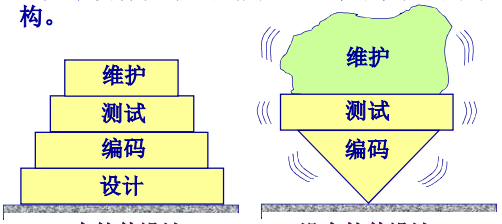
软件工程
第二部分 结构化方法学
总体设计

4.1 软件设计的概念

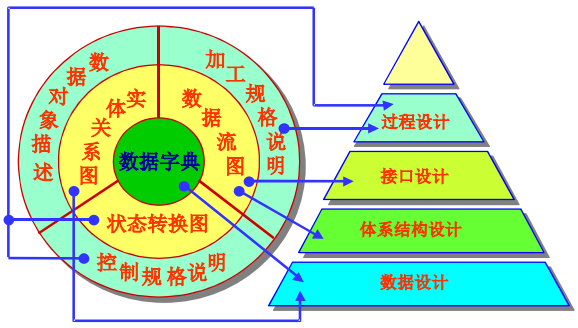
- 软件设计是后续开发步骤及软件维护工作的基础。如果没有设计，只能建立一个不稳定的系统结构。



- 软件设计的基本目标是用比较抽象概括的方式确定目标系统如何完成预定的任务，即软件设计是确定系统的物理模型。
- 从技术观点来看，软件设计包括数据设计、体系结构设计、接口设计、过程设计。

- 数据设计:将实体关系图中描述的对象和关系，以及数据字典中描述的详细数据内容转化为数据结构的定义。
- 体系结构设计:定义软件系统各主要成份之间的关系。
- 接口设计:根据数据流图定义软件内部各成份之间、软件与其它协同系统之间及软件与用户之间的交互机制。
- 过程设计:把结构成份转换成软件的过程性描述。

将分析模型转换为设计



- 从工程管理角度来看，软件设计分两步完成：总体设计和详细设计。
- 为什么在详细设计之前先进行总体设计？

  可以站在全局高度上，花较少成本，从较抽象的层次上分析对比多种可能的系统实现方案和软件结构，从中选出最佳方案和最合理的软件结构，从而用较低成本开发出较高质量的软件系统。

- 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。总体设计又称为概要设计或初步设计。
- 总体设计阶段的主要任务：
  - ✓确定系统实现方案：划分出组成系统的物理元素(黑盒子级)--程序、文件、数据库、人工过程和文档等等。
  - ✓设计软件的结构：确定系统中每个程序是由哪些模块组成的，及这些模块相互间的关系。

4.2 总体设计的过程

- 总体设计过程通常由两个主要阶段组成：
  - ✓系统设计：确定系统的具体物理实现方案；
  - ✓结构设计：确定软件结构。
- 典型的总体设计过程包括下述9个步骤：
  1. 设想供选择的方案：基于数据流图进行各种处理的分组方法。
  2. 选取合理的方案：选取合理方案并进行系统流程、成本/效益、进度计划等分析。
  3. 推荐最佳方案：推荐并确定最优方案。进入结构设计阶段。

4. 功能分解:从实现的角度对复杂的功能进一步分解，是对数据流图的进一步细化。
5. 设计软件结构:系统模块化分解，并采用工具表示出来。
6. 设计数据库
7. 制定测试计划
8. 书写文档(系统说明、用户手册、测试计划、详细的实现计划、数据库设计结果)
9. 审查和复审

4.3.1软件设计的原理

- 软件设计的原理和启发规则是进行软件设计的主要手段和应遵循的原则。
- 基本的软件设计原理包括：
  - ❑ 模块化
  - ❑ 抽象
  - ❑ 逐步求精
  - ❑ 信息隐藏和局部化
  - ❑ 模块独立

1.模块化

- 模块是由边界元素限定的相邻程序元素的序列，而且有一个总体标识符代表它。
- 按照模块的定义，过程、函数、子程序和宏等，都可作为模块。面向对象方法学中的对象是模块，对象内的方法也是模块。
- 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，把这些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。

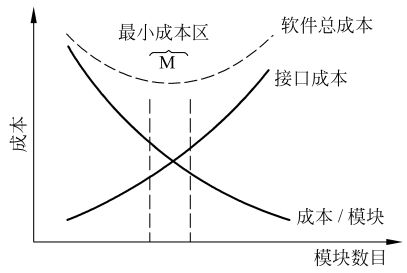
- 软件设计应当遵循模块化的原则
  - ✓每个模块可独立地开发、测试，最后组装成完整的程序。模块是构成程序的基本构件。
  - ✓出发点:复杂问题“分而治之”的原则,目的是使程序的结构清晰，容易阅读、理解、测试、修改。
  - ✓模块化的理论依据:

    C(x):问题x的复杂程度，E(x):解决问题x需要的工作量

    如果 C(P1)>C(P2)，则E(P1)>E(P2)；

    C(P1+P2)>C(P1)+C(P2) （据经验知）

    E(P1+P2)>E(P1)+E(P2)



模块化和软件成本

- Meyer的良好模块设计方法的标准
  - a) 模块可分解性　可将系统按问题 / 子问题分解的原则分解成系统的模块层次结构。
  - b) 模块可组装性　可利用已有的设计构件组装成新系统，不必一切从头开始。
  - c) 模块可理解性　一个模块可不参考其他模块而被理解。
  - d) 模块连续性　对软件需求的一些微小变更只导致对某个模块的修改而整个系统不用大动。
  - e) 模块保护　将模块内出现异常情况的影响范围限制在模块内部。

2.抽象

- 抽象是一种重要的思维工具，是指抽出事物的本质特性而暂时不考虑他们的细节（思维能力的限制）。
- 软件开发的过程多次应用抽象机制。每一步是对软件解法的抽象层次的不断精化：

  可行性研究 → 需求分析（需求规格说明）
  → 总体设计（设计方案）→详细设计
  → 实现（程序）。

- 软件设计应遵循抽象化的原则，包含过程抽象、数据抽象和控制抽象：
  - ✓ 过程抽象　是指在软件设计中将处理过程的实现细节隐藏在操作抽象中，可以直接通过模块接口使用这些处理操作（如方法声明）。
  - ✓ 数据抽象　是指采用抽象数据类型表示数据，实现数据封装。使用者可通过接口使用数据而不必关心数据结构的实现（如学生记录）。
  - ✓ 控制抽象 是指没有指定内部细节的程序控制机制(如操作系统中的同步信号)。

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（规格说明，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。”
- 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。
- 逐步求精最初是由Niklaus Wirth提出的一种自顶向下的设计策略。

- 软件设计应遵循逐步求精的原则，建立一个层次的结构。
  - ✓ 将软件体系结构自顶向下，对过程细节、数据细节和控制细节从抽象到具体，逐层细化，直到用编程语言的语句能够实现为止。
  - ✓ 逐步求精方法确保每个问题都将被解决，而且每个问题都将在适当的时候被解决。
  - ✓ 逐步求精与模块化、抽象等概念紧密相关，求精与抽象是互补的。

4.信息隐藏和局部化

- 信息隐藏原理指出：应该这样设计和确定模块，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。
- 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。
- 局部化有助于实现信息隐藏。隐藏的不是有关模块的一切信息，而是模块的实现细节。

- 软件设计应遵循信息隐蔽和局部化的原则
  - ✓ Parnas主张在开发时，将每个程序的成分隐藏在模块内，定义每一个模块时尽可能少地显露其内部的处理。
  - ✓ 每个模块的实现细节对于其它模块是隐蔽的，将来修改软件时偶然引入错误所造成的影响就可以局限在一个或几个模块内部，不致波及到软件的其它部分。
  - ✓ 在可预见将来可能修改的场合，信息隐蔽可以提高软件的可修改性、可测试性和可移植性。

5. 模块独立性

- 如果一个模块能够独立于其他模块被编程、测试和修改，则该模块具有模块独立性，也称功能独立性。
- 模块独立性是模块化、抽象和信息隐藏的直接产物。
- 模块独立性很重要：
  - ✓ 有效的模块化(即具有独立的模块)的软件比较容易开发，
  - ✓ 独立的模块比较容易测试和维护。

- 1978年Myers提出用两个准则来度量模块独立性，即模块间的耦合和模块的内聚。
  - ✓内聚性：内聚是一个模块内部各个元素彼此结合的紧密程度的度量。
  - ✓耦合性：耦合是模块间互相连接的紧密程度的度量，它取决于各个模块之间接口的复杂度、调用方式以及哪些信息通过接口。

模块间的耦合

耦合性 →

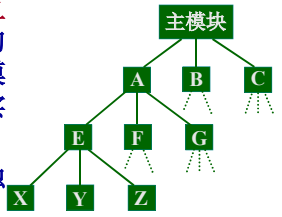
非直接耦合	数据耦合	特征耦合	控制耦合	外部耦合	公共环境耦合	内容耦合

← 模块独立性

- 模块之间耦合性越强，模块独立性越差，这样形成的模块结构界面不好。

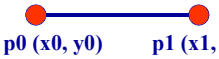
非直接耦合

- 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。
- 非直接耦合的模块独立性最强。



数据耦合

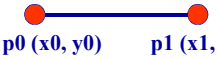
- 一个模块访问另一个模块时，彼此之间是通过简单数据参数(不是控制参数、公共数据结构或外部变量) 来交换输入/输出信息。
- 数据耦合是低耦合。



数据耦合 LINE (x0, y0, x1, y1)

特征耦合

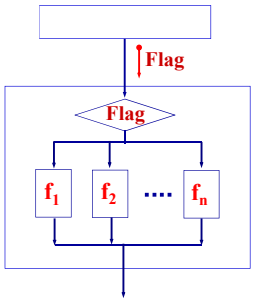
- 特征耦合：把整个数据结构作为参数传递，而被调用的模块只需使用其中一部分数据元素。可能导致对数据的控制。
- 特征标记又称为标记耦合，是数据耦合的一种变形。



标记耦合 LINE (p0, p1)

控制耦合

- 控制耦合:如果一个模块通过传送开关、标志、名字等控制参数，明显地控制选择另一模块的功能。
- 控制耦合是数据耦合的一种变形。可以通过适当分解由数据来代替。中等程度耦合。



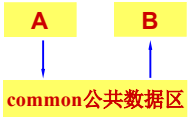
外部耦合

- 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。
- 外部耦合在软件系统中必然存在。耦合程度比较强，要加以限制。

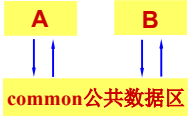
公共环境耦合

- 若一组模块通过同一个公共数据环境相互作用（如访问），则它们之间的耦合就称为公共环境耦合。
- 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。
- 公共环境耦合的复杂程度随耦合模块的个数增加而显著增加。

- 公共环境耦合有两种情况：
  - ✓ 松散公共耦合
  - ✓ 紧密公共耦合



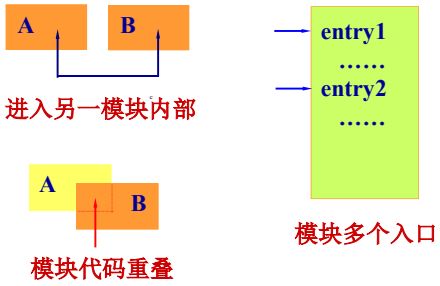
松散的公共耦合



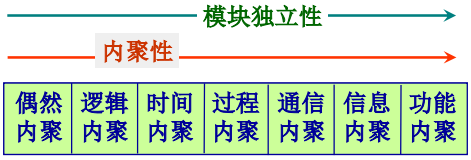
紧密的公共耦合

内容耦合

- 如果发生下列情形，模块之间的耦合就是内容耦合：
  - ✓一个模块直接访问另一个模块的内部数据
  - ✓一个模块不通过正常入口转到另一模块内部
  - ✓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)；
  - ✓一个模块有多个入口（模块有多个功能）。
- 应该坚决避免使用内容耦合。



模块内聚



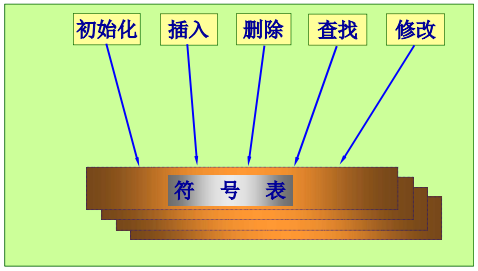
- 模块内聚性越强，模块独立性越好，对于形成的模块结构有比较好的作用。
- 要求模块结构达到高内聚，低耦合。

功能内聚

- 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。该模块为功能内聚模块。
- 功能内聚模块的功能独立性最强。

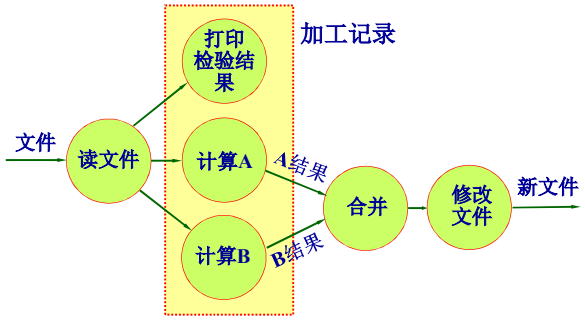
信息内聚

- 信息内聚的模块：完成多个功能，各个功能相互独立但都在同一数据结构（如符号表）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。
- 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。



通信内聚

- 如果一个模块内所有元素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。模块内各个功能是紧密相连的。
- 通信内聚属于中内聚。



过程内聚

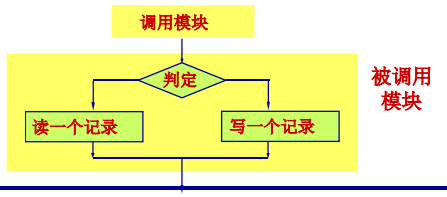
- 过程内聚：如果一个模块内的处理元素是相关的，而且必须以特定的次序执行。
- 通常，把程序流程图中的某一部分划出组成模块，就得到过程内聚模块。例如，把流程图中的循环部分、判定部分、计算部分分成三个模块，这三个模块都是过程内聚模块。
- 过程内聚属中内聚。

时间内聚

- 时间内聚又称为经典内聚。一个模块内包含的任务必须在同一时间段内执行。
- 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（如初始化模块）。
- 时间内聚属低内聚。

逻辑内聚

- 逻辑内聚：如果一个模块完成的任务在逻辑上属于相同或相似的一类。属于低内聚。
- 把几种相关的功能组合在一起。调用时，由判定参数来确定执行的具体功能。



偶然内聚

- 偶然内聚：模块完成一组任务，各任务之间没有联系，或联系很松散。例如，当几个模块内正好有一段代码是相同的，将它们抽取出来形成单独的模块。属于低内聚。

▪软件设计要遵循模块独立性原则：

- ✓软件设计中应该追求高内聚、低耦合的系统。
- ✓尽量使用数据耦合、少用控制耦合和特征耦合、限制外部耦合和公共环境耦合的范围、不用内容耦合；
- ✓力求做到高内聚;通常中等程度的内聚也是可以采用的，而且效果和高内聚相差不多；坚决不使用低内聚。

4.3.2 启发式原则

- 人们在开发计算机软件的长期实践中积累了丰富的经验，总结这些经验得出了一些启发式规则。不是设计的目标，也不是应该普遍遵循的原理。
- 这些启发式规则在许多场合能给软件工程师以有益的启示，往往能帮助他们找到改进软件设计提高软件质量的途径。

1. 提高模块独立性

通过模块分解或合并，力求降低耦合提高内聚。

例如：消除重复功能，改善软件结构

- ✓ 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。
- ✓ 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。

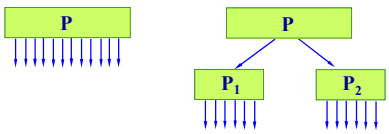
2. 模块功能完善化

一个完整的模块应当有以下几部分：

- ✓ 执行规定的功能的部分；
- ✓ 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。
- ✓ 如果需要返回数据给它的调用者，在完成数据加工或结束时， 应当给它的调用者返回一个状态码。

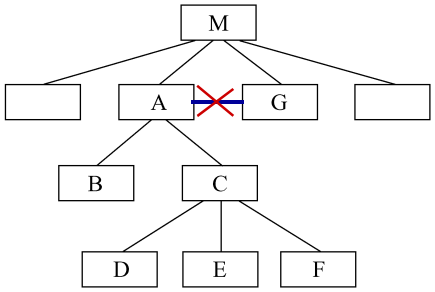
3. 深度、宽度、扇出、扇入都应适当

例如：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。



4. 模块的作用域应在控制域之内

- ✓模块的控制域包括它本身及其所有的从属模块。
- ✓模块的作用域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域包含在这个判定所在模块的控制域之内，则这种结构是简单的，否则，它的结构是不简单的。



5. 设计单入口单出口的模块

尽量避免内容耦合，而且易理解、易维护。

6. 力争降低模块接口的复杂程度

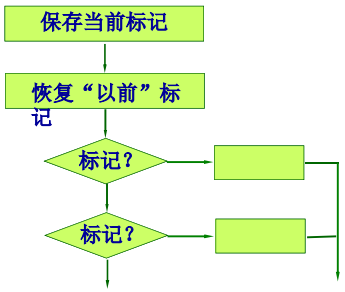
模块接口复杂是软件发生错误的一个主要错误。设计模块接口应注意信息传递简单、且和模块功能一致。

例子：SQRT(Table, root)

7. 模块的大小要适中

8. 模块的功能可预测

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。



功能不可预测的模块

- Davis的软件设计原则

①设计应具有可跟踪性，能回溯到软件需求;

②设计不必每次都从头做起，可以复用已有的设计模式和数据模式。

③设计应当缩小软件与现实世界中问题的“智力距离”，尽量逼近问题领域的结构;

④设计应具有一致性和集成性。整个系统应具有统一的风格和格式，具有良好的接口。

⑤设计结果应能适应未来可能的变更;

⑥ 设计不是编码，编码也不是设计。设计模型的抽象级别比源代码高。在编码级别上唯一的设计决策是补充一些实现细节。

⑦ 设计应具有容错性和异常处理能力。对于异常数据、事件、操作条件等能够平滑处理。

⑧ 在建立设计方案时就应能评估设计质量，而不是在系统编码之后。

⑨ 应坚持设计评审，减少概念性（语义性）的错误。