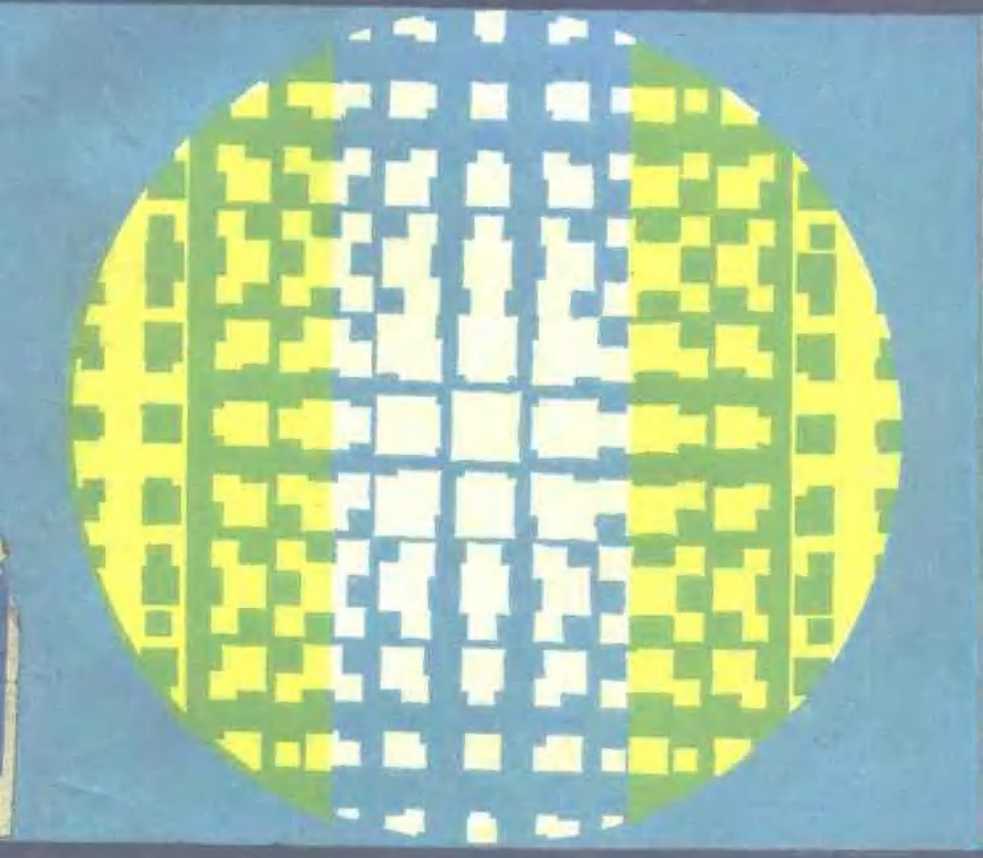
冯玉琳　仲萃豪　陈友君　编著

程序设计方法学



北京科学技术出版社

程序设计方法学

（第二版）

冯玉琳　仲萃豪　陈友君　编著

北京科学技术出版社

内 容 提 要

本书论述了程序设计方法学的科学原理和方法，把那些公认为成熟的、有实用价值的、最基本的内容收入本书，并适当地介绍了一些尚待进一步发展的研究课题。本书主要内容包括程序结构分析、逐步求精、递归程序设计、数据抽象和模块、程序规范、程序的形式推导和程序变换等，通过大量程序实例，寓理论于实际的程序设计技术之中。本书主要章节后面附有习题。

本书可供从事计算机工作的科技人员阅读，也可作为大专院校计算机专业学生和研究生的教科书或参考书。

程序设计方法学

（第二版）

冯玉琳　仲萃豪　陈友君　编著

北京科学技术出版社出版

（北京西直门外南路19号）

新华书店首都发行所发行　各地新华书店经售

北京市怀柔平义分印刷厂印刷

787×1092毫米　32开本　11.25印张　248千字

1985年1月第一版　1989年8月　第二版第二次印刷

印数　1—3609册

ISBN 7-5304-0470-9/T·90　定价：4.40元

第二版序言

本书是1985年北京科技出版社出版的“程序设计方法学”的修订版，内容作了适当增补，文字亦经过反复加工。书末给出了部分习题的参考答案和提示。

本书第一版问世之后，国内许多大专院校用它作教材或教学参考书，并提出了不少好的建议和宝贵意见，对这次修改定稿帮助很大。值此本书再版机会，特向关心本书的同志们表示衷心的谢意。

编 著 者

1988年3月于北京

序　言

程序设计方法学是一门非常年青而发展又极其迅速的学科，仅仅才十多年时间，已经取得了一系列重大的进展。我们编写这本教程，旨在阐明程序设计方法学的科学原理和方法，并按教学要求，尽可能地把那些公认为成熟的、具有实际应用意义的最基本内容收入书中，还适当地介绍了一些尚待进一步发展的研究课题。

全书分为三部分，共12章。

第一部分为1～4章，介绍程序的基本结构和程序设计的基本知识，并由此熟悉本教程所使用的程序设计语言T。

第二部分为5～8章，构成结构程序设计的基本内容，包括自顶向下的逐步求精、数据抽象和模块，以及递归程序的设计方法等。

第三部分为9～12章，初步讨论了形式化程序设计方法。第9章介绍VDM规范；第10章介绍程序的形式推导技术；第11章为程序变换技术；第12章论述了研制程序工具和环境的重要意义。这几章的内容都是有待进一步深入研究的课题。

书后的附录1是程序设计语言T的文本。

本书内容侧重于程序设计科学，并寓理论于实际的程序设计技术中。书中给出了大量的程序设计实例，使读者能从中体会并掌握基本的程序设计原理和方法，并灵活应用到自己的程序设计活动中去。

有关程序的规范、推导和变换是形式化程序设计的研究课题。实际上，程序的规范和推导技术就是前几章结构程序设计的继续，把程序设计提高到更加数学化、科学化的水平。为此，我们在介绍了数据抽象和模块之后，专辟VDM一章进一步阐述模型规范技术；我们将D.Gries的近著“程序设计科学”一书的内容取精，集中在第10章加以介绍；书中关于程序变换的内容取自F.L.Bauer的研究工作。虽然程序变换技术还未付于实用，但它已成为本世纪程序设计自动化研究的重要课题，可以使我们思路开阔，有助于指导实际工作。预计不久的将来，它将会成为程序设计的重要方法之一。

除本书所列内容之外，程序设计方法学还应包括程序语义学、程序设计模式、程序逻辑、程序代数、并发程序设计等等。本书未列入这些内容，一方面是为了使教材适当集中，自成体系，风格保持一致；另一方面，我们不想使它成为一本理论性教材，而是希望它是一本锐有趣味的融程序设计的理论、技术和实践于一体的教程。

本书可供高等学校计算机科学系高年级学生或低年级研究生作为教科书或教学参考书，也可供同等学历的软件技术人员自学之用。

我们假定读者已学过数据结构和程序设计语言（如Pascal），具备数理逻辑的基本知识，懂得代数结构和集合的一般概念，并有程序设计的初步经验。

讲授本教程约需40～60学时，各章节介绍的内容顺序可根据学生的情况加以改变，实例也不一定全部讲解。对计算机系的学生，前三章的内容可以不讲，而另外增加补充材料。

书中各章后均附有练习，以此作为正文内容的补充。其中有部分练习比较难，希望教师给以提示或进行集体讨论。附录2列出部分习题的答案和提示。

作者真诚地感谢袁崇义同志仔细地审阅了本书的全部手稿，感谢樊哲民同志提供了本教程第11章的部分内容。本书内容曾在中国科学技术大学、中国科学院研究生院、北京大学、清华大学、武汉大学、北京工业大学、西安交通大学、郑州大学等十多所院校讲演过，得到所在单位师生的支持，提出了许多宝贵意见，这里一并表示感谢。

限于经验和水平，书中难免有错误和不妥之处，恳切希望计算机科学界的同行和读者们不吝指教。

编　著　者

1984　年

目　　录

第1章 引 论

1.1 什么是程序设计方法学

回顾程序设计发展的历史，大体上可划分为三个不同的时期。

50年代的程序都是用指令代码或汇编语言来编写的。这种程序的设计相当麻烦，编制和调试一个稍许大一点的程序常常要花费一年半载的时间。在早期讲授程序设计的时候，总是用一台机器的指令系统或它的汇编语言作为抽象机，然后在它上面讲授如何设计计算公式、分叉、循环和子程序等程序设计知识。当时评论程序的好坏标准，就是看它能否作到指令条数少，存储单元省，执行速度快。按这种教学方式，培养一个熟练的程序员要经过长期的训练和实习，这种局面严重影响了计算机的普及应用。

FORTRAN语言的出现，改变了这种局面。1958年，FORTRAN语言刚被使用时，本身还存在很多缺陷，如编译时间长、质量差、错误多等等，但是使用高级语言大大简化了程序设计，缩短了解题周期，因此显示出强大的生命力。此后，编程序不只是软件专业人员才能做的事了，一般工程技术人员花上几周时间学习，也能使用计算机解题。

随着计算机的应用日益广泛地渗透到各学科和技术领域，60年代发展了一系列不同风格的、为不同对象服务的程

序设计语言，总数竟达三四百种之多。其中较为著名的有ALGOL、COBOL、LISP、SNOBOL、PL/1、APL和PASCAL等十多种语言。高级语言的蓬勃兴起，使得编译和形式语言理论相应地日趋完善，这是该时期的主要特征。但就整个程序设计方法而言，并无实质性的改进。

自60年代末到70年代初，出现了大型软件系统，如操作系统、数据库，这给程序设计带来了新的问题。大型系统的研制需要花费大量的资金和人力，可是，研制出来的产品却是可靠性差，错误多，维护和修改也很困难。一个大型操作系统有时需要几千人年的工作量，而所获得的系统又常常会隐藏着几百甚至几千个错误。当时，人们称这种现象为“软件危机”。

“危机”震动了软件界，程序设计的传统习惯和工作方式导致了不清晰的程序结构，使得程序可靠性很难保证；另一方面，程序设计工具的严重缺乏也使大系统开发陷入困境。

追根寻源，人们开始重新研究程序设计中的一些最为基本的问题。例如，程序的基本组成部分是什么？应该用什么样的方法来设计程序？如何保证程序设计正确？程序设计的主要方法和技术应如何规范化等等。

1969年，E.W.Dijkstra首先提出了结构程序设计的概念，它强调了从程序结构和风格上来研究程序设计。经过几年的争论、探索和实践，结构程序设计的应用确实取得了成效，用结构程序设计的方法编出的程序不仅结构良好，易写易读，而且易于证明正确性。

结构程序设计方法的推行，其意义远不止于当前的实用价值，更深远的意义在于它向人们揭示了研究程序设计方法

的必要性。程序设计并不纯粹是一种技术性的活动，与任何学科一样，它自身也有一套基本的原理和方法。70年代程序设计技术的发展是以方法论的研究为特征的。除结构程序设计的方法日势完善外，在数据类型抽象、程序的推导和综合、程序变换和程序自动化等方面，都取得了重要的进展。

80年代，人们把注意力集中到程序设计环境的改善方面。程序工具和环境的研制，从理论高度探讨程序设计的一般规律，同时将方法学的研究成果实现为有实用意义的工程系统，在提高程序生产率和改善程序质量方面迈开了一大步。

科学总是在不断产生矛盾又不断解决矛盾的过程中诞生和发展的。软件危机促使程序设计自身的一些理论和方法得以完善和进步。60年代，一个程序员只要能正确理解程序设计语言的意义，掌握程序设计的一些基本技巧，经过一定的实践，就可以编制适应当时要求的程序来。学习程序设计，也是多采用师付带徒弟的办法，在编程的具体实践中摸索。因此，那时候，人们常把程序设计视作一门“艺术”。随着70年代和80年代关于程序设计理论和方法学研究的不断深入，人们总结出一整套的关于如何进行程序设计的原则和方法，这里不仅包括结构化程序设计，而且还涉及到程序语义性质的表示、程序正确性、程序的形式化开发和程序的效率等许多方面。这就使程序设计方法学作为计算机科学的一门前沿学科应运而生了。程序设计方法学的目标在于改善程序设计的过程，使之更加科学化、规范化。

程序设计方法学的几个基本原则是很富有哲理性的。抽象、枚举和归纳，这是人们通常进行思维的方法，也是进行程序设计的基本原则。

抽象是程序设计的一个最重要的基本原则。为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方面。在分析了问题的要求之后，我们往往首先将问题抽象为模型，再确定其抽象算法。也就是：略去细节，抓住问题本质属性，确定抽象数据，实施一系列抽象操作。然后，作为下一步，再考虑这些抽象数据和操作的具体实现。在抽象级，我们只要知道“做什么”，而在实现级，我们才考虑“如何做”。由于抽象技术的采用，使大问题分解成了相对独立的一些子问题，它们分别只涉及局部的环境和条件，可以独立地一个个地被解决，从而使整个问题得到圆满解决。

枚举和归纳是程序设计的另外两个基本原则。对于有限种情形的分析和计算，我们可以用一一枚举的办法，程序中的条件选取结构就是枚举原则的应用；对于不能一一枚举的不定数目情形的分析和计算，我们常用归纳的办法，循环重复结构和递归过程就是归纳原则的应用。

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。有人认为，以前不学程序设计方法学，不是也能写出程序吗？美国康奈尔大学D.Gries教授曾做过一次试验。他邀请了40～50名研究生和一些来自工业界的专业人员和程序员一起来解决一个问题（Justification问题）。结果竟有半数人，程序编得不正确，而Gries本人使用了方法论中的一些行之有效的技术，写出的程序不仅结论正确、结构清晰，而且比所有其他人的程序更有效，编写程序的时间也并不比其他人长。这个例子说明：虽然不学程序设计方法学也可以编写程序，但是这样的程序设计既费劲，又易出错，这当然不是我们所希望的。因此，要下决心对程序员进行程序设计方法学的教育，

以便从根本上摒弃原有的程序设计的观点和习惯。

程序设计发展至今，要求一个称职的程序设计人员必须具有两方面的素质：一方面要掌握程序设计的基本原理和方法，另一方面要积累程序设计的丰富实践经验。只有这样才能在程序设计的广阔领域里充分发挥创造性的智慧和才能。

在程序设计方法学的发展过程中，常常因对一些问题的看法不一致而引起争议。针对同一个问题，有时因为概念上理解的误差，往往就会带来不同的结论。为此，在进入本教材正文之前，先讨论一下几个代表性的有争议的概念。

·关于小规模程序设计和大规模程序设计

与任何大型软件系统比较，任何一本程序设计方法学的教程中所举的示范性实例，都只是所谓小规模程序。人们之所以着重于小规模程序的训练，这是因为：只有学会有效地研制小的程序，才能期望有效地研制大的程序或程序系统。

这个观点曾得到E.W.Dijkstra的有力论证。假设一个程序是由n个小部分组成，每部分正确性的概率是p，那末，整个程序正确性的概率P一定满足

$$P<p^n$$

当n足够大时，如果我们希望P是个接近于1的有意义的值的话，p就必须非常接近于1。

另一方面，Dijkstra也告诫人们，不要低估“量”的因素。如果写1000行程序需一个人月，那么写10000行程序是否需十个人月就够了？事实是：当程序的规模从1000行增加到10000行，虽然代码行增加到十倍，但是程序复杂程度却远不止十倍。因此，为了进行大规模程序或系统的研制，仅仅凭借已有的小规模程序设计的经验是不够的。

尽管如此，小规模程序设计的原则和方法，如模块化方

法、自顶向下的逐步求精等，对于大规模程序仍然适用，而且更为重要。大规模程序设计，归根到底要分解为一块块的小程序。设想如果小程序尚且编不好，如何能够安排组织一个大程序设计呢？

·关于证明程序的正确性

证明自己写的程序正确，这应是程序员义不容辞的职责。所谓证明，就是使自己和别人确信一个断言真实性的论证。这种意义下的证明并非一定指从公理出发的形式推导，更非机械证明。正如D.Knuth所指出的，一个正确性证明，实际上是指一种理解，这种证明，可以是形式的，也可以是非形式的。

自从1967年R.Floyd提出赋给程序意义的概念以来，程序正确性证明的技术获得了很大进展，其主要标志是C.A.R.Hoare的公理化和E.W.Dijkstra的最弱前谓词推演。尽管这些研究目前尚难以工程化，但是，它们从不同侧面刻划了程序性质，对于程序设计方法学的发展，无疑起了很大的促进作用。

基于机械证明技术，现在也已发展有各种不同风格的自动证明系统或程序自动设计系统。这些系统的研制，对于程序设计自动化和程序设计方法学，显然是有益的贡献。但是，它只能是一种辅助工具，盲目地追求庞大的全自动性的证明系统，正如A.Perlis所说的那样，注定是要失败的。

·关于好结构和效率

所谓好结构程序，指程序结构清晰，易于理解，也必然易于验证的程序。好结构程序从效率上看，不一定是最好的程序。但是，它能提高程序的可靠性，便于检查和维护。结构程序设计的观点要求好结构程序。在70年代初，曾发生过要不要