软件工程

——方法、工具和实践

冯玉琳 赵保华 编

（中国科学技术大学）

安徽微型计算机函授大学

写在前面

从教学的需要出发，我们编写了此软件工程讲义，这是根据我校近几年来教育软件工程的实践並参考国内外专业文献整理而成，因时间关系，其中难免有疏忽谬误之处，敬请读者指正，至于软件工程的一些高一级材料，我们将在下次印刷时再行增补。

本教程的主要特点是简明，教材内容与教学环节紧密配合，重点介绍软件工程的方法和工具，並强调学生的实践训练，适合于大专院校计算机软件专业作为软件工程课的基本教材或教学参考书。

为顾及一般读者，本教程在整理过程中，力求叙述通俗易懂，重在思想方法介绍，略去带*号的章节，可供具有程序设计初步经验的广大工程技术人员自学之用。

编者衷心悉谢中科院软件所仲萃豪教授、中国科大唐策善付教授、中科院软件所丁茂顺同志、华东计算技术研究所蔡林希同志等在本教程初稿形成过程中的许多有益讨论。悉谢安徽大学李庆同志在组织本教程教学过程中的大力协助，並指导了很好的课题设计实习，悉谢安徽微型机函授大学的同志为本书做了大量实际工作。

编者，1986、8、24。

目　录

第一章 导 言

§1.1 什么是软件工程

软件工程是研究“大”程序设计的方法、工具和管理的工程科学。

首先，软件工程的要点是“大”程序设计。假设一个人在一个月内能够完成1000行代码的设计，按照如此的工作效率，一个10，000行代码的程序是否一个人花十个月或者十个人花一个月就能完成呢？回答是否定的。当程序的规模从1000行增加到10，000行时，源代码行增加了十倍，但是，程序复杂性程度的增加却远不止十倍。

当许多人共同设计一个由许多模块组成的大型系统时，在许多年的时间会出现有许多不同的版本，面对着如此的许多人、许多模块、许多年和许多版本…，如此之“多”的活动，如何进行设计、开发和维护，这里不仅涉及到技术的方面，如设计方法、版本控制等等，而且关系到行政管理的方面，如价格评估，人员组织等。

考虑到研制一个软件系统同研制一台机器或建造一座楼房有许多共同之处，因此可以参照机械工程、建筑工程中的一些技术来进行软件的研制，人们试图用“工程化”的思想作指导来解决软件研制中面临的困难和混乱。1968年北大西洋公约组织的学术会议第一次使用了“软件工程”这个名词。

正象所有的工程科学一样，软件工程自身遵循着一套科学的设计原理和方法，以这些方法为前导，人们发展有一系列的软件工具系统来帮助工程软件的开发。但是，软件工程又与其它的各种工程科学很不相同。软件是抽象的、逻辑性的产品，不是实物性的。研制和维护软件本质上是一个“思考”的过程，很难对它进行控制。软件的设计基本上无统一的设计标准，无准确的数量分析，无足够的可靠性保证以及无有效的维护手段，这就决定了软件的研制和开发较之其它工程项目要困难得多。

综言之，软件工程的目标在于研究一套科学的工程方法，並与此相适应，发展一套方便的工具系统，力求用较少的投资获得高质量的软件。

本讲义是软件工程的一本简明教程，目的在于使学生对软件工程科学的全貌有个概要的了解，从而使学生的注意力从个人小程序的独立活动转移到或者扩展到大程序设计方面。

与其它一些课程的特点不同，软件工程课是一门“设计”课程，强调实际训练。课堂教学只讲授软件工程学中的一些基本概念、方法和原则，更重要的是，学生必须参加一个从计划、分析、设计到测试的软件开发的全过程，以便从中得到实际的从事软件工程的训练和经验。课程将提出一些设计题目供学生选用。

§1.2软件工程面临的问题

摆在软件工程科学面前有许多方面的棘手问题，以下就软件价格、软件可靠性、软

件维护、软件生产率和软件再应用等几个方面作一些简单分析。

1.2.1 软件价格

由于新的电子器件的出现，计算机硬件的功能和质量在不断提高，而成本却大幅度下降。计算机硬、软件投资所占比例中，软件投资在迅速上升。据1972年U.S.Air Force发布，美国空军计算机投资中软件所占比例，1955年低于20%，1970年剧增到60%，预计1985年为85%，软投资比重上升的曲线如图1.1所示。

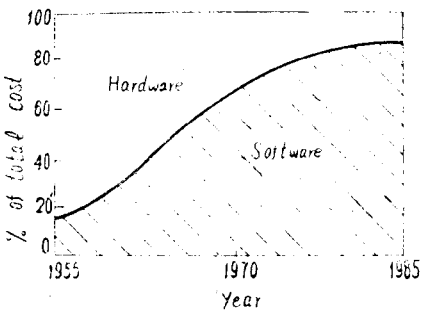


图1.1 软硬件投资比例

再看纽约Wall Street Joural, Jan。23，1979年公布的Carter 1980年财政年度预算，总计共有570亿美元用之于发展计算机系统，其中320亿美元，即56%用之于发展计算机软件。

随着计算机应用愈来愈广泛，新的应用领域要求开发新的软件系统。另一方面，当新一代的计算机取代现有的旧的计算机时，软件系统也要随之更新。而程序设计是高度劳动密集型的技术，所以，软件价格上升的势头仍将会继续下去。

1.2.2 软件可靠性

指软件系统能否在既定的环境条件下运行并实现所期望的结果。通常大约有40%左右的软件开发的代价要花在软件设计之后的测试和排错上，即使如此，也不能保证经测试的软件就没有错误。现代的程序设计技术，如结构程序设计，能有助于减少程序设计中的错误；但是不幸的是，我们很难保证一个大的软件系统确实是无任何错误的。

大的软件系统的测试是一件很复杂很费时的事情，程序的任何改变可能会涉及到许多人，这就使得软件系统花在测试阶段的代价非常之大。为了提高软件的可靠性，就要付出相应的代价，重要的是在可靠性和代价之间作出权衡。

除了软件的正确性以外，在更广泛的意义下，软件的可靠性还应该包括安全性和健壮性。

对于合理的一组输入，系统会给出正确的结果；而对于用户的有意或者无意的不合理输入，系统应能拒绝这种输入，並指出输入的不合理性，提醒用户注意。这是软件系统的安全性。对于不合理的输入束手无策，甚至导致系统失败，这样不安全的系统是不能使用的。

健壮性是指软件系统对环境变化的适应性。当软件系统所处的环境发生变化时，例如存贮溢出，硬件故障等意外事件发生时，系统都能按照某种预定的方式作适当处理，有效地控制事故的蔓延，不致丢失重要的信息，从而避免灾难性的后果。

1.2.3 软件维护

软件维护是指修正已经发行了的软件系统所需要做的工作。几乎所有的软件系统在运行了若干年以后，仍然需要作这样或者那样的修改和补充。一个软件开发机构，可能把60%以上的精力用在维护已有的软件上。随着更多软件的生产，这个百分比还可能继续

增加，这样势必会束缚开发组织无法去生产新的软件。

为什么维护需要花费如此大的精力呢？这是因为已经运行的程序还总是在变化，故障要排除，改进要加进去，而且优化工作也要做。不仅当前的版本要改变，仍在使用的去年的版本也要修改，可能，即将投入使用的明年的版本也要修改，除了解决原有的问题需要精力外，改变本身又会带来新的问题，也要精力去解决。

因此，如何减少软件维护的总工作量，也就成为软件工程的主要目标之一。

1.2.4 软件生产率

计算机的广泛应用使得软件的需求量迅速上升。世界各国普遍感到软件人力资源的缺乏，这种趋势仍在继续增长下去。但是，目前的软件生产仍然基本上处于手工状态，生产率很是低下，而生产出来的软件的质量又很不理想，不适应市场对软件的大量需求。如何以较少的投资，获得“高产优质”的软件，这是软件工程的主要目标之一。其实现途径有二，即良好的软件工程设计方法和方便的软件设计工具环境。

软件工程设计方法告诉人们“什么时候做什么？什么时候怎么做？”。由于软件研制过程毕竟是相当复杂的，涉及的因素很多，所以这些软件方法又有不同程度的灵活性与试探性。一般说来，软件方法规定了：

△明确的工作步骤

△具体的文档格式

△确定的评价标准

近十几年来，软件工程已研究发展了各种各样的软件工程设计方法，这些方法的实用范围是各不相同的，有的方法适用于数据处理系统，而有的方法适用于实时控制系统；各种软件方法的风格也迥然不同，有的方法仅仅是一组指导性的原则，而有的方法则有较具体的处理规则，有的方法建立在严密的数学基础之上，而有的方法则是实际经验的总结。本课程只能选择一些比较实用的、有代表性的方法进行讨论。

方法和工具两者之间有着密切的联系，相辅相成。在软件开发维护的不同阶段，都应该有这样或者那样的作为软件工具的程序系统，帮助人们自动地完成许多数据分析和处理工作。这样的工具环境愈完善，软件的设计和维护也就愈方便。这是当前提高软件生产率的一个很重要的方面。人们希望研究出一套系统的软件方法，一套配套成龙的软件工具，从而为软件人员提供一个能复盖整个软件生命期的良好的工作环境，软件生产率必将大为提高。

1.2.5 软件再应用

软件再应用也是提高软件生产率、降低软件成本的一个重要方面。当人们一次又一次的去设计一些互相类同的程序时，很多的劳动都花费在一些基本上重复的工作上，软件再应用的技术就是旨在减少这种重复。

软件设计中可以发现许多基本上是固定的模式。例如，对于不同语言的结构化编辑，除了语言的结构框架或者模板不同之外，编辑系统其它部分的设计基本上都是类同的；对于高级程序语言编译中的语法分析和代码生成也是如此。我们能否设计出一个关于一定的结构模式的系统，给定一些特定的具体说明和要求，这样的系统就能生产出特定的软件来。通常所说的许多程序自动生成系统，就是这种软件再应用的例子。

与软件再应用密切相关联的另外两个方面是软件移植和软件规范化。

成功的软件移植的例子是UNIX。七十年代初UNIX操作系统首先由Bell实验室设设，而现在它已风靡美国和全世界。UNIX已移植在许多种不同型号的机器上而得到广泛的应用。

软件规范化的成功的例子是一些数学软件，如标准函数计算，统计软件，矩阵计算等。至于程序设计语言的标准化，早期有FORTRAN，ALGOL，COBOL，PL／1，PASCAL等，还有八十年代初美国国防部制定的通用语言ADA，但在其它方面，软件标准化的进程仍然很迟缓。原因在于软件所面向的应用领域太广泛了。人们期望会有愈来愈多的规范化或者半规范化的软件问世。

软件再应用是近几年来愈益受人们重视的一个新课题，预料将会有很强的生命力。

以上提出了软件工程所面临的几个主要问题，可以说，这些问题均还未得到很好的解决。软件工程学还很年青，有待进一步的发展和提高。

§1.3软件和软件生命期

什么是软件？很难对这个名词以精确的定义，Boehm指出，软件是程序以及开发、使用和维护程序所需要的所有文档。

根据这样的定义，软件不再仅仅是程序，研制软件也就不仅仅是编写程序。

为了用工程化的方式有效地管理研制软件的全过程，软件系统的生命期可以分为如下六个阶段：

（1）软件计划

在设计任务确立之前，首先要进行调研和可行性研究，理解工作范围和所花代价，作出软件计划。

（2）软件需求分析

对用户要求具体进行分析，並用软件需求规范书表达出来，作为用户和软件人员之间相互共同的约定。

（3）软件设计

设计阶段包括总体设计和详细设计。总体设计决定系统的模块结构，並给出模块的相互调用关系、模块间传送的数据以及每个模块的功能说明。详细设计则考虑每一模块的内部实现算法。

（4）软件编码

编码阶段的任务是按软件设计说明书的要求为每个模块编写程序。

（5）软件测试

测试的任务是发现和排除程序中留存的错误，分模块测试和联合测试两种，经过测试排错，得到可运行的软件。

（6）软件维护

经过测试的软件仍然可能有错，用户需求和系统的操作环境也有可能发生变化，因此，交付运行的软件仍然需要继续排错、修改和扩充，这是软件的维护。

表1.1列出每个阶段的基本任务，上一阶段完成确定的任务后，都要产生一定格

式的文档交给下一阶段。

阶　段	基本任务	工作结果
计　划	理解工作范围	计划任务书
需求分析	确定设计要求	需求规范书
设　计	建立系统结构	设计说明书
编　码	编写程序	程　序
测　试	发现和排除错误	可运行的系统
维　护	运行和管理	改进的系统

表　1

软件生命期划分为上述六个阶段，这就为工程化地研制软件提供了一个框架。但是，必须指出，实际系统的研制工作，不可能是直线进行的，常常存在着反复。研制人员往往需要从后面的阶段回复到前面。例如，在设计阶段发现需求规范书有不完整或者不精确之处，就需要回到需求分析阶段进行“再分析”，测试阶段发现了模块内部或者接口中的错误，就要回到设计阶段对原来的设计进行修正。事实上的工程设计流程如图1.2所示。

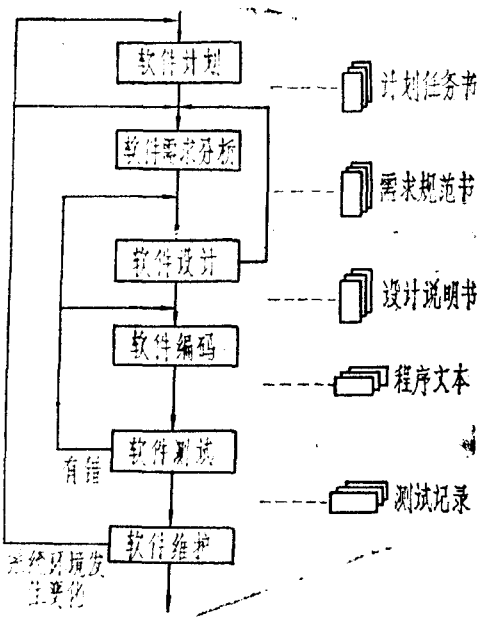


图　1.2

软件生命期的前五个阶段，即计划、分析、设计、编码和测试，常称之为软件的开发期。最后一个阶段称之为软件的维护期。在软件开发期中，测试阶段的工作量最大，约占到开发期总工作量的40%，而在软件的整个生命期中，维护的周期最长，工作量也非常大。

值得指出的是，我们常常会碰到许多所谓的小项目软件工程，这些项目的工作量不大，从计划到测试的全过程可能就是由一、二个人去完成的，即使是这样的情形，也必须按照软件工程的方法和步骤进行开发，只不过是其中的有些部分可以简略，有些文档可以缩短和合并，但总的说，软件工程的原则不变。

第二章 软件概念

§2.1 软件的质量标准

软件工程学的目标之一是获得高质量的软件。如何评价软件的质量呢？随着计算机软件和硬件技术的发展，好软件的质量标准也有所改变。在计算机发展的早期，计算机的内存容量很有限，执行速度也不高，人们设计软件时特别强调效率。近十年来，随着软件规模和复杂性的增加，人们倾向于从以下几方面对软件质量作比较全面的评价。

首先，软件应能按照既定的要求进行工作，在功能和速度方面都符合设计要求。软件系统能够可靠地进行工作，不仅表现于在合法的输入情况下它能够正确的运行，而且它还应能够安全的排除非法的侵入和处理意外事件，保证系统不致受到损害。

第二，软件应是好结构的，一方面指软件系统的内部结构清晰，软件人员易于阅读和理解，从而方便于软件的修改和维护；另一方面还指系统的人机界面清晰，对用户是“友好”的，用户乐于使用。

最后，软件必须文档齐全，软件不仅仅是可执行的程序代码，而是应包括软件开发过程中所产生的所有文档，这些文档资料是软件维护所不可缺少的。

以上几个好软件的质量标准是相互联系的，也是相辅相成的。但是，值得指出的是，软件质量的一些指标常常又是矛盾的。例如，片面强调了执行效率，设计出来的软件就可能结构复杂，难于理解，也难于修改维护。追求可靠性一般的也要以一定的时间和空间作为代价。在实际的软件开发过程中，软件人员应充分考虑各种可能的候选方案，在各种矛盾的目标之间作权衡，并在一定的限制条件下（如研制经费、研制时间、可使用的资源等）使所期望的诸目标最大限度地得到满足。

综上所述，一个大型软件系统的质量应该从可靠性、易理解性、易维护性和效率等几方面全面地进行评价。为此，在分阶段介绍软件工程设计规范之前，有必要对软件本身作一些定性和定量分析，以加强对什么是好软件质量的认识。

§2.2 软件结构

这一节是对软件结构的一种定性分析。

模块化结构是所有设计良好的软件系统的基本属性。任何一个大的程序系统，总是由若干功能相对独立的模块组成的。

2.2.1 结构和过程

了解结构和过程间的差别，是了解软件总体设计和详细设计的前提。

软件结构的层次模型指出了软件的各个组成模块之间的关系。软件结构的导出由问题定义开始。当问题的每一部分由一个或多个软件元素解决了的时候，整个问题也就解决了，相应的解决此问题的软件结构也就得到了。这个过程代表了由软件需求分析到设计的过渡。

软件结构表示程序的总体结构，隐含地指出程序的控制层次体系。它並不表示软件的过程方面，诸如处理的序列、判定的出现和次序、或者操作的重复，所有这些过程属性，在软件结构中並无反映。

好的软件结构体现自顶向下的方式分配控制。软件结构的如此分解，不仅可以简化软件的设计和实现，加强可测试性，而且能够以一种更为有效的方式进行维护。

一个“问题”可以由许多不同的软件结构来解决。如图2.1所示。不同的软件设计方法学根据不同的原则，因此，对于同样的一组软件要求，可能会得出不同的软件结构。

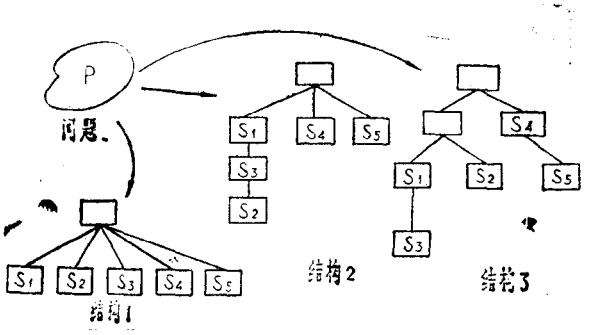


图 2.1

关于软件结构，常使用许多术语。在结构图中的每一个方框表示一个模块——程序的可单独编址元素。深度表示控制的层数，宽度表示控制的总跨度。“扇出”是对直接由一个模块所控制的模块数目的度量。“扇入”指出有多少模块直接控制一个给定的模块。如图2.2。

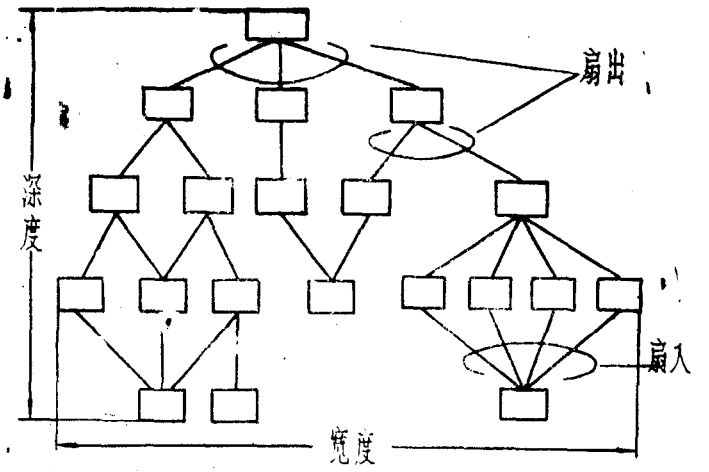


图 2.2

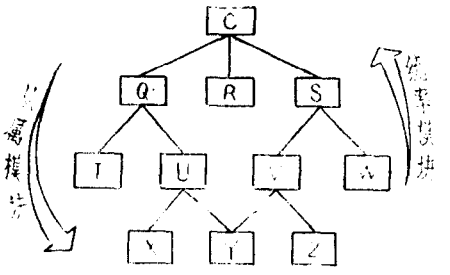


图 2.3

模块之间的控制表现为统率和从属的关系。如果一个模块控制另一模块便称前一模块“统率了”后一模块，反之，我们称后一模块“从属于”前一模块。如图2.3，模块C统率了模块Q，R和S，模块V从属于模块S並且最终从属于模块C。

2.2.2 模块化

在计算机软件中，模块化的概念已经采用了二十多年，上节所讨论的软件结构也正是体现了模块化，也就是说，软件被划分成若干可单独命名和编址的元素，它们被称作模块，这些模块组成整体以满足问题的要求。

模块化是使得程序能够对付复杂问题所应具备的属性，也是使得程序能够被有效地管理维护所应具备的属性。如果一个大的程序仅由一个模块组成，它是很难被人所理解的。为了阐明这一点，我们如下考虑基于问题求解的一些观察结果。

令C(X)是确定问题X的复杂程度的函数，E(X)是决定解决问题X所需工作量(按时间计算)的函数。对于两个问题P_1和P_2：

如果　$C(p_1)>C(p_2)$，

显然有　$E(p_1)>E(p_2)$，

因为解决一个困难的问题确实需要更多的时间。

根据人类问题求解的经验，另一个有趣的规律是

$$C(p_1+p_2)>C(p_1)+C(p_2)$$

也就是说，如果一个问题由p_1和p_2组合而成，那末它的复杂程度大于分别考虑每个问题时的复杂程度之和。最后，我们得到下述不等式

$$E(p_1+p_2)>E(p_1)+E(p_2)$$

这就是“分而治之”的策略。把一个复杂的问题划分成许多可对付的小问题，它就比较容易求解了。

由上面的不等式似乎能得出结论说：如果把软件无限细分，那末最后开发软件所需的工作量就小得可以忽略了。不幸的是，其它的因素也在起作用，从而使得上述结论不能成立。参考图2.4，当模块总数增加时，开发一个单个软件模块的工作量(代价)确实减小了。然而，随着模块数目的增加，与模块接口有关的工作量(代价)也随之增加。这些因素导致了图2.4中的总工作量(代价)曲线。存在一个模块数目M，使得开发总代价为最小。

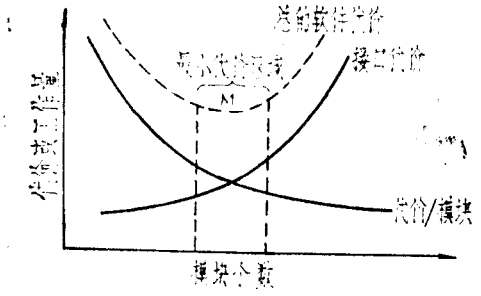


图 2.4

虽然我们并不能精确地决定M的数值，但是在考虑模块化的时候，总成本曲线为我们提供了有用的指南。

重要的是要注意，即使一个系统的实现必须是“整体”的，它的设计仍然可以是模块化的。有些情形(如实时软件和微处理机软件)，虽然子程序的应用会带来速度降低、存贮量加大等问题，但是，软件仍然可以而且应该根据模块化的原则来设计；程序源代码可以是一行接着一行，乍看起来不是模块化的，但由于设计时已遵循模块化的基本原则，这样的程序仍将具有模块化的许多优点。

模块的最重要特征有二：一是抽象，二是信息隐藏。

模块反映了数据和过程的抽象。在模块化问题求解时，在最高抽象级可以采用面向问题的环境语言的抽象术语进行叙述；而在较低抽象级，则可采用过程性术语。模块化

的概念加上逐步求精的方法就把面向问题的术语和面向实现的术语两者结合起来，前者是后者的一种抽象。

模块的另一重要特征是信息掩藏。一个模块内部所包含的信息（数据或过程），如果它不允许外部的模块访问的话，其它模块是不能对它们访问的。“信息隐藏”意味着有效的模块化可以通过定义一组独立的模块来实现，这些独立的模块彼此之间仅仅交换那些为了完成系统功能所必须交换的信息。

抽象和信息掩藏从两个不同方面说明了模块化设计的特征。“抽象”帮助定义构成软件的过程实体，而“信息掩藏”实施对过程细节的存取约束。所有这些都给模块化设计带来莫大裨益。

2.2.3 模块独立性

模块独立性的概念是模块抽象和信息掩藏的直接结果。

发展具有独立功能且与其它模块没有过多相互作用的模块就可以做到模块独立。也就是说，我们希望这样设计软件，使得每个模块与精化的一个特定子功能相联系，而且与软件结构的其它部分具有简单的接口。

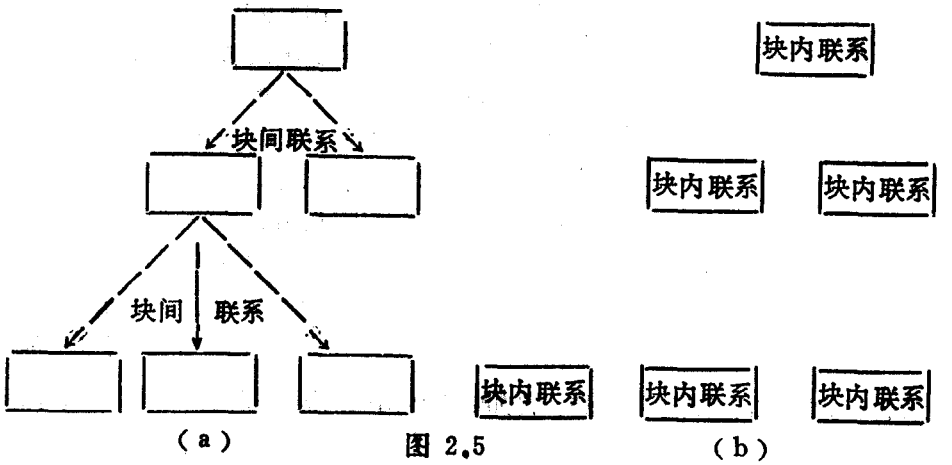
模块独立性是好的软件质量的关键。具有独立模块的软件容易开发，这是由于能够对软件的功能加以分割，而相互接口又不复杂，可由一组人员同时开发。由于模块相互独立，在各自设计和修改代码时所引起的二次影响不大，错误传播少。

模块独立性有两个定性的标准度量，即内聚（也称块内联系）和耦合（也称块间联系）。

块间联系是指模块之间的联系，如图2.5（a）。它是对模块独立性的直接衡量，很显然，块间联系越小则模块的独立性越高。

块内联系是指一个模块内部各成分（语句或语句段）之间的联系，如图2.5（b）所示。如果一个模块的块内联系大，模块的独立性则会提高。

为得到好的模块结构，理想的情况是块间联系尽可能小，块内联系尽可能大。实际上，块间联系和块内联系是同一事物的两个方面，程序中各组成成分是有联系的，如果将密切相关的成分分散在不同的模块中，就会造成很大的块间联系，反之将密切相关的成分放在同一模块中，块内联系大了，相应地块间联系就会小些。



（a）　　图 2.5　　（b）

下面再详细讨论一下耦合（块间联系）和内聚（块内联系）的类型。

一、块间联系的类型

块间联系按从小到大排列有下述七种。

1、无块间联系

图2.6中，模块C和D分别从属于模块A和B，C和D之间又没有联系，相互独立，称它们无块间联系。

2、数据性块间联系

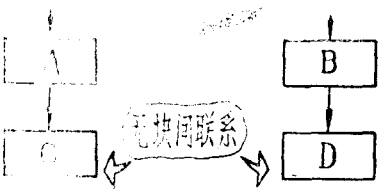


图 2.6

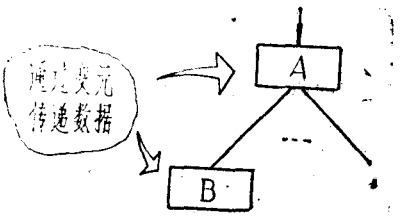


图 2.7

图2.7中模块B从属于模块A，A对B的访问是通过一个约定的变元表进行数据传递的，这里只传递数据。称A和B之间的联系为数据性块间联系。

3、标志性块间联系

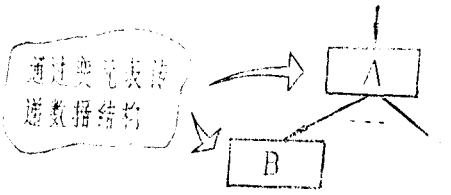


图 2.8

图2.8中A对B的访问通过一个模块接口界面传递数据结构的一部分，不是简单变元。称A和B之间的联系为标志性块间联系。（有些资料中把这种块间联系称为是数据性块间联系的变形，不单独列出）

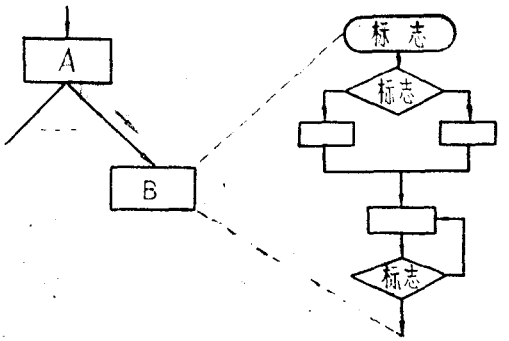


图 2.9

4、控制性块间联系

图2.9中，模块A传递一个“标志”给模块B，通过这个“标志”控制模块B根据该“标志”进行抉择。A和B间的联系称之为控制性块间联系。

5.外部块间联系

当模块受软件外部环境约束时，就会出现外部块间联系。

图2.10中，模块A和B都访问同一个有名公共区，就是其中的一种。再如输入输出把模块联接到指定的设备、固定格式以及通讯约定。外部联系是必要的，但是在一个结构内应该限制在少数模块内。

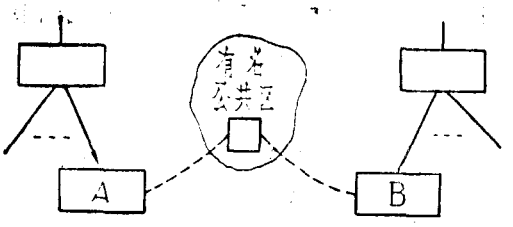


图 2.10

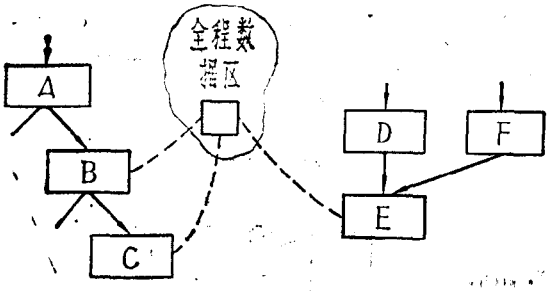


图 2.11

6、公共块间联系

当两个以上模块引用一个全程数据区时，就产生公共块间联系。在图2.11中，模块B、C和E都存取全程数据区中的一个数据项，如一个磁盘文件。

假定模块B读该项，调用模块C对该项重新计算並进行更新。如果C错误地更新了该项，再往下的处理中，模块E读该项，就要发生错误。从表面上看，问题发生在模块E，实际上问题出在模块C。所以，在具有大量公共耦合的结构中，诊断错误是费时和困难的。但是，这並不是说，使用全程数据必定不好，问题是软件设计者必须了解公共耦合可能造成的后果，设计时要特别小心，防止出错。

7、内容块间联系

当一个模块使用保持在另一个模块内部的数据或控制信息时，或者转移进入另一个模块中间时，是最高程度的耦合。这种耦合叫内容耦合（内容块间联系），应该避免使用。

总之，最好一个模块只做一件事情。如果一件事情由几个模块来完成，互连就高了，块间联系就高，这是一般情况。特殊情况，为了减少接口代价，就要合並一些。

二、块内联系的类型

块内联系由小到大排列有下述六种。

1.偶然性块内联系

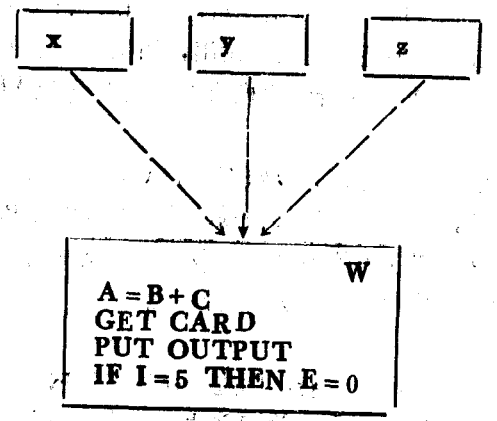


图 2.12

图2.12模块W中的几个语句间实际上无任何联系，它们是因为模块X、Y、Z中都含有这几个语句，为了节省空间将它们放在模块W中，W的块内联系就属于“偶然性块内联系”。

偶然性块内联系因各成分之间没有联系，此模块很难理解、测试和维护；另一