第5章 chapter 5

信息系统的设计

完成系统的需求分析之后，就进入了系统的设计阶段。系统设计的任务是把需求分析阶段产生的系统需求说明转换为用适当手段表示的系统设计文档。按照结构化设计方法，从项目管理观点来看，通常将系统设计分为总体设计和详细设计两个阶段。总体设计用来确定系统的结构，即系统的组成以及各组成元素之间的相互关系；详细设计用来确定模块内部的算法和数据结构，产生描述各模块程序过程的详细设计文档。本章首先介绍系统设计的基本原理和优化规则，然后对总体设计、详细设计进行介绍。

5.1 系统设计的基本原理和优化规则

5.1.1 系统设计的基本原理

在系统设计过程中应该遵循的基本原理包括模块化设计原理、抽象原理、信息隐蔽和局部化原理、逐步求精原理以及模块独立性原理等。

1. 模块化设计原理

所谓模块是指具有相对独立性的，由数据说明、执行语句等程序对象构成的集合。程序中的每个模块都需要单独命名，通过名字可实现对指定模块的访问。在高级语言中，模块具体表现为函数、子程序以及过程等。一个模块具有输入输出接口、功能、内部数据和程序代码4个特征。输入输出接口用于实现本模块与其他模块间的数据传送，即向本模块传入所需的原始数据及从本模块传出得到的结果数据。功能是指模块所完成的工作，模块的输入输出接口和功能构成了模块的外部特征。内部数据是指仅能在模块内部使用的局部量。程序代码用于描述实现模块功能的具体方法和步骤。模块的内部数据和程序代码反映的是模块的内部特征。

模块化是指将整个程序划分为若干个模块，每个模块用于实现一个特定的功能。划分模块对于解决大型的、复杂的问题是非常必要的，因为这样可以大大降低解决问题的难度。为了说明这一点，可对问题复杂性、开发工作量和模块数之间的关系进行以下分析。

首先，设$C(x)$为问题x所对应的复杂度函数，$E(x)$为解决问题x所需要的工作量函

数。对于两个问题 P_1 和 P_2，如果

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

即问题 P_1 的复杂度比 P_2 高，则显然有

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

即解决问题 P_1 比 P_2 所需的工作量大。

在人们解决问题的过程中，发现存在另一个有趣的规律：

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

即解决由多个问题复合而成的大问题的复杂度大于单独解决各个问题的复杂度之和。也就是说，对于一个复杂问题，将其分解成多个小问题来分别解决比较容易。由此可以推出

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

即将复杂问题分解成若干个小问题，逐个解决，这种方法所需的工作量小于直接解决复杂问题所需的工作量。

根据上面的推导，可以得到这样一个结论：模块化可以降低解决问题的复杂度，从而减少系统开发的工作量。但是不是模块划分得越多越好呢？虽然增加程序中的模块数可以减少开发每个模块的工作量，但同时却增加了设计模块接口的工作量。图 5-1 表达了模块数与系统开发成本之间的关系，可以看出，当划分的模块数处于最小成本区域时，开发系统的总成本最低。虽然目前还不能精确地算出 M 的数值，但是在考虑程序模块化的时候，总成本曲线确实是有用的指南。后面介绍的模块独立性原理和优化规则，可以在一定程度上帮助开发者决定合适的模块数目。

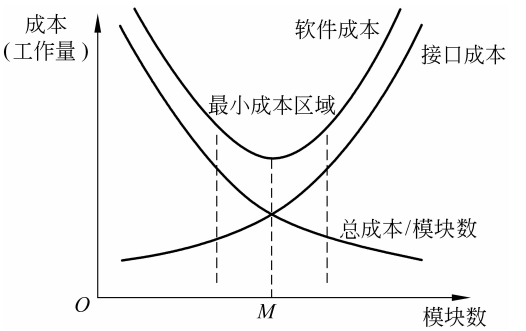

图 5-1　模块数与系统开发成本的关系

采用模块化设计原理可以带来以下好处：

(1) 模块化使系统结构清晰，易于设计，也容易阅读和理解。

(2) 程序错误通常局限在有关的模块及它们之间的接口中，所以模块化能使系统容易测试和调试，从而有助于提高系统的可靠性。

(3) 系统的变动往往只涉及少数几个模块，所以模块化还能够提高系统的可修改性。

(4) 模块化使得一个复杂的大型程序可以由许多程序员分工编写，并且可以进一步分配技术熟练的程序员编写困难的模块，有助于系统开发工程的组织管理。

(5) 模块化有利于提高程序代码的可重用性。

2. 抽象原理

抽象是人类在解决复杂问题时经常采用的一种思维方式，它是指将现实世界中具有共性的一类事物的相似的、本质的方面集中起来，加以概括，而暂时忽略它们之间的细节差异。在系统开发中运用抽象的概念，可以将复杂问题的求解过程分层，在不同的抽象层次上实现难度的分解。在抽象级别较高的层次上，可以将琐碎的细节信息暂时隐藏起来，以利于解决系统中全局性的问题。

结构化程序设计中自顶向下、逐步求精的模块划分思想正是人类思维中运用抽象方法解决复杂问题的体现。系统结构中顶层的模块抽象级别最高，是控制并协调系统的主要功能且影响全局；系统结构中位于底层的模块抽象级别最低，是具体实现数据的处理过程。采用自顶向下、由抽象到具体的思维方式，不但降低了系统开发中每个阶段的工作难度，简化了系统的设计和实现过程，而且还有助于提高系统的可读性、可测试性和可维护性。此外，在程序设计中运用抽象的方法还能够提高代码的可重用性。

3. 信息隐蔽和局部化原理

应用模块化设计原理时，自然会产生的一个问题：为了得到最好的一组模块，应该怎样分解系统呢？信息隐蔽原理指出：在设计和确定模块时，应该使得一个模块内包含的信息（过程和数据）对于不需要这些信息的模块来说是不能访问的。这一原理是由帕纳斯（D. L. Parnas）在 1972 年提出的，也就是说，有效的模块化可以通过一组独立的模块来实现，这些独立的模块彼此间仅仅交换那些为了完成系统功能而必须交换的信息。这一指导思想的目的是为了提高模块的独立性，即当修改或维护模块时减少模块的错误扩散到其他模块中去的机会。因此，信息隐蔽简化了系统结构，提供了程序模块设计标准化的可能性。

局部化的概念和信息隐蔽的概念密切相关。局部化是指把一些关系密切的系统元素物理地放得比较近，严格控制数据对象可以访问的范围。在模块中使用局部数据元素就是局部化的一个例子。显然，局部化有助于实现信息隐蔽。

4. 逐步求精原理

逐步求精是人类解决复杂问题时采用的基本方法，也是许多软件工程技术（例如规格说明技术、设计和实现技术）的基础。可以把逐步求精定义为：为了能集中精力解决主要问题而尽量推迟对问题细节的考虑。

逐步求精之所以如此重要，是因为人类的认知过程遵守米勒（Miller）法则：一个人在任何时候都只能把注意力集中在 7±2 个知识块上。但是，在开发系统的过程中，软件工程师在一段时间内需要考虑的知识块数远远大于 7。例如，一个程序通常不止使用 7 个数据，一个用户也往往有不止 7 个方面的需求。逐步求精方法的强大作用就在于它能够帮助软件工程师把精力集中在与当前开发阶段最相关的方面上，而忽略对整体解决方案来说虽然是必要的，但目前还不需要考虑的细节，这些细节将留到以后再考虑。米勒法则是人类智力的基本局限，我们不可能战胜自己的自然本性，只能接受这个事实，承认

自身的局限性,并在这个前提下尽我们的最大努力工作。

5. 模块独立性原理

模块独立性概括了把系统划分为模块时需要遵守的准则,同时也是判断模块构造是否合理的标准。模块独立性是指每个模块只完成系统要求的独立的子功能,与其他模块的联系最少,并且接口简单。

为什么模块独立性会这么重要呢?这是因为模块化程度较高的系统,其功能易于划分,接口简单,因此其开发比较容易,特别是在几个开发人员共同开发一个系统时,这一点尤为突出。同时,这样的系统也比较容易测试和维护,修改所引起的副作用也小,而且,模块从系统中取出或插入也比较简单。

模块独立性可以从两个方面来衡量:模块本身的内聚和模块之间的耦合。前者反映的是模块内部各个成分之间的联系,所以也称块内联系;后者反映的是一个模块与其他模块之间的联系,所以又称块间联系。模块的独立性越高,则块内联系越强,块间联系越弱,因此必须尽可能设计出高内聚、低耦合的模块。

1) 内聚

模块的内聚是指模块内部各成分间联系的紧密程度。一个模块内部各成分之间的联系越紧密,该模块独立性就越高。按照内聚程度由低到高的顺序,把模块的内聚分为7种类型,如图5-2所示。

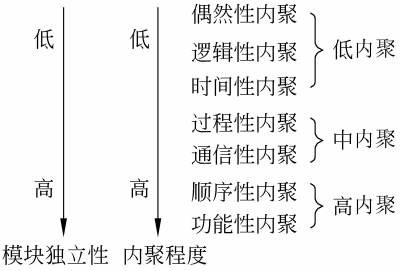

图5-2 内聚的划分

(1) 偶然性内聚。如果几个模块内有一段代码是相同的,那么将它们抽取出来形成单独的模块,即偶然性内聚模块。例如,几个模块都要执行"读A"的操作,为避免重复书写而将这些操作汇成一个模块。偶然性内聚是内聚程度最低的一种,具有偶然性内聚的模块独立性差,不易理解和修改,会给系统开发带来很大的困扰,出错的概率要比其他类型的模块大得多,因此在系统设计时应尽量避免使用。

(2) 逻辑性内聚。逻辑性内聚是指模块内执行几个逻辑上相似的功能,通过参数确定模块完成的功能。例如,将产生各种类型错误的信息输出放在一个模块中,或将从不同设备上的输入放在一个模块中,形成一种单入口、多功能的模块。这种模块内聚程度有所提高,各部分之间在功能上也有相互关系,但不易修改,因为当某个调用模块要求修改模块公用代码时,而另一些调用模块又不要求修改。另外,调用时需要进行控制参数的传递,造成模块间的控制耦合,而且在调用此模块时,不用的部分也占据了主存,从而降低了系统效率。

(3) 时间性内聚。若模块中包含了需要在同一时间段中执行的多个任务,则称该模块的内聚为时间性内聚。例如,将多个变量的初始化放在同一个模块中实现,或将需要同时使用的多个库文件的打开操作放在同一个模块中,都会产生时间性内聚的模块。由于时间性内聚模块中的各个部分在时间上的联系,其内聚程度比逻辑性内聚高一些。但

这样的模块往往会和其他相关模块有着紧密的联系，因而会造成耦合程度的提高。

（4）过程性内聚。当模块中包含的任务必须按照某一特定的次序执行时，就称之为过程性内聚模块。例如，用高斯消去法解线性方程组的流程为：建立方程组系数矩阵→高斯消去→回代，将其纳入一个模块中就形成了一个过程性内聚模块。

（5）通信性内聚。如果模块内各功能部分使用了相同的输入数据，或产生了相同的输出数据，则称之为通信性内聚模块。例如，模块完成“建表”“查表”两部分功能，都使用同一数据结构——名字表；又如，模块完成生产日报表、周报表和月报表，都使用同一数据——日产量。通信性内聚的模块各部分都紧密关联于同一数据（或者数据结构），所以内聚程度要高于前几种类型。同时，如果把某一数据结构、文件及设备等操作都放在一个模块内，那么可达到信息隐藏的作用。

（6）顺序性内聚。若模块中的各个部分都与同一个功能密切相关，并且必须按照先后顺序执行（通常前一部分的输出数据就是后一部分的输入数据），则称该模块的内聚为顺序性内聚。例如，在处理学生成绩的模块中，前一部分根据成绩统计及格的学生人数，后一部分根据及格人数计算学生的及格率。由于顺序性内聚模块中的各个部分在功能和执行顺序上都密切相关，因此内聚程度很高且易于理解。

（7）功能性内聚。模块中各个部分都是为了完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作、紧密联系、不可分割的，则称该模块为功能性内聚模块。例如，求一组数的最大值这样一个单一功能的模块就是功能性内聚模块。功能性内聚是所有内聚中内聚程度最高的一种，功能性内聚的模块易理解，易修改，有利于实现模块的重用，从而提高了系统开发的效率。

综上所述，对于内聚应该采取这样的设计原则：禁用偶然性内聚和逻辑性内聚，限制使用时间性内聚，少用过程性内聚和通信性内聚，提倡使用顺序性内聚和功能性内聚。

2）耦合

模块的耦合是指模块之间相互联系的程度。相互联系复杂的模块耦合程度高，模块独立性低；相互联系简单的模块耦合程度低，模块独立性高。按照耦合程度由低到高的顺序，模块的耦合也分为 7 种类型，如图 5-3 所示。

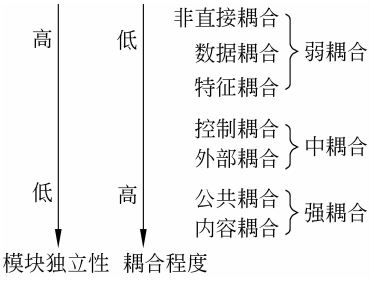

图 5-3 耦合的划分

（1）非直接耦合。

非直接耦合是指两个模块之间没有直接关系，相互之间没有信息传递，它们之间的联系完全是通过主模块的控制和调用来实现的。因此，模块间的这种耦合程度最低，但模块独立性最高。

（2）数据耦合。

数据耦合是指两个模块之间仅通过模块参数交换信息，且交换的信息全部为简单数据信息，相当于高级语言中的值传递。数据耦合的耦合程度较低，模块的独立性较高。通常软件系统中都包含数据耦合。

（3）特征耦合。

特征耦合是指两个模块之间传递的是数据结构。其实模块间传递的是这个数据结

构的地址,两个模块必须清楚这些数据结构,并按要求对其进行操作,这样就降低了可理解性。可采用信息隐蔽的方法,把该数据结构以及在其上的操作全部集中在一个模块中,就可消除这种耦合,但有时因为还有其他功能的缘故,特征耦合往往是不可避免的。

(4) 控制耦合。

控制耦合是指一个模块调用另一个模块时传递的是控制变量(如开关、标志等),被调模块通过该控制变量的值有选择地执行块内某一功能。因此被调模块内应具有多个功能,哪个功能起作用要受其调用模块的控制。控制耦合增加了理解、编程及修改的复杂性,调用模块必须知道被调模块内部的逻辑关系,即被调模块处理细节不能实现信息隐藏,从而降低了模块的独立性。

(5) 外部耦合。

外部耦合是指一组模块访问同一个全局变量。

(6) 公共耦合。

公共耦合是指一组模块访问同一个全局性数据结构。如果在模块之间共享的数据很多,而且通过参数来传递很不方便时,才使用公共耦合,因为公共耦合会引起以下问题:

① 耦合的复杂程度随模块个数的增加而增加,无法控制各个模块对公共数据的存取。若某个模块有错,可通过公共区将错误延伸到其他模块,会影响系统的可靠性。

② 使系统的可维护性变差。若某一模块修改了公共区的数据,则会影响与此有关的所有模块。

③ 降低了系统的可理解性。因为各个模块使用公共区的数据的方式往往是隐含的,所以数据被哪些模块共享就不容易很快搞清。

(7) 内容耦合。

如果发生下列情形,两个模块之间就会发生内容耦合。

① 一个模块可以直接访问另一个模块的内部数据。

② 一个模块不通过正常入口转到另一模块内部。

③ 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)。

④ 一个模块有多个入口。

内容耦合是最高程度的耦合,也是最差的耦合。模块间过于紧密的联系会给后期的开发和维护工作带来很大的麻烦。

综上所述,对于耦合应该采取这样的设计原则:尽量使用非直接耦合、数据耦合和特征耦合,少用控制耦合和外部耦合,限制公共耦合,完全不用内容耦合。

内聚和耦合是密切相关的,模块内的高内聚往往意味着模块间的低耦合。内聚和耦合都是进行模块化设计的有力方法。实践表明,内聚更重要,所以应该把更多的注意力集中到提高模块的内聚程度上。

5.1.2 系统设计的优化规则

软件工程师在开发计算机系统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优化规则。这些优化规则虽然不像前面讲述的基本原理那样普遍适用,

但是在许多场合仍然能给信息系统开发者有益的启示，往往能帮助他们找到改进系统设计、提高系统质量的途径，因此有助于实现有效的模块化。下面介绍几条常用的优化规则。

(1) 改进系统结构，提高模块独立性。

设计出系统的初步结构以后，应该审查、分析这个结构，通过模块分解或合并，力求降低耦合程度，提高内聚程度，保持模块的相对独立性，优化初始的系统结构。

(2) 模块的作用域应处于其控制域范围之内。

模块的作用域是指受该模块内一个判定条件影响的所有模块范围。模块的控制域是指该模块本身以及所有该模块的下属模块(包括该模块可以直接调用的下层模块和可以间接调用的更下层的模块)。例如，如图 5-4 所示，模块 C 的控制域为模块 C、E 和 F；若在模块 C 中存在一个对模块 D、E 和 F 均有影响的判定条件，即模块 C 的作用域为模块 C、D、E 和 F(图中带阴影的模块)，显然模块 C 的作用域超出了其控制域。由于模块 D 在模块 C 的作用域中，因此模块 C 对模块 D 的控制信息必然要通过上层模块 B 进行传递，这样不但会提高模块间的耦合程度，而且会给模块的维护和修改带来麻烦(若要修改模块 C，可能会对不在它的控制域中的模块 D 造成影响)。因此，进行系统设计时，应使各个模块的作用域处于其控制域范围之内。若发现不符合此优化原则的模块，可通过下面的方法进行改进：

① 将判定位置上移。例如，将图 5-4 中的模块 C 中的判定条件上移到上层模块 B 中，或将模块 C 整个合并到模块 B 中。

② 将超出作用域的模块下移。例如，将图 5-4 中的模块 D 移至模块 C 的下一层，使模块 D 处于模块 C 的控制域中。

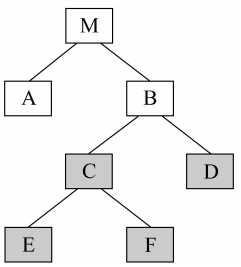

图 5-4 模块的作用域和控制域

(3) 系统结构图的深度和宽度不宜过大。

所谓深度是指系统结构图从上至下的层数，它能够粗略地反映出软件系统的规模和复杂程度；所谓宽度是指系统结构图内同一层次上模块数的最大值，通常宽度越大的系统越复杂。例如，在如图 5-5 所示的系统结构图中，深度为 5，宽度为 8。深度在程序中表现为模块的嵌套调用，嵌套的层数越多，程序就越复杂，程序的可理解性也就随之下降。深度过大的问题可通过将系统结构图中过于简单的模块分层与上一级模块合并来解决；而宽度过大的问题则可通过增加中间层来解决。显然，系统结构图的深度和宽度是相互对立的两个方面，减小深度会引起宽度的增加，而减小宽度又会带来深度的增加。

(4) 模块应具有高扇入和适当的扇出。

对一个模块来说，扇入是指有多少上层模块直接调用它，例如，图 5-6(a)中模块 M 的扇入为 n；扇出是指一个模块可以直接调用的下层模块数，例如，图 5-6(b)中模块 M 的扇出为 k。模块的扇入越大，则说明共享该模块的上层模块越多，或者说该模块在程序中的重用性越高；而对于扇出，根据实践经验，在设计良好的典型系统中，模块的平均扇出通常为 3 或者 4。

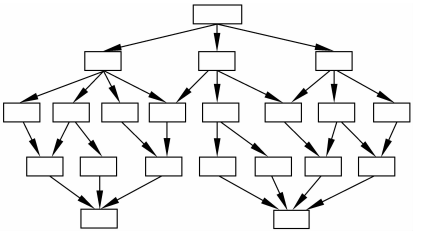

图 5-5 系统结构图示例

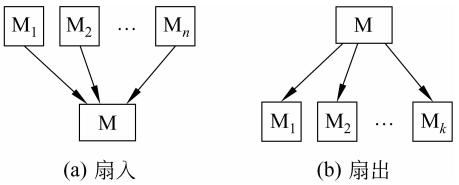

图 5-6 模块的扇入和扇出

在一个好的系统结构中,模块应具有较高的扇入和适当的扇出,但绝不能为了单纯追求高扇入或适当的扇出而破坏了模块的独立性。此外,经过对大量软件系统的研究后发现,在设计良好的系统结构中,通常顶层的扇出较大,中间层的扇出较小,底层的扇入较大。

(5) 保持适中的模块规模和复杂度。

程序中模块的规模过大,会降低程序的可读性;而模块规模过小,势必会导致程序中的模块数过多,增加接口的复杂性。对于模块的适当规模并没有严格的规定,但普遍的观点是模块中的语句数最好保持在 50～150 条,可以用一两页打印纸打印,便于人们的阅读与研究。为了使模块的规模适中,在保证模块独立性的前提下,可对程序中规模过小的模块进行合并或对规模过大的模块进行分解。

模块复杂度的限制是基于 McCabe 复杂度度量方法提出的,该方法是计算由程序流程图得到的程序图中的环的个数 $V(G)$,实践表明 $V(G)=10$ 是模块规模和复杂度的合理上限。

(6) 降低模块接口的复杂度。

复杂的模块接口是导致系统出现错误的主要原因之一,因此在系统设计中应尽量使模块接口简单、清晰。降低模块的接口复杂度,这样可以提高系统的可读性,减少出现错误的可能性,并有利于系统的测试和维护。例如,求一元二次方程的根模块 quad_root(tbl,x),其中 tbl 和 x 分别是系数组参数和根数组参数,就不如将接口的参数简单化,使模块变为 quad_root(a,b,c,x1,x2),这样不仅易于理解,也不容易发生传递错误。

(7) 设计单入口、单出口的模块。

这条规则告诫软件工程师不要使模块间出现内容耦合,设计出的每一个模块都应该只有一个入口和一个出口,不要随便使用 goto 语句。当控制流从顶部进入模块并且从底部退出模块时,系统是比较容易理解的,因此也是比较容易维护的。

(8) 模块功能可以预测。

要求设计出的模块的功能能够预测,但也要防止模块功能过分局限。

5.2 总体设计概述

5.2.1 总体设计的目的和任务

总体设计的基本目的就是回答“概括地说,系统应该如何实现”这个问题,因此,总体

设计又称为概要设计或初步设计。通过这个阶段的工作将划分出组成系统的物理元素——程序、文件、数据库、人工过程和文档等，但是每个物理元素仍然处于黑盒子级，这些黑盒子里的具体内容将在以后仔细设计。总体设计阶段还有一项工作就是设计系统的结构，也就是要确定系统中每个程序是由哪些模块组成的，以及这些模块相互间的关系。

总体设计的基本任务包括如下几点。

1. 设计软件系统结构

为了实现目标系统，最终必须设计出组成这个系统的所有程序和数据库（文件），对于程序，则首先进行结构设计，具体方法如下：

（1）采用某种设计方法，将一个复杂的系统按功能划分成模块。

（2）确定每个模块的功能。

（3）确定模块之间的调用关系。

（4）确定模块之间的接口，即模块之间传递的信息。

（5）评价模块结构的质量。

从以上内容看，系统结构的设计是以模块为基础的。在需求分析阶段，通过某种分析方法把系统分解成层次结构。在设计阶段，以需求分析的结果为依据，从实现的角度划分模块，并组成模块的层次结构。系统结构的设计是总体设计关键的一步，直接影响详细设计与编码的工作。系统的质量及一些整体特性都在软件结构的设计中决定，因此，应由经验丰富的软件人员担任，采用一定的设计方法，选取合理的设计方案。

2. 数据结构及数据库设计

在系统结构设计中，应对需求分析阶段所生成的数据字典加以细化，从计算机技术实现的角度出发，要确定系统涉及的文件系统及各种数据的结构，主要包括确定输入输出文件的数据结构及确定算法所需的逻辑数据结构等。在需求分析阶段仅为系统所需的数据库建立了概念数据模型（最常用的是 E-R 模型），系统结构设计阶段需要将原本独立于数据库实现的概念模型与具体的数据库管理系统的特征结合起来，建立数据库的逻辑结构，主要包括确定数据库的模式、子模式以及对数据库进行规范和优化等。

3. 编写总体设计文档

总体设计阶段应交付的文档通常包括总体设计说明书、数据库设计说明书、用户手册及系统初步测试计划。

（1）总体设计说明书：给出系统结构设计的结果，为系统的详细设计提供基础。

（2）数据库设计说明书：主要给出使用的数据库管理系统的简介、数据库的概念模型、逻辑设计和结果。

（3）用户手册：对需求分析阶段编写的用户手册进行补充。

(4) 系统初步测试计划：明确测试中应采用的策略、方案、预期的测试结果及测试的进度安排。

4. 评审

在该阶段，对设计部分是否完整地实现了需求中规定的功能、性能等要求，设计方案的可行性，关键的处理，内外部接口定义正确性、有效性以及各部分之间的一致性，等等，都要一一进行评审。

5.2.2 总体设计说明书

总体设计说明书是总体设计阶段结束时提交的技术文档，它的主要内容如下：

(1) 引言：包括编写目的、背景、定义和参考资料。

(2) 总体设计：包括需求规定、运行环境、基本设计概念、处理流程和结构。

(3) 接口设计：包括用户接口、外部接口和内部接口。

(4) 运行设计：包括运行模块的组合、运行控制和运行时间。

(5) 系统数据结构设计：包括逻辑结构设计、物理结构设计、数据结构与程序的关系。

(6) 系统出错处理设计：包括出错信息、补救措施和系统恢复设计。

5.2.3 总体设计的图形工具

用于总体设计的图形工具有 HIPO 图和结构图，它们主要用来描述系统模块的层次结构。

1. HIPO 图

HIPO(Hierarchy Plus Input/Processing/Output)图是 IBM 公司在 20 世纪 70 年代开发出来的用于描述系统结构的图形工具。它实质上是在描述系统总体模块结构的层次图(hierarchy，简称 H 图)的基础上加入了用于描述每个模块输入输出数据和处理功能的 IPO 图，因此它的中文全名为层次图加输入/处理/输出图。

1) HIPO 图中的 H 图

在 HIPO 图中，为了使 H 图更具有可追踪性，可以为除顶层矩形框以外的其他矩形框加上能反映层次关系的编号。例如，工资计算系统的 H 图如图 5-7 所示。

2) HIPO 图中的 IPO 图

例如，工资计算系统中的计算工资模块的 IPO 图如图 5-8 所示。

2. 结构图

尤顿(Yourdon)提出的结构图(Structure Chart，SC)是进行系统结构设计的另一个有力工具。结构图能够描述出软件系统的模块层次结构，清楚地反映出程序中各模块之间的调用关系和联系。结构图中的基本符号及其含义如表 5-1 所示。