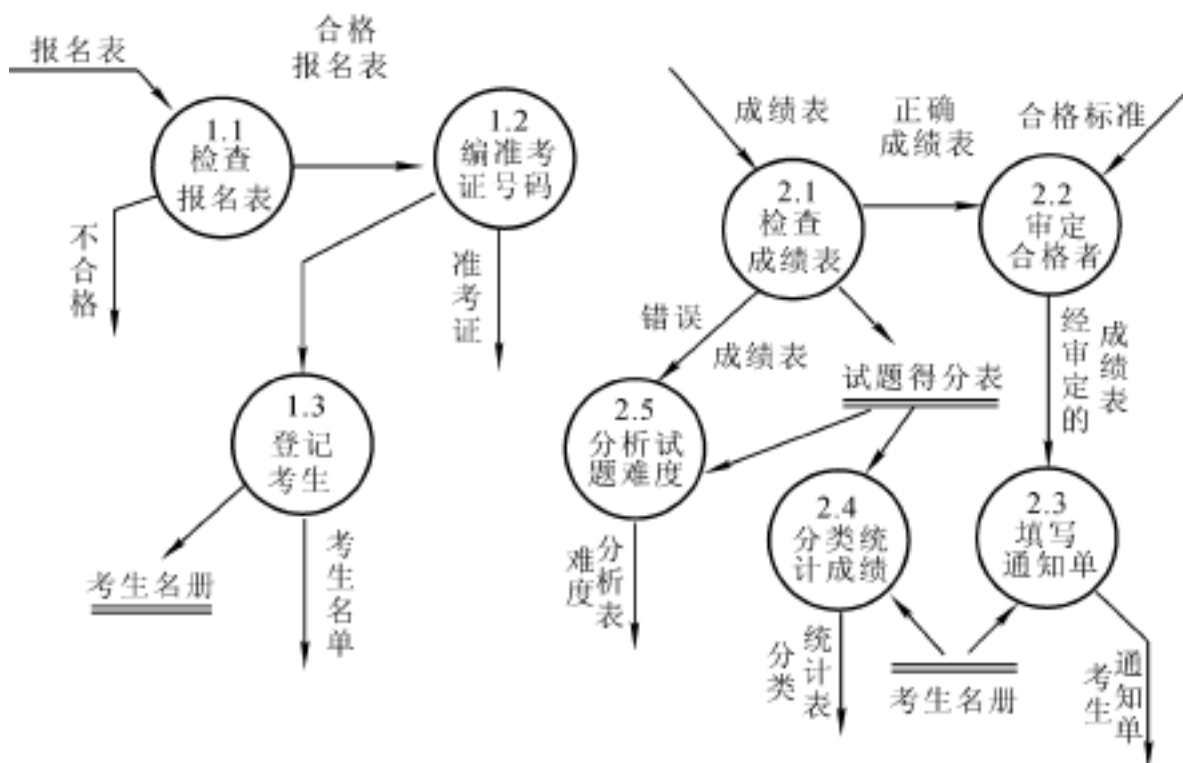
6. 考务处理系统的分层数据流图如下图所示。



(a) 顶层数据流图



(b) 第一层数据流图



(c) 加工 1 的子图

(d) 加工 2 的子图

该考务处理系统有如下功能：

对考生送来的报名表进行检查；

对合格的报名表编好准考证号码后将准考证送给考生，并将汇总后的考生名单送给阅卷站；

对阅卷站送来的成绩表进行检查，并根据考试中心指定的合格标准审定合格者；

填写考生通知单（内容包含考试成绩及合格 / 不合格标志），送给考生；

按地区、年龄、文化程度、职业、考试级别等进行成绩分类统计及试题难度分析，产生统计分析表。分别回答如下问题：

（1）图（c）中，加工 1.1 的输入数据流，输出数据流；图（b）中，加工 2 的输出数据流各是什么？

（2）图（b）中加工 2 的输出数据流是由什么组成的？

（3）图（d）中的文件“试题得分表”是否在图（b）中漏掉了？

7. 数据词典的作用是什么？它有哪些基本词条？

8. 传统的软件开发模型的缺陷是什么？原型化方法的类型有哪些？原型开发模型的主要优点是什么？

9. 试简述原型开发的过程和运用原型化方法的软件开发过程。

10. 软件需求分析说明书主要包括哪些内容？

11. 阅读下列关于开发人事管理系统的交互式工作方式的叙述，再回答问题。

某大企业最近决定采用高性能微机开发人事管理系统，将四台联机终端分置于人事处的三个科室。该系统可供操作员和程序员使用，也可供人事处负责人和主管人事的副厂长等查询人事信息用。人事管理系统通过录入人事数据和修改、删除等操作，产生和更新各类人事文件，通过搜索这些文件进行各类人事信息的查询。

该企业有 3 000 多个工人、干部和技术人员，大体可分成机关科室、生产车间、后勤服务和开发研制部门等几类部门。厂领导决定由计算机应用科来负责协调和开发应用系统。计

算机应用科科长指示系统工程师张某负责进行系统分析。

考虑到人事处有大量的查询信息要求、频繁的人事信息修改和文件存档、查阅等特点，计算机应用科决定认真设计人机交互界面，首先设计好在终端上的交互式会话的方式。

系统工程师张某通过调查收集到如下 10 条意见：

(1) 某程序员认为：系统在屏幕格式、编码等方面应具有一致性和清晰性，否则会影响操作人员的工作效率。

(2) 某操作人员认为：在交互式会话过程中，操作人员可能会忘记或记错某些事情，系统应当提供 HELP 功能。

(3) 某操作人员认为：既然是交互式会话，那么对所有的输入都应当作出响应，不应出现击键后，计算机没有任何反应的情况。

(4) 某操作人员认为：在出错的时候，交互式会话系统应当给出出错信息，并且尽可能告诉我们出错的性质和错在什么地方。

(5) 某程序员认为：终端会话也应当符合程序员编制程序时的习惯，这样可以更高效地维护人事管理系统。

(6) 教育科干部甲认为：应当对操作员进行一些必要的培训，让他们掌握交互式会话系统的设计技巧，有助于提高系统的使用效率。

(7) 教育科干部乙认为：尽管操作人员的指法已经强化训练但在交互式会话时应尽可能缩短和减少操作员输入的信息，以降低出错概率。

(8) 某程序员认为：由于本企业中有很多较大的文件，文件的查找很费时间，交互式会话系统在响应时间较长时应给予使用者以提示信息。

(9) 人事处干部丙认为：我们企业的人事资料相当复杂，格式非常之多，希望交互式系统使用十分清晰的格式，并容易对输入数据中的错误进行修改。

(10) 人事处干部丁认为：人事管理系统应当具有相当的保密性和数据安全性，因此在屏幕上显示出的信息应该含混一些，以免泄密。

系统工程师张某对上述调查情况和其他要求作了分析后，发现收集到的 10 条意见中有 3 条意见是不能接受的，写出编号并各用最多 40 个字叙述理由。

12. 逐步求精、分层过程与抽象等概念之间的相互关系如何？

13. 系统规格说明文档的主要内容包括哪些？

第四章 设计方法

本章重点与难点：

- (1) 软件设计的目标和任务。
- (2) 软件设计的过程。
- (3) 衡量设计的技术标准。
- (4) 软件设计的原则。
- (5) 软件设计的方法。

一、填空题

1. 一组语句在程序中多处出现，为了节省内存空间把这些语句放在一个模块中，该模块的内聚性是_____的。
2. 将几个逻辑上相似的成分放在同一个模块中，通过模块入口处的一个判断决定执行哪一个功能。该模块的内聚性是_____的。
3. 模块中所有成分引用共同的数据，该模块的内聚性是_____的。
4. 模块内的某成分的输出是另一些成分输入，该模块的内聚性是_____的。
5. 模块中所有成分结合起来完全一项任务，该模块的内聚性是_____的。它具有简明的外部界面，由它构成的软件易于理解、测试和维护。
6. 块间联系和块内联系是评价程序模块结构质量的重要标准。联系的方式、共用信息的作用、共用信息的数量和接口的_____等因素决定了块间联系的大小。
7. 在块内联系中，_____的块内联系最强。
8. SD 方法的总的原则是使每个模块执行_____功能。
9. SD 方法的总的原则是模块间传送_____参数。
10. SD 方法的总的原则是模块通过_____语句调用其他模块。
11. D 方法的总的原则是模块间传送的参数应尽量_____。
12. SD 方法还提出了判定的作用范围和模块的控制范围等概念。SD 方法认为，作用范围应该是_____的子集。
13. 在众多的设计方法中，SD 方法是最受人注意的，也是最广泛应用的一种，这种方法可以同分析阶段的 SA 方法及编程阶段的_____方法前后衔接。
14. SD 方法是考虑如何建立一个结构良好的程序结构，它提出了评价模块结构质量的两个具体标准——块间联系和块内联系。SD 方法的最终目标是_____。
15. SD 方法中用于表示模块间调用关系的图叫_____。
16. 软件详细设计工具可分为三类，即：图示工具、设计语言和表格工具。图示工具中，流程图简单而应用广泛、_____表示法中，每一个处理过程用一个盒子表示，盒子可以嵌套。
17. _____可以纵横延伸，图形的空间效果好。

18. _____是一种设计和描述程序的语言。

19. PDL 程序设计语言是一种面向_____的语言。

20. 在完成软件概要设计，并编写出相关文档之后，应当组织对概要设计工作的评审。评审的内容包括：分析该软件的系统结构、子系统结构，确认该软件设计是否覆盖了所有已确定的软件需求，软件每一成分是否可_____到某一项需求。

21. 在完成软件概要设计，并编写出相关文档之后，应当组织对概要设计工作的评审。评审的内容包括：分析软件各部分之间的联系，确认该软件的内部接口与外部接口是否已经明确定义。模块是否满足高内聚）和_____的要求。

22. 在完成软件概要设计，并编写出相关文档之后，应当组织对概要设计工作的评审。评审的内容包括：模块作用范围是否在其_____之内。

二、单项选择题

1. 从下列有关系统结构图的叙述中选出正确的叙述（ ）。

系统结构图中反映的是程序中数据流的情况。

系统结构图是精确表达程序结构的图形表示法。因此，有时也可将系统结构图当作程序流程图使用

一个模块的多个下属模块在系统结构图中所处的左右位置是无关紧要的

在系统结构图中，上级模块与其下属模块之间的调用关系用有向线段表示。这时，使用斜的线段和水平、垂直的线段具有相同的含义

2. 软件的开发工作经过需求分析阶段，进入（ ）。

程序设计 设计阶段 总体设计 定义阶段

3. 软件的开发工作经过需求分析阶段，以后就开始着手解决“怎么做”的问题。下面（ ）不属于常用的软件设计方法。

Jackson 方法 LCP (Wanier) 方法

SA 方法 SD 方法

4. 下述有关模块独立性的各种模块之间的耦合，耦合度最低的是（ ）。

内容耦合 控制耦合 非直接耦合 标记耦合

5. 下述有关模块独立性的各种模块之间的耦合，耦合度最高的是（ ）。

内容耦合 控制耦合 外部耦合 公共耦合

6. 下述有关模块独立性的各种模块内聚，内聚度（强度）最高的是（ ）。

巧合内聚 时间内聚 功能内聚 通信内聚

7. 下述有关模块独立性的各种模块内聚，内聚度（强度）最低的是（ ）。

巧合内聚 时间内聚 过程内聚 逻辑内聚

三、简答题

1. 完成良好的软件设计应遵循哪些原则？
2. 如何理解模块独立性？用什么指标来衡量模块独立性？
3. 模块独立性与信息隐蔽（反映模块化有效程度的属性）有何关系？
4. 模块的内聚性程度与该模块在分层结构中的位置有关系吗？说明你的论据。
5. 耦合性的概念和软件的可移植性有什么关系？请举例说明你的论述。
6. 递归模块（即自己调用自己的模块）的概念如何能够与本章所介绍的设计原理与方法相适应？
7. 举例说明你对概要设计与详细设计的理解。有不需概要设计的情况吗？
8. 如何用 PDL 语言来实施逐步求精的设计原理？

第五章 编程方法

本章重点与难点：

- (1) 编程的方法与技巧。
- (2) GOTO 语句的使用。

一、单项选择题

1. 允许用户建立、修改、存储正文的计算机程序是 ()。
Bootstrap Editor Loader Textformatter
2. 程序语言的编译系统和解释系统相比，从用户程序的运行效率来看 ()。
前者运行效率高 两者大致相同
后者运行效率高 不能确定
3. FORTRAN 语言的源程序是 () 结构。
块状 分程序嵌套
既是块状，又是嵌套 既不是块状，又不是嵌套的
4. 国际上最广泛使用的商用及行政管理语言是 ()。
COBOL BASIC FORTRAN PL/ 1
5. 国际上最流行的数值计算的程序设计语言是 ()。
BASIC ALGOL FORTRAN C
6. 美国国防部主持开发了高级程序设计语言 Ada，在它研制开始时，经反复比较，确定以高级语言 () 作为 Ada 研究的出发点。
LISP ALGOL ALGOL68 PL/ 1
7. 在人工智能领域，目前最广泛使用的高级语言是 ()。
Ada FORTRAN COBOL LISP
8. 汇编程序是指 ()。
用汇编语言写的程序 符号程序 汇编语言的处理程序
9. 为了实现递归子程序的正确调用，人们必须用 () 来保存有关信息。
堆栈 线性表 队列 树
10. 为了实现递归子程序的正确调用，人们必须用堆栈来保存 () 及有关信息。
返回地址 线性表 断点 入口点
11. UNIX 操作系统是 () 研制的。
Bell 实验室 DEC 公司 IBM 公司 微软
12. UNIX 操作系统是用程序语言 () 书写实现的。
C BASIC ALGOL68 PASCAL

二、多项选择题

1. 从下列关于模块化程序设计的叙述中选出 5 条正确的叙述。

程序设计比较方便，但比较难以维护。

便于由多个人分工编制大型程序。

软件的功能便于扩充。

程序易于理解，也便于排错。

在主存储器能够容纳得下的前提下，应使模块尽可能大，以便减少模块的个数。

模块之间的接口叫做数据文件。

只要模块之间的接口关系不变，各模块内部实现细节的修改将不会影响别的模块。

模块间的单向调用关系叫做模块的层次结构。

模块越小，模块化的优点越明显。一般来说，模块的大小都在 10 行以下。

2. 从下面关于程序编制的叙述中，选出三条正确的叙述。

在编制程序之前，首先必须仔细阅读给定的程序说明书。然后，必须如实地依照说明书编写程序。说明书中常会有含糊不清或难以理解的地方。程序员在作业时应该对这些地方作出适当的解释。

在着手编制程序时，重要的是采用既能使程序正确地按设计说明书进行处理，又易于排错的编写方法。

在编制程序时，首先应该对程序的结构充分考虑，不要急于开始编码，而要象写软件文档那样，很好地琢磨程序具有什么样的功能，这些功能如何安排等等。

考虑到以后的程序变更，为程序编写完整的说明书是一项很重要的工作。只要有了完整的程序说明书，即使程序的编写形式难以让他人看懂也没有什么关系。

编制程序时不可缺少的条件是，程序的输入和输出数据的格式都应确定。其他各项规定都是附带的，无足轻重。

作为一个好的程序，不仅处理速度要快，而且易读易修改等等也都是重要的条件。为了能得到这样的程序，不仅要熟悉程序设计语言的语法，还要注意采用适当的规程和单纯的表现方法，注意使整个程序的结构简洁。

3. 从下列叙述中选出 5 条符合程序设计风格指导原则的叙述。

嵌套的重数应加以限制。

尽量多使用临时变量。

不滥用语言特色。

不用可以省略的括号。

使用有意义的变量名。

应尽可能把程序编得短些。

把常见的局部优化工作留给编译程序去做。

注解越少越好。

程序的格式应有助于读者理解程序。

应尽可能多用 GOTO 语句。

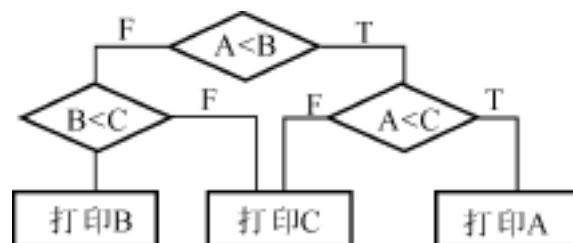
三、分析解答题

1. 下图是使用 Basic 语言编写的一个打印 A, B, C 三数中最小者的程序的流程图。其中出现了 6 个 GOTO 语句, 一个向前, 5 个向后, 程序可读性很差。

```

        if ( A < B ) goto 120;
        if ( B < C ) goto 110;
100    print C;
        goto 140;
110    print B;
        goto 140;
120    if ( A < C ) goto 130;
        goto 100;
130    print A;
140

```



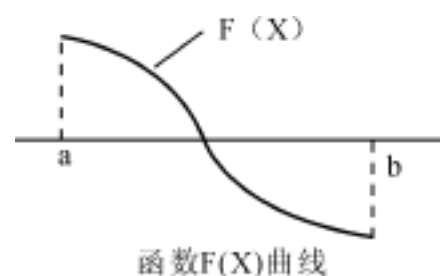
试利用基本控制结构, 将程序中的 GOTO 语句消去。

2. 设在闭区间 $[a, b]$ 上函数 $F(X)$ 有唯一的一个零点, 如下图所示。下面给出一个用 C 语言写出的程序段, 用二分法求方程 $F(X) = 0$ 在区间 $[a, b]$ 中的根。程序段中 X_0 , X_1 是当前求根区间 $[X_0, X_1]$ 的下上界, X_m 是该区间的中点, eps 是一个给定的很小正数, 用于迭代收敛的判断。在程序中采取了用 goto 语句和标号 finish 控制在循环中途转出循环。

```

【程序】  F0 = F(a); F1 = F(b);
           if ( F0 * F1 <= 0 ) {
X0 = a; X1 = b;
for ( i = 1; i <= n; i++ ) {
    Xm = (X0 + X1) / 2; Fm = F(Xm);
    if ( abs(Fm) < eps || abs(X1 - X0) < eps )
        goto finish;
    if ( F0 * Fm > 0 )
        { X0 = Xm; F0 = Fm; }
    else
        X1 = X;
}
finish: printf ( " \n The root of this equation is %d \n", Xm );}

```



这类循环结构出现了两个循环出口。一个是 for 循环的正常出口: 当循环控制变量 i 超

出了循环终值 n 时退出循环；另一个是 for 循环的非正常出口：当某种条件满足时，从循环中间某处转出循环，执行循环后面的语句。它不满足结构化的要求。

试利用结构化程序设计要求的几种基本控制结构，消除其中的 goto 语句，使得每一个部分都是单入口单出口。

3. 试说明下面的两个程序段的功能是什么？可否用另一些等效的程序段来代替它，以提高其可读性。

(1) $A[I] = A[I] + A[T];$

$A[T] = A[I] - A[T];$

$A[I] = A[I] - A[T];$

(2) for ($i = 1; i \leq n; i++$)

for ($j = 1; j \leq n; j++$)

$V[i][j] = (i/j) * (j/i);$

4. 结构化程序设计有时被错误地称为“无 GOTO 语句”的程序设计。请说明为什么会出现这样的说法，并讨论环绕着这个问题的一些争论。

5. 用某种软件复杂性度量算法来度量不同类型的程序时，得出的度量值是否真正反映了它们的复杂性？如果对同类型的程序进行度量，其结果是否就比较有价值？

6. 软件复杂性有哪几类？软件复杂性度量模型应遵循哪些基本原则？

第六章 检测和测试方法

本章重点与难点:

- (1) 软件测试的基本概念。
- (2) 软件测试策略。
- (3) 设计测试用例。

一、单项选择题

1. 软件测试的目的是 ()。
评价软件的质量
发现软件的错误
找出软件中的所有错误
证明软件是正确的
2. 为了提高测试的效率, 应该 ()。
随机地选取测试数据
取一切可能的输入数据作为测试数据
在完成编码以后制定软件的测试计划
选择发现错误的可能性大的数据作为测试数据
3. 使用白盒测试方法时, 确定测试数据应根据 () 和指定的覆盖标准。
程序的内部逻辑
程序的复杂程度
使用说明书
程序的功能
4. 与设计测试数据无关的文档是 ()。
该软件的设计人员
程序的复杂程度
源程序
项目开发计划
5. 软件的集成测试工作最好由 () 承担, 以提高集成测试的效果。
该软件的设计人员
该软件开发组的负责人
该软件的编程人员
不属于该软件开发组的软件设计人员
6. 程序的三种基本控制结构是 ()。
过程, 子程序, 分程序
顺序, 条件, 循环
递归, 堆栈, 队列
调用, 返回, 转移
7. 程序的三种基本控制结构的共同点是 ()。
不能嵌套使用
只能用来写简单的程序
已经用硬件实现
只有一个入口和一个出口
8. 结构化程序设计的一种基本方法是 ()。
筛选法
递归法
归纳法
逐步求精法
9. 软件测试的目的是 ()。
证明程序中没有错误
发现程序中的错误
测量程序的动态特性
检查程序中的语法错误

10. 软件调试的目的是 ()。

找出错误所在并改正之

排除存在错误的可能性

对错误性质进行分类

统计出错的次数

11. 对可靠性要求很高的软件, 例如操作系统, 由第三者对源代码进行逐行检查, 这属于 ()。

仿真器

代码审查

模拟器

黑盒测试

12. 已有的软件被改版时, 由于受到变更的影响, 改版前正常的功能可能发生异常, 性能也可能下降。因此, 对变更的软件进行测试是必要的。这属于 ()。

退化测试

白盒测试

域测试

黑盒测试

13. 在意识到被测试模块的内部结构或算法的情况下进行测试。这属于 ()。

退化测试

白盒测试

域测试

黑盒测试

14. 为了确认用户的需求, 先做出系统的主要部分, 提交给用户试用。这属于 ()。

仿真器

代码审查

模拟器

原型

15. 在测试具有层次结构的大型软件时, 有一种方法是从上层模块开始, 由上到下进行测试。此时, 有必要用一些 () 模块替代尚未测试过的下层模块。

仿真

驱动

模拟

桩

16. 软件测试方法可分为黑盒测试法和白盒测试法两种。黑盒测试法是通过分析程序的 () 来设计测试用例的方法。

应用范围

内部逻辑

功能

输入数据

17. 黑盒测试法除了测试程序外, 它还适用于对 () 阶段的软件文档进行测试。

编码

软件详细设计

软件总体设计

需求分析

18. 白盒测试法是根据程序的 () 来设计测试用例的方法。

应用范围

内部逻辑

功能

输入数据

19. 白盒测试法除了测试程序外, 它也适用于对 () 阶段的软件文档进行测试。

编码

软件详细设计

软件总体设计

需求分析

20. 白盒法测试程序时常按照给定的覆盖条件选取测试用例。判定覆盖比 () 覆盖严格, 它使得每一个判定的每一条分支至少经历一次。

语句

判定

条件

判定/ 条件

21. 白盒法测试中 () 覆盖既是判定覆盖, 又是条件覆盖, 但它并不保证使各种条件都能取到所有可能的值。

语句

判定

条件

判定/ 条件

22. 白盒法测试中 () 覆盖比其他条件都要严格, 但它不能保证覆盖程序中的每一条路径。

语句

判定

条件

多重条件

23. 单元测试一般以 () 为主。

白盒法

黑盒法

条件

多重条件

24. 单元测试的依据是 ()。

模块功能规格说明

系统模块结构图

系统需求规格说明

多重条件

25. 软件测试中常用的静态分析方法是 () 和接口分析。

引用分析 算法分析 可靠性分析 效率分析

26. 软件测试中常用的静态分析方法 () 用于检查模块或子程序间的调用是否正确。

效率分析 算法分析 可靠性分析 接口分析

27. 软件测试中分析方法 (白盒方法) 中常用的方法是 () 方法。

路径测试 等价类 因果图 归纳测试

28. 软件测试中非分析方法 (黑盒方法) 中常用的方法是等价类方法和 () 方法是根
据输出对输入的依赖关系设计测试用例。

路径测试 等价类 因果图 相对图

二、多项选择题

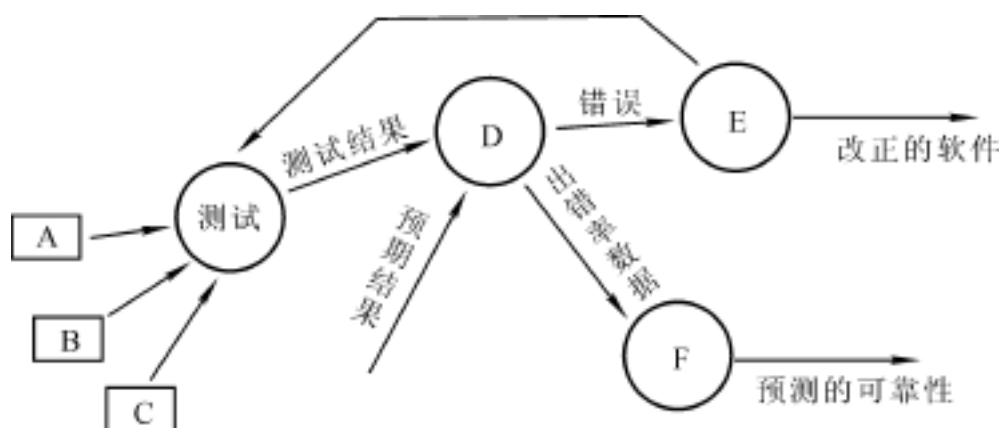
1. 为了把握软件开发各个环节的正确性和协调性, 人们需要进行 (A) 和 (B) 工作。
(A) 的目的是想证实在一给定的外部环境中软件的逻辑正确性。它包括 (C) 和 (D), (B)
则试图证明在软件生存期各个阶段, 以及阶段间的逻辑 (E)、(F) 和正确性。

A, B. 操作 确认 验证 测试 调试

C, D. 用户的确认 需求规格说明的确认 程序的确认 测试的确认

E, F. 可靠性 独立性 协调性 完备性 扩充性

2. 测试过程需要三类输入: (A)、(B) 和 (C)。请选择正确的答案填入下图中以完成
测试信息处理的全过程。



A ~ C. 接口选择 软件配置 硬件配置 测试配置
 测试环境 测试工具

D ~ F. 排错 可靠性分析 结果分析 数据分类

3. 软件测试是软件质量保证的主要手段之一, 测试的费用已超过 (A) 的 30% 以上。
因此, 提高测试的有效性十分重要。“高产”的测试是指 (B)。根据国家标准 GB 8566 - 88
《计算机软件开发规范》的规定, 软件的开发和维护划分为 8 个阶段, 其中, 单元测试是在
(C) 阶段完成的, 集成测试的计划是在 (D) 阶段制定的, 确认测试的计划是在 (E) 阶段
制定的。

A. 软件开发费用 软件维护费用 软件开发和维护费用
 软件研制费用 软件生存期全部

B. 用适量的测试用例运行程序, 证明被测程序正确无误
 用适量的测试用例运行程序, 证明被测程序符合相应的要求

用少量的测试用例运行程序，发现被测程序尽可能多的错误

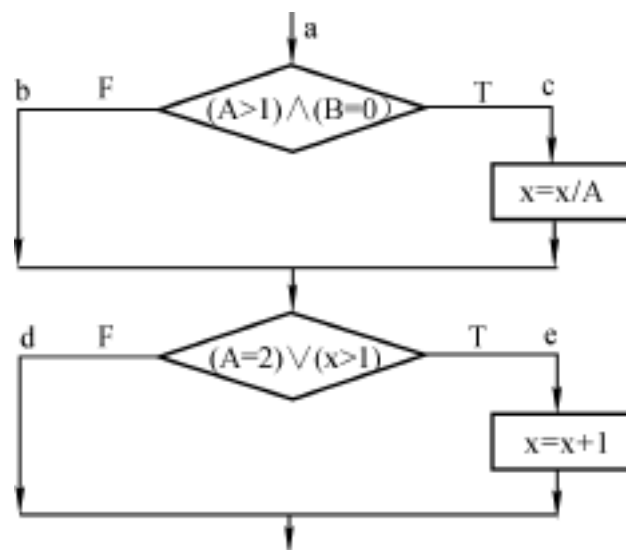
用少量的测试用例运行程序，纠正被测程序尽可能多的错误

C ~ E.	可行性研究和计划	需求分析	概要设计	详细设计
	实现	集成测试	确认测试	使用和维护

4. 集成测试也叫做 (A) 或 (B)。通常，在 (C) 的基础上，将所有模块按照设计要求组装成为系统。子系统的集成测试特别称为 (D)，它所做的工作是要找出子系统和系统需求规格说明之间的 (E)。需要考虑的问题是：在把各个模块连接起来的时候，穿越模块接口的数据是否会 (F)；一个模块的功能是否会对另一个模块的功能产生不利的影响；各个 (G) 组合起来，能否达到预期要求的 (H)；(I) 是否有问题；单个模块的误差累积起来是否会放大。

A ~ D.	单元测试	部件测试	组装测试	
	系统测试	确认测试	联合测试	
E ~ I.	子功能	丢失	父功能	局部数据结构
	全局数据结构		不一致	一致

5. 如图所示的程序有四条不同的路径。分别表示为 L_1 (a c e), L_2 (a b d), L_3 (a b e) 和 L_4 (a c d), 或简写为 ace, abd, abe 及 acd。由于覆盖测试的目标不同，逻辑覆盖方法可以分为语句覆盖、判定覆盖、条件覆盖、判定 - 条件覆盖、条件组合覆盖和路径覆盖。



从备选的答案中选择适当的测试用例与之匹配。(A) 属于语句覆盖；(B)、(C) 属于判定覆盖；(D)、(E) 属于条件覆盖；(F)、(G) 属于判定 - 条件覆盖；(H) 属于条件组合覆盖；(I) 属于路径覆盖。

A . ~ I. 【(2, 0, 4), (2, 0, 3)】覆盖 ace; 【(1, 1, 1), (1, 1, 1)】覆盖 abd;

判断条件覆盖

【(1, 0, 3), (1, 0, 4)】覆盖 abe;

【(2, 1, 1), (2, 1, 2)】覆盖 abe;

【(2, 0, 4), (2, 0, 3)】覆盖 ace;

【(2, 1, 1), (2, 1, 2)】覆盖 abe; 【(3, 0, 3), (3, 1, 1)】覆盖 acd;

【(2,0,4),(2,0,3)】覆盖 ace; 【(1,0,1),(1,0,1)】覆盖 abd;
 【(2,1,1),(2,1,2)】覆盖 abe;
 【(2,0,4),(2,0,3)】覆盖 ace; 【(1,1,1),(1,1,1)】覆盖 abd;
 【(1,1,2),(1,1,3)】覆盖 abe; 【(3,0,3),(3,0,1)】覆盖 acd;
 【(2,0,4),(2,0,3)】覆盖 ace; 【(1,1,1),(1,1,1)】覆盖 abd;
 【(1,0,3),(1,0,4)】覆盖 abe; 【(2,1,1),(2,1,2)】覆盖 abe;

三、分析题

1. 对小的程序进行穷举测试是可能的，用穷举测试能否保证程序是百分之百正确呢？

2. 在任何情况下单元测试都是可能的吗？都是需要的吗？

3. 下面是选择排序的程序，其中 datalist 是数据表，它有两个数据成员：一是元素类型为 Element 的数组 V，另一个是数组大小 n。算法中用到两个操作，一是取某数组元素 V[i] 的关键码操作 getKey()，一是交换两数组元素内容的操作 Swap()::

```

void SelectSort ( datalist & list ) {
// 对表 list.V [0] 到 list.V [n - 1] 进行排序，n 是表当前长度。
    for ( int i = 0; i < list.n - 1; i++ ) {
        int k = i;      // 在 list.V [i] .key 到 list.V [n - 1] .key 中找具有最小
        关键码的对象
        for ( int j = i + 1; j < list.n; j++ )
            if ( list.V [j] .getKey ( ) < list.V [k] .getKey ( ) ) k = j;
            // 当前具最小关键码的对象
        if ( k != i ) Swap ( list.V [i], list.V [k] );      // 交换
    }
}
  
```

- (1) 试计算此程序段的 McCabe 复杂性;
- (2) 用基本路径覆盖法给出测试路径;
- (3) 为各测试路径设计测试用例。

4. 根据下面给出的规格说明，利用等价类划分的方法，给出足够的测试用例。

“一个程序读入三个整数。把此三个数值看成是一个三角形的三个边。这个程序要打印出信息，说明这个三角形是三边不等的、是等腰的、还是等边的。”

5. 设要对一个自动饮料售货机软件进行黑盒测试。该软件的规格说明如下：

“有一个处理单价为 1 元 5 角钱的盒装饮料的自动售货机软件。若投入 1 元 5 角硬币，按下“可乐”、“雪碧”或“红茶”按钮，相应的饮料就送出来。若投入的是 2 元硬币，在送出饮料的同时退还 5 角硬币。”

(1) 试利用因果图法，建立该软件的因果图；

(2) 设计测试该软件的全部测试用例。

6. 应该由谁来进行确认测试？是软件开发者还是软件用户？为什么？

7. 下面是快速排序算法中的一趟划分算法，其中 `datalist` 是数据表，它有两个数据成员：一是元素类型为 `Element` 的数组 `V`，另一个是数组大小 `n`。算法中用到两个操作，一是取某数组元素 `V[i]` 的关键码操作 `getKey()`，一是交换两数组元素内容的操作 `Swap()`：

```
int Partition ( datalist &list, int low, int high ) {
// 在区间 [ low, high ] 以第一个对象为基准进行一次划分，k 返回基准对象回放位置。
int k = low; Element pivot = list.V [low];           // 基准对象
    for ( int i = low + 1; i <= high; i++ )           // 检测整个序列，进行划分
        if ( list.V [i] .getKey () < pivot.getKey () && ++ k != i )
            Swap ( list.V [k], list.V [i] );           // 小于基准的交换到左侧去
        Swap ( list.V [low], list.V [k] );           // 将基准对象就位
    return k;                                           // 返回基准对象位置
}
```

(1) 试画出它的程序流程图；

(2) 试利用路径覆盖方法为它设计足够的测试用例（循环次数限定为 0 次，1 次和 2 次）。

8. 从下列关于软件测试的叙述中，选出 5 条正确的叙述。

- (1) 用黑盒法测试时，测试用例是根据程序内部逻辑设计的。
- (2) 尽量用公共过程或子程序去代替重复的代码段。
- (3) 测试是为了验证该软件已正确地实现了用户的要求。
- (4) 对于连锁型分支结构，若有 n 个判定语句，则有 2^n 条路径。
- (5) 尽量采用复合的条件测试，以避免嵌套的分支结构。
- (6) GOTO 语句概念简单，使用方便，在某些情况下，保留 GOTO 语句反能使写出的程序更加简洁。
- (7) 发现错误多的程序模块，残留在模块中的错误也多。
- (8) 黑盒测试方法中最有效的是因果图法。
- (9) 在做程序的单元测试时，桩（存根）模块比驱动模块容易编写。
- (10) 程序效率的提高主要应通过选择高效的算法来实现。

第七章 维护方法

本章重点与难点：

- (1) 软件维护的重要性。
- (2) 软件维护的步骤、方法。
- (3) 软件维护的管理。

简答题

1. 维护工作分哪几类？

2. 某些软件工程师不同意“目前国外许多软件开发组织把 60% 以上的人力用于维护已有的软件”的说法，他们争论说：“我并没有花费我的 60% 的时间去改正我所开发的程序中的错误”。请问，你对上述争论有何看法？

3. 为什么大型软件的维护成本高达开发成本的 4 倍左右？

模拟考试题（一）

一、填空题

1. 计算机系统的元素可分为过程、_____、_____、_____、_____和文档。
2. 一组语句在程序中多处出现，为了节省内存空间把这些语句放在一个模块中，该模块的内聚性是_____的。
3. 将几个逻辑上相似的成分放在同一个模块中，通过模块入口处的一个判断决定执行哪一个功能。该模块的内聚性是_____的。
4. 模块中所有成分引用共同的数据，该模块的内聚性是_____的。
5. 模块内的某成分的输出是另一些成分的输入，该模块的内聚性是_____的。
6. 模块中所有成分结合起来完成一项任务，该模块的内聚性是_____的。它具有简明的外部界面，由它构成的软件易于理解、测试和维护。
7. 块间联系和块内联系是评价程序模块结构质量的重要标准。联系的方式、共用信息的作用、共用信息的数量和接口的_____等因素决定了块间联系的大小。

二、单项选择题

1. 软件需求分析阶段的工作，可以分为以下 4 个方面：对问题的识别、分析与综合、编写需求分析文档以及（ ）。
总结 阶段性报告 需求分析评审 以上答案都不正确
2. 各种需求方法都有它们共同适用的（ ）。
说明方法 描述方式 准则 基本原则
3. 在结构化分析方法中，用以表达系统内数据的运动情况的工具有（ ）。
数据流图 数据词典 结构化英语 判定表与判定树
4. 在结构化分析方法中用状态 迁移图表达系统或对象的行为。在状态 迁移图中，由一个状态和一个事件所决定的下一状态可能会有（ ）个。
1 2 多个 不确定
5. 在结构化分析方法中用实体 关系图表达系统中的对象及其关系。在实体 关系图中，表达对象的实例之间的关联有三种类型：一对一联系、（ ）联系、多对多联系。
多对一 一对多 不存 多种多样的
6. 从下列有关系统结构图的叙述中选出正确的叙述（ ）。
系统结构图中反映的是程序中数据流的情况
系统结构图是精确表达程序结构的图形表示法。因此，有时也可将系统结构当作

程序流程图使用

一个模块的多个下属模块在系统结构图中所处的左右位置是无关紧要的

在系统结构图中，上级模块与其下属模块之间的调用关系用有向线段表示。这时，使用斜的线段和水平、垂直的线段具有相同的含义

7. 软件的开发工作经过需求分析阶段，进入（ ）。

程序设计 设计阶段 总体设计 定义阶段

8. 软件的开发工作经过需求分析阶段，以后就开始着手解决“怎么做”的问题。下面（ ）不属于常用的软件设计方法。

Jackson 方法 LCP (Wanier) 方法
SA 方法 SD 方法

三、多项选择题

从供选择的答案中选出适当字句填入下列关于软件发展过程的叙述中的（ ）内。

有人将软件的发展过程划分为 4 个阶段：

第一阶段（1950 ~ 1950 年代末）称为“程序设计的原始时期”，这时既没有（A），也没有（B），程序员只能用机器指令编写程序。

第二阶段（1950 年代末 ~ 1960 年代末）称为“基本软件期”。出现了（A），并逐渐普及。随着（B）的发展，编译技术也有较大的发展。

第三阶段（1960 年代末 ~ 1970 年代中期）称为“程序设计方法时代”。这一时期，与硬件费用下降相反，软件开发费急剧上升。人们提出了（C）和（D）等程序设计方法，设法降低软件的开发费用。

第四阶段（1970 年代中期 ~ 现在）称为“软件工程时期”。软件开发技术不再仅仅是程序设计技术，而是包括了与软件开发的各个阶段，如（E）、（F）、编码、单元测试、综合测试、（G）及其整体有关的各种管理技术。

供选择的答案：

A ~ D:	汇编语言	操作系统	虚拟存储器概念	高级语言
	结构式程序设计	数据库概念	固件	模块化程序设计
E ~ G:	使用和维护	兼容性的确认	完整性的确认	设计
	需求定义	图象处理		

四、分析题

1. 什么是软件危机？为什么会产生软件危机？

2. 有人说：软件开发时，一个错误发现得越晚，为改正它所付出的代价就越大。对否？请解释你的回答。

3. 在软件需求分析时，首先建立当前系统的物理模型，再根据物理模型建立当前系统的逻辑模型。试问：什么是当前系统？当前系统的物理模型与逻辑模型有什么差别？

4. 软件需求分析是软件工程过程中交换意见最频繁的步骤。为什么交换意见的途径会经常阻塞？

5. 你认为一个系统分析员的理想训练和基础知识是什么？请说明理由。

6. 如何理解模块独立性？用什么指标来衡量模块独立性？

模拟考试题（二）

一、填空题

1. 技术可行性研究要考虑的情况包括：_____风险；_____有效性。
2. 在块内联系中，_____的块内联系最强。
3. SD 方法的总的原则是使每个模块执行_____功能。
4. SD 方法的总的原则是模块间传送_____参数。
5. SD 方法的总的原则是模块通过_____语句调用其他模块。
6. D 方法的总的原则是模块间传送的参数应尽量_____。
7. SD 方法还提出了判定的作用范围和模块的控制范围等概念。SD 方法认为，作用范围应该是_____的子集。

二、单项选择题

1. 软件需求分析的任务不应包括（ ）。
问题分析 信息域分析 结构化程序设计 确定逻辑模型
2. 进行需求分析可使用多种工具，但（ ）是不适用的。
数据流图 判定表 PAD 图 数据词典
3. 在需求分析中，分析员要从用户那里解决的最重要的问题是（ ）。
要让软件做什么 要给该软件提供哪些信息
要求软件工作效率如何 要让软件具有什么样的结构
4. 需求规格说明书的内容不应当包括（ ）。
对重要功能的描述 对算法的详细过程性描述
软件确认准则 软件的性能
5. 需求规格说明书文档在软件开发中具有重要的作用，但其作用不应当包括（ ）。
软件设计的依据
用户和开发人员对软件要“做什么”的共同理解
软件验收的依据 软件可行性分析的依据
6. 下述有关模块独立性的各种模块之间的耦合，耦合度最低的是（ ）。
内容耦合 控制耦合 非直接耦合 标记耦合
7. 下述有关模块独立性的各种模块之间的耦合，耦合度最高的是（ ）。
内容耦合 控制耦合 外部耦合 公共耦合
8. 下述有关模块独立性的各种模块内聚，内聚度（强度）最高的是（ ）。
巧合内聚 时间内聚 功能内聚 通信内聚
9. 下述有关模块独立性的各种模块内聚，内聚度（强度）最低的是（ ）。
巧合内聚 时间内聚 过程内聚 逻辑内聚

三、多项选择题

开发软件时对提高软件开发人员工作效率至关重要的一项是 (A)。软件工程中描述生存周期的瀑布模型一般包括计划、(B)、设计、编码、测试、维护等几个阶段,其中设计阶段在管理上又可以依次分成 (C) 和 (D) 两步。

- | | | | |
|-------|--------|------|-------------|
| A. | 程序开发环境 | | 操作系统的资源管理功能 |
| | 程序人员数量 | | 计算机的并行处理能力 |
| B. | 需求分析 | 需求调查 | 可行性分析 |
| | | | 问题定义 |
| C, D. | 方案设计 | 代码设计 | 概要设计 |
| | 运行设计 | 详细设计 | 数据设计 |
| | | | 故障处理设计 |
| | | | 软件体系结构设计 |

四、分析题

1. 试说明“软件生存周期”的概念。
2. 软件工程学的基本原则有哪些?试说明之。
3. 信息和信息结构有什么区别?有没有不存在信息流的系统?有没有不存在信息结构的系统?
4. 软件需求分析的操作性原则和需求工程的指导性原则是什么?
5. 模块独立性与信息隐蔽(反映模块化有效程度的属性)有何关系?
6. 举例说明你对概要设计与详细设计的理解。有不需概要设计的情况吗?

习题与模拟试题参考答案

第一章

一、多项选择题

1. A. B. C.

分析：软件是计算机系统中与硬件相互依存的另一部分，它是包括程序、数据及相关文档的完整集合。其中，程序是按事先设计的功能和性能要求执行的指令序列。数据是使程序能够正确操纵信息的数据结构。文档是与程序开发、维护和使用有关的图文材料。需要注意的是，程序与算法在含义上有不同：算法的每一条指令必须是最基本的、必须通过有限步做完，而程序没有这个要求。

2. A. B. C. D.

分析：有合适的程序开发环境可以提供有用的工具，大大提高开发人员的工作效率。

软件工程中描述软件生存周期的瀑布模型一般包括计划、需求分析、设计、编码、测试和运行维护等六个阶段。需求分析完成对软件产品在功能、性能、用户接口、运行环境、可靠性、安全性、开发资源、开发进度、开发成本等方面的需求定义。问题定义、可行性分析、需求调查都可能是需求分析中要做的一部分工作。

软件设计在管理上划分为概要设计与详细设计两个步骤。概要设计的目标是建立软件的体系结构，完成全局数据结构设计，同时进行处理方式设计、运行配置设计、出错处理设计、故障恢复设计等。详细设计是对每一个模块的操作的控制流程和局部数据结构进行设计。

3. A. B. C. D. E. F. G.

注意，C与D的答案顺序可互换。

分析：在软件发展的早期，汇编语言和高级语言尚未出现，人们只能用机器指令来编写程序。为了好读，有的指令系统可以用八进制代码书写。其后，由于汇编语言的出现，人们摆脱了繁重的地址分配等工作，可以用符号编程。随着高级语言的出现和普及，人们用近似于自然语言的语句编写程序，大大减轻了程序员的负担。高级语言的处理，从解释执行到编译执行，处理效率和存储利用率不断提高。

结构化程序设计和模块化程序设计是科学家为解决软件危机，借用其他领域的技术改进程序设计方法而提出来的。由于这些技术的使用，提高了程序的可读性、局部性、抽象性、清晰性、简单性、确定性、一致性等，降低了程序开发的费用。后来发展到软件工程阶段，明确地划分了软件开发阶段，规范了软件开发过程，明确了各个阶段的任务以及应交付的成果和里程碑，使得软件开发逐步达到工程化和标准化。

二、简答题

1. 答：软件工程过程的基本过程活动有 4 步：

(1) P (Plan)：软件规格说明。规定软件的功能及其运行的限制。

(2) D (Do)：软件开发。产生满足规格说明的软件。

(3) C (Check)：软件确认。确认软件能够完成客户提出的要求。

(4) A (Action)：软件演进。为满足客户的变更要求，软件必须在使用的过程中演进。

2. 答：软件危机泛指在计算机软件的开发、维护和使用过程中所遇到的一系列严重问题。

从宏观上说，软件危机主要是指：软件的发展赶不上计算机硬件的发展；软件的发展赶不上社会对于软件需求的增长。

从具体的软件来说，软件危机是指：软件往往不能按计划、按预算、按时完成；已开发的软件不能很好地使用，甚至很快就不用。

产生软件危机的主要原因包括：

(1) 软件需求分析不充分；

(2) 软件开发的规范性不够；

(3) 软件开发计划的科学性不够；

(4) 缺少对于软件的评测手段。

3. 答：软件与任何一个事物一样，有它的孕育、诞生、成长、成熟、衰亡的生存过程。这就是软件的生存周期。它主要分为 6 个阶段：软件项目计划、软件需求分析和定义、软件设计、程序编码、软件测试，以及运行维护。

(1) 软件项目计划：在这一步要确定软件工作范围，进行软件风险分析，预计软件开发所需要的资源，建立成本与进度的估算。根据有关成本与进度的限制分析项目的可行性。

(2) 软件需求分析和定义：在这一步详细定义分配给软件的系统元素。可以用以下两种方式中的一种对需求进行分析和定义。一种是正式的信息域分析，可用于建立信息流和信息结构的模型，然后逐渐扩充这些模型成为软件的规格说明。另一种是软件原型化方法，即建立软件原型，并由用户进行评价，从而确定软件需求。

(3) 软件设计：软件的设计过程分两步走。第一步进行概要设计，以结构设计和数据设计开始，建立程序的模块结构，定义接口并建立数据结构。此外，要使用一些设计准则来判断软件的质量。第二步做详细设计，考虑设计每一个模块部件的过程描述。经过评审后，把每一个加细的过程性描述加到设计规格说明中去。

(4) 程序编码：在设计完成之后，用一种适当的程序设计语言或 CASE 工具生成源程序。应当就风格及清晰性对代码进行评审，而且反过来应能直接追溯到详细设计描述。

(5) 软件测试：单元测试检查每一单独的模块部件的功能和性能。组装测试提供了构造软件模块结构的手段，同时测试其功能和接口。确认测试检查所有的需求是否都得到满足。在每一个测试步骤之后，要进行调试，以诊断和纠正软件的故障。

(6) 软件维护：为改正错误，适应环境变化及功能增强而进行的一系列修改活动。与软件维护相关联的那些任务依赖于所要实施的维护的类型。

4. 答：瀑布模型规定了各项软件工程活动，包括：制定软件项目计划，进行需求分析

和定义，软件设计，程序编码，测试及运行维护。并且规定了它们自上而下，相互衔接的固定次序，如同瀑布流水，逐级下落。然而软件开发的实践表明，上述各项活动之间并非完全是自上而下，呈线性图式。实际情况是，每项开发活动均应具有以下特征：

(1) 从上一项活动接受本项活动的工作对象，作为输入；

(2) 利用这一输入实施本项活动应完成的内容；

(3) 给出本项活动的工作成果，作为输出传给下一项活动；

(4) 对本项活动实施的工作进行评审。若其工作得到确认，则继续进行下一项活动，否则返回前项，甚至更前项的活动进行返工。

5. 答：软件工程包括三个要素：方法、工具和过程。

软件工程方法为软件开发提供了“如何做”的技术。它包括了多方面的任务，如项目计划与估算、软件系统需求分析、数据结构、系统总体结构的设计、算法过程的设计、编码、测试以及维护等。软件工程方法常采用某一种特殊的语言或图形的表达方法及一套质量保证标准。

软件工具为软件工程方法提供了自动的或半自动的软件支撑环境。目前，已经推出了许多软件工具，已经能够支持上述的软件工程方法。特别地，已经有人把诸多的软件工具集成起来，使得一种工具产生的信息可以为其他的工具所使用，这样建立起一种被称之为计算机辅助软件工程（CASE）的软件开发支撑系统。CASE 将各种软件工具、开发机器和一个存放开发过程信息的工程数据库组合起来形成一个软件工程环境。

软件工程的过程则是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。过程定义了方法使用的顺序、要求交付的文档资料、为保证质量和协调变化所需要的管理、及软件开发各个阶段完成的里程碑。

6. 答：在软件开发过程中必须遵循下列软件工程原则。

(1) 抽象：采用分层次抽象，自顶向下、逐层细化的办法进行功能分解和过程分解，可以由抽象到具体、由复杂到简单，逐步得到问题的解。

(2) 信息隐蔽：遵循信息封装，使用与实现分离的原则，将模块设计成“黑箱”，可以将实现的细节隐藏在模块内部，使用者只能通过模块接口访问模块中封装的数据。

(3) 模块化：按模块划分系统的体系结构，使得各模块间有良好的接口。这样有助于信息隐蔽和抽象，有助于表示复杂的系统。

(4) 局部化：按抽象数据类型思想及问题域中的概念来建立模块，确保模块之间低耦合，模块内部高内聚。这有助于控制解的复杂性。

(5) 确定性：软件开发过程中所有概念的表达应是确定的、无歧义性的、规范的。这有助于人们之间的沟通，保证整个开发工作协调一致。

(6) 一致性：强调软件开发过程的标准化、统一化。包括文档格式的一致，工作流程的一致，内、外部接口的一致，系统规格说明与系统行为的一致等。

(7) 完备性：软件系统不丢失任何重要成分，可以完全实现系统所要求功能。

(8) 可验证性：开发大型的软件系统需要对系统自顶向下、逐层分解。系统分解应遵循系统易于检查、测试、评审的原则，以确保系统的正确性。

7. 答：软件开发时，一个错误发现得越晚，为改正它所付出的代价就越大。这个说法是对的。在 1970 年代，GTE、TRW 和 IBM 等三家公司对此问题做了独立研究，最后它们

得到相似的结论：

阶 段	需求分析	软件设计	程序编码	单元测试	验收测试	维 护
相对修复代价	0.1 ~ 0.2	0.5	1	2	5	20

从表中可以看出，在需求分析阶段检查和修复一个错误所需的代价只有编码阶段所需代价的 $1/5$ 到 $1/10$ ，而在维护阶段做同样的工作所付出的代价却是编码阶段的 20 倍。

8. 答：软件是为了特定目的而开发的程序、数据和文档的集合。

程序：能够执行特定功能的计算机指令序列。

数据：执行程序所必须的数据和数据结构。大量的数据都是按照一定的数据结构由用户在使用软件的过程中积累起来的。

文档：与程序开发，维护和使用有关的图文资料。

9. 答：按软件的功能进行划分：可分为系统软件和应用软件。

按软件工作方式划分：可分为分时软件、交互式软件、并行处理软件。

(1) 分时软件：允许多个联机用户同时使用计算机软件。

(2) 交互式软件：能实现人机通信的软件。

(3) 并行处理软件：能够将一件任务，分配给多个处理器，同时协同处理，达到高速完成的效果的软件。

10. 答：包括 4 个方面，即抽象性、复杂性、维护长期性、高成长性。

11. 答：软件的发展经历了三个阶段，即程序设计阶段、程序系统阶段、软件工程阶段。

12. 答：软件工程是用工程、科学和数学的原则与方法研制、维护计算机软件的有关技术及管理方法。

13. 答：软件工程的目标是为了解决软件开发和生产中的各种问题，获得高质量、低成本、高可靠性、易维护、并能及时投放市场的软件产品。

14. 答：主要有 4 条：

(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；

(3) 实行严格的产品控制；

(4) 结果应能清楚地审查。

15. 答：软件工程的研究对象由三个具有层次关系的要素组成：过程、方法和工具。

16. 答：软件工程过程：软件工程过程包含软件开发、维护以及软件开发和维护时所需的管理活动。

软件生命周期：软件生命周期是指软件产品从考虑其概念开始，到该软件产品不再能使用为止的整个时期。具体地说，软件生命周期是指从时间角度对软件开发和维护的复杂问题进行分解，把软件生命的漫长周期依次划分为若干个阶段，每个阶段都有相对独立的任务，然后逐步完成每个阶段的任务。一般将软件生命周期划分为六个阶段：软件项目计划、软件需求分析和定义、软件设计、程序编码、软件测试，以及运行维护。

计划阶段所需要完成的工作包括软件的可行性分析和制定开发计划。主要回答的问题是

“用户要解决的问题是什么”。软件系统的开发必须在可行性研究的基础上进行，要从技术上、经济上和社会因素方面进行研究，通过具体的成本 - 效益数值说明软件项目开发的可行性。通过对原有旧系统的调查，将新建的系统用规范的描述工具描述，得出新系统的模型，对新建系统的模型进行论证，最终形成可行性研究报告，并交给有关人员审查以决定软件项目是否可以开发。对于可行的软件项目要进行开发，必须要审定项目的开发计划、估算费用、确定资源分配和项目开发的速度安排，这就需要制定出软件项目的开发计划。

需求分析阶段的主要任务是确定所要开发的软件系统需要具备哪些功能，也就是说准确地确定软件系统“做什么”的问题。

设计阶段的主要任务是将分析阶段得出的系统逻辑模型转化为具体的计算机软件方案，也就是确定软件“怎么做”的问题。设计阶段主要包括软件的总体结构设计和对各个具体模块的详细设计，有时也可以分别称为概要设计和详细设计。设计阶段的成果是软件设计说明书。

编码阶段是将设计阶段的结果“翻译”成指定的计算机程序语言的源程序。

测试阶段软件测试是为了发现错误而执行程序的过程。

运行维护阶段是软件生存期的最后一个阶段。根据对软件进行维护的目的不同，可以将软件维护分为改正性维护、适应性维护和完善性维护三种。

17. 答：软件过程可以通过软件过程模型来表示。如瀑布模型、原型模型、快速应用开发模型、演化模型、喷泉模型等，其中最经典的过程模型是瀑布模型。该模型将软件生存周期的各项活动规定为按照固定顺序连接的若干阶段工作，自上而下，相互衔接，如同瀑布流水，逐级下落。它包括软件可行性分析和制定项目开发计划、需求分析、概要设计、概要设计、详细设计、软件编码、软件测试软件维护等阶段，各阶段相互衔接、次序固定。优点是支持结构化软件开发、控制了软件开发的复杂性、促进了软件开发工程化。缺点是缺少灵活性，前一阶段遗留下的问题如果不能及时解决，将给软件开发的后期带来隐患。

18. 答：常见的软件开发方法有结构化方法、面向数据结构的软件开发方法、面向对象的软件开发方法。

结构化开发方法是一种面向数据流的开发方法。结构化方法总的指导思想是自顶向下，逐步求精，其基本原则是功能的分解与抽象。

Jackson 方法是一种面向数据结构的开发方法，这种方法首先描述问题的输入及其输出的数据结构，分析其对应性，然后推导出相应的程序结构，最后得到所求问题的软件过程描述。

面向对象方法是以对象、类作为最基本的元素，通过对对象和类的描述解决应用问题，包括面向对象的分析、面向对象的设计、面向对象的实现等过程。

19. 答：软件生命周期是指软件产品从考虑其概念开始，到该软件产品不再能使用为止的整个时期，包括六个阶段：计划、需求分析、设计、编码、测试、运行维护。

把软件生命的漫长周期依次划分为若干个阶段，每个阶段都有相对独立的任务，然后逐步完成每个阶段的任务。在将软件的生命周期分解成几个相对独立的阶段之后，在每个阶段只需专注于该阶段问题的解决，也便于不同的人员分工合作，从而降低了整个软件开发的难度，使整个软件开发以一种有条不紊的方式进行，保证了软件的质量，也提高了软件的可维护性。

20. 答：线性顺序模型也称瀑布模型。瀑布模型将软件生存周期的各项活动规定为按照固定顺序连接的若干阶段工作，自上而下，相互衔接，如同瀑布流水，逐级下落。

优点：它在支持结构化软件开发、控制软件开发的复杂性、促进软件开发工程化等方面起着显著的作用。

缺点：首先，瀑布模型它要求在软件开发的初始阶段明确软件系统的全部需求，在实际中做到这一点是很困难的，甚至是不现实的。其次，使用瀑布模型开发软件，用户和项目管理者要等很长时间才能得到一份软件的最初版本，如果用户对该软件提出了较大的改进意见，将使整个项目蒙受巨大的损失。

21. 答：螺旋模型保留了传统生命周期逐步求精和细化的方法，但把它们综合到一个重复的框架以后，就可以对这个真实世界做出更加现实的反映。将风险分析紧密地融入到软件开发中，使得开发人员和客户对每个演化层出现的风险有所了解，继而做出应有的反应。

它可能难以使用户相信这种演化方法是可控的。更为重要的是，它的成功依赖于相当丰富的风险评估经验和专门知识，如果项目风险较大，又未能及时发现，势必造成重大损失。

第二章

一、填空题

1. 硬件、软件、人、数据库
2. 开发的、资源的、技术

二、简答题

1. 答：(1) 所期望的功能和性能是什么？
(2) 可靠性和质量问题是哪些？
(3) 总的系统目标是什么？
(4) 成本与进度限制如何？
(5) 制造的需求是什么？
(6) 市场与竞争情况怎样？
(7) 有效的技术有哪些？
(8) 将来可能有哪些扩充？

2. 答：可行性研究目录：

- (1) 引言：问题；实现环境；约束条件。
- (2) 管理：重要的发现；注解；建议；效果。
- (3) 方案选择：选择系统配置；选择方案标准。
- (4) 系统描述：缩写词；各子系统的可行性。
- (5) 成本—效益分析。
- (6) 技术风险评价。
- (7) 有关法律问题。
- (8) 其他。

3. 答：评审系统定义其目的是要保证：

- (1) 正确地定义了项目的范围；
- (2) 恰当地定义了功能、性能和接口；
- (3) 环境的分析和开发风险证明了系统是可行的；
- (4) 开发人员与用户对系统的目标达成了共识。

4. 答：系统技术评审的评审内容包括：

- (1) 系统的功能复杂性是否与开发风险、成本、进度的评估相一致？
- (2) 功能分配是否定义得足够详细？
- (3) 系统元素之间的接口，系统元素与环境的接口是否定义得足够详细？
- (4) 在规格说明中是否考虑了性能、可靠性和可维护性问题？
- (5) 系统规格说明是否为后续的硬件和软件工程步骤提供了足够的基础？

5. 答：(1) 是否已经建立了稳定的商业需求？系统可行性是否合理？

- (2) 特定的环境（或市场）是否需要所描述的系统？
- (3) 考虑了哪些候选方案？
- (4) 每个系统元素的开发风险有哪些？
- (5) 资源对于系统的开发是有效的吗？
- (6) 成本与进度界限合理吗？

6. 答：数据流图可以用来抽象地表示系统或软件。它从信息传递和加工的角度，以图形的方式刻画数据流从输入到输出的移动变换过程，同时可以按自顶向下、逐步分解的方法表示内容不断增加的数据流和功能细节。因此，数据流图既提供了功能建模的机制，也提供了信息流建模的机制，从而可以建立起系统或软件的功能模型。

数据流图的基本成分有 4 种：



加工。输入数据在此进行变换产生出数据，其中要注明加工的名字。



数据输入的源点 (Source) 或数据输出的汇点 (Sink)。其中要注意的源点或汇点的名字。



数据流。被加工的数据与流向，箭头边应给出数据流名字，可用名词或名词性短语命名。



数据存储文件。也必须加以命名，用名词或名词性短语命名。

7. 答：可行性研究主要做 4 个方面的研究：

经济可行性：进行成本 / 效益分析。从经济角度判断系统开发是否“合算”。

技术可行性：进行技术风险评价。从开发者的技术实力、以往工作基础、问题的杂性等出发，判断系统开发在时间、费用等限制条件下成功的可能性。

法律可行性：确定系统开发可能导致的任何侵权、妨碍和责任。

方案的选择：评价系统或产品开发的几个可能的候选方案。最后给出结论意见。

第三章

一、填空题

系统的可行性、经济、技术、硬件、软件、人、进度、规格说明

二、单项选择题

1.

分析：作为需求分析阶段工作的复查手段，在需求分析的最后一步，应该对功能的正确性、完整性和清晰性，以及其他需求给予评价。一般评审的结果都包括了一些修改意见，待修改完成后经评审通过，才可进入设计阶段。

2.

分析：虽然各种分析方法都有独特的描述方法，但所有的分析方法还是有它们共同适用的基本原则。这些基本原则包括：

- 需要能够表达和理解问题的信息域和功能域；
- 要能以层次化的方式对问题进行分解和不断细化；
- 要分别给出系统的逻辑视图和物理视图。

3.

分析：数据流图从数据传递和加工的角度，以图形的方式刻画数据流从输入到输出的移动变换过程，所以，它不是描述数据的静态结构，而是描述数据流的传递和变换。数据词典主要用于定义数据和控制对象的细节，结构化英语、判定表和判定树主要用于描述加工规格说明，都不是表达数据在系统内运动情况的工具。

4.

分析：在状态迁移图中，由一个状态和一个事件所确定的下一状态可能会有多个。实际会迁移到哪一个状态，是由更详细的内部状态和更详细的事件信息来决定的，此时在状态迁移图中可能需要使用加进判断框和处理框的记法。状态迁移图的优点：第一，状态之间的关系能够直观地捕捉到，这样用眼睛就能看到是否所有可能的状态迁移都已纳入图中，是否存在不必要的状态等。第二，由于状态迁移图的单纯性，能够机械地分析许多情况，可很容易地建立分析工具。

5.

分析：使用实体关系图，可以建立系统中各个数据对象及对象之间的关系。对象的实例间的关联称为“基数”，共有3种类型的基数：一对一，一对多，多对多。它反映了现实世界中实体之间的联系，多对一的情况可以归入一对多的关联中去。

6.

分析：结构化程序设计是在详细设计和编码阶段所采用的技术，而不是需求分析阶段要采用的技术。

7.

分析：在需求分析阶段，分析人员可以用数据流图描述系统的数据流的变换和流向，用

数据词典定义在数据流图中出现的数据流、数据文件、加工或处理，用判定表表示复杂条件和动作组合的情况。但 PAD 图是在详细设计阶段使用的描述加工逻辑的工具，不适用于需求分析。

8.

分析：软件需求分析阶段只确定软件系统要“做什么”，完成对重要功能、性能、确认准则的描述，至于“怎么做”由后续的设计阶段完成，对算法的详细过程性描述也是在设计阶段给出。

9.

分析：软件需求分析阶段只确定软件系统要“做什么”，完成对重要功能、性能、确认准则的描述，至于“怎么做”由后续的设计阶段完成，对算法的详细过程性描述也是在设计阶段给出。

10.

分析：软件可行性分析应在需求分析之前，所以需求分析规格说明不能成为可行性分析的依据。

11.

分析：使用原型的原型化方法特别适用于需求不确定性较高的软件系统的开发。它的基本思想是根据用户给出的基本需求，通过快速实现构造出一个小型的可执行的系统界面模型，满足用户的基本要求。

12.

分析：原型的原型化方法的基本思想是根据用户给出的基本需求，通过快速实现构造出一个小型满足用户的基本要求的可执行模型，这个可执行的模型就是系统界面原型。

13.

分析：通过快速实现构造出一个小型的可执行的模型，满足用户的基本要求，让用户在计算机上实际运行这个用户界面原型，在试用的过程中得到亲身感受和受到启发，做出反应和评价。

14.

分析：让用户在计算机上实际运行用户界面原型，在试用的过程中得到亲身感受和受到启发，做出反应和评价，提出同意什么和不同意什么。然后开发者根据用户的意见对原型加以改进。随着不断试验、纠错、使用、评价和修改，获得新的原型版本，如此周而复始，逐步减少分析和通信中的误解，弥补不足之处，进一步确定各种需求细节，适应需求的变更，从而提高了最终产品的质量。

15.

分析：通常，原型是指模拟某种产品的原始模型。在软件开发中，原型是软件的一个早期可运行的版本，它反映最终系统的部分重要特性。

原型化方法的基本思想是根据用户给出的基本需求，通过快速实现构造出一个小型的可执行的系统界面原型。让用户计算机上实际运行这个用户界面原型，在试用的过程中得到亲身感受和受到启发，做出反应和评价，提出同意什么和不同意什么。然后开发者根据用户的意见对原型加以改进。随着不断试验、纠错、使用、评价和修改，获得新的原型版本，如此周而复始，逐步减少分析和通信中的误解，弥补不足之处，进一步确定各种需求细节，适应

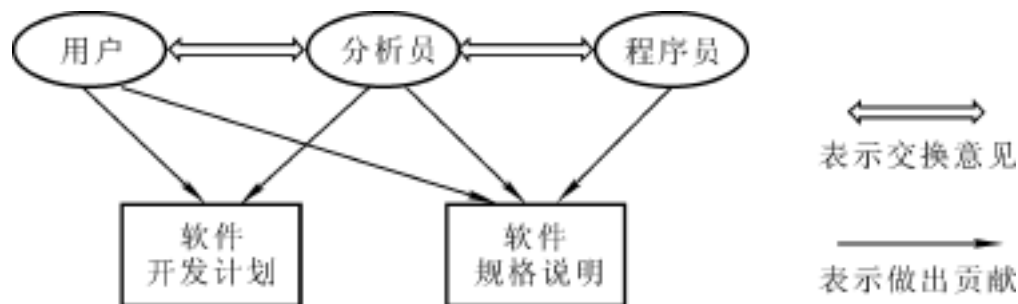
需求的变更，从而提高了最终产品的质量。这是一种自外向内的设计过程。

三、分析解答题

1. 答：所谓当前系统可能是需要改进的某个已在计算机上运行的数据处理系统，也可能是一个人工的数据处理过程。当前系统的物理模型客观地反映当前系统实际的工作情况。但在物理模型中有许多物理的因素，随着分析工作的深入，有些非本质的物理因素就成为不必要的负担，因而需要对物理模型进行分析，区分出本质的和非本质的因素，去掉那些非本质的因素即可获得反映系统本质的逻辑模型。所以当前系统的逻辑模型是从当前系统的物理模型抽象出来的。

2. 答：软件需求分析过程中，由于最初分析员对要解决的问题了解很少，用户对问题的描述、对目标软件的要求也很凌乱、模糊，再加上分析员和用户共同的知识领域不多，导致相互间通信的需求。首先，由于分析员和用户之间需要通信的内容相当多，业务知识上的不足，表达方式的不足，可能对某些需求存在错误解释或误解的可能性，造成需求的模糊性。其次，用户和分析员之间经常存在无意识的“我们和他们”的界限，不是按工作需要组成统一的精干的队伍，而是各自定义自己的“版图”，并通过一系列备忘录、正式的意见书、文档，以及提问和回答来相互通信。历史已经证明，这样会产生大量误解。忽略重要信息，无法建立成功的工作关系。

3. 答：系统分析员处在用户和高级程序员之间，负责沟通用户和开发人员的认识和见解，起着桥梁的作用。一方面要协助用户对所开发的软件阐明要求，另一方面还要与高级程序员交换意见，探讨用户所提要求的合理性以及实现的可能性。最后还要负责编写软件需求规格说明和初步的用户手册。



为能胜任上述任务，分析员应当具备如下的素质：

- (1) 能够熟练地掌握计算机硬、软件的专业知识，具有一定的系统开发经验。
- (2) 善于进行抽象的思维和创造性的思维，善于把握抽象的概念，并把它们重新整理成为各种逻辑成分，并给出简明、清晰的描述。
- (3) 善于从相互冲突或混淆的原始资料中抽出恰当的条目来。
- (4) 善于进行调查研究，能够很快学习用户的专业领域知识，理解用户的环境条件。
- (5) 能够倾听他人的意见，注意发挥其他人员的作用。
- (6) 具有良好的书面和口头交流表达能力。

4. 答：什么是信息？广义地讲，信息就是消息。宇宙三要素（物质、能量、信息）之一。它是现实世界各种事物在人们头脑中的反映。此外，人们通过科学仪器能够认识到的也是信息。信息的特征为：可识别、可存储、可变换、可处理、可传递、可再生、可压缩、可利用、可共享。我们通常讲的信息域就是对信息的多视角考虑。信息域包含 3 个不同的视

图：信息内容和关系、信息流和信息结构。为了完全理解信息域，必须了解每一个视图。

信息结构：它是信息在计算机中的组织形式。一般表示了各种数据和控制对象的内部组织。数据和控制对象是被组织成 n 维表格，还是组织成有层次的树型结构？在结构中信息与其他哪些信息相关？所有信息是在一个信息结构中，还是在几个信息结构中？一个结构中的信息与其他结构中的信息如何联系？这些问题都由信息结构的分析来解决。

信息流：表示数据和控制传递在系统中传递时的变化方式。输入对象首先被变换成中间信息（数据或控制），然后再变换成输出结果信息。沿着变换路径，可能从已有的数据存储（如磁盘文件或内存缓冲区）中引入附加的信息。对数据进行变换是程序中应有的功能或子功能。两个变换功能之间的数据传递就确定了功能间的接口。

所以，没有信息流的系统相当于没有功能的系统，这样的系统的存在是毫无意义的。而没有信息结构的系统是没有信息的系统，这样的系统不是计算机能够处理的系统。

5. 答：所有的需求分析方法都与一组操作性原则相关联：

- (1) 必须理解和表示问题的信息域。
- (2) 必须定义软件将完成的功能。
- (3) 必须表示软件的行为（作为外部事件的结果）。
- (4) 必须对描述信息、功能和行为的模型进行分解，能够以层次方式揭示其细节。
- (5) 分析过程应当从要素信息转向细节的实现。

(6) 通过使用这些原则，分析员可以系统地处理问题。首先检查信息域以便更完整地理解目标软件的功能，再使用模型以简洁的方式表达目标软件的功能和行为，并利用自顶向下、逐层分解的手段来降低问题的复杂性。在这些处理过程中，因处理需求带来的逻辑约束和因其他系统元素带来的物理约束需要通过软件要素和视图的实现加以检验和确认。

除此以外，Davis 建议了一组针对“需求工程”的指导性原则：

(1) 在开始建立分析模型之前应当先理解问题。如果问题没有很好理解就急于求成，常常会产生一个解决错误问题的完美的软件。

(2) 强力推荐使用原型。这样做可以使用户了解将如何与计算机交互，而人们对软件质量的认识常常是基于对界面“友好性”的切身体会。

(3) 记录每一个需求的起源和原因。这是建立对用户要求的可追溯性的第一步。

(4) 使用多个视图，建立系统的数据、功能和行为模型。这样做可帮助分析员从多方面分析和理解问题，减少遗漏，识别可能的不一致之处。

(5) 给需求赋予优先级。因为过短的时限会减少实现所有软件需求的可能性。因此，对需求排定一个优先次序，标识哪些需求先实现，哪些需求后实现。

(6) 注意消除歧义性。因为大多数需求都是以自然语言描述，存在叙述的歧义性问题，造成遗漏和误解。采用正式的技术评审是发现和消除歧义性的好方法。

遵循以上原则，就可能开发出较好的软件需求规格说明，为软件设计奠定基础。

6. 答：(1) 报名表、合格报名表、统计分析表。

(2) 难度分析表和分类统计表。

(3) “试题得分表”是图 (b) 中加工的内部文件，不必在图 (b) 中画出。

应注意的问题：

(1) 适当地为数据流、加工、文件、数据的源 / 汇点命名。名字应反映该元素的实际含

义，避免空洞的名字。如数据、信息处理、计算等名字都不好。

(2) 画数据流时不要夹带控制流。数据流图中各种数据的加工没有考虑时序关系，引入控制流后，加工之间就有了时序关系，这与画数据流图不考虑实现细节的初衷相违背。

(3) 一个加工的输出数据流不要与该加工的输入数据流重名，即使它们的组成成分相同。例如图 (c) 中加工 1.1 的输入数据流“报名表”与输出数据流“合格报名表”。

(4) 允许一个加工有多个数据流流向另一个加工，也允许一个加工有两个相同的输出数据流流向两个不同的加工。

(5) 保持父图与子图的平衡。就是说，父图与它的子图的输入数据流与输出数据流应当在数量与名字上都相同。特别的是，如果父图的一个输入（或输出）数据流对应于子图中几个输入（或输出）数据流，但子图中这几个数据流中的数据项合起来正好是父图中的那个数据流，这时它们还算是平衡的。例如，图 (b) 中加工 2 的输出数据流“统计分析表”是由“难度分析表”和“分类统计表”组成，那么图 (b) 与图 (d) 仍满足父图与子图平衡的条件。

(6) 在自顶向下的分解过程中，若一个文件首次出现时只与一个加工有关，那么这个文件应作为这个加工的内部文件而不必画出。例如，图 (d) 中的文件“试题得分表”就是图 (b) 中加工的内部文件，所以在图 (b) 中没有画出。

(7) 保持数据守恒。就是说，一个加工的所有输出数据流中的数据必须能从该加工的输入数据流中直接获得，或者是通过该加工产生的数据。

7. 答：分析模型中包含了对数据对象、功能和控制的表示。在每一种表示中，数据对象和控制项都扮演一定的角色。为表示每个数据对象和控制项的特性，建立了数据词典。数据词典精确地、严格地定义了每一个与系统相关的数据元素，并以字典式顺序将它们组织起来，使得用户和分析员对所有的输入、输出、存储成分和中间计算有共同的理解。

在数据词典的每一个词条中应包含以下信息：

- (1) 名称：数据对象或控制项、数据存储或外部实体的名字。
- (2) 别名或编号。
- (3) 分类：数据对象？加工？数据流？数据文件？外部实体？控制项（事件 / 状态）？
- (4) 描述：描述内容或数据结构等。
- (5) 何处使用：使用该词条（数据或控制项）的加工。

8. 答：传统软件生存期范型的典型代表是“瀑布模型”。这种模型的核心是将软件生存期划分为软件计划、需求分析、软件设计、编码、测试和运行维护等阶段，根据不同阶段工作的特点，运用不同的方法、技术和工具来完成该阶段的任务。软件开发人员遵循严格的规范，在每一阶段工作结束时都要进行严格的阶段评审和确认，以得到该阶段的一致、完整、正确和无歧义性的文档资料，并以它们做为下一阶段工作的基础。

传统思想强调每一阶段的严格性，尤其是开发初期要有良好的软件规格说明，主要是源于过去软件开发的经验教训，即在开发的后期或运行维护期间来修改不完善的规格说明要付出巨大的代价。但是，要想得到一个完整准确的规格说明不是一件容易的事。特别是对于一些大型的软件项目，在开发的早期用户往往对系统只有一个模糊的想法，很难完全准确地表达对系统的全面要求，软件开发人员对于所要解决的应用问题认识更是模糊不清。经过详细的讨论和分析，也许能得到一份较好的规格说明，但却很难期望该规格说明能将系统的各个

方面都描述得完整、准确、一致，并与实际环境相符。很难通过它在逻辑上推断出（不是在实际运行中判断评价）系统运行的效果，以此达到各方对系统的共同理解。随着开发工作向前推进，用户可能会产生新的要求，或因环境变化，要求系统也能随之变化；开发人员又可能在设计与实现的过程中遇到一些没有预料到的实际困难，需要以改变需求来解脱困境。因此规格说明难以完善，需求的变更，以及通信中的模糊和误解，都会成为软件开发顺利推进的障碍。尽管在传统软件生存期管理中通过加强评审和确认，全面测试，甚至依靠维护阶段能够缓解上述问题，但不能从根本上解决这些问题。

为了解决这些问题，逐渐形成了软件系统的快速原型的概念。由于运用原型的目的和方式不同，原型可分为以下两种不同的类型：

(1) 废弃型：先构造一个功能简单而且质量要求不高的模型系统，针对这个模型系统反复进行分析修改，形成比较好的设计思想，据此设计出更加完整、准确、一致、可靠的最终系统。系统构造完成后，原来的模型系统就被废弃不用。

(2) 追加型或演化型：先构造一个功能简单而且质量要求不高的模型系统，作为最终系统的核心，然后通过不断地扩充修改，逐步追加新要求，最后发展成为最终系统。

建立快速原型进行系统的分析和构造，有以下的优点：

(1) 增进软件者和用户对系统服务需求的理解，使比较含糊的具有不确定性的软件需求（主要是功能）明确化。由于这种方法能在早期就明确了用户的要求，因此可防止以后由于不能满足用户要求而造成的返工，从而避免了不必要的经济损失，缩短了开发周期。

(2) 软件原型化方法提供了一种有力的学习手段。通过原型演示，用户可以亲身体验早期的开发过程，获得关于计算机和被开发系统的专门知识。软件开发人员也可以获得用户对系统的确切要求，学习到应用范围的专业知识。

(3) 使用原型化方法，可以容易地确定系统的性能，确认各项主要系统服务的可应用性，确认系统设计的可行性，确认系统作为产品的结果。因而它可以作为理解和确认软件需求规格说明的工具。

(4) 软件原型的最终版本，有的可以原封不动地成为产品，有的略加修改就可以成为最终系统的一个组成部分，这样有利于建成最终系统。

9. 答：原型的开发和使用过程叫做原型生存期。下图是原型生存期的模型及其细化。

(1) 快速分析：在分析者和用户的紧密配合下，快速确定软件系统的基本要求。

(2) 构造原型：根据基本规格说明，尽快实现一个可运行的原型系统。

(3) 运行和评价原型：用户试用原型，考核评价原型的特性。纠正过去交互中的误解和分析中的错误，增补新的要求，提出全面的修改意见。

(4) 修正和改进：根据修改意见进行修改。如果用修改原型的过程代替快速分析，就形成了原型开发的迭代过程。在一次次的迭代过程中不断将原型完善，以接近系统的最终要求。

(5) 判定原型完成：经过修改或改进的原型，达到参与者一致认可，则原型开发的迭代过程可以结束。为此，应判断有关应用的实质是否已经掌握，判定的结果有两个不同的转向，一是继续迭代验证，一是进行详细说明。

(6) 判断原型细部是否说明：判断组成原型的细部是否需要严格地加以说明。

(7) 原型细部的说明：通过文件加以说明那些不能通过原型说明的项目。

(8) 判定原型效果：考察新加入的需求信息和细部说明信息，看其对模型有什么影响。是否会影响模块的有效性。如果模型受到影响，则要进行修正和改进。

(9) 整理原型和提供文档。

快速原型方法的提出使得传统的软件生存期在思想方法上受到了影响。如果只是在局部运用原型化方法，若将原型开发过程用于软件生存期的某一个阶段内，那么传统的软件生存期依然不变，只是阶段内部的软件定义或开发活动采用了新的方法。但若原型开发过程代替了传统生存期中的多个阶段，则软件开发过程就成为一种新的形式。

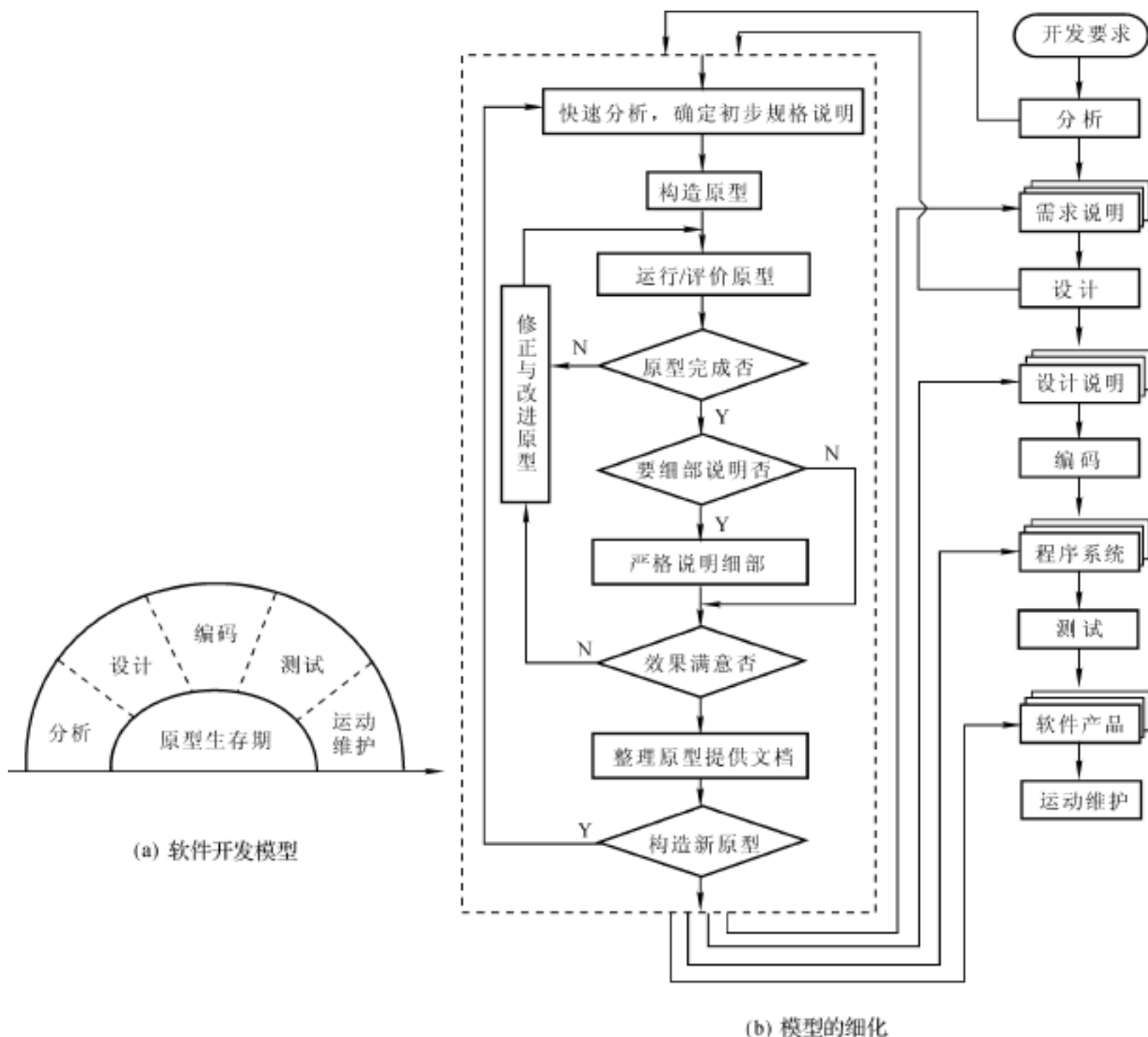


图 (a) 表示了使用原型方法的软件生存期模型。原型开发过程处于核心，表示可在生存期的任何阶段中引入原型开发过程，也可合并若干阶段，用原型开发过程代替。图 (b) 详细描述了在各个阶段可能引入原型开发过程的软件开发过程。其中在原型开发过程的最后加上一个“是否构造新原型”的判断，这是针对在系统开发的过程中有可能为不同的目的而要使用多个原型的情况而设。

(1) 辅助或代替分析阶段：在分析阶段利用快速原型方法可以得到良好的需求规格说

明。在整体上仍然采用传统的模式，但使用原型化方法来补充和完善需求说明以达到一致、准确、完整、无多义性地反映用户要求，从而代替了传统的仅由复审和确认来提高需求规格说明质量的方法。并能在早期克服潜在的误解、遗漏和错误，尽量不让潜在的问题遗留到开发的后期，减少将来维护的代价。

(2) 辅助设计阶段：在设计阶段引入原型，可根据需求分析得到的规格说明进行快速分析，得到实现方案后立即构造原型，通过运行，考察设计方案的可行性与合理性。在这个阶段引入原型，可以迅速得到完善的设计规格说明。原型可能成为设计的总体框架，也可能成为最终设计的一部分或补充的设计文档。

(3) 代替分析与设计阶段：这时不再遵循传统的严格按阶段进行软件开发的要求，而是把原型方法直接应用到软件开发的整体过程。在实施原型开发的过程中，不再考虑完善的需求说明，把分析、定义和设计交织在一起，通过原型的构造、评价与改进的迭代过程，逐步向最终系统的全面要求靠近。由于在分析的同时也考虑了设计与实现的要求，能更有效地确定系统的需求和设计规格说明。

(4) 代替分析、设计和实现阶段：在软件开发环境的支持下，通过原型生存期的反复迭代，直接得到软件的程序，交付系统测试。这属于进化型的原型开发，由初始的基本需求得到最初的原型开始，一直进化到软件的整体系统，并满足用户的一切可能的要求。

(5) 代替全部定义与开发阶段：这是典型的进化型原型开发方法。完全摆脱了传统的软件生存期模式，通过反复的原型迭代过程，直接得到最终的软件产品。系统测试作为原型评价工作的一部分，融入原型的开发过程。

10. 答：软件需求规格说明是分析任务的最终产物，通过建立完整的信息描述、详细的功能和行为描述、性能需求和设计约束的说明、合适的验收标准，给出对目标软件的各种需求。

软件需求规格说明的框架如下：

- (1) 引言：系统参考文献；整体描述；软件项目约束。
- (2) 信息描述：信息内容表示；信息流表示：数据流，控制流。
- (3) 功能描述：功能划分；功能描述：处理说明，限制/局限，性能需求，设计约束，支撑图；控制描述：控制规格说明，设计约束。
- (4) 行为描述：系统状态；事件和响应。
- (5) 检验标准：性能范围；测试种类；期望的软件响应；特殊的考虑。
- (6) 参考书目。
- (7) 附录。

11. 答：不能接受的3条意见是(5)，(6)，(10)。人机交互界面首先考虑的是用户如何使用起来方便，与编程习惯、设计技巧无关。此外，屏幕上信息应很清晰易懂，安全保密与屏幕显示无关。

12. 答：“自顶向下，逐步求精”是Niklaus Wirth提出的设计策略：即将软件的体系结构按自顶向下方式，对各个层次的过程细节和数据细节逐层细化，直到用程序设计语言的语句能够实现为止，从而最后确立整个的体系结构。

这样的结构实际就是一个模块的分层结构，即分层的过程。在实施时，采用抽象化的方法，自顶向下，给出不同的抽象层次。在最高的抽象层次上，可以使用问题所处环境的语言

概括地描述问题的解法。而在较低的抽象层次上，则采用过程化的方法。在描述问题的解法时，我们可以配合使用面向问题的术语和面向现实的术语。但最后在最低的抽象层次上，我们应使用能够直接实现的方式来描述这个解法。

13. 答：(1) 引言：文档的范围和目的、概述；目标；限制条件。

(2) 功能和数据描述：系统结构（结构环境图（ACD）；描述 ACD）。

(3) 子系统描述：对于子系统 n 的结构图描述：结构流程图（AFD）；系统模块描述；性能问题；设计限制条件；系统部件的分配。

结构字典。结构互连图及其描述。

(4) 系统模型化和模拟结果：用于模拟的系统模型、模拟结果、特殊的性能问题。

(5) 项目问题：项目的开发成本、项目进度计划。

第四章

一、填空题

- | | | |
|----------|---------|----------|
| 1. 巧合内聚 | 2. 逻辑内聚 | 3. 通信内聚 |
| 4. 过程内聚 | 5. 功能内聚 | 6. 简单性 |
| 7. 功能内聚 | 8. 一个 | 9. 数据型 |
| 10. 标准调用 | 11. 少 | 12. 控制范围 |

分析：(6 ~ 12 题)

模块之间的耦合（块间联系）和模块的内聚（块内联系）是评价程序模块结构质量的重要标准。联系的方式、共用信息的作用、共用信息的数量和接口的简单性等因素决定了块间联系的大小。在块内联系中，以功能内聚模块的块内联系最强。

SD 方法的总的原则是使每个模块只做一件事，就是说，只执行一个功能。模块之间尽可能传送简单的数据型参数。模块要通过标准调用语句调用其他模块，不要直接引用另一个模块内部的数据。同时模块之间传送的参数应尽量少。此外，SD 方法还提出了判定的作用范围和模块的控制范围等概念。SD 方法认为，模块的作用范围应该是其控制范围的子集。

- | | | |
|-------------|-----------------|-----------|
| 13. SP | 14. 块间联系小，块内联系大 | |
| 15. SC（结构图） | 16. NS 图 | 17. PAD 图 |
| 18. PDL | 19. 人 | 20. 追溯 |
| 21. 低耦合 | 22. 控制范围 | |

二、单项选择题

1.

分析：系统结构图反映的是系统中模块的调用关系和层次关系，谁调用谁，有一个先后次序（时序）关系。所以系统结构图既不同于数据流图，也不同与程序流程图。数据流图仅描述数据在系统中如何流动，如何处理和存储，它不考虑时序关系。图中的有向线段表示了数据流。程序流程图描述程序中控制流的情况，即程序中处理的执行顺序和执行序列所依赖的条件，图中的有向线段（流线）表示的是控制流，从一个处理走到下一个处理。但在系统

结构图中的有向线段表示调用时程序的控制从调用模块移到被调用模块，并隐含了当调用结束时控制将交回给调用模块。

如果一个模块有多个下属模块，这些下属模块的左右位置可能与它们的调用次序有关。例如，在用结构化设计方法依据数据流图建立起来的变换型系统结构图中，主模块的所有下属模块按逻辑输入、中心变换、逻辑输出的次序自左向右一字排开，左右位置不是无关紧要的。所以只有最后的一个叙述是正确的。

2.

分析：进入设计阶段之后，就开始着手解决“怎么做”的问题。一般把设计阶段的工作分成两步：即概要设计和详细设计。在概要设计阶段应着重解决实现需求的程序模块划分问题，在详细设计阶段则要决定每个模块的具体算法。

3.

分析：进入设计阶段之后，就开始着手解决“怎么做”的问题。一般把设计阶段的工作分成两步：即概要设计和详细设计。在概要设计阶段应着重解决实现需求的程序模块划分问题，在详细设计阶段则要决定每个模块的具体算法。

常见的软件概要设计方法有 3 大类：

- (1) 以数据流图为基础构造模块结构的结构化设计方法 (SD)；
- (2) 以数据结构为基础构造模块结构的 Jackson 方法和 LCP (Wanier) 逻辑构造方法；
- (3) 以对象、类、继承和通信为基础的面向对象设计方法 (OOD)。

此外，以信息隐蔽为原则的 Parnas 方法虽然没有给出系统化的设计方法，但它提出了一组原则，要求预先估计未来生存周期中可能发生的种种情况，并采取相应措施以提高软件系统的可维护性和可靠性。

面向数据结构的 Jackson 方法和 LCP 方法进一步解释如下：

Jackson 方法是一种典型的面向数据结构开发软件的方法。它的基本思想是首先根据实际问题，给出处理问题所需要和产生的数据结构，一旦搞清了问题的输入 / 输出数据结构，就可以以简单的方式直接导出程序的处理结构，然后应用 Jackson 的描述符号，将这个处理结构转换为程序的过程性描述。

LCP 方法是另一种面向数据结构的方法，它也要先给出用 Wanier 图表示的处理问题所需要和产生的数据结构，再在 Wanier 图上直接将数据结构转换为加工处理的形式化表示，最后生成描述加工过程的伪代码，进行验证和优化。

4.

5.

分析：参看“内容提要”中有关模块独立性的介绍。

6.

分析：在状态-迁移图中，由一个状态和一个事件所确定的下一状态可能会有多个。实际会迁移到哪一个状态，是由更详细的内部状态和更详细的事件信息来决定的，此时在状态-迁移图中可能需要使用加进判断框和处理框的记法。状态-迁移图的优点：第一，状态之间的关系能够直观地捕捉到，这样用眼睛就能看到是否所有可能的状态迁移都已纳入图中，是否存在不必要的状态等。第二，由于状态-迁移图的单纯性，能够机械地分析许多情况，可很容易地建立分析工具。

7.

三、简答题

1. 答：软件设计既是过程又是模型。设计过程是一系列迭代的步骤，使设计人员能够描述被开发软件的方方面面。设计模型体现了自顶向下、逐步细化的思想，首先构造事物的整体，再逐步细化，引导人们构造各种细节。为了给软件设计人员提供一些指导，1995 年 Davis 提出了一系列软件设计的原则如下，其中有些修改和补充：

(1) 设计过程不应受“隧道视野”的限制。一位好的设计者应当考虑一些替代的手段。根据问题的要求，可以用基本的设计概念，如抽象、逐步求精、模块化、软件体系结构、控制层次、结构分解、数据结构、软件过程、信息隐蔽等，来决定完成工作的资源。

(2) 设计应能追溯到分析模型。由于设计模型中每一个单个成分常常可追溯到多个需求上，因此有必要对设计模型如何满足需求进行追踪。

(3) 设计不应当从头做起。系统是使用一系列设计模式构造起来的，很多模式很可能以前就遇到过。这些模式通常被称为可复用的设计构件。可以使用它们代替那些从头开始做的方法。时间短暂而资源有限！设计时间应当投入到表示真正的新思想和集成那些已有的设计模式上去。

(4) 设计应当缩短软件和现实世界中问题的“智力差距”，就是说，软件设计的结果应尽可能模拟问题领域的结构。

(5) 设计应具有一致性和集成性。如果一个设计从整体上看像是一个人完成的，那它就是一致的。在设计工作开始之前，设计小组应当定义风格和格式的规则，如果仔细定义了设计构件之间的接口，则该设计就是集成的。

(6) 使用上述的基本的设计概念，将设计构造得便于将来的修改。

(7) 应将设计构造得即使遇到异常的数据、事件或操作条件，也能平滑地、轻松地降级。设计良好的计算机程序应当永不“彻底停工”，它应能适应异常的条件，并且当它必须中止处理时也能以从容的方式结束。

(8) 设计不是编码，编码也不是设计。即使在建立程序构件的详细的过程设计时，设计模型的抽象级别也比源代码要高。在编码级别上作出的唯一设计决策是描述如何将过程性设计转换为程序代码的小的实现细节。

(9) 在开始着手设计时就应当能够评估质量，而不是在事情完成之后。利用上述的基本的设计概念和已有的设计方法，可以帮助设计者评估质量。

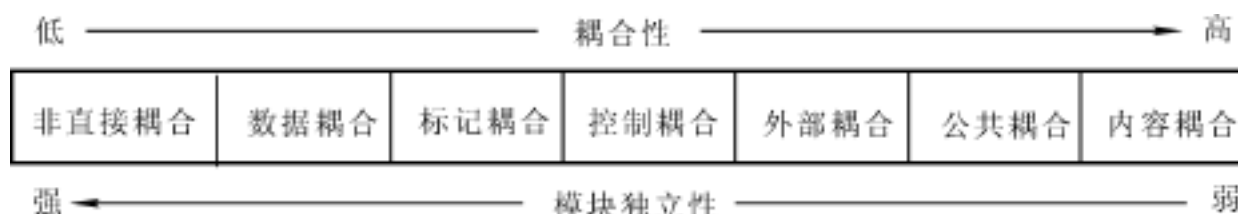
(10) 应当坚持设计评审以减少概念上（语义上）的错误。有时人们在设计评审时倾向于注重细节，只见树木不见森林。在关注设计模型的语法之前，设计者应能确保设计的主要概念上的成分遗漏、含糊、不一致都已检查过。

2. 答：如果两个模块互相独立，那么对其中一个模块进行编码、测试或修改时可以完全不考虑另一个模块对它的影响。因此，用模块独立性作为衡量模块结构是否容易编码、容易测试、容易修改的标准是合适的。但是，在一个系统的模块结构中没有哪两个模块可以完全独立，所以，要力争模块之间尽量独立，以得到一个质量良好的模块结构。

一般采用两个准则度量模块独立性。即模块间的耦合和模块的内聚。模块间的耦合是模块之间的相对独立性（互相连接的紧密程度）的度量。模块之间的连接越紧密，联系越多，

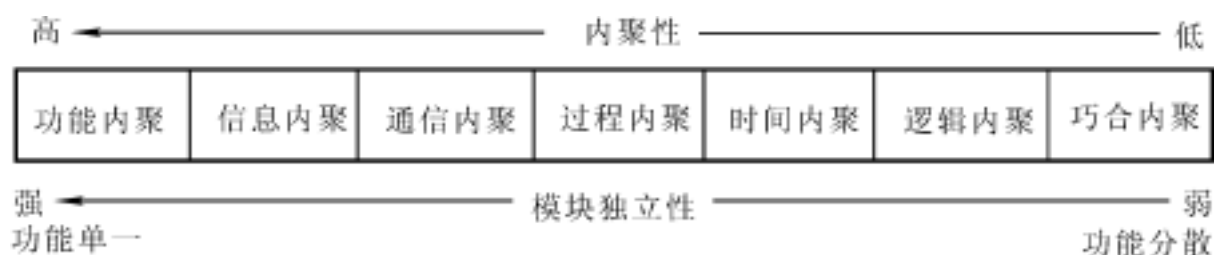
耦合性就越高，而其模块独立性就越弱。内聚是模块功能强度（一个模块内部各个成分彼此结合的紧密程度）的度量。一个模块内部各个成分之间的联系越紧密，则它的内聚性就越高，相对地，它与其他模块之间的耦合性就会减低，而模块独立性就越强。因此，模块独立性比较强的模块应是高内聚低耦合的模块。

一般模块之间可能的连接方式有 7 种，构成耦合性的七种类型。它们之间的关系为



低耦合的情形有非直接耦合、数据耦合和标记耦合，它们都是比较好的模块间的连接。特点是模块间的接口简单、规范。中度耦合的情形有控制耦合，它通过参数表传递控制参数。相对高的耦合情形有外部耦合和公共耦合，它们都是通过全局数据传递模块间的信息，不是说它们一定“坏”，但一定要注意使用这类耦合可能产生的后果，特别要防范这种后果。

一般模块的内聚性分为七种类型，它们的关系如下图所示。



在上面的关系中可以看到，位于高端的几种内聚类型最好，位于中段的几种内聚类型是可以接受的，但位于低端的内聚类型很不好，一般不能使用。因此，人们总是希望一个模块的内聚类型向高的方向靠。模块的内聚在系统的模块化设计中是一个关键的因素。

内聚和耦合是相互关联的。在程序结构中各模块的内聚程度越高，模块间的耦合程度就越低。但这也不是绝对的。我们的目标是力求增加模块的内聚，尽量减少模块间的耦合，但增加内聚比减少耦合更重要，应当把更多的注意力集中到提高模块的内聚程度上来。

3. 答：所谓“模块独立性”是指软件系统中每个模块只涉及软件要求的具体的子功能，而和软件系统中其他的模块的接口是简单的。所谓的“信息隐蔽”是指每个模块的实现细节对于其他模块来说是隐蔽的。也就是说，模块中所包含的信息（包括数据和过程）不允许其他不需要这些信息的模块使用。

如果软件系统做到了信息隐蔽，即定义和实施了对模块的过程细节和局部数据结构的存取限制，那么这些模块相互间的接口就是简单的。这组模块的独立性就比较强。事实上，衡量模块独立性的一个准则就是模块内聚，达到信息隐蔽的模块是信息内聚模块，它是高内聚情形，模块独立性当然很强了。

一个对象的抽象数据类型，就是信息隐蔽的示例。例如，对于栈 stack，可以定义它的操作 maknull（置空栈）、push（进栈）、pop（退栈）、gettop（取栈顶）和 empty（判栈空）。这些操作所依赖的数据结构是什么样的？它们是如何实现的？都被封装在其实现模块中。软件的其他部分可以直接使用这些操作，不必关心它的实现细节。一旦实现栈 stack 的模块里内部过程或局部数据结构发生改变，只要它相关操作的调用形式不变，则软件中其他

所有使用这个栈 stack 的部分都可以不修改。这样的模块结构具有很强的模块独立性。

4. 答：模块的内聚性与该模块在分层模块结构中的位置无关。事实上，一个好的模块化的程序系统，它所有的模块可以都是功能内聚的，即每一个模块就只干了一件事。用结构化设计方法建立起来的模块结构中的每一个模块都符合这个要求。把讨论范围再拓宽点，在纯面向对象范型的软件系统中，整个系统看作是一个类，它的子类可以看作是系统的子系统或高层模块，它们还可以有子类，……，这就形成一个类的层次结构。类的构造可以看成是一个抽象数据类型，实际上是信息内聚的。所以整个系统中从上到下，所有模块（对象类）都是信息内聚的模块。

5. 答：所谓“耦合性”是指模块之间联系的紧密程度的一种度量，而软件的“可移植性”是指将一个软件系统从一个计算机系统或环境移植到另一个计算机系统或环境中运行时所需工作量的大小。可移植性是用一组子特性，包括简明性、模块独立性、通用性、可扩充性、硬件独立性和软件系统独立性等来衡量的。如果一个软件具有可移植性，它必然耦合性低，这样模块独立性要强。例如，有一个图形处理软件，它应具有二维几何图形处理、三维几何图形处理、图形显示、外设控制、数据库管理、用户界面控制、设计分析等模块。如果这些模块之间都是通过参数表来传递信息，那么它们之间的耦合就是数据耦合或标记耦合等，都是低耦合。将来如果想要把它们移植到另一个外部环境中，这些模块容易修改（功能内聚），且接口清晰，修改可局部化。反言之，如果这些模块都是功能内聚或信息内聚的模块，模块之间的耦合都是低耦合，也对可移植性有促进。但不能讲具有低耦合性模块结构的软件一定具有可移植性，因为是否具有可移植性还有其他因素的影响。

6. 答：递归过程在求解复杂的大型问题时非常有效。常用的求解问题的方法，一种叫做“分而治之”的策略和“回溯”的策略，都可以用递归方法来解决。所谓“分而治之”的方法即是把大而复杂的问题化为规模稍小的子问题，用同样方法求解。如果分解后的子问题能够直接解决，就直接解出，然后再回推得到原来问题的解。所谓“回溯”方法就是如果一个大的问题在求解过程中从某一步出发有可选的多种解决方案，先选择一种解决方案，试探求解。如果求解失败，撤消原来的选择，再选一种解决方案，试探求解……。如果用某一方案求解成功，则退回上一步并报告这一步求解成功；如果所有可选方案都试过，都求解失败，则退回上一步并报告求解失败。

软件设计过程中的“自顶向下，逐层分解”的做法与上述求解问题的策略是一致的。如果分解出来的子问题（子功能、子模块）相互独立性比较强，这种分解可以降低模块的复杂性，做到模块化。所以，只要分解出来的子问题与原来问题满足递归的情况，用递归方法建立模块结构也是可行的。

7. 答：软件设计是一个把软件需求变换成软件表示的过程。最初这种表示只是描绘出软件的总的框架，然后进一步细化，在此框架中填入细节，把它加工成在程序细节上非常接近于源程序的软件表示。正因为如此，所以从工程管理的角度来看，软件设计分两步完成。首先做概要设计，将软件需求转化为数据结构和软件的系统结构。然后是详细设计，即过程设计。通过对结构表示进行细化，得到软件的详细的数据结构和算法。

由于概要设计建立起整个系统的体系结构框架，并给出了系统中的全局数据结构和数据库接口，人机接口，与其他硬、软件的接口。此外还从系统全局的角度，考虑处理方式、运行方式、容错方式、以及系统维护等方面的问题，并给出了度量和评价软件质量的方法，所

以它奠定了整个系统实现的基础。没有概要设计，直接考虑程序设计，就不能从全局把握软件系统的结构和质量，实现活动处于一种无序状态，程序结构划分不合理，导致系统处于一种不稳定的状态，稍一做改动就会失败。所以，不能没有概要设计。

8. 答：使用 PDL 语言，可以做到逐步求精：从比较概括和抽象的 PDL 程序起，逐步写出更详细的更精确的描述。下面举一个例子。

```

PROCEDURE spellcheck                                查找错拼的单词
BEGIN
    split document into singlewords                把整个文档分离成单词
    look up words in dictionary                    在字典中查这些单词
    display words which are not in dictionary      显示字典中查不到的单词
    create a new dictionary                        造一新字典
END spellcheck

```

这个例子只是搭起一个处理问题的框架。为进一步表明查找拼错的单词的 4 个步骤如何实现，可以对它每一步进行细化：

```

PROCEDURE spellcheck
BEGIN
    - - * split document into single words
    LOOP get next word
        add word to word list in sortorder
    EXITWHEN all words processed
    END LOOP
    - - * look up words in dictionary
    LOOP get word from word list
        IF word not in dictionary THEN
            - - * display words not in dictionary
            display word, prompt on user terminal
            IF user response says word OK THEN
                add word to good word list
            ELSE
                add word to bad word list
            END IF
        END IF
    END LOOP
    EXIT WHEN all words processed
    END LOOP
    - - * create a new words dictionary
    dictionary: = merge dictionary and good word list
END spellcheck

```

第五章

一、单项选择题

1.

分析：计算机用户通常使用“编辑程序 (Editor)”对源程序文本进行录入、编辑、存储，不用自举程序 (Bootstrap)、连接程序 (Loader)、文本格式化程序 (Textformatter)。

2.

分析：解释系统是边解释源程序边执行该源程序，编译程序是先编译出源程序的对应目标代码，再执行这些目标代码。所以编译程序编出的目标代码运行效率高。

3.

分析：FORTRAN 程序是以 SUBROUTINE 为单元的块状结构，对每一个 SUBROUTINE 进行编译后通过连接形成整个程序系统。它不是嵌套的。

4.

5.

6.

分析：美国国防部主持开发高级程序设计语言 Ada 时，曾确定以 ALGOL 语言作为 Ada 研究的出发点。所以，Ada, ALGOL, Pascal, BASIC 和 C 都是 ALGOL 系的一些程序语言。

7.

8.

分析：汇编程序实际是指汇编语言的处理程序，而用汇编语言写成的源程序一般称为汇编语言程序。

9.

分析：为了实现递归子程序的正确调用，一般使用堆栈来保存每次调用后返回到上一层程序的返回地址、本次递归调用时的形式参数、局部变量等。

10.

分析：为了实现递归子程序的正确调用，一般使用堆栈来保存每次调用后返回到上一层程序的返回地址、本次递归调用时的形式参数、局部变量等。

11.

分析：UNIX 操作系统是 Bell 实验室研制的。

12.

分析：UNIX 操作系统是 Bell 实验室研制的，用 C 语言写出来的。

二、多项选择题

1.

分析：如果程序结构的模块化满足评价的标准 (高内聚，低耦合)，这样的结构是容易编码，容易测试，容易理解，容易修改，容易维护的。程序的功能也容易扩充。特别适合于大型程序编制时，多人分工合作，协同完成任务的情形。因为是采用自顶向下，逐层分解来

划分模块结构的，所以模块之间的调用关系是分层次的模块结构，就叫做模块的层次结构。模块之间的信息传递叫做模块的接口，模块之间传递信息可以通过参数表、全局变量或全局数据结构、数据文件、专门的通信模块，不是专指数据文件。划分模块时，模块大小要适中。模块太大，控制路径数目多、涉及的范围广、变量的数目多、总体复杂性高，可理解性、可修改性、可靠性就会变差。模块太小，模块个数增大，调用的系统开销就会增大。所以要有一个权衡。

2.

分析：编制程序的过程实际上是根据设计的结果，用某种机器能够识别的程序设计语言，将设计翻译成机器代码的过程。因此，必须如实地按照设计说明书编写程序。至于设计说明书中含糊不清的地方，应当在编程时与分析人员或设计人员协商，对这些地方做出适当的解释。另外，考虑到将来的程序修改，必须为程序编写完整的说明书，同时程序必须编写得容易让别人看得懂，这样程序才容易修改，修改时不容易出错，而且容易验证修改后的结果。还有，编写程序的人不须重新考虑程序要完成什么功能，这些已经在软件分析与设计过程中充分考虑过了。

3.

分析：因为 条件语句和循环语句嵌套得过多会增加程序的复杂性，从而增加程序的出错率。虽然国际以至国内已经发表了编程语言的标准，但各个计算机厂商在推出自己的计算机系统的同时，也推出了针对自己机器特色的程序设计语言的非标准版本，如果利用这些语言的非标准特性编写程序，就会给将来程序的移植带来困难。为了提高程序的可移植性，应当只使用语言的标准版本，不要滥用语言的非标准特色。给在程序中使用的变量赋予与实际含义相符的名字，可以提高程序的可读性，从而提高程序的可维护性。程序优化的工作最好交给编译程序来做，程序员应把主要注意力放在提高程序的可读性、清晰性、简洁性、正确性、一致性等方面，从而保证软件的可靠性和可维护性。程序的可读性是至关重要的。所以程序的格式应有助于读者理解程序。程序中加入临时变量，可能会改变程序执行中的时序关系，造成程序出错。在表达式中加入括号，可以明确标明表达式的运算优先关系，避免因语言方面的原因可能潜藏的错误，程序模块的大小要适中，不是编得越短越好，注解加多少，由问题得难度来决定，但决不是可有可无的。最后要限制 GOTO 语句的使用，因为它可能会造成思路混乱、极易出错。

三、分析解答题

1. 答：使用 if - then - else 结构化构造，则上述程序段可改成如下形式。

```
if ( A < B and A < C ) then
    print A
else if ( A > = B and B < C ) then
    print B
else
    print C;
```

分析：理清程序流程图中每一条执行路径，适当利用复合的条件测试与判断语句，对每一个最终的打印处理，保留一个分支进入它。这样可以消除众多的 GOTO 语句，甚至可以消除嵌套的判断语句结构。这种程序结构清晰，可读性好。

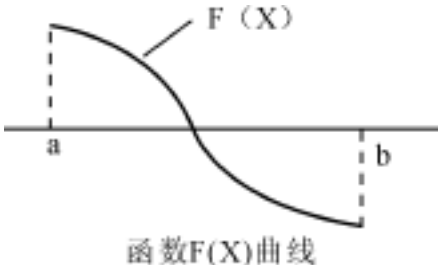
2. 答：利用一个布尔变量 `finished` (该变量的初值为 `false`)，当循环中求到了要求的结果时，将此变量的值改变为 `true`，表示循环应结束，`while` 循环测试到 `finished` 为 `true`，就自动退出循环，执行后续的语句。

【程序】

```

F0 = F(a); F1 = F(b);
if ( F0 * F1 <= 0 ) {
    X0 = a; X1 = b; i = 1; finished = 0; /* 设置控制循环结束的布尔变量 */
    while ( i <= n && finished == 0 ) { /* 单入口单出口 */
        Xm = (X0 + X1) / 2; Fm = F (Xm);
        if ( abs (Fm) < eps || abs (X1 - X0) < eps ) finished = 1;
        if ( finished == 0 ) {
            if ( F0 * Fm > 0 )
                { X0 = Xm; F0 = Fm; }
            else
                X1 = X;
        }
    }
}

```



分析：此程序段中各种结构均为单入口与单出口，且没有 `goto` 语句，可以说它是一个结构化的程序。它的结构很简单，只有一重循环，但由于引入一个布尔变量来控制循环结束，可读性比较差。在只有一重循环的情形，差的程度还不很明显，在多重循环的情形，引入多个布尔变量，可读性就很差了。理解程序的时间差几倍到几十倍不止。因此，对于这种单入口多出口的循环，R. S. Pressman 明确说明，用 `goto` 语句可得到较好的清晰性。在 C 中提供了一个可直接转向循环之后的语句 `break`，使用它可获得较好的效果。

【程序】

```

F0 = F (a); F1 = F (b); /* 区间两端点的函数值 */
if ( F0 * F1 <= 0 ) { /* 区间中没有根，不做 */
    X0 = a; X1 = b; /* 设置当前求根区间的两个端点 */
    for ( i = 1; i <= n; i++ ) { /* 最多允许迭代 n 次 */
        Xm = (X0 + X1) / 2; Fm = F (Xm);
        /* 求中点及中点的函数值 */
        if ( abs (Fm) < eps || abs (X1 - X0) < eps ) break;
        /* 求到，转出循环 */
        if ( F0 * Fm > 0 ) /* 没有求到，缩小求根区间 */
            { X0 = Xm; F0 = Fm; } /* 向右缩小区间 */
        else
            X1 = X; /* 向左缩小区间 */
    }
}
}

```

这段程序不是结构化的程序，利用了 C 语言中的一个语句 break，它的功能是将控制转移到它所在循环的后面第一个后续语句处。由于将转移语句与转出条件的判断直接联系在一起，可读性较好。

3. 答：(1) 的功能是对换 A [I] 与 A [T] 的内容。等效的程序段可以是：

```
WORK = A [T]; A [T] = A [I];    A [I] = WORK;
```

(2) 的功能是建立一个单位矩阵 V。等效的程序段可以是：

```
for ( i = 1; i <= n; i ++ )
    for ( j = 1; j <= n; j ++ )
        if ( i == j ) V [i] [j] = 1;
        else V [i] [j] = 0;
```

分析：阅读这两段程序，读者可能不易看懂，有时还需用实际数据试验一下。对于 (1)，如果我们给 A [I] 赋值 3，给 A [T] 赋值 5，在运算后发现 A [I] 中变成了 5，A [T] 中变成了 3。这段程序就是交换 A [I] 和 A [T] 中的内容。目的是为了节省一个工作单元。如果改一下：

```
WORK = A [T]; A [T] = A [I];    A [I] = WORK;
```

就能让读者一目了然。

对于 (2)，除法运算 (/) 在除数和被除数都是整型量时，其结果只取整数部分，而得到整型量。因此，

当 $i < j$ 时 i / j 为 0

当 $j < i$ 时 j / i 为 0，

只有当 $i = j$ 时 $(i / j) * (j / i) = 1$

这样得到的结果 V 是一个矩阵。如果直截了当地说明作者的意图：

```
if ( i == j ) V [i] [j] = 1; else V [i] [j] = 0;
```

让读者可以很容易地理解。所以，在程序设计时，应当首先考虑清晰性，不要玩技巧。

4. 答：早在 1963 年，针对当时流行的 ALGOL 语言，Peter Naur 指出，在程序中大量地，没有节制地使用 GOTO 语句，会使程序结构变得非常混乱。但是很多人还不太注意这一问题。以致许多人写出来的程序仍然是纷乱如麻的。

1965 年，E. W. Dijkstra 在一次会议上提出，应当把 GOTO 语句从高级语言中取消。并指出，程序的质量与程序中包含的 GOTO 语句的数量成反比。在这种思想的影响下，当时新开发的几种高级程序设计语言，例如 LISP、ISWIM、BLISS 等，都把 GOTO 语句取消了。

1966 年，Bohm 与 Jacopini 证明了任何单入口单出口的没有“死循环”的程序都能由三种最基本的控制结构构造出来。这三种基本控制结构就是“顺序结构”、“选择 IF - THEN - ELSE 结构”、“重复 DO - WHILE 或 DO - UNTIL 结构”。

1968 年，Dijkstra 在写给《ACM》(美国计算机协会通讯)杂志编辑部的信中再次建议从一切高级语言中取消 GOTO 语句，只使用三种基本控制结构编写程序。他的建议引起了激烈的争论。争论集中在如何看待 GOTO 语句的问题上。赞成取消 GOTO 语句的一方认为，GOTO 语句对程序清晰性有很大破坏作用，凡是使用 GOTO 语句多的程序，其控制流时而在 GOTO 向前，时而 GOTO 向后，常使程序变得很难理解，从而增加查错和维护的困难，降低程序的可维护性。但以 D. E. Knuth 为代表的另一方认为，GOTO 语句虽然存在着

破坏程序清晰性的问题，但不应完全禁止。因为 GOTO 语句概念简单，使用方便，在某些情况下，保留 GOTO 语句反能使写出的程序更加简洁，并且 GOTO 语句可直接得到硬件指令的支持。经过争论，人们认识到，不是简单地去掉 GOTO 语句的问题，而是要创立一种新的程序设计思想、方法和风格，以显著提高软件生产率和软件质量，降低软件维护的成本。

70 年代初 N. Wirth 在设计 Pascal 语言时对 GOTO 语句的处理可被当做对 GOTO 语句争论的结论。在 Pascal 语言中设置了支持上述三种基本控制结构的语句；另一方面，GOTO 语句仍然保留在该语言中。不过，N. Wirth 解释说，通常使用所提供的几种基本控制结构已经足够，习惯于这样做的人不会感到 GOTO 语句的必要。也就是说，在一般情况下，可以完全不使用 GOTO 语句。如果在特殊情况下，由于特定的要求，偶然使用 GOTO 语句能解决问题，那也未尝不可，只是不应大量使用罢了。

事实上，大量采用 GOTO 语句实现控制路径，会使程序路径变得复杂而且混乱，从而使程序变得不易阅读，给程序的测试和维护造成困难，还会增加出错的机会，降低程序的可靠性。因此要控制 GOTO 语句的使用。但有时完全不用 GOTO 语句进行程序编码，比用 GOTO 语句编出的程序可读性差。例如，在查找结束时，文件访问结束时，出现错误情况要从循环中转出时，使用布尔变量和条件结构来实现就不如用 GOTO 语句来得简洁易懂。

5. 答：开发规模相同，但复杂性不同的软件，花费的成本和时间会有很大的差异。因此到目前为止，还没有一个软件复杂性度量的方法能够全面、系统地度量任一软件的复杂性，某一种度量方法只偏重于某一方面。所以，用某一种软件复杂性来度量不同类型的程序，所得到的度量值不一定真正反映它们的复杂性。但对同一类型的程序，按某种视点来度量它们的复杂性，其结果还是比较有价值的。

6. 答：K. Magel 从 6 个方面描述软件复杂性：

- (1) 理解程序的难度；
- (2) 改错及维护程序的难度；
- (3) 向他人解释程序的难度；
- (4) 按指定方法修改程序的难度；
- (5) 根据设计文档编写程序的工作量；
- (6) 执行程序时需要资源的程度。

软件复杂性度量模型应遵循的基本原则：

- (1) 软件复杂性与程序大小的关系不是线性的；
- (2) 控制结构复杂的程序较复杂；
- (3) 数据结构复杂的程序较复杂；
- (4) 转向语句使用不当的程序较复杂；
- (5) 循环结构比选择结构复杂，选择结构又比顺序结构复杂；
- (6) 语句、数据、子程序和模块在程序中的次序对软件复杂性都有影响；
- (7) 全程变量、非局部变量较多时程序较复杂；
- (8) 参数按地址传递比按值传递更复杂；
- (9) 函数副作用比显式参数传递更难以琢磨；
- (10) 具有不同作用的变量共用一个名字时较难理解；

(11) 模块间或过程间联系密切的程序较复杂;

(12) 嵌套深度越深程序越复杂。

最典型的两种程序复杂性度量的方法中, McCabe 环路复杂性度量就是针对基本原则制定的度量模型; Halstead 软件科学则是针对程序中操作符和操作数的出现频度而制定的度量模型。

第六章

一、单项选择题

1.

2.

分析: 软件测试的目的是发现软件中的错误。因为不可能把所有可能的输入数据都拿来测试(时间花费不起), 为了提高测试的效率, 应该选择发现错误的可能性大的数据作为测试数据。

3.

分析: 使用白盒测试方法时, 确定测试数据应根据程序的内部逻辑和指定的覆盖标准, 可以不考虑程序的功能。与设计测试数据无关的文档是项目开发计划。

4.

5.

分析: 软件的集成测试工作最好由不属于该软件开发组的软件设计人员承担, 以提高集成测试的效果。

6.

分析: 1966 年, Bohm 与 Jacopini 提出任何单入口单出口的没有“死循环”的程序都能由三种最基本的控制结构构造出来。这三种基本控制结构就是“顺序结构”、“选择 IF - THEN - ELSE 结构”、“重复 DO - WHILE 或 DO - UNTIL 结构”。

7.

分析: 三种基本控制结构的共同点是只有一个入口和一个出口。

8.

分析: E. W. Dijkstra 提出了程序要实现结构化的主张, 并将这一类程序设计称为结构化程序设计。这种方法的一个重要原则就是采用自顶向下、逐步求精的方法编写程序。N. Wirth 曾做过如下说明: “我们对付一个复杂问题的最重要的方法就是抽象。因此, 对于一个复杂的问题, 不要急于马上用计算机指令、数字和逻辑符号来表示它, 而应当先用较自然的抽象的语句来表示, 从而得到抽象的程序。抽象程序对抽象的数据类型进行某些特定的运算, 并用一些合适的记号(可以是自然语言)来表示。下一步对抽象程序再做分解, 进入下一个抽象的层次。这样的细化过程一直进行下去, 直到程序能被计算机接受为止。此时的程序已经是用某种高级语言或机器指令书写的了。”

9.

10.

分析：软件调试则是在进行了成功的测试之后才开始的工作。它与软件测试不同，软件测试的目的是尽可能多地发现软件中的错误，但进一步诊断和改正程序中潜在的错误，则是调试的任务。调试活动由两部分组成：确定程序中可疑错误的确切性质和位置。对程序（设计，编码）进行修改，排除这个错误。

11.

分析：对可靠性要求很高的软件，由第三者对源代码进行逐行检查，这是代码审查。

12.

分析：软件变更时可能发生退化现象：原来正常的功能可能发生异常，性能也可能下降。因此，对变更的软件要进行退化测试。

13.

分析：基于被测试模块的内部结构或算法设计测试用例进行测试，这是白盒测试。

14.

分析：为了确认用户的需求，先做出系统的原型，提交给用户试用。

15.

分析：自顶向下对具有层次结构的大型软件进行集成测试时，需要设计一些虚拟模块来替代尚未测试过的下层模块，这些模块叫做桩模块。

16.

分析：软件测试方法可分为黑盒测试法和白盒测试法两种。黑盒测试法是基于程序的功能来设计测试用例的方法。除了测试程序外，它还适用于对需求分析阶段的软件文档进行测试。

17.

分析：黑盒测试法是基于程序的功能来设计测试用例的方法。除了测试程序外，它还适用于对需求分析阶段的软件文档进行测试。

18.

分析：白盒测试法是根据程序的内部逻辑来设计测试用例的方法。

19.

分析：白盒测试法除了测试程序外，它也适用于对软件详细设计阶段的软件文档进行测试。

20.

分析：白盒法测试程序时常按照给定的覆盖条件选取测试用例。判定覆盖比语句覆盖严格，它使得每一个判定的每一条分支至少经历一次。

21.

分析：判定/条件覆盖既是判定覆盖，又是条件覆盖，但它并不保证使各种条件都能取到所有可能的值。

22.

分析：多重条件覆盖，也叫组合条件覆盖，比其他条件都要严格，但它不能保证覆盖程序中的每一条路径。

23.

分析：单元测试一般以白盒法为主，测试的依据是系统的模块功能规格说明。

24.

分析：单元测试一般以白盒法为主，测试的依据是系统的模块功能规格说明。

25.

分析：软件测试中常用的静态分析方法是引用分析和接口分析。接口分析用于检查模块或子程序间的调用是否正确。

26.

27.

分析：分析方法（白盒方法）中常用的方法是路径测试方法。

28.

分析：非分析方法（黑盒方法）中常用的方法是等价类（划分）方法和因果图方法。因果图方法根据输出对输入的依赖关系设计测试用例。

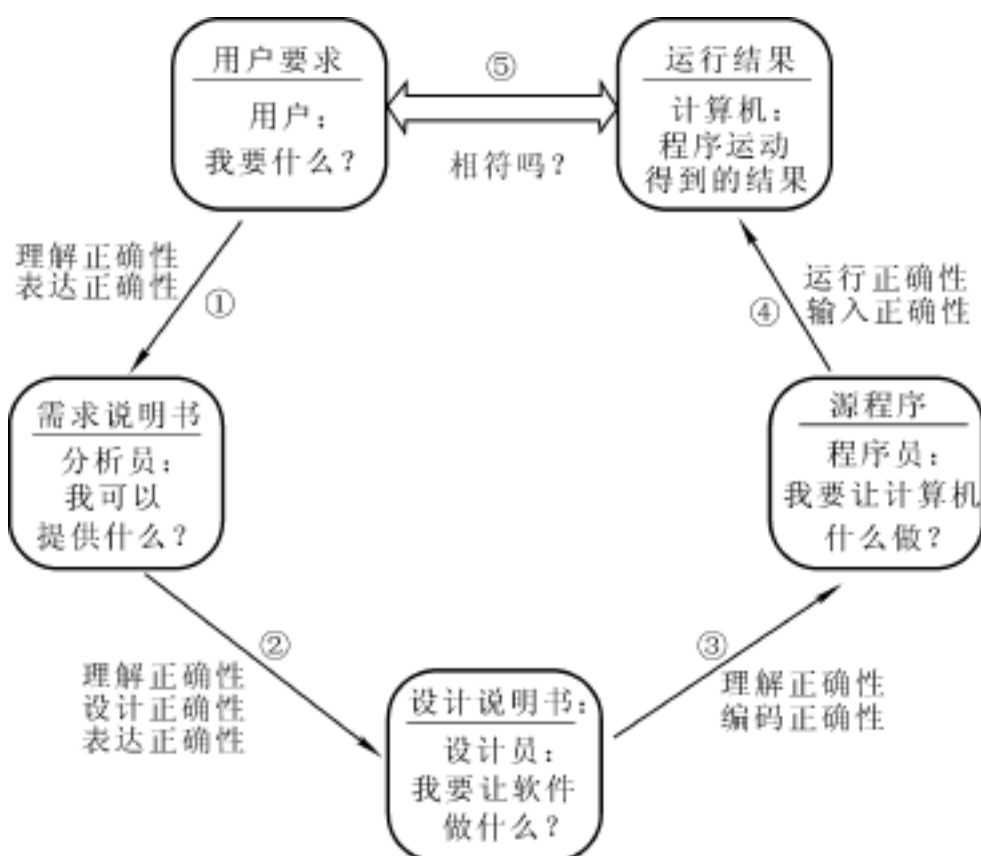
二、多项选择题

1. A. B. C. D. E. F.

分析：到程序的测试为止，软件开发工作已经经历了许多环节，每个环节都可能发生问题。为了把握各个环节的正确性，人们需要进行各种确认和验证工作。

所谓确认，是一系列的活动和过程，其目的是想证实在一个给定的外部环境中软件的逻辑正确性。它包括需求规格说明的确认和程序的确认，而程序的确认又分为静态确认与动态确认。静态确认一般不在计算机上实际执行程序，而是通过人工分析或者程序正确性证明来确认程序的正确性；动态确认主要通过动态分析和程序测试来检查程序的执行状态，以确认程序是否有问题。

所谓验证，则试图证明在软件生存期各个阶段，以及阶段间的逻辑协调性、完备性和正确性。下图中所示的就是软件生存期各个重要阶段之间所要保持的正确性。它们就是验证工作的主要对象。



确认与验证工作都属于软件测试。在对需求理解与表达的正确性、设计与表达的正确性、实现的正确性以及运行的正确性的验证中，任何一个环节上发生了问题都可能在软件测试中表现出来。

2. A. B. C. D. E. F.

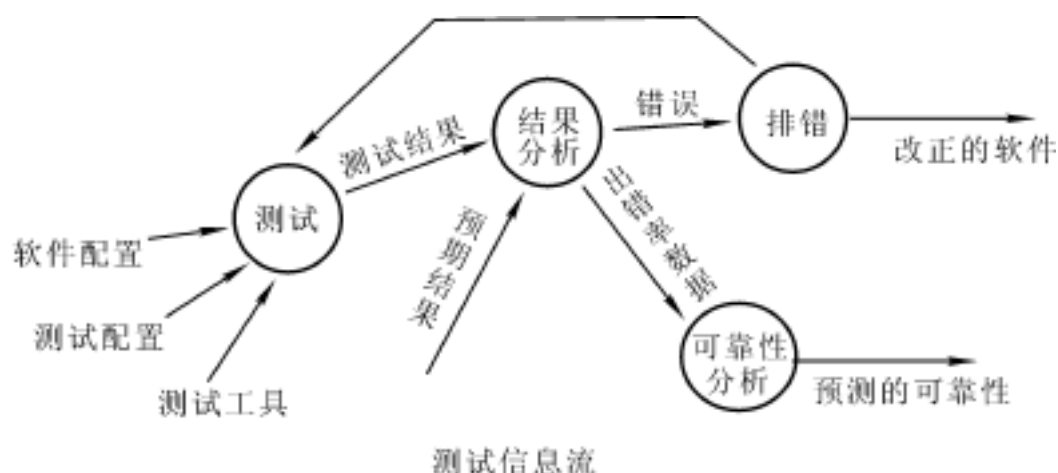
分析：测试信息流如图所示。测试过程需要三类输入：

(1) 软件配置：包括软件需求规格说明、软件设计规格说明、源代码等。

(2) 测试配置：包括表明测试工作如何进行的测试计划、给出测试数据的测试用例、控制测试进行的测试程序等；实际上，测试配置是软件配置的一个子集。

(3) 测试工具：为提高软件测试效率，测试工作需要测试工具的支持，它们的工作就是为测试的实施提供某种服务。例如，测试数据自动生成程序、静态分析程序、动态分析程序、测试结果分析程序、以及驱动测试的测试数据库等等。

测试之后，要对所有测试结果进行分析，即将实测的结果与预期的结果进行比较。如果发现出错的数据，就意味着软件有错误，然后就需要开始排错（调试）。即对已经发现的错误进行错误定位和确定出错性质，并改正这些错误，同时修改相关的文档。修正后的文档一般都要经过再次测试，直到通过测试为止。通过收集和分析测试结果数据，开始对软件建立可靠性模型。



最后，如果测试发现不了错误，那么几乎可以肯定，测试配置考虑得不够细致充分，错误仍然潜伏在软件中。这些错误最终不得不由用户在使用中发现，并在维护时由开发者去改正。但那时改正错误的费用将比在开发阶段改正错误的费用要高出 40 倍到 60 倍。

3. A. B. C. D. E.

分析：由于原始问题的复杂性，软件本身的复杂性和抽象性，软件开发各个阶段工作的多样性，以及参加开发各种层次人员之间工作的配合关系等因素，使得开发的每个环节都可能产生错误。这些错误和缺陷如果在软件交付投入生产性运行之前不能加以排除的话，在运行中迟早会暴露出来。但到那时，不仅改正这些错误的代价更高，而且往往造成很恶劣的后果。因此，为了保证所开发软件的质量，测试是不可少的。测试的任务就是设计出测试用例，运行被测程序，发现软件中隐藏的各种错误。目前在大中型软件项目的开发中。测试占据着重要的地位，测试所花费费用已超过软件开发费用的 70% 以上。

如何组织好测试，特别是如何选择好的测试用例，对于降低测试费用，保障测试质量有着重要的意义。一个高效的、或“高产”的测试，是指设计少量的测试用例，用它们来运行程序，发现被测程序的尽可能多的问题。

测试按照测试的内容可分为三种：单元测试（对程序单元或模块单独进行测试）、集成测试（把已通过单元测试的模块按照设计的要求组装起来，测试模块间的接口以及设计中的问题）和确认测试（对软件的有效性做全面测试，以确定开发的软件是否符合验收标准）。

根据国标 GB 8566 - 88 《计算机软件开发规范》的规定，软件的开发和维护划分为 8 个阶段：可行性研究及计划、需求分析、概要设计、详细设计、实现、集成测试、确认测试、使用维护，并且规定，单元测试是在实现阶段完成的，集成测试的计划应当在概要设计阶段制定，确认测试的计划应当在需求分析阶段制定。

4. A. B. C. D. E. F. G. H. I.

A, B 的答案可互换

分析：集成测试也叫做组装测试或联合测试。通常，在单元测试的基础上，将所有模块按照设计要求组装成为系统。子系统的集成测试特别称为部件测试，它所做的工作是要找出子系统和系统需求规格说明之间的不一致。这时需要考虑的问题是：

- (1) 在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；
- (2) 一个模块的功能是否会对另一个模块的功能产生不利的影响；
- (3) 各个子功能组合起来，能否达到预期要求的父功能；
- (4) 全局数据结构是否有问题；
- (5) 单个模块的误差累积起来是否会放大。

5. A. B. C. D. E. F. G. H. I.

B, C 的答案可互换。D, E 的答案可互换。F, G 的答案可互换。

分析：(1) 语句覆盖：就是设计若干个测试用例，运行被测程序，使得每一可执行语句至少执行一次。例如在本题所给出的例子中，正好所有的可执行语句都在路径 L1 上，所以选择路径 L1 设计测试用例，就可以覆盖所有的可执行语句。

测试用例的设计格式如下

【输入的 (A, B, x), 输出的 (A, B, x)】

本题满足语句覆盖的测试用例是：

【(2, 0, 4), (2, 0, 3)】覆盖 ace 【L1】 (答案)

从程序中每个可执行语句都得到执行这一点来看，语句覆盖的方法似乎能够比较全面地检验每一个可执行语句。但需要注意的是，这种覆盖也绝不是完美无缺的。假设图中两个判断的逻辑运算有问题，例如，第一个判断中的逻辑运算符“&”错写成了“&”，或者第二个判断中的逻辑运算符“>”错写成了“<”，利用上面的测试用例，仍可覆盖所有 4 个可执行语句。这说明虽然做到了语句覆盖，但可能发现不了判断中逻辑运算中出现的错误。与后面所介绍的其他覆盖相比，语句覆盖是最弱的逻辑覆盖准则。

(2) 判定覆盖：就是设计若干个测试用例，运行被测程序，使得程序中每个判断的取真分支和取假分支至少经历一次。判定覆盖又称为分支覆盖。对于本题所给出的例子，如果选择路径 L1 和 L2，就可得满足要求的测试用例：

【(2, 0, 4), (2, 0, 3)】覆盖 ace 【L1】 (答案)

【(1, 1, 1), (1, 1, 1)】覆盖 abd 【L2】

如果选择路径 L3 和 L4, 还可得另一组可用的测试用例:

【(2, 1, 1), (2, 1, 2)】覆盖 abe 【L3】 (答案)

【(3, 0, 3), (3, 1, 1)】覆盖 acd 【L4】

所以, 测试用例的取法不唯一。注意有例外情形, 例如, 若把图例中第二个判断中的条件 $x > 1$ 错写成 $x < 1$, 那么利用上面两组测试用例, 仍能得到同样结果。这表明, 只是判定覆盖, 还不能保证一定能查出在判断的条件中存在的错误。因此, 还需要更强的逻辑覆盖准则去检验判断内部条件。

(3) 条件覆盖: 就是设计若干个测试用例, 运行被测程序, 使得程序中每个判断的每个条件的可能取值至少执行一次。例如在本题所给出的例子中, 我们事先可对所有条件的取值加以标记。例如,

对于第一个判断: 条件 $A > 1$ 取真值为 T1, 取假值为 $\overline{T1}$

条件 $B = 0$ 取真值为 T2, 取假值为 $\overline{T2}$

对于第二个判断: 条件 $A = 2$ 取真值为 T3, 取假值为 $\overline{T3}$

条件 $x > 1$ 取真值为 T4, 取假值为 $\overline{T4}$

则可选取测试用例如下:

测试用例 (答案)	通过路径	条件取值	覆盖分支
【(2, 0, 4), (2, 0, 3)】	ace (L1)	T1 T2 T3 T4	c, e
【(1, 0, 1), (1, 0, 1)】	abd (L2)	$\overline{T1}$ $\overline{T2}$ $\overline{T3}$ $\overline{T4}$	b, d
【(2, 1, 1), (2, 1, 2)】	abe (L3)	T1 $\overline{T2}$ T3 $\overline{T4}$	b, e

或

测试用例 (答案)	通过路径	条件取值	覆盖分支
【(1, 0, 3), (1, 0, 4)】	ace (L3)	$\overline{T1}$ T2 $\overline{T3}$ T4	b, e
【(2, 1, 1), (2, 1, 2)】	abe (L3)	T1 $\overline{T2}$ T3 $\overline{T4}$	b, e

注意, 前一组测试用例不但覆盖了所有判断的取真分支和取假分支, 而且覆盖了判断中所有条件的可能取值。但是后一组测试用例虽满足了条件覆盖, 但只覆盖了第一个判断的取假分支和第二个判断的取真分支, 不满足判定覆盖的要求。为解决这一矛盾, 需要对条件和分支兼顾, 有必要考虑以下的判定 - 条件覆盖。

(4) 判定 - 条件覆盖: 就是设计足够的测试用例, 使得判断中每个条件的所有可能取值至少执行一次, 同时每个判断本身的所有可能判断结果至少执行一次。换言之, 即是要求各个判断的所有可能的条件取值组合至少执行一次。例如, 对于本题所给例子中的各判断, 若 T1, T2, T3, T4 及 $\overline{T1}$, $\overline{T2}$, $\overline{T3}$, $\overline{T4}$ 的含意如前所述, 则只需设计以下两个测试用例便可覆盖图中的 8 个条件取值以及 4 个判断分支。

测试用例 (答案)	通过路径	条件取值	覆盖分支
【 (2, 0, 4), (2, 0, 3)】	ace (L1)	T1 T2 T3 T4	c, e
【 (1, 1, 1), (1, 1, 1)】	abd (L2)	$\overline{T1}$ $\overline{T2}$ $\overline{T3}$ $\overline{T4}$	b, d

判定 - 条件覆盖也有缺陷。从表面上来看，它测试了所有条件的取值。但是事实并非如此。因为往往某些条件掩盖了另一些条件。对于条件表达式 $(A > 1) \text{ and } (B = 0)$ 来说，若 $(A > 1)$ 的测试结果为真，则还要测试 $(B = 0)$ ，才能决定表达式的值；而若 $(A > 1)$ 的测试结果为假，可以立刻确定表达式的结果为假。这时，往往就不再测试 $(B = 0)$ 的取值了。因此，条件 $(B = 0)$ 就没有检查。同样，对于条件表达式 $(A = 2) \text{ or } (X > 1)$ 来说，若 $(A = 2)$ 的测试结果为真，就可以立即确定表达式的结果为真。这时，条件 $(X > 1)$ 就没有检查。因此，采用判定 - 条件覆盖，逻辑表达式中的错误不一定能够查得出来。

(5) 条件组合覆盖：就是设计足够的测试用例，运行被测程序，使得每个判断的所有可能的条件取值组合至少执行一次。现在我们仍来考察本题所给出的例子，先对各个判断的条件取值组合加以标记。例如：记

$A > 1, B = 0$ 作 T1 T2, 属第一个判断的取真分支;
 $A > 1, B = 0$ 作 T1 $\overline{T2}$, 属第一个判断的取假分支;
 $A = 1, B = 0$ 作 $\overline{T1}$ T2, 属第一个判断的取假分支;
 $A = 1, B = 0$ 作 $\overline{T1}$ $\overline{T2}$, 属第一个判断的取假分支;
 $A = 2, x > 1$ 作 T3 T4, 属第二个判断的取真分支;
 $A = 2, x = 1$ 作 T3 $\overline{T4}$, 属第二个判断的取真分支;
 $A = 2, x > 1$ 作 $\overline{T3}$ T4, 属第二个判断的取真分支;
 $A = 2, x = 1$ 作 $\overline{T3}$ $\overline{T4}$, 属第二个判断的取假分支;

对于每个判断，要求所有可能的条件取值的组合都必须取到。在图中的每个判断各有两个条件，所以各有 4 个条件取值的组合。我们取 4 个测试用例，就可用以覆盖上面 8 种条件取值的组合。必须明确，这里并未要求第一个判断的 4 个组合与第二个判断的 4 个组合再进行组合。要是那样的话，就需要 $4^2 = 16$ 个测试用例了。

测试用例 (答案)	通过路径	覆盖条件	覆盖组合号
【 (2, 0, 4), (2, 0, 3)】	ace (L1)	T1 T2 T3 T4	,
【 (2, 1, 1), (2, 1, 2)】	abe (L3)	T1 $\overline{T2}$ T3 $\overline{T4}$,
【 (1, 0, 3), (1, 0, 4)】	abe (L3)	$\overline{T1}$ T2 $\overline{T3}$ T4	,
【 (1, 1, 1), (1, 1, 1)】	abd (L2)	$\overline{T1}$ $\overline{T2}$ $\overline{T3}$ $\overline{T4}$,

这组测试用例覆盖了所有条件的可能取值的组合，覆盖了所有判断的可取分支，但路径漏掉了 L4。测试还不完全。

(6) 路径测试：就是设计足够的测试用例，覆盖程序中所有可能的路径。若还是看本题所给出的例子，则可以选择如下的一组测试用例来覆盖该程序段的全部路径。

测试用例 (答案)	通过路径	覆盖条件
【(2, 0, 4), (2, 0, 3)】	ace (L1)	T1 T2 T3 T4
【(1, 1, 1), (1, 1, 1)】	abd (L2)	$\overline{T1}$ $\overline{T2}$ $\overline{T3}$ $\overline{T4}$
【(1, 1, 2), (1, 1, 3)】	abe (L3)	$\overline{T1}$ $\overline{T2}$ $\overline{T3}$ T4
【(3, 0, 3), (3, 0, 1)】	acd (L4)	T1 T2 $\overline{T3}$ $\overline{T4}$

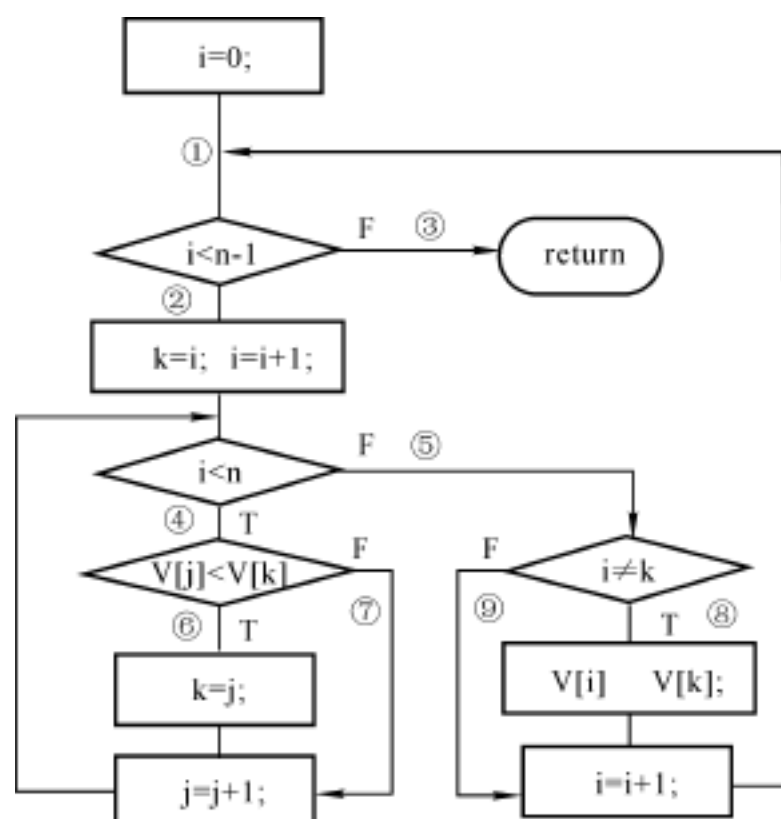
三、分析题

1. 答：对小程序进行穷举测试，不见得能保证程序百分之百正确。所谓穷举测试是拿所有可能的输入数据来作为测试用例（黑盒测试），或覆盖程序中所有可能的路径（白盒测试）。对于小程序来说，实际上并不能真正作到穷举测试。例如前面讲过，一个小程序 P 只有两个输入 X 和 Y 及输出 Z，在字长为 32 位的计算机上运行。如果 X, Y 只取整数，考虑把所有的 X、Y 值都做为测试数据，按黑盒方法进行穷举测试，这样做可能采用的测试数据组 (X_i, Y_i) ，基数 (radix) i 的最大可能数目为： $2^{32} \times 2^{32} = 2^{64}$ 。如果程序 P 测试一组 X, Y 数据需要 1 毫秒，而且假定一天工作 24 小时，一年工作 365 天，要完成 2^{64} 组测试，需要 5 亿年。

2. 答：单元测试又称模块测试，是针对软件设计的最小单位——程序模块，进行正确性检验的测试工作。其目的在于发现各模块内部可能存在的各种差错。单元测试需要从程序的内部结构出发设计测试用例。多个模块可以平行地独立进行单元测试。

单元测试是在编码阶段完成的，每编写出一个程序模块，就开始做这个模块的单元测试，所以只要采用模块化方法开发软件，单元测试都是必需的。它可由编写程序的人来完成。因为它需要根据程序的内部结构设计测试用例，对于那些不了解程序内部细节的人，这种测试无法进行。

3. 答：(1) McCabe 环路复杂性 = 5。



(2) 独立路径有 5 条：

{
...
...
...
...
}

(3) 为各测试路径设计测试用例：

路径 ： 取 $n = 1$
路径 ...： 取 $n = 2$ ，
 预期结果： 路径 不可达
路径 ...： 取 $n = 2$ ，
 预期结果： 路径 不可达
路径 ：
取 $n = 2, V[0] = 2, V[1] = 1$ ，预期结果： $k = 1, V[0] = 1, V[1] = 2$
路径 ：
取 $n = 2, V[0] = 2, V[1] = 1$ ，预期结果： $k = 1$ ，路径 不可达
路径 ：
取 $n = 2, V[0] = 1, V[1] = 2$ ，预期结果： $k = 0$ ，路径 不可达
路径 ：

取 $n = 2, V[0] = 1, V[1] = 2$ ，预期结果： $k = 0, V[0] = 1, V[1] = 2$

4. 答： 设三条线段分别为 A, B, C。如果它们能够构成三角形的三条边，必须满足：

$A > 0, B > 0, C > 0$ ，且 $A + B > C, B + C > A, A + C > B$ 。

如果是等腰的，还要判断是否 $A = B$ ，或 $B = C$ ，或 $A = C$ 。

对于等边的，则需判断是否 $A = B$ ，且 $B = C$ ，且 $A = C$ 。

列出等价类表：

输入条件	有效等价类	无效等价类
是否为三角形的三条边	$(A > 0) (1), (B > 0) (2),$ $(C > 0) (3), (A + B > C), (4)$ $(B + C > A) (5), (A + C > B) (6)$	$A (0 (7), B (0 (8), C (0 (9),$ $A + B (C (10), A + C (B(11),$ $B + C (A (12)$
是否为等腰三角形	$(A = B) (13), (B = C) (14),$ $(A = C) (15)$	$(A (B) \text{ and } (B (C) \text{ and } (A (C) (16)$
是否为等边三角形	$(A = B) \text{ and } (B = C) \text{ and } (A = C)$ (17)	$(A (B) (18), (B (C) (19),$ $(A (C) (20)$

设计测试用例:输入顺序是【A,B,C】

【3,4,5】覆盖等价类 (1), (2), (3), (4), (5), (6)。满足即为一般三角形。

【0,1,2】覆盖等价类 (7)。不能构成三角形

【1,0,2】覆盖等价类 (8)。同上

【1,2,0】覆盖等价类 (9)。同上

若不考虑特定 A, B, C,
三者取一即可

【1,2,3】覆盖等价类 (10)。同上

【1,3,2】覆盖等价类 (11)。同上

【3,1,2】覆盖等价类 (12)。同上

若不考虑特定 A, B, C,
三者取一即可

【3,3,4】覆盖等价类 (1), (2), (3), (4), (5), (6), (13)。

【3,4,4】覆盖等价类 (1), (2), (3), (4), (5), (6), (14)。

【3,4,3】覆盖等价类 (1), (2), (3), (4), (5), (6), (15)。

满足即为等腰
三角形,若不考
虑特定 A, B, C,
三者取一即可

【3,4,5】覆盖等价类 (1), (2), (3), (4), (5), (6), (16)。不是等腰三角形

【3,3,3】覆盖等价类 (1), (2), (3), (4), (5), (6), (17)。是等边三角形

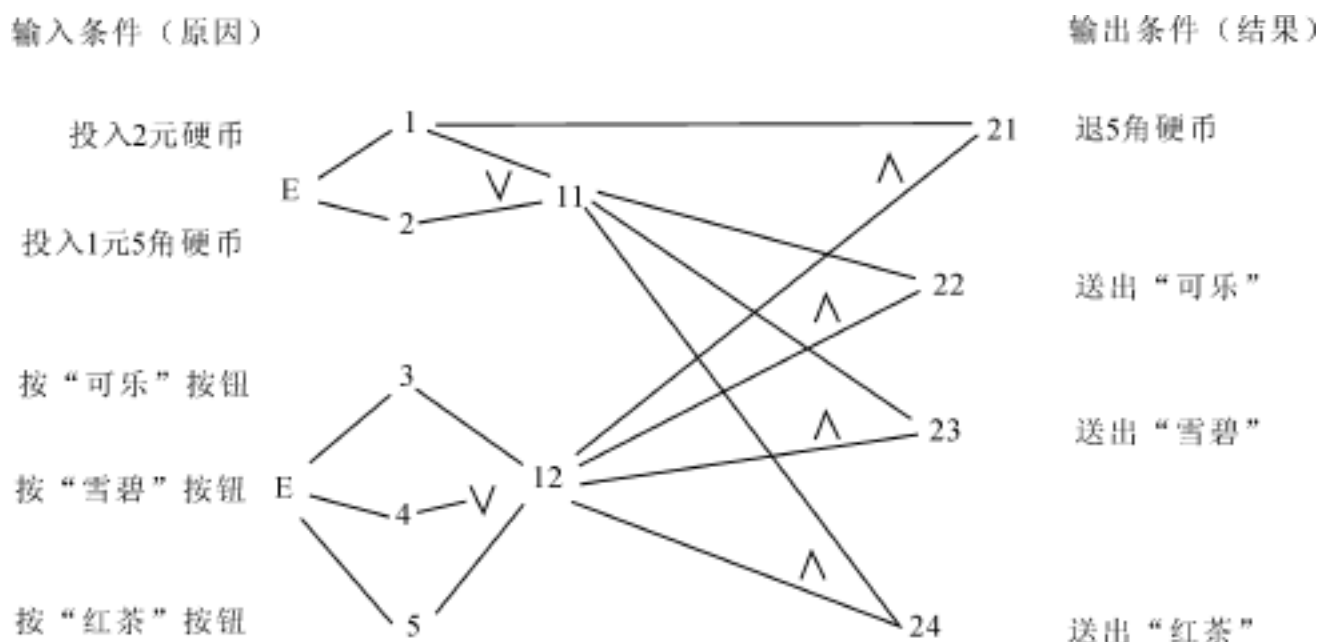
【3,4,4】覆盖等价类 (1), (2), (3), (4), (5), (6), (14), (18)。

【3,4,3】覆盖等价类 (1), (2), (3), (4), (5), (6), (15), (19)。

【3,3,4】覆盖等价类 (1), (2), (3), (4), (5), (6), (13), (20)。

不是等边三角形,若
不考虑特定 A, B, C,
三者取一即可

5. 答:(1) 因果图如下:



(2) 测试用例设计如下:

			1	2	3	4	5	6	7	8	9	10	11
输入	投入 1 元 5 角硬币	(1)	1	1	1	1	0	0	0	0	0	0	0
	投入 2 元硬币	(2)	0	0	0	0	1	1	1	1	0	0	0
	按“ 可乐 ”按钮	(3)	1	0	0	0	1	0	0	0	1	0	0
	按“ 雪碧 ”按钮	(4)	0	1	0	0	0	1	0	0	0	1	0
	按“ 红茶 ”按钮	(5)	0	0	1	0	0	0	1	0	0	0	1
中间 结点	已 投 币	(11)	1	1	1	1	1	1	1	1	0	0	0
	已 按 钮	(12)	1	1	1	0	1	1	1	0	1	1	1
输出	退还 5 角硬币	(21)	0	0	0	0	1	1	1	0	0	0	0
	送出“ 可乐 ”饮料	(22)	1	0	0	0	1	0	0	0	0	0	0
	送出“ 雪碧 ”饮料	(23)	0	1	0	0	0	1	0	0	0	0	0
	送出“ 红茶 ”饮料	(24)	0	0	1	0	0	0	1	0	0	0	0

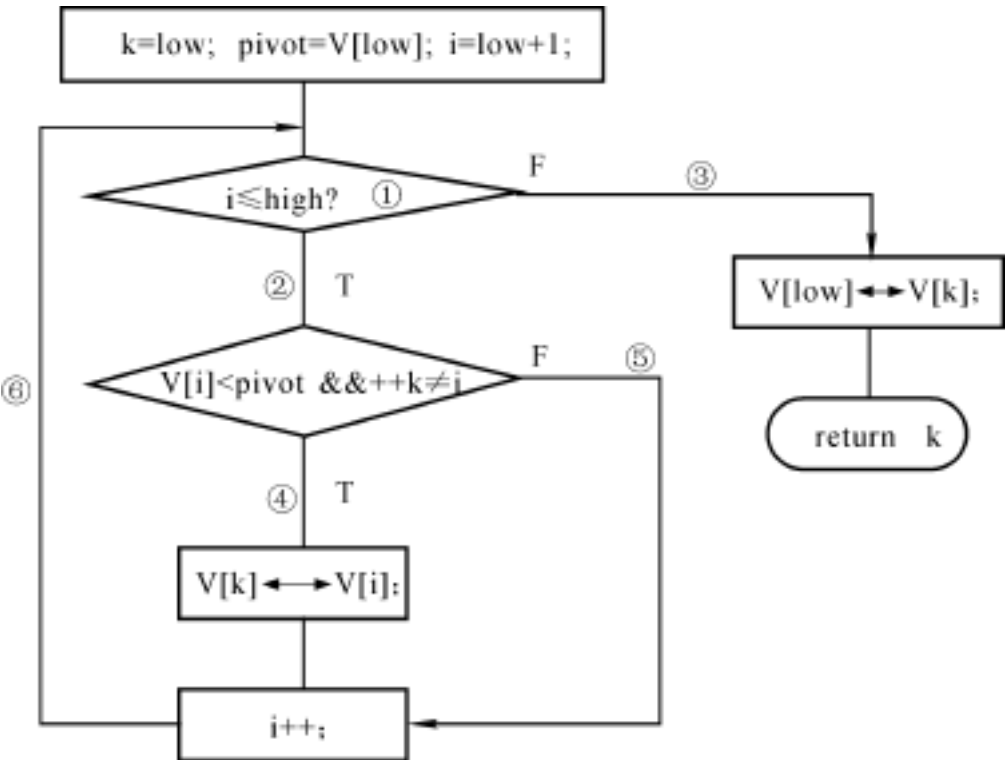
测试用例每一纵列为一个测试用例

6. 答：在对照需求做有效性测试和软件配置审查时，是由软件开发者在开发环境下进行的测试。而接下来做验收测试时则以用户为主。软件开发人员和 QA（质量保证）人员也应参加。由用户参加设计测试用例，使用用户界面输入测试数据，并分析测试的输出结果。一般使用生产中的实际数据进行测试。

如果软件是为多个客户开发的，则需要进行测试和测试。测试是由一个用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。软件在一个自然设置状态下使用。开发者坐在用户旁边，随时记下错误情况和使用中的问题。这是在受控制的环境下进行的测试。

测试是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。这些用户是与公司签定了支持产品预发行合同的外部客户，他们要求使用该产品，并愿意返回有关错位错误信息给开发者。与测试不同的是，开发者通常不在测试现场。因而，测试是在开发者无法控制的环境下进行的软件现场应用。

7. 答：（1）流程图如下：



(2) 测试用例设计

循环 次数	输 入 条 件							输 出 结 果					执 行 路 径
	low	high	k	i	V[0]	V[1]	V[2]	k	i	V[0]	V[1]	V[2]	
0	0	0	0	1	—	—	—	0	1	—	—	—	
1	0	1	0	1	1	2	—	0	2	1	2	—	
	0	1	0	1	2	1	—	1	2	1	2	—	
	0	1	0	1	1	1	—	0	2	1	1	—	
2	0	2	0	1	1	2	3	0	3	1	2	3	
	0	2	0	1	1	2	1	0	3	1	2	1	
	0	2	0	1	2	3	1	1	3	1	2	3	
	0	2	0	1	3	2	1	2	3	1	2	3	
	0	2	0	1	2	1	2	1	3	1	2	2	
	0	2	0	1	2	1	3	1	3	1	2	3	
	0	2	0	1	1	1	2	0	3	1	1	2	
	0	2	0	1	2	2	1	1	3	1	2	2	
	0	2	0	1	2	2	2	0	3	2	2	2	

分析：画程序流程图是设计测试用例的关键。从以往同学解题的经验来看，在画流程图时就出错了。所以首先要把流程图中的逻辑关系搞清楚再画出正确的流程图。考虑测试用例设计需要首先有测试输入数据，还要有预期的输出结果。对于此例，控制循环次数靠循环控制变量 i 和循环终值 $high$ 。循环 0 次时，取 $low = high$ ，此时一次循环也不做。循环一次时，取 $low + 1 = high$ ，循环二次时，取 $low + 2 = high$ 。

根据 BRO 策略, 条件 $V[i] < pivot \ \&\& \ ++k \ i$ 的约束集合为 $\{ (<, <), (<, =), (=, <), (>, <) \}$,

因此，测试用例设计如下：

[illegible]

续 表

循环 次数	输 入 条 件							pivot	输 出 结 果					
	low	high	k	i	V[0]	V[1]	V[2]		k	i	V[0]	V[1]	V[2]	
2	0	2	0	1	1	2	3	1	0 0	2 3	1 1	2 2	3 3	(> , <)(> , <)
	0	2	0	1	1	2	1	1	0 0	2 3	1 1	2 2	1 1	(> , <)(= , <)
	0	2	0	1	2	3	1	2	0 1	2 3	2 2 1	3 1 2	1 3 3	(> , <)(< , <)
2	0	2	0	1	不	可	达							(> , <)(< , =)
	0	2	0	1	3	2	1	3	1 2	2 3	3 3 1	2 2 2	1 1 3	(< , =)(< , =)
	0	2	0	1	2	1	2	2	1 1	2 3	2 2 1	1 1 2	2 2 2	(< , =)(= , <)
	0	2	0	1	2	1	3	3	1 1	2 3	2 2 1	1 1 2	3 3 3	(< , =)(> , <)
	0	2	0	1	不	可	达							(< , =)(< , <)
	0	2	0	1	1	1	2	1	0 0	2 3	1 1	1 1	2 2	(= , <)(> , <)
	0	2	0	1	2	2	1	2	0 1	2 3	2 2 1	2 1 2	1 2 2	(= , <)(< , <)
	0	2	0	1	2	2	2	2	0 0	2 3	2 2	2 2	2 2	(= , <)(= , <)
	0	2	0	1	不	可	达							(= , <)(< , =)
	0	2	0	1	不	可	达							(< , <)(* , *)

8. 答：正确的叙述有（4），（5），（6），（7），（10）。

分析：黑盒测试主要是根据程序的有关功能规格说明和覆盖准则来设计测试用例，进行测试的，不是根据程序的内部逻辑来设计测试用例，这是白盒测试做的事情。在所有黑盒测试方法中，最有效的不是因果图法，而是边界值分析方法。测试的目的是尽可能多地发现软件中的错误，其附带的收获才是验证该软件已正确地实现了用户的要求。测试的一条重要原则是：发现错误多的程序模块，残留在模块中的错误也多。软件可靠性模型（Shooman）就

是依据这个原则建立它的公式的。对于连锁型分支结构，若有 n 个判定语句，则有 2^n 条路径。因此，随着 n 的增大，路径数增长非常快。单元测试时，因为桩模块要模拟子模块的功能，这不是一件容易的事情，而驱动模块只是控制被测模块的执行，所以桩模块的编写比驱动模块的编写要难得多。

在程序设计风格方面，如果重复的代码段没有明显的功能，不可以抽取出来形成独立的公共过程或子程序，只有在这些代码段表现出独立的功能时，才可把它们抽取出来形成独立的公共过程或子程序。另外，程序效率的提高主要应通过选择高效的算法或使用高效的语言编译器来实现。GOTO 语句概念简单，使用方便，在某些情况下，保留 GOTO 语句反能使写出的程序更加简洁。这句话是正确的。

第七章

1. 答：软件维护包括 4 项内容：改正性维护；适应性维护；完善性维护；预防性维护。

2. 答：这是由于对软件维护的概念理解有误造成的，软件维护并非仅仅指改正程序中的错误，软件维护包括了改正性维护；适应性维护；完善性维护；预防性维护。其中为了消除程序中潜藏的错误而进行的改正性维护，仅仅占全部维护活动的 $1/5$ 左右。把 60% 以上的人力用于维护已有的软件，指的是软件开发组织内人力分配的整体状况。

3. 答：首先由于软件维护包括 4 大内容，无论是哪一方面的维护，都必须修改原来的设计和程序代码。修改之前必须深入理解待修改的软件产品，修改之后还应该进行必要的测试，以保证所做的修改是正确的而且没有副作用。如果是改正性维护还必须预先进行调试以确定错误的准确位置。综上所述，软件维护远比一般产品维修要艰巨复杂得多。所以软件维护的工作量和成本自然就很高。

模拟试题（一）

一、填空题

- | | | |
|----------------|---------|---------|
| 1. 硬件、软件、人数、据库 | 2. 巧合内聚 | 3. 逻辑内聚 |
| 4. 通信内聚 | 5. 过程内聚 | 6. 功能内聚 |
| 7. 简单性 | | |

二、单项选择题

1. 2. 3. 4. 5. 6. 7. 8.

三、多项选择题

- A. B. C. D. E. F. G.

注意，C 与 D 的答案顺序可互换。

四、分析题

1. 答：软件危机泛指在计算机软件的开发、维护和使用过程中所遇到的一系列严重问题。

从宏观上说，软件危机主要是指：软件的发展赶不上计算机硬件的发展；软件的发展赶不上社会对于软件需求的增长。

从具体的软件来说，软件危机是指：软件往往不能按计划、按预算、按时完成；已开发的软件不能很好地使用，甚至很快就不用。

产生软件危机的主要原因包括：

- (1) 软件需求分析不充分；
- (2) 软件开发的规范性不够；
- (3) 软件开发计划的科学性不够；
- (4) 缺少对于软件的评测手段。

2. 答：软件开发时，一个错误发现得越晚，为改正它所付出的代价就越大。这个说法是对的。在 1970 年代，GTE，TRW 和 IBM 等三家公司对此问题做了独立研究，最后它们得到相似的结论：

阶 段	需求分析	软件设计	程序编码	单元测试	验收测试	维 护
相对修复代价	0.1 ~ 0.2	0.5	1	2	5	20

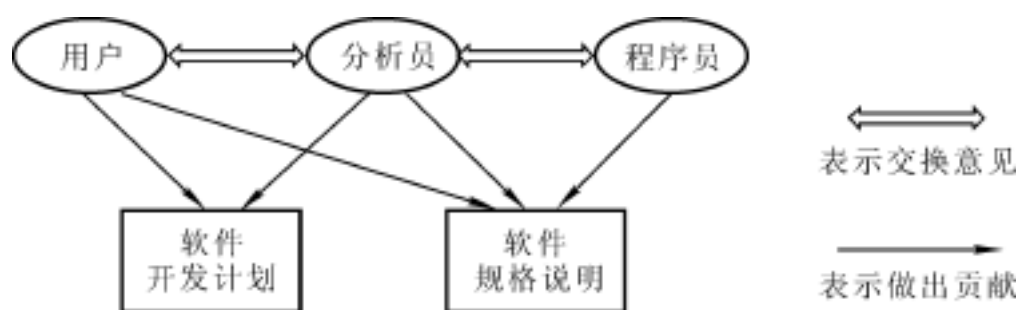
从表中可以看出，在需求分析阶段检查和修复一个错误所需的代价只有编码阶段所需代价的 $1/5$ 到 $1/10$ ，而在维护阶段做同样的工作所付出的代价却是编码阶段的 20 倍。

3. 答：所谓当前系统可能是需要改进的某个已在计算机上运行的数据处理系统，也可能是一个人工的数据处理过程。当前系统的物理模型客观地反映当前系统实际的工作情况。但在物理模型中有许多物理的因素，随着分析工作的深入，有些非本质的物理因素就成为不必要的负担，因而需要对物理模型进行分析，区分出本质的和非本质的因素，去掉那些非本质的因素即可获得反映系统本质的逻辑模型。所以当前系统的逻辑模型是从当前系统的物理模型抽象出来的。

4. 答：软件需求分析过程中，由于最初分析员对要解决的问题了解很少，用户对问题的描述、对目标软件的要求也很凌乱、模糊，再加上分析员和用户共同的知识领域不多，导致相互间通信的需求。首先，由于分析员和用户之间需要通信的内容相当多，业务知识上的不足，表达方式的不足，可能对某些需求存在错误解释或误解的可能性，造成需求的模糊性。其次，用户和分析员之间经常存在无意识的“我们和他们”的界限，不是按工作需要组成统一的精干的队伍，而是各自定义自己的“版图”，并通过一系列备忘录、正式的意见书、文档，以及提问和回答来相互通信。历史已经证明，这样会产生大量误解。忽略重要信息，无法建立成功的工作关系。

5. 答：系统分析员处在用户和高级程序员之间，负责沟通用户和开发人员的认识和见解，起着桥梁的作用。一方面要协助用户对所开发的软件阐明要求，另一方面还要与高级程序员交换意见，探讨用户所提要求的合理性以及实现的可能性。最后还要负责编写软件需求

规格说明和初步的用户手册。



为能胜任上述任务，分析员应当具备的素质如下：

- (1) 能够熟练地掌握计算机硬、软件的专业知识，具有一定的系统开发经验。
- (2) 善于进行抽象的思维和创造性的思维，善于把握抽象的概念，并把它们重新整理成为各种逻辑成分，并给出简明、清晰的描述。
- (3) 善于从相互冲突或混淆的原始资料中抽出恰当的条目来。
- (4) 善于进行调查研究，能够很快学习用户的专业领域知识，理解用户的环境条件。
- (5) 能够倾听他人的意见，注意发挥其他人员的作用。
- (6) 具有良好的书面和口头交流表达能力。

6. 答：如果两个模块互相独立，那么对其中一个模块进行编码、测试或修改时可以完全不考虑另一个模块对它的影响。因此，用模块独立性作为衡量模块结构是否容易编码、容易测试、容易修改的标准是合适的。但是，在一个系统的模块结构中没有哪两个模块可以完全独立，所以，要力争模块之间尽量独立，以得到一个质量良好的模块结构。

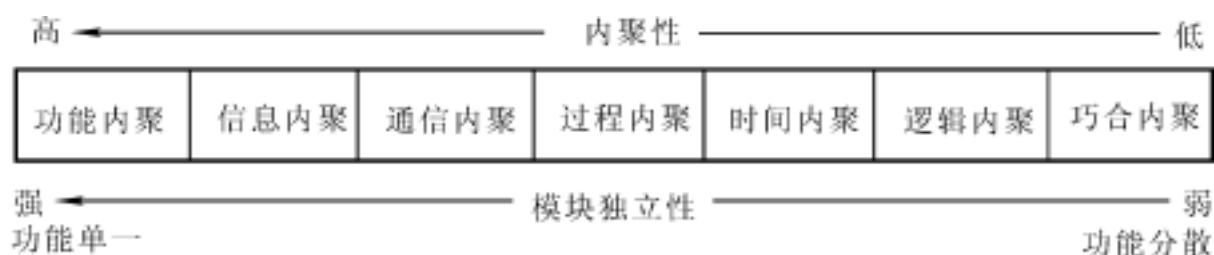
一般采用两个准则度量模块独立性。即模块间的耦合和模块的内聚。模块间的耦合是模块之间的相对独立性（互相连接的紧密程度）的度量。模块之间的连接越紧密，联系越多，耦合性就越高，而其模块独立性就越弱。内聚是模块功能强度（一个模块内部各个成分彼此结合的紧密程度）的度量。一个模块内部各个成分之间的联系越紧密，则它的内聚性就越高，相对地，它与其他模块之间的耦合性就会减低，而模块独立性就越强。因此，模块独立性比较强的模块应是高内聚低耦合的模块。

一般模块之间可能的连接方式有七种，构成耦合性的七种类型。它们之间的关系为



低耦合的情形有非直接耦合、数据耦合和标记耦合，它们都是比较好的模块间的连接。特点是模块间的接口简单、规范。中度耦合的情形有控制耦合，它通过参数表传递控制参数。相对高的耦合情形有外部耦合和公共耦合，它们都是通过全局数据传递模块间的信息，不是说它们一定“坏”，但一定要注意使用这类耦合可能产生的后果，特别要防范这种后果。

一般模块的内聚性分为七种类型，它们的关系如下图所示。



在上面的关系中可以看到，位于高端的几种内聚类型最好，位于中段的几种内聚类型是可以接受的，但位于低端的内聚类型很不好，一般不能使用。因此，人们总是希望一个模块的内聚类型向高的方向靠。模块的内聚在系统的模块化设计中是一个关键的因素。

内聚和耦合是相互关联的。在程序结构中各模块的内聚程度越高，模块间的耦合程度就越低。但这也不是绝对的。我们的目标是力求增加模块的内聚，尽量减少模块间的耦合，但增加内聚比减少耦合更重要，应当把更多的注意力集中到提高模块的内聚程度上来。

模拟试题（二）

一、填空题

1. 开发的、资源的
2. 功能内聚
3. 一个
4. 数据型
5. 标准调用
6. 少
7. 控制范围

二、单项题

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.

三、多项选择题

- A.
- B.
- C.
- D.

四、分析题

1. 答：软件与任何一个事物一样，有它的孕育、诞生、成长、成熟、衰亡的生存过程。这就是软件的生存周期。它主要分为 6 个阶段：软件项目计划、软件需求分析和定义、软件设计、程序编码、软件测试，以及运行维护。

(1) 软件项目计划：在这一步要确定软件工作范围，进行软件风险分析，预计软件开发所需要的资源，建立成本与进度的估算。根据有关成本与进度的限制分析项目的可行性。

(2) 软件需求分析和定义：在这一步详细定义分配给软件的系统元素。可以用以下两种方式中的一种对需求进行分析和定义。一种是正式的信息域分析，可用于建立信息流和信息结构的模型，然后逐渐扩充这些模型成为软件的规格说明。另一种是软件原型化方法，即建立软件原型，并由用户进行评价，从而确定软件需求。

(3) 软件设计：软件的设计过程分两步走。第一步进行概要设计，以结构设计和数据设计开始，建立程序的模块结构，定义接口并建立数据结构。此外，要使用一些设计准则来判断软件的质量。第二步做详细设计，考虑设计每一个模块部件的过程描述。经过评审后，把每一个加细的过程性描述加到设计规格说明中去。

(4) 程序编码：在设计完成之后，用一种适当的程序设计语言或 CASE 工具生成源程序。应当就风格及清晰性对代码进行评审，而且反过来应能直接追溯到详细设计描述。

(5) 软件测试：单元测试检查每一单独的模块部件的功能和性能。组装测试提供了构造软件模块结构的手段，同时测试其功能和接口。确认测试检查所有的需求是否都得到满足。在每一个测试步骤之后，要进行调试，以诊断和纠正软件的故障。

(6) 软件维护：为改正错误，适应环境变化及功能增强而进行的一系列修改活动。与软件维护相关联的那些任务依赖于所要实施的维护的类型。

2. 答：在软件开发过程中必须遵循下列软件工程原则。

(1) 抽象：采用分层次抽象，自顶向下、逐层细化的办法进行功能分解和过程分解，可以由抽象到具体、由复杂到简单，逐步得到问题的解。

(2) 信息隐蔽：遵循信息封装，使用与实现分离的原则，将模块设计成“黑箱”，可以将实现的细节隐藏在模块内部，使用者只能通过模块接口访问模块中封装的数据。

(3) 模块化：按模块划分系统的体系结构，使得各模块间有良好的接口。这样有助于信息隐蔽和抽象，有助于表示复杂的系统。

(4) 局部化：按抽象数据类型思想及问题域中的概念来建立模块，确保模块之间低耦合，模块内部高内聚。这有助于控制解的复杂性。

(5) 确定性：软件开发过程中所有概念的表达应是确定的、无歧义性的、规范的。这有助于人们之间的沟通，保证整个开发工作协调一致。

(6) 一致性：强调软件开发过程的标准化、统一化。包括文档格式的一致，工作流程的一致，内、外部接口的一致，系统规格说明与系统行为的一致等。

(7) 完备性：软件系统不丢失任何重要成分，可以完全实现系统所要求功能。

(8) 可验证性：开发大型的软件系统需要对系统自顶向下、逐层分解。系统分解应遵循系统易于检查、测试、评审的原则，以确保系统的正确性。

3. 答：什么是信息？广义地讲，信息就是消息。宇宙三要素（物质、能量、信息）之一。它是现实世界各种事物在人们头脑中的反映。此外，人们通过科学仪器能够认识到的也是信息。信息的特征为：可识别、可存储、可变换、可处理、可传递、可再生、可压缩、可利用、可共享。我们通常讲的信息域就是对信息的多视角考虑。信息域包含 3 个不同的视图：信息内容和关系、信息流和信息结构。为了完全理解信息域，必须了解每一个视图。

信息结构：它是信息在计算机中的组织形式。一般表示了各种数据和控制对象的内部组织。数据和控制对象是被组织成 n 维表格，还是组织成有层次的树型结构？在结构中信息与其他哪些信息相关？所有信息是在一个信息结构中，还是在几个信息结构中？一个结构中的信息与其他结构中的信息如何联系？这些问题都由信息结构的分析来解决。

信息流：表示数据和控制传递时的变化方式。输入对象首先被变换成中间信息（数据或控制），然后再变换成输出结果信息。沿着变换路径，可能从已有的数据存储（如磁盘文件或内存缓冲区）中引入附加的信息。对数据进行变换是程序中应有的功能或子功能。两个变换功能之间的数据传递就确定了功能间的接口。

所以，没有信息流的系统相当于没有功能的系统，这样的系统的存在是毫无意义的。而没有信息结构的系统是没有信息的系统，这样的系统不是计算机能够处理的系统。

4. 答：所有的需求分析方法都与一组操作性原则相关联：

- (1) 必须理解和表示问题的信息域。
- (2) 必须定义软件将完成的功能。
- (3) 必须表示软件的行为（作为外部事件的结果）。
- (4) 必须对描述信息、功能和行为的模型进行分解，能够以层次方式揭示其细节。
- (5) 分析过程应当从要素信息转向细节的实现。

通过使用这些原则，分析员可以系统地处理问题。首先检查信息域以便更完整地理解目标软件的功能，再使用模型以简洁的方式表达目标软件的功能和行为，并利用自顶向下、逐层分解的手段来降低问题的复杂性。在这些处理过程中，因处理需求带来的逻辑约束和因其他系统元素带来的物理约束需要通过软件要素和视图的实现加以检验和确认。

除此以外，Davis 建议了一组针对“需求工程”的指导性原则：

- (1) 在开始建立分析模型之前应当先理解问题。如果问题没有很好理解就急于求成，常常会产生一个解决错误问题的完美的软件。
- (2) 强力推荐使用原型。这样做可以使用户了解将如何与计算机交互，而人们对软件质量的认知常常是基于对界面“友好性”的切身体会。
- (3) 记录每一个需求的起源和原因。这是建立对用户要求的可追溯性的第一步。
- (4) 使用多个视图，建立系统的数据、功能和行为模型。这样做可帮助分析员从多方面分析和理解问题，减少遗漏，识别可能的不一致之处。
- (5) 给需求赋予优先级。因为过短的时限会减少实现所有软件需求的可能性。因此，对需求排定一个优先次序，标识哪些需求先实现，哪些需求后实现。
- (6) 注意消除歧义性。因为大多数需求都是以自然语言描述，存在叙述的歧义性问题，造成遗漏和误解。采用正式的技术评审是发现和消除歧义性的好方法。

遵循以上原则，就可能开发出较好的软件需求规格说明，为软件设计奠定基础。

5. 答：所谓“模块独立性”是指软件系统中每个模块只涉及软件要求的具体的子功能，而和软件系统中其他的模块的接口是简单的。所谓的“信息隐蔽”是指每个模块的实现细节对于其他模块来说是隐蔽的。也就是说，模块中所包含的信息（包括数据和过程）不允许其他不需要这些信息的模块使用。

如果软件系统做到了信息隐蔽，即定义和实施了对模块的过程细节和局部数据结构的存取限制，那么这些模块相互间的接口就是简单的。这组模块的独立性就比较强。事实上，衡量模块独立性的一个准则就是模块内聚，达到信息隐蔽的模块是信息内聚模块，它是高内聚情形，模块独立性当然很强了。

一个对象的抽象数据类型，就是信息隐蔽的示例。例如，对于栈 stack，可以定义它的操作 makemull（置空栈）、push（进栈）、pop（退栈）、gettop（取栈顶）和 empty（判栈空）。这些操作所依赖的数据结构是什么样的？它们是如何实现的？都被封装在其实现模块中。软件的其他部分可以直接使用这些操作，不必关心它的实现细节。一旦实现栈 stack 的模块里内部过程或局部数据结构发生改变，只要它相关操作的调用形式不变，则软件中其他所有使用这个栈 stack 的部分都可以不修改。这样的模块结构具有很强的模块独立性。

6. 答：软件设计是一个把软件需求变换成软件表示的过程。最初这种表示只是描绘出软件的总的框架，然后进一步细化，在此框架中填入细节，把它加工成在程序细节上非常接近于源程序的软件表示。正因为如此，所以从工程管理的角度来看，软件设计分两步完成。

首先做概要设计，将软件需求转化为数据结构和软件的系统结构。然后是详细设计，即过程设计。通过对结构表示进行细化，得到软件的详细的数据结构和算法。

由于概要设计建立起整个系统的体系结构框架，并给出了系统中的全局数据结构和数据库接口，人机接口，与其它硬、软件的接口。此外还从系统全局的角度，考虑处理方式、运行方式、容错方式、以及系统维护等方面的问题，并给出了度量和评价软件质量的方法，所以它奠定了整个系统实现的基础。没有概要设计，直接考虑程序设计，就不能从全局把握软件系统的结构和质量，实现活动处于一种无序状态，程序结构划分不合理，导致系统处于一种不稳定的状态，稍一做改动就会失败。所以，不能没有概要设计。