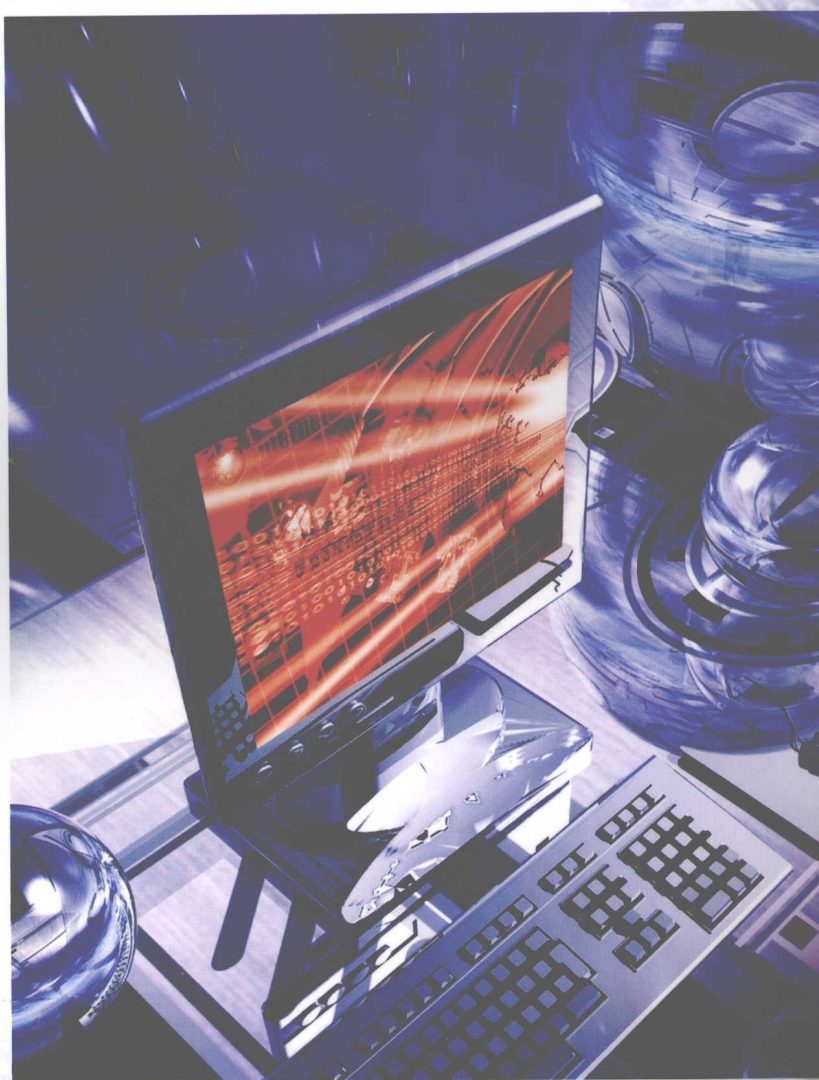
C/C++ 程序设计实训

陈 章 主编



上海财经大学出版社

C/C++程序设计实训

陈　章　主编

上海财经大学出版社

图书在版编目(CIP)数据

C/C++程序设计实训/陈章主编. —上海:上海财经大学出版社,2007.7
ISBN 978-7-81098-985-5/TP·005
Ⅰ.C… Ⅱ.陈… Ⅲ.C语言-程序设计-高等学校-教材 Ⅳ.TP312
中国版本图书馆 CIP 数据核字(2007)第 077882 号

□责任编辑 袁 敏
□封面设计 周卫民

C/C++ CHENGXU SHEJI SHIXUN
C/C++ 程 序 设 计 实 训
陈 章 主编

上海财经大学出版社出版发行
(上海市武东路 321 号乙 邮编 200434)
网 址:http://www.sufep.com
电子邮箱:webmaster@sufep.com
全国新华书店经销
上海译文印刷厂印刷
上海望新印刷厂装订
2007 年 7 月第 1 版 2007 年 7 月第 1 次印刷

787mm×1092mm 1/16 15.75 印张 486 千字
印数:0 001—5 000 定价:26.00 元

前　言

C语言以其小巧、灵活、高效等特点成为当今软件开发的主要编程语言，近年来，C程序设计语言已作为大学生的入门语言，本书与《C/C++程序设计》和《C/C++语言程序设计学习及应试指导》配套，成为学习C语言的“三剑客”。

本书是根据《C/C++程序设计》的教学要求从提高学生编程能力入手，从软件开发的成功案例为切入点，同时配套几十个实习题，使学生从理论到实践，在实践中体会软件文档的作用，体会程序测试的必要性，总结程序设计的方法，学会相互协作。

本书分为四章：第1章“绪论”，主要介绍了软件开发中所需的基本知识；第2章“程序设计和系统开发基础”，主要介绍了程序开发的环境、程序开发的方法；第3章“成功案例与实践”，分析了成功案例的设计思想、技术难点并给出三类几十个典型实习题，旨在帮助学生摆脱程序设计等于程序设计语法知识学习的怪圈，提高解决实际问题的能力；第4章“习题解析”，对《C/C++语言程序设计学习及应试指导》一书第6章～第10章中的习题进行解答、归纳和总结，从而导出程序设计的规律和方法。附录I为《C/C++语言程序设计学习及应试指导》一书第1章～第5章习题的答案。

本书旨在帮助学生利用C/C++开发工具开发一个小型系统，让学生了解软件开发的整个过程，掌握软件工程的基本方法，培养学生初步应用软件工程相关技术进行软件开发的能力。

本书注重训练环节，体现了在理论指导下让学生动手、动脑的基本思想方法，提出理性思维和理性实践。按照建构主义的学习理论，学生作为学习的主体在与客观环境（指所学内容）的交互过程中构建自己的知识结构。本书引导学生在解题编程中探索其中带规律性的认识，将感性认识升华到理性高度，这样学生就能举一反三。本书可供各层面学生、教师、自学应试者阅读。

本书主编为上海理工大学计算机基础教学一线教师陈章。参加编写的有臧劲松、黄春梅、黄小瑜、徐宇清、马晓旦、吴存孝、黄玉林、黄鹤鸣、马立新、刘丽霞。担任本书主审的是计算机等级考试命题组专家夏耘老师。

在编写过程中，还组织了集体统稿、定稿，并得到了清华大学、交通大学、复旦大学、华东师范大学、华东理工大学、上海理工大学、上海大学等校计算中心各位老师的帮助，在此一并致谢。

由于时间仓促和水平有限，本书中难免还存在一些不妥之处，请广大读者批评指正。

编　者

2007年5月

目　录

第1章 绪 论

1.1 软件与软件工程

1.1.1 软件及其特性

软件是由计算机程序的发展而形成的一个概念，它是计算机系统操作相关的程序、规程、规则及其文档和数据的统称。软件是计算机的“灵魂”。

软件的质量是软件的生命，提高软件的质量是需求方的目标，而软件的质量因素有很多，如正确性、性能、可靠性、容错性、易用性、灵活性、可扩充性、可理解性、可维护性等。有些因素相互重叠，有些则相互抵触。

软件工程的目标是提高软件的质量与生产率，最终实现软件的工业化生产。软件工程学是将计算机科学理论与现代工程方法论相结合，围绕软件生产过程自动化和软件产品质量保证，展开对软件生产方式、生产管理、软件开发方法、生产工具系统和产品质量保证的系统研究。

软件工程的主要环节有：人员管理、项目管理、可行性与需求分析、系统设计、程序设计、测试、维护等，如图1－1所示。

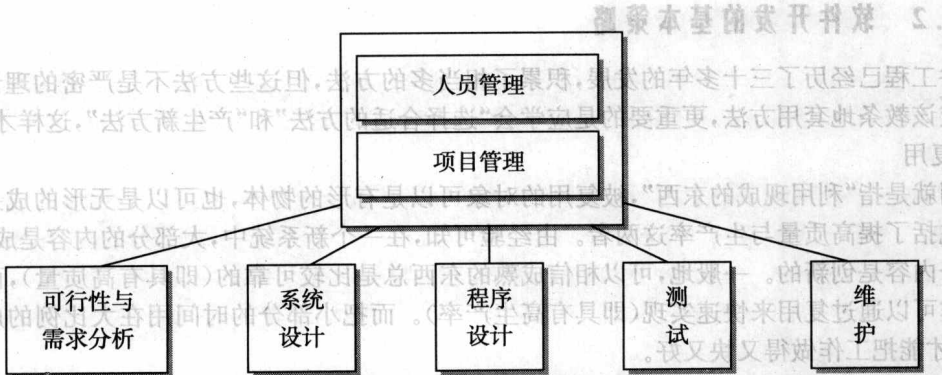


图1－1 软件工程的主要环节

软件工程模型建议用一定的流程将各个环节连接起来，并可用规范的方式操作全过程，如同工厂的生产线。常见的软件工程模型有：线性模型(见图1－2)、渐增式模型(见图1－3)、螺旋模型、快速原型模型、形式化描述模型等[Pressmam(1999)，Sommerville(1992)]。

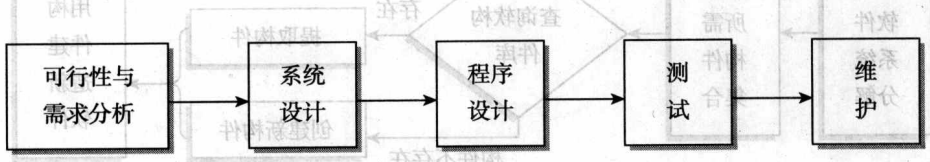


图1－2 软件工程的线性模型

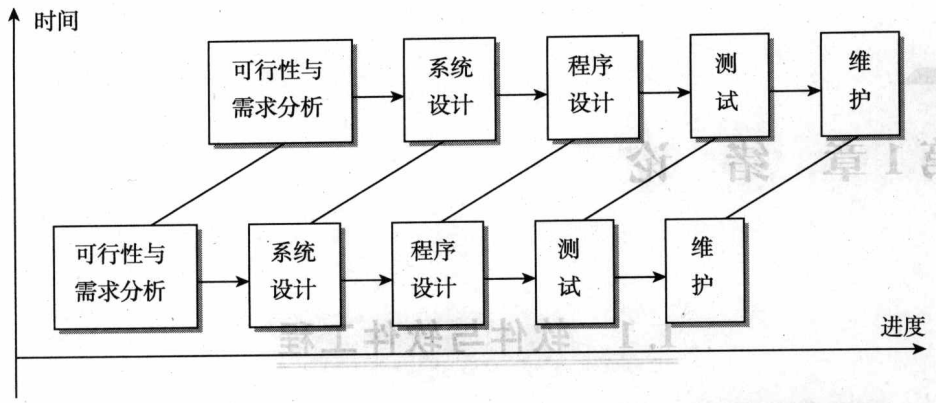
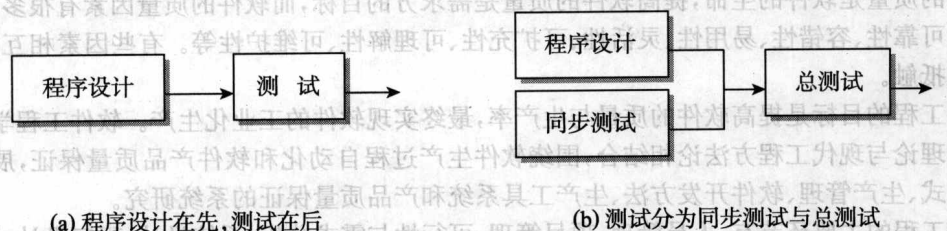


图 1－3　软件工程的渐增式模型

“程序设计”与“程序测试”之间的关系，习惯上总以为程序设计在先，测试在后，如图 1－4(a)所示。而对于一些复杂的程序，将测试分为同步测试与总测试更有效，如图 1－4(b)所示。



(a) 程序设计在先，测试在后　　(b) 测试分为同步测试与总测试

图 1－4　“程序设计”与“程序测试”之间的关系

1.1.2　软件开发的基本策略

软件工程已经历了三十多年的发展，积累了相当多的方法，但这些方法不是严密的理论。实践人员不应该教条地套用方法，更重要的是应学会“选择合适的方法”和“产生新方法”，这样才会成功。

1. 复用

复用就是指“利用现成的东西”，被复用的对象可以是有形的物体，也可以是无形的成果。复用的内涵包括了提高质量与生产率这两者。由经验可知，在一个新系统中，大部分的内容是成熟的，只有小部分内容是创新的。一般地，可以相信成熟的东西总是比较可靠的(即具有高质量)，而大量成熟的工作可以通过复用来快速实现(即具有高生产率)。而把小部分的时间用在大比例的成熟工作中，这样才能把工作做得又快又好。

软件复用不仅要使自己拿来方便，还要让别人拿去方便，是“拿来拿去主义”。面向对象的方法、Microsoft 公司的 COM 规范[Rogerson(1999)]，都能很好地用于实现大规模的软件复用。利用复用策略开发应用软件的过程见图 1－5，其中软构件就是复用对象。

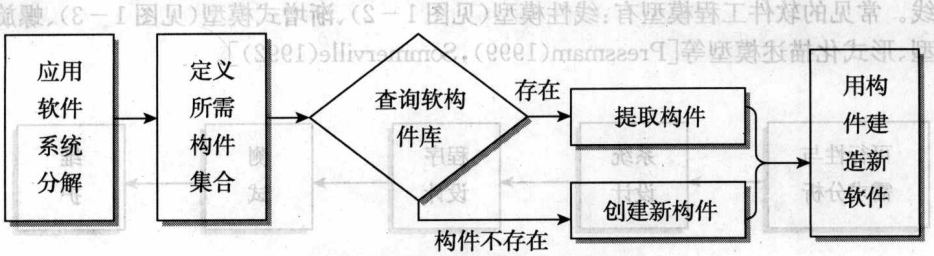


图 1－5　利用软构件生产应用软件的过程

2. 分而治之

分而治之是指把一个复杂的问题分解成若干个简单的问题，然后逐个解决(见图 1－6)。这种朴素的思想来源于人们生活与工作的经验，完全适合于技术领域。软件人员在执行分而治之的时候应该着重考虑：复杂问题分解后，每个问题能否用程序实现？所有程序最终能否集成为一个软件系统并有效解决原始的复杂问题？

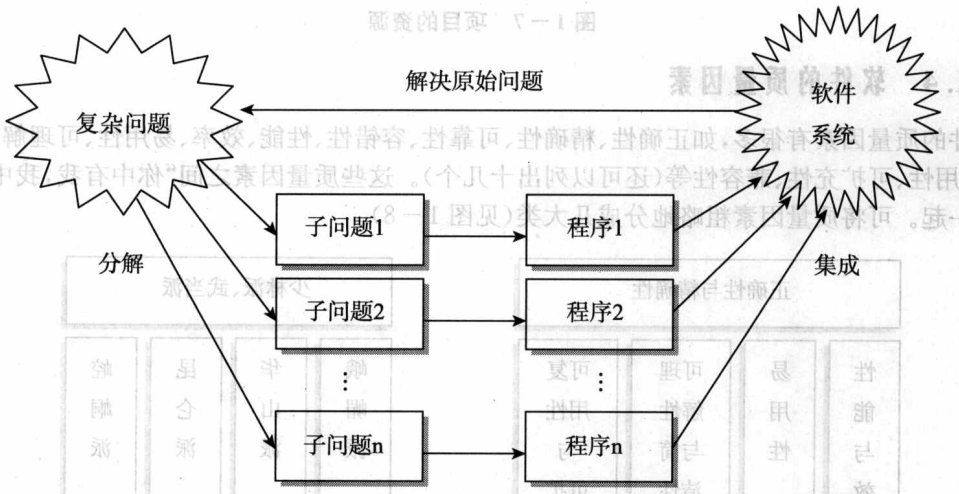


图 1－6 软件领域的分而治之策略

3. 优化—折衷

软件的优化是指优化软件的各个质量因素，如提高运行速度、提高对内存资源的利用率、使用户界面更加友好、使三维图形的真实感更强等。优化工作的复杂之处是很多目标存在千丝万缕的关系，可谓数不清理还乱。当不能够使所有的目标都得到优化时，就需要“折衷”策略。

软件中的折衷策略是指通过协调各个质量因素，实现整体质量的最优。下面用“优化—折衷”的策略解决“鱼和熊掌不可兼得”的难题。

问题提出：假设鱼每千克 10 元，熊掌每千克 1 万元。有个倔脾气的人只有 20 元钱，非得要吃上 1 千克美妙的“熊掌烧鱼”，怎么办？

解决方案：花 9.99 元买 999 克鱼肉，花 10 元钱买 1 克熊掌肉，可做一道“熊掌烧鱼”菜。剩下的 1 分钱还可建立奖励基金。

1.1.3 项目计划

首先要了解项目的规模、难度与时间限制，才可以确定应该投入多少人力、物力去做这个项目。在可行性分析阶段就要考虑这个问题。

项目的时间限制有两类：第一类，项目应该完成的日期写在合同中，如果延期了，则开发方要做出相应的赔偿；第二类是开发自己的软件产品，虽然只确定了该产品大致的发行日期并允许有延误，但如果拖延太久则会失去商机而造成损失。

项目的资源分为三类：“人”、“可复用的软构件”和“软硬件环境”，如图 1－7 所示。

1. 人是最有价值的资源。项目计划的制定者要确定开发人员的名单，要根据他们的专长进行分工。

2. 可复用的软构件是次要价值的资源。

3. 软硬件环境虽然不是最重要的资源，却是必需的资源。原则上软硬件环境只要符合项目的开发要求即可。

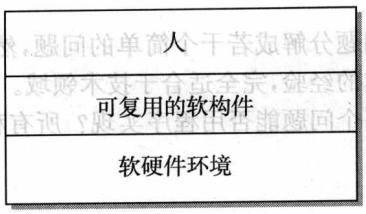


图 1－7　项目的资源

1.1.4　软件的质量因素

软件的质量因素有很多，如正确性、精确性、可靠性、容错性、性能、效率、易用性、可理解性、简洁性、可复用性、可扩充性、兼容性等(还可以列出十几个)。这些质量因素之间“你中有我，我中有他”，交织在一起。可将质量因素粗略地分成几大类(见图 1－8)。

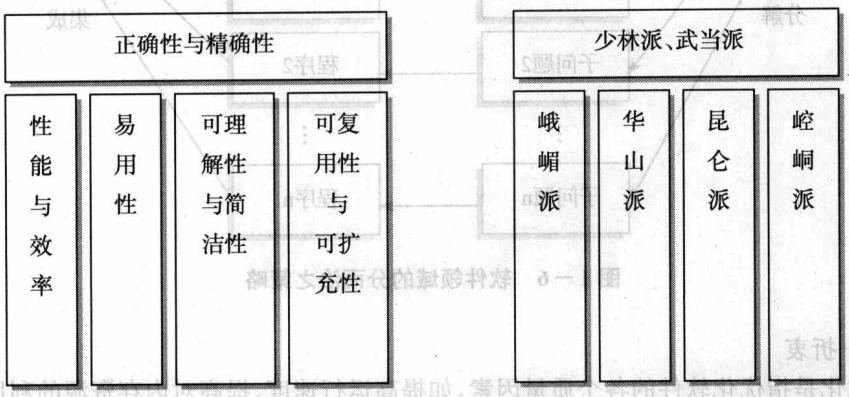


图 1－8　软件质量因素分类和武学分类

正确性与精确性之所以排在质量因素的第一位，是因为如果软件运行不正确或者不精确，就会给用户造成不便甚至造成损失。机器不会主动欺骗人，软件运行不正确或者不精确一般都是人为的。即使一个软件能百分之百地按需求规格执行，但是如果需求分析错了，那么对客户而言这个软件也存在错误。即使需求分析完全符合客户的要求，但是如果软件没有百分之百地按需求规格执行，那么这个软件也存在错误。开发一个大的软件项目，程序员应将“正确”和“精确”作为自己的目标。

与正确性、精确性相关的质量因素是容错性和可靠性。容错性首先承认软件系统存在不正确与不精确的因素，为了防止潜在的不正确与不精确因素引发灾难，系统为此设计了安全措施。在一些高风险的软件系统(如航空航天、武器、金融等系统)中，容错性设计非常重要。

1.1.5　可行性分析与需求分析

可行性分析是要决定“做还是不做”。需求分析是要决定“做什么，不做什么”。不论是为客户做软件项目还是为自己做软件产品，都要进行需求分析。需求分析最难的是难以在项目刚启动时搞清楚需求，如果在项目做了一大半时需求发生了变化，那将使项目陷入困境。

1. 可行性分析案例之一

这个案例讲述了一个大学生创业的一些经历和感受。

有位大学生从本科三年级开始编写图形程序，一见钟情后便如痴如醉，不管一切地抛弃了本科与硕士所读的专业。1997 年春季，他到了向往已久的浙江大学 CAD&CG 国家重点实验室读博士学位。他幸福地幻想着大干一番自己喜爱的专业。开学的第一天，他就兴冲冲地奔向实验室。进门不到 5 分钟，就因不懂规矩被看门的训了几次。为了不再冒犯规矩，他就老老实实地抓起一份计算机报纸并且站着阅读。突然，一个气得脸色铁青的男人(机房管理员之一)对他断喝：“你在干什么！你

怎么可以不经允许就翻看别人的报纸!”

不出几日,他就发现在实验室里人们大多轻言寡语、小心翼翼、井水不犯河水。初到此实验室的北方同学极为迷惑地问他:“你们浙江人是不是都这样?我看你不太像嘛。”

他颇费周折地考入CAD&CG实验室,却尚未热身就全力而退,决心自立门户(至今他都没有用实验室的计算机编过一行程序)。

那时他穷困潦倒,只有一床、一盆、一壶、一碗。他的那些穷朋友们像挤牙膏一样挤出一些钱资助他。他买了一台计算机,就在宿舍里开发软件。1997年8月,他去北京参加首届中国大学生电脑大赛软件展示。路费也是借的,同学为他壮胆时说:“如果不能获奖,就回到实验室干活吧。”他却说:“一定会拿第一名。”

于是他拿了第一名,号称大学生“软件明星”。回到浙大后看看没啥反应,他就向杭州的几家报社发了一份简讯。不久有一个记者来采访他,谈了一天,发现记者还是不太明白,干脆自己写新闻报道,并且含蓄地做了一个广告:万事俱备,只待投资。10月份,他被评为浙江省青年英才(大学生代表),又获得中国大学生跨世纪发展基金特等奖。他到东软集团(沈阳)参加“民族软件产业青年论坛”,大大咧咧地作了一次演讲。由于他能说会道,频频上电视,引来近十个投资者。他选择了一位年龄比他大一倍、非常精明的商人作为合伙人,成立了“杭州临境软件开发有限公司”。那时,他可谓光芒四射,卓而不群,名片上印着“以振兴民族软件产业为己任,做真实、正直、优秀的科技人员”。

他当时想开发一套名为Soft3D的图形系统,此系统下至开发工具,上至应用软件,无所不包。公司名字起为“临境”有两个含义:一是表示身临其境,这是他对图形技术的追求;二是表示快到了与SGI公司称兄道弟的境界,这是他对事业的追求。

他从实验室挖来一位聪明绝顶的师弟做技术伙伴。他俩一拍即合,常常为Soft3D的设计方案自我倾倒。一想到Microsoft公司的二维Windows系统即将被Soft3D打击得狼狈不堪时,他们就乐不可支、冲劲十足。到了1998年7月,他们做了一套既不是科研又不全像商品的软件,宣传了几个月都没有人要。1998年10月,他用光了30万元资金,只好关闭公司。

分析其错误原因,有以下几点:

(1) 年轻气盛,在不具备条件的情况下,想一下子做成石破天惊的事。设计方案技术难度很大(有一些是热门的研究课题),只有30万元资金的小公司根本没有财力与技术力量去做这种事。分析经济与技术可行性,即可否定其设计方案。

(2) 以技术为中心而没有以市场为中心去做产品,以为自己喜欢的软件别人也一定喜欢。他涉足的是在国内尚不成熟的市场,无法估计该市场有多大以及人们到底要什么。分析市场可行性也可否定其设计方案。

(3) 投资方是个精明的商人,他把设计方案交给美国的一家软件公司分析,结论是否定的。但他觉得这个人很有利用价值,希望可以做成功其他事情,即使Soft3D软件做不成功,只要挣到钱就行。这种赌博心态使得正确的可行性分析变得毫无价值。

2. 需求分析的难点

(1) 客户说不清楚需求

有些客户对需求只有朦胧的感觉,当然说不清楚具体的需求。例如,全国各地的很多公司在搞网络建设,这些单位的领导和办公人员大多不清楚计算机网络有什么用,反而要软件系统分析人员替他们设想需求。这类工程的需求是如此主观,以致产生很多不良现象。

有些客户心里非常清楚想要什么,但却说不明白。就举日常生活的事例吧,比如说买鞋子。大多数人可能非常了解自己的脚,但没法说清楚脚的大小和形状,只能去试穿,试穿时感觉到舒服才会买鞋(居然也有神通广大的售货员,看一眼客户的脚,就知道应该穿什么尺码的鞋)。如果客户本身就懂软件开发,能把需求说得清清楚楚,这样的需求分析将会非常轻松、愉快。如果客户完全不懂软件,但信任软件开发方,这事也好办。分析人员可以引导客户,先阐述常规的需求,再由客户否定不需要的,最终确定客户真正的需求。最怕的就是“不懂装懂”或者“半懂充内行”的客

户，他们会提出不切实际的需求。如果这些客户甚至觉得自己是上帝的爸爸，那么沟通和协商都会变得很困难。

(2) 需求自身经常变动

软件的需求会变化吗？

答：据历史记载，没有一个软件的需求改动少于三次。惟一只改动需求两次的客户是个死人。这个可怜的家伙还是在运送第三次需求的路上被车子撞死的[Cline(1995)]。

让我们先接受“需求会变动”这个事实吧，免得在需求变动时惊慌失措。明白“需求会变动”这个道理后，在进行需求分析时就要留点神：

① 尽可能分析清楚哪些是稳定的需求，哪些是易变的需求。以便在进行系统设计时，将软件的核心构筑在稳定的需求上，否则将会吃尽苦头。

② 在合同中一定要说清楚“做什么”和“不做什么”。如果合同含含糊糊，日后扯皮的事情就会很多。

(3) 分析人员或客户理解有误

有个外星人间谍潜伏到地球刺探情报，他给上司写了一份报告：“主宰地球的是车。它们喝汽油，靠四个轮子滚动前进。嗓门极大，在夜里双眼能射出强光。……有趣的是，车里住着一种叫作‘人’的寄生虫，这些寄生虫完全控制了车。”

软件系统分析人员不可能都是全才。客户表达的需求，不同的分析人员可能有不同的理解。如果分析人员理解错了，可能会导致开发人员白干活，吃力不讨好。分析人员写好需求说明书后，要请客户方的各个代表验证。如果问题很复杂，双方都不太明白，就有必要请开发人员快速构造软件的原型，双方再次论证需求说明书是否正确。

由于客户大多不懂软件，他们可能觉得软件是万能的，会提出一些无法实现的需求。有时客户还会把软件系统分析人员的建议或答复给想歪了。

有一个软件人员滔滔不绝地向客户讲解在“信息高速公路上做广告”的种种好处，客户听得津津有味。最后，心动的客户对软件人员说：“很好，就让我们马上行动起来吧。请您决定广告牌的尺寸和放在哪条高速公路上，我立即派人去做。”

3. 如何进行需求分析

(1) 应该了解什么

需求分析不像侦探推理那样从蛛丝马迹着手。应该先了解宏观的问题，再了解细节的问题，如图 1－9 所示。

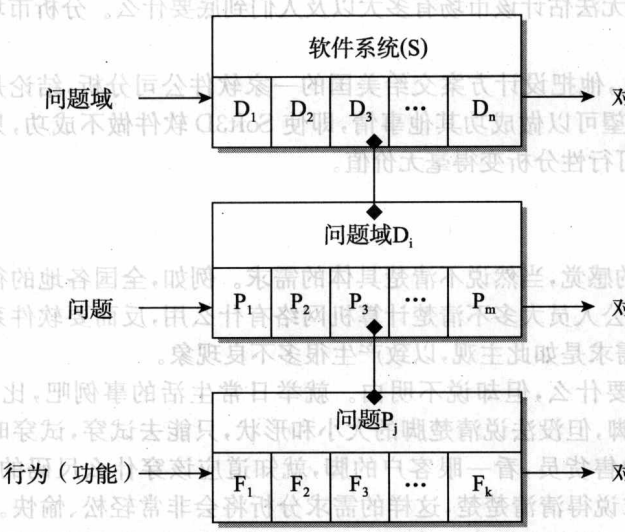


图 1－9　进行需求分析时要了解的内容

一个软件系统(记为 S)的涉及面可能很广,可以按不同的问题域(记为 D)进行分类,每个问题域对应一个软件子系统:

$$S=\{D_1,D_2,D_3,\cdots,D_n\}$$

问题域 D_i 由若干个问题(记为 P)组成,每个问题对应子系统中的一个软构件:

$$D_i=\{P_1,P_2,P_3,\cdots,P_m\}$$

问题 P_j 有若干个行为(或功能,记为 F),每个行为对应软构件中的接口:

$$P_j=\{F_1,F_2,F_3,\cdots,F_k\}$$

按图 1-9 结构写成的需求说明书,对于那些只想了解宏观需求的领导以及需要了解细节的技术员而言都很合适。在写需求说明书时还应该注意两个问题:

① 最好为每个需求注释"为什么",这样可以让程序员了解需求的本质,以便选用最合适的技术来实现此需求。

② 需求说明不可有二义性,更不能前后相矛盾。如果有二义性或前后相矛盾,则要重新分析此需求。

(2) 通过什么方式去了解

了解需求的方式有好几种:

① 直接与客户交谈。如果分析人员生有足球评论员的那张"大嘴",就非常容易侃出需求。

② 有些需求客户讲不清楚,分析人员又猜不透,这时就要请教行家。有些高手真的很厉害,你还没有开始问,他就能讲出前因后果。让你感到"听君一席话,胜读十年书"。

③ 有很多需求可能客户与分析人员想都没有想过,或者想得太幼稚。要经常分析优秀的和蹩脚的同类软件,看到了优点就尽量吸取,看到了缺点就引以为戒。前人既然付了学费,后人就不要拒绝坐享其成。

1.2 系统设计

系统设计是把需求转化为软件系统的最重要的环节。系统设计的优劣在根本上决定了软件系统的质量。

Windows NT 的一位系统设计师拥有 8 辆法拉利跑车,让 Microsoft 公司的一些程序员十分眼红。但你只能羡慕而不能愤恨,因为并不是每个程序员都有本事成为复杂软件系统的设计师。系统设计要比纯粹的编程困难得多。即便你清楚客户的需求,却未必知道应该设计什么样的既能挣最多的钱又能让客户满意的软件系统。

下面讲述系统设计的四方面内容:体系结构设计、模块设计、数据结构与算法设计、用户界面设计。如果将软件系统比喻为人体,那么:

1. 体系结构就如同人的骨架。如果某个家伙的骨架是猴子,那么无论怎样喂养和美容,这家伙始终都是猴子,不会成为人。

2. 模块就如同人的器官,具有特定的功能。人体中最出色的模块设计之一是手,手只有几种动作,却能做无限多的事情。

3. 数据结构与算法就如同人的血脉和神经,它让器官具有生命并能发挥功能。数据结构与算法分布在体系结构和模块中,它将协调系统的各个功能。人的耳朵和嘴巴虽然是相对独立的器官,但如果耳朵失聪了,嘴巴就只能发出"啊"、"呜"的声音,等于丧失了说话的功能,可人们却能用手势代替说话。人体的数据结构与算法设计真是十分神奇并且十分可笑。

4. 用户界面就如同人的外表。如人类追求心灵美和外表美那样,软件系统也追求(内在的)功能强大和(外表的)界面友好。

在进行系统设计时，要关注软件的质量因素，如正确性与精确性、性能与效率、易用性、可理解性与简洁性、可复用性与可扩充性等。即使把系统设计做好了，也并不意味着就能产生好的软件系统。在程序设计、测试、维护等环节还要做大量的工作，无论哪个环节出了差错，都会把好事搞砸了。

1.2.1 体系结构设计

软件系统的本质是体系结构，体系结构可以从以下两个方面论述：

1. 体系结构是对复杂事物的一种抽象。良好的体系结构是普遍适用的，它可以高效地处理多种多样的个体需求。一提起“房子”，人们脑中马上就会出现房子的印象（而不是地洞的印象）。“房子”是人们对住宿或办公环境的一种抽象。不论是办公楼还是民房，同一类建筑物（甚至不同类的建筑物）之间都具有非常相似的体系结构和构造方式。如果13亿中国人民每个人都要用特别的方式构造奇异的房子，那么，960万平方公里的土地将会变得千疮百孔，终日不得安宁。

2. 体系结构在一定的时间内保持稳定。只有在稳定的环境下，人们才能干点事情，社会才能发展。科学告诉我们，宇宙间万物无时无刻不在运动、飞行。由于人类的生活环境在地球上保持相对稳定，以至于人类可以无忧无虑地吃饭和睡觉，压根就意识不到自己是活生生的导弹。软件开发最怕的就是需求变化，但“需求会发生变化”是个无法逃避的现实。人们希望在需求发生变化时，最好只对软件做些皮毛的修改，可千万别改动软件的体系结构。就如人们对住宿的需求也会变动，你可以经常改变房间的装潢和摆设，但不会在每次变动时都要去折墙、拆柱、挖地基。如果当需求发生变化时，程序员不得不去修改软件的体系结构，那么，这个软件的系统设计是失败的。

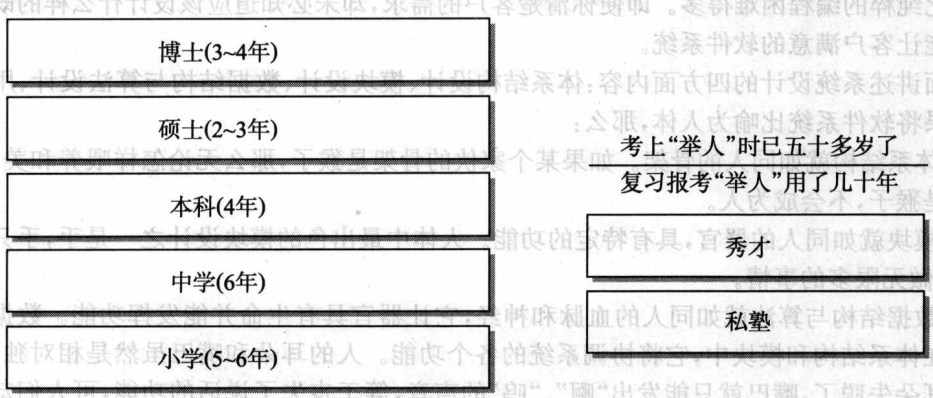
良好的体系结构意味着普适、高效和稳定。下面将介绍两种非常通用的软件体系结构：层次结构和客户机/服务器（Client/Server）结构。

（1）层次结构

层次结构表达了这么一种常识：有些事情比较复杂，无法一口气干完，就把事情分为好几层，一层一层地去做。高层的工作总是建立在低层的工作之上。层次关系主要有两种：上下级关系和顺序相邻关系。

① 上下级关系的层次结构

从小学一直读到博士研究生毕业，要读二十多年，可以分为五个层次。而范进的知识结构只有两层：“私塾”和“秀才”，但读了五十多年，如图1－10所示。一般地说，处于较高层次的学生应该懂得所有低层次的知识，而处于低层次的学生无法懂得所有高层次的知识。图1－10的层次结构存在上下级关系，如同在军队中，上级可以命令下级，而下级不能命令上级。如果把图1－10的层次结构当成是一个软件系统的结构，那么，上层子系统可以使用下层子系统的功能，而下层子系统不能够使用上层子系统的功能。



(a) 从小学读到博士存在的五个学习阶段　　(b) 范进的知识结构

图1－10　上下级关系层次结构图

② 顺序相邻关系的层次结构

顺序相邻关系的层次结构表明通讯只能在相邻两层之间发生，信息只能被一层一层地顺序传递。这种层次结构的经典之作是计算机网络的OSI参考模型，如图1－11所示。为了减少设计的复杂性，大多数网络都按层(Layer)或级(Level)的方式组织。每一层的目的都是向它的上一层提供一定的服务，而把如何实现这一服务的细节对上一层加以屏蔽。一台机器上的第n层与另一台机器上的第n层进行对话。通话的规则就是第n层的协议。数据不是从一台机器的第n层直接传送到另一台机器的第n层。发送方把数据和控制信息逐层向下传递。最底层是物理介质，它进行实际的通讯。接收方则将数据和控制信息逐层向上传递。

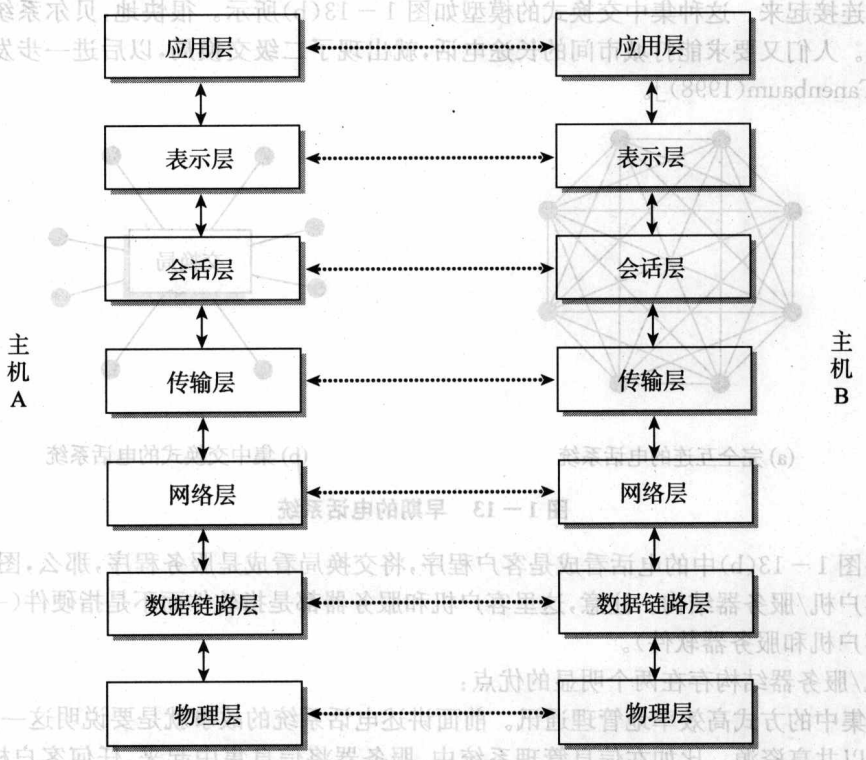


图1－11 计算机网络的OSI参考模型

每一对相邻层之间都有接口。接口定义了下层提供的原语操作和服务。当网络设计者在决定一个网络应包含多少层、每一层应当做什么的时候，其中很重要的工作是在相邻层之间定义清晰的接口。接口可以使得同一层能轻易地用某一种实现(Implementation)来替换另一种完全不同的实现(如用卫星信道来代替所有的电话线)，只要新的实现能向上层提供同一组服务就可以了。

③ 其他层次结构

目前在大型商业应用软件系统中还流行一种包含中间件(Middleware)的层次结构，如图1－12所示[Jacobson(1997)]。中间件支持与平台无关的分布式计算，可以用DCOM和CORBA对象来实现。

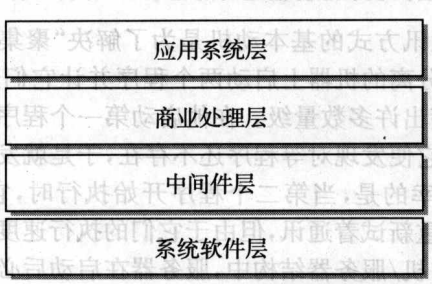
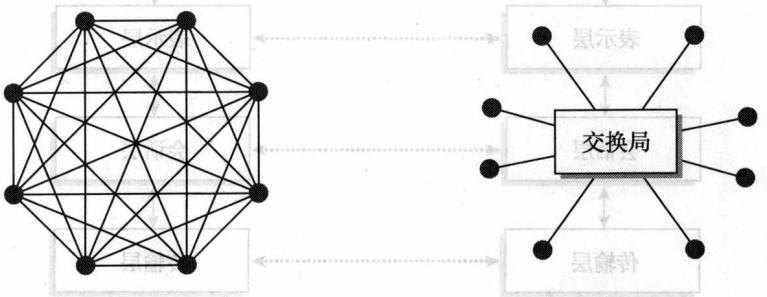


图1－12 包含中间件的层次结构

2. 客户机/服务器结构

先回顾一下早期的电话系统。贝尔(Alexander Graham Bell)于1876年申请了电话专利。那时期的电话必须一对一对地卖,用户自己在两个电话之间拉一根线。如果一个电话用户想和其他几个电话用户通话,他必须拉n根单独的线到每个人的房子里。于是在很短的时间内,城市里到处都是穿过房屋和树木的混乱的电话线。很明显,企图把所有的电话完全互连[如图1-13(a)所示]是行不通的。

贝尔电话公司在1878年开办了第一个交换局。公司为每个客户架设一条线。打电话时,客户摇动电话的曲柄使电话公司办公室的铃响起来,操作员听到铃声以后根据要求将呼叫方和被呼叫方用跳线手工连接起来。这种集中交换式的模型如图1-13(b)所示。很快地,贝尔系统的交换局就出现在各地。人们又要求能打城市间的长途电话,就出现了二级交换局,以后进一步发展为多个二级交换局[Tanenbaum(1998)]。



(a) 完全互连的电话系统　　(b) 集中交换式的电话系统

图1-13　早期的电话系统

如果将图1-13(b)中的电话看成是客户程序,将交换局看成是服务程序,那么,图1-13(b)就是典型的客户机/服务器结构。注意,这里客户机和服务器都是指软件而不是指硬件(一台计算机可以放多个客户机和服务器软件)。

客户机/服务器结构存在两个明显的优点:

(1) 以集中的方式高效率地管理通讯。前面讲述电话系统的故事就是要说明这一点。

(2) 可以共享资源。比如在信息管理系统中,服务器将信息集中起来,任何客户机都可以通过访问服务器来获得所需的信息。

客户机和服务器之间的通讯以"请求—响应"的方式进行。客户机先向服务器发起"请求"(Request),服务器再响应(Response)这个请求,如图1-14所示。

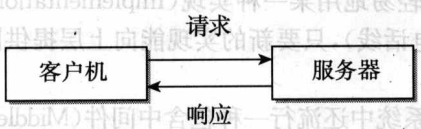


图1-14　客户机和服务器之间的通讯以"请求—响应"的方式进行

采用"请求—响应"这种通讯方式的基本动机是为了解决"聚集"(Rendezvous)问题。为了理解这一问题,设想一个人试图在分离的机器上启动两个程序并让它们进行通讯。还需记住,计算机的运行速度要比人的操作速度高出许多数量级。在他启动第一个程序后,该程序开始执行并向对等程序发送消息。在几个微秒内,它便发现对等程序还不存在,于是就发出一条错误消息,然后退出。此后,他启动了第二个程序。不幸的是,当第二个程序开始执行时,它也找不到第一个程序(早已退出)。即使这两个程序连续地重新试着通讯,但由于它们的执行速度那么高,以至于它们在同一瞬间联系上的概率非常低。在客户机/服务器结构中,服务器在启动后必须(无限期地)等待客户机的"请求",因此就形成了"请求—响应"的通讯方式。

在 Internet/Intranet 领域，目前“浏览器—Web 服务器—数据库服务器”结构是一种非常流行的客户机/服务器结构，如图 1－15 所示。这种结构最大的优点是：客户机统一采用浏览器，这不仅让用户使用方便，而且使得客户机端不存在维护的问题。当然，软件开发和维护的工作不是自动消失了，而是转移到了 Web 服务器端。在 Web 服务器端，程序员要用脚本语言编写响应页面。例如，用 Microsoft 的 ASP 语言查询数据库服务器将结果保存在 Web 页面中，再由浏览器显示出来。

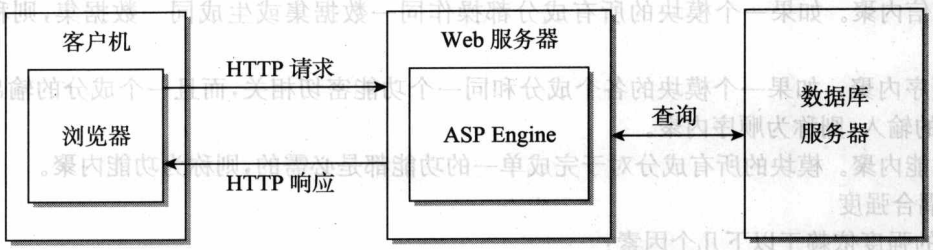


图 1－15 “浏览器—Web 服务器—数据库服务器”结构

1.2.2 模块设计

在设计好软件的体系结构后，就已经在宏观上明确了各个模块应具有什么功能，应放在体系结构的哪个位置。人们习惯地从功能上划分模块，保持“功能独立”是模块化设计的基本原则。这是因为，“功能独立”的模块可以降低开发、测试、维护等阶段的代价。但是“功能独立”并不意味着模块之间保持绝对的孤立。一个系统要完成某项任务，需要各个模块相互配合才能实现，此时模块之间就要进行信息交流。

例如，手和脚是两个“功能独立”的模块，没有脚时，手照样能干活，而没有手时，脚仍可以走路。但如果希望跑得快，那么迈左脚时一定要伸右臂甩左臂，迈右脚时则要伸左臂甩右臂。在设计一个模块时不仅要考虑“这个模块就该提供什么样的功能”，还要考虑“这个模块应该怎样与其他模块交流信息”。

下面将介绍评价模块设计优劣的三个特征因素：“信息隐藏”、“内聚与耦合”和“封闭—开放性”。

1. 信息隐藏

在一节不和谐的课堂里，老师叹气道：“要是坐在后排聊天的同学能像中间打牌的同学那么安静，就不会影响到前排睡觉的同学。”

这个故事告诉我们，如果不想让坏事传播开来，就应该把坏事隐藏起来，“家丑不可外扬”就是这个道理。为了尽量避免某个模块的行为不干扰同一系统中的其他模块，在设计模块时就要注意信息隐藏。应该让模块仅仅公开必须要让外界知道的内容，而隐藏其他一切内容。

模块的信息隐藏可以通过接口设计来实现。一个模块仅提供有限个接口(Interface)，执行模块的功能或与模块交流信息必须且只能通过调用公有接口来实现。如果模块是一个 C++对象，那么该模块的公有接口就对应于对象的公有函数。如果模块是一个 COM 对象，那么该模块的公有接口就是 COM 对象的接口。一个 COM 对象可以有多个接口，而每个接口实质上是一些函数的集合。

2. 内聚与耦合

内聚(Cohesion)是一个模块内部各成分之间相关联程度的度量。耦合(Coupling)是模块之间依赖程度的度量。内聚和耦合是密切相关的，与其他模块存在强耦合的模块通常意味着弱内聚，而强内聚的模块通常意味着与其他模块之间存在弱耦合。模块设计追求强内聚和弱耦合。

(1) 内聚强度

内聚按强度从低到高有以下几种类型：

① 偶然内聚。如果一个模块的各成分之间毫无关系，则称为偶然内聚。

② 逻辑内聚。几个逻辑上相关的功能被放在同一模块中，则称为逻辑内聚。如一个模块读取各种不同类型外设的输入。尽管逻辑内聚比偶然内聚合理一些，但逻辑内聚的模块各成分在功能上

并无关系，即使局部功能的修改有时也会影响全局，因此，这类模块的修改会比较困难。

③ 时间内聚。如果一个模块完成的功能必须在同一时间内执行(如系统初始化)，但这些功能只是因为时间因素关联在一起，则称为时间内聚。

④ 过程内聚。如果一个模块内部的处理成分是相关的，而且这些处理必须以特定的次序执行，则称为过程内聚。

⑤ 通信内聚。如果一个模块的所有成分都操作同一数据集或生成同一数据集，则称为通信内聚。

⑥ 顺序内聚。如果一个模块的各个成分和同一个功能密切相关，而且一个成分的输出作为另一个成分的输入，则称为顺序内聚。

⑦ 功能内聚。模块的所有成分对于完成单一的功能都是必需的，则称为功能内聚。

(2) 耦合强度

耦合的强度依赖于以下几个因素：

- 一个模块对另一个模块的调用；
- 一个模块向另一个模块传递的数据量；
- 一个模块施加到另一个模块的控制的多少；
- 模块之间接口的复杂程度。

耦合按从强到弱的顺序可分为以下几种类型：

① 内容耦合。当一个模块直接修改或操作另一个模块的数据，或者直接转入另一个模块时，就发生了内容耦合。此时，被修改的模块完全依赖于修改它的模块。

② 公共耦合。两个以上的模块共同引用一个全局数据项就称为公共耦合。

③ 控制耦合。一个模块在界面上传递一个信号(如开关值、标志量等)控制另一个模块，接收信号的模块的动作根据信号值进行调整，称为控制耦合。

④ 标记耦合。模块间通过参数传递复杂的内部数据结构，称为标记耦合。此数据结构的变化将使相关的模块发生变化。

⑤ 数据耦合。模块间通过参数传递基本类型的数据，称为数据耦合。

⑥ 非直接耦合。模块间没有信息传递时，属于非直接耦合。

如果模块间必须存在耦合，就尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，坚决避免使用内容耦合。

3. 封闭—开放性

如果一个模块可以作为一个独立体被其他程序引用，则称模块具有封闭性。如果一个模块可以被扩充，则称模块具有开放性。

从字面上看，让模块具有“封闭—开放性”是矛盾的，但这种特征在软件开发过程中是客观存在的。当着手一个新问题时，很难一次性解决问题。应该先综观问题的一些重要方面，同时做好以后补充的准备。因此，让模块存在“开放性”并不是坏事。“封闭性”也是需要的，因为不能等到完全掌握解决问题的信息后再把程序做成别人能用的模块。

模块的“封闭—开放性”实际上对应于软件质量因素中的可复用性和可扩充性。采用面向过程的方法进行程序设计，很难开发出既具有封闭性又具有开放性的模块。采用面向对象的设计方法可以较好地解决这个问题。

1.2.3 数据结构与算法设计

学会设计数据结构与算法，可以编写出高效率的程序。也许有人要问，在计算机速度日新月异的今天，为什么还需要高效率的程序？

计算速度和存储容量上的革新仅仅提供了处理更复杂问题的有效工具，所以高效率的程序永远不会过时。

设计高效率的程序是基于良好的数据结构与算法，而不是基于编程小技巧。大多数计算机科学