《软件工程导论》期末考试复习知识点

第 1 章　软件工程学概述

1.1 软件危机

1.1.1　软件危机的介绍

软件危机(软件萧条、软件困扰)：是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。

软件危机包含下述两方面的问题：

如何开发软件，满足对软件日益增长的需求；

如何维护数量不断膨胀的已有软件。

软件危机的典型表现：

（1）对软件开发成本和进度的估计常常很不准确；

（2）用户对“已完成的”软件系统不满意的现象经常发生；

（3）软件产品的质量往往靠不住；

（4）软件常常是不可维护的；

（5）软件通常没有适当的文档资料；

（6）软件成本在计算机系统总成本中所占的比例逐年上升；

（7）软件开发生产率提高的速度，远远跟不上计算机应用迅速普及深入的趋势 。

1.1.2　产生软件危机的原因

（1）与软件本身的特点有关

（2）与软件开发与维护的方法不正确有关

1.1.3　消除软件危机的途径

对计算机软件有正确的认识。

认识到软件开发是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。

应该推广使用在实践中总结出来的开发软件的成功技术和方法，并继续研究探索。

应该开发和使用更好的软件工具。

总之，为了解决软件危机，既要有技术措施(方法和工具)，又要有必要的组织管理措施。

1.2

1.2.1　软件工程的介绍

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

软件工程的本质特性：

软件工程关注于大型程序的构造

软件工程的中心课题是控制复杂性

软件经常变化

开发软件的效率非常重要

和谐地合作是开发软件的关键

软件必须有效地支持它的用户

在软件工程领域中是由具有一种文化背景的人替具有另一种文化背景的人创造产品

1.2.2 软件工程的基本原理

用分阶段的生命周期计划严格管理

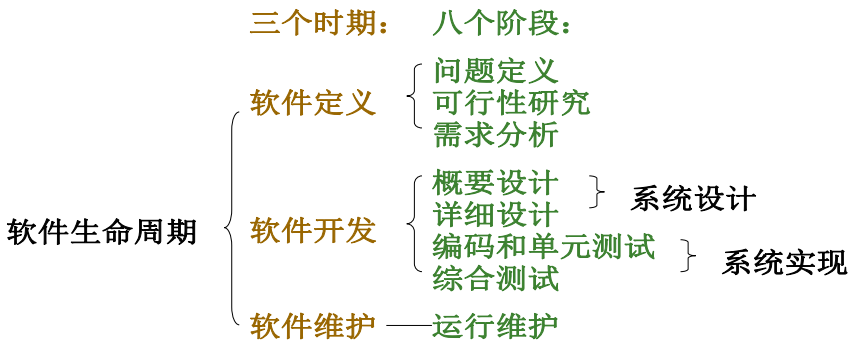
坚持进行阶段评审
实行严格的产品控制
采用现代程序设计技术
结果应能清楚地审查
开发小组的人员应该少而精
承认不断改进软件工程实践的必要性

1.2.3 软件工程方法学

软件工程包括技术和管理两方面的内容。
软件工程方法学 3 要素：方法、工具、过程
1. 传统方法学(生命周期方法学或结构化范型)——强调自顶向下
2. 面向对象方法学——强调主动地多次反复迭代
面向对象方法学 4 个要点：对象、类、继承、消息

1.3 软件生命周期 （必考）

三个时期八个阶段：软件生命周期由软件定义、软件开发和运行维护(也称为软件维护)三个时期组成，每个时期又进一步划分成若干个阶段。



1.4 软件过程

1.4.1 瀑布模型
1.4.2 快速原型模型
1.4.3 增量模型
1.4.4 螺旋模型
1.4.5 喷泉模型

第 2 章 可行性研究

2.1 可行性研究的任务

可行性研究的目的：
不是解决问题，而是确定问题是否值得去解决。
可行性研究的实质：
进行一次大大压缩简化了的系统分析和设计的过程，也就是在较高层次上以较抽象的方式进行的系统分析和设计的过程。
可行性研究的内容：
首先进一步分析和澄清问题定义，导出系统的逻辑模型；

然后从系统逻辑模型出发，探索若干种可供选择的主要解法(即系统实现方案)；
对每种解法都研究它的可行性，至少应该从三方面研究每种解法的可行性 。
主要方面：
技术可行性，经济可行性，操作可行性，
其他方面：
运行可行性， 法律可行性，

2.2 可行性研究过程

1. 复查系统规模和目标
2. 研究目前正在使用的系统
3. 导出新系统的高层逻辑模型
4. 进一步定义问题
5. 导出和评价供选择的解法
6. 推荐行动方针
7. 草拟开发计划
8. 书写文档提交审查

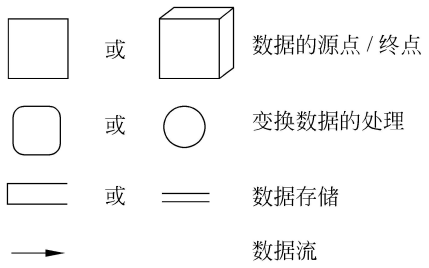
2.3 系统流程图

系统流程图：是概括地描绘物理系统的传统工具。表达的是数据在系统各部件之间流动的情况，而不是对数据进行加工处理的控制过程。

2.4 数据流图

2.4.1 符号

基本符号：



数据存储：数据存储是处于静止状态的数据；
数据流：数据流是处于运动中的数据。
附加符号：
星号（*）：表示“与”关系
加号（+）：表示“或”关系
异或（⊕）：表示互斥关系

2.5 数据字典

数据流图和数据字典共同构成系统的逻辑模型。

2.5.1 数据字典的内容

数据字典的组成：数据流 数据流分量(即数据元素) 数据存储 处理

2.5.2 定义数据的方法

方法：对数据自顶向下分解。

数据组成方式(三种基本类型)：顺序　选择　重复　附加类型：可选
符号：
=意思是等价于(或定义为)；
+意思是和(即，连接两个分量)；
[ ]意思是或(即，从方括弧内列出的若干个分量中选择一个)，通常用“|”号隔开供选择的分量；
{ }意思是重复(即，重复花括弧内的分量)；常常使用上限和下限进一步注释表示重复的花括弧。
( )意思是可选(即，圆括弧里的分量可有可无)。

2.5.3 数据字典的实现

计算机实现　人工实现

2.6 成本/效益分析

2.6.1 成本估计：1. 代码行技术　2. 任务分解技术　3. 自动估计成本技术

2.6.2 成本/效益分析的方法

成本/效益分析涉及的 4 个概念：

1. 货币的时间价值
2. 投资回收期
3. 纯收入
4. 投资回收率：P = F1/(1 + j) + F2/(1 + j)2 + …+ Fn(1 + j)n

第 3 章　需求分析

需求分析的任务：

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。
确定系统必须完成哪些工作，也就是对目标系统提出完整、准确、清晰、具体的要求。
系统分析员应该写出软件需求规格说明书，以书面形式准确地描述软件需求

3.1 需求分析的任务

确定对系统的综合要求
分析系统的数据要求
导出系统的逻辑模型
修正系统开发计划

3.1.1 确定对系统的综合要求

1. 功能需求
2. 性能需求
3. 可靠性和可用性需求
4. 出错处理需求
5. 接口需求
6. 约束
7. 逆向需求
8. 将来可能提出的要求

3.1.2 分析系统的数据要求

建立数据模型——ER 图

描绘数据结构——层次方框图和 Warnier 图
数据结构规范化

3.2　与用户沟通获取需求的方法

访谈：1. 正式访谈 2. 非正式访谈　3. 调查表　 4. 情景分析技术
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：(1) 第四代技术（4GL）(2) 可重用的软件构件 (3) 形式化规格说明和原型环境

3.3 分析建模与规格说明

3.3.1　分析建模

需求分析过程应该建立 3 种模型：数据模型　功能模型　行为模型
数据字典是分析模型的核心
实体-联系图用于建立数据模型的图形
数据流图是建立功能模型的基础
状态转换图是行为建模的基础

3.4 实体-联系图

数据模型中包含 3 种相互关联的信息：数据对象、数据对象的属性、数据对象彼此间相互连接的关系

3.4　状态转换图

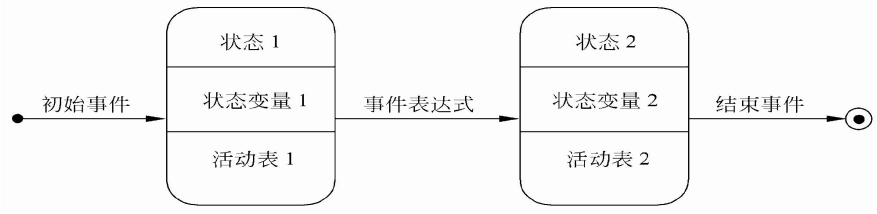
3.6.1 状态

状态图分类：
表示系统循环运行过程，通常不关心循环是怎样启动的。
表示系统单程生命期，需要标明初始状态和最终状态。

3.6.2 事件

事件就是引起系统做动作或(和)转换状态的控制信息。

3.6.3　符号



3.7　其他图形工具

3.7.1　层次方框图

3.7.2 Warnier 图

3.7.3 IPO 图

3.8　验证软件需求（重点）

3.8.1　从哪些方面验证软件需求的正确性

一致性　完整性　　现实性　有效性

第五章　总体设计

5.1 设计过程

由两个主要阶段组成：

系统设计阶段，确定系统的具体实现方案：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　制定测试文档　书写文档　　审查和复查

5.2 设计原理

5.2.1　模块化

模块化的作用：

采用模块化原理可以使软件结构清晰，不仅容易设计也容易阅读和理解。

模块化使软件容易测试和调试，因而有助于提高软件的可靠性。

模块化能够提高软件的可修改性。

模块化也有助于软件开发工程的组织管理。

5.2.2 抽象

5.2.3 逐步求精

5.2.4 信息隐藏和局部化

5.2.5　模块独立

尽量使用数据耦合，

少用控制耦合和特征耦合，

限制公共环境耦合的范围，

完全不用内容耦合。

七种内聚的优劣评分结果：

高内聚：功能内聚

　　顺序内聚

中内聚：通信内聚

　　过程内聚

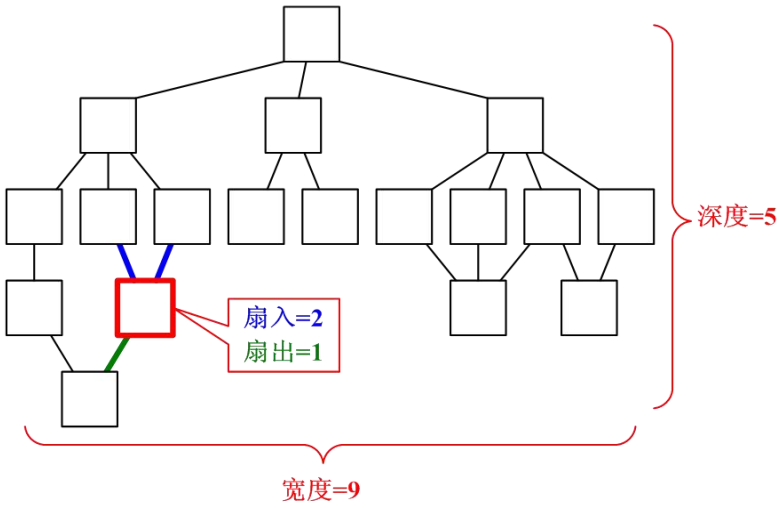
低内聚：时间内聚

　　逻辑内聚

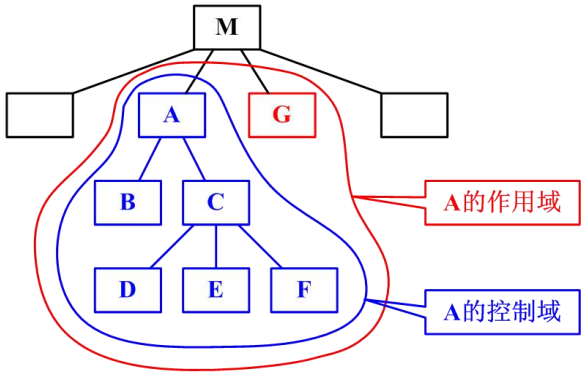
　　偶然内聚

5.3　启发规则

1. 改进软件结构提高模块独立性
2. 模块规模应该适中
3. 深度、宽度、扇出和扇入都应适当



4. 模块的作用域应该在控制域之内



违反规则的情况

5. 力争降低模块接口的复杂程度
6. 设计单入口单出口的模块
7. 模块功能应该可以预测

5.4 描绘软件结构的图形工具

5.4.1 层次图和 HIPO 图

1. 层次图(H 图)

层次图用来描绘软件的层次结构。很适于在自顶向下设计软件的过程中使用。

2. HIPO 图

5.4.2 结构图

5.5 面向数据流的设计方法

结构化设计方法(简称 SD 方法)，也就是基于数据流的设计方法。

5.5.1 概念

面向数据流的设计方法把信息流映射成软件结构，信息流的类型决定了映射的方法。

信息流有两种类型：变换流　　事务流

第 6 章　详细设计

6.1　结构程序设计

经典的结构程序设计：

只允许使用顺序、IF-THEN-ELSE 型分支和 DO-WHILE 型循环这 3 种基本控制结构；

扩展的结构程序设计：

如果除了上述 3 种基本控制结构之外，还允许使用 DO-CASE 型多分支结构和 DO-UNTIL 型循环结构；

修正的结构程序设计：

再加上允许使用 LEAVE(或 BREAK)结构。

6.2 人机界面设计

6.2.1 设计问题

设计人机界面过程中会遇到的 4 个问题：

系统响应时间：长度　易变性

用户帮助设施：集成的帮助设施附加的帮助设施

出错信息处理

命令交互

6.2.3　人机界面设计指南

一般交互指南

信息显示指南

数据输入指南

6.3 过程设计的工具

6.3.1　程序流程图（程序框图）

程序流程图的主要缺点：

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。

6.3.2 盒图(N-S 图)

盒图具有下述特点：

功能域明确。

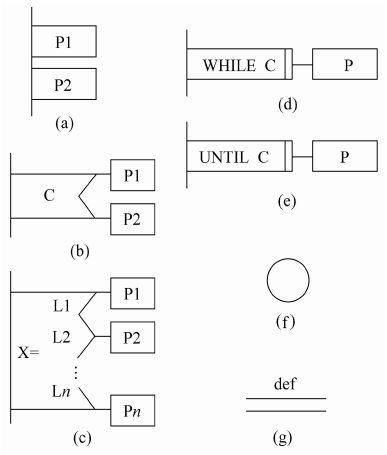
不可能任意转移控制。

很容易确定局部和全程数据的作用域。

很容易表现嵌套关系，也可以表示模块的层次结构。

6.3.3　PAD 图

它用二维树形结构的图来表示程序的控制流，将这种图翻译成程序代码比较容易。



PAD 图的主要优点如下：

使用表示结构化控制结构的 PAD 符号设计出来的程序必然是结构化程序。

PAD 图所描绘的程序结构十分清晰。

PAD 图表现程序逻辑易读、易懂、易记。

容易将 PAD 图转换成高级语言源程序，这种转换可用软件工具自动完成。

即可表示程序逻辑，也可描绘数据结构。

PAD 图的符号支持自顶向下、逐步求精方法的使用。

6.3.4 判定表

判定表却能够清晰地表示复杂的条件组合与应做的动作之间的对应关系。

所有条件	条件组合矩阵
所有动作	条件组合 对应的动作

判定表的缺点：

判定表的含义不是一眼就能看出来的，初次接触这种工具的人理解它需要有一个简短的学习过程。

当数据元素的值多于两个时，判定表的简洁程度也将下降。

6.3.5 判定树

判定树的优点：

它的形式简单，一眼就可以看出其含义，因此易于掌握和使用。

判定树的缺点：

简洁性不如判定表，数据元素的同一个值往往要重复写多遍，而且越接近树的叶端重复次数越多。

画判定树时分枝的次序可能对最终画出的判定树的简洁程度有较大影响。

6.3.6 过程设计语言(伪码）

伪代码的基本控制结构：

简单陈述句结构：避免复合语句。

判定结构：IF_THEN_ELSE 或 CASE_OF 结构。

选择结构：WHILE_DO 或 REPEAT_UNTIL 结构。

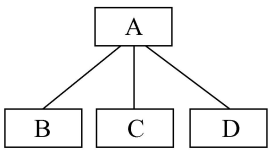
PDL 的优点：

可以作为注释直接插在源程序中间。有助于保持文档和程序的一致性，提高了文档的质量。
可以使用普通的正文编辑程序或文字处理系统，很方便地完成 PDL 的书写和编辑工作。
已经有自动处理程序存在，而且可以自动由 PDL 生成程序代码。
PDL 的缺点：
不如图形工具形象直观，描述复杂的条件组合与动作间的对应关系时，不如判定表清晰简单。

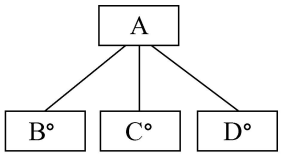
6.4　面向数据结构的设计方法

面向数据结构的设计方法的最终目标是得出对程序处理过程的描述。

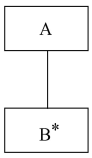
6.4.1Jackson



A 由 B、C、D 3 个元素顺序组成



根据条件 A 是 B 或 C 或 D 中的某一个



A 由 B 出现 N 次(N≥0)组成

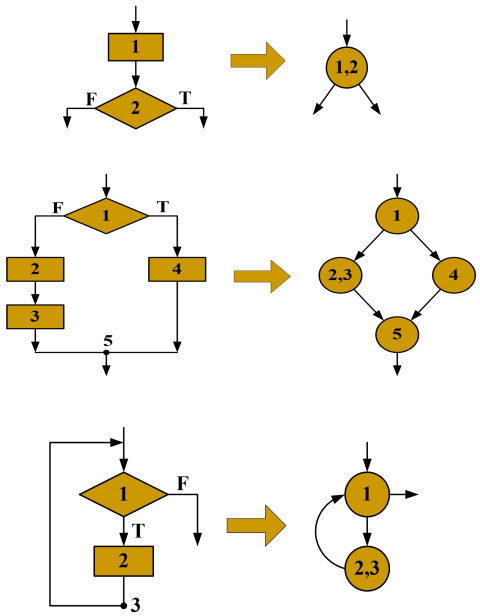
6.4.2　改进的 Jackson 图

6.4.3　Jackson 方法

6.5　程序复杂程度的定量度量

6.5.1　McCabe 方法

1. 流图（程序图）



2. 计算环形复杂度的方法

V(G)=流图中的区域数

V(G)=E-N+2

其中 E 是流图中的边数，N 是结点数

V(G)=P+1

其中 P 是流图中判定结点的数目

6.5.2 Halstead 方法

令 N1 为程序中运算符出现的总次数，N2 为操作数出现的总次数，程序长度 N 定义为：

N=N1+N2

程序中使用的不同运算符(包括关键字)的个数 n1，以及不同操作数(变量和常数)的个数 n2。

预测程序长度的公式如下：

H = n1 log2n1 + n2 log2n2

预测程序中包含错误的个数的公式如下：

E = N log2 (n1+n2)/3000

第 7 章 实现

编码和测试统称为实现。

7.1 编码

7.1.1 选择程序设计语言

主要的实用标准：

系统用户的要求
可以使用的编译程序
可以得到的软件工具
工程规模
程序员的知识
软件可移植性要求
软件的应用领域

7.1.2 编码风格

1. 程序内部的文档:恰当的标识符　适当的注解　程序的视觉组织
2. 数据说明
3. 语句构造
4. 输入输出
5. 效率:程序运行时间　存储器效率　输入输出的效率

7.2 软件测试基础

7.2.1 软件测试的目标

测试是为了发现程序中的错误而执行程序的过程；
好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案；
成功的测试是发现了至今为止尚未发现的错误的测试。

7.2.3 测试方法

黑盒测试(功能测试)：
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；
是在程序接口进行的测试。
白盒测试(结构测试)：
把程序看成装在一个透明的盒子里；
测试者完全知道程序的结构和处理算法；
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。

7.2.4 测试步骤

1. 模块测试(单元测试)

保证每个模块作为一个单元能正确运行；

发现的往往是编码和详细设计的错误。

2. 子系统测试

把经过单元测试的模块放在一起形成一个子系统来测试；

着重测试模块的接口。

3. 系统测试

把经过测试的子系统装配成一个完整的系统来测试；

发现的往往是软件设计中的错误，也可能发现需求说明中的错误；

不论是子系统测试还是系统测试，都兼有检测和组装两重含义，通常称为集成测试。

4. 验收测试(确认测试)

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,而且可能主要使用实际数据(系统将来要处理的信息)进行测试；

发现的往往是系统需求说明书中的错误。

5. 平行运行

7.2.5 测试阶段的信息流

输入信息有两类：

软件配置，包括需求说明书、设计说明书和源程序清单等；

测试配置，包括测试计划和测试方案。

7.3 单元测试

单元测试集中检测模块；

单元测试和编码属于软件过程的同一个阶段；

可以应用人工测试和计算机测试这样两种不同类型的测试方法；

单元测试主要使用白盒测试技术，对多个模块的测试可以并行地进行。

7.3.1 测试重点

模块接口

局部数据结构

重要的执行通路

出错处理通路

边界条件

7.3.2 代码审查

由审查小组正式进行测试称为代码审查；

一次审查会上可以发现许多错误，可以减少系统验证的总工作量。

7.3.3 计算机测试

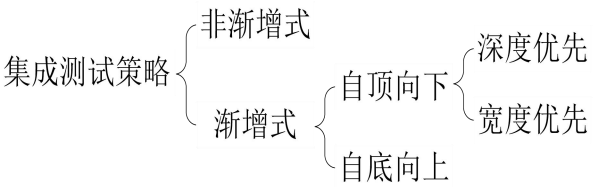
驱动程序是一个“主程序”，它接收测试数据，传送给被测试的模块，并且印出有关的结果。存根程序代替被测试的模块所调用的模块。它使用被它代替的模块的接口，可能做最少量的数据操作，印出对入口的检验或操作结果，并且把控制归还给调用它的模块。

7.4 集成测试

集成测试是测试和组装软件的系统化技术，主要目标是发现与接口有关的问题。

由模块组装成程序时有两种方法：

7.4.3 不同集成测试策略的比较



混合策略：
改进的自顶向下测试方法
混合法

7.4.4　回归测试

7.5　确认测试

确认测试也称为验收测试，它的目标是验证软件的有效性。

7.5.3　Alpha 和 Beta 测试

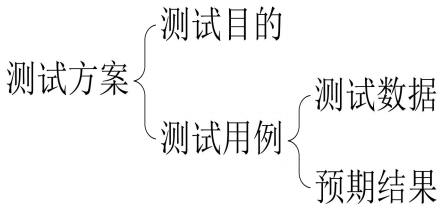
Alpha 测试是在受控的环境中进行的。
Beta 测试是软件在开发者不能控制的环境中的“真实”应用。

1. 接口测试
2. 路径测试
3. 功能测试
4. 健壮性测试
5. 性能测试
6. 用户界面测试
7. 信息安全测试
8. 压力测试
9. 可靠性测试
10. 安装/反安装测试确认测试也称为验收测试，它的目标是验证软件的有效性。

Alpha 测试是在受控的环境中进行的。
Beta 测试是软件在开发者不能控制的环境中的“真实”应用。

4. 接口测试
5. 路径测试
6. 功能测试
4. 健壮性测试
5. 性能测试
6. 用户界面测试
7. 信息安全测试
8. 压力测试
9. 可靠性测试
10. 安装/反安装测试

7.6　白盒测试技术



7.6.1 逻辑覆盖

语句覆盖

判定覆盖 ：比语句覆盖强，但对程序逻辑的覆盖程度仍不高。

条件覆盖 ：判定覆盖不一定包含条件覆盖，条件覆盖也不一定包含判定覆盖。

判定/条件覆盖 ：有时判定/条件覆盖也并不比条件覆盖更强。

条件组合覆盖 ：条件组合覆盖标准的测试数据并不一定能使程序中的每条路径都执行到。

6. 点覆盖（语句覆盖标准相同）

7. 边覆盖（判定覆盖一致）

8. 路径覆盖

7.6.2 控制结构测试覆盖

1. 基本路径测试

基本路径测试是 Tom McCabe 提出的一种白盒测试技术。

首先计算程序的环形复杂度；

以该复杂度为指南定义执行路径的基本集合；

2.条件测试

从该基本集合导出的测试用例可保证程序中的每条语句至少执行一次，而且每个条件在执行时都将分别取真、假两种值。

3. 循环测试

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。

在结构化的程序中通常只有 3 种循环，即简单循环、串接循环和嵌套循环。

7.7 黑盒测试技术

7.7.1 等价划分

7.7.2 边界值分析

7.7.3 错误推测

7.9 软降可靠性

7.9.1 基本概念

软件可靠性：

程序在给定的时间间隔内，按照规格说明书的规定成功地运行的概率。

软件的可用性：

程序在给定的时间点，按照规格说明书的规定，成功地运行的概率。

第 8 章 维护

软件工程的目的是要提高软件的可维护性，减少软件维护所需要的工作量，降低软件系统的总成本。

8.1　软件维护的定义

软件维护：在软件已经交付使用之后，为了改正错误或满足新的需要而修改软件的过程。可分为 4 项活动：

改正性维护
适应性维护
完善性维护
预防性维护

8.2 软件维护的特点

8.2.1　结构化维护与非结构化维护差别巨大
8.2.2　维护的代价高昂
8.2.3　维护的问题很多

8.3 软件维护过程

1.维护组织　2.维护报告 3.维护的事件流 4.保存维护记录 5.评价维护活动

8.4　软件的可维护性

决定软件可维护性的因素主要有 7 个：

可理解性
可测试性
可修改性
可靠性
可移植性
可使用性
效率

第 9 章　面向对象方法学引论

9.1 面向对象方法学概述

9.1.1 面向对象方法学要点

(1)认为客观世界是由各种对象组成的，任何事物都是对象
(2)把所有对象都划分成各种类对象，每个对象类都定义了一组数据和一组方法
(3)按照子类和父类的关系，把若干个对象类组成一个层次结构的系统
(4)对象彼此之间仅能通过传递消息相互联系

9.1.2　面向对象开发方法

面向对象=对象+类 +继承+通信

9.1.4 面向对象方法组成

面向对象的分析
面向对象的设计
面向对象的程序设计

9.1.6　面向对象方法的优点

1.与人类习惯的思维方式一致
2.稳定性好
3.可重用性好
4.可维护性好
5.较易开发大型软件产品

9.2　面向对象的概念

9.2.1 对象

是客观事物或概念的抽象表述，即对客观存在的事物的描述统称为对象，对象可以是事、物、或抽象概念 ，是将一组数据和使用该数据的一组基本操作或过程封装在一起的实体。
对象的特点
(1) 以数据为中心。
(2) 对象是主动的。
(3) 实现了数据封装。
(4) 本质上具有并行性。
(5) 模块独立性好。

9.2.2 类

是一组具有相同属性和相同操作的对象的集合。

9.2.3 实例

由某个特定的类所描述的一个具体的对象。

9.2.4 消息

向对象发出的服务请求（互相联系、协同工作等）。一个消息包含 3 个部分：接收消息的对象，消息名，消息变元

9.2.5 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。

9.2.6 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。

9.2.7 封装

对象封装了对象的数据以及对这些数据的操作。

9.2.8 继承(I)

继承是子类自动地共享基类中定义的数据和方法的机制。
单重继承：子类仅从一个父类继承属性和方法
多重继承：子类可从多个父类继承属性和方法

9.2.9 多态性

9.2.10 重载

9.3 面向对象建模(II)

面向对象开发软件，需要建立 3 种形式的模型。
对象模型。描述系统数据结构—数据结构。
动态模型。描述系统控制结构—执行操作。
功能模型。描述系统功能—数值变化。

9.4 对象模型

9.4.1 类图的基本符号(I)

1. 定义类

2. 定义属性

可见性　　属性名 ：类型　= 缺省值　{性质串}

可见性(visibility)表示该属性对类外的元素是否可见。

分为：

public（+） 公有的，即模型中的任何类都可以访问该属性。

private（-） 私有的，表示不能被别的类访问。

protected（#） 受保护的，表示该属性只能被该类及其子类访问。

如果可见性未申明，表示其可见性不确定。

3. 定义操作

可见性 操作名（参数表)：返回类型{性质串}

9.4.2　表示关系的符号(I)

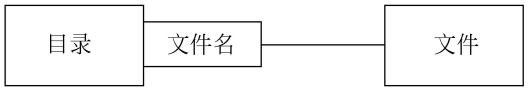
9.4.2.1 关联(I)

关联表示两个类的对象之间存在某种语义上的联系。

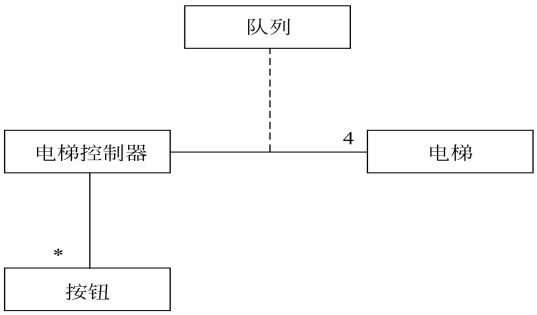
（1） 普通关联

递归关联：一个类与本身有关联关系

（3） 限定关联



（4） 关联类



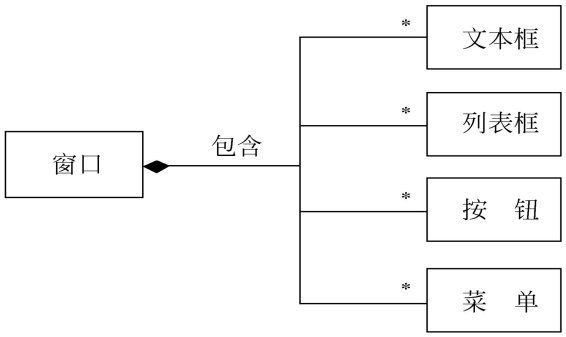
9.4.2.2 聚集(I)

（1） 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。

（2） 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。

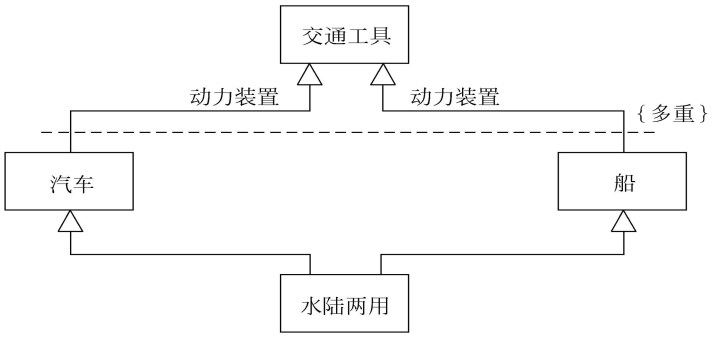


9.4.2.3 泛化(I)

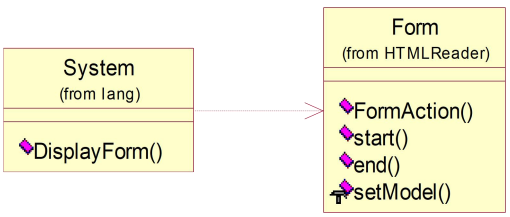
（1）普通泛化

（2）受限泛化

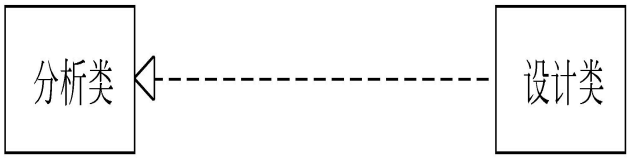
预定义的约束有 4 种： 多重、不相交、完全和不完全。



9.4.2.4 依赖



9.4.2.5 细化



9.5 动态模型

9.6 功能模型

9.6.1 用例图

模型元素：系统、行为者、用例及用例之间的关系(扩展关系、使用关系）
用例的实例是脚本

第 10 章 面向对象分析

10.1 面向对象分析的基本过程

面向对象分析：抽取和整理用户需求并建立问题域精确模型的过程.

理解----用户、分析员和领域专家

表达----需求规格说明书（对象模型、动态模型、功能模型）

验证----二义性，完善性

对象模型最基本、最重要、最核心。

3 个子模型　静态结构（对象模型）
交互次序（动态模型）
数据变换（功能模型）

主题层
类与对象层
结构层
属性层
服务层

复杂问题的对象模型的 5 个层次

面向对象分析的过程

寻找类与对象
识别结构
识别主题
定义属性
建立动态模型
建立功能模型
定义服务

10.2　需求陈述

需求陈述是阐明“做什么”，而不是“怎样做”

问题范围

功能需求
性能需求
应用环境
假设条件

第 11 章 面向对象设计

11.1 面向对象设计的准则

1. 模块化 2. 抽象 3. 信息隐藏
4. 弱耦合

耦合指不同对象之间相互关联的紧密程度。

对象之间的耦合分两类：

交互耦合

如果对象之间的耦合通过消息连接来实现，则这种耦合就是交互耦合。交互耦合应尽可能松散 。

继承耦合

与交互耦合相反，应该提高继承耦合程度。

5. 强内聚

在面向对象设计中存在下述 3 种内聚：

服务内聚：一个服务应该完成一个且仅完成一个功能。

类内聚：一个类应该只有一个用途，它的属性和服务应该是高内聚的。

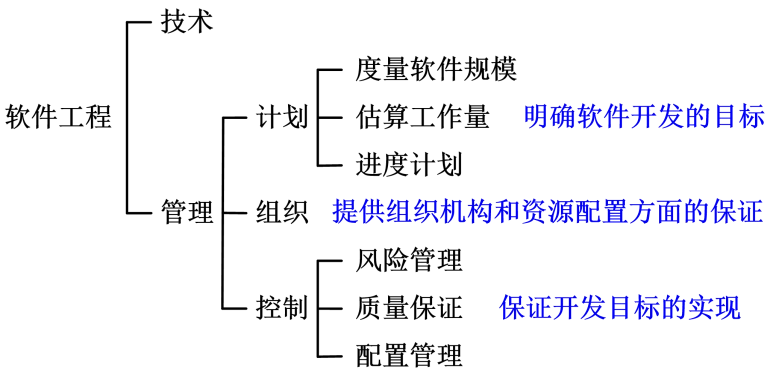
一般-特殊内聚：设计出的一般-特殊结构，应该符合多数人的概念

6. 可重用

11.2 启发规则

1. 设计结果应该清晰易懂
2. 一般-特殊结构的深度应适当
3. 设计简单的类
4. 使用简单的协议
5. 使用简单的服务
6. 把设计变动减至最小

第 13 章 软件项目管理



13.1 估算软件规模

13.1.1 代码行技术

估算方法：

由多名有经验的软件工程师分别做出估计。

每个人都估计程序的最小规模(a)、最大规模(b)和最可能的规模(m)，

分别算出这 3 种规模的平均值之后，再用下式计算程序规模的估计值：

$$L=\frac{\bar{a}+4\bar{m}+\bar{b}}{6}$$

单位：

LOC 或 KLOC。

代码行技术的优点：

代码是所有软件开发项目都有的“产品”，而且很容易计算代码行数；

有大量参考文献和数据 。

代码行技术的缺点：

源程序仅是软件配置的一个成分，由源程序度量软件规模不太合理；

用不同语言实现同一个软件所需要的代码行数并不相同；

不适用于非过程性语言。

13.1.2 功能点技术

功能点技术依据对软件信息域特性和软件复杂性的评估结果，估算软件规模。

这种方法用功能点(FP)为单位度量软件规模。

1. 信息域特性

输入项数(Inp)、输出项数(Out)、查询数(Inq)、主文件数(Maf)、外部接口数(Inf)

每个特征根据其复杂程度分配一个功能点数，即信息域特征系数 a1，a2，a3，a4，a5

2. 估算功能点的步骤

(1) 计算未调整的功能点数 UFP

$$UFP=a1\times Inp+a2\times Out+a3\times Inq+a4\times Maf+a5\times Inf$$

(2) 计算技术复杂性因子 TCF

技术因素对软件规模的综合影响程度 DI：

$$\mathbf{DI}=\sum_{i=1}^{14}F_i$$

技术复杂性因子 TCF 由下式计算：

$$TCF = 0.65 + 0.01 \times DI$$

因为 DI 的值在 0~70 之间，所以 TCF 的值在 0.65~1.35 之间。

(3) 计算功能点数 FP

$$FP = UFP \times TCF$$

功能点技术优点：与所用的编程语言无关，比代码行技术更合理。

功能点技术缺点：在判断信息域特性复杂级别和技术因素的影响程度时主观因素较大，对经验依赖性较强。

13.2 工作量估算

13.2.1 静态单变量模型

E = A + B × (ev) C

ev 是估算变量（KLOC 或 FP）。$E = a \times \mathrm{KLOC}^{b} \times \prod_{i=1}^{17} f_i$

13.2.2 动态多变量模型

动态多变量模型也称为软件方程式，该模型把工作量看作是软件规模和开发时间这两个变量的函数。

E=(LOC×B0.333/P)3×(1/t)4

13.2.3 COCOMO2 模型（构造性成本模型）

3 个层次的估算模型：

应用系统组成模型：这个模型主要用于估算构建原型的工作量，模型名字暗示在构建原型时大量使用已有的构件。

早期设计模型：这个模型适用于体系结构设计阶段。

后体系结构模型：这个模型适用于完成体系结构设计之后的软件开发阶段。

COCOMO2 使用的 5 个分级因素：项目先例性、开发灵活性、风险排除度、项目组凝聚力、过程成熟度

13.3 进度计划

13.3.1 估算开发时间

Brooks 规律：向一个已经延期的项目增加人力，只会使得它更加延期。

13.3.2 Gantt 图

Gantt 图的主要优点：

Gantt 图能很形象地描绘任务分解情况，以及每个子任务(作业)的开始和结束时间。

具有直观简明和容易掌握、容易绘制的优点。

Gantt 图的 3 个主要缺点：

不能显式地描绘各项作业彼此间的依赖关系；

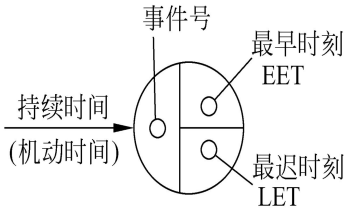
进度计划的关键部分不明确，难于判定哪些部分应当是主攻和主控的对象；

计划中有潜力的部分及潜力的大小不明确，往往造成潜力的浪费。

13.3.3 工程网络

工程网络是系统分析和系统设计的强有力的工具。

13.3.4 估算工程进度



计算最早时刻 EET 使用下述 3 条简单规则：

考虑进入该事件的所有作业；

对于每个作业都计算它的持续时间与起始事件的 EET 之和；

选取上述和数中的最大值作为该事件的最早时刻 EET。

计算最迟时刻 LET 使用下述 3 条规则：

考虑离开该事件的所有作业；
从每个作业的结束事件的最迟时刻中减去该作业的持续时间；
选取上述差数中的最小值作为该事件的最迟时刻 LET。

13.3.5 关键路径

关键事件：EET=LET

13.3.5　机动时间=(LET)结束－(EET)开始－持续时间
=右下角－左上角－持续时间

13.4　人员组织

13.4.1　民主制程序员组

如果小组内有 n 个成员，则可能的通信信道共有 n(n-1)/2 条。

13.4.2　主程序员组

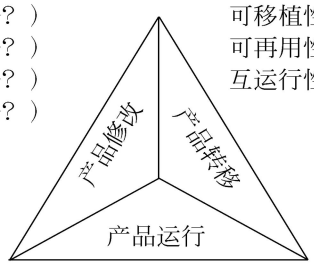
主程序员组的两个重要特性：专业化、 层次性

13.4.3　现代程序员组

13.5 质量保证

13.5.1　软件质量

可理解性（我能理解它吗？）
可维修性（我能修复它吗？）
灵活性　（我能改变它吗？）
可测试性（我能测试它吗？）



可移植性（我能在另一台机器上使用它吗？）
可再用性（我能再用它的某些部分吗？）
互运行性（我能把它和另一个系统结合吗？）

正确性（它按我的需要工作吗？）
健壮性（对意外环境它能适当地响应吗？）
效率　（完成预定功能时它需要的计算机资源多吗？）
完整性（它是安全的吗？）
可用性（我能使用它吗？）
风险　（能按预定计划完成它吗？）

13.5.2　软件质量保证措施

13.6　软件配置管理

基于非执行的测试（复审或评审），主要用来保证在编码之前各阶段产生的文档的质量；
基于执行的测试（软件测试），需要在程序编写出来之后进行，它是保证软件质量的最后一道防线；
程序正确性证明，使用数学方法严格验证程序是否与对它的说明完全一致。

1. 技术复审的必要性
2. 走查:参与者驱动法、文档驱动法
3. 审查：综述　准备　审查　返工　跟踪
4. 程序正确性证明

软件配置管理是在软件的整个生命期内管理变化的一组活动。
具体地说，这组活动用来：①标识变化；②控制变化；③确保适当地实现了变化；④向需要知道这类信息的人报告变化。
软件配置管理的目标:使变化更正确且更容易被适应,在必须变化时减少所需花费的工作量。

13.6.1 软件配置

1.软件过程的输出信息（软件配置项）：

计算机程序（源代码和可执行程序）；

描述计算机程序的文档（供技术人员或用户使用）；

数据（程序内包含的或在程序外的）

2. 基线

基线就是通过了正式复审的软件配置项。

13.6.2 软件配置管理过程

软件配置管理主要有 5 项任务：

1. 标识软件配置中的对象：基本对象、聚集对象
2. 版本控制
3. 变化控制
4. 配置审计
5. 状态报告

13.7 能力成熟度模型

1. 初始级

软件过程的特征是无序的，有时甚至是混乱的。

2. 可重复级

软件机构建立了基本的项目管理过程(过程模型)，可跟踪成本、进度、功能和质量。

3. 已定义级

软件机构已经定义了完整的软件过程（过程模型），软件过程已经文档化和标准化。

4. 已管理级

软件机构对软件过程（过程模型和过程实例）和软件产品都建立了定量的质量目标，所有项目的重要的过程活动都是可度量的。

5. 优化级

软件机构集中精力持续不断地改进软件过程。这一级的软件机构是一个以防止出现缺陷为目标的机构，它有能力识别软件过程要素的薄弱环节，并有足够的手段改进它们。

名词解释

数据词典——是描述数据信息的集合，它对数据流图中的各个元素按规定格式进行详细的描述和确切的解释,是数据流图的补充工具。

1. **数据流图——他以图形的方式反映系统的数据流程**
2. 白盒测试——**按照程序内部的结构测试程序，检验程序中的每条路径是否都能按预定要求正确工作。有两种测试法既逻辑覆盖测试法和路径测试法**
3. 黑盒测试——**按照程序的功能测试程序，检验与程序功能有关的输入、输出与程序执行是否正确。有四种方法既等价分类法、边界值分析法、错误猜测法和因果图法**
4. **完善性维护——**为了适应用户业务和机构的发展变化而对软件的功能、性能进行修改、扩充的过程称为完善性维护。因为各种用户的业务和机构在相当长的时期内不可能是一成不变的，所以功能、性能的增加是不可避免的，而且这种维护活动在整个维护工作中所占的比重很大
5. 软件可靠性——指在给定的时间内，程序按照规定的条件成功地运行的概率
6. 软件配置——是一个软件在生存周期内，他的各种形式、各种版本的文档与程序的总称
7. **软件再工程——运用逆向工程、重构等技术，在充分理解原有软件的基础上，进行分解、综合、并重新构建软件，用于提高软件的可理解性、可维护性可复用性或演化性。**
8. **α 测试——**是在一个受控的环境下，由用户在开发者的“指导”下进行的的测试，由开发者负责记录错误和使用中出现的问题。

9. **β测试——**是由软件的最终用户（多个）在一个或多个用户场所来进行。由用户负责记下遇到的所有问题，包括主观认定的和真实的问题，定期向开发者报告，开发者在综合用户的报告之后进行修改，最后将软件产品交付给全体用户使用。
10. **聚集关系——**表示类或对象之间的整体与部分的关系
11. **泛化关系——**表示类或对象之间的一般与特殊的关系
12. 内聚**——一个模块内部各个元素彼此结合的紧密程度**的度量。
13. 耦合**——**一个软件结构内不同模块之间互连程度的度量。

名词解释：

一章：

软件危机：是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。

产生软件危机的原因：一方面与软件本身的特点有关，另一方面也和软件开发与维护方法不正确有关。

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

软件工程的 7 条基本原理：（1）用分阶段的生命周期计划严格管理;(2) 坚持进行阶段评审;(3) 实行严格的产品控制;(4) 采用现代程序设计技术;(5) 结果可以清楚地审查;(6) 开发小组的人员应该少而精;(7) 承认不断改进软件工程实践的必要性.

软件工程方法学 3 要素：方法、工具、过程

软件过程：是为了获得高质量软件所需要完成的一系列任务的框架，它规定了完成各项任务的工作步骤。

软件生命周期的概念：有软件定义、软件开发和软件维护 3 个小时期组成

软件生命周期 8 个阶段的主要任务：（1）问题定义：“需要解决的问题是什么？”(2) 可行性研究：“寻求可行的解决方案？”(3) 需求分析：“解决这些问题需要系统做什么？”(4) 总体设计（概要设计）：“应该怎样实现目标系统？”(5) 详细设计（模块设计）：“如何具体地实现这个系统？”(6) 编码和单元测试：“写代码，测试每个模块！”(7) 综合测试：“通过各类测试和调试来完善软件”(8) 软件维护：“通过各种必须的维护活动使系统持久满足用户的需要！”

二章：

可行性研究的五个方案：技术可行性，经济可行性，操作可行性，法律可行性，社会效益

可行性研究过程：1.复查系统规模与目标、2.研究目前的系统、3. 导出新系统的高层逻辑模型、4. 进一步定义问题、5. 导出和评价供选择的解法、6. 推荐行动方针、7. 草拟开发计划、8.书写文档提交审查

系统流程图：用来描述物理系统的工具。系统流程图表达的是数据在系统各部件之间流动的情况，而不是对数据进行加工处理的控制过程。即：系统流程图≠程序流程图。

数据流图：用来描述逻辑系统的工具。数据流图(DFD)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，即数据流图描绘数据在软件中流动和被处理的逻辑过程。

三章：

需求分析在软件生命周期中位置：最后一个阶段；**任务：**完整、准确、清晰、具体地确定系统所要完成的工作。

软件系统的综合要求:功能需求,性能需求,可靠性和可用性需求，出错处理需求，接口需求，约束，逆向需求，将来可能提出的要求

获取需求的方法：访谈、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

3 种分析模型：数据模型（ER 图），功能模型（DFD），行为模型（状态转换图）

需求分析阶段的主要图形工具：层次方框图（描绘数据的层次结构）；Warnier 图（描绘数据的层次结构）；IPO 图（IPO 图是输入、处理、输出图的简称）

五章：

总体设计的两个阶段：（1）系统设计阶段（2）结构设计阶段

总体设计的设计原理：模块化，抽象，逐步求精，信息隐藏和局部化，*模块独立（耦合，内聚）

耦合：是对一个软件结构内不同模块之间互连程度的度量；包括：

（1）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

（2）控制耦合——如果传递的信息中有控制信息(尽管有时这种控制信息以数据的形式出现)

（3）特征耦合——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

（4）公共环境耦合——两个或多个模块通过一个公共数据环境相互作用

（5）内容耦合——如果出现下列情况之一，两个模块间就发生了内容耦合

低——高

3．内聚：标志一个模块内各个元素彼此结合的紧密程度；包括：

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

(2)逻辑内聚——如果一个模块完成的任务在逻辑上属于相同或相似的一类。

(3)时间内聚——如果一个模块包含的任务必须在同一段时间内执行。

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

（5)通信内聚——如果模块中所有元素都使用同一个输入数据和(或)产生同一个输出数据。

（6）顺序内聚——如果一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行(通常一个处理元素的输出数据作为下一个处理元素的输入数据)。

（7）功能内聚——如果模块内所有处理元素属于一个整体，完成一个单一的功能。

低——中——高

7 条启发规则：改进软件结构，提高模块独立性，模块规模适中，*深度、宽度、扇出和扇入合理（**深度**表示软件结构中控制的层数；**宽度**是软件结构内同一个层次上的模块总数的最大值；**扇出**：调用其它的模块数（3-4）；**扇入**：被上一级模块调用数（越多越好）），*模块的作用域应在控制域范围内(**模块的作用域定义**为受该模块内一个判定影响的所有模块的集合。**模块的控制域是**这个模块本身以及所有直接或间接从属于它的模块的集合。)，尽量降低模块接口的复杂程度，设计单入口、单出口的模块，模块功能可以预测

六章：

过程设计的工具：程序流程图(程序框图)，盒图(N-S 图)，PAD 图（问题分析图），*判定表，判定树，过程设计语言

面向数据结构的设计方法：（1）Jackson 图（程序中数据元素彼此间的逻辑关系只有三类：顺序结构，选择结构，重复结构）(2) *改进的 Jackson 图(3)* Jackson 方法

程序的处理过程：(1)确定输入和输出数据结构;(2)找有对应关系的输入输出数据单元;(3)从数据结构图导出程序结构图 (均用 Jackson 图表示)；(4)列出所有操作和条件，并且把它们分配到程序结构图的适当位置；(5)用伪码表示程序。

七章：

软件测试的定义或目标：测试是为了证明程序有错，而不是证明程序无错误；一个好的测试用例在于它能发现至今未发现的错误；一个成功的测试是发现了至今未发现的错误的测试。

测试步骤：模块测试，子系统测试，系统测试，验收测试，平行运行

逻辑覆盖：（1）语句覆盖 （2）判定覆盖 （3）条件覆盖(4) 判定－条件覆盖 (5)条件组合覆盖 (6) 路径覆盖（7）点覆盖（8) 边覆盖

黑盒测试步骤：等价划分，边界值分析，错误推测，因果图法

白盒测试（结构测试）技术：分析程序内部一每个分支通路。**过程：**按照程序内部的逻辑测试程序，检测程序中主要执行通路是否按预定要求正确工作。

软件可靠性的定义：程序在给定的时间间隔内，按照规格说明书的规定成功运行的概率。

软件可用性的定义：程序在给定的时间点，按照规格说明书的规定，成功运行的概率。

八章：

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

软件维护的分类：改正性维护（17%~21%），适应性维护(17%~21%)，完善性维护(50%~66%)，预防性维护（4%~5%）

十三章：

项目管理的概念：就是通过计划、组织和控制等一系列活动，合理地配置和使用各种资源，以达到既定目标的过程。

软件工作量的估算：静态单变量模型（E=A+B×(ev)C），动态多变量模型，COCOMO2 模型

制定项目的进度工具：Gantt（甘特）图，工程网络

估算软件规模：（1）**代码行技术**：L = $\frac{a+4m+b}{6}$ 其中 L 为估计的程序规模，a 为程序的最小规模、b 为程序的最大规模、m 为最可能的规模。

优点：代码是所有软件开发项目都有的“产品”，而且很容易计算代码行数。

缺点：源程序仅是软件配置的一个成分，用他的规模代表整个软件的规模似乎不太合理，用不同语言实现同一个软件所需要的代码行数并不相同，这种方法不适用于非过程语言。

三、名词解释题每小题 3 分，共 15 分。

31．软件生存周期模型：是描述软件开发过程中各种活动如何执行的模型。

32．数据字典（DD）：数据字典是用来定义数据流图中的各个成分的具体含义的。它以一种准确的、无二义性的说明方式为系统的分析、设计及维护提供了有关元素的一致的定义和详细的描述。

33．内聚性：内聚性是模块独立性的衡量标准之一，它是指模块的功能强度的度量，即一个模块内部各个元素彼此结合的紧密程度的度量。

34．JSP 方法：JSP 方法是面向数据结构的设计方法，其定义了一组以数据结构为指导的映射过程，它根据输入，输出的数据结构，按一定的规则映射成软件的过程描述，即程序结构。

35．多态性：指相同的操作或函数、过程可作用于多种类型的对象上并获得不同结果。或（不同的对象，收到同一消息可以产生不同的结果。）

1．程序的可维护性：为满足用户新的需求，或当环境发生了变化，或运行中发现了新的错误时，对一个已投入运行的软件进行相应诊断和修改所需工作量的大小。

2．容错技术：对那些无法避开的差错，使其影响减少至最小的技术。也就是说，当错误发生时，尽可能地不影响其它的系统元素，或是把用户的影响限制在某些容许的范围内。

3.结构化维护:如果维护工作是从评价完整的软件配置开始入手，确定软件的重要结构特点、性能特点以及接口特点；估量要求的改动将带来的影响，并且计划实施途径。然后首先修改设计并且对所做的修改进行仔细审查。接下来编写相应的源程序代码；使用在测试说明书中包含的信息进行回归测试；最后，把修改后的软件再次交付使用。

4.软件生存周期是指从提出软件开发要求开始，直到该软件报废不用为止的整个时期。这个时期又分为若干个阶段，对软件生产的管理和进度控制有重要作用，使软件的开发有相应的

模式、流程、工序和步骤。
5.模块独立性:是模块化、抽象和信息隐蔽的直接产物。每个模块只要完成独立的功能，与其它模块联系越少，则模块的独立性就越强。通过模块与模块之间的耦合性和模块内部的内聚性来衡量模块的独立性。
1．数据流图：是描述数据处理过程的工具。它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。
2．软件维护是软件生命周期的最后一个阶段，是在软件已经交付给用户使用之后，为了改正软件错误或满足新的需要而修改软件的过程。它包括四种类型的维护活动：改正型维护、适应型维护、预防型维护和完善型维护。
3．软件测试是一个为了寻找软件错误而运行程序的过程。目的就是为了发现软件中的错误。一个好的测试用例是指很可能找到迄今为止尚未发现的错误的用例。一个成功的测试是指揭示了迄今为止尚未发现的错误的测试。
4．程序的可维护性：为满足用户新的需求，或当环境发生了变化，或运行中发现了新的错误时，对一个已投入运行的软件进行相应诊断和修改所需工作量的大小。
5.软件生存周期:是指从提出软件开发要求开始，直到该软件报废不用为止的整个时期。这个时期又分为若干个阶段，对软件生产的管理和进度控制有重要作用，使软件的开发有相应的模式、流程、工序和步骤。

《软件工程导论》课后习题答案

第一章 软件工程概论

1．什么是软件危机？

软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。这些问题表现在以下几个方面:

(1)用户对开发出的软件很难满意。

(2）软件产品的质量往往靠不住。

（3）一般软件很难维护.

（4)软件生产效率很低。

(5)软件开发成本越来越大。

（6）软件成本与开发进度难以估计。

（7）软件技术的发展远远满足不了计算机应用的普及与深入的需要.

2．为什么会产生软件危机？

（1） 开发人员方面，对软件产品缺乏正确认识，没有真正理解软件产品是一个完整的配置组成.造成开发中制定计划盲目、编程草率，不考虑维护工作的必要性。

(2） 软件本身方面,对于计算机系统来说，软件是逻辑部件,软件开发过程没有统一的、公认的方法论和规范指导,造成软件维护困难.

（3） 尤其是随着软件规模越来越大,复杂程度越来越高，原有软件开发方式效率不高、质量不能保证、成本过高、研制周期不易估计、维护困难等一系列问题更为突出,技术的发展已经远远不能适应社会需求。

3．怎样克服软件危机?

（1） 充分吸收和借鉴人类长期以来从事各种工程项目中积累的行之有效的有效原理、概念、技术与方法，特别是吸取几十年来人类从事计算机硬件研究和开发的经验教训。在开发软件的过程中努力作到良好的组织,严格的管理，相互友好的协作。

（2） 推广在实践中总结出来的开发软件的成功的技术和方法，并研究更好、更有效的技术和方法，尽快克服在计算机系统早期发展阶段形成的一些错误概念和作法。

（3） 根据不同的应用领域,开发更好的软件工具并使用这些工具。将软件开发各个阶段使用的软件工具集合成一个整体,形成一个很好的软件开发支环环境。

总之为了解决软件危机，既要有技术措施(方法和工具)，又要有必要的组织管理措施.

4。构成软件项目的最终产品：

应用程序、系统程序、面向用户的文档资料和面向开发者的文档资料。

5．什么是软件生存周期?

软件生存周期是指从软件定义、开发、使用、维护到淘汰的全过程。

6．软件生存周期为什么划分成阶段?

（1） 任何一个阶段的具体任务不仅独立，而且简单,便于不同人员分工协作,从而降低整个软件开发工作的困难程度。

（2） 可以降低每个阶段任务的复杂程度，简化不同阶段的联系,有利于工程的组织管理,也便于采用良好的技术方法。

（3） 使软件开发的全过程以一种有条不紊的方式进行，保证软件的质量，特别是提高了软件的可维护性.

7．应该怎样来划分阶段？

(1）每一个阶段的任务尽可能独立;

（2)同一阶段内的任务性质尽可能相同；

（3)每一个阶段任务的开始和结束有严格的标准。

8．软件开发模型有几种？它们的开发方法有可特点？

软件开发模型有瀑布型、渐增型和变换型.

瀑布型开发方法是按照软件生存周期的划分依次实施，每一个阶段有明确规定的任务。它的特点:

(1）各个阶段的顺序性和依赖性；

(2）划分逻辑设计与物理设计，尽可能推迟程序的物理实现;

(3)每个阶段必须完成规定的文档，对其中问题通过复审及早发现,及早解决.

渐增型开发方法及特点:

(1） 从部分需求出发，先建立一个不完全的系统，通过测试运行该系统取得经验和信息反馈，加深对软件需求的理解，进一步使系统扩充和完善.如此反复，直至软件人员和用户对所设计完成的软件系统满意为止.

（2） 在渐增型开发下的软件是随软件开发的过程而逐渐形成的。

（3) 渐增型开发方法适合于知识型软件的开发，设计系统时对用户需求的认识开始不是很清楚的，需要在开发过程中不断认识、不断获得新的知识去丰富和完善系统。多数研究性质的试验软件，一般采用此方法。

变换型开发方法及特点:

(1)从软件需求的形式化说明出发，经过一系列的程序变换,得到最终的程序系统。

(2）该方法必须有严格的数学理论和形式化技术的支持。

9.什么是软件工程？

软件工程是指导计算机软件开发和维护的工程学科。

（1) 它采用工程的概念、原理、技术和方法来开发和维护软件;

（2） 它将管理技术与当前经过时间考验的而证明是正确的技术方法结合起来；

（3） 它强调使用生存周期方法学和结构分析和结构技术;

（4） 经过人们长期的努力和探索，围绕着实现软件优质高产这个目标,从技术到管理两个方面做了大量的努力，逐渐形成了”软件工程学”这一新的学科.

10．什么是软件工程环境：

方法与工具的结合，加上配套的软、硬件支持称为软件工程环境。它能支持开发者按照软件工程的方法，全面完成生存周期中的各项任务.

第二章　可行性研究

1。 问题定义的任务和主要工作？

问题定义的任务：将用户提出的要求具体化、定量化;确定研制系统的范围,明确研制的边界.

问题定义阶段的工作：

（1） 通过调查研究，了解系统需求；

（2） 确定系统的功能需求、性能需求、可靠性需求、安全及保密性、资源、开发费用及开发进度等的需求；

(3） 问题定义阶段的产品-—系统目标与范围说明书.

2。 可行性研究目的？

确定在问题定义中所提出的问题是否值得去解,在限制条件下，问题能否解决。

3．可行性研究的任务？

（1） 进一步分析和澄清问题的定义，在澄清问题的基础上,导出系统的逻辑模型；

（2） 从系统逻辑模型中，选择问题的若干种主要解法，研究每一种解法的可行性,为以后的行动提出建议；

(3） 如果问题没有可行的解，建议停止系统开发;如果问题有可行的解，应该推荐一个较好的解决方案,并为工程制定一个初步的计划。

4．可行性研究包括哪几方面的内容？

(1）技术可行性：现有技术能否实现本系统,现有技术人员能否胜任，开发系统的资源能否满足；

(2)经济可行性：经济效益是否超出开发成本；

（3）操作可行性：系统操作在用户内部行得通吗？

(4）法律可行性:新系统开发是否会侵犯他人、集体或国家利益，是否违反国家法律。

5．可行性研究的步骤？

（1）复查系统的规模和目标；

(2)研究目前正在使用的系统，总结现有系统的优劣，提出新系统的雏形；

(3）导出新系统的高层逻辑模型；

（4)推荐建议方案；

(5)推荐行动方针;

（6）书写计划任务书(可行性报告)；

(7）提交审查。

6. 可行性研究报告的主要内容？

可行性分析的结果是可行性研究报告，内容包括：

（1) 系统概述:说明开发的系统名称，提出单位和开发单位。

(2） 可行性研究的前提：系统目标；要求;约束和限制;可行性研究的基本准则等。

（3） 对现有系统的分析：处理流程，图示说明现有系统的处理流程和数据流程；现有系统存在的问题。

（4） 系统需求：主要功能;主要性能及其要求；操作要求;信息要求；限制性要求。

（5) 建议系统：系统目标；处理流程;系统结构，功能，性能;系统技术可行性；投资和效益分析;操作可行性；法律可行性.

（6） 其它可选方案：与国内外同类型方案的比较;提出一两个可行性方案供论证和探讨。

(7） 制定下一阶段的预算。

（8) 结论性意见：由用户方、设计方和投资方共同签署意见.

第三章　需求分析

1. 需求分析的描述工具有哪些?

有数据流图、数据字典、判定表、判定树、结构化自然语言、层次方框图、Warnier 图、IPO 图和需求描述语言等。

2。 需求分析的基本任务是什么?

准确定义未来系统的目标,确定为了满足用户的需要系统必须做什么。

3. 怎样建立目标系统的逻辑模型？要经过哪些步骤？

建立目标系统的逻辑模型的过程也就是数据流图的分解过程.

4. 什么是结构化分析？它的结构化体现在哪里？

结构化分析：使用数据流程图、数据字典、结构化英语、判定表和判定树等工具，来建立一种新的、称为结构化说明书的目标文档—需求规格说明书。

结构化体现在将软件系统抽象为一系列的逻辑加工单元，各单元之间以数据流发生关联。

5. 软件需求规格说明书由哪些部分组成？

组成包括：

(1） 引言:编写目的、背景说明、术语定义及参考资料等。
(2） 概述主要功能、约束条件或特殊需求。
(3） 数据流图与数据字典。
(4) 用户接口、硬件接口及软件接口。
（5） 性能需求、属性等。
(6） 其它需求，如数据库、操作及故障处理等.

6。 为什么数据流图要分层？画分层的 DFD 要遵循哪些原则?

分层的目的：便于逐步细化、结构清晰。
画分层的 DFD 要遵循哪些原则:
（1)父图与子图之间数据要平衡。
(2)分解的深度和层次达到使加工足够简单、易于理解的基本加工为止。
（3）区分局部文件和局部外部项(局限于数据流中某一层或某几层的文件和外部项）.
(4）不要把控制流作为数据流.
（5）忽略琐碎的枝节。
（6)每个数据流要有一个合适的名字，尽量使用现实系统中有具体意义的名字。

7。 系统流程图与数据流程图有什么区别？

系统流程图描述系统物理模型的工具,数据流程图描述系统逻辑模型的工具。

系统流程图从系统功能的角度抽象的描述系统的各个部分及其相互之间信息流动的情况.

数据流程图从数据传送和加工的角度抽象的描述信息在系统中的流动和数据处理的工作状况。

8.数据字典包括哪些内容?它的作用是什么？

数据字典是描述数据流图中数据的信息的集合.它对数据流图上每一个成分：数据项、文件（数据结构)、数据流、数据存储、加工和外部项等给以定义和说明;它主要由数据流描述、加工描述和文件描述三部分组成。对用户来讲,数据字典为他们提供了数据的明确定义;对系统分析员来讲,数据字典帮助他们比较容易修改已建立的系统逻辑模型.

9.描述加工逻辑的工具有哪些?

判定树、判断表和结构化语言等。

第四章 总体设计

1. 系统设计包括哪两个阶段？

系统设计包括总体设计与详细设计两个阶段。

2. 总体设计的主要任务是什么?

总体设计的主要任务是完成软件结构的设计，确定系统的模块及其模块之间的关系。

3。 什么是模块?模块具有哪几个特征？总体设计主要考虑什么特征？

模块是数据说明、可执行语句等程序对象的集合，可以单独命名且可通过名字来访问。

模块具有输入和输出（参数传递）、功能、内部数据结构（局部变量)和程序代码四个特性。

概要设计主要考虑输入、输出（参数传递）和功能两个特性.

4. 什么是模块化？模块设计的准则?

模块化是按规定的原则将一个大型软件划分为一个个较小的、相对独立但又相关的模块。

模块设计的准则：

（1） 改进软件结构， 提高模块独立性：在对初步模块进行合并、分解和移动的分析、精化过程中力求提高模块的内聚，降低藕合。

(2） 模块大小要适中：大约 50 行语句的代码,过大的模块应分解以提高理解性和可维护性；过小的模块，合并到上级模块中。

（3) 软件结构图的深度、宽度、扇入和扇出要适当。一般模块的调用个数不要超过 5 个。

(4） 尽量降低模块接口的复杂程度；

（5） 设计单入口、单出口的模块。

（6) 模块的作用域应在控制域之内。

5。 变换型数据流由哪几部分组成?

变换型结构由三部分组成:传入路径、变换（加工)中心和传出路径。

6。 变换分析设计的步骤？

（1) 区分传入、传出和变换中心三部分，划分 DFD 图的分界线；

（2） 完成第一级分解：建立初始 SC 图的框架；

（3) 完成第二级分解:分解 SC 图的各个分支;

(4) 对初始结构图按照设计准则进行精化与改进。

7。 事务型数据流由哪几部分组成？

事务型结构由至少一条接受路径、一个事务中心与若干条动作路径组成。

8。 事务分析设计的步骤?

（1) 在 DFD 图中确定事务中心、接收部分（包含全部接收路径）和发送部分（包含全部动作路径）;

（2） 画出 SC 图框架,把 DFD 图的三部分分？quot；映射"为事务控制模块，接收模块和动作发送模块。一般得到 SC 图的顶层和第一层（如果第一层简单可以并入顶层）;

（3) 分解和细化接收分支和动作分支,完成初始的 SC 图；

(4) 对初始结构图按照设计准则进行精化与改进。

9。 比较层次方框图与结构图是的异同?

(1） 层次方框图描绘数据的层次结构， 结构图描绘的是软件结构。

（2） 二者都采用多层次矩形框树形结构。层次方框图的顶层矩形框代表完整的数据结构， 下面各层矩形框依次代表上个框数据的子集；结构图 是在层次图的每一个方框内注明模块的名字或主要功能,方框之间的直线表示模块的调用关系，用带注解的箭头表示模块调用过程中传递的信息。

第五章 详细设计

1. 详细设计的目的？

为软件结构图(SC 图或 HC 图）中的每一个模块确定采用的算法和块内数据结构，用某种选定的表达工具给出清晰的描述。

2。 详细设计的主要任务?

编写软件的“详细设计说明书”。软件人员要完成的工作：

(1） 为每一个模块确定采用的算法， 选择某种适当的工具表达算法的过程,写出模块的详细过程描述.

（2) 确定每一模块使用的数据结构。

(3） 确定模块结构的细节，包括对系统外部的接口和用户界面，对系统内部其它模块的接口，以及关于模块输入数据、输出数据及局部数据的全部细节。

（4） 为每一个模块设计出一组测试用例，以便在编码阶段对模块代码（即程序)进行预定的测试。

3. 结构化程序设计的基本原则?

在详细设计中所有模块都使用单入口、单出口的顺序、选择、循环三种基本控制结构。

4。 比较面向数据流和面向数据结构两类设计方法的异同？

相同点：

（1) 遵守结构程序设计“由顶向下”逐步细化的原则，并以其为共同的基础；

（2） 均服从“程序结构必须适应问题结构”的基本原则，各自拥有从问题结构（包括数据结构)导出程序结构的一组映射规则。

不同点:

(1) 面向数据流的设计以数据流图为基础，在分析阶段用 DFD 表示软件的逻辑模型，在设计阶段按数据流类型,将数据流图转换为软件结构。面向数据结构的设计以数据结构为基础，从问题的数据结构出发导出它的程序结构。

(2) 面向数据流的设计的最终目标是软件的最终 SC 图，面向数据结构的设计的最终目标是程序的过程性描述。

5。 比较 Jackson 方法和 LCP 方法的异同？

Jackson 与 LCP 设计方法都是以数据结构为出发点，以程序的过程描述为最终目标,设计步骤基本相似。它们的主要差别是:

（1)使用不同的表达工具，其中 LCP 方法中的表达工具 Warnier 图
比 Jackson 设计方法中的表达工具 Jackson 图有更大的通用性;

（2)Jackson 方法的步骤和指导原则有一定的灵活性，而 LCP 设计
方法则更加严密.

6。 详细设计的描述工具应具备什么功能？

无论哪类描述工具不仅要具有描述设计过程，如控制流程、处理功能、数据组织及其它方面的细节的能力,而且在编码阶段能够直接将它翻译为用程序设计语言书写的源程序.

第六章　编码

1。 编码的任务？

使用选定的程序设计语言，把模块的过程性描述翻译为用语言书写的源程序（源代码)。

2. 对源程序基本要求？

源程序要求：正确可靠、简明清晰、效率高.

(1）源程序的正确性是对程序质量的最基本要求；

（2)源程序的简明清晰，便于验证源代码和模块规格说明的一致性，容易进行测试和维护；

（3)对于大多数模块，编码时应该把简明清晰放在第一位；

（4）除了编码阶段产生源代码外，在测试阶段也需要编写一些测试程序，用于对软件的测试。

3。 程序设计语言的特点?

（1） 名字说明:程序中使用对象的名字，能为编译程序所检查和识别；

（2） 类型说明：定义对象的类型,确定该对象的使用方式；

（3） 初始化:为变量提供适当的初始值或由系统给变量赋一特殊的表明未初始化的值;

(4) 对象的局部性：程序中真正需要的那部分才能访问的对象;

(5） 程序模块:控制程序对象的名字；

(6） 循环控制结构:如 FOR 语句、WHILE—DO 语句、REPEAT—UNTIL 语句等;

（7） 分支控制结构:如 IF 语句、CASE 语句等；

(8） 异常处理:为程序运行过程中发生的错误和意外事件提供检测和处理上的帮助;

(9） 独立编译：能分别编译各个程序单元.

4。 选择程序设计语言需要考虑的因素？

（1） 选择用户熟悉、便于用户维护的语言。

(2） 选择目标系统的环境中可以提供的编译程序所能选用的语言。

（3） 选择可以得到的软件工具,能支持程序开发中可以利用的语言。

(4) 根据工程规模的大小、目标系统应用范围,如实时应用选择 Ada 语言或汇编语言，系统软件开发选择 C 语言或汇编语言,软件开发中若含有大量数据操作则选择 SQL、dBASE 等数据库语言等。

（5） 选择程序员熟悉的语言.

(6) 选择标准化程度高、程序可移植性好的语言。

（7) 根据算法与计算的复杂性、数据结构的复杂性选择。如对于系统程序和结构复杂的应用程序,选择支持数组、记录（或结构）与指针动态数据结构的 Pascal 语言或 C 语言。

(8) 根据实时要求系统需要的响应速度和效率选择相应的语言。

5。 编码风格的指导原则。

（1） 源程序：包括适当的标识符、适当的注解、程序清单的合理布局与清晰；

（2) 数据说明:数据结构或数据类型的说明次序标准化；变量名称尽量有意义;对复杂的数据结构在注解中要说明在程序设计中实现这个数据结构的方法.

（3） 语句的构造简单明了：不要为节省空间将多个语句写在同一行；尽量避免复杂的条件及“非”条件的测试；避免大量使用循环嵌套和条件嵌套；括号的使用是为了使逻辑表达式和算术表达式的运算顺序清晰直观。

（4) 效率:考虑程序运行的时间存储器效率、输入/输出的效率；在处理程序正确性、清晰与效率之间的关系时先求程序正确后求快；先求清楚后求快；保持程序简单以求快；书写清楚，不为“效率”牺牲清晰.

6. 第四代语言（4GL）应具备哪些的特征？

（1) 具有很强的数据管理能力，能对数据库进行有效的存取、查询和其它有关操作;

(2） 能提供一组高效的、非过程化的命令，组成语言的基本语句,编程时用户只需用这些命令说明“做什么”,不必描述实现的细节；

（3) 能满足多功能、一体化的要求.为此，语言中除必须含有控制程序逻辑与实现数据库操作的语句外，还应包括生成与处理报表、表格、图形,以及实现数据运算和分析统计功能的各种语句，共同构成一个一体化的语言,以适应多种应用开发的需要。

第七章　软件测试

1. 软件测试的基本任务?

软件测试是按照特定的规则，发现软件错误的过程；好的测试方案是尽可能发现迄今尚未发现错误的测试;成功的测试方案是发现迄今尚未发现错误的测试;

2. 测试与调试的主要区别?

（1)　测试从一个侧面证明程序员的失败；调试证明程序员的正确;

（2)　测试从已知条件开始,使用预先定义的程序，且有预知的结果，不可预见的仅是程序是否通过测试；调试从不可知内部条件开始，除统计性调试外，结果是不可预见的;

（3）　测试有计划并且要进行测试设计；调试不受时间约束；

（4)　测试是发现错误、改正错误、重新测试的过程;调试是一个推理的过程;

（5）　测试执行是有规程的；调试执行要求程序员进行必要的推理；

(6）　测试由独立的测试组在不了解软件设计的件下完成；调试由了解详细设计的程序员完成;

(7)　大多数测试的执行和设计可由工具支持；调试用的工具主要是调试器。

3。人工复审的方式和作用?

人工复审的方式：代码会审、走查和排练和办公桌检查;

人工复审的作用：检查程序的静态错误.

4。什么是黑盒测试?黑盒测试主要采用的技术有哪些?

黑盒测试：也称为功能测试,它着眼于程序的外部特征，而不考虑程序的内部逻辑结构。测试者把被测程序看成一个黑盒，不用关心程序的内部结构。黑盒测试是在程序接口处进行测试，它只检查程序功能是否能按照规格说明书的规定正常使用，程序是否能适当地接收输入数据产生正确的输出信息，并且保持外部信息（如数据库或文件)的完整性。

黑盒测试主要采用的技术有:等价分类法、边沿值分析法、错误推测法和因果图等技术。

5.什么是白盒测试？白盒测试主要采用的技术有哪些?

测试者了解被测程序的内部结构和处理过程,对程序的所有逻辑路径进行测试，在不同

点检查程序状态,确定实际状态与预期状态是否一致。

白盒测试主要采用的技术有:路径测试技术和事务处理流程技术，对包含有大量逻辑判断或条件组合的程序采用基于逻辑的测试技术.

6. 路径测试技术中几种主要覆盖的含义？举例说明?

语句覆盖：至少执行程序中所有语句一次。

判定覆盖:使被测程序中的每一个分支至少执行一次。故也称为分支覆盖。

条件覆盖:执行所有可能的穿过程序的控制路流程。

条件组合测试:设计足够的测试用例，使每个判定中的所有可能条件取值组合至少执行一次。

7。 等价分类法的测试技术采用的一般方法?举例说明？

（1） 为每个等价类编号;

（2） 设计一个新的测试方案，以尽可能多的覆盖尚未被覆盖的有效等价类，重复这一步骤，直到所有有效等价类被覆盖为止.

（3） 设计一个新的测试方案,使它覆盖一个尚未被覆盖的无效等价类, 重复这一步骤,直到所有无效等价类被覆盖为止。

8. 软件测试的一般步骤？

单元测试、子系统测试、系统测试、验收测试、平行测试。

9. 比较集成试的两种方式的优劣?

非渐增式测试方式:分别测试模块,再把所有模块按设计要求放在一起组成所要的程序。该方法编写测试软件工作量大，模块间的接口错误发现得晚,错误定位较难诊断,总体测试有的错误容易漏掉，测试时间相对较少，可以并行测试所有模块,能充分利用人力,加快工程进度。。

渐增式测试方式: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。该方法利用已测试过的模块作测试软件，开销小,较早发现模块间的接口错误,错误定位往往和最近入的模块相关，对已测试好的模块可在新加入模块的条件下受到新的检验，测试更彻底，需要较多的测试时间,不能并行测试。

总的来说，渐增式测试方法比较好。

10.软件测试的策略？

(1） 在任何情况下都应使用边界值分析的方法。

(2) 必要时用等价类划分法补充测试方案.

(3） 必要时再用错误推测法补充测试方案。

(4) 对照程序逻辑，检查已设计出的测试方案。

(5） 根据对程序可靠性的要求采用不同的逻辑覆盖标准，再补充一些测试方案.

第八章 软件维护

1. 为什么说软件的维护是不可避免的？

因为软件的开发过程中，一般很难检测到所有的错误,其次软件在应用过程中需要随用户新的要求或运行环境的变化而进行软件的修改或完成功能的增删等,为了提高软件的应用水平和使用寿命，软件的维护是不可避免的。

2。 软件的维护一般分为哪几类？

改正性维护:满足用户对已开发产品的性能与运行环境不断提高的要求，进而达到延长软件寿命的目的.

适应性维护：对程序使用期间发现的程序错误进行诊断和改正的过程,配合变化了的环境进行修改软件的活动;

完善性维护：满足用户在使用过程中提出增加新的功能或修改已有功能的建议而进行的工作；

预防性维护：为了改善未来的可维护性或可靠性而修改软件的工作.

3。 影响软件维护的因素有哪些?

开发方法：采用模块化详细设计文档有助于理解软件的结构、界面功能和内部流程；开发过程中严格而科学的管理规划及清晰可靠的文档资料对发生错误后的理解与纠错是至关重要的；开发过程中模块的独立程度越高，对软件修改越容易，对软件的改进和移植越方便.

开发条件：软件开发及维护人员的水平决定了软件开发的质量和维护的效率；开发过程中使用标准的程序设计语言和标准的操作系统接口，可以大大提高软件的可维护性；在测试过程中用例的有效性,可极大地减少软件存在的错误；其次使用规范化的文档资料可为维护提供更好的依据。

4。 软件维护困难主要表现在什么方面？

（1） 一般来讲，维护人员对开发人员写的程序及文档，理解都比较困难，对维护工作不会喜欢;

（2） 维护持续时间都很长，在开发人员不在现场的轻快下，维护软件通常是很困难的；

(3） 绝大多数软件在设计时对将来的软件修改都没有考虑或考虑不多，尤其未能在设计中强调并认真解决好模块的独立性,使软件的修改既困难又易发生差错。

5。 决定软件可维护性的因素？

（1) 软件的可理解性、可测试性、可修改性；

（2) 文档描述符合要求、用户文档简洁明确、系统文档完整并且标准。

6. 软件价格应该计入维护成本吗？为什么？

在软件的生命周期中，软件维护的工作量非常大,不同应用领域的维护成本差别也很大。一般大型软件的维护成本远远高于开发成本若干倍。因此软件价格中应该计入维护成本。

第九章 软件工程管理

1。 软件工程管理的内容？

(1） 费用管理: 对软件开发进行成本核算，使软件生产按照商品生产的规律办事.包括:以简单、科学方法估算软件开发费用，作为签定开发合同的根据；管理开发费用的有效使用,即用经济手段来保证产品如期按质完成。

(2) 质量管理: 按项目的质量保证计划,确保各个开发阶段的开发和维护工作全部按软件工程的规范进行,保证软件产品的质量。

(3） 配置管理:通过对于程序、文档和数据的各种版本所进行的管理,保证资料的完整性

与一致性。

（4) 项目管理：制定《项目实施计划》，按照计划的内容组织和实施软件的工程化生产。最终目标是以合理的费用和进度,圆满完成计划所规定的软件项目。

2。 软件项目有哪些特点？

(1） 软件项目与其他任何产业项目不同,它是算法、思想、概念、组织、流程、效率、优化等的融合体；

(2) 开发软件项目产品，在多数情况下,用户给不出明确的想法和要求。

（3） 在开发过程中，程序及其相关的文档资料常常需要修改,在修改过程中又可能带来新的问题,且这些问题要在很久以后才会发现.

(4) 在研制开发过程中，文档资料是不可缺少的，但工作量又是巨大的，往往也是人们不愿去作的。

(5) 参加软件项目的工作人员,要求具有一定的业务水平和实际工作经验，而很难完全避免的人员流动，对工作的影响是很大的。离开的人员不仅带走了重要的信息，而且带走了工作经验.

3. 软件成本估算的一般方法？

自顶向下估计：首先估算出项目总的开发成本，然后在项目内部进行成本分配。由少数专家参与，依靠他们过去的经验，将要开发的软件与过去开发过的软件进行”类比”，以估计新的软件开发所需要的工作量和成本。

自底向上估计：将开发任务分成若干子任务,子任务又分成子子任务,直到每一个单元内容足够明确为止；把各个任务单元的成本估计出来，汇合成项目的总成本.该方法得到的结果比较接近实际。

4. 为什么在软件开发中，不能用简单增加人员的方法来缩短开发时间?

大量软件开发实践说明:向一个已经延迟的项目追加开发人员,可能使它完成得更晚。因为当开发人员以算术级数增长时，而人员之间的通信将以几何级数增长,往往"得不偿失"。

5。 影响软件质量的主要因素有哪些?

(1) 产品运行:正确性、风险性、效率、完整性、健壮性和可用性;

(2） 产品修改:可理解性、可维护性、灵活性、可测试性；

（3) 产品转移:可移植性、可重用性和互运行性.