注：若因本提纲导致考试丢分，概不负责。

注：加红表示书中有原题，可能会考大题。

注：加蓝表示难以概括，建议直接看书。

## 第一章：软件工程学概述

**1.1 软件危机**

**1.1.1 软件危机的介绍**

**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。它包括两方面：如何开发软件，以满足对软件日益增长的需求；如何维护数量不断增长的已有软件。

**软件危机的典型表现**

对软件开发成本和进度的估计常常很不准确；

用户对已完成的软件系统不满意的现象经常发生；

软件产品的质量往往靠不住；

软件常常是不可维护的；

软件通常没有适当的文档资料；

软件成本在计算机系统中总成本所占比的比例逐年上升；

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 产生软件危机原因**

**产生软件危机原因**

与软件本身的特点有关：是逻辑部件，缺乏可见性；规模庞大、复杂、修改、维护困难。

软件开发与维护的方法不正确：忽略需求分析；认为软件开发等于程序编写；轻视软件维护。

供求矛盾将是一个永恒的主题：面对日益增长的软件需求，人们显得力不从心。

**1.1.3 消除软件危机的途径**

**消除软件危机的途径**

对计算机软件有一个正确的认识（软件≠程序）；

必须充分认识到软件开发不是某种个体劳动的神秘技巧，而应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目；

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；

开发和使用更好的软件工具。

**什么是程序，文档，数据**

软件是程序，数据及相关文档的完整集合；

程序：是能够完成预定功能和性能的可执行的指令序列；

数据：是使程序能够适当地处理信息的数据结构；

文档：是开发、使用和维护程序所需要的图文资料。

**1.2 软件工程**

**1.2.1 软件工程的介绍**

**什么是软件工程**

1993年IEEE的定义：软件工程是：把系统的、规范的、可度量的途径应用于软件开发、运行和维护过程，也就是把工程应用于软件；研究前者提到的途径。

**软件工程本质特性**

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；

在软件工程领域中通常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**

**软件工程基本原理**

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**

**软件工程方法学**

软件工程方法学包含3个要素：方法、工具和过程。

方法是完成软件开发的各项任务的技术方法，回答“怎样做”的问题；

工具是为运用方法而提供的自动的或半自动的软件工程支撑环境；

过程是为了获得高质量的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。

目前使用的最广泛的软件方法工程学，分别是传统方法学和面向对象方法学。

传统方法学：也称为生命周期方法学或结构化范型。优点：把软件生命周期划分成基干个阶段，每个阶段的任务相对独立，而且比较简单，便于不同人员分工协作，从而降低了整个软件开发过程的困难程度。缺点：当软件规模庞大时，或者对软件的需求是模糊的或会承受时间而变化的时候，发开出的软件往往不成功；而且维护起来仍然很困难。

面向对象方法学：优点：降低了软件产品的复杂性；提高了软件的可理解性；简化了软件的开发和维护工作；促进了软件重用。

**1.3 软件生命周期**

**什么是软件生命周期**

软件生命周期模型是跨越整个生存期的系统开发、运作和维护所实施的全部过程、活动和任务的结构框架。

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）3个时期组成。

软件定义划分为三个阶段：问题定义；可行性研究；需求分析。

软件开发划分为四个阶段：总体设计；详细设计；编码和单元测试；综合测试。

运行维护划分为一个阶段：软件维护。

**1.4 软件过程**

**什么是软件过程**

软件过程是为了开发出高质量的软件产品所需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。

**1.4.1 瀑布模型**

**瀑布模型的特点，优缺点**

特点：阶段间具有顺序性和依赖性；推迟实现的观点；质量保证的观点。

优点：它提供了一个模板，这个模板使得分析、设计、编码、测试和支持的方法可以在该模板下有一个共同的指导。虽然有不少缺陷但比在软件开发中随意的状态要好得多。

缺点：实际的项目大部分情况难以按照该模型给出的顺序进行，而且这种模型的迭代是间接的，这很容易由微小的变化而造成大的混乱。

**1.4.2 快速模型原型**

**快速模型原型的优缺点**

优点：使用户能够感受到实际的系统，使开发者能够快速地构造出系统的框架。

缺点：产品的先天性不足，因为开发者常常需要做实现上的折中，可能采用不合适的操作系统或程序设计语言，以使原型能够尽快工作。

**1.4.3 增量模型**

**增量模型优缺点**

优点：人员分配灵活，刚开始不用投入大量人力资源，当核心产品很受欢迎时，可增加人力实现下一个增量；当配备的人员不能在设定的期限内完成产品时，它提供了一种先推出核心产品的途径，这样就可以先发布部分功能给客户，对客户起到镇静剂的作用。

缺点：至始至终开发者和客户纠缠在一起，直到完全版本出来；适合于软件需求不明确、设计方案有一定风险的软件项目。

## 第二章：可行性研究

**2.1 可行性研究的任务**

**了解可行性研究任务**

可行性研究的目的是确定问题是否值得去解决

可行性研究最根本的任务是对以后的行动方针提出建议。

首先需要分析和澄清问题定义

在澄清了问题定义之后，分析员应该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)。对每种解法都应该仔细研究它的可行性。

技术可行性：使用现有的技术能实现这个系统吗?

经济可行性：这个系统的经济效益能超过它的开发成本吗?

操作可行性：系统的操作方式在这个用户组织内行得通吗?

必要时还应该从法律、社会效益等更广泛的方面研究每种解法的可能性。

**可行性研究的目的**

用最小的代价在尽可能短的时间内确定问题是否能够解决。

**2.2 可行性研究过程**

**可行性研究的步骤**

复查系统规模和目标；

研究目前正在使用的系统；

导出新系统的高层逻辑模型；

进一步定义问题；

导出和评价供选择的解法；

推荐行动方针；

草拟开发计划；

书写文档提交审查。

**2.4 数据流图**

**了解数据流图概念**

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。

在数据流图中没有任何具体的物理部件，它只是描绘数据在软件中流动和被处理的逻辑过程。

数据流图是系统逻辑功能的图形表示。

**了解数据流图的符号**

设计数据流图时只需考虑系统必须完成的基本逻辑功能，不需要考虑具体实现。

（看书P31页）所示，数据流图有四种基本符号：

正方形(或立方体)表示数据的源点或终点；

圆角矩形(或圆形)代表变换数据的处理；

开口矩形(或两条平行横线)代表数据存储；

箭头表示数据流，即特定数据的流动方向。

注意，数据流与程序流程图中用箭头表示的控制流有本质不同，千万不要混淆。在数据流图中应该描绘所有可能的数据流向，而不应该描绘出现某个数据流的条件。

**了解数据流图的用途**

画数据流图的基本目的是利用它作为交流信息的工具。

数据流图应该分层，并且在把功能级数据流图细化后得到的处理超过9个时，应该采用画分图的办法，也就是把每个主要功能都细化为一张数据流分图，而原有的功能级数据流图用来描绘系统的整体逻辑概貌。

数据流图的另一个主要用途是作为分析和设计的工具。

当用数据流图辅助物理系统的设计时，以图中不同处理的定时要求为指南，能够在数据流图上画出许多组自动化边界，每组自动化边界可能意味着一个不同的物理系统，因此可以根据系统的逻辑模型考虑系统的物理实现。

**2.5 数据字典**

**了解数据字典**

字典的用途是供人查阅对不了解的条目的解释，数据字典的作用也正是在软件分析和设计的过程中给人提供关于数据的描述信息。

数据字典是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义的集合。

数据流图和数据字典共同构成系统的逻辑模型。

数据字典是对所有与系统相关的数据元素（数据存储、数据流、数据项）的一个有组织的列表和精确的、严格的定义，使得用户和系统分析员对输入、输出、存储成分和[甚至]中间计算有共同的理解。

数据字典要求：完整性；一致性；可用性。

**数据流图与数据字典的对比**

数据流图只能给出

系统逻辑功能的一个总体框架

而缺乏详细、具体的内容。

数据字典是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义和解释的文字的集合。对数据流图的各种成分起注释、说明的作用，给这些成分赋以实际的内容。还对系统分析中其他需要说明的问题进行定义和说明。

**了解数据字典的内容**

数据字典包含下列4类元素的定义：数据流；数据流分量(即数据元素)；数据存储；处理。

除了数据定义之外，数据字典中还应该包含关于数据的一些其他信息：一般信息；定义；使用特点；控制信息；分组信息。

尽量减少出现别名：

对于同样的数据，不同的用户使用了不同的名字；

一个分析员在不同时期对同一个数据使用了不同的名字；

两个分析员分别分析同一个数据流时，使用了不同的名字。

**定义数据的方法**

数据字典中的定义就是对数据自顶向下的分解。

当分解到不需要进一步定义，每个和工程有关的人也都清楚其含义的元素时，这种分解过程就完成了。

由数据元素组成数据的方式只有下述三种基本类型：

顺序，即以确定次序连接两个或多个分量；

选择，即从两个或多个可能的元素中选取一个；

重复，即把指定的分量重复零次或多次；

可选，即一个分量是可有可无的(重复零次或一次)。

**了解数据字典的符号**

=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)；

()意思是可选(即，圆括弧里的分量可有可无)。

**数据字典的用途**

数据字典最重要的用途是作为分析阶段的工具。

在数据字典中建立的一组严密一致的定义很有助于改进分析员和用户之间的通信，因此将消除许多可能的误解。

对数据的这一系列严密一致的定义也有助于改进在不同的开发人员或不同的开发小组之间的通信。

数据字典是开发数据库的第一步，而且是很有价值的一步。

**数据字典的实现**

目前，数据字典几乎总是作为CASE“结构化分析与设计工具”的一部分实现的。

如果在开发小型软件系统时暂时没有数据字典处理程序，建议采用卡片形式书写数据字典，每张卡片上保存描述一个数据的信息。

每张卡片上主要应该包含下述这样一些信息：名字、别名、描述、定义、位置。

**了解数据字典的部分概念**

数据项：数据的最小单位，描述数据的静态特性；

数据流：由一个或一组固定的数据项组成；

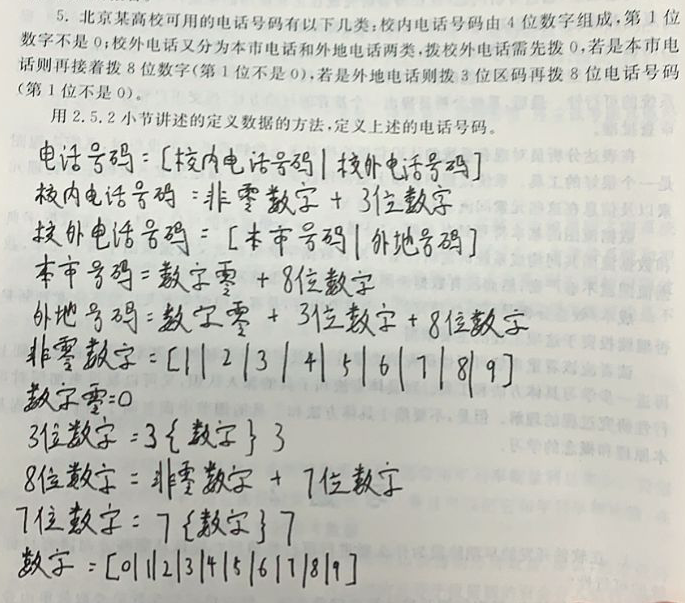
文件：描述数据的逻辑存储结构；

加工；

外部项。

**习题2**

**第5题**



## 第三章 需求分析

**3.1 需求分析的任务**

**3.1.1 确定对系统的综合要求**

**对系统的综合要求**

功能需求：这方面的需求指定系统必须提供的服务；

性能需求：性能需求指定系统必须满足的定时约束或容量约束，通常包括速度（响应时间）、信息量速率、主存容量、磁盘容量、安全性等方面的需求；

可靠性和可用性需求：可靠性需求定量地指定系统的可靠性；

出错处理需求：这类需求说明系统对环境错误应该怎样响应；

接口需求：接口需求描述应用系统与它的环境通信的格式；

约束：设计约束或实现约束描述在设计或实现应用系统时应遵守的限制条件；

逆向需求：逆向需求说明软件系统不应该做什么；

将来可能提出的要求：应该明确地列出那些虽然不属于当前系统开发范畴，但是据分析将来很可能会提出来的要求。

**3.2 与用户沟通获取需求的方法**

**3.1.1 访谈**

**了解访谈，并区分正式访谈与非正式访谈**

正式访谈时，系统分析员将提出一些事先准备好的具体问题，例如，询问客户公司销售的商品种类、雇佣的销售人员数目以及信息反馈时间应该多块等。

在非正式访谈中，分析员将提出一些用户可以自由回答的开放性问题，以鼓励被访问人员说出自己的想法，例如，询问用户对目前正在使用的系统有哪些不满意的地方。

当需要调查大量人员的意见时，向被调查人分发调查表是一个十分有效的做法。分析员仔细阅读收回的调查表，然后再有针对性地访问一些用户，以便向他们询问在分析调查表时发现的新问题。

在访问用户的过程中使用情景分析技术往往非常有效。所谓情景分析就是对用户将来使用目标系统解决某个具体问题的方法和结果进行分析。

**什么是正式访谈和非正式访谈**

正式访谈：正式访谈时，系统分析员将提出一些事先准备好的具体问题。

非正式访谈：在非正式访谈中，分析员将提出一些用户可以自由回答的开放性问题，以鼓励被访问人员说出自己的想法。

**3.3 分析建模与规格说明**

**了解数据模型（ER图），功能模型（数据流图），行为模型（状态图）**

看书P62页。

**3.4 实体-联系图**

**3.4.3 联系**

**区分1-1，1-n，n-m**

一对一联系（1：1）；一对多联系（1：N）；多对多联系（M：N）。

具体内容结合书本，例题和前置课程（数据库）知识。

**3.5 数据规范化**

**了解第一范式，第二范式，第三范式**

第一范式：每个属性值都必须是原子值，即仅仅是一个简单值而不含内部结构。

第二范式：满足第一范式条件，而且每个非关键字属性都由整个关键字决定（而不是由关键字的一部分来决定）。

第三范式：符合第二范式的条件，每个非关键字属性都仅由关键字决定，而且一个非关键字属性不能仅仅是对另一个非关键字属性的进一步描述（即一个非关键字属性值不依赖于另一个非关键字属性值）。

范式越高，冗余度越小，表越多，理解越困难。

具体内容结合书本，例题和前置课程（数据库）知识。

**3.6 状态转换图**

**学会画状态转换图**

看书P65-P67页。

## 第五章 总体设计

**总体设计的目的**

回答“概括地说，系统应该如何实现”这个问题。

**5.1 设计过程**

**了解总体设计过程**

设想供选择的方案；

选取合理的方案；

推荐最佳方案；

功能分解；

设计软件结构；

设计数据库；

制定测试计划；

书写文档（包括系统说明，用户手册，测试计划，详细的实现计划，数据库设计结果）；审查和复查。

**5.2 设计原理**

**5.2.1 模块化**

**了解模块化，抽象，逐步求精**

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

抽象：人类在认识复杂现象的过程中使用的最强有力的思维工具是抽象。人们在实践中认识到，在现实世界中一定事物、状态或过程之间总存在着某些相似的方面（共性）。把这些相似的方面集中和概括起来，暂时忽略它们之间的差异，这就是抽象。或者说抽象就是抽出事物的本质特性而暂时不考虑它们的细节。

逐步求精：是人类解决复杂问题时采用的基本方法，也是许多软件工程技术（例如，规格说明技术，设计和实现技术）的基础。

**5.2.5 模块独立**

**了解耦合**

耦合是对一个软件结构内不同模块之间互连程度的度量。耦合强弱取决于模块间接口的复杂程度，进入或访问一个模块的点，以及通过接口的数据。

**什么叫公共环境耦合（全局变量）**

当两个或多个模块通过一个公共数据环境相互作用时，它们之间的耦合称为公共环境耦合。

**了解内聚，内聚的种类**

内聚标志着一个模块内各个元素彼此结合的紧密程度，它是信息隐藏和局部化概念的自然拓展。简单地说，理想内聚的模块只做一件事情。

内聚分为低内聚和高内聚。

低内聚分为如下几类：如果一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的，就叫做偶然内聚；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，则称为逻辑内聚；如果一个模块包含的任务必须在同一段时间内执行，就叫时间内据。

高内聚也有两类：如果一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行，则称为顺序内聚；如果模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。

**5.3 启发规则**

**了解启发规则**

改进软件结构提高模块独立性；

模块规模应该适中；

深度、宽度、扇出和扇入都应适当；

模块的作用域应该在控制域之内；

力争降低模块接口的复杂程度；

设计单入口单出口的模块；

模块功能应该可以预测。

**5.4 描绘软件结构的图形工具**

**5.4.1 层次图和HIPO图**

**了解层次图**

层次图用来描绘软件的层次结构。

层次图中的一个矩形框代表一个模块，方框间的连线表示调用关系而不像层次方框图那样表示组成关系。

层次图很适于在自顶向下设计软件的过程中使用。

## 第六章 详细设计

**了解章节前言的第一段（代码质量由详细设计分析人决定的）**

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

**6.2 人机界面设计**

**6.2.1 设计问题**

**了解设计问题**

系统响应时间；

用户帮助设施；

出错信息处理；

命令交互。

**6.3 过程设计的工具**

**了解判定树**

看书P129页图6.7。

**6.5 程序复杂程度的定量度量**

**什么来衡量程序复杂，环形复杂度**

McCabe方法和Halstead方法。

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

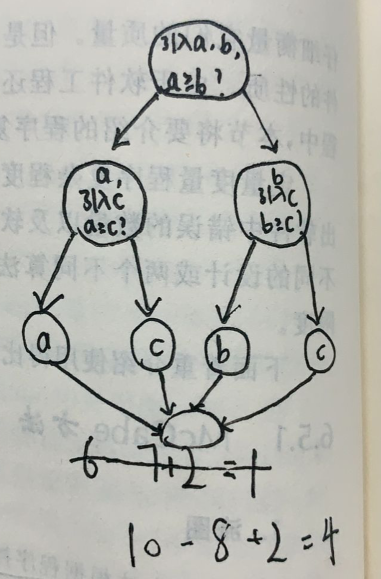
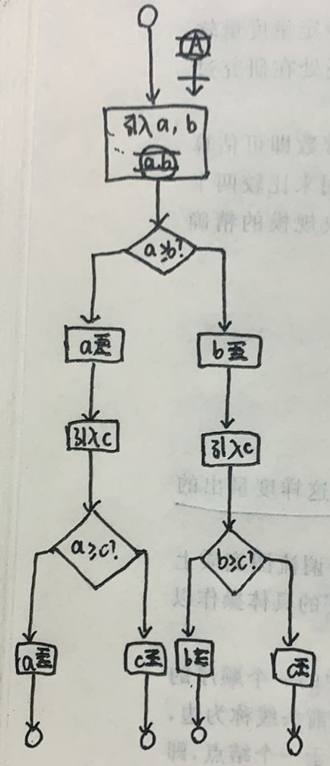
Halstead方法根据程序中运算符和操作数的总数来度量程序的复杂程度。

**学会画流程图，学会转换**

流图中线性无关的区域数等于环形复杂度；流图G的环形复杂度V(G)=E-N+2，其中，E是流图中边的条数，N是结点数；流图G的环形复杂度V(G)=P+1，其中，P是流图中判定结点的数目。

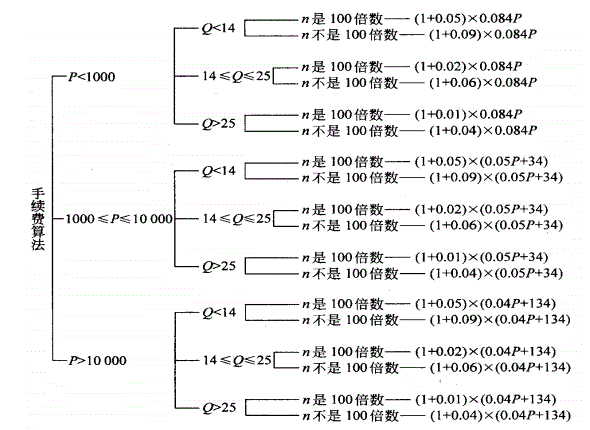
看书P137页。

下方流程图图例。



**习题6**

**第7题**



## 第七章 实现

**7.1 编码**

**7.1.2 编码风格**

**了解编码风格，认识什么是好编码**

源程序代码的逻辑简明清晰、易读易懂是好程序的一个重要标准

应该遵循下述规则：程序内部的文档；数据说明；语句构造；输入输出；效率。

**了解程序内部的文档**

程序清单的布局对于程序的可读性也有很大影响。应该利用适当的阶梯形式使程序的层次结构清晰明显。

**了解数据说明**

在设计期间确定了数据结构的组织和复杂程度，然而数据说明的风格却是在写程序时确定的。

应该遵循的简单原则：

数据说明的次序应该标准化。有次序就容易查阅，因此能够加速测试、调试和维护的过程。当多个变量名在一个语句中说明时，应该按字母顺序排列这些变量。

如果设计时使用了一个复杂的数据结构，则应该用注解说明用程序设计语言实现这个数据结构的方法和特点。

**了解语句构造**

设计期间确定了软件的逻辑结构，然而个别语句的构造却是编写程序的一个主要任务。

构造语句时应该遵循的原则是，每个语句都应该简单而直接，不能为了提高效率而使程序变得过分复杂。

常用规则：不要为了节省空间而把多个语句写在同一行；尽量避免复杂的条件测试；尽量减少对“非”条件的测试；避免大量使用循环嵌套和条件嵌套；利用括号使逻辑表达式或算术表达式的运算次序清晰直观。

**了解输入输出**

在设计和编写程序时应该考虑下述有关输入输出风格的规则：对所有输入数据都进行检验；检查输入项重要组合的合法性；保持输入格式简单；使用数据结束标记，不要要求用户指定数据的数目；明确提示交互式输入的请求，详细说明可用的选择或边界数值；当程序设计语言对格式有严格要求时，应保持输入格式一致；设计良好的输出报表；给所有输出数据加标志。

**了解效率**

效率主要指处理机时间和存储器容量两个方面。

3条基本原则：

首先，效率是性能要求，因此应该在需求分析阶段确定效率方面的要求；

其次，效率是靠好设计来提高的；

第三，程序的效率和程序的简单程度是一致的，不要牺牲程序的清晰性和可读性来不必要地提高效率。

**7.2 软件测试基础**

**7.2.1 软件测试的目标**

**什么是测试的目标**

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。

**7.2.2 软件测试准则**

**了解软件测试的准则**

所有测试都应该能追溯到用户需求；

应该远在测试开始之前就制定出测试计划；

把Pareto原理应用到软件测试中；

应该从“小规模”测试开始，并逐步进行“大规模”测试；

穷举测试是不可能的；

为了达到最佳的测试效果，应该由独立的第三方从事测试工作。

**7.2.4 测试步骤**

**了解测试的步骤**

模块测试；

子系统测试；

系统测试；

验收测试；

平行运行。

**了解模块测试**

好的软件系统：每个模块完成一个清晰定义的子功能，而且这个子功能和同级其他模块的功能之间没有相互依赖关系。

模块测试的目的是保证每个模块作为一个单元能正确运行，所以模块测试通常又称为单元测试。

所发现的往往是编码和详细设计的错误。

**了解子系统测试**

子系统测试是把经过单元测试的模块放在一起形成一个子系统来测试。

模块相互间的协调和通信是这个测试过程中的主要问题，

着重测试模块的接口。

**了解系统测试**

系统测试是把经过测试的子系统装配成一个完整的系统来测试。

不仅应该发现设计和编码的错误，还应该验证系统确实能提供需求说明书中指定的功能，而且系统的动态特性也符合预定要求。

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。

**了解验收测试**

验收测试把软件系统作为单一的实体进行测试，测试内容与系统测试基本类似，但是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。

验收测试的目的是验证系统确实能够满足用户的需要，在这个测试步骤中发现的往往是系统需求说明书中的错误。验收测试也称为确认测试。

**了解平行运行**

关系重大的软件产品在验收之后往往并不立即投入生产性运行，而是要再经过一段平行运行时间的考验。

平行运行就是同时运行新开发出来的系统和将被它取代的旧系统，以便比较新旧两个系统的处理结果。这样做的具体目的有如下几点：

可以在准生产环境中运行新系统而又不冒风险；

用户能有一段熟悉新系统的时间；

可以验证用户指南和使用手册之类的文档；

能够以准生产模式对新系统进行全负荷测试，可以用测试结果验证性能指标。

**7.3 单元测试**

**7.3.1 测试重点**

**了解测试重点**

模块接口；

局部数据结构；

重要的执行通路；

出错处理通路。

**7.3.2 代码审查**

**了解代码审查**

人工测试源程序可以由编写者本人非正式地进行，也可以由审查小组正式进行。后者称为代码审查（走读）

代码审查是一种非常有效的程序验证技术，对于典型的程序来说，可以查出30%～70%的逻辑设计错误和编码错误。

审查小组最好由下述4人组成：

组长，应该是一个很有能力的程序员，而且没有直接参与这项工程；

程序的设计者；

程序的编写者；

程序的测试者。

**7.3.3 计算机测试**

**存根程序，驱动软件的概念**

模块并不是一个独立的程序，因此必须为每个单元测试开发驱动软件和(或)存根软件。

通常驱动程序也就是一个“主程序”，它接收测试数据，把这些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。因此存根程序也可以称为“虚拟子程序”。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。

驱动程序和存根程序代表开销：为了进行单元测试必须编写测试软件，但是通常并不把它们作为软件产品的一部分交给用户。

许多模块不能用简单的测试软件充分测试，为了减少开销可以使用渐增式测试方法，在集成测试的过程中同时完成对模块的详尽测试。

模块的内聚程度高可以简化单元测试过程。如果每个模块只完成一种功能，则需要的测试方案数目将明显减少，模块中的错误也更容易预测和发现。

**7.4 集成测试**

**了解集成测试**

集成测试是测试和组装软件的系统化技术，例如，子系统测试即是在把模块按照设计要求组装起来的同时进行测试，主要目标是发现与接口有关的问题(系统测试与此类似)。

例如：

数据穿过接口时可能丢失；

一个模块对另一个模块可能由于疏忽而造成有害影响；

把子功能组合起来可能不产生预期的主功能；

个别看来是可以接受的误差可能积累到不能接受的程度；

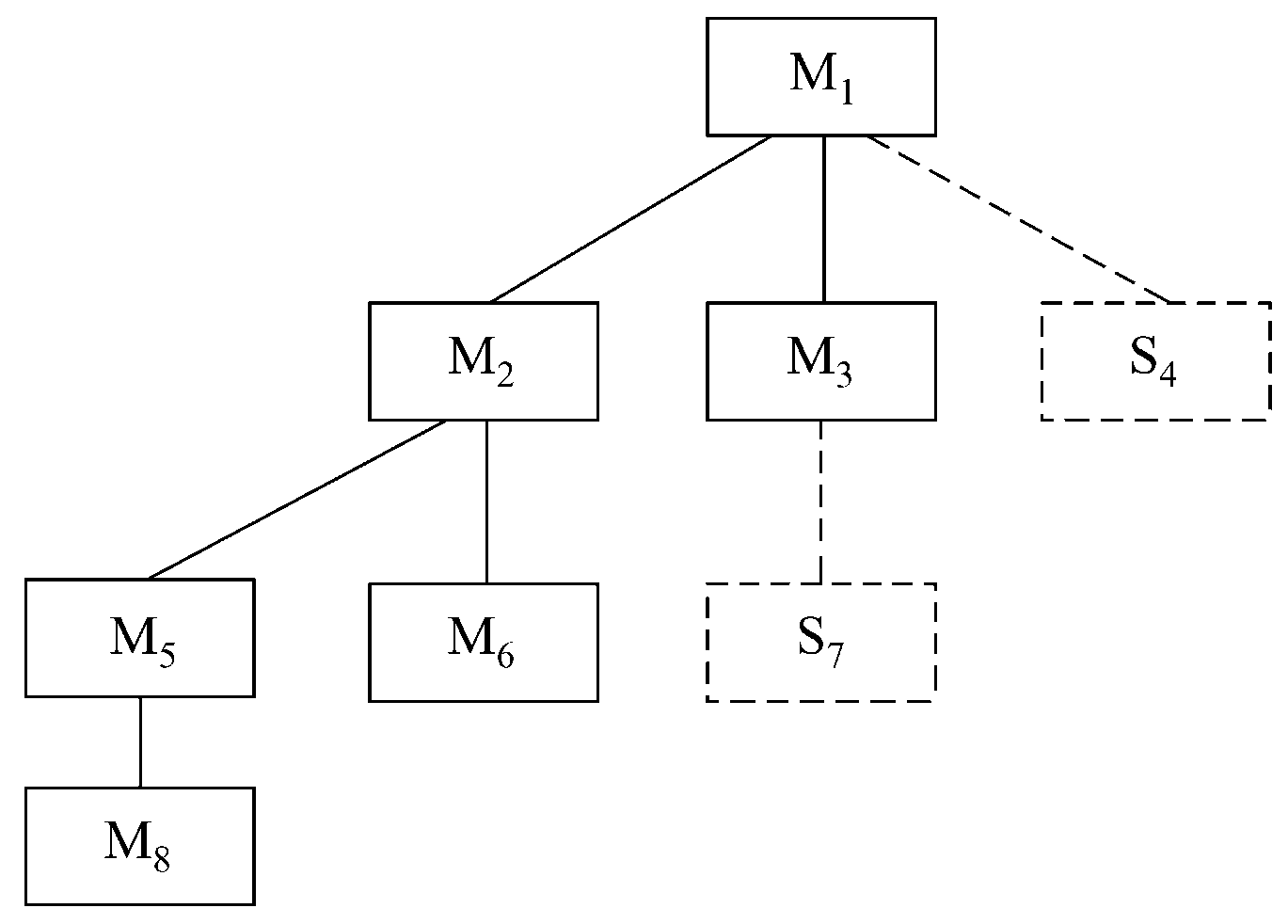
全程数据结构可能有问题等等。

**7.4.1 自顶向下集成**

**了解自顶向下集成**

自顶向下集成方法从主控制模块开始，沿着程序的控制层次向下移动，逐渐把各个模块结合起来。在把附属于（及最终附属于）主控制模块的那些模块组装到程序结构中去时，或者使用深度优先的策略，或者使用宽度优先的策略。

参看下图，深度优先的结合方法先组装在软件结构的一条主控制通路上的所有模块。选择一条主控制通路取决于应用的特点，并且有很大任意性。而宽度优先的结合方法是沿软件结构水平地移动，把处于同一个控制层次上的所有模块组装起来。



把模块结合进软件结构的具体过程由下述4个步骤完成：

第一步，对主控制模块进行测试，测试时用存根程序代替所有直接附属于主控制模块的模块；

第二步，根据选定的结合策略(深度优先或宽度优先)，每次用一个实际模块代换一个存根程序(新结合进来的模块往往又需要新的存根程序)；

第三步，在结合进一个模块的同时进行测试；

第四步，为了保证加入模块没有引进新的错误，可能需要进行回归测试(即，全部或部分地重复以前做过的测试)。

**7.4.2 自底向上集成**

**了解自底向上集成**

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

用下述步骤可以实现自底向上的结合策略：

第一步，把低层模块组合成实现某个特定的软件子功能的族；

第二步，写一个驱动程序(用于测试的控制程序)，协调测试数据的输入和输出；

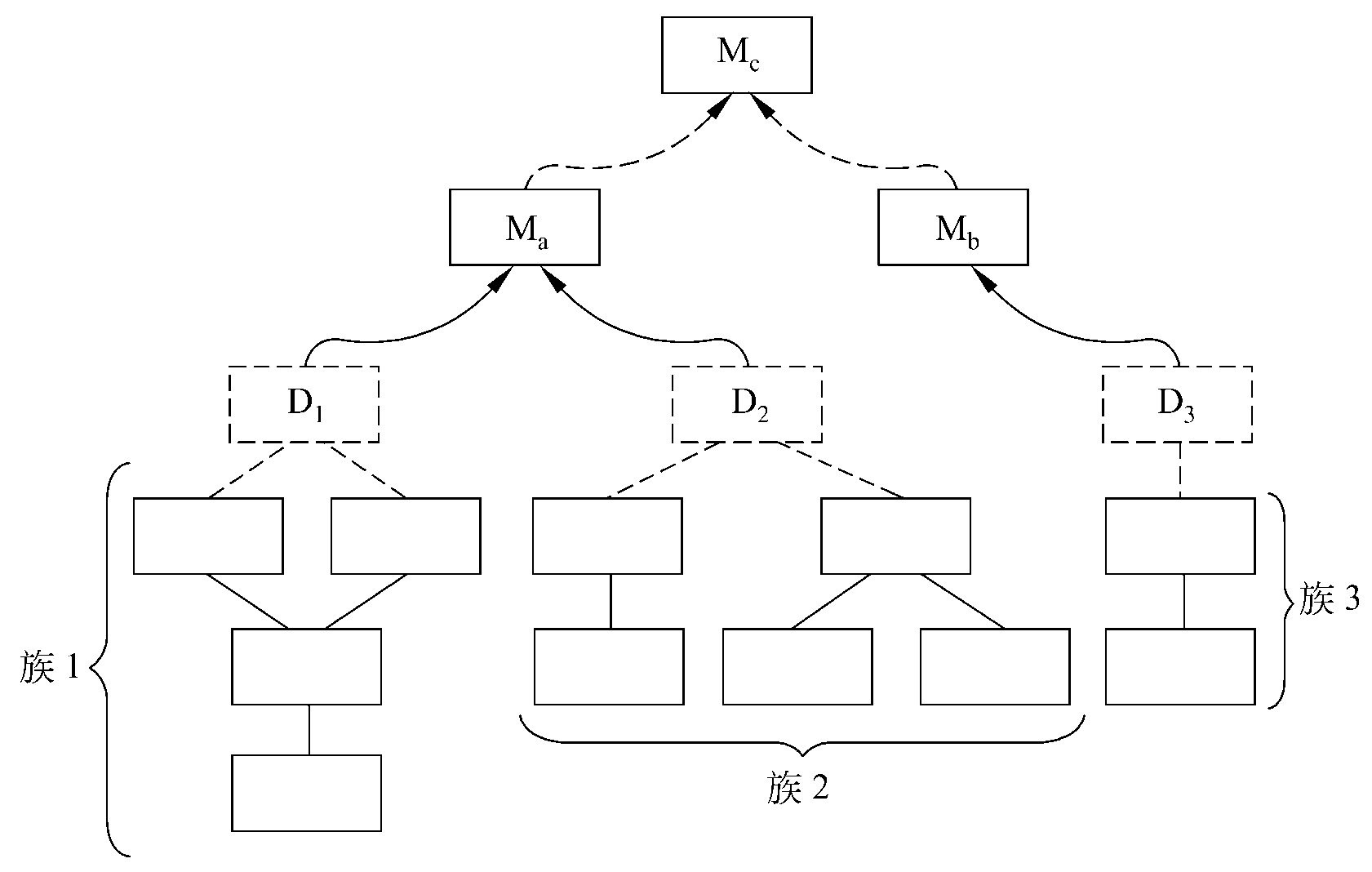
第三步，对由模块组成的子功能族进行测试；

第四步，去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成更大的子功能族

上述第二步到第四步实质上构成了一个循环。下图描绘了自底向上的结合过程。

随着结合向上移动，对测试驱动程序的需要也减少了。

事实上，如果软件结构的顶部两层用自顶向下的方法组装，可以明显减少驱动程序的数目，而且族的结合也将大大简化。



**7.4.3 不同集成测试策略的比较**

**了解不同集成测试策略的比较**

一种方法的优点正好对应于另一种方法的缺点。

自顶向下测试方法的主要优点是不需要测试驱动程序，能够在测试阶段的早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误。

自顶向下测试方法的主要缺点是需要存根程序，可能遇到与此相联系的测试困难，低层关键模块中的错误发现较晚，而且用这种方法在早期不能充分展开人力。

可以看出，自底向上测试方法的优缺点与上述自顶向下测试方法的优缺点刚好相反。

**7.5 确认测试**

**7.5.3 Alpha和Beta测试**

**了解Alpha beta测试**

验收测试是由最终用户而不是系统的开发者进行的。事实上，验收测试可以持续几个星期甚至几个月。

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

Beta测试由软件的最终用户们在一个或多个客户场所进行。与Alpha测试不同，开发者通常不在Beta测试的现场，因此，Beta测试是软件在开发者不能控制的环境中的“真实”应用。用户记录在Beta测试过程中遇到的一切问题（真实的或想像的），并且定期把这些问题报告给开发者。接收到在Beta测试期间报告的问题之后，开发者对软件产品进行必要的修改，并准备向全体客户发布最终的软件产品。

**7.6 白盒测试技术**

**了解白盒测试概念**

设计测试方案是测试阶段的关键技术问题。

测试方案包括具体的测试目的（例如，预定要测试的具体功能），应该输入的测试数据和预期的结果。通常又把测试数据和预期的输出结果称为测试用例。其中最困难的问题是设计测试用的输入数据。

不同的测试数据发现程序错误的能力差别很大，为了提高测试效率降低测试成本，应该选用高效的测试数据。因为不可能进行穷尽的测试，选用少量“最有效的”测试数据，做到尽可能完备的测试就更重要了。

**7.6.1逻辑覆盖**

**了解逻辑覆盖**

有选择地执行程序中某些最有代表性的通路是对穷尽测试的惟一可行的替代办法。

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

测试数据执行(或叫覆盖)程序逻辑的程度可以划分成哪些不同的等级呢?从覆盖源程序语句的详尽程度分析，大致有以下一些不同的覆盖标准。

**了解语句覆盖**

语句覆盖的含义是，选择足够多的测试数据，使被测程序中每个语句至少执行一次。

**了解判定覆盖**

判定覆盖又叫分支覆盖，它的含义是，不仅每个语句必须至少执行一次，而且每个判定的每种可能的结果都应该至少执行一次，也就是每个判定的每个分支都至少执行一次。

判定覆盖比语句覆盖强，但是对程序逻辑的覆盖程度仍然不高。

**了解条件覆盖**

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

条件覆盖通常比判定覆盖强，因为它使判定表达式中每个条件都取到了两个不同的结果，判定覆盖却只关心整个判定表达式的值。

**7.7 黑盒测试技术**

**了解什么是黑盒测试技术**

黑盒测试着重测试软件功能。黑盒测试并不能取代白盒测试，它是与白盒测试互补的测试方法，它很可能发现白盒测试不易发现的其他类型的错误。

白盒测试在测试过程的早期阶段进行，而黑盒测试主要用于测试过程的后期。

设计黑盒测试方案时，应该考虑下述问题：

怎样测试功能的有效性？

哪些类型的输入可构成好测试用例？

系统是否对特定的输入值特别敏感？

怎样划定数据类的边界？

系统能够承受什么样的数据率和数据量？

数据的特定组合将对系统运行产生什么影响？

**应用黑盒测试技术，能够设计出满足下述标准的测试用例集**

所设计出的测试用例能够减少为达到合理测试所需要设计的测试用例的总数；

所设计出的测试用例能够告诉我们，是否存在某些类型的错误，而不是仅仅指出与特定测试相关的错误是否存在。

**力图发现错误的类型**

功能不正确或遗漏了功能；

界面错误；

数据结构错误或外部数据库访问错误；

性能错误；

初始化和终止错误。

**7.7.1 等价划分**

**什么是等价划分**

等价划分是一种黑盒测试技术，这种技术把程序的输入域划分为若干个数据类，据此导出测试用例。一个理想的测试用例能独自发现一类错误（例如对所有负整数的处理都不正确）。等价划分法力图设计出能发现若干类程序错误的测试用例，从而减少必须设计的测试用例的数目。

**7.7.2 边界值分析**

**了解边界值分析**

边界值分析是一种测试用例设计技术，它是等价划分的补充。

经验表明，处理边界情况时程序最容易发生错误。

例如，许多程序错误出现在下标、纯量、数据结构和循环等等的边界附近。

**7.7.3 错误推测**

**了解错误推测**

不同类型不同特点的程序通常又有一些特殊的容易出错的情况。

有时分别使用每组测试数据时程序都能正常工作，这些输入数据的组合却可能检测出程序的错误。

一般说来，即使是一个比较小的程序，可能的输入组合数也往往十分巨大，因此必须依靠测试人员的经验和直觉，从各种可能的测试方案中选出一些最可能引起程序出错的方案。

对于程序中可能存在哪类错误的推测，是挑选测试方案时的一个重要因素。

**7.8 调试**

**了解调试的概念**

调试（也称为纠错）作为成功测试的后果出现，也就是说，调试是在测试发现错误之后排除错误的过程。

虽然调试应该而且可以是一个有序过程，但是，目前它在很大程度上仍然是一项技巧。

软件工程师在评估测试结果时，往往仅面对着软件错误的症状，也就是说，软件错误的外部表现和它的内在原因之间可能并没有明显的联系。调试就是把症状和原因联系起来的尚未被人深入认识的智力过程。

**7.8.2 调试过程**

**了解调试的途径**

有3种调试途径：蛮干法，回溯法，原因排除法。

**了解回溯法**

回溯法：是一种相当常用的调试方法，当调试小程序时这种方法是有效的。具体做法是，从发现症状的地方开始，人工沿程序的控制流往回追踪分析源程序代码，直到找出错误原因为止。但是，随着程序规模的扩大，应该回溯的路径数目也变得越来越大，以至彻底回溯变成完全不可能了。

**7.9 软件可靠性**

**7.9.1 基本概念**

**了解软件可靠性的定义**

软件可靠性是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。

**了解软件可用性的定义**

软件可用性是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。

By：岛上的黄鸡