软件设计基础

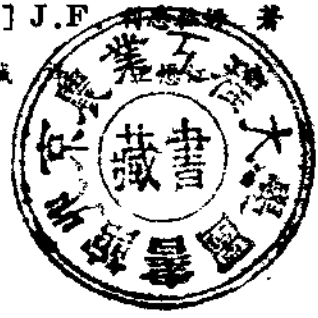
[美] J·F·利思拉姆 著

世界图书出版公司

软件设计基础

[美] J.F [illegible] 著

[illegible]



责任编辑　李宗慧

Foundations of
Software Design

J.F.Leathrum

Reston Publishing Company,Inc.

软件设计基础

〔美〕 J.F.利思拉姆 著
索诚 译　　王懋江 校

世界图书出版公司出版
（北京朝内大街137号）
新华书店北京发行所发行
赵全营印刷厂印刷
开本：850×1168 1/32 6.5印张
1987年9月第一版 1687年9月第一次印刷
ISBN7-5062-0018-X/Z·4
统一书号：17292.008
定价：1.90元

内 容 简 介

本书系美国大学和研究生培训教学用书。内容包括软件设计的理论、方法、要求分析、设计与测试、程序设计方法的实例研究等。

本书篇幅紧凑，叙述深入浅出。适合大学计算机专业、管理专业和工 程技术专业的高年级学生、研究生、教师和有关的科研、技术人员阅读。

译者的话

本书名为“软件设计基础”，实际上是最近几年在软件设计研究领域中有影响的学术论文的综合，因而它具有较高的学术价值，可供我国软件研究人员、软件设计人员和大专院校研究生参考使用。

在翻译此书过程中，译者遇到不少技术性细节难于处理。如“Delete module”中Delete一词，一方面在程序设计语言中是一个符号名，而另一方面有删除的意义，是一个删除模块的程序名。这样Delete一词就具有两重作用，而翻译时对类似这种词译不译成中文确实费了不少心思，不译吧，书中类似这种词确实不少，翻译成中文吧，又不太符合目前程序设计语言的习惯，即失去了作为符号名的作用，最后，译者考虑到目前我国中文信息处理系统的飞速发展，当前已有不少软件允许采用中文名字作为程序名、文件名和标识符名，故还是将类似Delete的一些词译成了中文。这样做是否妥当还请有关专家、学者指正。

鉴于译者水平有限，对原书的理解不够深刻，译文和一些名词的译法错误难免，敬请读者随时指教。

译　者

一九八六年四月

前　言

本书是一本大学教材，供计算机科学系、管理科学系和工程系的高年级大学生和研究生使用。书中收集了几种刊物中已发表的一些文章。目的是把这些文章有效地组织起来，编成单行本，以适合课堂使用。很显然，“软件设计”这一课题应当受到学术界的注意，因为它已经深刻地改变了我们对科学计算和程序设计的旧观念。

这个课题称为“软件设计”再恰当不过了。有时它也叫做软件工程、软件可靠性或者简称程序设计。本书主要介绍程序设计中涉及到人的因素和社会方面的问题，但就建立软件的工具、软件设计方法学和关于软件的观察分析而言，软件设计本身显然是个技术问题。

虽然本书讨论了程序设计方法学，但是作者不打算只局限于此。如果想采用本书作为一年课程的教材，同时还想进一步深入讲授程序设计方法学，那末就需要补充一些其它教材。

课题中所具有的数学味道的多少是因题目而异的，这也正是软件设计的特点。这有助于了解大多数其它设计原则都具有的这种特点。无论怎么说，不能只因为数学基础不成熟，或者技术的定量基础是建立在纯经验知识基础之上的，就说一个工程师回避设计上遇到的矛盾是对的。在软件设计中，很突出的一点是讨论要求说明书时，缺乏数学味道（见第三章）。在测试技术领域中（见第七章）很明显地看到急需发展数学基础，而在研究程序设计方法学方面显然具备了雄厚的数学基础（见第五章）。

本书中的材料已在高年级大学课程和研究生课程中各讲授过一个学期。这两期课程都要求学生搞一个“软件设计”项目并写出报告。速成研究生课程通常还要补充一些程序设计方法论的材料。研究生班如果采用本书作为主要教材，则建议加上一门多处理机系统的程序设计方法学课程。

作者在编写这本书的过程中，得到了图伦大学、特拉华大学和克莱姆逊大学的大力支持，特表示感谢！对这些大学已经使用本书初版学生的耐心和理解力表示感谢。特别要感谢C.考尔卡特(C.Calcutt)夫人，她帮助打印了最后一稿，对帮助准备例子和练习的R.富尔布赖特(R.Fulbright)、M.吉尔哈特(M.Gearhart)和W.戈弗雷(W.Godfrey)先生也表示感谢。

J.F.利思拉姆

目　录

1.1 软件经验

软件技术发展到目前这样水平是受到了技术和经济方面种种压力所推动的结果，这些压力既有历史的也有当前的。自从数字电子计算机问世以来，人们对程序(软件)设计及其实现的态度和采取的经济刺激手段发生了根本的变化。早期，计算机的存储器似乎是限制程序员的最主要的设备。程序员和机时也是当时要考虑的重要问题，但是由于当时对程序员和计算机的需求量不大，所以还不是太受限制的条件。(要想体会早期软件经验的一个简便方法，是为现有的任何一种可编程序袖珍计算器单独编写一个程序）程序员在当时情况下的主要奋斗目标是要把代码编写得"紧凑"，因为代码本身不允许占有很多内存空间。由于程序编得都不很大，所以程序的基本结构很容易看懂，了解起来不会很困难。

随着磁芯存储器的出现，内存容量对程序的限制很快就缓和了。但是，计算机执行时间仍然是一个严重的限制，只有到后来计算机技术取得进展，问题才得到改善。对程序讲究"时效"，也是促使"紧凑"代码产生的一个原因。在这种压力之下，为了节约几个机器周期时间，人们过分热衷于程序结构。程序设计被看成是几个小程序合成一个完整程序的过程。初期使用的编译程序这个词就代表了这种观点。

练习 1-1

编写一个计算机程序，利用泰勒级数近似法，逼近cos（θ），$\theta=45^\circ+i$，$i=0, 1, 2, \cdots\cdots$。假设总的存储容量限制在100个单元。每条指令和每个数需要一个存储单元。用典型的计算机指令系统，但是不用浮点指令。(注：这是作者在1955年给第一学期的微积分班留的第一个程序设计作业。用的是马里兰州阿伯丁试验场的EDVAC计算机)

事后人们认识到，程序员早就应该避免初期的软件缺陷，特别是因为许多程序员本身就是工程师。然而，一门技术的发展总要经历这样的过程，这是常见的事。程序设计的这个初期阶段是必不可少的。这一点甚至能够用修造桥梁一类的技术经验来证明。后来IBM公司开发了360系列计算机用的OS-360操作系统，才真正推动软件走向复兴时期。为开发这个软件，IBM公司投入了几千人年。对这个系统的性能评价是有争议的，但是，一致认为它从来就没有达到原计划要求。OS-360操作系统维护困难和维护费用高是毫无疑问的。OS-360操作系统的经验可以算作是个反面教材，以致在软件现象学的许多讨论中成了被引用的典型例子（见第二章）。

几乎在OS-360系统软件开发的同时，在荷兰由E.W.迪克斯特拉（E.W.Dijkstra）领导的一个小组也在开发THE（Technische Hogeschool of Eindhoven)操作系统。THE的研究成果主要表现在软件方法论方面（见表1—1）。它导致了迪克斯特拉象明星一样突然出现在软件学术界，解决了OS—360操作系统的问题。他单枪匹马地改变了程序设计和软件设计的方向。

回顾过去，纵观1968年到1978年这十年，必定认为这是软件工程发展的非同寻常的时期。在这之前，人们的兴趣主要集中在研究各种各样的程序设计工具、程序设计语言和方法论上。当时一般的见解是程序越多、越大就越好。这种观点的最好例子是PL/I大型程序设计语言和试验可扩充的程序语言。虽然后者对软件方法论影响很小，但还是受到这种观点的影响，认为程序设计工具越多，方法学就越能得到改善。对实习程序员和程序设计团体来说，实行详细分析已成为它们的职业术语。曾经有这样一个趋势，即任何问题都可用愈来愈多的专用工具解决，这些工具是适合软件日益细微化观点的（甚至程序员的能力测试也反映了这种细微分析的关系）。

表1—1 软件技术大事纪

软件名称	设计者	软件设计和实现的主要活动时期	技术意义
Fortran编译程序	国际商用机器公司(IBM)	1954—1958	第一个高级程序语言工具。
B-5500Algol	宝来公司(Burroughs)	1961—1964	设计的计算机考虑了软件的兼容性；Algol用作“系统程序设计语言”。
THE	迪克斯特拉(Dijkstra)	1963—1965	结构程序设计的先驱者。
OS-360	国际商用机器公司(IBM)	1963—1967	第一个失败的重大软件设计项目。
PL/1编译程序	国际商用机器公司(IBM)	1963—1968	用高级语言建立了一大批软件设计工具。
MULTICS	麻省理工学院(M.I.T)	1955—1969	综合性的硬件-软件系统设计。这是有学术界和工业界参加的一个联合计划。
Safequard	美国国防部	1972—	提高了军事部门对软件费用高的认识。
Ada	美国国防部	1980—	是建立软件设计工具方面的一个新起点。

从1968年到1978年这个时期是对软件工程是否应优先考虑充满怀疑和争论不休的时期。小而简单的软件引起了研究人员和专业人员的注意。UNIX①操作系统和程序设计语言“C”就是这个时期最著名的成果。UNIX设计者充分阐述的原理是：

对许多人来说，UNIX系统体现了舒马赫(schumacher)的格言，“小的就是美的…”。〔摘自MPT78，P.1890〕*

在这个时期发展的方法学也使用了少量的积木构件。构造程序被认为是一个抽象化过程，即抛开构件各自互不相干的功能不管，而把它们抽象地拼成一个程序。那时衡量软件系统优劣的标准是看软件是否简单和软件设计是否完整，而不是看使用程序设计工具的多寡或使用不同功能程序语言的多少。

从这些互不相让而又相互矛盾的发展趋向来看，重温一下迪克斯特拉原先提出的设计原则是值得的。这一设计原则可概括为：

1.各种正确的程序结构要求各种程序在智力上是可管理的。

2.实现人的智力管理的关键是程序本身的结构。

3.经过培训，程序员运用几个积木构件就有助于保证程序的正确性。

迪克斯特拉对程序设计方法学的发展所做贡献是多方面的。其中有两件工作特别有影响。第一，他写了一篇题为“GO TO语句是有害的”(Di68**)文章，从而揭开了GO TO论战的序幕。至于他本人的观点，就在这篇文章里讲明了。但是文章的题目和中心则导致了将争论转向有关GO TO语句是否应加到程序设计语言中的问题。第二，他对“结构程序设计”的贡献非常透彻地表明了他

①UNIX是贝尔实验室的商标。

* “MPT”是M.D.McIlroy,E.N.Pinson,和B.A.Tague三个人名的缩写，“78”是1978年，见参考文献。——译者注

** “Di”是E.W.Dijkstra的缩写；“68”是1968年，见参考文献。文中的这类注释以后不再说明。——译者注

对程序结构的观点（DDH72）。他个人的才能及其观点的雄辩力对以后的软件发展产生了不可估量的作用。但同时也应该公正地看到变革的时机就要到来了。要求发展一种新的方法学的经济和技术的压力是如此巨大，以致变革是不可避免地要发生的。

1968—1978年，正是这个时期，由于程序太大和容易出错才导致产生了迪克斯特拉的结构程序设计原理。程序设计的理论基础不断加强。许多有用的程序设计技术出现了。这种新的程序设计方法现已成熟，我们可以考虑用它对各级人员讲授。

1978年以后这段时期的显著特点是对新方法学尚未达到预期目的的一些反应。软件理论基础存在的内在缺陷找到了，而且还发展了一些相互有竞争力的方法学，它们各有所长，彼此补充（例如，验证方法学和测试方法学）。

自1978年以后，发生了另一个奇怪的现象，即存在着某种倒退，退到了软件工程1968年以前的时代。这种倒退的部分原因是由于高度集成化计算机硬件设计的影响，对此，软件方法学没有相应地发展。Ada程序设计语言（第四章，4.5节和附录A-2）是解决软件问题的最明显的一种尝试，但其本质上则代表着一种倒退，这真是令人啼笑皆非。尽管Ada作为一个软件设计工具提出了在此以前没有解决的问题，但是语言本身规模庞大并且由于积木构件太多而负担过重。

练习 1-2

在不使用GO TO语句的情况下，用Algol或PL/I这样的高级语言为巴布尔（Bubble）排序算法编写一个程序。引入的新变量在多大程度上克服了不用GO TO语句所带来的困难？

参 考 文 献

(DDH72) Dahl, O. J., E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming*. New York: Academic Press, Inc., 1972.

(Di68) Dijkstra, E. W., "Go To Statement Considered Harmful," *Communications of the ACM*, vol. 11 (1968).

(MPT78) McIlroy, M. D., E. N. Pinson, and B. A. Tague, "Unix Time-Sharing System: Foreword," *The Bell System Technical Journal*, 57 (1978), 1899-1904.

在程序结构问题出现的同时，出现了指导软件设计和维护的基本原理。能否用实验证实这些原理同样引起人们的兴趣。这种情况是物理学中多次出现的典型情况。在物理学中每提出一种一般理论，紧接着就有一个证明这个理论是否成立的实验阶段。通过现象学了解软件的方法是最近发展起来的，现在已经出现了一些有价值的见解。

2.1 软件生存周期

用一个典型的软件生存周期分析软件现象是有帮助的。据此可以验证观察到的量，而且能够把建立模型的活动集中在充分限定的范围内。最好按下面的顺序一步一步地观察生存周期：

1. 要求说明书
2. 软件设计
3. 编码
4. 测试
5. 维护

这种精心安排的划分将在后面的章节中仔细研究。因此就本节而言，只要非常简单地说明各个阶段就足够了：（1）要求说明书，包括阐明软件的目的：是做什么，而不是如何做；（2）软件设计，主要是把软件变成能够在计算机上实现的一些模块过程；（3）编码，就是把设计翻译成程序设计语言的过程；（4）测试，包括每个模块的功能测试以及整个软件包的整体测试两部分；（5）维护，是在完善软件的过程中和随着对软件要求的变化而产生的。

在按照这样一个生存周期观察、研究软件时，为了得到定量认识必须采用某些度量标准。很明显，生存周期的度量标准就是价格、错误率和时间。通过直接观察可以断定维护是生存周期最昂贵的阶段。这个阶段一般约占总费用的百分之七十(见图2-1)。

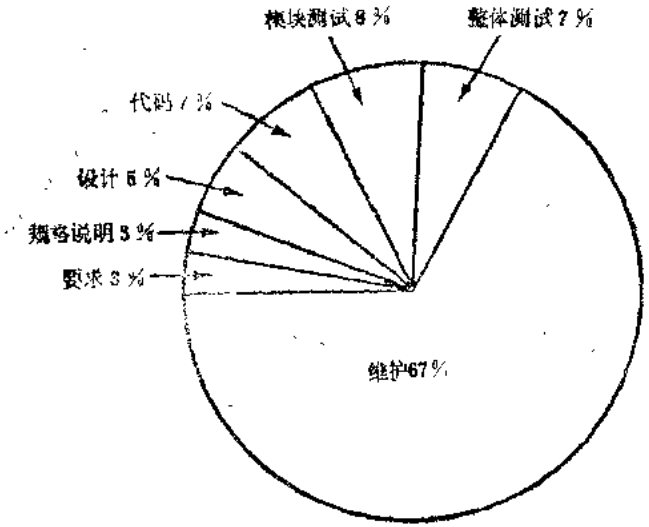


图2-1 在许多大规模软件系统中的真实工作量(摘自Ze78, P. 199)。

维护阶段包括消除在其它阶段所产生的错误，以及随要求的改变所要做的工作，如果不指出这一点，这个简单的观察很可能引起误解。这个技术问题首先是一个开发改正错误（测试和维护）的工具问题以及为避免出错（设计和实现）而有系统地阐述技术的问题。经验表明设计阶段是软件最容易出错的阶段，因而也是软件费用最高的阶段（BL79）。这里的设计应理解为包括生存周期中的要求说明和软件设计两个阶段。

软件生存周期的持续时间是略有变化的，并且对方法论中的微小变化是敏感的。软件计划，包括所有发行和修改在内，直到明显过时为止一般可持续六到十年。在这样的生存期间内，周期很可能重复几次。对于软件的全部或部分可能提出新的要求或新的设计，并且，每一次改动都可能遍历整个周期。因此，在维护阶段内，最好把生存周期看作是一些类似周期的循环。总的持续时间主要取决于在设计新模型之前所允许的循环总次数。