第四章 面向过程的软件设计方法

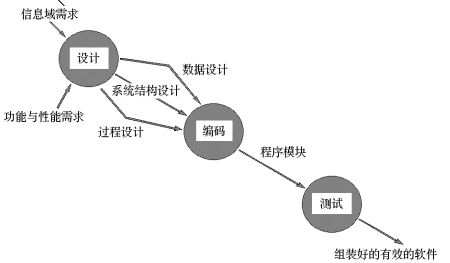
- 软件设计的目标和任务
- 软件设计基础
- 模块独立性
- 结构化设计方法
- 数据设计和文件设计
- 过程设计

软件设计的目标和任务

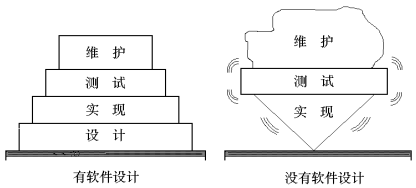
- 根据用信息域表示的软件需求，以及功能和性能需求，进行
  - 数据设计
  - 系统结构设计
  - 过程设计。

- 数据设计侧重于数据结构的定义。
- 系统结构设计定义软件系统各主要成份之间的关系。
- 过程设计则是把结构成份转换成软件的过程性描述。在编码步骤，根据这种过程性描述，生成源程序代码，然后通过测试最终得到完整有效的软件。

开发阶段的信息流

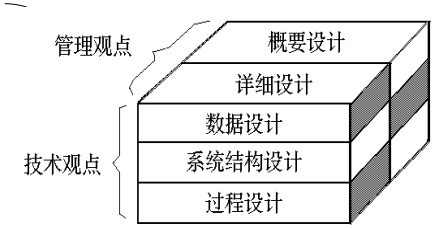


- 软件设计是后续开发步骤及软件维护工作的基础。如果没有设计，只能建立一个不稳定的系统结构



软件设计任务

- 从工程管理的角度来看，软件设计分两步完成。
  - 概要设计，将软件需求转化为数据结构和软件的系统结构。
  - 详细设计，即过程设计。通过对结构表示进行细化，得到软件的详细的数据结构和算法。



软件设计过程

1. 制定规范

- 在进入软件开发阶段之初，首先应为软件开发组制定在设计时应该共同遵守的标准，以便协调组内各成员的工作。包括：

- 阅读和理解软件需求说明书，确认用户要求能否实现，明确实现的条件，从而确定设计的目标，以及它们的优先顺序
- 根据目标确定最合适的设计方法
- 规定设计文档的编制标准
- 规定编码的信息形式，与硬件，操作系统的接口规约，命名规则

2. 软件系统结构的总体设计

- 基于功能层次结构建立系统。
  - 采用某种设计方法，将系统按功能划分成模块的层次结构
  - 确定每个模块的功能
  - 建立与已确定的软件需求的对应关系
  - 确定模块间的调用关系
  - 确定模块间的接口
  - 评估模块划分的质量

3. 处理方式设计

- 确定为实现系统的功能需求所必需的算法，评估算法的性能
- 确定为满足系统的性能需求所必需的算法和模块间的控制方式
  - 周转时间
  - 响应时间
  - 吞吐量
  - 精度
- 确定外部信号的接收发送形式

4. 数据结构设计

- 确定软件涉及的文件系统的结构以及数据库的模式、子模式，进行数据完整性和安全性的设计
- 确定输入，输出文件的详细的数据结构
- 结合算法设计，确定算法所必需的逻辑数据结构及其操作
- 确定对逻辑数据结构所必需的那些操作的程序模块(软件包)

- 限制和确定各个数据设计决策的影响范围
- 若需要与操作系统或调度程序接口所必须的控制表等数据时，确定其详细的数据结构和使用规则
- 数据的保护性设计
  - 防卫性设计：在软件设计中就插入自动检错，报错和纠错的功能

- 一致性设计：
  - 保证软件运行过程中所使用的数据的类型和取值范围不变
  - 在并发处理过程中使用封锁和解除封锁机制保持数据不被破坏
- 冗余性设计：针对同一问题，由两个开发者采用不同的程序设计风格不同的算法设计软件，当两者运行结果之差不在允许范围内时，利用检错系统予以纠正，或使用表决技术决定一个正确结果。

5. 可靠性设计

- 可靠性设计也叫做质量设计
- 在运行过程中，为了适应环境的变化和用户新的要求，需经常对软件进行改造和修正。在软件开发的一开始就要确定软件可靠性和其它质量指标，考虑相应措施，以使得软件易于修改和易于维护

6. 编写概要设计阶段的文档

- 概要设计阶段完成时应编写以下文档：
  - 概要设计说明书
  - 数据库设计说明书
  - 用户手册
  - 制定初步的测试计划

7. 概要设计评审

- 可追溯性：确认该设计是否复盖了所有已确定的软件需求，软件每一成份是否可追溯到某一项需求
- 接口：确认该软件的内部接口与外部接口是否已经明确定义。模块是否满足高内聚和低耦合的要求。模块作用范围是否在其控制范围之内
- 风险：确认该设计在现有技术条件下和预算范围内是否能按时实现

- 实用性：确认该设计对于需求的解决方案是否实用
- 技术清晰度：确认该设计是否以一种易于翻译成代码的形式表达
- 可维护性：确认该设计是否考虑了方便未来的维护
- 质量：确认该设计是否表现出良好的质量特征

- 各种选择方案：看是否考虑过其它方案，比较各种选择方案的标准是什么
- 限制：评估对该软件的限制是否现实，是否与需求一致
- 其它具体问题：对于文档、可测试性、设计过程..等进行评估

详细设计

- 在详细设计过程中，需要完成的工作是：
  1. 确定软件各个组成部分内的算法以及各部分的内部数据组织
  2. 选定某种过程的表达形式来描述各种算法。
  3. 进行详细设计的评审

软件设计基础

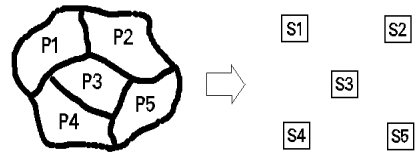
- 自顶向下，逐步细化
- 软件结构
- 程序结构
- 结构图
- 模块化
- 抽象化
- 信息隐蔽

自顶向下，逐步细化

- 将软件的体系结构按自顶向下方式，对各个层次的过程细节和数据细节逐层细化，直到用程序设计语言的语句能够实现为止，从而最后确立整个的体系结构。

软件结构

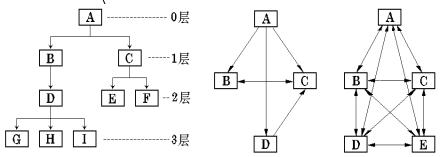
- 软件结构包括两部分。程序的模块结构和数据的结构
- 软件的体系结构通过一个划分过程来完成。该划分过程从需求分析确立的目标系统的模型出发，对整个问题进行分割，使其每个部分用一个或几个软件成份加以解决，整个问题就解决了



需要通过软件解决的"问题"　　软件的"解决方案"

程序结构

- 程序结构表明了程序各个部件(模块)的组织情况，是软件的过程表示。

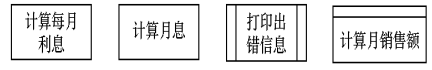


(a) 树状结构　(b) 网状结构　(c) 网状结构

结构图

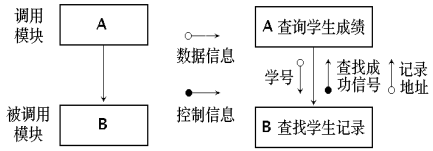
- 结构图反映程序中模块之间的层次调用关系和联系：它以特定的符号表示模块、模块间的调用关系和模块间信息的传递

① 模块：模块用矩形框表示，并用模块的名字标记它。



以功能做模块名　以功能的缩写做模块名　已定义模块　子程序(或过程)

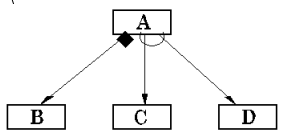
② 模块的调用关系和接口：模块之间用单向箭头联结，箭头从调用模块指向被调用模块，表示调用模块调用了被调用模块。



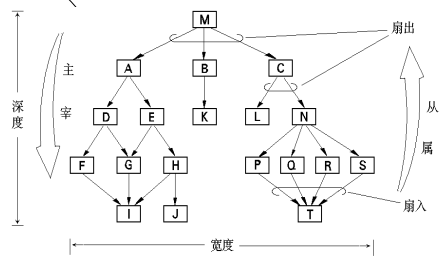
(a) 模块调用关系　(b) 模块间接口的表示

③ 模块间的信息传递：当一个模块调用另一个模块时，调用模块把数据或控制信息传送给被调用模块，以使被调用模块能够运行。而被调用模块在执行过程中又把它产生的数据或控制信息回送给调用模块

④ 在模块A的箭头尾部标以一个菱形符号，表示模块A有条件地调用另一个模块B。当一个在调用箭头尾部标以一个弧形符号，表示模块A反复调用模块C和模块D。



程序的系统结构图



模块化

- 软件系统的模块化是指整个软件被划分成若干单独命名和可编址的部分，称之为模块。这些模块可以被组装起来以满足整个问题的需求。
- 把问题／子问题的分解与软件开发中的系统／子系统或系统／模块对应起来，就能够把一个大而复杂的软件系统划分成易于理解的比较单纯的模块结构。

抽象化

- 软件系统进行模块设计时，可有不同的抽象层次。
- 在最高的抽象层次上，可以使用问题所处环境的语言概括地描述问题的解法。
- 在较低的抽象层次上，则采用过程化的方法。

(1) 过程的抽象

在软件工程中，从系统定义到实现，每进展一步都可以看做是对软件解决方法的抽象化过程的一次细化。

❒ 在软件需求分析阶段，用“问题所处环境的为大家所熟悉的术语”来描述软件的解决方法。

❒ 在从概要设计到详细设计的过程中，抽象化的层次逐次降低。当产生源程序时到达最低抽象层次。

例说开发一个CAD软件时的三种抽象层次

- 抽象层次Ⅰ. 用问题所处环境的术语来描述这个软件：
  该软件包括一个计算机绘图界面，向绘图员显示图形，以及一个数字化仪界面，用以代替绘图板和丁字尺。所有直线、折线、矩形、圆及曲线的描画、所有的几何计算、所有的剖面图和辅助视图都可以用这个CAD软件实现……。

- 抽象层次Ⅱ. 任务需求的描述。

```
CAD SOFTWARE TASKS
    user interaction task;
    2-D drawing creation task;
    graphics display task;
    drawing file management task;
end.
```

在这个抽象层次上，未给出“怎样做”的信息，不能直接实现。

- 抽象层次Ⅲ. 程序过程表示。以2-D (二维)绘图生成任务为例:

```
PROCEDURE：2-D drawing creation
REPEAT UNTIL (drawing creation task terminates)
    DO WHILE (digitizer interaction occurs)
    digitizer interface task;
        DETERMINE drawing request CASE;
            line：      line drawing task;
            rectangle：rectangle drawing task;
            circle：    circle drawing task;
            ……
```


(2) 数据抽象

在不同层次上描述数据对象的细节，定义与该数据对象相关的操作。

例如，在CAD软件中，定义一个叫做drawing的数据对象。可将drawing规定为一个抽象数据类型，定义它的内部细节为:


```
TYPE drawing IS STRUCTURE
DEFIND
    number IS STRING LENGTH(12);
    geometry DEFIND ……
    notes IS STRING LENGTH(256);
    BOM                    DEFIND
END drawing TYPE;
```


- 数据抽象drawing本身由另外一些数据抽象，如geometry、BOM (bill of materials) 构成
- 定义drawing的抽象数据类型之后，可引用它来定义其它数据对象，而不必涉及drawing的内部细节
- 例如，定义:

```
blue-print IS INSTANCE OF drawing;
```

或

```
schematic IS INSTANCE OF drawing;
```

信息隐蔽

- 由 parnas 方法提倡的信息隐蔽是指，每个模块的实现细节对于其它模块来说是隐蔽的。也就是说，模块中所包含的信息（包括数据和过程）不允许其它不需要这些信息的模块使用。

模块的独立性

- 模块（Module）

“模块”，又称“组件”。它一般具有如下三个基本属性:

- ◆功能：描述该模块实现什么功能
- ◆逻辑：描述模块内部怎么做
- ◆状态：该模块使用时的环境和条件

- 在描述一个模块时，还必须按模块的外部特性与内部特性分别描述
- 模块的外部特性
  - ◆模块的模块名、参数表、其中的输入参数和输出参数，以及给程序以至整个系统造成的影响
- 模块的内部特性
  - ◆ 完成其功能的程序代码和仅供该模块内部使用的数据

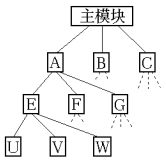
- 模块独立性
  - ◆ 模块独立性, 是指软件系统中每个模块只涉及软件要求的具体的子功能, 而和软件系统中其它的模块的接口是简单的
  - ◆ 例如, 若一个模块只具有单一的功能且与其它模块没有太多的联系, 则称此模块具有模块独立性
  - ◆ 一般采用两个准则度量模块独立性。即模块间耦合和模块内聚

- ◆ 耦合是模块之间的互相连接的紧密程度的度量。
- ◆ 内聚是模块功能强度(一个模块内部各个元素彼此结合的紧密程度)的度量。
- ◆ 模块独立性比较强的模块应是高内聚低耦合的模块。

模块间的耦合

非直接耦合(Nondirect Coupling)

如果两个模块之间没有直接关系，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，这就是非直接耦合。这种耦合的模块独立性最强。



数据耦合 (Data Coupling)

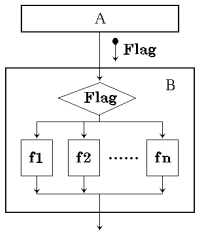
如果一个模块访问另一个模块时，彼此之间是通过简单数据参数（不是控制参数、公共数据结构或外部变量）来交换输入、输出信息的，则称这种耦合为数据耦合。

标记耦合 (Stamp Coupling)

如果一组模块通过参数表传递记录信息，就是标记耦合。这个记录是某一数据结构的子结构，而不是简单变量。

控制耦合 (Control Coupling)

如果一个模块通过传送开关、标志、名字等控制信息，明显地控制选择另一模块的功能，就是控制耦合。



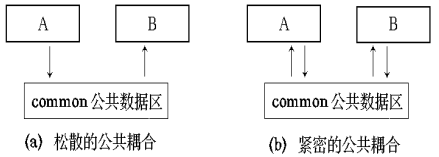
外部耦合（External Coupling）

一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称之为外部耦合。

公共耦合（Common Coupling）

若一组模块都访问同一个公共数据环境，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等。

- 公共耦合的复杂程度随耦合模块的个数增加而显著增加。若只是两模块间有公共数据环境，则公共耦合有两种情况。松散公共耦合和紧密公共耦合。



(a) 松散的公共耦合　(b) 紧密的公共耦合

内容耦合 (Content Coupling)

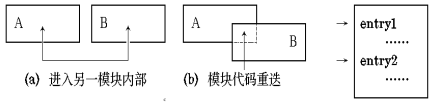
如果发生下列情形，两个模块之间就发生了内容耦合

(1) 一个模块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迭(只可能出现在汇编语言中);

(4) 一个模块有多个入口。



(a) 进入另一模块内部　(b) 模块代码重迭

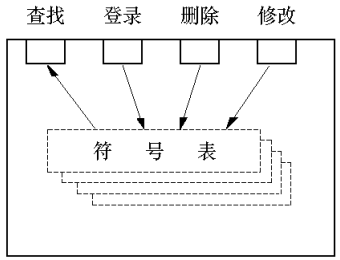
模块内聚

功能内聚 (Functional Cohesion)

一个模块中各个部分都是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。则称该模块为功能内聚模块。

信息内聚 (Informational Cohesion)

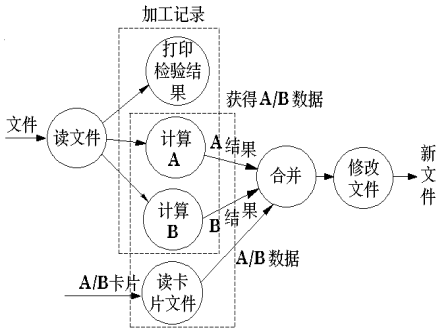
这种模块完成多个功能，各个功能都在同一数据结构上操作，每一项功能有一个唯一的入口点。这个模块将根据不同的要求，确定该执行哪一个功能。由于这个模块的所有功能都是基于同一个数据结构（符号表），因此，它是一个信息内聚的模块。



- 信息内聚模块可以看成是多个功能内聚模块的组合，并且达到信息的隐蔽。即把某个数据结构、资源或设备隐蔽在一个模块内，不为别的模块所知晓。

通信内聚 (Communication Cohesion)

如果一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据，则称之为通信内聚模块。通常，通信内聚模块是通过数据流图来定义的。



过程内聚（Procedural Cohesion）

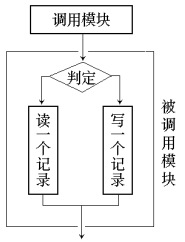
使用流程图做为工具设计程序时，把流程图中的某一部分划出组成模块，就得到过程内聚模块。例如，把流程图中的循环部分、判定部分、计算部分分成三个模块，这三个模块都是过程内聚模块。

时间内聚（Classical Cohesion）

时间内聚又称为经典内聚。这种模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行。例如初始化模块和终止模块。

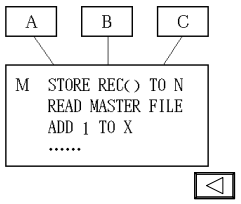
逻辑内聚（Logical Cohesion）

这种模块把几种相关的功能组合在一起，每次被调用时，由传送给模块的判定参数来确定该模块应执行哪一种功能。



巧合内聚（Coincidental Cohesion）

巧合内聚又称为偶然内聚。当模块内各部分之间没有联系，或者即使有联系，这种联系也很松散，则称这种模块为巧合内聚模块，它是内聚程度最低的模块。



结构化设计方法

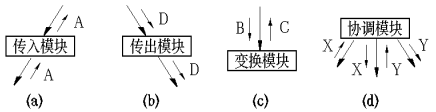
- 首先研究、分析和审查数据流图。从软件的需求规格说明中弄清数据流加工的过程，对于发现的问题及时解决。
- 然后根据数据流图决定问题的类型。数据处理问题典型的类型有两种：变换型和事务型。针对两种不同的类型分别进行分析处理。

- 由数据流图推导出系统的初始结构图。
- 利用一些启发式原则来改进系统的初始结构图，直到得到符合要求的结构图为止。
- 修改和补充数据词典。
- 制定测试计划。

在系统结构图中的模块

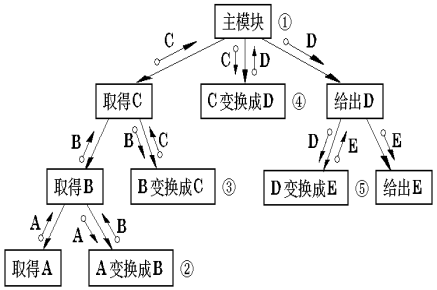
- 传入模块 — 从下属模块取得数据，经过某些处理，再将其传送给上级模块。它传送的数据流叫做逻辑输入数据流。
- 传出模块 — 从上级模块获得数据，进行某些处理，再将其传送给下属模块。它传送的数据流叫做逻辑输出数据流。

- 变换模块 — 它从上级模块取得数据，进行特定的处理，转换成其它形式，再传送回上级模块。它加工的数据流叫做变换数据流。
- 协调模块 — 对所有下属模块进行协调和管理的模块。



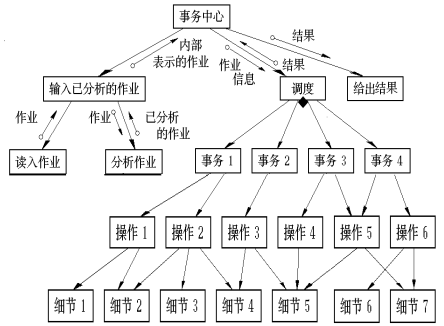
变换型系统结构图

- 变换型数据处理问题的工作过程大致分为三步，即取得数据，变换数据和给出数据。
- 相应于取得数据、变换数据、给出数据，变换型系统结构图由输入、中心变换和输出等三部分组成。



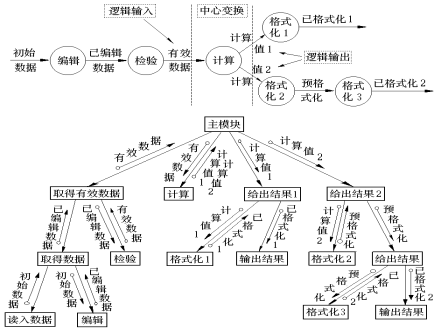
事务型系统结构图

- 它接受一项事务，根据事务处理的特点和性质，选择分派一个适当的处理单元，然后给出结果。
- 在事务型系统结构图中，事务中心模块按所接受的事务的类型，选择某一事务处理模块执行。各事务处理模块并列。每个事务处理模块可能要调用若干个操作模块，而操作模块又可能调用若干个细节模块。

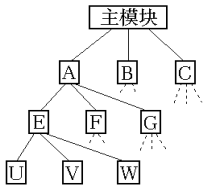


变　换　分　析

- 变换分析方法由以下四步组成：
  - 重画数据流图；
  - 区分有效（逻辑）输入、有效（逻辑）输出和中心变换部分；
  - 进行一级分解，设计上层模块；
  - 进行二级分解，设计输入、输出和中心变换部分的中、下层模块。



① 在选择模块设计的次序时，必须对一个模块的全部直接下属模块都设计完成之后，才能转向另一个模块的下层模块的设计。



② 在设计下层模块时，应考虑模块的耦合和内聚问题，以提高初始结构图的质量。

③ 使用“黑箱”技术：在设计当前模块时，先把这个模块的所有下层模块定义成“黑箱”，在设计中利用它们时，暂时不考虑其内部结构和实现。在这一步定义好的“黑箱”，在下一步就可以对它们进行设计和加工。这样，又会导致更多的“黑箱”。最后，全部“黑箱”的内容和结构应完全被确定。

④ 在模块划分时，一个模块的直接下属模块一般在5个左右。如果直接下属模块超过10个，可设立中间层次。

⑤ 如果出现了以下情况，就停止模块的功能分解：

❒当模块不能再细分为明显的子任务时；

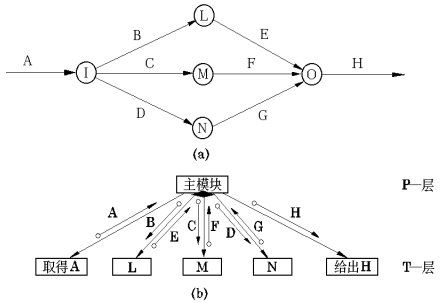
❒当分解成用户提供的模块或程序库的子程序时；

❒当模块的界面是输入／输出设备传送的信息时；

❒当模块不宜再分解得过小时。

事　务　分　析

- 在很多软件应用中，存在某种作业数据流，它可以引发一个或多个处理，这些处理能够完成该作业要求的功能。这种数据流就叫做事务。
- 与变换分析一样，事务分析也是从分析数据流图开始，自顶向下，逐步分解，建立系统到结构图。



事务分析过程

① 识别事务源

利用数据流图和数据词典，从问题定义和需求分析的结果中，找出各种需要处理的事务。通常，事务来自物理输入装置。有时，设计人员还必须区别系统的输入、中心加工和输出中产生的事务。

② 规定适当的事务型结构

在确定了该数据流图具有事务型特征之后，根据模块划分理论，建立适当的事务型结构。

③ 识别各种事务和它们定义的操作

从问题定义和需求分析中找出的事务及其操作所必需的全部信息，对于系统内部产生的事务，必须仔细地定义它们的操作。

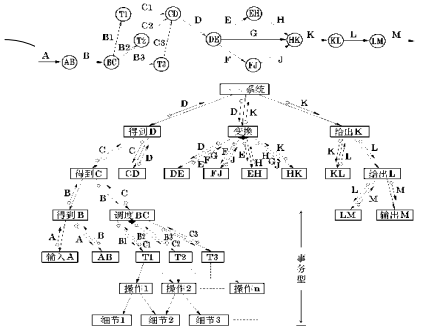
④ 注意利用公用模块
在事务分析的过程中，如果不同事务的一些中间模块可由具有类似的语法和语义的若干个低层模块组成，则可以把这些低层模块构造成公用模块。

⑤ 对每一事务，或对联系密切的一组事务，建立一个事务处理模块；如果发现在系统中有类似的事务，可以把它们组成一个事务处理模块。

⑥ 对事务处理模块规定它们全部的下层操作模块

⑦ 对操作模块规定它们的全部细节模块

变换分析是软件系统结构设计的主要方法。一般，一个大型的软件系统是变换型结构和事务型结构的混合结构。所以，我们通常利用以变换分析为主，事务分析为辅的方式进行软件结构设计。



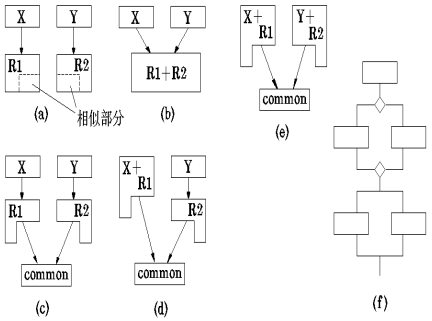
软件模块结构的改进

- 模块功能的完善化

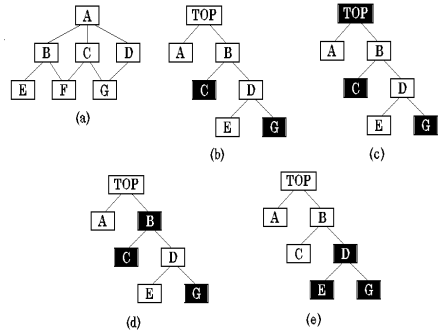
一个完整的模块应当有以下几部分：
① 执行规定的功能的部分；
② 出错处理的部分。当模块不能完成规定的功能时，必须回送出错标志，出现例外情况的原因。
③ 如果需要返回一系列数据给它的调用者，在完成数据加工或结束时，应当给它的调用者返回一个结束状态标志。

- 消除重复功能，改善软件结构

① 完全相似：在结构上完全相似，可能只是在数据类型上不一致。此时可以采取完全合并的方法。
② 局部相似：找出其相同部分，分离出去，重新定义成一个独立的下一层模块。还可以与它的上级模块合并。

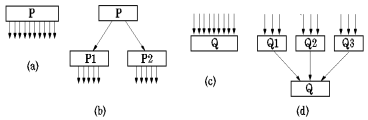


- 模块的作用范围应在控制范围之内
  - 模块的控制范围包括它本身及其所有的从属模块。
  - 模块的作用范围是指模块内一个判定的作用范围，凡是受这个判定影响的所有模块都属于这个判定的作用范围。
  - 如果一个判定的作用范围包含在这个判定所在模块的控制范围之内，则这种结构是简单的，否则，它的结构是不简单的。



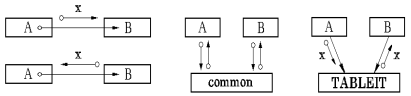
- 尽可能减少高扇出结构，随着深度增大扇入。

如果一个模块的扇出数过大，就意味着该模块过分复杂，需要协调和控制过多的下属模块。应当适当增加中间层次的控制模块。



- 避免或减少使用病态联接

应限制使用如下三种病态联接：
① 直接病态联接 即模块A直接从模块B内部取出某些数据，或者把某些数据直接送到模块B内部。



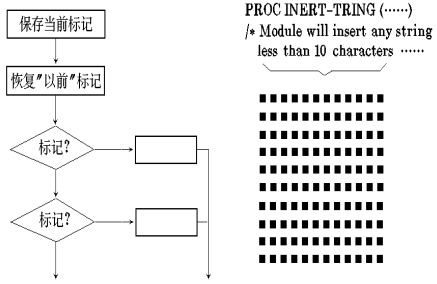
② 公共数据域病态联接 模块A和模块B通过公共数据域，直接传送或接受数据，而不是通过它们的上级模块。这种方式将使得模块间的耦合程度剧增。它不仅影响模块A和模块B，而且影响与公共数据域有关联的所有模块。

③ 通信模块联接 即模块A和模块B通过通信模块TABLEIT传送数据。从表面看，这不是病态联接，因为模块A和模块B都未涉及通信模块TABLEIT 的内部。然而，它们之间的通信（即数据传送）没有通过它们的上级模块。从这个意义上讲，这种联接是病态的。

- 模块的大小要适中

模块的大小，可以用模块中所含语句的数量的多少来衡量。把模块的大小限制在一定的范围之内。通常规定其语句行数在50～100左右，保持在一页纸之内，最多不超过500行。

- 设计功能可预测的模块，但要避免过分受限制的模块
  - 一个功能可预测的模块，不论内部处理细节如何，但对相同的输入数据，总能产生同样的结果。但是，如果模块内部蕴藏有一些特殊的鲜为人知的功能时，这个模块就可能是不可预测的。对于这种模块，如果调用者不小心使用，其结果将不可预测。



- 如果一个模块的局部数据结构的大小、控制流的选择或者与外界(人、硬软件) 的接口模式被限制死了，则很难适应用户新的要求或环境的变更。
- 为了能够适应将来的变更，软件模块中局部数据结构的大小应当是可控制的，控制流的选择对于调用者来说，应当是可预测的。而与外界的接口应当是灵活的。

- 软件包应满足设计约束和可移植性

为了使得软件包可以在某些特定的环境下能够安装和运行，对软件包提出了一些设计约束和可移植的要求。例如，设计约束有时要求一个程序段在存储器中覆盖自身。当这种情况出现时，设计出来的软件程序结构不得不根据重复程度、访问频率、调用间隔等等特性，重新加以组织。

设 计 的 后 处 理

- 为每一个模块写一份处理说明
- 为每一个模块提供一份接口说明
- 确定全局数据结构和局部数据结构
- 指出所有的设计约束和限制
- 进行概要设计的评审
- 进行设计的优化(如果需要和可能的话)

数据设计及文件设计

- 数据设计的原则
- 文件设计

数 据 设 计 的 原 则

- R. S. Pressman数据设计的过程
  - ◆ 为在需求分析阶段所确定的数据对象选择逻辑表示，需要对不同结构进行算法分析，以便选择一个最有效的结构；设计对于这种逻辑数据结构的一组操作，以实现各种所期望的运算。

  - ◆ 确定对逻辑数据结构所必需的那些操作的程序模块(软件包)，以便限制或确定各个数据设计决策的影响范围。

- Pressman提出了一组原则，用来定义和设计数据。实际上，在进行需求分析时往往就开始了数据设计。

1. 用于软件的系统化方法也适用于数据。在导出、评审和定义软件的需求和软件系统结构时，必须定义和评审其中所用到的数据流、数据对象及数据结构的表示。应当考虑几种不同的数据组织方案，还应当分析数据设计给软件设计带来的影响。

2. 确定所有的数据结构和在每种数据结构上施加的操作。设计有效的数据结构，必须考虑到要对该数据结构进行的各种操作。
3. 应当建立一个数据词典并用它来定义数据和软件的设计。数据词典清楚地说明了各个数据之间的关系和对数据结构内各个数据元素的约束。

4. 低层数据设计的决策应推迟到设计过程的后期进行。在进行需求分析时确定的总体数据组织，应在概要设计阶段加以细化，在详细设计阶段才规定具体的细节。
5. 数据结构的表示只限于那些必须直接使用该数据结构内数据的模块才能知道。此原则就是信息隐蔽和与此相关的耦合性原则。

6. 应当建立一个存放有效数据结构及相关操作的库。数据结构应当设计成为可复用的。建立一个存有各种可复用的数据结构模型的部件库。
7. 软件设计和程序设计语言应当支持抽象数据类型的定义和实现。

以上原则适用于软件工程的定义阶段和开发阶段。“清晰的信息定义是软件开发成功的关键”。

文 件 设 计

文件设计的过程，主要分两个阶段。第一个阶段是文件的逻辑设计，主要在概要设计阶段实施。

(1) 整理必须的数据元素：

在软件设计中所使用的数据，有长期的，有短期的，还有临时的。它们都可以存放在文件中，在需要时对它们进行访问。因此首先必须整理应存储的数据元素，给它们一个易于理解的名字，指明其类型和位数，以及其内容涵义。

(2) 分析数据间的关系：

分析在业务处理中哪些数据元素是同时使用的。把同时使用次数多的数据元素归纳成一个文件进行管理。分析数据元素的内容，研究数据元素与数据元素之间的逻辑关系，根据分析，弄清数据元素的含义及其属性。

(3) 确定文件的逻辑设计：

根据数据关联性分析，明确哪些数据元素应当归于一组进行管理，把应当归于一组的数据元素进行统一布局，产生文件的逻辑设计。应用关系模型设计文件的逻辑结构时，必须使其达到第三范式(3NF)，以减少数据的冗余，提高存取的效率。

顾客文件

数据元素名	属性	长度	备注
顾客号码	X	6	
顾客姓名	K	15	
住址1	K	10	省，市
住址2	K	10	区，街
住址3	K	10	门牌号
电话号码	X	12	
邮政编码	X	6	

商品文件

数据元素名	属性	长度	备注
商品号码	X	8	3英文+
商 品 名	K	20	5数字
单 价	N	7	
单 位	X	3	
期首库存量	N	7	
现在库存量	N	7	

X : 英文字母＋数字;　K : 汉字;　N : 数字

第二个阶段是文件的物理设计，主要在软件的详细设计阶段实施

(4) 理解文件的特性：

对于文件的逻辑规格说明，研究从业务处理的观点来看所要求的一些特性，包括文件的使用率、追加率和删除率，以及保护和保密等。考虑需要采用什么文件组织形式。

(5) 确定文件的组织方式：

一般要根据文件的特性，来确定文件的组织方式。

顺序文件：
- 连续文件
- 串联文件。

直接存取文件：
- 无关键字直接存取文件
- 带关键字直接存取文件
- 桶式直接存取文件。

索引顺序文件：

其基本数据记录按顺序文件组织，记录排列顺序必须按关键字值升序或降序安排，且具有索引部分，也按同一关键字进行索引。

分区文件：

这类文件主要用于存放程序。它由若干称为成员的顺序组织的记录组和索引组成。

虚拟存储文件：

这是基于操作系统的请求页式存储管理功能而建立的索引顺序文件。

倒排文件：

按候选属性建立索引表。

(6) 确定文件的存储介质；

(7) 确定文件的记录格式；

确定文件记录中各数据　　项以及 它们在记录中的物理安排。

❒记录的长度：设计记录的长度要确保能满足需要，还要考虑使用设备的制约和效率，尽可能与读写单位匹配，并尽可能减少处理过程中内外存的交换次数。

❒数据项的顺序：对于可变长记录，应在记录的开头记入长度信息；对于关键字项，应尽量按级别高低，顺序配置；联系较密切的数据项，应归纳在一起进行配置。

❒数据项的属性：属性相同的数据项，应尽量归纳在一起配置；数据项应按双字长，全字长，半字长和字节的属性，顺序配置。

❒预留空间：考虑到将来可能的变更或扩充，应当预先留一些空闲空间。不必统一地预留，可在有可能变更或扩充的项旁边，在相邻接处预留。

(8) 估算存取时间和存储容量。

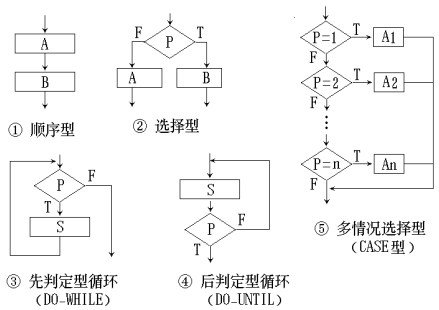
过 程 设 计

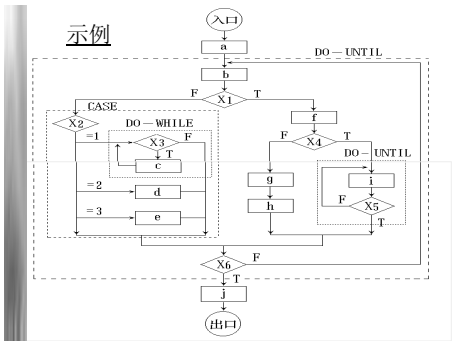
- 从软件开发的工程化观点来看，在使用程序设计语言编制程序以前，需要对所采用算法的逻辑关系进行分析，设计出全部必要的过程细节，并给予清晰的表达。这就是过程设计的任务。

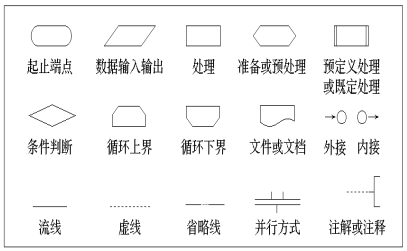
- 在过程设计阶段，要决定各个模块的实现算法，并精确地表达这些算法。表达过程规格说明的工具叫做详细设计工具，它可以分为以下三类：
  - 图形工具
  - 表格工具
  - 语言工具

程序流程图

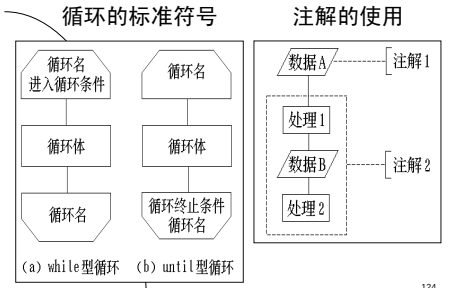
- 程序流程图也称为程序框图，程序流程图使用五种基本控制结构是：



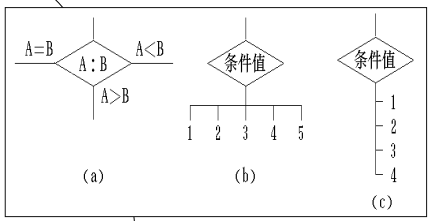




程序流程图的标准符号

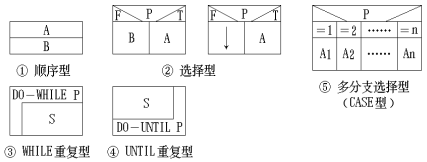


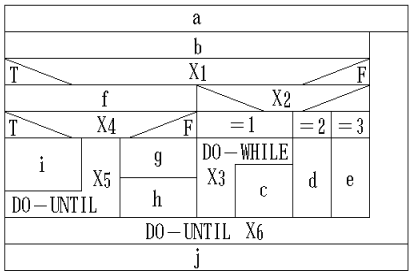
多出口判断



N-S图

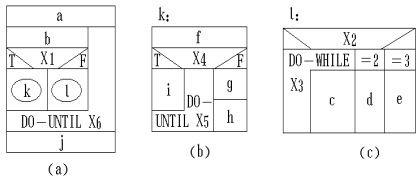
- N-S图也叫做盒图。五种基本控制结构由五种图形构件表示。





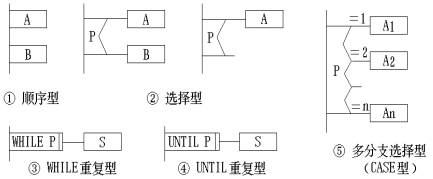
示例

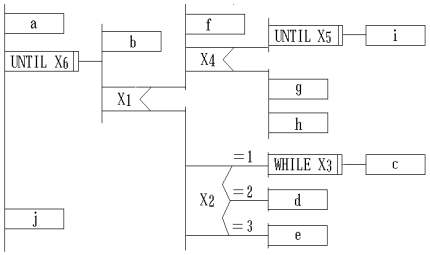
N-S图的嵌套定义形式



问题分析图(PAD)

- PAD也设置了五种基本控制结构的图式，并允许递归使用。



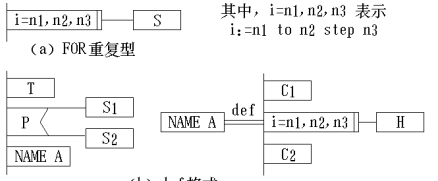


PAD描述的示例

对应于增量型循环结构

for *i* := *n*1 **to** *n*2 **step** *n*3 **do**

在PAD中有相应的循环控制结构

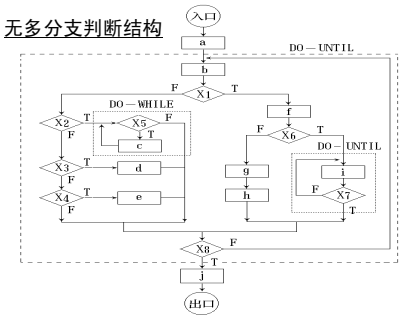


PAD的扩充控制结构

判定表

- 判定表用于表示程序的静态逻辑
- 在判定表中的条件部分给出所有的两分支判断的列表，动作部分给出相应的处理
- 要求将程序流程图中的多分支判断都改成两分支判断

无多分支判断结构



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
X1	T	T	T	T	T	F	F	F	F	F	F	F	F	F
X2	–	–	–	–	–	T	T	T	F	F	F	F	F	F
X3	–	–	–	–	–	–	–	–	F	F	T	T	F	F
X4	–	–	–	–	–	–	–	–	F	F	–	–	T	T
X5	–	–	–	–	–	T	F	F	–	–	–	–	–	–
X6	T	T	T	F	F	–	–	–	–	–	–	–	–	–
X7	T	T	F	–	–	–	–	–	–	–	–	–	–	–
X8	T	F	–	T	F	–	T	F	F	T	T	F	T	F
a	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
b	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
c	–	–	–	–	–	Y	–	–	–	–	–	–	–	–
d	–	–	–	–	–	–	–	–	–	–	Y	Y	–	–
e	–	–	–	–	–	–	–	–	–	–	–	–	Y	Y
f	Y	Y	Y	Y	Y	Y	–	–	–	–	–	–	–	–
g	–	–	–	Y	Y	–	–	–	–	–	–	–	–	–
h	–	–	–	Y	Y	–	–	–	–	–	–	–	–	–
i	Y	Y	Y	–	–	–	–	–	–	–	–	–	–	–
j	Y	–	–	Y	–	–	Y	–	–	Y	Y	–	Y	–

建立判定表的步骤

- 列出与一个具体过程(或模块)有关的所有处理。
- 列出过程执行期间的所有条件(或所有判断)。
- 将特定条件取值组合与特定的处理相匹配，消去不可能发生的条件取值组合。
- 将右部每一纵列规定为一个处理规则，即对于某一条件取值组合将有什么动作。

PDL (Program Design Language)

- PDL是一种用于描述功能模块的算法设计和加工细节的语言。称为设计程序用语言。它是一种伪码。
- 伪码的语法规则分为“外语法”和“内语法”。
- PDL具有严格的关键字外语法，用于定义控制结构和数据结构，同时它的表示实际操作和条件的内语法又是灵活自由的，可使用自然语言的词汇。

示例：拼词检查程序

```
PROCEDURE spellcheck IS
  BEGIN
    split document into single words
    lood up words in dictionary
    display words which are not in dictionary
    create a new dictionary
  END spellcheck
```


PDL的特点

- 提供全部结构化控制结构、数据说明和模块特征。能对PDL正文进行结构分割，使之变得易于理解。
- 为了区别关键字，规定关键字一律大写，其它单词一律小写。或者规定关键字加下划线，或者规定它们为黑体字。

- 内语法使用自然语言来描述处理特性。内语法比较灵活，只要写清楚就可以，不必考虑语法错，以利于人们可把主要精力放在描述算法的逻辑上。
- 有数据说明机制，包括简单的(如标量和数组)与复杂的(如链表和层次结构)的数据结构。
- 有子程序定义与调用机制，用以表达各种方式的接口说明。

使用PDL语言，逐步求精：

```
PROCEDURE spellcheck
BEGIN
  --* split document into single words
  LOOP get next word
    add word to word list in sortorder
    EXIT WHEN all words processed
  END LOOP
  --* look up words in dictionary
  LOOP get word from word list
```



```
    IF word not in dictionary THEN
      --* display words not in dictionary
      display word
      prompt on user terminal
      IF user response says word OK THEN
        add word to good word list
      ELSE
        add word to bad word list
      ENDIF
    ENDIF
    EXIT WHEN all words processed
  END LOOP
```



```
  --* create a new words dictionary
  dictionary :=
    merge dictionary and good word list
END spellcheck
```