第3章 结构化设计

CHAPTER

传统的软件工程方法学采用结构化设计技术完成软件设计(概要设计和详细设计)工作。结构化设计技术的基本要点如下。

- 软件系统由层次化结构的模块构成。
- 模块是单入口和单出口的。
- 构造和联结模块的基本准则是模块独立。
- 用图来描述软件系统的结构,并且使软件结构与问题结构尽量一致。

3.1 软件设计的任务

3.1.1 概要设计

概要设计也称为总体设计或初步设计,这个设计阶段主要完成下述两项任务。

1. 方案设计

首先设想实现目标系统的各种可能的方案。然后,根据系统规模和目标,综合考虑技术、经济、操作等各种因素,从设想出的供选择的方案中选取若干个合理的方案。最后,综合分析、对比所选取的各种合理方案的利弊,从中选出一个最佳方案,并且制定这个最佳方案的详细实现计划。

2. 软件体系结构设计

所谓软件体系结构设计,就是确定软件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。设计出初步的软件结构之后,还应该从多方面进一步改进软件结构,以便得到更好的体系结构。

3.1.2 详细设计

详细设计阶段主要完成以下三项任务。

- 过程设计,即设计软件体系结构中所包含的每个模块的实现算法。
- 数据设计,即设计软件中所需要的数据结构。
- 接口设计,即设计软件内部各个模块之间、软件与协作系统之间以及软件与使用它的人之间的通信方式。

3.2 分析与设计的关系

系统分析的基本任务是定义用户所需要的软件系统,也就是回答系统必须"做什么"这个问题;系统设计的基本任务是设计实现目标系统的具体方案,也就是回答"怎样做"这个问题。虽然分析与设计的任务性质不同,但是二者之间有着非常密切的关系。

软件工程师必须依据用户对软件的需求来设计软件,因此,结构化分析的结果是进行结构化设计的最基本、最重要的输入信息。

体系结构设计的任务是,确定程序由哪些模块组成以及这些模块相互间的关系。在需求分析阶段画出的数据流图是进行体系结构设计的主要依据,为体系结构设计提供最基本的输入信息。

数据设计把需求分析阶段创建的信息模型转变成实现软件所需要的数据结构。在实体-联系图中定义的数据和数据之间的关系,以及数据字典中给出的详细的数据定义,共同为数据设计活动奠定坚实的基础。

接口设计的结果描述了软件内部、软件与协作系统之间以及软件与使用它的人之间的通信方式。接口意味着信息的流动(数据流或控制流),因此,数据流图提供了进行接口设计所需要的基本信息。

过程设计决定程序中包含的每个模块的实现算法,需求分析阶段画出的IPO图(表)为过程设计奠定了基础。

虽然需求分析为结构化设计提供了最基本、最重要的输入信息,但是,并不是说可以简单地把结构化分析的结果映射成结构化设计的结果。实际上,结构化设计过程综合了下述诸多因素:从以往开发类似软件的经验中获得的直觉和判断力,指导软件模型演化的一组原理(也称为准则)和启发规则,评价软件质量的一组标准,以及导出最终的设计结果的迭代过程。

软件工程师在软件设计过程中所作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。

软件设计之所以如此重要,是因为设计是软件开发过程中决定软件产品质量的关键阶段。设计为我们提供了可以进行质量评估的软件表示(即软件模型),设计是把用户需求准确地转变为最终的软件产品的唯一方法。软件设计是后续的一切软件开发和维护步骤的基础,如果不进行设计,就会冒构造出不稳定的软件系统的风险:稍做改动这样的系统就可能崩溃;这样的系统很难维护;这样的系统很难测试;直到软件工程过程的后期(例如,编码结束)才能评价这样的系统的质量,但是,这时才发现软件质量问题已经为时过晚了。

3.3　设 计 原 理

为了能获得高质量的设计结果，在软件设计过程中应该遵循下述原理(或准则)。

3.3.1　模块化与模块独立

模块化和模块独立是关系非常密切的两条设计原理。

1. 模块化

所谓模块就是由边界元素限定的相邻程序元素的序列，并且有一个标识符代表它。

模块化就是把程序划分成独立命名且可独立访问的模块。每个模块完成一个子功能，把全部模块集成起来构成一个整体，可以完成指定的功能，满足用户的需求。

模块化可以使一个复杂的大型程序能被人的智力所管理，是软件应该具备的最重要的属性。

事实上，每个程序都相应地有一个最适当的模块数目，可使软件系统的开发成本最小。

采用模块化原理可以使软件结构清晰，不仅容易设计也容易阅读和理解。因为程序错误通常局限在有关的模块及它们之间的接口中，所以模块化使软件容易测试和调试，因而有助于提高软件的可靠性。因为变动往往只涉及少数几个模块，所以模块化能够提高软件的可修改性。模块化也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较复杂的模块。

2. 模块独立

只有合理地划分和组织模块，才能获得模块化所带来的好处，极大地提高软件的质量。指导模块划分和组织最重要的原理就是模块独立。

开发具有独立功能而且和其他模块之间没有过多的相互作用的模块就可以做到模块独立。换句话说，希望这样设计软件结构，使得每个模块完成一个相对独立的特定子功能，并且和其他模块之间的关系很简单。

为什么模块的独立性很重要呢？主要有两条理由：第一，有效的模块化(即具有独立的模块)的软件比较容易开发出来。这是由于能够分割功能而且接口可以简化，当许多人分工合作开发同一个软件时，这个优点尤其重要。第二，独立的模块比较容易测试和维护。这是因为相对说来，修改设计和程序需要的工作量比较小，错误传播范围小，需要扩充功能时能够“插入”模块。总之，模块独立是软件设计的关键，而设计又是决定软件质量的关键环节。

3.3.2　抽象

抽象是人类在认识复杂现象、解决复杂问题的过程中使用的最强有力的思维工具。

在现实世界中,一定事物、状态或过程之间总会存在某些相似的方面(共性),把这些相似的方面集中和概括起来,暂时忽略它们之间的差异,这就是抽象。或者说抽象就是提取出事物的本质特性而暂时不考虑其他细节。

由于人类思维能力的限制,如果一次面临的因素太多,是不可能作出精确思维的。设计复杂系统的唯一有效的方法是用层次的方式分析和构造它。一个复杂的软件系统应该首先用一些高级的抽象概念来理解和构造,这些高级概念又可以用一些较低级的概念来理解和构造,如此进行下去,直至最低层的具体元素。

这种层次的思维和解题方式必须反映在程序结构中,每级抽象层次中的一个概念将以某种方式对应于程序的一组成分。

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;最后,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。

3.3.3 逐步求精

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 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。

事实上,可以把逐步求精看做是一项把一个时期内必须解决的种种问题按优先级排序的技术。它让软件工程师把精力集中在与当前开发阶段最相关的那些问题上,而忽略那些对整体解决方案来说是重要的,然而目前还不需要考虑的细节性问题,这些细节将留到以后再考虑。逐步求精技术确保每个问题都将被解决,而且每个问题都在适当的时候解决,但是,在任何时候一个人都不需要同时处理 7 个以上知识块。

在用逐步求精方法解决问题的过程中,问题的某个特定方面的重要性是随时间变化的。最初,问题的某个方面可能无关紧要、无须考虑,但是后来同样的问题会变得很重要,必须解决。因此,逐步求精方法能够确保每个问题都得到解决,并且在适当的时间解决,在任何时刻都不需要同时处理 7 个以上知识块。

求精实际上是细化过程。从在高抽象级别定义的功能陈述(或信息描述)开始。也就是说,该陈述仅仅概念性地描述了功能或信息,但是并没有提供功能的内部工作情况或信息的内部结构。求精要求设计者细化原始陈述,随着每个后续求精(细化)步骤的完成而提供越来越多的细节。

抽象与求精是一对互补的概念。抽象使得设计者能够说明过程和数据,同时却忽略低层细节。事实上,可以把抽象看做是一种通过忽略多余的细节,同时强调有关的细节,而实现逐步求精的方法。求精则帮助设计者在设计过程中揭示出低层细节。这两个概念都有助于设计者在设计演化过程中创造出完整的设计模型。

3.3.4　信息隐藏

信息隐藏原理指出，在设计软件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。

实际上，应该隐藏的不是有关模块的一切信息，而是模块的实现细节。

“隐藏”意味着可以通过定义一组独立的模块来实现有效的模块化，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。

使用信息隐藏原理设计软件模块有助于减少修改软件时所犯的错误。

3.3.5　局部化

所谓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。局部化与信息隐藏密切相关。显然，局部化有助于实现信息隐藏。

3.4　度量模块独立性的标准

模块的独立程度可以由两个定性标准来度量，这两个标准分别称为内聚和耦合。内聚衡量一个模块内部各个元素彼此结合的紧密程度；耦合衡量不同模块彼此间互相依赖（连接）的紧密程度。

3.4.1　内聚

内聚度量一个模块内的各个元素彼此结合的紧密程度，它是信息隐藏和局部化概念的自然扩展。

设计软件时应该力求做到高内聚（功能内聚和顺序内聚），通常中等程度的内聚（通信内聚和过程内聚）也是可以使用的，而且效果和高内聚相差不多；但是，低内聚（偶然内聚、逻辑内聚和时间内聚）效果很差，不要使用。

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松耦合。内聚和耦合都是进行模块化设计的有力工具，但是实践表明内聚更重要，应该把更多注意力集中到提高模块的内聚程度上。

3.4.2　耦合

耦合是对一个软件结构内不同模块之间互连程度的度量。耦合的强弱取决于模块间接口的复杂程度，进入或访问一个模块的点，以及通过接口的数据。

在软件设计中应该追求尽可能松散耦合的系统。模块间耦合松散，有助于提高系统的可理解性、可测试性、可靠性和可维护性。

模块之间典型的耦合有数据耦合、控制耦合、特征耦合、公共环境耦合和内容耦合。

应该采用下述的设计准则：尽量使用数据耦合，少用控制耦合和特征耦合，限制公共环境耦合的范围，完全不用内容耦合。

3.5 启发规则

总结长期以来开发软件所积累的丰富经验,得出了一些启发式规则。这些启发式规则在许多场合都能给软件工程师以有益的启示,往往能帮助工程师找到改进软件设计、提高软件质量的途径。下面是几条典型的启发式规则。

(1) 改进软件结构、提高模块独立性。设计出软件的初步结构以后,应该仔细审查分析这个结构,通过模块分解或合并,力求降低耦合、提高内聚。

(2) 模块规模应该适中。模块规模过大,则可理解程度很低;模块规模过小则开销大于有效操作。通过模块分解或合并调整模块规模时,不可降低模块独立性。

(3) 深度、宽度、扇出和扇入都应适当。

(4) 模块的作用域应该在控制域之内。

(5) 力争降低模块接口的复杂程度。接口复杂或与模块功能不一致,是紧耦合或低内聚的征兆,应该重新分析这个模块的独立性。

(6) 设计单入口、单出口的模块。这条启发式规则警告软件工程师不要使模块间出现内容耦合。

(7) 模块功能应该可以预测。模块功能应该能够预测(即只要输入的数据相同就产生同样的输出),但也不要使模块功能过分局限。

3.6 描绘软件结构的图形工具

1. 层次图和 HIPO 图

层次图用于描绘软件的层次结构。层次图中的一个矩形框代表一个模块,方框间的连线表示模块间的调用关系。层次图很适于在自顶向下设计软件的过程中使用。

HIPO 图是“层次图加输入/处理/输出图”的英文缩写。它用层次图描绘软件结构,和层次图中每个方框相对应,有一张 IPO 图(或表)描绘这个方框代表的模块的处理过程。

2. 结构图

结构图和层次图类似,也是描绘软件结构的图形工具。图中一个矩形框代表一个模块,框间连线表示模块间的调用关系。通常还用带注释的箭头描述在模块调用过程中传递的信息。

3.7 面向数据流的设计方法

面向数据流的设计方法的目标是给出设计软件结构的一个系统化的途径。这种设计方法定义了一些“映射”规则,利用这些映射可以把数据流图变换成软件结构。

3.7.1　数据流的类型

面向数据流的设计方法把数据流图映射成软件结构，数据流的类型决定了映射的方法。数据流有下述两种类型。

1. 变换流

如果信息沿输入通路进入系统，同时由外部形式变换成内部形式，进入系统的信息通过变换中心，经加工处理以后再沿输出通路变换成外部形式离开软件系统，则具有上述特征的数据流就称为变换流。

2. 事务流

原则上所有信息流都可以归类为变换流，但是，如果信息沿输入通路到达一个称为事务中心的处理 T，这个处理根据输入数据的类型在若干个候选的动作序列中选取出一个来执行，则这类数据流应该划为一类特殊的数据流，称为事务流。

3.7.2　设计步骤

面向数据流方法主要有下述几个设计步骤。

1. 复查基本系统模型

复查经结构化分析过程画出的基本系统模型，以确保系统的输入输出数据符合实际。

2. 复查并精化数据流图

认真复查需求分析阶段画出的数据流图，并在必要时加以精化。不仅要确保数据流图给出了正确的目标系统逻辑模型，而且应该使数据流图中的每个处理都代表一个规模适中、相对独立的子功能。

3. 确定数据流图具有变换特性还是事务特性

一般地说，一个系统中的所有信息流都可以认为是变换流，但是，当遇到有明显事务特性的信息流时，建议采用事务分析方法进行设计。在这一步，设计人员应该根据数据流图中占优势的属性确定数据流的全局特性。此外，还应该把具有和全局特性不同的特点的局部区域孤立出来，以后可以按照这些子数据流的特点精化根据全局特性得出的软件结构。

4. 确定数据流的边界

对于变换流来说，分析确定输入流和输出流的边界，从而孤立出变换中心。对于事务流来说，分析确定输入流的边界，从而孤立出事务中心。

5. 完成“第一级分解”

软件结构代表对控制的自顶向下的分配,所谓分解就是分配控制的过程,而第一级分解就是分配顶层控制。

对于变换流的情况,位于软件结构最顶层的总控模块协调下述三个从属模块的控制功能。

- 输入信息处理控制模块,此模块协调对所有输入数据的接收。
- 变换中心控制模块,此模块管理对内部形式的数据的所有操作。
- 输出信息处理控制模块,此模块协调输出信息的产生过程。

对于事务流的情况,位于软件结构最顶层的总控模块管理下属的接收分支和发送分支的工作。接收分支由输入流映射而成。发送分支的顶层是一个调度模块,它根据输入数据的类型调用相应的活动分支。

机械地遵循上述映射规则很可能会得出一些不必要的控制模块,如果它们确实用处不大,那么可以而且应该把它们合并。反之,如果控制模块功能过分复杂,则应该把它分解为两个或多个控制模块,或者适当地增加中间层次的控制模块。

6. 完成“第二级分解”

所谓第二级分解就是把数据流图中的每个处理映射成软件结构中一个适当的模块。

对于变换流来说,完成第二级分解的方法是从变换中心的边界开始沿着输入通路向外移动,把输入通路中每个处理依次映射成软件结构中“输入信息处理控制模块”控制下的一个低层模块;然后从变换中心的边界开始沿着输出通路向外移动,把输出通路中每个处理依次映射成直接或间接受“输出信息处理控制模块”控制的一个低层模块;最后把变换中心内的每个处理映射成受“变换中心控制模块”控制的一个模块。

对于事务流来说,映射出接收分支结构的方法和变换分析映射出输入结构的方法很相似。发送分支的结构包含一个调度模块,它控制下层的所有活动模块,然后把数据流图中的每个活动流通路映射成与它的流特征相对应的结构。

设计一个大型系统时,通常把变换分析和事务分析应用到同一个数据流图的不同部分,由此得到的子结构形成“构件”,可以使用它们构造完整的软件结构。

7. 优化

对第一次分割得到的软件结构,总可以根据模块独立原理和启发式设计规则进行优化。为了产生合理的分解,得到尽可能高的内聚、尽可能松散的耦合,最重要的是,为了得到一个易于实现、易于测试和易于维护的软件结构,应该对初步分割得到的模块进行再分解或合并。

3.8 人机界面设计

人机界面设计是接口设计的一个重要的组成部分。对于交互式系统来说,人机界面设计和数据设计、体系结构设计、过程设计一样重要。

人机界面的设计质量直接影响用户对软件产品的评价，从而影响软件产品的竞争力和使用寿命，因此，必须对人机界面设计给予足够重视。

由于对人机界面的评价，在很大程度上由人的主观因素决定，因此，使用基于原型的系统化的设计策略是成功地设计人机界面的关键。

3.8.1　应该考虑的设计问题

在设计用户界面的过程中，设计者几乎总会遇到下述 4 个问题：系统响应时间、用户帮助设施、出错信息处理和命令交互。最好在设计人机界面的初期就把这些问题作为重要的设计问题来考虑，这时修改比较容易，代价也低。下面讨论这 4 个问题。

1. 系统响应时间

系统响应时间是许多交互式系统用户经常抱怨的问题。一般说来，系统响应时间指从用户完成某个控制动作（例如，按回车键或点击鼠标）到软件给出预期的响应（输出或做预期的动作）之间的这段时间。

系统响应时间有两个重要属性，分别是长度和易变性。系统响应时间的长短应该适当，而且应该尽量稳定。

2. 用户帮助设施

常见的帮助设施有集成的和附加的两类。集成的帮助设施从一开始就设计在软件里面，通常它对用户的工作内容是敏感的，因此用户可以从与刚刚完成的操作有关的主题中选择一个请求帮助。显然，这可以缩短用户获得帮助所需的时间，并能增加界面的友好性。附加的帮助设施是在系统建成后再添加到软件中的，在多数情况下，它实际上是一种查询能力有限的联机用户手册。人们普遍认为，集成的帮助设施优于附加的帮助设施。

3. 出错信息处理

一般说来，交互式系统给出的出错信息或警告信息，应该具有下述属性。

- 信息应该以用户可以理解的术语描述问题。
- 信息应该提供有助于从错误中恢复的建设性意见。
- 信息应该指出错误可能导致哪些负面后果（例如，破坏数据文件），以便用户检查是否出现了这些问题，并在确实出现问题时予以改正。
- 信息应该伴随着听觉上或视觉上的提示，也就是说，在显示信息时应该同时发出警告声，或者信息用闪烁方式显示，或者信息用明显表示出错的颜色显示。
- 信息不能带有指责色彩，也就是说，不能责怪用户。

当确实出现了问题的时候，有效的出错信息能够提高交互式系统的质量，减少用户的挫折感。

4. 命令交互

命令行曾经是用户和系统软件交互的最常用方式，而且也曾经广泛地用于各种应用

软件中。现在,面向窗口的、点击和拾取方式的界面已经减少了用户对命令行的依赖,但是,许多高级用户仍然偏爱面向命令的交互方式。在多数情况下,用户既可以从菜单中选择软件功能,也可以通过键盘命令序列调用软件功能。

3.8.2 人机界面设计过程

用户界面设计是一个迭代的过程。也就是说,通常先创建设计模型,再用原型实现这个设计模型,并由用户试用和评估,然后根据用户的意见进行修改,直至满意为止。

3.8.3 人机界面设计指南

1. 一般交互指南

一般交互指南涉及信息显示、数据输入和系统的整体控制,因此,这些指南是全局性的,忽略它们将冒较大风险。一般交互指南如下。

- 保持一致性。
- 提供有意义的反馈。
- 在执行有较大破坏性的动作之前要求用户确认。
- 允许取消绝大多数操作。
- 减少在两次操作之间必须记忆的信息量。
- 提高对话、鼠标移动和思考的效率。
- 允许用户犯错误。
- 按功能对动作分类,并据此设计屏幕布局。
- 提供对工作内容敏感的帮助设施。
- 用简单动词或动词短语作为命令名。

2. 信息显示指南

- 只显示与当前工作内容有关的信息。
- 用便于用户迅速地吸取信息的方式来显示数据。
- 使用一致的标记、标准的缩写和可预知的颜色。
- 允许用户保持可视化的语境。
- 产生有意义的出错信息。
- 使用大小写、缩进和文本分组来帮助理解。
- 使用窗口分隔不同类型的信息。
- 使用“模拟”方式显示信息。
- 高效率地使用显示屏。

3. 数据输入指南

- 尽量减少用户的输入动作。
- 保持信息显示和数据输入之间的一致性。

- 允许用户自定义输入。
- 交互应该是灵活的，可调整成用户喜欢的输入方式。
- 使得在当前动作语境中不适用的命令不起作用。
- 让用户控制交互流。
- 对所有输入动作都提供帮助。
- 消除冗余的输入。

3.9　过 程 设 计

过程设计应该在数据设计、体系结构设计和接口设计完成之后进行，它是详细设计阶段应该完成的主要任务。

过程设计的任务还不是具体地编写程序，而是要设计出程序的“蓝图”，以后程序员将根据这个蓝图写出实际的程序代码。因此，过程设计的结果基本上决定了最终程序代码的质量。考虑程序代码的质量时必须注意，程序的“读者”有两个，那就是计算机和人。在软件的生命周期中，设计测试方案、诊断程序错误、修改和改进程序等都必须首先读懂程序。实际上对于长期使用的软件系统而言，人读程序的时间可能比写程序的时间要长得多。因此，衡量程序的质量不仅要看它的逻辑是否正确，性能是否满足要求，更主要的是要看它是否容易阅读和理解。过程设计的目标不仅仅是逻辑上正确地实现每个模块的功能，更重要的是设计出的处理过程应该尽可能简明易懂。结构程序设计技术是实现上述目标的关键技术，因此是过程设计的逻辑基础。

狭义的结构程序设计定义为：如果一个程序的代码块仅仅通过顺序、选择和循环这三种控制结构进行连接，并且每个代码块只有一个入口和一个出口，则称这个程序是结构化的。

广义的结构程序设计定义为：结构程序设计是尽可能少用 GO TO 语句的程序设计方法。最好仅在检测出错误时才使用 GO TO 语句，而且最好应该使用前向 GO TO 语句。

也可以把结构程序设计技术具体地划分为下述三种类型：如果只允许使用顺序、IF-THEN-ELSE 型分支和 DO-WHILE 型循环这三种基本控制结构，则称为经典的结构程序设计；如果除了上述三种基本控制结构之外，还允许使用 DO-CASE 型多分支结构和 DO-UNTIL 型循环结构，则称为扩展的结构程序设计；如果再加上允许使用 LEAVE（或 BREAK）结构，则称为修正的结构程序设计。

3.10　过程设计的工具

描述程序处理过程的工具称为过程设计的工具，它们可以分为图形、表格和语言三类。不论是哪类工具，对它们的基本要求都是能提供对设计的无歧义的描述，也就是应该能指明控制流程、处理功能、数据组织以及其他方面的实现细节，从而在编码阶段能把对设计的描述直接翻译成程序代码。此外，这类工具应该尽可能的形象直观，易学、易懂。

1. 程序流程图

程序流程图又称为程序框图,它是历史最悠久、使用最广泛的描述过程设计的方法,然而它也是用得最混乱的一种方法。

2. 盒图

出于要有一种不允许违背结构程序设计精神的图形工具的考虑,Nassi 和 Shneiderman 发明了盒图,又称为 N-S 图。

盒图没有箭头,因此不能够随意转移控制。坚持使用盒图作为过程设计的工具,可以使程序员逐步养成用结构化的方式思考问题、解决问题的习惯。

3. PAD 图

PAD 是问题分析图(problem analysis diagram)的英文缩写,它用二维树状结构的图来表示程序的控制流,将这种图翻译成程序代码比较容易。

4. 判定表

当算法中包含多重嵌套的条件选择时,用程序流程图、盒图、PAD 图或后面即将介绍的过程设计语言(PDL)都不易清楚地描述。然而,判定表却能够清晰地表示复杂的条件组合与应做的动作之间的对应关系。

一张判定表由 4 部分组成,左上部列出所有条件,左下部是所有可能做的动作,右上部是表示各种条件组合的一个矩阵,右下部是和每种条件组合相对应的动作。判定表右半部的每一列实质上是一条规则,规定了与特定的条件组合相对应的动作。

5. 判定树

判定表虽然能清晰地表示复杂的条件组合与应做的动作之间的对应关系,但其含义不是一眼就能看出来的,初次接触这种工具的人要理解它需要有一个简短的学习过程。

判定树是判定表的变种,它能清晰地表示复杂的条件组合与应做的动作之间的对应关系。判定树的优点在于,它的形式简单到不需任何说明,一眼就可以看出其含义,因此易于掌握和使用。多年来判定树一直受到人们的重视,是一种比较常用的系统分析和设计的工具。

6. 过程设计语言(PDL)

PDL 也称为伪码,这是一个笼统的名称,现在有许多种不同的过程设计语言在使用。它是用正文形式表示数据和处理过程的设计工具。

PDL 具有严格的关键字外部语法,用于定义控制结构和数据结构;另外,PDL 表示实际操作和条件的内部语法通常又是灵活自由的,以便可以适应各种工程项目的需要。因此,一般说来 PDL 是一种"混杂"语言,它使用一种语言(通常是某种自然语言)的词汇,同

时却使用另一种语言(某种结构化的程序设计语言)的语法。

3.11　面向数据结构的设计方法

计算机软件本质上是信息处理系统,因此,可以根据软件所处理信息的特征来设计软件。前面曾经复习了面向数据流的设计方法,也就是根据数据流确定软件结构的方法,本节将复习面向数据结构的设计方法,也就是根据数据结构设计程序处理过程的方法。

在许多应用领域中信息都有清楚的层次结构,输入数据、内部存储的信息(数据库或文件)以及输出数据都可能有独特的结构。数据结构既影响程序的结构又影响程序的处理过程,重复出现的数据通常由具有循环控制结构的程序来处理,选择数据(即可能出现也可能不出现的信息)要用带有分支控制结构的程序来处理。层次的数据组织通常和使用这些数据程序的层次结构十分相似。

面向数据结构设计方法的最终目标是得出对程序处理过程的描述。这种设计方法并不明显地使用软件结构的概念,模块是设计过程的副产品,对于模块独立原理也没有给予应有的重视。因此,这种方法最适合于在详细设计阶段使用,也就是说,在完成了软件结构设计之后,可以使用面向数据结构的方法来设计每个模块的处理过程。

Jackson 结构程序设计方法是典型的面向数据结构的设计方法,它由以下 5 个步骤组成。

第 1 步,分析并确定输入数据和输出数据的逻辑结构,并用 Jackson 图描绘这些数据结构。

第 2 步,找出输入数据结构和输出数据结构中有对应关系的数据单元。所谓有对应关系是指有直接的因果关系,在程序中可以同时处理的数据单元(对于重复出现的数据单元必须重复的次序和次数都相同才可能有对应关系)。

第 3 步,用下述 3 条规则从描绘数据结构的 Jackson 图导出描绘程序结构的 Jackson 图。

规则 1,为每对有对应关系的数据单元,按照它们在数据结构图中的层次在程序结构图的相应层次画一个处理框(注意,如果这对数据单元在输入数据结构和输出数据结构中所处的层次不同,则和它们对应的处理框在程序结构图中所处的层次与它们之中在数据结构图中层次低的那个对应)。

规则 2,根据输入数据结构中剩余的每个数据单元所处的层次,在程序结构图的相应层次分别为它们画上对应的处理框。

规则 3,根据输出数据结构中剩余的每个数据单元所处的层次,在程序结构图的相应层次分别为它们画上对应的处理框。

总之,描绘程序结构的 Jackson 图应该综合输入数据结构和输出数据结构的层次关系而导出来。在导出程序结构图的过程中,由于改进的 Jackson 图规定在构成顺序结构的元素中不能有重复出现或选择出现的元素,因此可能需要增加中间层次的处理框。

第 4 步,列出所有操作和条件(包括分支条件和循环结束条件),并且把它们分配到程序结构图的适当位置。

第5步,用伪码表示程序。

3.12 程序复杂程度的定量度量

定量度量程序复杂程度的价值在于:把程序的复杂程度乘以适当常数即可估算出软件中错误的数量以及开发该软件需要用的工作量,因此,定量度量的结果可以用来比较两个不同的设计或两个不同算法的优劣;程序的定量复杂度可以作为模块规模的精确限度。

3.12.1 McCabe 方法

1. 流图

McCabe方法根据程序控制流的复杂程度定量度量程序复杂程度,这样度量出的结果称为程序的环形复杂度。

为了突出描述程序的控制流,人们常常使用流图(也称为程序图)。流图实质上是"退化了的"程序流程图,它仅仅描绘程序的控制流,完全不表现对数据的具体操作以及分支或循环的具体条件。

在流图中用圆表示结点,一个圆代表一条或多条语句。程序流程图中一个顺序执行的处理框序列和一个菱形判定框,可以映射成流图中的一个结点。流图中的箭头线称为边,代表控制流。在流图中一条边必须终止于一个结点,即使这个结点并不代表可执行的语句(相当于一个空语句)。由边和结点围成的面积称为区域,当计算区域数时应该包括图外部没被围起来的那个区域。

当过程设计的结果中包含复合条件时,应该把复合条件分解为若干个简单条件,每个简单条件对应流图中一个结点。

2. 计算环形复杂度的方法

有了描绘程序控制流的流图之后,可以用下述3种方法之一来计算环形复杂度。

(1) 环形复杂度等于流图中的区域数。

(2) 流图 G 的环形复杂度 $V(G)=E-N+2$,其中 E 是流图中边的条数,N 是结点数。

(3) 流图 G 的环形复杂度 $V(G)=P+1$,其中 P 是程序中判断的数目。在源代码中,IF语句、WHILE循环或FOR循环都相当于1个判断,而CASE语句或其他多分支语句相当的判断数等于可能的分支数减1。

3.12.2 Halstead 方法

Halstead方法根据程序中运算符和操作数的总数来度量程序的复杂程度。

令 N_1 为程序中运算符出现的总次数,N_2 为操作数出现的总次数,程序长度 N 定义为

$$N = N_1 + N_2$$

若详细设计结果中使用的不同运算符(包括关键字)的个数为 n_1,不同操作数(变量和常数)的个数为 n_2,则 Halstead 方法预测程序长度为

$$H = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

实践表明,预测的长度 H 比较接近实际长度 N。

习　　题

1. 用逐步求精方法解决下述更新顺序主文件的问题。

美国某杂志社需要一个软件,以更新存有该杂志订户姓名、地址等数据的顺序主文件。共有插入、修改和删除等 3 种类型的事务,分别对应于事务代码 1、2 和 3。也就是说,事务类型如下:

类型 1: INSERT(插入一个新订户到主文件中);

类型 2: MODIFY(修改一个已有的订户记录);

类型 3: DELETE(删除一个已有的订户记录)。

事务是按订户名字的字母顺序排序的。如果对一个订户既有修改事务又有删除事务,则已对那个订户的事务排好次序了,以便使修改发生在删除之前。

2. 分析图 3.1 所示的层次图,确定每个模块的内聚类型。

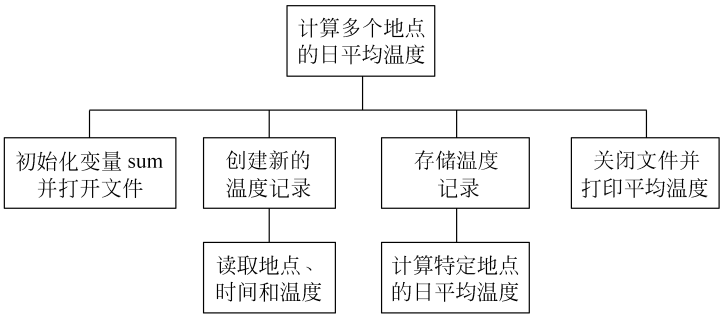


图 3.1　计算多地点日平均温度的程序

3. 分析图 3.2,确定模块之间的耦合类型。

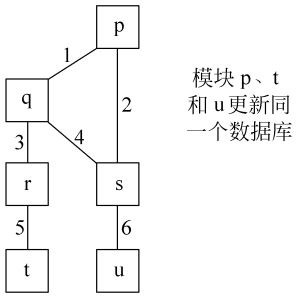


图 3.2　一个程序的模块互连图

在图3.2中已经给模块之间的接口编了号码,表3.1描述了模块间的接口。

表3.1 模块接口描述

编　　号	输　　入	输　　出
1	飞机类型	状态标志
2	飞机零件清单	
3	功能代码	
4	飞机零件清单	
5	零件编号	零件制造商
6	零件编号	零件名称

4. 用面向数据流方法设计本书第2章第11题所述的工资支付系统的软件结构。

5. 用3种方法计算图3.3所示流图的环形复杂度。

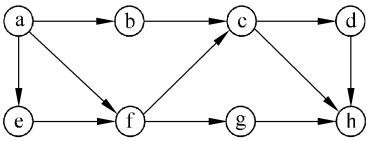


图3.3 一个程序的流图

6. 下面列出的代码用重复执行的加法来计算两个正整数 X 和 Y 的乘积,请用Halstead方法预测程序长度,并把预测出的长度与实际长度相比较。

```
Z=0;
while X>0
    Z=Z+Y;
    X=X-1;
end_while;
print(Z);
```

7. 图3.4是用程序流程图描绘的程序算法,请把它改画为等价的盒图。

8. 某交易所规定给经纪人的手续费计算方法如下:总手续费等于基本手续费加上与交易中的每股价格和股数有关的附加手续费。如果交易总金额少于1000元,则基本手续费为交易金额的8.4%;如果交易总金额在1000～10 000元之间,则基本手续费为交易金额的5%,再加34元;如果交易总金额超过10 000元,则基本手续费为交易金额的4%加上134元。当每股售价低于14元时,附加手续费为基本手续费的5%,除非买进、卖出的股数不是100的倍数,在这种情况下附加手续费为基本手续费的9%。当每股售价在14～25元之间时,附加手续费为基本手续费的2%,除非交易的股数不是100的倍数,在这种情况下附加手续费为基本手续费的6%。当每股售价超过25元时,如果交易的股数零散(即不是100的倍数),则附加手续费为基本手续费的4%,否则附加手续费为基本手续费的1%。

要求:

(1) 用判定表表示手续费的计算方法。

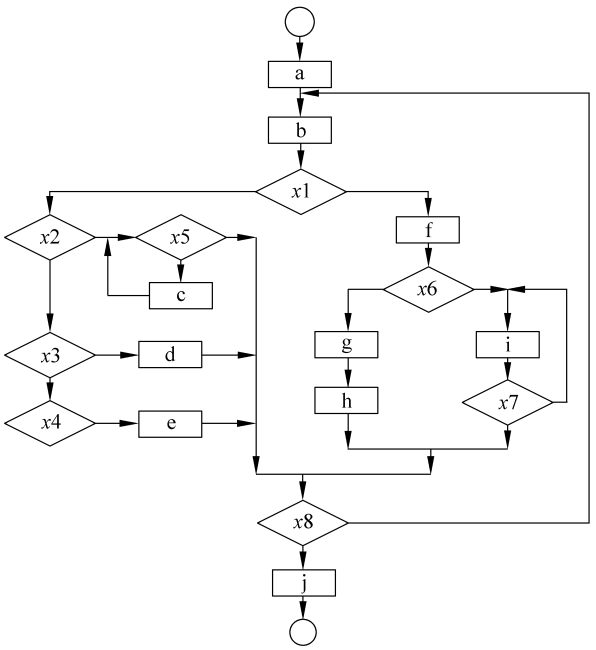


图 3.4　程序流程图

(2) 用判定树表示手续费的计算方法。

9. 画出下列伪码程序的程序流程图和盒图。

```
START
IF P THEN
    WHILE q DO
          f
    END DO
  ELSE
    BLOCK
          g
          n
    END BLOCK
END IF
STOP
```

10. 图 3.5 给出的程序流程图代表一个非结构化的程序,请问:

(1) 为什么说它是非结构化的?

(2) 设计一个与它等价的结构化程序。

(3) 在(2)题的设计中你使用附加的标志变量 flag 了吗? 若没用,请再设计一个使用 flag 的程序;若用了,请再设计一个不用 flag 的程序。

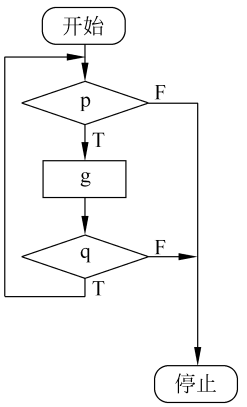


图 3.5　一个非结构化程序

11. 研究下面给出的伪码程序,要求:

(1) 画出它的程序流程图。

(2) 它是结构化的还是非结构化的?说明你的理由。

(3) 若是非结构化的,则:

ⓐ 把它改造成仅用三种控制结构的结构化程序;

ⓑ 写出这个结构化设计的伪码;

ⓒ 用盒图表示这个结构化程序。

(4) 找出并改正程序逻辑中的错误。

```
COMMENT: PROGRAM SEARCHES FOR FIRST N REFERENCES
         TO A TOPIC IN AN INFORMATION RETRIEVAL
         SYSTEM WITH T TOTAL ENTRIES
         INTPUT N
         INPUT KEYWORD(S) FOR TOPIC
         I=0
         MATCH=0
         DO WHILE I≤T
             I=I+1
             IF WORD=KEYWORD
               THEN MATCH=MATCH+1
                    STORE IN BUFFER
             END
             IF MATCH=N
               THEN GOTO OUTPUT
             END
         END
         IF N=0
             THEN PRINT "NO MATCH"
OUTPUT: ELSE CALL SUBROUTINE TO PRINT BUFFER
          INFORMATION
       END
```

12. 研究图3.6给出的程序流程图,要求:

(1) 写出它的伪码表示。

(2) 设计一个等价的结构化程序。

(3) 用另一种方法重做第(2)题。

13. 从伪码转变为程序流程图或从程序流程图转变为伪码是否是唯一的?请说明理由。

14. 用Ashcroft_Manna技术可以将非结构化的程序转换为结构化程序,图3.7是一个转换的例子。

(1) 你能否从这个例子总结出Ashcroft_Manna技术的一些基本方法?

(2) 进一步简化图3.7(b)给出的结构化设计。

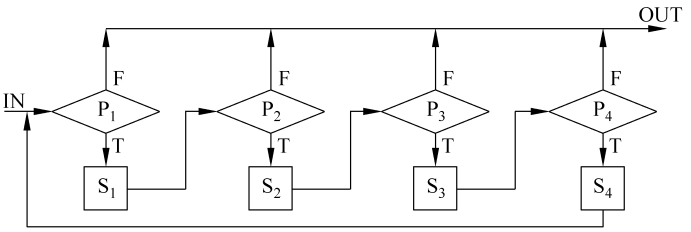
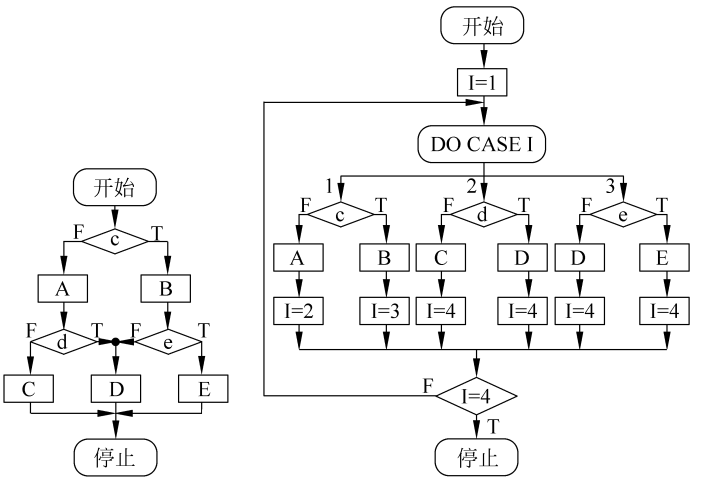


图 3.6　一个非结构化设计



(a) 非结构化设计　　(b) 转换成的等价的结构化设计

图 3.7　用 Ashcroft_Manna 技术的例子

15. 用 Jackson 图描绘下述的一列火车的构成：

一列火车最多有两个火车头。只有一个火车头时则位于列车最前面，若还有第二个火车头时，则第二个火车头位于列车最后面。火车头既可能是内燃机车也可能是电气机车。车厢分为硬座车厢、硬卧车厢和软卧车厢 3 种。硬座车厢在所有车厢的前面，软卧车厢在所有车厢的后面。此外，在硬卧车厢和软卧车厢之间还有一节餐车。

习题解答

1. 答：解决任何问题之前都必须首先理解问题，对问题理解得越深入，解决起来也就越容易。为了获得对顺序主文件更新问题的直观、具体的认识，首先设想一个典型的主文件（称为旧的主文件）、一个事务文件和更新后得到的新的主文件及异常情况报告，如图 3.8 所示。

为了简单起见，在图 3.8 中忽略了主文件和事务文件中所包含的订户地址信息。

从图 3.8 可以看出，更新顺序主文件系统有下述两个输入文件：

① 旧的主文件（由包含订户姓名、地址信息的记录组成）。

② 事务文件。

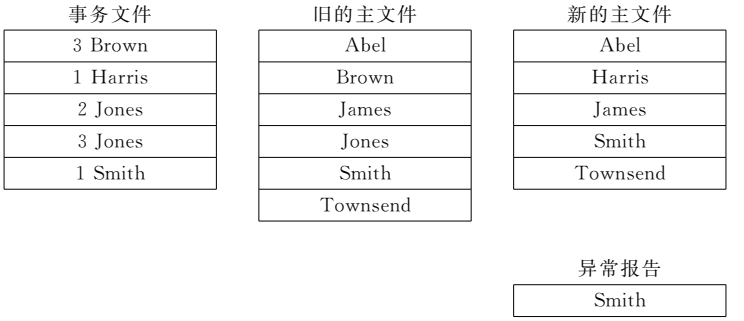


图 3.8 典型的顺序主文件更新问题

系统还有三个输出文件：

① 新的主文件。

② 异常报告。

③ 摘要和工作结束信息。

图 3.9 描绘了设想的顺序主文件更新系统的概貌。

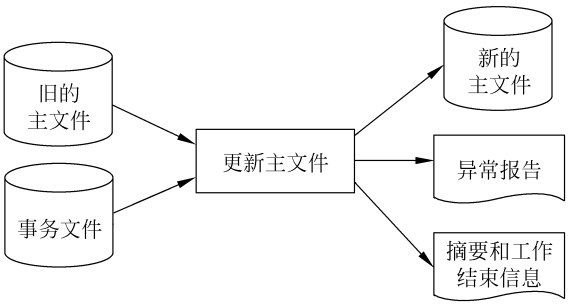


图 3.9 顺序主文件更新系统概貌

然后用逐步求精方法设计图 3.9 中关键的黑盒子“更新主文件”的实现算法。逐步求精方法实质上是“自顶向下”的设计方法，它通过不断分层细化解决问题的算法来设计软件。它不像“各个击破”技术那样把整个问题分解为若干个重要程序相同的子问题。在用逐步求精方法设计软件的过程中，软件的某个特定方面的重要性在一次又一次的求精中是变化的。最初，某个问题可能无关紧要，但后来同样的问题会变得相当重要。换句话说，可以把逐步求精方法看做是建立某个阶段内需要解决的各种问题的优先级的一种技术。它能确保每个问题都在恰当的时间得到解决，而且在任何时刻都不需要同时解决 7 个以上问题。使用逐步求精方法设计软件的难点在于，在当前的求精步骤中确定哪些是必须处理的重要事项，哪些事项应该推迟到后面的求精步骤中去处理。

作为对“更新主文件”的第一步求精，把它分解为 3 个处理框，分别称为输入、处理和输出，如图 3.10 所示。

在这个设计步骤中假设，当“处理”需要一个记录时，能够在那个时候输入正确的记录。同样，也能够在当时把正确的记录写入到正确的文件中。也就是说，在把逐步求精方