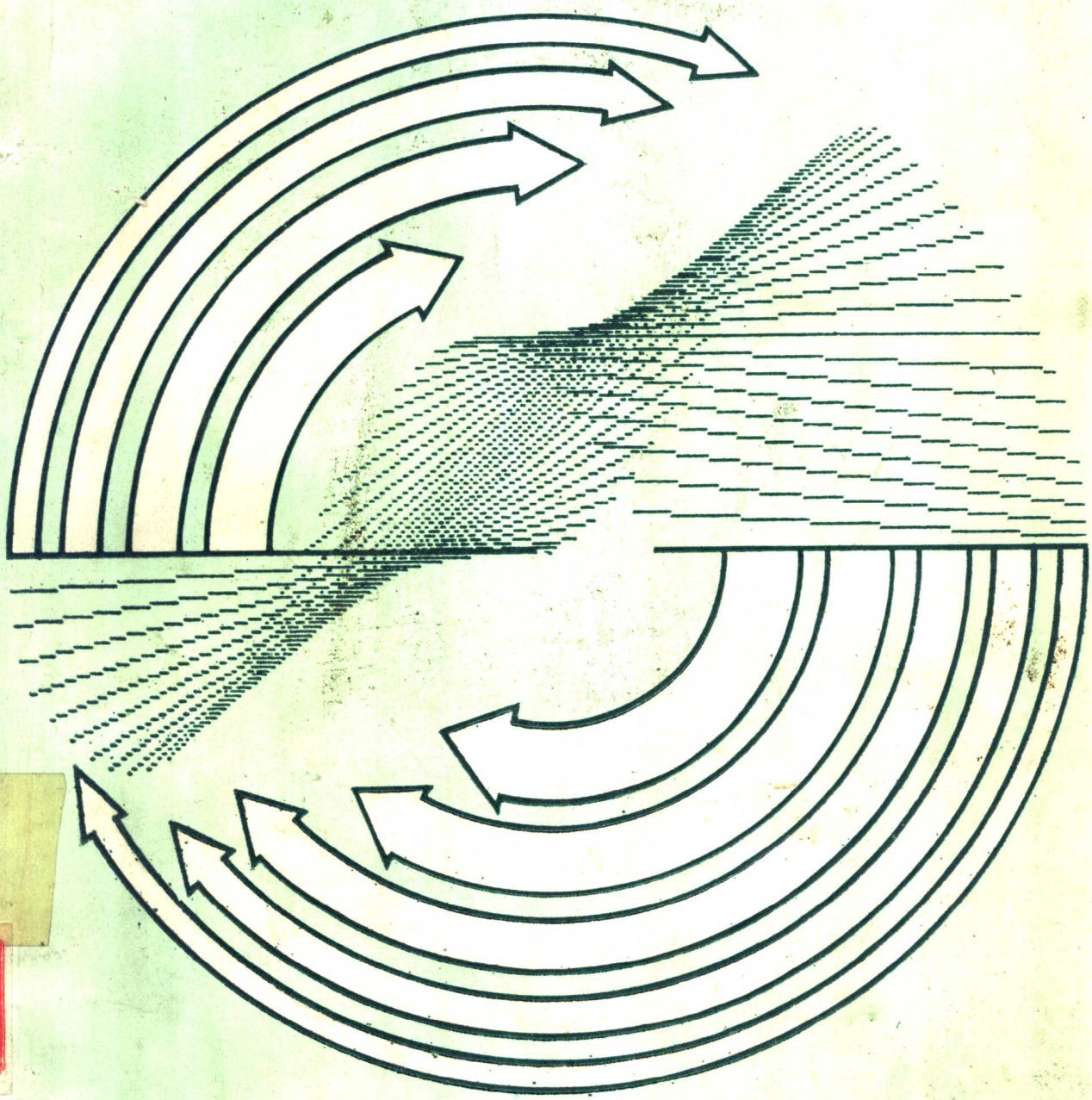
高等学校教学用书

软件工程学

[美] W. D. 哥莱特　S. V. 波拉克　著

鹿树理等　译　闻人德泰　校



電子工業出版社

软件工程学

〔美〕W.D.哥莱特　S.V.波拉克　著

鹿树理等　译

闻人德泰　校

電子工業出版社

内 容 提 要

本书以贯穿全书的实例，讲述软件工程学中最普遍使用的方法和工具。它包括：分析阶段的数据流图；设计阶段的结构图及伪码描述法；编码阶段的结构化程序设计方法；测试阶段的增量法、驱动块和存根块法等。

全书附有大量的习题，既是学生的作业，完成后又是实用的软件产品。

AN INTRODUCTION TO ENGINEERED SOFTWARE

Will D.Gillett　Seymour V.Pollack

Holt-Saunders

软件工程学

〔美〕W. D. 哥莱特　S.V.波拉克　著

鹿树理等　译

闻人德泰　校

责任编辑　邓又强

*

电子工业出版社出版（北京万寿路）

山东电子工业印刷厂印刷（淄博市周村）

新华书店北京发行所发行　各地新华书店经销

*

开本：787×1092　1/16　印张：14　字数：328千字

1991年3月第1版　1991年3月第1次印刷

印数：3000册　　定价：3.80元

ISBN 7-5053-1206-5/TP·194

译 者 序

计算机软件的发展，到60年代末产生了所谓的“软件危机”。为了解决这一问题，提出了“软件工程”的概念，并形成了软件工程学这一新兴的学科。

软件工程学是想借助于传统工程设计的一些基本思想、方针和工具，寻求开发和生产大型软件的系统化的方法和工具，以减少软件结构、软件管理的复杂性，提高软件的品质。

二十年来，已经出现了一系列行之有效的方法和工具，软件工程学已经为广大软件科学工作者所重视，一个软件，如果不以软件工程学为指导而开发，就不太可能是一个高品质的软件。

本书从实用的规点出发，不是全面系统地介绍软件工程学中出现的各种方法和工具；而是通过贯穿全书的例子，叙述了一些最普遍使用的方法和工具。例如，需求分析阶段的数据流图，设计阶段的结构图及伪码描述法，编码阶段的结构化程序设计方法，测试阶段的增量法、驱动块和存根块法等。这些方法和工具简单实用，可供读者在开发软件系统时参考。

书中附有大量的习题，尤其是第三章末的十几个问题，规格适中，内容实用，问题的覆盖面很广，既可作为学习之后的实践课题，完成之后也是一些可供实用的软件产品。

本书出版后很快被译成了日文。我们翻译中参照了日文译本。

全书由鹿树理副教授主译，闻人德泰副教授校。此外，徐崇庶、吴晴、徐晨、石炼同志参加了部分翻译工作。

本书可供有一定实践经验的软件开发人员阅读，也可作为“软件工程”课的教材或教学参考书。

译者 1990．2

前　言

计算机科学技术的不断革命，不仅体现于硬件，也以软件开发过程观念的改变而体现出来。我们都曾经历过这样的时期，即把软件开发主要当作一门艺术来处理。虽然软件开发不是一门科学，但是这种处理已经从它原来的特定方法上前进到了一个新的起点，它的许多工作都可加以系统化。再有，当今对软件的范围和复杂性都有更大和更高的要求，如果没有充分的系统化方法，一般来说，要承担大型软件课题是不可能的。

过去的几年中，已经开始把有条理地生产计算机程序的一些方法，综合成一些原则、技术和工具的集合，这就是人们熟知的软件工程。使用软件工程这一术语并非随心所欲，在很大程度上，软件工程的概念来源于传统的工程实践，这种实践已经广泛地、有效地应用于产品中。其中心思想可简单地表述为：程序也是产品，可以用指导开发其它更具体产品的同样方法加以处理。作为一种尝试性意见而提出的这种观点，已经很迅速地赢得了信任，受到商业、工业和政府部门中愈来愈多的取得成功经验的人们的支持。结果，软件在日益增多的组织中得到了应用，每个软件应该达到的目的都是经过仔细研究确定的，这就促使在完成软件时，要经过一个有规划的设计过程。以一组“蓝图”加以文档化的这一活动的成果，可以说是在指导着软件的实际编制和测试工作。与此相关，“程序设计”活动(即编制可执行的代码)成为这样一项任务，它的目的是实现已经定义的，经过检查能被接受的算法处理。换句话说，软件产品的制作已经在形式上与它的设计分开了。

随着软件工程的发展已经看出，由于采用了大量的辅助手段，减少了与软件生产有关的混乱，并且这种方法得到了加强。这种方法中，最杰出的技术是功能分解、结构化程序设计、逐步求精和系统化测试。每一种技术都能使某一方面更有条理而为总的软件生产过程作出贡献。有效地应用这些技术，通过使用诸如结构图、伪码、存根块及驱动块等方便的工具使这些技术得到增强。这里我们所关心的正是这些概念、技术和工具的结合。

本书给出了生产有效的可靠的软件的一种严明的方法，它集中表现在下列处理过程中：

- 作出一个紧凑的产品设计；
- 将设计系统化地分解成一些合理的、且又可装配的组元；
- 分别地完成每个组元；
- 把完成后的组元加入到总的产品设计中；
- 在工作进展的同时，把有关活动文档化。

本书在提供关于系统化软件生产的背景和动力的第一章之后，以两种形式详细探讨了软件设计的有关活动；第二章讨论并阐述设计过程及与它有关的技术，然后通过应用它(在第三章中)去设计一个实际的软件产品，使第二章的内容得到扩充。第三章包含一组很广泛课题的规格说明，它们涉及各种领域，具有不同的难度，覆盖着广泛的应用范

围。大多数都取材于实际问题，使得学生能和现实世界保持接触。这些课题和为了强调具体技术或概念而设计的更有针对性的问题和练习一起，留待以后各章使用。

第四章，通过考查结构化程序设计的哲理、技术和工具，推进了上述过程。在第五章将把这些哲理、技术和工具应用到在第三章中已设计出来的产品中。

在最后三章中，详细介绍了软件的实现和测试过程。由诸如存根块、驱动块及有计划的“检测仪”等工具支持的逐步求精和系统化测试的高效技术，被应用到各种实际例子中。结果，使它们的功能很快地体现出来，并提高了它们的效率。

系统化软件开发最惊人的方面在于它的技术和方法的简单性。因此，断然不能延误对它们的学习和使用。不系统地学习和实践，是没有意义的。因此，本书可作为一或二学期的教材(具体期限依赖于所布置课题的数目和范围)。学生必备的知识是精通一种高级程序设计语言。事实上，本书曾成功地用于学生只熟悉一种语言基本特征的场合，其余的经验和专门的知识将和书中的方法一起学会。

如果把这本书的内容仅认为是在计算机科学领域中有用，那就背离了作者的本意。编写清晰而且可靠的软件的有效方法，在任何需要编写程序的场合下都是必要的。因此，对任何准备深入地从事计算机工作的学生、本书为他们提供了必要的基础知识和技巧。

作者衷心感谢 Messrs 的帮助，感谢Robert、Benson 和 Tom Bugnitz为我准备手稿而提供的方便。感谢Jerome R.Cox博士提供了在一些班级中使用这本教材的机会。同时，也感谢 Myrna Harbison 帮我复印了手稿以供学生使用，最后特别感谢 Cathy Gurganus在出版过程中对我们的帮助。

Will D.Gillett

Seymour V.Pollack

目　录

第一章　作为产品的程序

计算机应用的迅速增长，不但可以和电话、汽车以及电视机的增长相媲美，而且超过了它们。它之所以能这样持续增长，有如下两个明显的原因：首先，计算机科学技术已进到一些以前从未起过作用的领域。其中有些新领域由于被处理数据的复杂性和大量性，必须依靠计算机才能完成。其次，在一些已经应用了计算机的领域中，又要根据新的情况扩大使用。因此，引进计算机的组织，似乎可以分成两类，一类是扩充或替换它们的配置；另一类是根据需要引进计算机。

第二种增长特别有趣，因为刺激这种增长的计算机新应用，往往比以前的应用更有前景，也更为复杂。随之应用的实现一般说来又要求更大的程序，这些程序具有更复杂的过程逻辑。基于早期应用的成功经验，人们希望能把一些特定的编程方法应用到具有更高要求的应用之中。但是很遗憾，在计算机发展的最初二十年中，对程序所遵循的规格说明、开发和完善的具有实践特性的方法，并不能无限制地扩大应用。

在许多组织中，实际需要与进行合理编程之间的差距变得如此之大，以致造成了“软件危机”。虽然一开始认识这个问题的进程很缓慢，但现在人们已不再争论这样的问题。对这些更大、也更具有挑战性的计算机化问题，是不能靠补充人手和机器来有效地解决这一点大家已不再争论。现在，可以清楚地看到，任何大型的软件产品，并非不可做到经济而可靠的实现，甚至与早期那些相当简单的软件相比，在成本方面（包括时间和人力）已低得惊人。关键在于：把程序作为一种设计好的产品去反映某些操作特性，以满足那些经过严密思考并仔细确定的要求。

在以下各节中，我们将剖析软件危机的实质，以便对其起因和特性有所洞察。通过这种剖析，将易于说明结构化软件设计方法是挣脱危机的有效措施。进而可以将它通用化，把它看作是一种普遍适用的方法，去生产任意大小的软件产品，包括那些看起来用特殊方法就能处理的足够简单的产品。

1.1　软件的主要问题

广泛使用计算机的一个有趣的副作用是把计算机作为“替罪羊”。当报纸的某专栏底部需要填充一英寸左右的空白时，或当电视台的制片人认为电视新闻内容不够时，那么可行的办法就是找(或编造)一个计算机出错的笑话。如有个可怜的Fenwich太太，听见有人敲门，她去开门，发现前院堆了417立方英尺的新鲜有机肥。态度生硬的司机认定这里就是送货单上标明的地址，只需她签字。当手足无措的Fenwich太太最后找到会计部门时，他们所能做的只是告诉她确实订了所送的货，因为计算机是这样说的。另一个笑话是计算机“发了狂”，它给同一个人打印出3729份工资，而这个人甚至没有替该公司办事。

绝大多数可笑的错误，如误录音、坏新闻，最终都可以追溯到程序中的错误或疏

忽，但为时已晚。因而，对我们来说，找出出错的原因是相当重要的，因为这种错误和疏忽造成了混乱。事实是许多已经“经过检验”并“经过运行”而推出的供常规使用的计算机程序中，仍存在着严重的缺陷。有时是程序主逻辑上的错误，因为程序未能准确地完成编程者希望它应完成的工作。有的程序虽然正确地实现了编程者的意图，但编程者对于问题的要求考虑得不够准确或不够全面。因此，有必要了解这些问题的规律性，以便能用可靠的理论和方法来解决。

1.1.1 软件危机的特点

当前对软件危机的关心，并不意味着编程工作、编程人员及程序曾经有过较好的“黄金时代”。我们过去在软件上经常出现的问题，现在大多数都已被克服，或把影响减小到了最低程度。此外，由于当时设备的费用占的比重很大，人们便很少提到生产软件的费用。目前情况已经发生了戏剧性的变化，在多数情况下，软件费用在整个系统费用中所占的比例不断增长。美国国防部的数字表明，60年代初，软件费用占整个系统费用的20%左右，到了80年代，这个比例(有可能)超过90%。各个企业和商品部门的调查结果也表明国防部的数字具有普遍性。

这些统计促使人们进行更仔细的调查，以便认清根本问题所在。仍在继续进行的调查已经表明，下面几个主要困难能够表征出危机的特点：

1．在把“已开发”且“实用”的软件推出之后，软件费用的大部分（往往超过50%）都花在产品的使用期。换句话说，软件维护是整个软件开发过程中的一个重要部分。

2．对于没有完善定义而给出的软件课题，难以确定完成它所需的资源(例如时间、费用和人力等)，超额现象非常普遍，以致如果一个软件课题能在规定时间内按技术要求完成，那么人们就会给它以高度赞赏并加以特殊的研究。

3．一个课题的应用范围和意义，是在进入了它的生存期之后才搞清楚的。如果其中的一些想法过高，甚至不现实，仅当耗费了相当可观的资源之后才能发现。于是在开发过程的后期再进行重新分组、重新定义，就很难避免混乱状态。

4．对于许多(可能还是大部分)计算机应用程序而言，进行更改和扩充是很正常的，而当前的许多软件都很难进行修改。简单的选择，也许是舍弃大量运行中的软件，而不是在已有的基础上进行更改和扩充。

上述危机症状的有无并不取决于计划的规模、信息处理过程的性质和差异，也不取决于某软件占用机时的长短。造成上述困难的原因远远超出了这些因素。对这些原因加以调查，就能在开发改进技术上提供有价值的见解。

目前产生严重软件危机的原因是：对任一给定系统，在整个生存期上分析软件费用的旧方法失效了。这个生存期由设计阶段延伸到最初的实现、调试，并包括修改和维护在内。过去的方法没有认识到对已有软件进行修改和维护的重要性。实际上，软件系统是“活的”、“增长中的”实体，应该对它十分关心和注意才能使它达到成熟。由于种种原因，不可能把一个复杂的软件系统的各个部分，在初始实现时就结合在一起，这样做也没有好处。因此，任何一种用于开发软件产品的方法，都必须编写成能被理解（被编程者以外的其他人理解）、能被验证和修改的代码。不具备上述性质的代码，甚至在

只需要进行一些微小修改的场合，也注定要被废弃。

1.1.2 传统的方法及目前的需要

计算机应用的历史与其他事物的历史一样混乱。在计算机时代的最初十年中，许多与此新技术有关的生产资源，都围绕着装备某种型式计算机的大学实验室而组织起来。当时机器的运算速度和处理能力与今天的水平相比是很低的，于是存储器和机时都是非常贵重的商品。由于各个这样的计算站都有自己的独立程序集、自己的实现技术、自己的运行设施及将程序文档化的约定。许多数学方面的努力也都是以这些实验室为中心而进行的，且具有师徒方式的特征。

这种气氛曾促使形成一些和计算机应用有关的下述看法。

1．认为程序的准备工作在计算机开发应用过程中是一个首要的因素(即使它不是决定性的因素)。因此，应该把大部分精力都用在编写、调试和完善程序上。(许多人至今仍恪守这一信条。)

2．认为产生有用指令序列的编程和编码过程是一项具有高度智能的工作，几乎是具有艺术活力的东西，对不同的人是千差万别的。程序员被认为是计算机应用成败与否的关键因素。

3．一个程序，如果占用的存储单元少(甚至巧)、执行得快，就认为这个程序是很理想的。编程人员则致力于减少程序长度，缩短执行时间。

这些趋势很快就发展成事实。结果，计算机化的过程从概念上被理解成以编程为中心的过程。一旦人们了解了一个应用的要求，就会马上对此进行编程，而这样去编写程序，就会遇到新的问题。

1.1.2.1 解错误的问题

在开发应用中急于尽早地开始编写程序，往往可能尚未完全理解问题的基本要求。如果编程的人就是使用程序的人，那么这个缺陷还能在编程过程中逐步得以解决。当这二者是不同的人时，所要求的程序和所编写的程序之间的矛盾，有时就要等到程序工作时(实际上，有时根本不能工作！)才能发现。一旦发现这个程序所提供的是对某错误问题的“正确”解答，就要进行重新编程和调试，这个修改工作有时需要与最初一样的工作量。有些情况下，为了尽可能多地保留下“好的”代码(毕竟该程序还能运行)，就可能对他们的要求作出让步(只要结果不是十分离奇)，以此使得解答能和问题匹配。

1.1.2.2 充满技巧的工作

甚至在装置和调度都没有受到限制的情况下，编程人员还往往强调程序的长度和速度。结果，程序员把只适用于具体机器或程序的高度单一化技术、窍门和秘密的通路等都揉合到一起，想产生时间和空间的经济性。例如，需要常数12，他可能借用程序中某条指令的操作码(这个代码的值恰好为12)，而不是存入一个数12。这种方法的确能从程序中削减几个字，在执行时省几毫秒的时间，然而在分析这类程序的结构时却非常困难。这就意味着在应用过程中，每一点小改动都要由编程者本人来完成。如果读不懂程序，而编程者又不在。就不能修改程序。另一方面，如果不能经常满足新的要求，就只能废弃该程序，用一个新的版本来代替它。

使得程序更有效的另一种方法，曾倾向于使用含有几千条指令的代码块。为了避免

高价的内部信息管理，甚至可以把一个应用看作是一个单一的模块，在这个模块中，所有的信息都是可用的。这种无组织的编程方法再与一些技巧相结合，就产生出一种不灵活的代码。因为在这类程序中，只要改变一部分，就会影响所有其它部分（例如，在前面提到的例子中，由于程序之改变，操作代码12不再保存在原有的存储单元中）。

1.1.2.3 情况在不断加剧

正当计算机化的特定方法形成为一套标准的操作过程时，计算机应用的持续快速发展又导致了新的困难。只要具体应用的规模还比较适中，那么错误的起步，不通的语句及其它失误都可被认为是“正常的职业风险”。随着计算机应用越来越热门，复杂性越来越高，上述方法的使用也就越来越麻烦，因为这种方法不能作相应的改进。结果，由于缺少结构性和规范，人们便把注意力逐步转到应用开发过程上。于是，费用开始上升，变得与工作量的增加不成比例。

在许多情况下，当一个具体课题非常复杂，一个人无法处理全部细节时，效率的衰降就更需要注意。于是，不可避免地要把编程任务分给若干人去共同承担，每人承担一项，然后协调这几个人的工作，使得各部分可以连成一体。由于每个人都不得不用自己的特殊方法和秘诀去编程，使得这种协调工作就非常困难。解决这一问题的一般方法是增加几个“水平较高”的编程人员。这些人无非是受过一些奇特训练，各有自己的一套方法。由于他们更精通编程知识，把更深的裂缝和更巧妙的花招带到该过程中，从而使得许多课题由于人员增多反而变慢了。

实现计算机编程过程中的问题越来越多，环境和应用范围也越来越广，使得仅仅把这一切看作是少数编程人员和计算机不适配的观点也越来越不合理了。目前，越来越多的计算站开始检验并且评价他们在计算机上实现应用的方法。与此同时，进行了更普遍的调查，以便找出问题所涉及的范围和可能的起因。从各方面的调查资料可以明显看出：在应用开发上出现特定方法的原因是程序应用的范围相当有限；完成了一项较庞大的课题之后，维护工作与实现工作具有同样的工作量。如果用户希望使应用不断深入，就需要有更系统化的方法。此外，不管这个系统化方法的特点如何，都不应该仅仅是编写大程序的一种改进方法。或者说，要把重点从程序本身转移在编程工作之前的编程准备活动上来。在后面几章中，我们将看到，这种转移已经非常引人注目，以致在所有的应用范围中，编码过程已失去了传统的主导地位。事实上，偏离传统方法的一个最重要方面，就是当视野更宽时，在概念化、设计及实现这一全过程中，明显地降低了编程的重要性。

基于上述背景，我们可以开始探索结构化编程方法的特征，作为对上述主要问题的回答。

1.2 新的惯例

舍弃神密性编程惯例的关键因素很简单，即人们开始认识到程序是一种产品，它们象其他“产品”一样被销售、出租甚至上税，其用户也希望能通过某种方式来保证持久使用。然而，在很大程度上，程序产品与其他制造业产品的准备工作是极不相同的。如果我们回想起早先所强调的那种传统的编程方法，对这种区别就会看得很清楚。如果工

人也用和传统编程方法相同的方法去生产产品，那么可以想象其情景将是：他们不太知道也无需知道要做什么，他们可以边做边决定如何做，等做好了之后，才知道做的是什么。

为了使得计算机化过程和其它制造业生产过程更紧密地联系起来，我们将明显地或有意地应用那些表征开发复杂产品的设计和工作方法，并由此引出一些基本规则。下面将介绍这些规则。

1.2.1 结构化方法

结构化方法的实质是采用具有严谨方法学的工程方法。在这种方法学中，明显不同的可识别活动都以一定的顺序去开始和完成。就是说，我们了解每个活动，也了解如何去表明它已完成，然而却没有具体规律。即在软件的设计和实现方面，没有一种自然的方法学可以被推崇为是“正规”的方法。因而制定并实施一个充分确定的活动序列，并能指明这些活动的效果是很困难的。因此，本书后面所提到的方法并不是唯一的方法，而是一种能在宽广的课题范围和环境下得到满意结果的方法。

1.2.2 了解产品

要了解产品是理所当然的，历史已表明，在编写程序之前，确切地了解要编写的是什么程序是绝对重要的。这就是强调要有一个独立的设计阶段，在此阶段，要对所期望的结果(该产品的规格说明书)作仔细而全面的规定。这一设计阶段的结果并不是程序，而是把整个应用概念化，这是用计算机做某件事的一种方法。每个有关的模块都应当看作是组成程序包的一种产品，一旦明确了应对具体程序包提供些什么，那么相应的编程要求也就确定了。对一些实际应用来说，这个设计过程是自然而又容易完成的。对于比较复杂的问题，千万不要急于着手编程。在任何情况下，不经设计阶段就沉思于编程，实质上就是自取失败。

1.2.3 减少表面的复杂性

前面已经提过，对于比较复杂的课题，软件方面的困难变得非常尖锐。出现这种情况的原因已经很清楚，人们已经意识到了这个问题并开始进行研究。计算机应用方面的经验，使得人们可以在某一抽象层去想象愈来愈复杂的应用。但人们对这种复杂性的关注并没有太多的增长。进一步说，有各种原因使我们认识到我们的能力并没有实质性的提高，似乎也正是这点阻碍了我们向更广泛、更复杂的领域发展。

减少表面复杂性是工程方法学的基础之一。表面这个词很关键，如果不放弃一些原有的目的，就无法减少问题的复杂性。我们可以把一个问题仔细而系统地分为互相联系的一些较小的子问题，而每个子问题都在人们力所能及的范围内。同时，通过使各子问题之间的内在联系尽量简单，这可明显减少表面复杂性。按照这个想法，每个子问题只在绝对必要的情况下，才与相邻的子问题发生联系。后面还将看到，这种方法叫做分解法，它有助于在确定子问题时，划分各个子问题之间的界限。同时，当每个子问题都以程序或过程的形式出现时，这种方法对它们的顺序实现也有一定的指导作用。

1.2.4 验证和验收

面向产品方法的另一个基本特点，就是在进入下一步工作之前，要保证本步工作的正确性。这个问题看起来很简单，却有一些普遍的结论：

首先，它假定在求解一个给定问题时，有一系列步骤，它们按某种顺序相接。在下一章中将要重点讨论这种情况。例如，我们首先要确信该问题是自己所要解决的问题，然后再着手去寻找它的解。同样，如果我们找到了一种解，并把它表达为一个算法，那么在把此算法表示成程序之前，应当确信这个算法的确解决了我们要解决的问题。

第二，我们要了解“正确”的含义是什么。工作的每一步都要求保证正确性，而正确性的概念可以有各种方法去体现。因此，重要的是应该明白，对于生产过程的每一步检验都是必不可少的。

第三，如果检验是通向后继活动的必经之路，那么就应该有一个机构，在未能确信前一步的正确性时，就阻止进入下一步，这就是验收。在许多成功地应用了结构化方法学的计算站中，编程之前对设计进行仔细复查，已经形成为固定的步骤。这与在产品开发中的每一步都要进行复查或检验的标准工程实践方法完全一致。

1.2.5 艺术与技术

当人们开始认识到软件需要一种结构化方法时，最强烈的反对意见总是认为编程应是一门艺术，并应保持这一特点。持反对意见者认为编程本身是一项创造性活动，结构化方法会窒息这种创造性。现在已经很清楚，这种创造性的坚持者已把许多本来很简单的程序设计得难以理解。他们实际上是混淆了放弃艺术和改革艺术之间的区别。

在软件设计和实现方面的规范，应使专业人员能把创造性用在应用的设计、实现该设计的算法、以及最终体现这些算法的程序结构中。于是，在各模块的编码工作准备就绪时，我们应非常精确地知道共有多少模块。他们之间是怎样衔接的，以及每一模块要做什么。将这些详细说明和确定各模块的实际规则结合在一起，就把实际的编码过程简化为一项直观的工作。经验已经反复证明，这样做的结果将是一个既具有工程产品特性，又具有开发艺术和技巧的软件。下一节将检验某些特性是怎样从结构化方法中引出的，反过来又可确定这种方法学的各种成分。

1.3 最终产品的性质

由于结构化方法学使软件的开发与工程实践更近于一致，因此可以期望最终产品的性质能反映出这种特色。其中特别感兴趣的特点有三个，它们是：主要成份的清晰性、易于求精和易于修改。

1.3.1 主要成分的清晰性

一个工程化了的软件产品，其优点之一就是它的清晰性。在对产品进行检验时，实际上它做什么应没有神秘感。另外，由于软件及其描述二者联系紧密，因此有关任一层的功能问题都能作出充分的描述。因为，软件及其描述二者中缺少任一个，该产品都是不

完整的。下面将从几方面考查清晰性的概念。

1.3.1.1 软件说明书

工程化软件的一个完整部分，就是对它所要解决问题有一份确切描述。该描述避开了和实现有关的任何细节，因此这种产品的用户无需进一步明确该产品做什么，以及为了使用它所需的资源(例如，机型、运行环境)。同时，这种描述作为一种工具，用以确定该软件所解决的是否就是用户所要解决的问题(这里所说的用户，可能是一开始就是要该软件的人，也可能是正在寻找可用软件的局外人)。说明书认可的应用可以进行，说明书未认可的应用不能进行，这就是正确性的保证。

1.3.1.2 算法的清晰性

调查软件产品的另一个层次，是试图明确如何求解一特定问题。算法的描述再次抽象地回答了该问题，使得调查者能够了解到所使用的实际方法，又不致陷入实现语言的细节之中。例如，一个计算机化的应用可能依据众多算法中的一个来解决，每个算法都利用不同的数学模型(不同的数学模型有一组不同的公式)。然而每一算法都试图解决同一问题。因此，这种描述应精确地明确所使用的具体方法。如以前一样，清晰性不只是表面文章，如果遵从结构化方法，那么这种描述就应成为以后程序实现的基础。总之，应该经常为陈述算法所用的形式规定一些规则。

1.3.1.3 程序的清晰性

由于给定应用所选用或设计的算法是以抽象的术语来开发和陈述的，因此它和编程语言关系很少或者根本没有关系。决定选用哪种语言将受实际条件的影响(如可用的硬件和软件资源)，同时在一定程度上也取决于编程者本人的编程风格和才能。

结构化方法学的经验已经形成了某些有用的限制，以避免出现混乱的和难以理解的程序结构。例如，模块化(即把一个程序分为几个小的子程序)并不严格限制模块的长短和范围，使得从概念上可以把各个成分都看作是一个单一的实体来处理，从而任何复杂程序的整体结构和运行都可以看成是这些“黑箱”的合成。在这一抽象层中，我们可以假定每个模块都能正常工作，而无需知道它是如何工作的。这样我们就可以排除附加细节中复杂性所造成的负担，集中精力去考查程序的总体结构。

其它的结构化规则，也都要求把注意力集中在程序的设计上，略去和设计内容无关的细节，以便有助于提高程序的清晰性。在许多情况下，这些规则中还包括某些形式工具(如程序的设计语言)，借助于它来叙述程序的结构。这对复查(验证)是有用的，同时对后面的实现过程起信息源的作用。

1.3.1.4 模块的清晰性

这里讨论一个最具体的层次。这里所要处理的是实际的源语言代码。尽管代码的内容直接来自程序的结构说明书，而大多数编程语言的内容很丰富，使我们可以用多种途径来实现。如在程序的总体组织中一样，运用一些结构化的编码规则，就有可能生成功能很容易通过直接检验而理解的模块。对特定的结构化编程技术，当这些规则限制了编码过程时，可以不用这种方法，而在更广的范围内选取一种方法，在不失结果清晰性的前提下把这些技术结合进去。

以上所强调的是一种有条理的、逐步由最抽象层降到实际代码层的过渡方法，要求我们在开发的每一步都产生并验证说明书。每一步的内容必须足够清楚，以便引出下一

步。因此清晰性不应是附带的要求，事实上任何供软件生产用的结构化方法学，它的一个明确目标就是清晰性。

1.3.2 易于求精

即使使用最严格的方法学，也不能完全避免程序的差错。然而我们可以通过结构化设计以便于发现差错。尤其是在着手编码之前，我们知道如何构造该程序以及每个模块做些什么，那么我们就能以框架形式建立起完整的应用。并且在独立地开发并校正了每每模块之后，再把它们联结在一起。联结完成之后，每个模块就可以在整个程序中起它应有的作用(至少在概念上是如此)。在整个软件中，若某个模块尚未准备好，可在其位置上用一个足够简单的替代物来代替，这种暂时的替代物叫存根。存根可以设计成什么也不做，只是为相应的模块保留一个位置。当然，也可以编写模拟实际模块的模块，以便对一些已知的测试情况能产生和实际模块相同的结果，但无需对实际模块的全部内容作实际处理。

按照这种方法，一个非常复杂的问题，可以按步就班地有规律地构造起来。这种方法最显著的优点，也许就是提供了一种易于求精的过程，以便把调试工作集中在更为局部的错误上。

有选择地实现求精的能力，是我们希望有的好特性，因为无论如何谁也不可能一口气编写一个五千行的程序，而且一写完就能正确工作。对于一个软件，在消除了句法错误之后，对给定的输入却不能产生相应的输出时，我们应从何处着手去检查逻辑错误呢？是检查输入部分？还是检查处理部分？还是检查输出部分？这个问题涉及的范围太大，特别是对一种复杂的软件产品，要是没有一种能力去分离、设计、实现和调试它的小而功能上完整的成份，那么该课题一开始就是注定要失败的。

1.3.3 易于修改

除了极少数的情况外，期望软件产品在很长的时间内不进行改进而保持有用是不现实的。即使是对以前确定“完全可理解”的系统，当时不加考虑的能力和特性，后来才发现需要它们时，如果不修改将被看成是严重的缺点。在另外一些情况下，一些外部因素，如税收法的改变、管理法的改变，都会引起修改程序的要求。甚至在程序应用很稳定的情况下，由于一些由其它改变而引起的次要因素(如更换新的机器、新的编译程序和操作系统)，也不得不修改程序。

结构化设计方法消除了许多由于修改所造成的影响。其主要原因是由于在从算法转换为程序或过程时，对模块性给予了极大的关注。模块化的基本目的就是减少整个产品的表面复杂性。为了这个目的，我们用最简单的内部联系来配备各模块，以保持它们功能的完整性。各模块清楚的界限极其有助于确定和实现对程序的修改。从概念上说，每次功能修改，都可以用一个可用的(即经过设计、实现和验证)具有新功能的模块代替具有老功能的原有模块。由于原有模块之间的联系很少，因此这种改变的影响是高度局部化的。大量的经验表明，这种相当简单的方法在实践中是很有用的。同时这样的修改也很容易验证，因为每次修改只涉及到软件中非常有限的一部分模块。

结构化设计方法成功地应用到各种类型的复杂软件课题中，其特点和优点只不过是

一些约定和要求。在以下各章中，我们将尽力弄清这些概念和技术，以便明白这些约定是能够满足的。

1.4 小　结

过去，软件系统的生产者没有认识到开发软件所需的总费用，必须包括系统的整个生命期，从它的设计开始，经过实现并测试后投入试用，直到对算法组元的维护和转用。由于存在一种弊病，在弄清问题之前就跳到编程阶段，因此许多问题在产品生存期的后期才暴露出来。解决这些问题所花的费用往往大大超过最初实现所花的费用。主要的困难如下：

●未验证的设计——实现的是错误的内容。

●未验证的实现——对程序的测试不恰当。

●蹩脚的编码实践——过程模糊不清，难以理解，因此也难以更正和修改。

●可维护性差——对程序、数据中的错误，采用不恰当的手段去确定和校正。

软件工程是一套严格开发软件产品的方法，目的在于消除上述问题，最起码也要把这些问题减少到可以接受的范围之内。软件工程的有关方法是为了生产具有如下性质的软件：

●设计和实现明确易懂。

●易于分解复杂系统，使得能方便地抓住模块的组元。

●设计和实现易于验证。

●软件的组织允许方便而直接地修改。

实践证明，为了减少大型复杂软件课题的表面复杂性，本书中所讲述的技术和方法学是方便且有效的，所以人们用它们成功地进行了系统化的开发。

思 考 题

1．为了了解某些类型的程序具体有多大(往往很惊人)，程序大小的一种粗略的度量方法是该程序中源语言的语句数。另一种度量法是看最终可执行程序的大小。试运用上述二种度量方法之一或者二者都用，看看你是否能对下列每一个情况得到有关大小的信息。

a．你所写过的最大的程序。

b．一个标准的高级语言编译程序(如FORTRAN，COBOL，PL/1，PASCAL)。

c．一个中规模处理器的基本汇编程序或宏汇编程序。

d．一个小型计算机的基本汇编程序或宏汇编程序。

e．一个能够根据公司内每个人的档案产生出工资单的程序。

f．一个通用的工程应用程序(如STRUDL应力分析程序)。

g．一个通用的统计学程序(如因子分析程序或方差分析程序ANOVA)。

h．一个通用的数学应用程序(如线性规划程序)。

i．一个通用的离散模拟系统(如GPSS或SIMSCRIPT)。