第5章 结构化实现

结构化实现是指软件开发团队在结构化设计的基础上，结合目标软件项目特点，使用合适的结构化程序设计语言将详细设计内容转换为软件系统，并对完成的软件系统进行质量保障的过程。具体而言，结构化实现包括结构化编码和结构化测试两个部分的内容。

为了获得高质量的程序代码，软件开发人员必须认真分析结构化设计阶段完成的系统架构图、IPO 表、数据设计和过程设计，将结构化设计的成果映射为软件模块、数据结构、软件文件结构，并依据系统结构图对完成的代码进行集成。

除此以外，软件测试人员还将结合结构化设计和软件需求来设计适合各个模块以及目标软件系统的白盒测试用例和黑盒测试用例，并按照软件测试策略对目标软件系统开展单元测试、集成测试、验收测试等工作。

本章将为大家介绍结构化实现过程中包括的工作内容。

5.1 结构化实现与结构化设计的关系

在结构化设计阶段，软件设计人员对目标软件系统的结构、模块接口、处理的数据以及模块的内部细节进行了设计。在结构化实现阶段，软件开发人员必须根据结构化设计阶段完成的内容，将结构化设计转换为数据结构、函数模块、代码文件和文件组织结构等。

1. 数据实现

在结构化设计阶段，软件设计人员通过数据模型对目标软件系统涉及的数据元素、数据结构、数据存储结构（文件、数据库）进行设计，软件开发人员可以直接将结构化设计阶段完成的数据设计转换为结构化实现阶段对应的数据结构、文件结构和数据库物理结构。

2. 函数模块

由于软件模块对应的 IPO 表、程序流程图与数据结构分别对模块的接口、处理流程和处理数据进行了描述，软件开发人员可以直接将结构化设计映射为结构化实现中的“函数”模块。

3. 代码文件及文件组织结构

在结构化实现过程中，软件开发人员可以依据结构化设计阶段完成的系统结构图来组织程序代码，并形成对应的代码文件和文件组织结构。

4. 软件测试用例设计

软件测试人员可以依据模块的内部结构来设计模块的白盒测试用例，根据 IPO 表和软件需求来设计模块的黑盒测试用例。

5. 软件项目集成

软件开发人员和软件测试人员可以结合不同层次的系统结构图和模块过程设计来集成各个软件模块，并设计集成测试用例。

结构化设计与结构化实现之间的对应关系如图 5-1 所示。

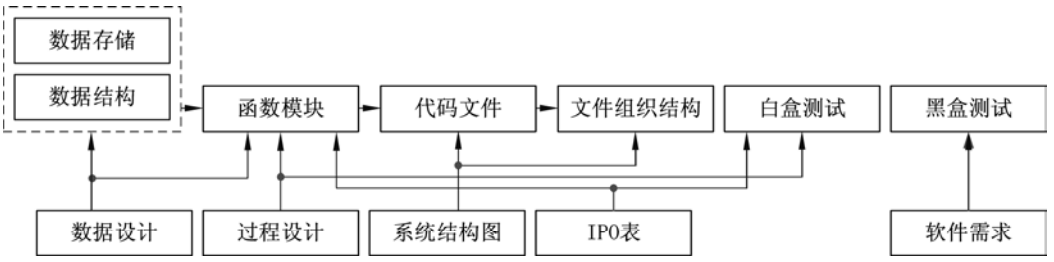

图 5-1　结构化设计与结构化实现之间的对应关系

可以发现，良好的结构化设计是得到高质量程序代码的保障。如果在结构化实现过程中发现了设计缺陷，或者在软件测试中发现了非程序性错误，软件开发团队必须重新返回到软件设计阶段，对相应的软件设计内容进行完善，然后再结合设计结果来开展结构化实现方面的工作。

5.2　结构化编码

在复杂的软件项目中，由某个软件开发人员单独完成整个软件系统的开发工作是不可能的。软件开发团队必须协同工作，共同完成目标软件系统的开发。

对于小型的软件项目而言，由于程序代码逻辑简单，且代码量较少，软件开发人员往往将所有程序代码写入到一个程序源文件中。然而，随着软件规模的不断扩大，软件涉及的业务流程分支急剧增多，代码量也成倍增长。此时，如果软件开发人员仍然将所有程序逻辑代码放在一个源代码文件中，则该源代码文件的内容将会变得非常庞大，且内容凌乱，不利于阅读，严重影响将来的代码修改和维护工作。

因此，在进行结构化编码之前，软件开发团队除了需要统一整个团队的编码规范以外，还需要结合软件设计来组织程序源代码的内容和程序源文件，提高编码的效率。

5.2.1　结构化程序的源代码组成

通常而言，高级程序设计语言均提供了分文件代码组织方式，即允许软件开发人员将软件代码分解到多个不同类型的文件中，以便对程序代码内容进行分类组织、管理。

以 C 语言为例，C 语言编写的项目源程序代码可以分为头文件（.h）、源程序文件（.c）和主程序源文件（包含 main 函数的源程序文件）三个部分。

1. 头文件

在 C 语言中，头文件是扩展名为.h 的源程序文件，主要用于全局变量、全局宏、函数的声明以及被其他源代码文件引用共享。

通常而言，C 语言的头文件包括以下四个方面的内容。

1）头文件区

头文件区用于包含项目开发过程中引用的系统头文件以及软件开发人员创建的头文件（如 MyFunc.h 等）。

2）全局宏区

全局宏区用于定义模块中公用的宏（#define），例如符号常量、带参数的宏等。

3）全局变量区

全局变量区中包含所有模块共同使用的公共变量（非 static）。

4）函数接口区

函数接口区给出源文件中定义的函数接口。

2. 源程序文件

源程序文件以.c 为扩展名，主要用于实现程序代码逻辑，包括主函数代码及头文件中定义的业务函数代码。

在编译软件项目时，编译器首先会在代码预处理阶段将多个程序源文件合并为一个临时中间文件，然后再对临时中间文件进行编译。可以发现，在项目开发过程中，软件开发人员将代码逻辑写入头文件或者源程序文件都可以参与编译，并最终得到目标代码。

既然如此，那么 C 语言为什么还要将程序源代码分为头文件和源程序文件呢？为什么一般在头文件中进行函数、全局变量、全局宏、全局结构体的声明，而在源程序文件中进行变量定义、函数实现呢？其实，在项目实现过程中，将程序代码分为头文件和源程序文件具有以下优点。

（1）将函数实现写入源程序文件，可以减少中间文件冗余。

如果把函数实现写在头文件中，当头文件被多次包含或者嵌套包含时，函数实现代码将会被多次嵌入到临时中间文件中，导致临时中间文件出现大量的冗余内容。与此同时，由于头文件中实现的函数均为全局函数，无法将函数定义为局部函数，当不同模块需要定义实现特定功能的、相同名称的函数时，将造成函数命名冲突。

（2）将公共变量写入头文件，有利于变量共享。

如果将变量写入到头文件中，编译器将为该变量生成一个全局空间，便于所有源程序文件共享该变量的内容（全局变量）。相反，如果将变量写到源程序文件中（未强制设为全局变量），编译器在编译源程序文件时，将根据文件中变量的数量来生成对应的空间。

（3）将宏和数据类型定义写入头文件中，有利于统一数据信息。

将公共使用的宏、结构体声明放到头文件中，可以方便地被源程序文件共享。当源程序文件需要使用公共的数据内容时，源程序文件仅需通过#include 语句包含定义数据内容的头文件即可。同时，当需要修改宏和结构体的内容时，软件开发人员可以直接修改头文件中的宏和结构体声明，无须进入各个源程序文件中进行修改。

（4）将业务逻辑写到源程序文件中，有利于保护实现业务逻辑的源程序内容。

在代码重用过程中，一种情况是将模块的头文件和实现业务逻辑的源程序文件都交给对方，直接进行代码复用。但是，直接代码复用方式不利于保护软件的知识产权。

如果程序的源代码不便（或者不准）向其他人员公布，软件开发人员为了避免其他人员直接阅读代码和修改程序业务逻辑，可以将函数声明写成头文件，将函数的业务逻辑代码写入源程序文件，并将源程序文件编译成“库”。

在高级程序设计语言中，“库”文件是经过编译的二进制文件，使用人员无法通过正常途径来查看库中的程序源代码。软件开发人员可以通过库文件对应的头文件来了解“库”中提供的函数功能，并进行功能调用。编译器也会从库中提取出相应的代码来响应调用。

（5）使用头文件可以加强类型安全检查。

在实现或者调用函数时，如果函数的定义形式与头文件中的声明不一致，编译器将会指出错误，减轻软件开发人员调试程序以及改正错误代码的负担。

因此，在利用多个文件对源程序代码进行组织时，将全局宏、全局变量和函数接口写入

头文件，将程序业务逻辑代码写入源程序文件是一个良好的编程习惯。

5.2.2 结构化程序的编译过程

在进行程序源代码的组织和文件划分之前，首先需要了解一下程序的编译过程。以 C 语言编写的软件项目为例，编译器以文件为单位来编译软件项目源代码，即编译器首先逐个读取各个源程序文件，将多个源程序文件合并为一个中间文件，通过对多个源程序文件中涉及的函数与变量进行重定位，最终生成可执行文件。

通常而言，高级程序设计语言编写的软件项目在编译时需要经过预处理、词法与语法分析、编译和链接四个阶段。

1. 预处理

C 语言编写的程序是从 main 函数开始执行的。在预处理阶段，编译器首先扫描所有的源程序文件，并找出包含 main 函数的源程序文件，即主程序源文件；接着，编译器逐行扫描主程序源文件，处理该文件中的文件包含、宏定义、变量声明、函数声明和程序逻辑等，并生成一个临时的中间“C 文件”。

1）处理#include 语句

当预处理过程扫描到了文件包含语句（#include）时，编译器将会把被包含的目标文件内容引入到临时编译文件中。例如，如果预处理过程遇到#include <stdio.h>语句，编译器将把 stdio.h 文件中的内容包含到临时编译文件中。

由于软件项目的多个源程序文件是并行开发的，软件项目在代码集成过程中可能会出现头文件嵌套包含的情况，即被包含的头文件中又包含了其他头文件。此时，编译器将根据语法关系，将多个头文件内容逐层导入到临时编译文件中。如果某个头文件被多次包含，该文件会被多次导入到临时中间文件中。然而，由于编译器并不会自动删除或者处理重复的包含内容，被多次导入的头文件内容将被多次定义，从而引发内容定义冲突。

在 C 语言中，软件开发人员可以在头文件的首部和尾部增加标志性宏定义，借助条件编译来避免头文件内容被重复包含的情况。例如图 5-2 所示代码借助条件编译#if 和#endif，结合标志 TOUWENJIANNAME 来避免头文件内容被重复包含。

```
#if !defined(TOUWENJIANNAME)
//TOUWENJIANNAME是项目中唯一的标识符号
//可以直接使用头文件的名称
#define TOUWENJIANNAME

//头文件内容

#endif
```

图 5-2　条件编译案例

以图 5-2 所示程序代码为例，当编译程序第一次读取并处理头文件中的内容时，宏 TOUWENJIANNAME 尚未被定义。此时，编译程序将执行头文件中的宏定义内容，并且将头文件中的内容引入到临时编译文件中；当编译程序再次遇到相同的头文件内容时，宏 TOUWENJIANNAME 已经被定义。此时，编译程序将忽略后期的头文件内容，即重复包含的头文件内容不再被引入临时编译文件。

2）处理宏定义#define

当预处过程扫描到宏定义语句（#define）时，编译器将使用宏定义中的内容来替换源程

序代码中出现的宏。例如，图 5-3（a）中的代码通过编译后将转换为图 5-3（b）所示的内容。

```
#define PI 3.1415926
#define POWER(x) x*x
#define MAX(a,b) (a>b)?a:b

void main()
{
    double CircleArea = PI * POWER(5);

    int t = MAX(100,20);
}
```

(a) 宏替换前

```
#define PI 3.1415926
#define POWER(x) x*x
#define MAX(a,b) (a>b)?a:b

void main()
{
    double CircleArea = 3.1415926 * 5 * 5;

    int t = (100>20)?100:20;
}
```

(b) 宏替换后

图 5-3　宏定义案例

可以发现，程序的预处理过程就是编译程序将程序源代码中的所有包含文件、全局变量等合并为一个临时中间文件，以及替换中间文件中的宏定义内容的过程。

当编译器将头文件中的内容引入到临时中间文件后，它将继续寻找与头文件名称相同的源程序文件。如果找到了头文件对应的源程序文件，编译器将在源程序文件中逐个定位头文件中声明函数的代码实现，继续编译；如果找不到头文件对应的源程序文件，同时在其他包含文件中也没有找到声明函数的实现代码，编译器将返回编译错误。

2. 词法与语法分析

在词法与语法分析阶段，编译器将根据 C 语言的语法规则对生成的临时中间文件进行语法检查，确保临时文件中没有语法错误。

3. 编译

在编译阶段，编译器先将临时中间文件编译成汇编语句，然后再将汇编语句编译成与操作系统相关的二进制代码，并生成对应的目标文件（.obj 文件）。

4. 链接

在链接阶段，编译器将根据软件开发人员设置的参数信息，将编译阶段产生的多个目标文件与系统库文件进行链接。在编译过程中，编译器将对文件中的代码进行绝对地址定位，生成与指定操作系统平台相关的可执行文件（*.exe）。

5.2.3　结构化程序多文件组织

在结构化实现过程中，软件开发人员将程序源代码分散到不同的源程序文件中除了有利于降低目标软件系统的实现复杂度以外，还可以开展多文件并行开发，提高软件产品的开发效率。那么，在结构化实现过程中，如何将结构化设计包含的模块和处理逻辑组织到不同的程序头文件和源程序文件中呢？

通常而言，软件开发人员可以将功能比较集中或者相关的业务放到一个程序源文件中，即将关系比较紧密的多个函数放入同一个源程序文件；将关联比较紧密的函数定义、变量定义、宏定义等内容放入同一个头文件。当然，软件开发人员也可以根据“系统结构图”中的模块划分来组织源程序文件中的内容。

一般情况下，以 C 语言编写的软件项目源代码可以分为头文件、源程序文件和主程序源文件三个部分。除了区分主程序源文件以外，软件开发人员仅需要考虑程序代码的头文件、源程序文件划分和源代码组织即可。

1. 公共头文件划分

程序代码内容划分的第一项工作就是为整个软件项目创建公共的头文件。此时，软件开发人员可以根据项目的内容来划分公共头文件的头文件区（包括所有必需的系统库函数头文件）、全局宏区（对整个项目中需要的宏进行定义）、全局变量区（定义项目中的公共变量）和函数接口区（定义主程序源文件中需要的函数）等多个部分。

2. 程序源文件划分

公共头文件设置完成以后，软件开发人员可以结合“系统结构图”来创建各个模块的头文件和源程序文件，然后再对各个模块的内容进行规划。

一般而言，软件开发人员可以遵循以下步骤来划分软件项目的程序源文件。

（1）创建主程序源文件（包括 main 函数的源程序文件）。

主程序源文件是指包括 main 函数的源程序文件，也可以认为是软件项目的代表文件。编译器在编译软件项目时，首先编译主程序源文件。通常，软件开发人员可以将系统结构图的顶层模块规划为主程序源文件。同时，为了标识主程序源文件，软件开发团队可以直接采用项目的名称或者代号为主程序源文件命名，例如，ChatServer.c、SkyWalker.c 等。

（2）为系统结构图中的其他模块定义头文件和源程序文件。

主程序源文件创建完成以后，软件开发人员可以根据软件设计内容，为系统结构图中的其他模块创建头文件和源程序文件。

此时，软件开发人员可以采用模块名对应的英文表述来命名各个模块对应的头文件和源程序文件。为了让软件项目的代码文件组织清晰，便于编译，软件开发人员可以将模块对应的头文件和源程序文件以相同的名称命名。例如，“保存文件”模块对应的头文件和源程序文件可以分别命名为“FileSave.h”和“FileSave.c”。

然而，由于系统结构图中的模块与程序代码中的函数相对应，如果为每个模块都创建头文件和源程序文件将极大地增加项目源程序文件的管理难度。在规划软件项目的代码内容时，软件开发人员可以结合模块的独立性原则和软件开发需要来拆分源程序文件，将简单底层模块的内容汇合到上层模块对应的源程序文件中，即如果源程序文件的内容比较复杂，软件开发人员可以与软件设计人员沟通，将模块内容进行再次拆分；相反，如果模块的规模比较小，软件开发人员也可以考虑将模块包含的内容放入上层模块对应的头文件和源程序文件中。

（3）为系统结构图中的各个模块定义对应的测试文件。

除了为系统结构图中的模块定义对应的头文件和源程序文件以外，软件开发人员还需要为各个模块定义相应的测试驱动文件。测试驱动文件以模块的名称为依据，采用统一的方式进行命名。例如，如果模块的名称为“FileSave”，则该模块对应的测试驱动头文件和源程序文件可以分别命名为“FileSaveTester.h”和“FileSaveTester.c”。

同时，软件测试人员可以在测试驱动文件中设置执行函数，直接调用待测模块的相关内容，结合多种设定的测试用例来测试模块，确保模块的功能符合设计预期。

3. 源程序文件组织

当目标软件项目的程序源代码划分文件以后，实现软件项目业务逻辑的程序代码不再拥挤在一个文件中，而是根据系统结构图中的内容分布到多个以项目或者模块命名的头文件和程序源文件中。软件开发团队可以对程序源代码进行分文件管理，并行地开发多个软件模块，在降低软件开发复杂度的同时提高软件开发效率。

然而，如果软件项目的规模较大，模块众多，软件项目涉及的源代码文件数量也会急剧

增加。此时，如果软件开发团队仍然将项目包含的所有头文件和源程序文件放到一个文件目录中，项目的文件管理就会变得非常凌乱，为软件项目的代码管理带来较大的负担。

为了提高软件项目的文件管理效率，软件开发团队可以结合“系统结构图”中的模块划分内容，利用文件夹来组织各个项目文件，即将多个头文件或者源程序文件放入以上层模块命名或者以适当信息命名的文件夹中，降低代码文件的管理复杂度。

同样以图 4-39 中的汽车仪表盘项目为例，软件开发团队可以采用图 5-4 所示的文件结构来组织项目源代码文件。

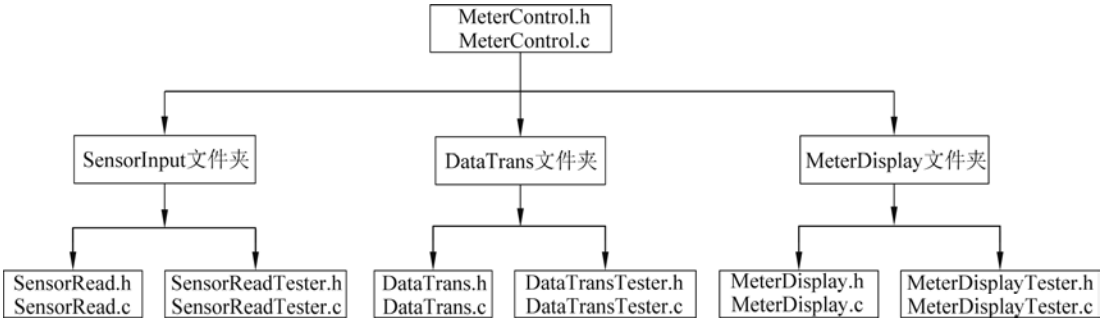

图 5-4　汽车仪表盘项目文件组织

5.2.4　结构化模块集成

当目标软件系统各个模块的编码、单元测试完成以后，软件开发团队就可以结合系统结构图和各个上层模块的处理流程来集成软件模块，得到目标软件系统或者相应子系统。

根据软件开发团队集成软件模块的方式不同，结构化程序的模块集成方式可以分为非渐增式组装和渐增式组装两种。

1）非渐增式组装

非渐增式组装，也称为整体拼装，是指软件开发团队将软件项目包含的所有模块同时集成到系统中。由于参与集成的软件模块数量众多，非渐增式组装的集成场面非常混乱。此时，软件开发团队很难定位集成过程中出现错误的原因，纠正错误困难。

2）渐增式组装

渐增式组装是指软件开发团队按照指定的顺序，逐一集成各个软件模块。

在实际的结构化集成过程中，软件开发团队常用自顶向下集成和自底向上集成两种渐增式组装方法来集成软件模块。

1. 自顶向下集成

自顶向下集成是指软件开发团队在集成软件模块时，首先从主控模块（主程序源文件）开始，沿着系统结构图和文件结构向下移动，将下层模块对应的源程序文件逐一集成到目标软件系统中。

根据底层模块的集成次序不同，自顶向下集成又可以分为深度优先和宽度优先两种模式。在深度优先集成模式中，软件开发团队依据系统结构图，逐一集成主控路径上的每一个软件模块；在宽度优先集成模式中，软件开发团队逐层集成所有下属软件模块。

通常而言，结构化实现中的自顶向下集成可以分为以下 4 个步骤。

（1）定位主程序源文件，从主控模块开始集成下层模块。

在主控模块集成过程中，目标软件系统的部分下层模块可能尚未集成，软件开发团队可

以采用桩模块来替代这些下层模块。

所谓桩模块，也称为存根模块，是指仅需保证接口正确，无须关注业务逻辑代码，直接返回所需处理结果的临时模块。桩模块主要用于替代下层未完成的软件模块，或者无法获得的下层模块。在软件测试过程中，桩模块可以作为下层模块以及被调模块的替代模块。

（2）集成下层模块。

软件开发团队根据所选择的自顶向下集成模式（深度优先或者宽度优先），采用实际的软件模块来替代上层模块集成中调用的下层桩模块。

（3）回归测试。

在目标软件系统采用一个实际的软件模块替换对应的桩模块以后，软件开发团队需要利用相应的方法对新引入的软件模块进行测试以及对软件系统原有集成的模块开展回归测试，确保新集成的软件模块不会引入新的错误。

（4）重复上述步骤（2）和步骤（3），直到所有模块集成完毕。

图 5-5 中给出了一个深度优先的自顶向下渐增式组装集成案例。其中，以字母 S 开头的模块表示为在软件集成过程中编写的桩模块。

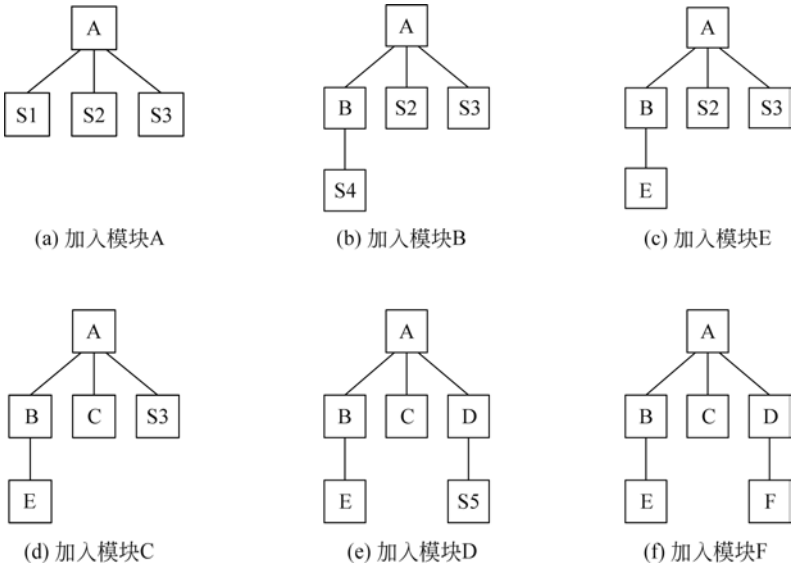

图 5-5　自顶向下集成案例（深度优先）

软件开发团队首先定位上层主控模块 A，并编写模块 A 需要的桩模块 S1、S2 和 S3；然后，软件开发团队使用模块 B 替代桩模块 S1，并为模块 B 编写桩模块 S4；由于软件开发团队按照深度优先方式集成软件，桩模块 S4 对应的模块 E 将被优先集成；当模块 E 集成完成以后，软件开发团队将集成桩模块 S2 对应的模块 C；接着，集成桩模块 S3 对应的模块 D，并为模块 D 撰写桩模块 S5；最后，软件开发团队使用桩模块 S5 对应的模块 F 替换桩模块 S5，完成整个软件系统的集成。

2. 自底向上集成

自底向上集成是指软件开发团队从系统结构图的最底层模块开始组装整个软件系统。

由于自底向上集成首先集成系统结构图中的底部模块，软件开发团队在集成上层模块时，上层模块调用的子模块（包括子模块的所有下属模块）在集成之前已经完成了集成，并且经过了回归测试。因此，在自底向上集成方式中，软件开发团队无须为上层模块编制桩模块。

根据自底向上集成的工作原理，软件开发团队可以采用以下 5 个步骤来集成目标软件。

（1）划分底层模块的功能簇。

在进行模块集成之前，软件开发团队先将底层模块组合成实现某个子功能的簇。

（2）编写功能簇的驱动程序。

软件开发团队为每个功能簇编写驱动程序，用于提供该功能簇的执行入口和输入、输出界面。

（3）编写功能簇测试程序。

软件开发团队为每个集成的功能簇编写测试驱动程序，排除功能簇在集成过程中出现的错误和缺陷。

（4）扩充底层功能簇。

软件开发团队在更高的层次上抽象功能簇以及在剩下的底层模块中寻找新功能簇，或者扩大原有的功能簇，根据自底向上集成方式的工作原理将新的底层模块集成到已完成的功能簇中。必要时，软件开发团队可以重新修改测试驱动程序，使其能够适应新功能簇的集成和测试。

（5）重复上述步骤（3）和步骤（4），直到所有模块集成为目标软件系统或子系统为止。

图 5-6 给出了一个自底向上的渐增式模块集成案例。

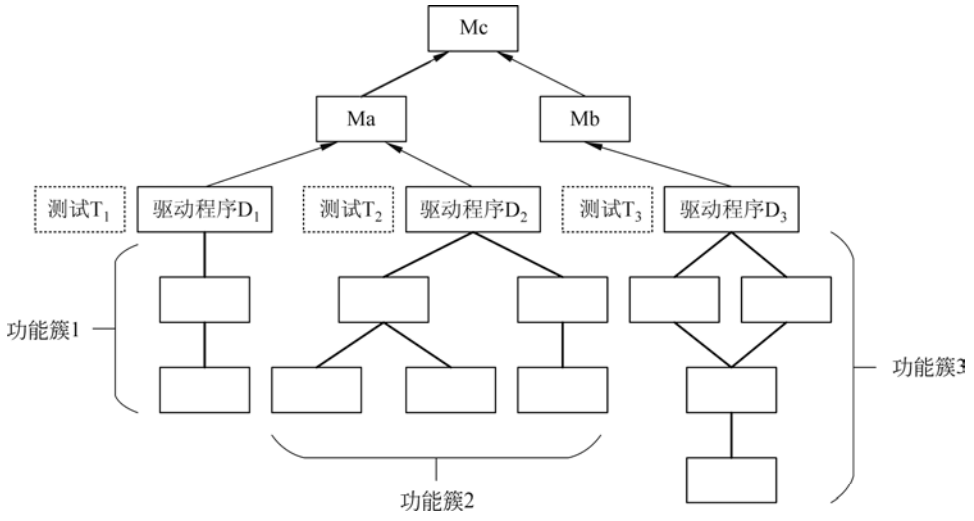

图 5-6　自底向上的模块集成案例

软件开发团队先从底层模块中寻找出功能簇 1、功能簇 2 和功能簇 3；然后，软件开发团队分别为功能簇 1、功能簇 2 和功能簇 3 编写驱动程序 D_1、驱动程序 D_2 和驱动程序 D_3，并对底层功能簇进行集成；接着，软件开发团队为每个集成的功能簇编写测试驱动程序，排除功能簇集成过程中出现的错误和缺陷；最后，软件开发团队从更高层次上抽象功能簇，不断向上集成，直到整个软件项目集成完毕为止。

3. 改进的自顶向下集成

尽管自顶向下集成和自底向上集成都有一定的优点，但是它们也有一些不可避免的缺点。

采用自顶向下集成方式可以直接使用主控模块来集成下层模块，软件开发团队无须编写集成、测试驱动程序。由于上层模块较早地参与集成，自顶向下集成方式能够较好地检测出上层模块在控制方面存在的问题。然而，自顶向下集成方式在模块集成过程中需要编写大量

桩模块，通过桩模块来模拟被调子模块的功能，因此自顶向下集成方式不容易发现底层模块中存在的错误和缺陷。如果在软件集成后期发现底层模块出现了错误，软件开发团队需要开展大量回归测试才能纠正、消除底层模块错误带来的影响。

相反，自底向上集成方式从系统结构图的最底层模块进行集成，容易发现底层模块中存在的问题。在模块集成初期，软件开发团队可以同时对多个功能簇进行并行集成和测试，最后再集成主控模块。但是，在自底向上集成方式中，主控模块中存在的错误或缺陷要到最后才能发现和解决。

为了克服自顶向下集成和自底向上集成中存在的缺点，有人提出将两种集成方式进行混合改进。也就是说，在软件模块集成过程中，软件开发团队基本上使用自顶向下集成。但是，在项目集成初期，软件开发团队可以先使用自底向上集成方式对系统结构图中的底层模块进行集成，减少自顶向下集成方式中撰写桩模块的工作量。

5.3 结构化测试

在结构化实现阶段，软件开发人员将根据结构化设计阶段的成果来实现各个软件模块，并按照特定的策略来集成软件模块，实现软件需求规格说明书中要求的内容。

那么在结构化方法中，软件开发团队如何保障各个软件模块以及整个软件系统的质量呢？其实，在结构化方法中，软件的开发过程同样伴随着软件测试。软件开发团队可以通过结构化测试来排除软件开发各个阶段中存在的错误，提高软件系统的质量。

通常而言，结构化测试包括对软件开发各个阶段的测试内容划分、结构化测试覆盖标准、测试用例设计和测试实施四个部分的内容。软件开发团队可以结合软件项目的实际情况来开展相关的测试内容。

5.3.1 结构化测试阶段

在结构化方法中，软件开发过程被分为结构化分析、结构化设计和结构化实现三个阶段。由于软件开发团队在不同阶段所需完成的工作内容和任务不同，各个阶段的测试内容和方式也不尽相同。

1. 结构化分析测试

在结构化分析阶段，软件开发团队的主要工作是对目标软件系统的数据处理过程、处理逻辑、系统状态、数据模型和数据字典等进行建模。因此，结构化分析阶段的测试内容主要包括以下几个方面。

1）检测数据处理流程是否完整

软件测试人员需要结合实际的业务处理流程来检测需求规格说明书中的数据流图是否完整，是否有遗漏的处理过程，各个数据处理之间交换的信息是否正确，数据在处理过程中是否存在丢失等。

2）测试处理对应的处理/加工逻辑说明是否正确

除了保证数据流图的完整性以外，对于数据流图中各个处理的内部细节测试也是结构化分析阶段测试的重点。软件测试人员可以结合实际的数据内容来测试各个处理，确保处理能够按照指定的业务规则产生正确的输出。

3）测试系统状态是否与实际要求一致

通过将完成的状态转换图与业务场景中的状态变迁进行比较，软件测试人员可以分析状