软 件 设 计 方 法

王 选

清 华 大 学 出 版 社

内 容 简 介

本书从结构程序设计的观点介绍了三种现代流行的软件设计方法，全书共四章。第一章主要叙述结构程序设计的内容、方法和工具；第二章介绍基于数据结构的 Jackson 设计方法；第三章介绍基于数据结构的 Warnier 的 LCP 方法；第四章介绍 Yourdon 等人的数据流图法。书中所述的方法用于开发华光型计算机-激光汉字编辑排版系统的大型软件，取得了满意的效果。

本书可供从事开发各种计算机软件的科技人员阅读，也可作高等院校计算机专业高年级学生和教师的教学参考书。

(京)新登字 158 号

软 件 设 计 方 法

王 选

☆

清华大学出版社出版

北京 清华园

印刷厂印刷

新华书店总店科技发行所发行

☆

开本：787×1092 1/32 印张：7.5 字数：188 千字

1992 年 4 月第 1 版 1992 年 4 月第 1 次印刷

印数：00001—12000

ISBN 7-302-00989-9/TP·365

定价：3.90 元

前　言

北京大学等单位研制的华光型计算机-激光汉字编辑排版系统包含一个大型软件。我们在研制这一系统中，深感大型软件的开发是一件十分艰苦的工作，深感在软件开发方法和工具方面需要有一个较大的改善。在研究了三、四种软件设计方法后，我们决定试用 Warnier 和 Jackson 的基于数据结构的设计方法以及 Warnier 图解这一工具。Warnier 的 LCP 方法简单易学，使用得当效果明显。这一方法在法、美和日本等国已经在一定范围内流行。基于数据结构的软件设计方法与传统方法大不相同。习惯于流程图(框图)的软件工作人员接受这一方法时需要有一个适应过程，但不难掌握。一旦学会使用，便会喜欢这一方法。这种方法对典型的数据处理问题是很有效的。这是因为，在这类应用中，内部运算和处理是由输入输出数据驱动的。我们试验的项目并非数据处理性质的，而是一个面向问题的专用语言的编译，即一个用于排科技书的排版软件。在该系统中，排数学公式和排化学式的软件是采用 LCP 方法设计和实现的。LCP 方法的设计步骤比较严格，有可能显著减少设计阶段的逻辑错误。如排数学公式的软件虽有近 10000 行汇编代码，但调试时间较短，调试过程中未发现设计阶段的逻辑错误。这说明了该试验是比较成功的。有人认为 Jackson 方法只适合小程序，看来这是一种误解，我们的实践和他人的实践都不支持这种说法。

本书第一章介绍结构程序设计的概念。第二章介绍Jackson方法。第三章介绍Warnier的LCP方法。这两种方法均属基于数据结构的设计方法。在第二和第三章中,我们使用了Warnier图解,并把Jackson的“回溯”概念引入Warnier图解中。第四章介绍Constantine和Yourdon等的结构化设计方法(即数据流图方法),这一方法与第二、三章所述方法大不相同,其适用面可能更广。本书介绍的这些方法虽然都各有其适用范围、优点和缺点,但这些方法所包含的概念、步骤和工具在某种程度上具有普遍意义。读者可以利用本书介绍的方法来进行某个软件项目的开发。只有通过使用才能真正掌握一种方法。

本书只涉及设计阶段的方法,而未涉及规范阶段的方法,尽管规范是十分重要的。

本书用国产华光Ⅳ型计算机-激光汉字编辑排版系统排版,使用了批处理科技版排版软件和会话式框图和表格排版软件。编者在此谨向有关的研制人员和负责本书排版的同志表示谢意。

由于本人水平有限,实践经验不够,书中错误之处在所难免,请读者多多指正。

目　录

第一章　结构程序设计的发展

1.1　软件生存期

软件生存期就象人的寿命一样,从出生到死亡,即从产生开发要求到软件报废为止。这个过程可细分为系统分析和规范制定、设计、编码、调试、运行维护这五个阶段。

一、系统分析和规范制定

1. 系统的初步分析(或可行性研究)

提供给可行性研究阶段的信息有:

(1) 对现有系统的分析(现有系统可能并非计算机化的系统,而是一个人工操作的系统;也可能是一个需要进一步更新的落后的计算机系统)。

(2) 设想的新系统的大体构思。

(3) 组织方面的因素。例如改变的阻力,以前使用系统的经验、教训,工作人员的素质和类型等。

(4) 财政方面的考虑。包括现有系统的价格和改成新系统后的预期价格等。

这些信息可以看成可行性研究的输入信息。

可行性研究的结果则是一个可行性报告,它可以看作这一阶段的输出信息,包括下述内容:

(1) 系统的初步描述。

(2) 系统的影响分析。指系统实现后对用户单位的影响。

(3) 价格分析。包括开发代价和操作代价。

(4) 实现计划。包括所需的人力物力资源和进度。

可行性报告将提供给决策部门,若决定上马,则从事下一步工作。

2. 系统的详细分析(或系统设计)

这一阶段的输入信息有:

(1) 系统的初步描述(即可行性报告的第一部分)。

(2) 系统的一系列目标。

(3) 对系统的物理要求。

(4) 对系统的操作要求。

系统设计阶段的输出结果是

(1) 所需物理设备的要求。例如计算机的规模,内、外存容量,运算速度以及所占的物理空间。

(2) 系统的操作特性。包括物理设备的使用方式,信息的组织方式和存取方式等。

(3) 系统功能结构的规范。系统如何通过物理设备提供所需的操作能力。

(4) 系统部件(模块)的规范。描述组成系统的各模块的层次结构及每个模块的功能。

当系统设计阶段完成时,这些规范一方面提供给用户,由用户来鉴定这是否是他们所要求的系统;另一方面这些规范将作为生存期下一阶段的输入,设计阶段将严格按照规范进行。

规范要指明的是"做什么",而不是"如何做";切忌在规范中混入"如何做"的细节,尽管系统设计人员在制定规范时,对关键地方如何实现可能有所考虑。要防止"只见树木不见森林"的现象(即只见局部,不见全局)。这些都是系统分析时需要注意的原则。

近十年来发展了一些方法和工具以提高系统分析的质量

和效率,这些方法常常被称为结构化的分析法。采用的工具则有数据结构图、数据流图、数据字典和结构化的英语叙述。也有一些人鼓吹严格形式化的规范,但估计短期内很难普遍推广。

二、设计

传统的设计方法是画程序框图,即流程图。人们都知道,不能一上来就编程序,先要画粗框图,然后再画细框图,最后再编程序。但几种主要的软件设计现代方法都是反对流程图的。软件设计阶段是整个软件生存期中的重要阶段,本书主要涉及这一阶段的方法和工具。

三、编码(有时也叫"实现")

用高级语言或汇编语言编制程序,实际上是对设计阶段的结果进行细化,最后写成计算机可执行的程序。编码阶段要体现设计阶段所规定的程序结构。应尽量采用高级语言,少用汇编语言。汇编语言中包含与特定机器有关的内容和限制,往往会破坏本来很清晰易懂的程序结构。只有与硬件直接有关的中断处理和设备驱动程序,以及运行效率要求很高的程序,才不得不使用汇编语言。对同一问题而言,语言越高级,对应的语句行数越少,因而采用高级语言可提高编码阶段的效率和减少编码错误。高级语言中某些强制性规定能迫使程序员减少编码错误,或使程序员的错误能被系统自动查出。例如,高级语言中调用子程序需列出实在参数,子程序的说明部分则详细列出形式参数和子程序所用的局部量,而实在参数和形式参数的不一致可由系统查出。

四、调试和验收

调试的目的是发现和修改程序中的毛病以得到可靠的软件。最近十多年来,不少人从事程序正确性证明的研究工作,

尽管这一方向颇有吸引力,但离实用尚有很大距离。因此,在今后相当长的一个时期内,调试仍是一个不可缺少的重要手段。

调试过程一般如下:根据规范和程序结构,设计调试用的输入数据;上机运行,产生输出结果;把输出结果与预定的正确结果进行比较,看是否一致,若不一致,则寻找造成错误结果的原因(可能是编码阶段的错误,也可能是设计阶段的错误);修改错误后继续上机运行,直到产生正确结果。

这里,调试用的输入数据的设计是很重要的。在分调或每个模块的调试阶段,要根据程序结构设计调试数据,以保证程序的每一条分支流程都走到过。在系统联调或验收时,要根据规范设计调试数据,以保证系统的各项功能确实已经实现。

系统验收时不可能消灭所有的软件故障,但必须使软件故障减少到系统能够正常使用和用户可以接受的水平。

五、运行维护

运行维护是使已经验收和交付使用的软件系统继续运行,并不断改进其性能。它包括下列三类性质不同的工作:修理(repair),更改(revisions)和增强(enhancements)。“修理”是找出并改正设计和编码中的错误,而丝毫不改变原来制定的功能规范;“更改”是指原来制定的功能规范由于缺乏经验并不完全符合实际要求,因而需要修改规范,当然随之必须修改设计和编码;“增强”是指原来的功能规范是根据当时情况制定的,当初是合理的,但现在环境变了,提出新的、更强的功能要求,因而要增强功能规范。

以上五个阶段中,前四个阶段属于软件开发阶段,第五阶段是软件维护阶段。

1.2 软件开发和维护中存在的问题

一、软件研制周期长

按国外统计,开发一个软件所需的人月随该软件的代码行数按指数曲线上升。下面是美国数据控制公司(CDC 公司)的一个统计:

软件代码行数	所需人月
100K①	500
200K	1000
300K	2000
400K	3000
500K	4500
600K	6000

软件系统变得越来越大,而大软件系统的开发是一件代价大的艰苦工作。IBM 公司开发 700 万行代码的 MVS 操作系统,共花了 20 亿美元,即平均每行代码 285 美元。

二、生存期中调试阶段所花的时间长

据统计,前四个阶段(即开发阶段)中第一、二阶段占 40%,第三阶段仅占 20%,而第四阶段(调试)占 40%。调试阶段中发现的错误,只有 30%属编码错误;其余 70%的错误属于规范和设计的错误。为改正这些错误即要反馈回第二阶段,甚至第一阶段。这种反馈最影响进度,软件工程的目标之一是减少这种反馈。

① K = 1024

三、软件的维护工作量很大

软件生存期的第五阶段(维护)占整个生存期工作量的50%～90%。也就是说,当一个软件通过验收并开始正式运行时,只完成了不到一半的工作量,下面的维护工作量还大得很! 正因为如此,美国在数据处理方面的总投资中,有一半化在软件维护上。

软件维护工作量大的原因有如下一些:

(1) 验收后常常还有很多隐蔽故障。正如 Dijkstra 所说,“程序调试是表明故障存在的有效方法,但在证明故障不存在方面是无能为力的。”事实上,大系统中几乎永远有故障。

(2) 在使用过程中,需求和规范常常不断有所修改。这就导致设计和程序修改,以及修改部分程序的重新调试。当系统的结构比较差,模块化程度低,互相牵连严重时,这种修改就很困难,动一发牵全身。一个小的修改常常可引起不少新的错误。例如 IBM 的 OS/ 360,每次新版本往往带来几百个新的错误。

(3) 长期以来,程序是一种手艺,各人风格不同。不同的程序员编制同一问题的程序,可能得出结构完全不同的程序。由于缺乏统一的规格,使得程序很难由别人来维护和修改。

上述原因造成软件生产率低、投资大。随着信息社会的发展,系统越来越依赖于计算机,而计算机则依赖于软件。一条指令的错误会造成几千万美元的损失。1968 年软件工程会议上首次确认了软件危机的存在,认识到发展大的复杂系统的艰巨性,由此对软件系统的可靠性问题开始给予高度重视。软件工程的目的在于用科学方法提高软件生存期各阶段的生产率,以得到廉价可靠的软件。

1.3 好程序的标准

结构程序设计概念和各种软件设计的现代方法都为了得到“好程序”。因此必须对好程序的标准有一明确的看法。对此,五十年代的观点与七十年代末的观点有很大差别。五十年代的计算机内存很小、速度慢,往往把程序的长度和执行速度放在很重要的、甚至是首要的地位。人们往往费尽心机来缩短程序长度和减少所需存储量。否则内存放不下,压根就无法执行。现在情况完全不同了,按照现在流行的观点,好程序需具备下列素质:

1) 能够工作。这无疑是最重要的。假如一个程序根本不能工作,则执行速度、所需资源、源程序行数等可以测量的指标就变得毫无意义。在程序的风格方面,要注意下列三原则:

(1) 先保证正确,再考虑执行速度快;

(2) 先保证安全,再考虑执行速度快;

(3) 先保证清晰,再考虑执行速度快。

2) 调试代价低,也即花在调试上的时间较少。

3) 易于维护。

4) 易于修改。

5) 不复杂的设计。为了使程序易于调试、维护和修改,最合理的办法(有时甚至是唯一的办法)是使程序简单。

6) 高效。据对一批 FORTRAN 程序的统计,通常情况下,一个典型的 FORTRAN 程序的 50% 的执行时间被 3% 的语句所占用。为了得到高性能的程序,应采用下面方法:

(1) 先用直接了当的方法写程序,强调简单、易读、可靠;

(2) 程序工作后,再把最花机器时间的那部分程序优化,不要作普遍的优化,只优化那些确实值得优化的程序。

需要注意:优化可能会使本来清晰的设计结构变得模糊。

我们的目标是得到好程序,而只有好的程序员才能写出好程序。培养好的程序员是发展软件事业的关键。实践表明,程序员的质量差别很大。国外曾做过这样一个实验,选两个题目,找 12 个有经验的程序员来编写和调试,并进行比较(注意,找的都是有经验的程序员)。结果发现差别很大,最坏的和最好的程序员在下列四方面的对比很能说明问题:

(1) 调试时间　　28∶1

(2) 编程序时间　16∶1

(3) 程序长度　　6∶1

(4) 程序速度　　6∶1

好坏差别最大的是调试时间。调试时间往往是衡量一个程序员好坏的重要标志。

要得到好的可靠的程序,关键在于有好的软件队伍。根据我国目前情况,迫切需要培养一批掌握软件现代开发方法的业务骨干,以开拓各个应用领域。下面我们介绍美国一些学者建议的软件工程硕士毕业生应具备的业务水平。

(1) 能在系统分析、设计、编码、调试这四个领域中的每个领域内,准确地使用至少一种最新方法,从事软件的开发工作。

(2) 有能力用精心设计的实验对自己编制的软件进行调试和测量,以验证软件系统的质量(例如可靠性和效率),并识别出系统中需进一步完善的那些部分。

(3) 有能力在至少一个有意义的领域内,研制出先进的专用软件系统。这里所说的有意义的领域可以是操作系统、编译系统、数据库管理系统、信息管理系统和其他各种有价值的应用领域。

(4) 能有效地管理一个中等规模的软件项目,即 3～7 人做 1～2 年的项目。

(5) 能与用户、经理和其他技术人员进行有效的讨论和联系。

(6) 能很快学会新的软件工程方法,能跟得上计算机科学领域内的有关进展。

(7) 有能力评价、选择和实现新方法。

1.4 结构程序设计的发展历史

结构程序设计被称为软件发展中的第三个里程碑,其影响将比前两个里程碑(子程序、高级语言)更为深远。结构程序设计概念最早由 Dijkstra 提出,他在 1965 年召开的 IFIP 会议上提出"GO TO 语句可以从高级语言中取消","一个程序的质量与程序中所含的 GO TO 语句的数量成反比"。但是,Dijkstra 讲话的影响相当小。这是因为人们当时正忙于从 FORTRAN Ⅱ 过渡到 FORTRAN Ⅳ,而 GO TO 语句是 FORTRAN 的支柱,同时,许多用户当时正埋头于 IBM360 系列的使用,而 IBM360 的主要语言之一是 FORTRAN。

1966 年 B hm 和 Jacopini 首次证明了:只要三种控制结构就能表达用一个入口和一个出口的框图(流程图)所能表达的任何程序逻辑。这三种控制结构如下:

(1) 顺序结构(sequence)

如图 1-1(a)所示。相当于

f; g;

(2) 选择结构(selection)

如图 1-1(b)所示。相当于

IF C THEN f ELSE g;

(3) 循环结构(repetition)

如图 1-1(c)所示。相当于

WHILE C DO f;

这种结构又称为 WHILE 型循环。

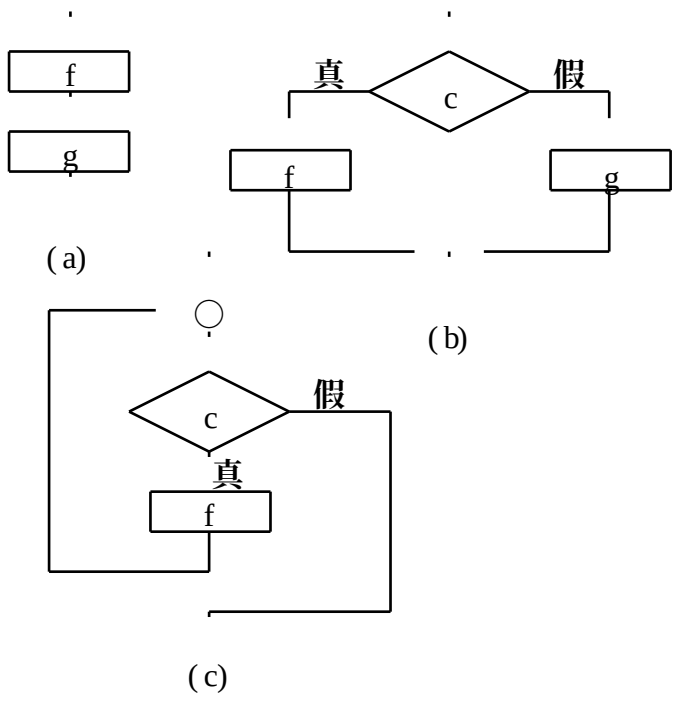

图1-1 三种基本控制结构

(a)顺序; (b)选择; (c)循环。

其实,控制结构 2 可以用控制结构 1 和 3 代替。

B hm 和 Jacopini 的工作为结构程序设计提供了理论基础。

1968 年 Dijkstra 写给《Communications of the ACM》杂志一篇短文,该杂志编辑部为早日发表,把它改成信件形式刊登,并冠以新的标题:“GO TO 语句被认为有害。”Dijkstra 建议:GO TO 语句太容易把程序弄乱,应从一切高级语言中去掉;只用三种基本控制结构就可以写各种程序,而这样的程序