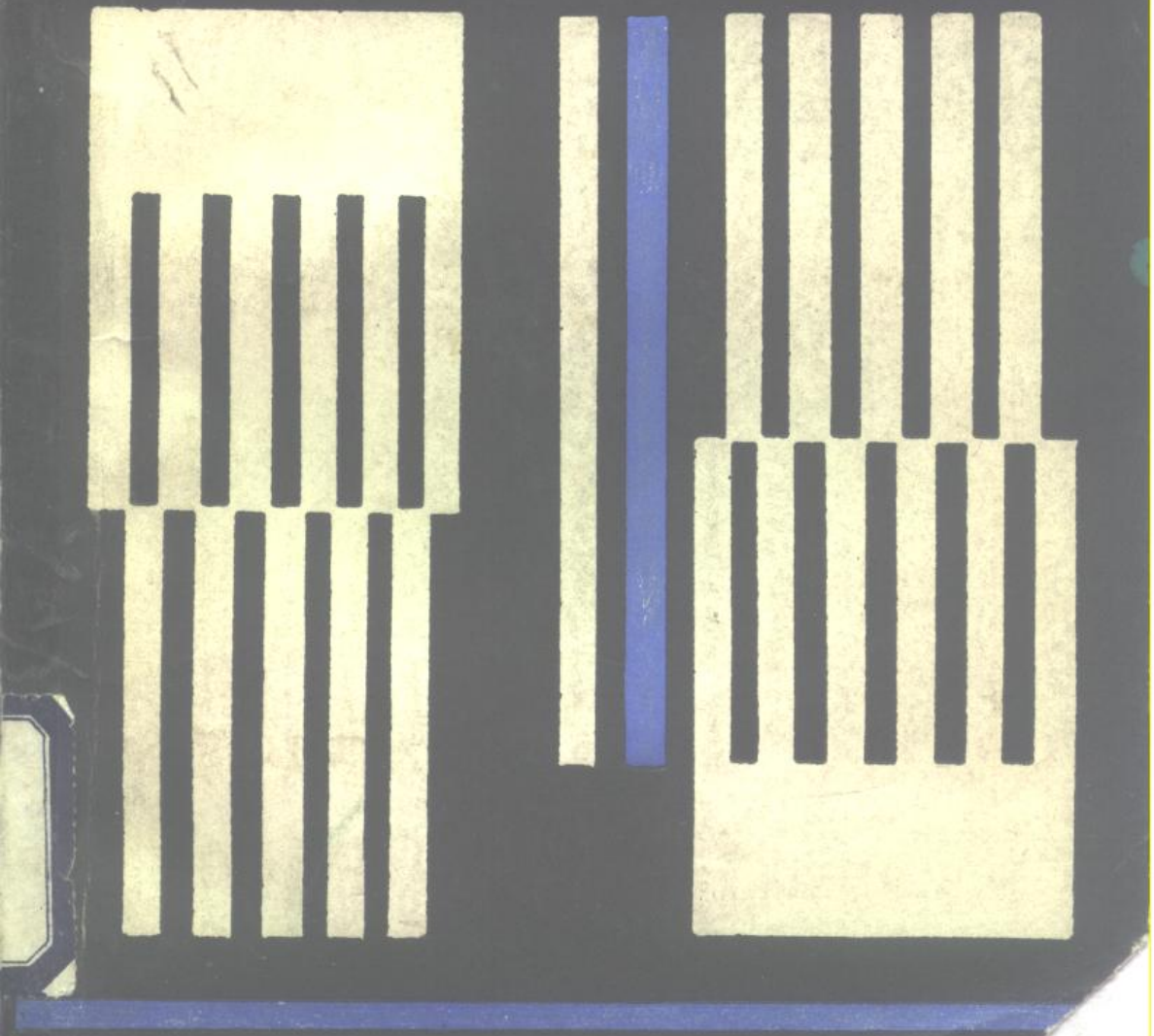
程序设计方法学引论

陶葆兰　李庆华　编　著



华中理工大学出版社

程序设计方法学引论

陶葆兰　李庆华编著

华中理工大学出版社

程序设计方法学引论

陶葆兰　李庆华　编著

责任编辑　陈少华

*

华中理工大学出版社出版发行

（武昌喻家山）

华中理工大学出版社沔阳印刷厂印刷

*

开本：850×1168 1/32　印张：7.875 字数：188 000

1989年12月第一版　1989年12月第一次印刷

印数：1—1 000

ISBN 7-5609-0320-7/TP·27

定价：1.60元

内 容 提 要

本书以程序推导技术为主要研究内容，论述了程序设计方法学的基本原理和方法，着重阐述程序构造和正确性证明同时进行，并以正确性证明为主导的研制程序的原则和方法。书中还探讨了程序质量问题，提出了提高程序效率和可读性的几种途径。对于程序设计方法学中有待深入研究的课题，书中也作了适当介绍。为帮助读者掌握书中所述理论和方法，除配有适当应用实例外，每章末均附有一定数量的针对性强的练习题。

本书内容兼顾理论与实际，适于读者系统学习。本书可作为大专院校软件专业学生、研究生的教材，也可供从事计算机工作的科技人员阅读参考。

前　言

程序设计方法学是计算机科学的一个分支学科，发展极为迅速，目前已渐趋成熟。本书的目的，旨在按照教学的要求，阐明这一学科的基本原理和方法，使读者能够掌握程序设计方法学的理论、方法和某些技巧，并能应用于实际编程中解决有关问题。

本书是根据多年教学实践中积累的经验，并学习和吸收了国内外同类教材的优点编写而成的。1982年以来，我们在华中理工大学计算机系一直讲授程序设计方法学课程，并曾根据讲稿编写成《程序设计理论基础》讲义，作为八三、八四与八五级软件硕士研究生的教材。在此基础上经过修改与补充，又编写了原讲义的修订本——《程序设计方法学引论》作为八六级研究生的教材，且在以下几部分作了适当的变动：删去了与离散数学课程重复的内容；补充介绍了4种程序设计方法；合并原讲义中的第十九章和第二十二章为新讲义的第十一章（“程序质量问题初议”）；在章节安排上将原讲义的22章调整为新讲义的13章。本书是在教学中参照读者提出的意见，对新讲义进行再修改后定稿的。其中，带“*”号的章节为选讲内容。

全书分为四篇：第一篇为绪论，阐述程序设计方法学的重要性，简要介绍当前应用比较广泛和继续发展的几种程序设计方法。第二篇包括第一至三章，介绍必须的预备知识；对于数组提出一种新的观点，引进适当的标记方法，还介绍了程序的正确性证明及其法则。第三篇包括第四至七章，介绍一个小型程序设计语言——岗哨命令语言，并用最弱前置条件定义其语义（第四篇中程序研制的实例都是用这种语言写成的）。第四篇包括第八至十三章，讨论研制程序的一种基本方法——程序的推导技术和有

试读结束，需要全本PDF请购买 www.ertongbook.com

关的理论，它以最弱前置条件为基础，同时，还用1章的篇幅，对程序质量问题作了初步探讨。

本书力求反映下列特点：

1．符合原教育部1983年颁布的《程序设计方法学引论》教学大纲要求；

2．在阐明程序设计方法学的主要思想时，强调Dijkstra和Gries的观点，即一边构造程序正确性的证明，一边编制程序；

3．突出重点，对程序的推导技术，讲深讲透，而对一般程序设计方法只作简要介绍；

4．自始至终强调理论联系实际，注重启发读者如何分析问题和解决问题；

5．每章末附有一定数量的具有针对性的习题，帮助读者进一步掌握所学的理论和方法。

本书可作为高等学校计算机软件硕士研究生和软件本科高年级学生的教材或教学参考书，也可供广大软件科技人员阅读参考。

在编写本书的过程中，丁忠俊同志参加过原讲义《程序设计理论基础》的编写，做了不少工作；使用讲义的兄弟院校给予了热情鼓励，我校八二至八六各年级软件硕士生宫一兴等提出了宝贵的意见，华中理工大学出版社、计算机科学与工程系及软件教研室的老师们给予了大力的支持，在此，我们一并表示衷心感谢！

由于我们的水平有限，难免有错误和不妥之处，敬请计算机科学界的同行、老师们和广大读者批评指正。

作者

1989.2

目　录

第一篇 绪 论

（一）程序设计方法学及其重要性

一、程序设计方法学

在计算机诞生的初期，计算机硬件是人们考虑中的重要因素。当时的程序都很简单，而且问题也都是适定的，因此人们对于如何进行程序设计这一问题，既没有加以认真的考虑，也没有做深入的研究。

60年代以来，在一些发达国家中，计算机的应用领域越来越广泛，它几乎涉及到了社会生活的各个方面。根据不同的应用要求，相应地产生了不同的软件系统。这些系统都相当庞大，逻辑复杂，而且功能上也需要不断更改和扩充。

人们发现，尽管研制大型软件系统需要投入大量的人力和物力，但系统的质量却难以保证。由于程序设计是一种困难的高级智能活动，既难于对它进行控制，又缺乏统一的规律可循，因此事前很难预料成功与否，而事后欲证明其正确性也并非易事。随着程序问题的增大，程序设计也就越来越困难。两个著名的例子是IBM公司的OS/360系统和美国空军某后勤系统，这两个系统都花费了几千人年的努力，历尽艰辛，但结果是令人失望的。OS/360系统的负责人Brooks生动地描述了研制过程中的困难和混乱：

“……象巨兽在泥潭中作垂死挣扎，挣扎得越猛，泥浆就沾得越多，最后没有一个野兽能逃脱淹没在泥潭中的命运……程序

设计就象这样一个泥潭……一批批程序员在泥潭中挣扎……没有人料到问题竟会这样棘手……”。[1]

于是人们认识到了程序设计确实存在问题，当时人们称之为“软件危机”、“程序设计危机”。[2]危机迫使人们重新开始研究程序设计的一些最为基本的问题：人们应该怎样进行程序设计？程序设计背后有什么理论？有什么原则性的东西？

E.W.Dijkstra在为 David Gries 著的《The Science of Programming》一书而写的序言中指出：“近10年来，‘程序’一词的潜在意义已发生了深刻的变化”。传统的（或者说习惯性的）程序设计方法已不能适应发展着的形势的需要，必须揭示隐藏在程序设计背后的理论和带原则性的东西，用以指导当今的程序设计。按David Gries教授的说法：“我们正到达一个可以谈论程序设计科学的阶段”。

当今的程序设计除要求程序逻辑正确、能被机器理解并执行外，还涉及到程序性质表示、程序的可读性、可靠性、可维护性以及程序的效率等等方面。为了改善整个程序设计过程，使之更加科学化、规范化，这就需要一整套关于程序设计的理论和方法。

程序设计方法学是研究程序设计的理论、原则和方法的科学，也称之为程序设计科学或程序设计学。

程序设计方法学的基本的主要目标有：①有规则地推出关于研制可读的、正确的程序的有效方法；②标识和说明求解程序设计问题（或其它有关问题）的技术和工具；③寻找程序设计时清晰地思考问题的策略。

程序设计方法学中最基本的原则有三个：

·抽象原则

为了解决一个复杂的问题，人的智力往往不可能一下子就触及到问题的细节。一般解决问题的过程是，在分析了问题的要求之后，先设计出一个抽象算法，在抽象数据上实施一系列抽象操

试读结束，需要全本PDF请购买 www.ertongbook.com

作。这些数据和操作反映了问题的本质属性，而将所有的细节都抽象掉了，然后再考虑这些抽象数据和操作的具体实现。在抽象级只要知道“做什么”，在实现级才考虑“如何做”。总之，由于抽象技术的采用，使大而复杂的问题分解成若干相对独立的子问题。这些子问题只涉及局部的环境和条件，独立地一个个地解决它们（David Gries称作“分而治之”），从而使整个问题得到圆满解决。

·枚举原则

程序中的条件选取结构是枚举原则的应用。

·归纳原则

程序中的循环重复结构是归纳原则的应用。

抽象、枚举和归纳，是人们通常进行思维的方法，也是程序设计的基本原则。

程序设计方法学与其它学科一样，是一门内容丰富、思想方法深刻而又有其自身理论体系的学科。它既有同数学一样的抽象性与严密性的特点，又有应用的广泛性与实际试验的高度技术性的特点。要掌握这门学科，必须自己亲临其境，亲自动手，在自己设计的系统中实际应用。David Gries教授指出：“这是一个数学性很强的领域，它需要程序设计工作者在数学方面、逻辑方面受过一定的教育，同时还有不少约束性的原则要老老实实地遵循。要能强迫自己用这种方式思维，确实还是不容易的。”[2]

二、研究程序设计方法的重要性

由于大规模集成电路的迅速发展，计算机硬件生产自动化程度相应提高，而软件生产却长期处于“手工”状态，在效率、质量等方面，都远远不能适应需要，因此世界市场上，软件、硬件成本的差距越来越大，对软件生产自动化的要求也就越迫切。改善程序设计过程，使之科学化、规范化、工程化，从而提高程序设计自动化程度，提高程序生产率和改善程序质量，这也是软件

工具和环境研制的目的。

方法和工具是同一问题的两个方面。方法是工具研制的先导，工具是方法的实在体现，程序设计方法研究的成果要最终实现为软件工具，由此可见，程序设计方法学研究的重要性。目前人们致力研究的“软件工程环境”（简称“环境”）正是方法和工具的结合。从事软件工具和环境的研制，既要从理论高度来探讨程序设计的一般规律，又要善于将方法中的理论、技术的成果实现为具体的有实用意义的工程系统。可以预言，“方法”的深入研究必将对计算机科学和工程本身乃至其它各个科学领域产生深远影响。

从计算机教育的角度来看，程序设计方法学将成为我们培养程序设计专门人才的必不可少的一门课程。虽然不学程序设计方法学也可以编程序，但是这样的程序设计既费劲，又易出错。David Gries教授曾作过一次试验，他邀请了若干名研究生和一些来自工业界的专业人员一块解决一个问题（行向右对齐问题，参看12·1节），结果竟有一半人编的程序是错的，而Gries本人使用新的程序设计方法，编制的程序不仅正确、结构清晰，而且比其他所有人的程序更为有效，所花的时间并不比其他人多。这个例子对程序设计方法学的重要性是一个有力的说明，因此，应该从根本上屏弃原有的程序设计的旧观点和旧习惯，以程序设计方法学中揭示的“新方法”来指导程序设计。

（二） 几种程序设计方法简介

回顾程序设计发展的历史，可以看到，程序设计经历了从艺术到科学的曲折历程。程序设计方法学是一门非常年轻、但发展极其迅速的学科，仅10多年的时间，已开始走向成熟，涌现了许多新颖的、值得人们深究的程序设计方法。这些方法在不同的应用中不仅产生了实际效果，而且向人们展示了具有深远意义的研

究方向，并提出了很有价值的课题。由于篇幅所限，我们只能简单介绍目前最有代表性的几种方法。

一、结构程序设计方法（简称SP方法）

结构程序设计是程序设计方法学的一个重要组成部分。

“结构程序设计”的概念是由E. W. Dijkstra于1969年提出来的。它强调从程序结构和风格上来研究程序设计。Tony Hoare对结构程序设计作了以下确切的定义：

“结构程序设计是，在合理的时间内，用所引出的方法，组织人们的思想，书写一个正确的可读的程序，达到计算任务具有可理解的表达格式。”〔3〕

这个定义指明，SP方法的任务是组织人们的思想，目的是达到程序的可读性和程序的正确性。

采用结构程序设计方法，关键在于程序正确性证明。本书将介绍的程序设计“新方法”，其要点在于：正确性证明与程序编制同时进行，而且以正确性证明为主导，从而保证所得到的程序的正确性。

SP方法建立在Böhm、Jacopini证明的“任何程序逻辑都可用顺序、选择和循环等三种基本结构来表示”的定理基础上，并由这三种基本结构的反复嵌套构成“结构化程序”。经过几年的探索、争论和实践，它的应用确实收到了实效，是目前日趋完善且使用较为广泛的方法，而且，大多数高级语言都支持SP方法。用结构程序设计方法编写的程序不仅结构良好，易于阅读，且易于证明其正确性。

由于受到了上述约束，用SP方法编写程序时，需要更多的思索，或者说“没有考虑思维的节省”；此外，SP方法有时用降低程序效率来换取程序的可靠性，但对于某些有特殊要求的程序，效率问题又是至关重要的。

在此声明一下，有时人们将三种基本结构的应用、自顶向

下、逐步细化、主程序员组等一系列技术统称为结构程序设计[4]。

二、逐步求精的设计方法

逐步求精是抽象原则的应用，是处理复杂问题的重要工具。为解决一个复杂的问题，开始时，只能对问题的全局作出决策，从问题的体系结构、求解策略和要求出发，设计一个对问题本身较为自然的、很可能是用自然语言表达的抽象算法，这个抽象算法由一些抽象数据及其相应的一些操作（即抽象语句）构成。一个这样的抽象算法，仅仅表示解决问题的一般策略和问题解的一般结构。

对抽象算法作进一步求精，而进入下一层的抽象。在求精过程的每一步，抽象语句和抽象数据都将进一步分解和精细化。如此继续，一直到最后的算法能够用某种程序设计语言明确地、完整地描述出来为止。为了直观地了解逐步求精的设计方法，下面看一个例子。

例1 设有不全为零的整数X、Y，求它们的最大公因数$\gcd(X,Y)$。

第一步，分析问题，提出全局性的求解策略。

由于函数$\gcd(x,y)$具有如下性质：

$$\gcd(x,y)=\gcd(y,x); \tag{0.2.1}$$

$$\gcd(x,y)=\gcd(-x,y); \tag{0.2.2}$$

$$\gcd(x,0)=|x|。 \tag{0.2.3}$$

因此，可假设X、Y为非负的不全为零的两个整数，且仅对Y进行讨论。

若$Y=0$，则由(0.2.3)立即可得：

$$\gcd(X,Y)=X;$$

若$Y>0$，将Y减少，但保持gcd不变，重复这个动作，直到$Y=0$为止。

根据上述讨论，可设计抽象算法如下：

(0.2.4)　(1) 引进变量x,y，置初值：

置x为X，置y为Y。

(2) 重复施行下列步骤：

(a) 若$y=0$，重复停止，x即为所求；

(b) 若$y>0$，按一定的方式减少y，改变x，保持x、y非负，且$\gcd(x,y)=\gcd(X,Y)$。

这样，问题被分解为两个子问题，其中子问题（2）是一个重复过程，每次重复执行（a）、（b），虽然x,y值发生了变化，但关系式

(0.2.5)　$$\gcd(x,y)=\gcd(X,Y)\wedge(x\geqslant 0)\wedge(y\geqslant 0)$$

却永远保持不变。我们称这样的关系式为循环不变式，它对于循环的设计和证明都是极其重要的。

第二步，求精算法(0.2.4)中子问题（2），即如何减少y和改变x，而使式(0.2.5)保持为真。解决这个问题有各种不同的策略，因而也就有将它进一步分解的不同方法。采用欧几里得算法，可得：

(0.2.6)　$$\gcd(x,y)=\gcd(y,x \bmod y),$$

且

$$0\leqslant x \bmod y<y,$$

其中$x \bmod y$表示y除x所得的余数。

关系式(0.2.6)提供了一种“减少y和改变x”且保持gcd不变的方法。引进变量r，算法(0.2.4)中的子问题（2）可进一步分解为：

置r为$x \bmod y$；

置x为y；

置y为r。

这样，算法(0.2.4)在如此求精后变为如下算法：

(0.2.7)　(1) 置x为X，y为Y；

(2) 重复执行下列步骤：

(a) 若$y=0$，停止重复，则x为所求；

(b) 若$y\neq 0$，则

置r为$x \mod y$，

置x为y，

置y为r。

算法(0.2.7)中的(2)必须在有限次循环后终止。事实上每重复执行一次，原来的y改变为$x \mod y$，y单调减少，但y又保持非负，所以这样的循环必在有限步之后，因y减少为0而终止，出口处

$$\gcd(X,Y)=\gcd(x,y)=x,\quad (y=0)$$

如果在程序设计语言中有求余操作，那么(0.2.7)便是最后求得的算法了。否则，还要进一步求精“$x \mod y$”，得到求$x \mod y$的算法。

至此，设计求gcd的算法的任务已告完成。设计中所采用的方法是，将整个问题分解成若干相对独立的子问题，只要这些子问题得到正确解决，整个问题也就解决了。重复这种分解和证明分解正确的过程，直到每个子问题都已简单到足够满意的程度。每一步分解都要作出分解方法的决策，不同的决策就会导致不同的算法。

由此可见，逐步求精是一种自顶向下的设计方法，从最能反映体系结构的概念出发，逐步精细化、具体化，逐步补充细节，直到设计出可以在机器上运行的程序。

自顶向下的方法有时效率不太高，因为逐步加细是逐步深入的过程，往往免不了要回溯，即对原来已决定的某些步骤加以修改。这时，必须采用由底向上的方法来对原先的设想进行改错、补充、加工和优化。因此，逐步求精过程应理解为一种自顶向下并且不断地用自底向上修正或补充的过程。

试读结束，需要全本PDF请购买 www.ertongbook.com