CHENGXU SHEJI FANGFAXUE

程序设计方法学

朱三元 编著

上海交通大学出版社

计算机软件与应用丛书

计算机软件与应用丛书

程序设计方法学

朱三元　编著

上海交通大学出版社

内 容 简 介

本书为《计算机软件与应用》丛书之一，旨在为读者提供一册程序设计方法学和软件工程基础的教材。全书共分十章：程序和软件的概念、软件工程的基本概念、控制结构、数据结构、软件结构、模块化原则、自顶向下原则、结构设计方法、Jackson 方法和软件确认技术。

本书内容丰富，许多材料选自 80 年代最新软件技术，可供大专院校师生及广大软件科学工作者使用。

程序设计方法学
上海交通大学出版社出版
（淮海中路1984弄19号）
新华书店上海发行所发行
常州村前印装厂印装

开本787×1092毫米 1/32 印张10.25 字数230000
1988年9月第1版 1988年10月第1次印刷
印数：1—00,000
ISBN7-313-00300-5/TP31 科技书目：180-249

定价：1.70元

前　言

随着计算机应用的不断发展，软件技术也相应得到了较大的发展。我国的软件产业正在建立和形成。为了使软件研制的成果迅速转变为软件产品，开发软件的个体手工艺作坊方式必须演变成集体大规模生产方式。这种转变就迫切需要"软件工程学"。

为了获得一个可用的、可维护的、可靠的软件产品，必须明确软件生存周期中各阶段的工作范围，实现每一阶段开发工作的规范化，并给出各种有效的软件开发技术、开发工具及使用时的管理方式。归纳起来是：原则与方法、工具与环境、规范与管理，这就是软件工程的三大支柱。

软件工程与计算机科学有密切的关系，但软件工程方法不会象数学那样公理化，它有自己的基础。因此，本书在介绍有关基础知识时，不准备给出严格的证明。另外，有些软件设计方法目前尚处在理论研究阶段或实验阶段，离实际使用还存在一定距离，尚未被工业界采用。所以这方面的基础知识也不准备在本书中作介绍。

本书共分十章。第一章和第二章介绍软件工程的基本概念，是全书的基础；第三章介绍程序设计中的控制结构；第四章介绍数据结构；第五章介绍软件结构；第六章介绍模块原则；第七章介绍自顶向下原则，这种原则适应于软件生存周期各阶段；第八章和第九章介绍两种主要的软件设计方法，也是国际上

较为流行的方法；第十章介绍软件确认技术中常用的一些方法，以便读者在开发软件时灵活应用。

由于本人水平有限，且软件工程技术发展迅速，对书中不妥之处欢迎读者批评指正。

上海软件中心的陈敏同志、计立奇同志为本书的编写作了不少有益的工作，北京大学杨芙清教授为本书提出了许多宝贵的意见，在此谨向他们表示衷心的感谢。

编　者

1987年12月

《计算机软件与应用丛书》

（第一辑）

计算机数据安全原理	周锡龄	著
情报检索	王和珍	编著
数据库设计	施伯乐等	编著
程序设计方法学	朱三元	编著
微型计算机应用例解	白英彩等	编著

（第二辑）

计算机仿真	邱百光	编著

《计算机软件与应用丛书》编委会名单

目　录

第一章　程序和软件的概念

1.1　程序的概念 ……（1）
1.1.1　程序的定义 ……（1）
1.1.2　程序的理解 ……（2）
1.1.3　程序设计的历史回顾 ……（3）
1.1.4　程序间的关系 ……（6）
1.1.5　程序与算法 ……（15）
1.1.6　程序的表示法 ……（17）
1.1.7　程序的结构 ……（21）
1.2　软件的概念 ……（22）
1.2.1　软件和程序的差异 ……（22）
1.2.2　系统软件和应用软件 ……（25）
1.2.3　软件的品质 ……（27）

第二章　软件工程的基本概念

2.1　软件工程定义 ……（33）
2.2　软件工程和一般工程的差异 ……（34）
2.3　软件工程发展简史 ……（36）
2.4　软件工程的现状与未来 ……（38）
2.5　软件的生存周期 ……（40）
2.6　软件研制的基本原则 ……（48）
2.7　软件设计的工具 ……（51）

2.8 软件结构 ……（60）
2.9 软件生产管理 ……（63）
第三章 控制结构
3.1 控制结构中的三个基型 ……（70）
3.1.1 顺序结构 ……（70）
3.1.2 选择结构 ……（73）
3.1.3 重复结构 ……（75）
3.2 术语的定义 ……（77）
3.2.1 程序公式 ……（77）
3.2.2 常态程序 ……（80）
3.2.3 初等程序 ……（82）
3.2.4 结构化程序 ……（84）
3.2.5 小结 ……（86）
3.3 三条定理 ……（88）
3.3.1 引理 ……（88）
3.3.2 结构性定理 ……（89）
3.3.3 正确性定理 ……（90）
3.3.4 展开定理 ……（91）
3.4 关于对GOTO语句的认识 ……（93）
3.4.1 滥用GOTO语句确实有害 ……（95）
3.4.2 如何避免滥用GOTO语句 ……（98）
3.4.3 带有GOTO语句的结构程序设计 ……（107）
3.4.4 结论 ……（115）
第四章 数据结构
4.1 概述 ……（116）
4.2 简单类型 ……（118）

4.2.1 基本类型 ……（118）
4.2.2 字符串类型和枚举类型 ……（121）
4.3 构造类型 ……（122）
4.3.1 数组 ……（122）
4.3.2 记录 ……（123）
4.3.3 集合 ……（124）
4.4 动态数据结构 ……（125）
4.4.1 顺序文卷 ……（125）
4.4.2 指引元类型 ……（126）
4.4.3 链表 ……（127）
4.4.4 树 ……（128）

第五章 软件结构

5.1 概述 ……（130）
5.2 模块的聚合 ……（132）
5.2.1 聚合的类型 ……（132）
5.2.2 聚合类型的区分 ……（136）
5.3 模块的关联和独立性 ……（136）
5.3.1 关联的类型 ……（137）
5.3.2 数据隐蔽 ……（139）
5.4 模块大小与标准化 ……（147）
5.4.1 模块的大小 ……（147）
5.4.2 标准模块 ……（150）
5.5 模块化及其优缺点 ……（151）
5.5.1 模块化的优点 ……（152）
5.5.2 模块化的缺点 ……（154）

第六章　模块化原则

6.1　概述 ……（156）
6.2　独立子程序和数据隐蔽 ……（157）
6.3　使用判定表 ……（158）
6.4　使用符号参数 ……（163）
6.5　分离输入输出功能 ……（170）
6.6　不共享临时存储单元 ……（172）
6.7　模块化系统的组装 ……（172）
6.8　模块化系统的优化 ……（176）

第七章　自顶向下原则

7.1　概述 ……（179）
7.2　自顶向下设计 ……（180）
7.2.1　基本概念 ……（180）
7.2.2　设计的要点 ……（181）
7.2.3　为设计开发结构化规范书 ……（189）
7.3　自顶向下编码 ……（190）
7.3.1　定义 ……（190）
7.3.2　采用自顶向下的理由 ……（191）
7.4　自顶向下设计的选择、变体及问题 ……（194）
7.4.1　简单方法 ……（195）
7.4.2　自顶向下和自底向上的结合 ……（196）
7.4.3　组织管理 ……（197）

第八章　结构设计方法

8.1　概述 ……（199）
8.1.1　术语定义 ……（200）
8.1.2　系统组织的模型 ……（202）

8.1.3 系统形态 ……（203）
8.1.4 作用域与控制域 ……（204）
8.2 转换分析法 ……（206）
8.2.1 用数据流程图表示用户问题 ……（207）
8.2.2 标识传入数据元素和传出数据元素 ……（209）
8.2.3 顶层分解 ……（211）
8.2.4 对各个分支的分解 ……（212）
8.2.5 系统结构的调整 ……（216）
8.3 事务分析法 ……（217）
8.3.1 事务分析策略 ……（218）
8.3.2 举例 ……（221）
8.3.3 分析法对比 ……（227）
第九章 Jackson方法
9.1 基本原理及程序组织形式 ……（229）
9.1.1 基本原理及目的 ……（229）
9.1.2 程序的组织形式 ……（231）
9.1.3 Jackson 法概述 ……（236）
9.2 Jackson法的基本技术 ……（239）
9.2.1 基本设计方法 ……（239）
9.2.2 顺序文卷处理技术 ……（239）
9.2.3 简例 ……（244）
9.2.4 多种数据结构的匹配 ……（248）
9.3 回溯技术 ……（253）
9.3.1 产生回溯的原因 ……（253）
9.3.2 回溯处理技术 ……（254）
9.3.3 副作用处理 ……（258）

9.4 结构冲突的消除技术 ……（261）
9.4.1 冲突类型 ……（261）
9.4.2 消除冲突的技术 ……（264）
9.4.3 结论 ……（275）
9.5 系统设计原理 ……（277）
9.6 Jackson法与结构设计法的比较……（279）
第十章 软件确认技术
10.1 概述……（281）
10.2 程序正确性证明……（284）
10.2.1 自底向上方法……（285）
10.2.2 自顶向下方法……（288）
10.3 静态分析技术……（288）
10.3.1 结构预查……（288）
10.3.2 流图分析……（290）
10.3.3 符号执行……（292）
10.3.4 小结……（297）
10.4 动态测试技术……（297）
10.4.1 简述……（297）
10.4.2 功能测试……（299）
10.4.3 结构测试……（303）
10.4.4 测试策略……（307）
10.4.5 测试计划的制订……（312）
10.5 比较……（313）
10.6 软件生存周期各阶段的验证活动……（313）
10.6.1 需求阶段……（313）
10.6.2 设计阶段……（315）

10.6.3 编码阶段……（315）
10.6.4 维护阶段……（316）

第一章　程序和软件的概念

1.1　程序的概念

1.1.1　程序的定义

在近30年中，计算机已成为商业、工业和科学研究中不可缺少的工具，并且正在走向社会化，深入到家庭。由于计算机功能强实、应用广泛，并且硬件成本逐年下降，其性能价格比每10年改进一个数量级，在经济上人们承受得起，所以受到普遍的欢迎。计算机通常有若干条基本**指令**，机器能够理解并能遵从这些指令，而且这些指令执行起来既方便又可靠。计算机功能强实就在于它能执行特别长的指令序列。

程序就是一串指令序列的集合，它能被计算机执行，这种对程序的认识是既十分重要又十分古典的。为什么这样说呢？关键在于对指令的认识。早期的计算机指令系统(或叫机器语言)是由组合电路控制的，但现在不少计算机是由微程序控制的。微程序也是程序，它是由一串**微指令**组成的。再则，60年代开始，已经广泛地采用高级程序设计语言来编写程序，而这种程序已变成一串**语句**的序列。由此可以看出对程序下定义是和具体的计算机系统有关。

那么程序的基本思想能否在不依赖计算机的情况下加以解释和理解呢？

在此，我们先介绍几个概念：

最重要的概念是**动作**，一个动作被理解成在有限的持续时间内，具有预定的和意义明确的效果。

每个动作都要求有执行**对象**，并且根据其**状态变化**能够辨认出动作的效果。

每个动作和对象还必须能用一种语言或公式系统来描述，这种描述称为**语句**(注意：这里的语句既可以理解为高级程序设计语言中的语句及说明，也可以理解为机器语言中的指令，同样还可以理解为微指令。如果我们再进一步引伸，它可以是自然语言中的语句，也可能是流程图中的某一框，如此等等)。

至此，我们就可以说，一个**程序**是一串语句的序列。一般说来，这些语句的文本顺序和其执行顺序并不完全一致。如果一个动作能分解成几部分，那么这样的动作就叫一个**过程**或一个计算。因此，描述过程的一个语句能够被进一步分解成若干个部分，此时的语句也就成为一个程序了。所以程序是对计算任务的动作和对象的描述。

由此可见，程序的概念早就存在了，它在计算机产生之前就形成了，即程序设计对于人们来说是先于计算机存在的。例如，早晨起床后要做一系列的动作：打开炉子、煮稀饭、买菜、洗脸、漱口、帮助孩子穿衣服等等。这些事情先做什么后做什么，就需要编排一番，亦即需要进行程序设计。当然，我们还可以举不少例子来说明程序设计不一定和计算机发生关系。这些编好的程序存在于人脑之中，而不一定传送给计算机，当然也可以把它存放到计算机中。

1.1.2 程序的理解

在过去的30多年中，随着计算机处理能力的增强，程序的

长度、复杂性和高度技巧性都打破了历史记录。人们越来越不满足单纯地为解算某一个题目而去编写程序，而是设法做成一个程序系统。可是我们的程序设计能力并不能满足日益增长的实际需要。人们已经意识到，程序设计是一种智力的挑战。程序设计技术是一种组织复杂性的技术，是一种控制巨量数据的技术，也是一种尽量有效地回避混乱的技术。

程序本身并不是我们的目的，程序的目的在于计算，而计算的目的在于得到某种期望的结果。虽然程序员的工作最终表现于写出一个程序，但是，这个程序所可能引起的计算却是他真正的职责所在。在把"静态"的程序文本变成"动态"的计算过程中，我们可以引伸出如下几个观点：

1．程序文本的结构应该准确无误地反映出计算的结构。

2．每当程序员说明他的程序正确性时，事实上是断言他的程序所可能引起的计算的正确性。

3．正如一个数学定理一旦被证明时它的意义就清楚一样，一个程序只有通过执行后，它的意义才能被理解。

4．静态的程序文本是顺序的，而动态的计算可能是顺序的，也可能是并发的，因此这种转换过程更增加了复杂性。

由于计算的复杂性而带来的程序复杂性是可想而知的，反之，程序越复杂其所引起的计算也越复杂，此时，我们的程序设计能力就不能满足这种需要。

1.1.3 程序设计的历史回顾

直到50年代末期，即高级程序设计语言能够真正用上之前，程序都是由一长串指令编码所组成，而这些编码采用的是最简单的符号(主要是0～9的数字)。很显然，这样的设计过程是非

常繁琐的。随着有大容量存储器的快速计算机的出现，这种繁琐的设计过程就变得越来越不适应了。通常，我们把这种程序叫做**手编程序**。

手编程序的缺点是：

1．程序员必须把他的程序编成适合于所用计算机的特点。因此，他必须考虑到机器的所有细节，包括计算机的结构和机器的指令系统。正象不同机器的指令系统是不能交换的一样，不同机器的程序也是不能交换的。甚至在一台机器上所用的编码方法最精细的知识，在使用另一台计算机时，也是完全无用的。

程序员经过千辛万苦花费了艰巨的劳动，从而得到的一个正确的手编程序，却被一台计算机束缚住。因此，程序并不能成为一个很好的资源，这显然是人类智慧的一个不明智的浪费。

2．程序员与某一类型计算机的紧密联系不仅可能，甚至促进了各类技巧的发明和应用，以便充分发挥机器的效能。当把这种"技巧"作为好的程序设计基础时，程序员就要花费相当的时间才能做到"优化"代码结构。优化的手编程序的证明通常是很困难的。事实上，理解同行所设计的一个程序的原理也是很困难的。因此，现在对于这种技巧已不能引以为荣了。有经验的程序员自觉地避免使用这种"技巧"，而选择系统化且简单明了的解决办法来代替它。

在计算机问世不久时，有些人十分注重如何节省程序的指令条数，甚至冒在执行中修改指令的危险使程序缩短。这种技巧使得计算机正在执行的程序，在两个不同时间求出的代码和有可能不相一致。历史证明，随着计算机性能的改善（如增加