靠分解来画分层 DFD 图；在设计阶段用分解来实现模块化设计。在 OO 软件工程中，靠分解来划分类和对象。不管采用哪种设计方法，都要将系统分解成它的组成部分，成为模块、对象或构件。

1965 年，G. A. Miller 在他的著名文章“奇妙的数字 7±2——人类信息处理能力的限度”中指出，普通人分辨或记忆同类信息的不同品种或等级的数量，一般不超过 5～9 项（即 7±2）。这表明，要使人的智力足以管理好程序，坚持模块性（modularity）不仅是一个良好的主意，也是必不可少的措施。正如不分段的长篇文章可能使读者感到头痛一样，大型的单模块软件（monolithic software）不仅可读性差，可靠性也常常难以保证。

对问题求解的大量实验进一步表明，将一个复杂的问题分解为几个较小的问题，能够减小解题所需要的总工作量。用数学公式来表示，可以写成：

$$C(P_1+P_2)>C(P_1)+C(P_2)$$

$$E(P_1+P_2)>E(P_1)+E(P_2)$$

其中，P_1、P_2 系由问题 P_1+P_2 分解而得，C 为问题的复杂度，E 为解题需要的工作量。工作量通常用人-年（man-year）或人-月（man-month）来表示。

继续进行分解，问题的总复杂度和总工作量将继续减小，但如果无限地分下去，是否会使总工作量越来越小，最终变成可以忽略呢？不会。因为在一个软件系统内部，各组成模块之间是相互关联的。模块划分的数量越多，各模块之间的联系也就越多。模块本身的复杂度和工作量虽然随模块的变小而减小，模块的接口工作量却随着模块数的增加而增大。如图 7.1 所示，每个软件都存在一个最小成本区，把模块数控制在这一范围，可以使总的开发工作量保持最小。

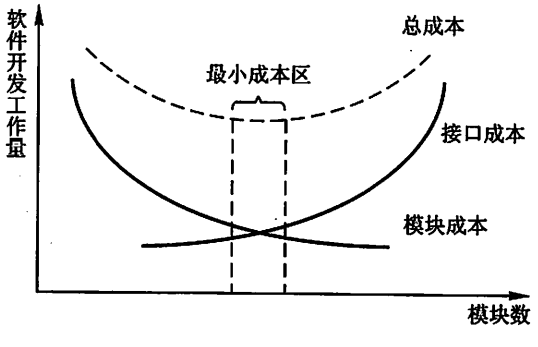


图 7.1　模块数与开发工作量的关系

2．模块独立性

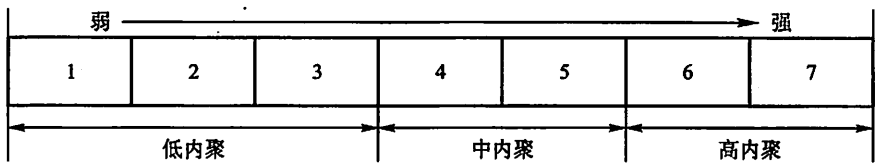
模块独立性（module independence）概括了把软件划分为模块时要遵守的准则，也是判断模块构造是否合理的标准。坚持模块的独立性，一般认为是获得良好设计的关键。

独立性可以从两个方面来度量，即模块本身的内聚（cohesion）和模块之间的耦合

(coupling)。前者指模块内部各个成分之间的联系，所以也称为块内联系或模块强度；后者指一个模块与其他模块间的联系，所以又称为块间联系。模块的独立性愈高，则块内联系越强，块间联系越弱。C. Myers 把内聚和耦合各划分为 7 类，现分别介绍如下。

（1）内聚

内聚是从功能的角度对模块内部聚合能力的量度。按照由弱到强的顺序，Myers 把它们分为 7 类，如图 7.2 所示，从左到右内聚强度逐步增强。



1. 偶然性内聚：coincidental cohesion
2. 逻辑性内聚：logical cohesion
3. 时间性内聚：temporal cohesion
4. 过程性内聚：procedural cohesion
5. 通信性内聚：communicational cohesion
6. 顺序性内聚：sequential cohesion
7. 功能性内聚：functional cohesion

图 7.2 内聚强度的划分

低内聚包括左侧 3 类模块，即：

① 偶然性模块。块内各组成成分在功能上是互不相关的。例如，如果有几个模块都需要执行“读 A”、“写 B”等相同的一组操作，为了避免重复书写，可以把这些操作汇成一个模块，供有关的模块调用。这类模块内部成分的组合纯属偶然，称为偶然性内聚。

② 逻辑性模块。通常由若干个逻辑功能相似的成分组成。例如一个用于计算全班学生平均分和最高分的模块。如图 7.3 所示，无论计算哪种分数，都要经过读入全班学生分数、进行计算、输出计算结果等步骤。实际上除中间的一步需按不同的方法计算外，前、后这两步都是相同的。把这两种在逻辑上相似的功能放在一个模块中，就可省去程序中的重复部分。其主要缺点是，执行中要从模块外引入用作判断的开关量，这会增大块间耦合。

③ 时间性模块。这类模块所包含的成分，是由相同的执行时间而联结在一起的。例如一个初始化模块可能包含“为变量赋初值”、“打开某个文件”等为正式处理作准备的功能。由于要求它们在同一时间内执行，故称为时间性内聚。

中内聚包括图 7.2 中的第 4、5 两类模块：

① 过程性模块。当一个模块中包含的一组任务必须按照某一特定的次序执行时，就称为过程性模块。图 7.4 显示了用高斯消去法解线性方程组的流程。如果把全部任务均纳入一个模块，便得到一个过程性模块。

② 通信性模块。图 7.5 显示了通信性模块的几个例子。这类模块的标志是，模块内部的

各个成分都使用同一种输入数据，或者产生同一个输出数据。它们靠公用数据而联系在一起，故称为通信性内聚。

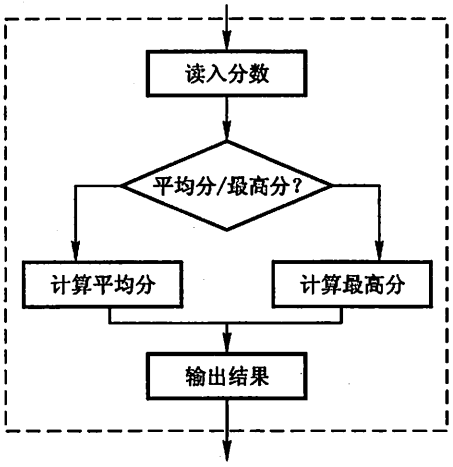


图 7.3　逻辑性模块示例

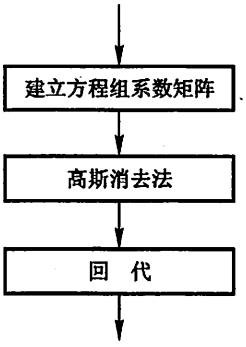


图 7.4　高斯消去法解线性方程组流程

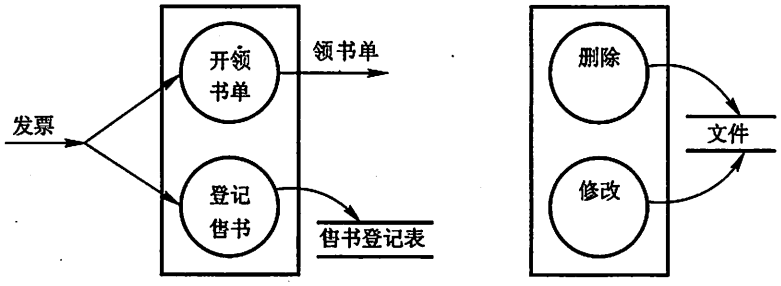


图 7.5　通信性模块示例

图 7.2 中最右端的两类属于高内聚模块。其中包括：

① 顺序性模块。顾名思义，这类模块中的各组成部分是顺序执行的。在通常情况下，上一个处理框的输出就是下一个处理框的输入。例如，把图 7.4 中的前两个任务组合在一起，就可得到一个顺序性模块。

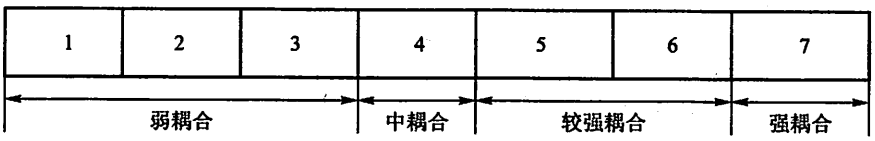
② 功能性模块。这是块内联系最强的一类模块。在这类模块中，所有的成分结合在一起，用于完成一个单一的功能。例如对一个数开平方，求一组数中的最大值，从键盘上读入一行等。在图 7.4 中，如果将每一处理框编制成一个模块，则产生的 3 个模块都是功能性模块。

显然，功能性模块具有内聚高、与其他模块的联系少等优点。“一个模块，一个功能”，

已成为模块化设计的一条准则，也是设计人员争取的目标。当然，其他的高内聚和中内聚模块也是允许使用的，低内聚模块因块内各成分的联系松散，可维护性和可重用性都比较差，在设计中应尽可能避免使用。

（2）耦合

耦合是对软件内部块间联系的度量。按照 Myers 的划分，也归纳为 7 类，如图 7.6 所示。



1. 非直接耦合：no direct coupling
2. 数据耦合：data coupling
3. 特征耦合：stamp coupling
4. 控制耦合：control coupling
5. 外部耦合：external coupling
6. 公共耦合：common coupling
7. 内容耦合：content coupling

图 7.6 耦合强度的等级

弱耦合包括图 7.6 左侧的 3 类情况。在图 7.7 所给出的示例中，模块 1 与模块 2 为同级模块，相互之间没有信息传递，属于非直接耦合。模块 3、4 都是模块 1 的下属模块。模块 1 调用它们时，可通过参数表与它们交换数据。如果交换的都是简单变量，便构成数据耦合（如模块 1、3 之间）；如果交换的是数据结构，便构成特征耦合（如模块 1、4 之间）。

例如在图 7.8 中，“计算应扣款”模块把“用水量”和“用电量”分别传递给“计算水费”与“计算电费”两个模块，然后从这两个模块分别获取“水费”和“电费”，上、下层模块间存在的耦合便是数据耦合。如果事先定义下列的数据结构：

房租水电=房租+用水量+用电量

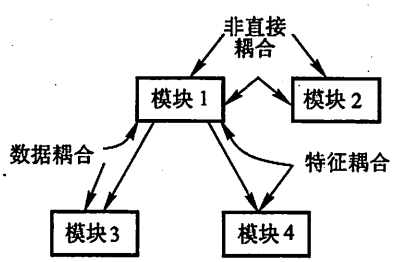


图 7.7 弱耦合示例

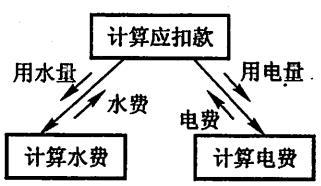


图 7.8 数据耦合示例

并把这一数据结构直接传递给“计算水费”与“计算电费”两个模块，在上、下层之间就构成特征耦合了，如图 7.9 所示。在这种情况下，不仅在模块间传递的数据量要增加，而且当

“房租水电”的数据结构或其格式发生变化时，图 7.9 中的 3 个模块都要作相应的更改。由此可见，特征耦合会使本来无关的模块（例如“计算水费”和“计算电费”）变为有关，耦合强度显然比数据耦合要高。

控制耦合是中等强度的耦合。此时在模块间传递的信息不是一般的数据，而是用作控制信号的开关值或标志量（flag）。以图 7.3 的逻辑性模块为例，当调用这一模块时，调用模块必须先把一个控制信号（平均分/最高分）传递给它，以便选择所需的操作。因此，控制模块必须知道被控模块的内部逻辑，从而增强了模块间的相互依赖。

较强耦合包括图 7.6 中第 5、6 这两类，若允许一组模块访问同一个全局变量，可称它们为外部耦合；若允许一组模块访问同一个全局性的数据结构，则称之为公共耦合。在图 7.10 中，假定模块 C、D、N 均可访问公用区中的某一数据结构。模块 C 首先从公共区中读出数据，然后启动模块 D，对该数据进行计算和更新。如果模块 D 有错误，计算与更新的数据也随之出错，则当模块 N 在以后某个时候读取上述数据并作处理时，就会得出错误的结果。虽然问题是在调用 N 时暴露的，但根源在于 D，这就增加了调试和排错的困难。

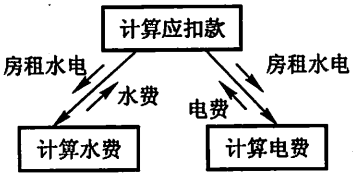


图 7.9　特征耦合示例

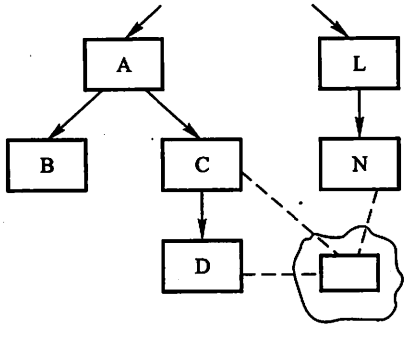


图 7.10　公共耦合示例

最后一类，也是最强的一类耦合是内容耦合。如果一个模块可以直接调用另一模块中的数据，或者允许一个模块直接转移到另一模块中去，就称它们间的耦合为内容耦合。如图 7.11 所示，假定在修改模块 N 时将一条指令从标号为 A 的指令前面移到它的后面，就需特别细心，检查会不会因此影响到模块 M 的执行结果。

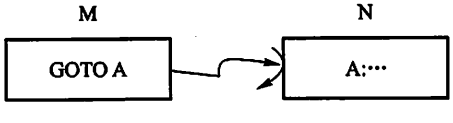


图 7.11　内容耦合示例

耦合越弱，则表明模块的独立性越强。但实际工作中，中等甚至较强的耦合不可能也不必完全禁用。例如 FORTRAN 语言中的 COMMON 区可以令访问它的模块间发生公共耦合，

当然不应滥用。但当在一组模块之间存在较多的公用数据时，使用 COMMON 区往往又使编程比较方便。所以问题不在于禁止使用它们，而是要了解各种耦合的特点与不足，以便在需要使用它们时能预见到可能产生的问题。至于最强类的内容耦合，由于会给维护工作带来很大的困难，有人称之为“病态联系”，故应该尽量不用。

7.2 面向对象设计建模

在基于面向对象分析阶段确定了问题领域的类/对象以及它们的关系和行为的基础上，就可以开始面向对象设计了。它主要考虑“如何实现”的问题，其注意的焦点从问题空间转移到解空间，着重完成各个不同层次的模块设计。因此，它不仅要说明为实现需求必须引入的类、对象以及它们之间是如何关联的，描述对象间如何传递消息和对象的行为如何实现，还需要从提高软件设计质量和效率等方面考虑如何改进类结构和可复用类库中的类。

7.2.1 面向对象设计模型

Pressman 把 OO 设计模型定义成一个金字塔层次结构。图 7.12 显示了怎样从 OO 分析模型导出 OO 设计模型，以及两种模型的相互关系。

如图 7.12 所示，右边的金字塔是一个 OO 设计模型，由系统架构层、类和对象层、消息层、责任层等 4 个层次组成。系统架构层描述了整个系统的总体结构，使所设计的软件能够满足客户定义的需求，并实现支持客户需求的技术基础设施；类和对象层包含类层次关系，使得系统能够以通用的方式创建并不断逼近特殊需求，该层同时包含了每个对象的设计表示；消息层描述对象间的消息模型，它建立了系统的外部和内部接口，包含使得每个对象能够和其协作者通信的细节；责任层包含针对每个对象的所有属性和操作的数据结构和算法的设计。

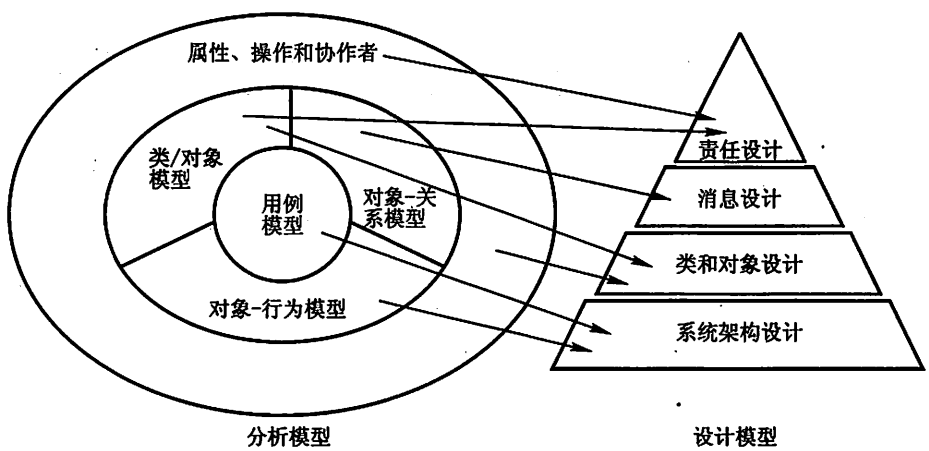


图 7.12 OO 分析模型转换到 OO 设计模型