第5章 chapter 5

信息系统设计

在经过结构化系统分析阶段的各项工作建立新系统的逻辑模型，解决了系统“做什么”的问题后，如果审议通过了系统分析报告，那么接下来的工作就是进行系统设计。结构化系统设计是结构化生命周期法中的又一重要阶段，这一阶段要从系统逻辑模型出发，在已获准的系统分析报告的基础上，结合实际的经济、技术条件及时间要求，进行系统物理模型设计，解决系统“如何做”的问题，为后续各项系统实施工作做好具体实施方案。

5.1 系统设计概述

系统设计又称物理设计，就是根据新系统逻辑模型所提出的各项功能要求，结合实际条件，科学、合理地设计出新系统的解决方案，并且为系统实施阶段的各项工作准备好必要的技术资料和有关文件。

系统设计通常可分为两个阶段进行。首先是总体设计，其任务是设计系统的框架和概貌并向用户单位和领导部门作详细报告；若获得认可，在此基础上进行第二阶段——详细设计。这两部分工作是互相联系的，需要交叉进行。概要设计包括系统结构设计、网络设计、数据库设计、输入输出设计等；详细设计则是对概要设计的结果进一步细化，包括处理过程设计、详细的数据库设计、详细的输入/输出设计等内容。本章将这两部分内容结合起来进行介绍。

系统设计工作由开发人员负责，他们将系统分析阶段得到的目标系统的逻辑模型转换为目标系统的物理模型，该阶段得到的工作成果——系统设计说明书是下一阶段——系统实施的工作依据。

5.1.1 系统设计的目标

系统设计的目标是在保证实现逻辑模型功能的基础上，尽可能提高目标系统的性能，使所设计的系统安全可靠、易于理解、便于维护并具有良好的经济性，将分析阶段所获得的系统逻辑模型转换成一个具体的计算机实现方案的物理模型，包括计算机物理系统配置方案报告和一份系统设计说明书。

系统设计的目标是评价和衡量系统设计方案优劣的基本标准，也是选择系统设计方案的主要依据。评价与衡量系统设计目标实现程度的主要指标有以下几方面。

（1）系统的可靠性

系统的可靠性是指系统抵御外界干扰的能力及受外界干扰时的恢复能力。可靠性是对系统的基本要求。

对系统的外界干扰来自很多方面，大致可分为对硬件、软件以及数据的干扰。这些干扰可能是无意的操作错误或恶意的侵入与篡改，也可能是外界不可控因素造成的。

提高系统的可靠性可以从立法，系统的硬件、软件、运行环境以及运行规程等多方面综合考虑。要选用可靠性较高的设备，适当考虑硬件结构的冗余度。在软件中设置身份验证及数据操作校验等各种检验及保证措施，以防止误操作和非法使用。要设置防火墙及杀毒软件等各种安全保障措施，制定明确的规章制度及运行规程。

（2）系统的可变更性

系统的可变更性指系统的可维护性或可修改性。系统投入运行后，由于系统的环境和条件会不断变化，系统在设计上的缺陷和功能上的不完善以及在使用过程中出现的硬件、软件故障等会影响系统的正常运行。因此，系统要不断修改和完善。可变更性强的系统便于维护和扩充完善。软件设计水平是影响系统可变更性的主要因素。结构化模块设计、提高数据存储结构规范化程度、系统功能设计的前瞻性等都是提高系统可变更性的重要措施。

（3）系统的效率

系统的效率可以通过系统对处理的响应时间或单位时间内处理的业务量进行衡量。系统的效率主要与硬件平台的选择、系统软件的性能、参数的设置情况、应用软件结构设计的合理性及中间文件调用的次数和数量等因素有关。

（4）系统的通用性

系统的通用性是指同一软件系统在不同使用单位中的可应用程度。提高软件系统的通用性可以扩大它的应用范围，降低研发成本，减少系统扩充时的工作量和费用，增强系统的生命力。这一指标对于商品化软件尤为重要。为提高系统的通用性，要进行充分的系统分析，使业务处理规范化、标准化、完善化。

（5）系统的工作质量

系统的工作质量是指系统处理数据的准确性、输出各种信息的易懂性和系统操作的方便性等。系统的工作质量与系统的硬件设备及软件质量有直接关系。系统设计阶段的各项工作几乎都与系统的工作质量有关，直接影响系统的使用效果。因此，在系统设计时既要考虑到实现系统功能的要求，又要考虑到使用者的要求和反应。

5.1.2 系统设计的原则

为保证系统设计的质量，在系统设计时要遵循以下原则。

① 系统性原则。系统是一个有机整体。因此，要从整个系统的角度进行考虑，系统中的信息代码要统一，设计规范要统一、标准，对系统的数据采集要做到数出一处、全局共享，使一次输入得到多次利用，系统功能应尽量完整。

② 灵活性原则。管理信息系统是需要修改和维护的。因此，系统设计人员要有一定的预见性，要从通用的角度考虑系统设计，系统应具有较好的开放性和结构的可变性，采用模块化结构，提高各模块的独立性，尽可能减少模块间的耦合，使各子系统间的数据依赖减至最低限度。

③ 可靠性原则。一个成功的信息系统必须具有较高的可靠性，如安全保密性、检错及纠错能力、抗病毒能力、系统恢复能力等。只有系统是可靠的，才能得到用户的认可与接受。

④ 经济性原则。经济性指在满足系统需求的前提下，尽可能减小系统的开销。一方面，在硬件投资上不能盲目追求技术上的先进，而应以满足应用需要为前提；另一方面，系统设计中应尽量避免不必要的复杂化，各模块应尽量简洁，以便缩短处理流程，减少处理费用。

⑤ 管理可接受原则。一个系统能否发挥作用和具有较强的生命力在很大程度上取决于管理上是否可以接受。因此，在系统设计时，要考虑到用户的业务类型、用户的基础管理工作、用户的人员素质、人机界面的友好程度、掌握系统操作的难易程度等诸多因素的影响，设计出用户可接受的系统。

5.1.3 系统设计的内容和步骤

系统设计一般分为初步设计和详细设计两个阶段。初步设计又称总体设计或概要设计，它的主要任务是完成系统总体结构和基本框架的设计。详细设计的主要任务是在系统初步设计的基础上，将设计方案进一步具体化、条理化和规范化。具体来说，系统设计的主要内容可以概括如下。

① 系统总体结构设计。系统总体结构设计是指根据系统分析阶段确定的新系统的目标和逻辑模型，科学合理地将系统划分成若干子系统和模块，确立模块间的调用关系和数据传递关系。

② 处理流程设计。主要包括系统处理流程设计和模块处理流程设计两方面。

③ 代码设计。为系统处理的实体或属性设计易于处理和识别的代码。

④ 人机界面设计。从系统角度出发，按照统一、友好、漂亮、简洁、清晰的原则设计人机界面。

⑤ 输出设计。根据用户的要求设计报表或其他类型输出信息的格式及内容。

⑥ 输入设计。设计系统运行所需输入的各种数据的输入格式，使其操作简单。

⑦ 数据库设计。根据数据字典和数据存取要求，确定数据库的结构。

⑧ 安全保密设计。为确保管理信息系统的运行安全和数据保密，提出安全保密设计方案。

⑨ 系统物理配置方案设计。进行具体的计算机软硬件及其网络的选择和配置。

⑩ 编写系统设计说明书。将系统设计阶段的各种资料进行整理，按照规定的格式编写，为系统实施提供依据。

5.2 系统结构设计

系统结构设计也称系统总体结构设计，是指根据系统分析阶段确定的新系统的目标和逻辑模型，科学合理地将系统划分成若干子系统和模块，确立模块间的调用关系和数据传递关系。

一个实际应用的信息系统通常要支持复杂的业务，需要实现业务需要的全部功能，其包含的软件数量是庞大的。这就需要对这些软件进行良好的组织，使系统结构合理，方便用户使用，运行效果良好。因此，对复杂的系统首先要进行子系统的划分，然后针对每个子系统再进行功能模块的划分及细化，完成系统结构的设计，使系统具有合理的功能结构。

系统结构设计的好坏将直接影响系统的质量和整体特性，在系统结构设计中要牢记“整体大于部分之和”思想，力求系统的整体性能最佳而不是各个局部性能最佳。

5.2.1 子系统划分

子系统划分就是综合考虑管理要求、业务特点、环境条件和开发工作等各方面因素将系统划分成若干相对独立的子系统。

在系统分析阶段，已经进行了初步的子系统划分，本阶段是从计算机实现的角度出发，对系统分析阶段划分的子系统进行校核，使其界面更加清楚和明确。目前，对子系统的划分并没有公认的准则，可以按功能、按业务先后顺序、按业务处理过程、按实际环境和网络分布等方法进行子系统的划分，而且在实际划分时还会受到个人实际工作经验及对问题理解程度等因素的影响。尽管没有确定的标准，但在进行系统划分时要遵循以下得到普遍认可的原则。

1. 划分的结构要易于理解

所划分的各子系统功能要明确，规模要适中且均衡，减少复杂性。划分时在合理可行的前提下应尽可能考虑现行系统的结构和用户的习惯，使划分的子系统易于用户理解和接受，便于新旧系统的转换。

2. 各子系统要具有相对独立性

尽量使子系统在逻辑上相对独立，让每个子系统支持某一方面的管理功能，以便于系统的分阶段实施。

3. 减少各子系统之间的依赖性

子系统之间的联系要尽量减少，接口要简单、明确。要尽量将联系较多的模块都划入子系统内部，使子系统内部联系强，这样必然会减少各子系统之间的联系。

4. 子系统的划分应考虑今后发展的需要

应充分考虑组织未来业务发展的需要，使子系统具有扩展性，如对外的接口、通信以及业务逻辑的可扩展性，并且适当考虑一些更高层次的管理决策需求。

5. 子系统的划分结果应尽量减少数据冗余

子系统划分不合理会使相关数据分布到各个不同的子系统中，这会导致子系统之间数据传递量增大或在子系统之间存在重复数据，增加系统的复杂性及易出现数据不一致性错误。

6. 应考虑各类资源的充分利用

恰当的系统划分应该既有利于各种设备资源的搭配使用，又能使各类信息资源的分布合理和充分使用，有效减少系统对网络资源的过分依赖，减少输入、输出、通信等设备压力。

5.2.2 功能模块划分

所谓功能模块划分是在子系统划分的基础上，以系统的逻辑功能和数据流关系为基础，借助于一套标准的设计准则和图表工具，通过“自上而下”和“自下而上”的多次反复，把各子系统分解为若干大小适当、功能明确、具有一定的独立性且容易实现的模块，从而把复杂系统的设计转变为多个简单模块的设计。合理地进行模块的分解和定义是系统结构设计的主要内容。

1. 模块及模块化

模块是系统结构中的基本组成单位，模块化是进行系统结构设计的重要指导思想。

1）模块

模块是组成目标系统逻辑模型和物理模型的基本单位，是可以组合、分解和更换的单元。系统中的任何一个处理功能都可以被看成一个模块。

一个模块应具备四个要素。

① 输入和输出。模块的输入来源和输出去向都是同一个调用者，即一个模块从调用者那里取得输入，进行加工后再把输出返回调用者。

② 处理功能。它指模块把输入转换成输出所做的工作。

③ 内部数据。它指仅供该模块本身引用的数据。

④ 程序代码。它指用来实现模块功能的程序。

前两个要素是模块的外部特性，反映了模块的外貌，后两个要素是模块的内部特性。在结构化设计中，主要考虑的是模块的外部特性，对其内部特性只做必要了解，具体的实现将在系统实施阶段完成。

2）模块化

模块化是把系统分割成能完成独立功能的模块，明确规定各模块的输入输出规格，使模块的界面清楚，功能明确。每个模块可独立命名和编址。在计算机软件领域，模块化的概念已被推崇 40 多年。目前，几乎所有的软件体系结构都体现了模块化的思想，即把软件划分为多个模块，每个模块完成一个子功能。当把所有模块组装到一起成为一个整体时，便可以完成指定的功能。

采用模块化原理的优点如下所示。

① 减少复杂性。模块化可以使软件结构清晰，容易设计、阅读和理解、测试和调试。

② 提高软件的可靠性和可维护性，因为变动往往只涉及少数几个模块。

③ 有助于软件开发工程的组织管理。一个复杂的大型程序可以由许多程序员分工编写不同的模块，并且可以进一步分配技术熟练的程序员编写困难的模块。

2. 模块独立性的度量

模块独立是模块化的具体表现。在一个管理信息系统中，系统的各组成部分之间总是存在着各种联系。将系统或子系统划分成若干模块，则在一个模块内部存在着块内联系，在模块之间存在着块间联系。模块的独立性与块内联系及块间联系的紧密程度密切相关，因此引入模块耦合和内聚的概念对模块的独立性进行度量。

1）耦合

耦合是对软件程序结构中各个模块之间相互依赖程度的一种度量，其强弱取决于模块间接口的复杂程度。一般由模块之间的调用方式、传递信息的类型和数量决定。在设计软件时应追求尽可能松散耦合的系统。因为对这类系统中任一模块的设计、测试和维护相对独立。

耦合可分为以下几种类型。

① 非直接耦合。如果两模块中任一个都不依赖对方而能独立工作，也就是说两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用实现的，则称这两个模块为非直接耦合。这种耦合的模块独立性最强。但是，在一个软件系统中不可能所有模块之间都没有任何联系。

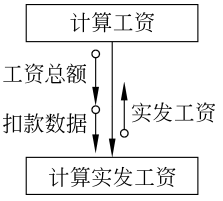


图 5-1 模块间的数据耦合

② 数据耦合。如果两模块间通过参数交换信息，而传递的信息仅限于数据，则称这两模块间为数据耦合，如图 5-1 所示。数据耦合是松散的耦合，模块的独立性也比较强。在软件程序结构中至少要有这类耦合，而且数据耦合也是模块间进行数据传输的一种不可缺少的形式。模块之间传输的数据元素越少，产生模块间相互影响的不利因素就越少。因此，要尽量减少两个模块之间不必要的数据传输，从而使模块之间的接口尽量简单。

③ 标记耦合。如果两个模块通过传递数据结构（不是简单数据，而是记录、数组等）加以联系，或者都与一个数据结构有关系，则称这两个模块间存在标记耦合。标记耦合要求这些模块都必须清楚该数据结构并按结构要求对它进行操作。在设计中应尽量避免这种耦合，因为它使在数据结构上的操作复杂化了。如果把在数据结构上的操作全部

集中在一个模块中，或者给不同的模块只传递与本模块相关的数据，则可以消除这种耦合。这种耦合介于数据耦合与控制耦合之间。

④ 控制耦合。如果一个模块把控制信息传递到另一个模块，对其功能进行控制，这种耦合就是控制耦合。例如，一个模块通过传送开关、标志、名称等控制信息明显地控制另一模块的内部选择功能。控制耦合意味着控制模块必须知道所控制模块内部的逻辑关系，而且对所控制模块的任何修改都会影响控制模块，这些都会降低模块的独立性。一般软件系统中都存在控制耦合，它是完成某些功能所必需的，但控制耦合通常会增加系统的复杂性，有时适当分解模块可以消除控制耦合。

⑤ 外部耦合。当若干模块均与同一个外部环境关联并受到约束时，它们之间便存在外部耦合。例如，I/O 处理使得所有 I/O 模块与特定的设备、格式和通信协议相关联。这是一种耦合程度比较强的耦合，外部耦合尽管需要，但应限制在少数几个模块上。

⑥ 公共耦合。当若干模块通过全局的数据环境相互作用时（一组模块都访问同一个公共数据环境），则它们之间存在公共耦合。全局数据环境中可能含有全局变量、公用区、内存公共覆盖区、任何存储介质上的文件、物理设备等。这种耦合的耦合程度较高，若修改某个数据，将会影响所有模块，而且无法控制各个模块对公共数据的存取，严重影响软件模块的可靠性和适应性。对公共数据的使用也会明显降低程序的可读性。

⑦ 内容耦合。当出现下列情形之一时，模块之间的耦合就是内容耦合。

- 一个模块直接访问另一个模块的内部数据。
- 一个模块不通过正常入口转到另一模块内部。
- 两个模块有一部分程序代码重叠（只可能出现在汇编语言中）。
- 一个模块有多个入口。一个模块要直接使用另一模块内部的数据或控制信息，或一个模块直接转移到另一模块内部，等等。

在内容耦合的情形中，所访问模块的任何变更或者用不同的编译器对它再编译都会造成程序出错。这种耦合应坚决避免。好在大多数高级程序设计语言已经设计成不允许出现内容耦合。它一般出现在汇编语言程序中。这种耦合是模块独立性最弱的耦合。

不同类型耦合与模块独立性的关系如图 5-2 所示。

图 5-2　耦合类型与模块独立性的关系

一般说来，设计软件时应尽量使用数据耦合，减少控制耦合，限制外部环境耦合和公共数据耦合，杜绝内容耦合。以上 7 种耦合类型只是从耦合的机制上所做的分类，按耦合的松紧程度的排列只是相对的关系，但它给设计人员在设计程序结构时提供了一个决策准则。有时，并不需要完全确定多个模块之间到底属于什么类型的耦合，只要大体掌握这个顺序，设计时尽量避免高耦合即可。

2）内聚

内聚是信息隐蔽和局部化概念的自然扩展，它标志一个模块内部各成分彼此结合的

紧密程度。理想的模块只完成一个功能，模块设计的目标之一是尽可能高地内聚。内聚有以下几类。

① 偶然内聚。它是指一个模块内各成分为完成一组功能而组合在一起，模块内各部分之间没有关系或关系很松散。常见情形是：当程序员写完一个程序后发现有一组语句多处出现，为节省内存便将这些语句单独组成一个模块，如图 5-3 所示。

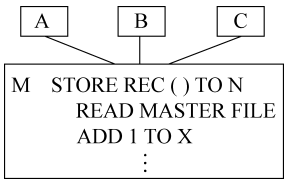


图 5-3 偶然内聚示例

这种模块的缺点首先是不易修改和维护。当修改此模块时，会涉及多个调用模块，有可能因相互制约而无法修改。其次，这种模块的内容不易理解，很难描述它所完成的功能，增加了程序的模糊性。因此，在通常情况下应避免构造这种模块。

② 逻辑内聚。如果一个模块完成的诸任务逻辑上相同或相关，则称之为逻辑内聚。这种模块把几种相关的功能组合在一起，每次调用时，由传送给模块的判定参数确定该模块应执行哪一种功能。这种模块是单入口多功能模块。例如错误处理模块，它接收出错信号，对不同类型的错误打印出不同的出错信息。逻辑内聚模块比偶然内聚模块的内聚程度要高，但是它所执行的不是一种功能，而是执行若干功能中的一种，因此它不易修改。另外，当调用时需要进行控制参数的传递，这就增加了模块间的控制耦合。而将未用的部分也调入内存将降低系统的效率。

③ 时间内聚。如果一个模块包含的诸任务必须在同一时间段内执行，则称之为时间内聚（又称经典内聚）。这种模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行，例如初始化模块、退出模块及紧急处理模块。而且，在一般情形下，时间内聚模块中各部分可以任意的顺序执行，因此它的内部逻辑更简单，存在的开关（或判定）转移更少。因而，时间内聚模块比逻辑内聚模块的内聚程度又高一些。

上述三种内聚形式通常被认为是低级内聚，其内聚程度较低。

④ 过程内聚。如果模块内各个组成部分的处理动作各不相同、彼此相关，并且受同一控制流支配，必须按特定的次序执行，则称之为过程内聚，例如统计并打印库存信息。过程内聚中各步骤连续发生，是一种较弱的内聚，并且隐含着时间内聚。这类模块的内聚程度比时间内聚模块的内聚程度更强一些，但是因为过程内聚模块的功能不唯一，所以它的内聚程度仍然较低，模块间的耦合程度还比较高。

⑤ 通信内聚。如果一个模块内各功能部分都使用了相同的输入数据或产生了相同的输出数据，则称之为通信内聚模块。例如产生工资报表及计算平均工资模块，模块中的两部分功能都使用工资记录。通信内聚模块的内聚程度比过程内聚模块的内聚程度要高，因为在通信内聚模块中包括了许多独立的功能。但是，这种模块的缺点是它容易产生重复的联结或重复的功能，因此维护起来不方便。

以上两种形式的内聚为中级内聚。

⑥ 顺序内聚。如果一个模块内的各个组成部分顺序执行几个处理动作，前一个处理动作产生的输出数据是下一个处理动作的输入数据，则称之为顺序内聚。顺序内聚的模

块中某一部分的执行依赖于另一部分，因此有高强度的内部紧凑性。但是，模块中可能包含了几个功能，也可能仅包含某个功能的一部分。因此，还不是最强的内聚。

⑦ 功能内聚。如果模块内所有成分形成一个整体，完成单个功能，则称为功能内聚。功能内聚模块中各个部分都是为完成某一具体功能所必不可少的组成部分，是紧密联系，不可分割的。如果把一个功能分成两个模块，就会导致模块之间有很强的耦合，而且它们不易单独理解和实现。功能内聚模块符合“黑盒”特性，即其他模块不必了解它的内部结构，就可以使用它，其内聚程度最高。功能内聚模块的优点是容易修改和维护。在把一个系统分解成模块的过程中，应当尽可能使模块达到功能内聚这一级，便于主程序的调用和控制。

以上两种形式的内聚为高级内聚。

各种内聚类型与模块独立性的关系如图 5-4 所示，其中偶然内聚是最低程度的内聚形式，功能内聚是最高程度的内聚形式。对于模块内聚类型的判定，可以通过如图 5-5 所示的判断树完成。对于任一给定的模块，都可以按照判断树所提的问题进行分析，当把问题回答出来以后，模块属于哪一种内聚也就清楚了。

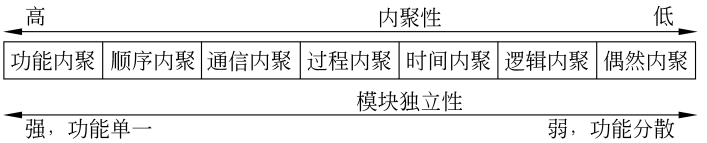


图 5-4　内聚类型与模块独立性的关系

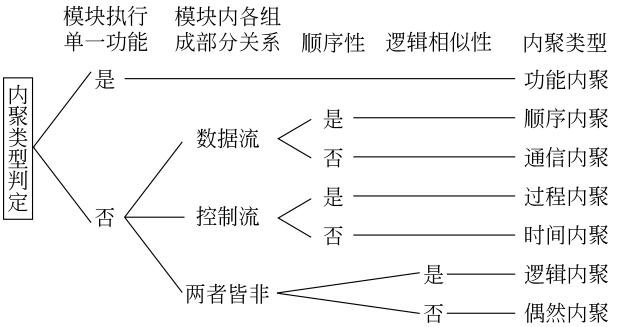


图 5-5　模块聚合强度判断树

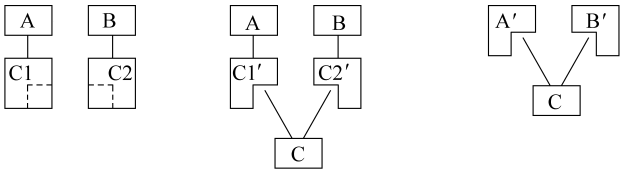
事实上，没有必要精确确定内聚的级别。重要的是设计软件时应当识别内聚度的高低并通过修改设计尽可能提高模块内聚度，从而获得较高的模块独立性。内聚和耦合是相互关联的。在程序结构中各模块的内聚程度越高，模块间的耦合程度就越低。但这也不是绝对的。在设计时要力求增加模块的内聚，尽量减少模块间的耦合；但增加内聚比减少耦合更重要，应当把更多的注意力集中到提高模块的内聚程度上。

3. 启发式规则

除上面介绍的基本原理和概念外，人们在开发计算机软件的长期实践中积累了丰富的经验，通过总结这些经验得出了一些启发式规则。这些启发式规则往往能帮助软件设计者找到改进软件结构，提高软件质量的途径。

1）通过模块分解或合并提高模块独立性

设计出软件的初始结构后，应该审查分析这个结构，通过模块分解或合并，力求降低耦合提高内聚。例如多个模块共有的一个子功能，可以将此功能独立成一个模块，由原模块调用；有时也可以通过分解或合并模块以减少控制信息的传递及对全程数据的引用，并且降低接口的复杂程度。如图 5-6(a)所示，C1、C2 有类似功能，也有不同功能，可把功能类似的部分分离出来，增加一个公共下属模块 C，如图 5-6(b)所示。如果余下的 C1′、C2′比较简单，也可以分别与其上级模块合并成 A′和 B′，以减少控制的传递和全局数据的引用，如图 5-6(c)所示。



(a) 具有共同子功能　　(b) 子功能分离　　(c) 余下部分与上级模块合并

图 5-6　模块的分解或合并

2）模块规模应该适中

模块的大小可以用模块中所含语句的数量的多少衡量。经验表明，一个模块的规模不应过大，最好能写在一页纸内(通常不超过 60 行语句)，控制在 50～100 行。语句行数过多会显著降低程序的可理解性。

过大的模块往往是由于分解不够充分，因此可以对功能进一步分解，生成一些下级模块或同层模块。但是进一步分解必须符合问题结构，一般说来，分解后不应该降低模块的独立性。

过小的模块开销大于有效操作，而且模块数目过多将使系统接口复杂。因此过小的模块有时不值得单独存在，特别是只有一个模块调用它时，通常可以把它合并到上级模块中去而不必单独存在。但是如果这个模块是功能内聚性模块，它为多个模块所共享，或者调用它的上级模块很复杂，则一定不要把它合并到其他模块中去。

3）深度、宽度、扇出和扇入都应适当

深度表示软件结构中控制的层数，它往往能粗略地标志一个系统的大小和复杂程度。深度和程序长度之间应该有粗略的对应关系，当然这个对应关系是在一定范围内变化的。如果层数过多，则应该考虑是否有许多管理模块可适当合并，如图 5-7 所示。

宽度是软件结构内同一个层数上的模块总数的最大值。一般来说，宽度越大，系统越复杂。对宽度影响最大的因素是模块的扇出。

扇出是一个模块直接控制(调用)的子模块数目。扇出过大意味着模块过分复杂，需要控制和协调过多的下级模块；扇出过小也不好。经验表明，一个设计良好的典型系统的平均扇

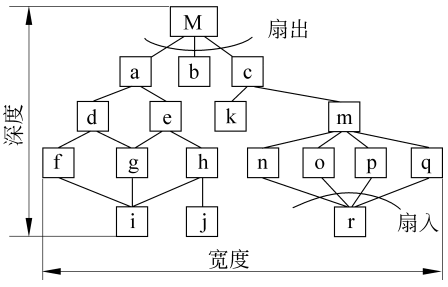


图 5-7　模块的深度、宽度及扇入和扇出