# 第5章　软件设计

## 5.1　软件设计基础

在软件需求确定后，就进入软件设计阶段。软件设计是软件工程的重要阶段。软件设计的基本目的就是回答“系统应该如何实现”这个问题。软件设计的任务，就是把软件需求规格说明书中规定的功能要素，考虑实际条件，转换为满足软件系统需求的技术方案，为下个阶段的软件实施工作奠定基础。

### 5.1.1　软件设计概述

软件设计是软件开发过程中决定软件产品质量的关键阶段。在软件设计阶段所做出的决策，将最终决定软件开发能否成功，更重要的是，这些设计决策将决定软件维护的难易程度。

软件设计活动是获取高质量、低耗费、易维护软件最重要的一个环节。其主要目的是绘制软件的蓝图，权衡和比较各种技术和实施方法的利弊，合理分配各种资源，构建软件系统的详细方案和相关模型，指导软件实施工作顺利开展。

软件设计是开发时期的起始阶段，关系到整个软件开发时期的质量。软件开发时期信息流描述了软件设计从软件需求到软件编码，起到承上启下的作用，如图 5-1 所示。

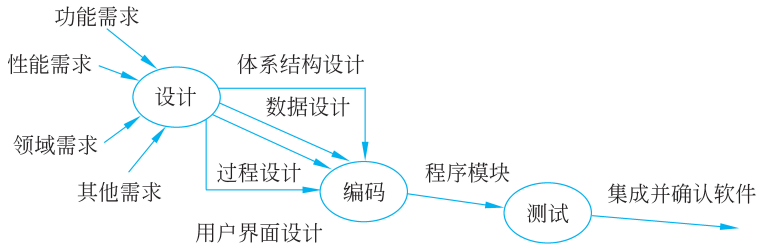


图 5-1　软件开发时期的信息流

#### 1. 软件设计的任务

软件设计的任务是从软件需求说明书出发，根据需求分析阶段确定的功能设计软件系统的整体结构、划分功能模块、确定每个模块的实现算法，形成软件的具体设计方案。软件设计是一种在设计者计划中通过诸如软件如何满足客户的需要，如何才能容易地实现和如何才能方便地扩展功能以适应新的需求等不同角度考虑的创造性活动。

从软件工程的角度，一般将软件设计分为概要设计和详细设计两个阶段，如图 5-2 所示。根据软件项目的规模和复杂度，概要设计和详细设计既可以合并为软件设计阶段，也

可以反复迭代，直至完全实现软件需求内容。

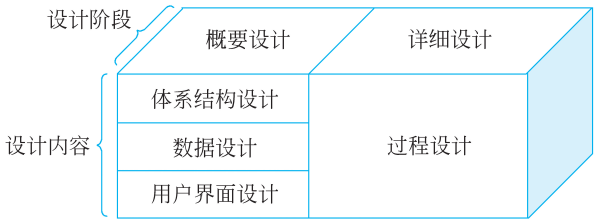


图 5-2 软件设计阶段的划分与任务

1）概要设计

概要设计也称总体设计，从需求分析阶段的工作结果出发，明确可选的技术方案，做好划分软件结构的前期工作，然后划分出组成系统的物理元素，并进行软件体系结构设计、数据设计和用户界面设计。

概要设计的主要参与者有软件分析人员、用户、软件项目管理人员以及相关的技术专家。软件分析人员完成对目标系统的物理方案和最终的软件结构设计；用户参与评价并最终审批系统的物理方案和最终的软件结构；软件项目管理人员参与评价软件分析人员设计的系统物理方案和软件结构，并对软件分析人员的设计工作进行指导；相关的技术专家则主要参与评价软件分析人员设计的系统物理方案以及软件结构。

概要设计的主要任务是完成体系结构设计、数据设计和用户界面设计。

（1）体系结构设计。确定各子系统模块间的数据传递和调用关系。在结构化设计中，体现为模块划分，并通过数据流图和数据字典进行转换。在面向对象设计中，体现为主题划分，主要确定类及类间关系。

（2）数据设计。数据设计包括数据库、数据文件和全局数据结构的定义。在结构化设计中，通过需求阶段的实体-联系图、数据字典建立数据模型。在面向对象设计中，通过类的抽象与实例化，以及类的永久存储设计，完成数据设计过程。

（3）用户界面设计。包括与系统交互的人机界面设计，以及模块间、系统与外部系统的接口关系。在结构化设计中，根据数据流条目，定义模块接口、全局的数据结构。在面向对象设计中，定义关联类、接口类、边界类等，既满足人机交互界面数据的统一，又完成类间数据的传递。

2）详细设计

详细设计的任务是在概要设计的基础上，具体实现各部分的细节，直至系统的所有内容都有足够详细的过程描述，使得编码的任务就是将详细设计的内容“翻译”成程序设计语言。确切地说，详细设计的任务是完成过程设计。


概要设计过程

过程设计包括确定软件各模块内部的具体实现过程及局部数据结构。在结构化设计中，模块独立性约束了数据结构与算法相分离的情况，使得二者在设计时务必有局部性，减少外部对二者的影响。在面向对象设计中，类的封装性较好地体现了算法和数据结构的内部性。类的继承性提供了多个类（类家族）共同实现过程设计的机制。

### 2. 软件设计的原则

随着软件开发技术不断进步，一些良好的设计原则不断被提出，并指导软件设计过程，确保软件质量。

（1）分而治之。分而治之是用于解决大型、复杂程度高的问题时所采用的策略。把大问题划分成若干小问题，把对一个大问题的求解转换为对若干小问题的解答，这样就极大地降低了问题的复杂度。模块化是软件设计实现分而治之思想的技术手段。在结构化设计中，模块可以是函数、过程，甚至是代码片段。在面向对象设计中，类是模块的主要形式。

（2）重用设计模式。重用是指同一事物不作修改或稍作改动就能多次使用的机制。由于概要设计完成的是系统软件结构，因而重用的内容是软件设计模式。软件设计模式针对一类软件设计的过程和模型，而不是面对一次具体的软件设计。通过重用设计模式，不仅使得软件设计质量得到保证，而且把资源集中于设计中的新流程、新方法中，并在设计时更进一步考虑新流程、新方法在将来的重用。

（3）可跟踪性。软件设计的任务之一就是确定软件各部分间的关系。因为设计系统结构就是要确定系统各部分、各模块间的相互调用或控制关系，以便在需要修改模块时，能掌握与修改模块有关的其他部分，并正确追溯问题根源。

（4）灵活性。设计的灵活性主要指设计具有易修改性。修改包括对已有设计的增加、删除、改动等活动。发生修改的原因主要有，用户需求发生变更，设计存在缺陷，设计需要进行优化，设计利用重用。

软件设计灵活性主要通过系统描述问题的抽象来体现。抽象是对事物相同属性或操作的统一描述，具有广泛性。因此，系统设计和设计模式的抽象程度越高，覆盖的范围就越大。如“鸟”对“麻雀”的抽象，既能体现麻雀能飞的特性，也覆盖了其他鸟类的说明。但抽象是一把“双刃剑”，过度的抽象反而会引起理解和设计上的困难。如用“生物”去抽象“麻雀”实体，则作为马的很多特征将难以在“生物”中定义。

（5）一致性。一致性在软件设计方法和过程中都得到体现。在软件设计中，界面视图的一致性保证了用户体验和对系统的忠诚度，如 Windows 操作系统的界面，虽历经多个版本的变更，但用户操作方式基本没有改变。用统一的规则和约束规范模块接口定义，确保编码阶段对接口和数据结构的统一操作，减少数据理解上的歧义，使得软件质量得到保证。

### 3. 软件设计说明书

软件设计说明书可分为概要设计说明书和详细设计说明书，软件设计说明书的完成标志着软件设计的完成，同时也为后续的软件编码提供指导和参考。

软件设计说明书主要包括如下内容。

（1）引言。说明编写设计说明书的目的，软件系统开发背景，用到的专业术语的定义和外文首字母组词的原词组，以及有关的参考资料。

（2）概要设计。说明系统主要输入输出项目、处理的功能及性能的要求；系统运行环

境的要求，基本设计概念和处理流程，系统的元素标识符和功能，功能需求与程序的关系，以及尚未解决而应在系统完成之前必须解决的问题。

（3）接口设计。说明系统涉及的用户接口、外部接口、内部接口。

（4）运行设计。说明系统运行模块组合，运行控制方式与运行时间。

（5）系统数据结构设计。包括逻辑结构设计要点、物理结构设计要点、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，故障出现后可能采取的变通措施，系统维护设计。

（7）程序系统的结构。列出系统内每个程序的名称、标识符和它们之间的层次结构关系。

（8）程序设计说明。逐个列出各个层次中的每个程序的设计思路，包括程序描述、功能、性能、输入项、输出项、算法、流程逻辑、接口、存储分配、注释设计、限制条件、测试计划，以及尚未解决应在软件完成之前应解决的问题。

### 5.1.2 软件设计基本原理

软件设计基本原理有软件的模块化、抽象与逐步求精、信息隐藏和局部化、模块独立性等。在软件工程中，模块化是大型软件设计的基本策略。

#### 1. 模块化

模块(Module)是能够单独命名，由边界元素限定的程序元素的序列。在软件的体系结构中，模块能独立地完成一定的功能，是可以组合、分解和更换的单元。

模块化(Modularization)是指把系统分割成能完成独立功能的模块，明确规定各模块及其输入输出规格，使模块的界面不会产生任何混乱。模块化对复杂问题进行分割后，每个模块的信息量小，问题简单，便于对系统进行理解和处理。

假设函数 $C(x)$ 定义了问题 $x$ 的复杂性，解决它所需的工作量函数为 $E(x)$。对于问题 $P_1$ 和 $P_2$，如果 $C(P_1)>C(P_2)$，即 $P_1$ 比 $P_2$ 复杂，那么 $E(P_1)>E(P_2)$，即问题越复杂，所需要的工作量越大。

根据解决一般问题的经验，规律为

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

即一个问题由两个问题组合而成的复杂度大于分别考虑每个问题的复杂度之和。这样可以推出

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

由此可知，开发一个大而复杂的软件系统，将它进行适当的分解，不但可降低其复杂性，还可减少开发工作量，从而降低开发成本，提高软件生产率。但是模块划分越多，块内的工作量减少，模块之间接口的工作量增加了，如图 5-3 所示。因此在划分模块时，应减少接口的代价，提高模块的独立性。

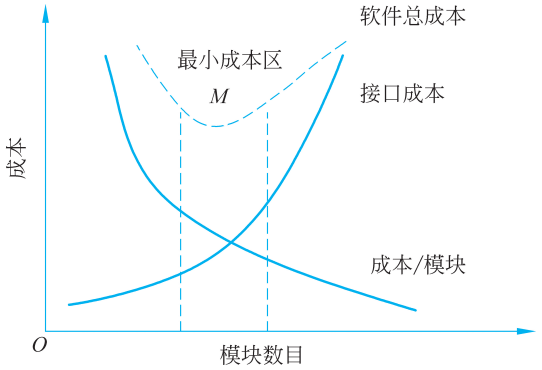


图 5-3　软件设计成本与模块数量关系图

### 2. 抽象与逐步求精

在现实世界中，事物、状态或过程之间存在共性。把这些共性集中和概括起来，忽略它们之间的差异，这就是抽象。抽象就是抽出事物的本质特性而暂时不考虑它们的细节。

当考虑对任何问题的模块化解法时，可以提出许多抽象的层次。在抽象的最高层次使用问题环境的语言，以概括的方式叙述问题的解法；在较低抽象层次采用更过程化的方法，把面向问题的术语和面向实现的术语结合起来叙述问题的解法；在最低的抽象层次用可以直接实现的方式叙述问题的解法。

软件工程过程的每步都是对软件解法的抽象层次的一次精化。在可行性研究阶段，软件作为系统的一个完整部件；在需求分析期间，软件解法是使用在问题环境内熟悉的方式描述的；当我们由总体设计向详细设计过渡时，抽象的程度也就随之减少了；当源程序写出来以后，也就达到了抽象的最底层。

逐步求精与抽象是紧密相关的，随着软件开发工程的进展，在软件结构每层中的模块，表示了对软件抽象层次的一次精化。层次结构的上一层是下一层的抽象，下一层是上一层的求精。事实上，软件结构顶层的模块，控制了系统的主要功能并且影响全局；在软件结构底层的模块，完成对数据的一个具体处理，用自顶向下、由抽象到具体的方式分配控制，简化了软件的设计和实现，提高了软件的可理解性和可测试性，并且使软件更容易维护。

### 3. 信息隐蔽和局部化

应用模块化原理时，将产生一个问题：为了得到一组模块，应该如何分解软件结构？信息隐蔽原理指出，每个模块的实现细节对于其他模块是隐蔽的，即模块中所包括的信息不允许其他不需要这些信息的模块调用。隐蔽表明有效的模块化可以通过定义一组独立的模块而实现，这些独立的模块间仅交换为完成系统功能而必须交换的信息。

模块间的通信仅使用对于实现软件功能的必要信息，通过抽象，可以确定组成软件的过程实体；而通过信息隐蔽，则可以定义和实施对模块的过程细节和局部数据结构的存取限制。局部化的概念和信息隐蔽概念密切相关，局部化是指把一些关系密切的软件元素

物理地放得彼此靠近，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。

如果在测试期间和以后的软件维护期间需要修改软件，使用信息隐蔽原理作为模块化系统设计的标准就会带来极大好处。因为绝大多数数据和过程对于软件的其他部分是隐蔽的，也就是看不见的，在修改期间由于疏忽而引入的错误传播到软件的其他部分的机会就很少。

### 4. 模块独立性

为了降低软件系统的复杂性，提高可理解性、可维护性，必须把系统划分成为多个模块。模块不能任意划分，应尽量保持其独立性。模块独立性指每个模块只完成系统要求的独立的子功能，并且与其他模块的联系最少且接口简单。

如何衡量软件的独立性呢？根据模块的外部特征和内部特征，提出了两个度量标准——耦合和内聚。

1）耦合

耦合是指软件系统结构中各个模块之间相互联系紧密程度的一种度量。模块之间联系越紧密，其耦合性就越强，模块的独立性则越差。模块间耦合高低取决于模块之间接口的复杂性、调用的方式及传递的信息。

模块之间的耦合性一般分为 7 种类型，如图 5-4 所示。

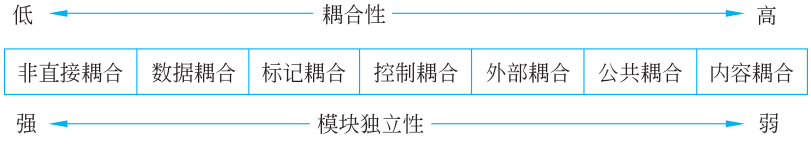


图 5-4　耦合的类型

非直接耦合指两个模块之间没有直接的关系，它们分别从属于不同模块的控制与调用，它们之间不传递任何信息；数据耦合指两个模块之间有调用关系，传递的是简单的数据值，相当于高级语言中的值传递；标记耦合指两个模块传递的是数据结构，例如，高级语言中的数组名、记录名、文件名等这些名字即为标记，其实传递的是这个数据结构的地址；控制耦合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些功能；外部耦合指一组模块都访问同一个全局简单变量而不是同一个全局数据结构，并且不通过参数表传递该全局变量的信息；公共耦合指通过一个公共数据环境相互作用的那些模块间的耦合；当一个模块直接使用另一个模块的内部数据，或通过非正常入口而转入另一个模块内部时，这种模块之间的耦合为内容耦合。

耦合性是影响软件复杂程度的一个重要因素，在设计中应该尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，完全不用内容耦合。

2）内聚

内聚是指模块的功能强度的度量。若一个模块内各元素（语句之间、程序段之间）联系得越紧密，则它的内聚性就越高。

模块之间的内聚性一般分为 7 种类型，如图 5-5 所示。

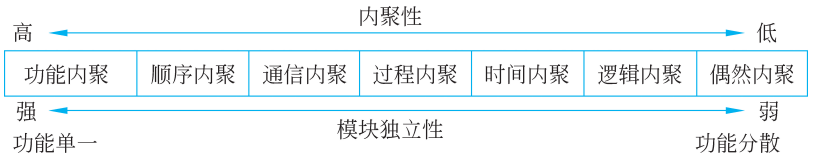


图 5-5　内聚的类型

模块内部所有元素都属于一个整体，它们组合在一起是为了完成某个独立的功能，则该模块的内聚是功能内聚；模块内部各部分彼此紧密联系，为实现某个功能结合在一起，并按照顺序方式执行，则该模块的内聚是顺序内聚；模块内部的所有元素都使用相同的输入数据或产生相同的输出结果，则该模块的内聚是通信内聚；模块内部的所有元素彼此相关，但必须遵循特定的过程次序执行，则该模块是过程内聚；模块内部的所有组成部分必须在同一段时间内执行完成（如所有的初始化或终止工作），则该模块是时间内聚；模块内部的各组成部分除了通过逻辑变量（也称控制参数）联系之外无任何联系，则该模块是逻辑内聚；组成模块的元素之间没有实质性的联系，则该模块是偶然内聚。

在设计时更应重视模块内聚，尽量追求功能内聚，少用逻辑内聚和偶然内聚，可以酌情使用顺序内聚和通信内聚。此外，没有必要精确定义内聚的级别，只要能够识别出低内聚的模块即可。

【例 5-1】　为实现一个堆栈，一个模块中含几个子程序，如 init_stack()、push()和 pop()；模块中同时还含有格式化报告数据和定义子程序中用到的所有全局数据和子程序。很难看出堆栈与报告子程序或全局数据部分有什么联系，因此模块的内聚性很差。这些子程序应该按照模块内聚的原则进行重新组织。

耦合性与内聚性是模块独立性的两个度量标准，将软件系统划分模块时，尽量做到高内聚、低耦合，提高模块的独立性，为设计高质量的软件结构奠定基础。

### 5. 软件设计原则

改进软件设计，提高软件质量需要遵循如下原则。

1）模块高独立性

设计出软件的初步结构以后，应该进一步分解或合并模块，力求降低耦合并提高内聚。例如，多个模块公有的一个子功能可以独立定义一个模块，由这些模块调用；有时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，并降低接口的复杂程度。

2）模块规模适中

大的模块往往是由于分解不充分，但是进一步分解必须符合问题结构，一般分解后不应该降低模块独立性。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而不必单独存在。

3）深度、宽度、扇出和扇入适当

深度表示软件结构中控制的层数，能够粗略地标志一个系统的大小和复杂程度，如

图 5-6 所示。它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过于简单，需要适当合并。宽度是软件结构内同一个层次上的模块总数的最大值。一般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。

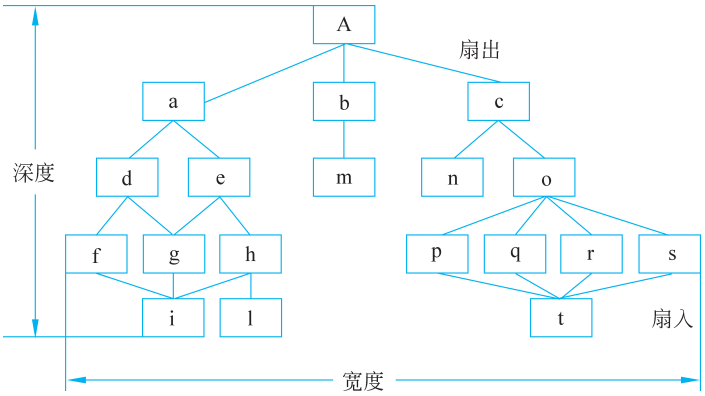


图 5-6 程序结构的有关术语

扇出是一个模块直接调用的模块数目，扇出过大意味着模块过于复杂，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，一个设计得很好的典型系统的平均扇出通常是 3 或 4。扇出太大一般是因为缺乏中间层次，应该适当增加中间层次的控制模块。扇出太小时可以把下级模块进一步分解成若干子功能模块，或者合并到它的上级模块中去。当然，分解模块或合并模块必须符合问题结构，不能违背模块独立原理。

一个模块的扇入表明有多少个上级模块直接调用它，扇入越大则共享该模块的上级模块数目越多，这是有好处的，但是，不能违背模块独立原理而单纯追求高扇入。

观察大量软件系统后发现，设计得优秀的软件结构通常顶层扇出比较高，中层扇出较少，底层扇入公共的实用模块中。

4）模块的作用域应该在其控制域之内

模块的作用域定义为受该模块判定影响的所有模块的集合。模块的控制域是这个模块本身以及所有直接或间接从属于它的模块的集合。例如，在图 5-7 中，模块 A 的控制域是 A、B、C、D、E、F 模块的集合。

在一个设计得很好的软件系统中，所有受判定影响的模块应该都从属于做出判定的那个模块，最好局限于做出判定的那个模块本身及它的直属下级模块。例如，如果图 5-7 中模块 A 做出的判定只影响模块 B，符合这条规则。但是，如果模块 A 做出的判定同时还影响模块 G 中的处理过程，这样的结构使得软件难于理解。为了使 A 中的判定能影响 G 中的处理过程，通常需要在 A 中给一个标记设置状态以指示判定的结果，并且应该把这个标记传递给 A 和 G 的公共上级模块 M，再由 M 把它传给 G。这个标记是控制信息而不是数据，因此将使模块间出现控制耦合。

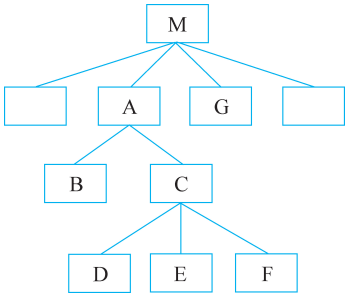


图 5-7 模块的作用域和控制域

可以通过修改软件结构使作用域是控制域的子集：一个方法是把做判定的点往上移，例如，把判定从模块 A 中移到模块 M 中；另一个方法是把那些在作用域内但不在控制域内的模块移到控制域内，例如，把模块 G 移到模块 A 的下面，使 G 成为 A 的直属下级模块。

5）模块接口的低复杂度

模块接口复杂是软件发生错误的主要原因之一。应该设计模块接口使得信息传递简单并且和模块的功能一致。

【例 5-2】 一元二次方程的根的模块定义为

```
QUAD_ROOT(TBL,X)
```

其中，用数组 TBL 表示方程的系数，用数组 X 回送求得的根。这种传递信息的方法不利于对这个模块的理解，不仅在维护期间容易引起混淆，在开发期间也可能发生错误。下面这种接口可能比较简单：

```
QUAD_ROOT(A,B,C,ROOT1,ROOT2)
```

其中，A、B、C 是方程的系数，ROOT1 和 ROOT2 是算出的两个根。

接口复杂或者不一致是高耦合或低内聚的原因所致，应该重新分析这个模块的独立性，力争降低模块接口的复杂程度。

6）单入口、单出口的模块

这条启发式规则表明，在设计软件结构时不要使模块间出现内容耦合。在结构上模块顶部有单入口，模块底部有单出口，这样的结构比较容易理解和维护。

7）模块功能应可预测

如果一个模块可以当作一个黑盒子，只要输入的数据相同就产生同样的输出，这个模块的功能就是可以预测的。带内部存储器的模块的功能可能是不可预测的，因为它的输出可能取决于内部存储器（例如，某个标记）的状态。由于内部存储器对于上级模块而言是不可见的，因此这样的模块不易理解，难于测试和维护。

如果一个模块只完成一个单独的子功能，则表现高内聚；但是，如果一个模块任意限制局部数据结构的大小，过分限制在控制流中可以做出的选择或者外部接口的模式，这种模块的功能就过分局限，使用范围也过于狭窄。在使用过程中将不可避免地需要修改功能过分局限的模块，以提高模块的灵活性，扩大它的使用范围；但是，在使用现场修改软件的代价是很高的。

设计相关概念

## 5.2 软件设计技术过程

软件设计从工程管理角度可分为总体设计和详细设计两个步骤，从技术角度可分为软件体系结构设计、数据设计、过程设计和用户界面设计四大部分。

### 5.2.1 软件体系结构设计

软件体系结构为软件系统提供了一个结构、行为和属性的高级抽象，由构成系统的元素的描述、元素间的相互作用、指导元素集成的模式，以及这些模式的约束组成。软件体系结构不仅指定了系统的组织结构和拓扑结构，显示了系统需求和构成系统的元素之间的对应关系，而且提供了一些设计决策的基本原理。良好的体系结构是普遍适用的，它可以高效地处理各种各样的个体需求。

#### 1. 软件体系结构设计过程

软件体系结构设计是软件设计的早期活动，侧重于建立系统的基本结构性框架，即系统的宏观结构，而不关心模块的内部算法。

一般的软件体系结构设计过程主要包括如下 3 项活动。

(1) 系统结构(System Structure)设计。系统结构设计将系统划分为一些主要的独立子系统，确定子系统间的通信方式。

(2) 控制建模(Control Modeling)。控制建模建立系统各部分之间的控制关系。

(3) 模块分解(Modular Decomposition)。模块分解将子系统分解为模块。

上述活动通常不是按顺序而是交错进行的。在任何一个过程中，设计者都应当提供更多的细节以供决策，最终使设计满足系统的需求。软件体系结构设计的好坏将会直接影响系统的性能、健壮性、可分布性和可维护性。

在实际设计过程中，并不需要真正地去建立一个全新的体系结构模型，而是从典型、成熟的模型中选择，大型、复杂的系统可能会选择多种结构。有经验的设计者一定会在其中做出某些变通，以适应实际系统的需求。

#### 2. 软件体系结构模型

随着计算机网络技术和软件技术的发展，软件体系结构和模式也在不断发生变化，下面介绍 3 种常见的软件体系结构模型。

1) 仓库模型

仓库模型(Repository Model)是一种集中式模型。在这种结构模型中，应用系统用一个中央数据仓库来存储各个子系统共享的数据，其他子系统可以直接访问这些共享数据。当然，每个子系统可能会有自己的数据库。为了共享数据，所有的子系统都是围绕中央数据仓库紧密耦合的。

仓库模型以数据为中心，能独立提供数据服务的封闭式数据环境。它不单独集成到某一应用系统中，而是为具体的应用系统提供服务。这些服务既有通用的公共服务，也有专门设计的领域服务。

仓库模型中，数据统一存储和管理，确保了数据的实时性；对数据复杂性的统一封装，有利于数据共享；采用黑板模型，与某类数据有关的应用系统能及时获取数据；采用数据订阅推送模型，应用系统在有数据更新时，能自动获得数据，而不用采取询问方式，这就提