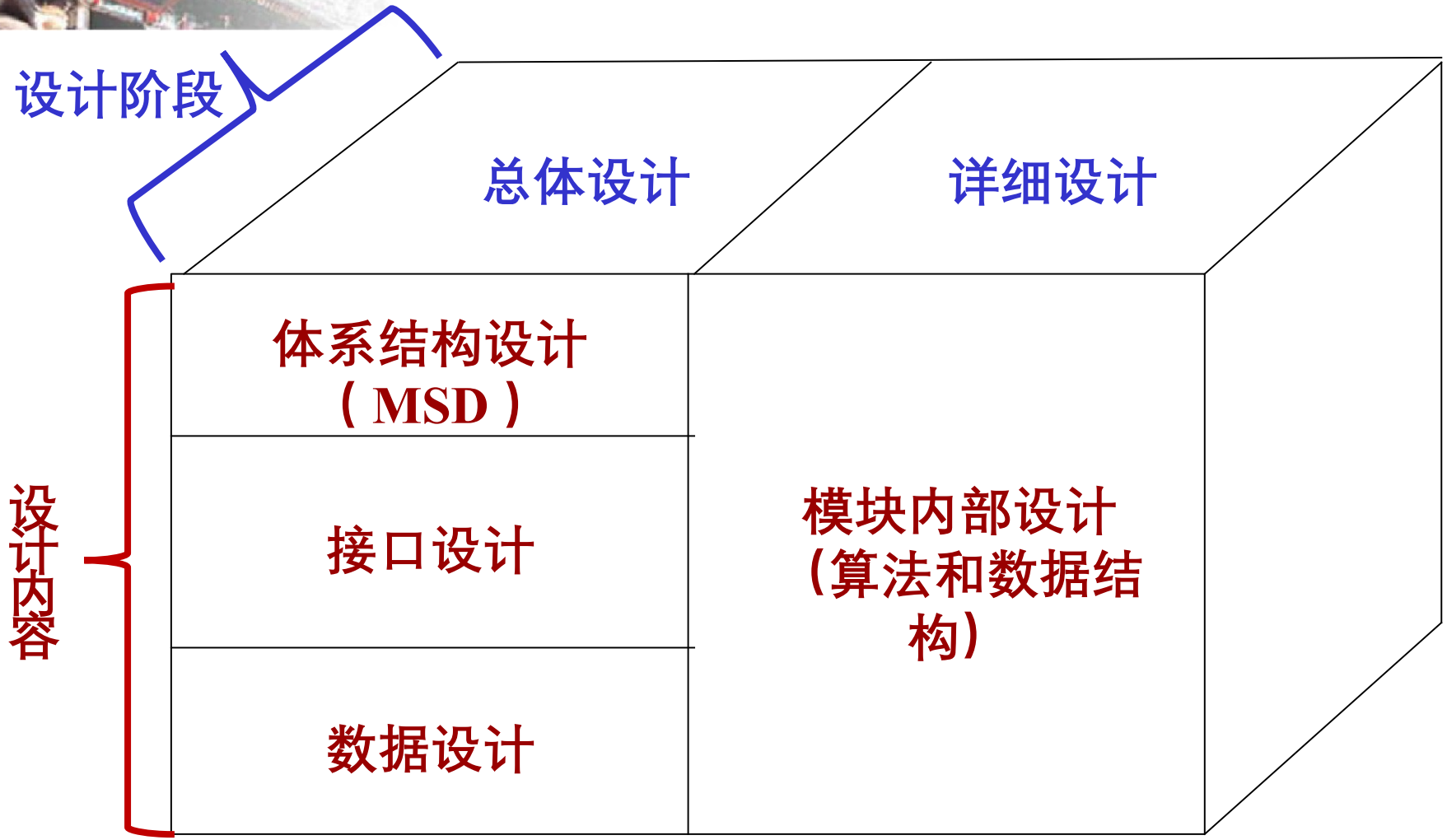


图 1　设计阶段和设计内容

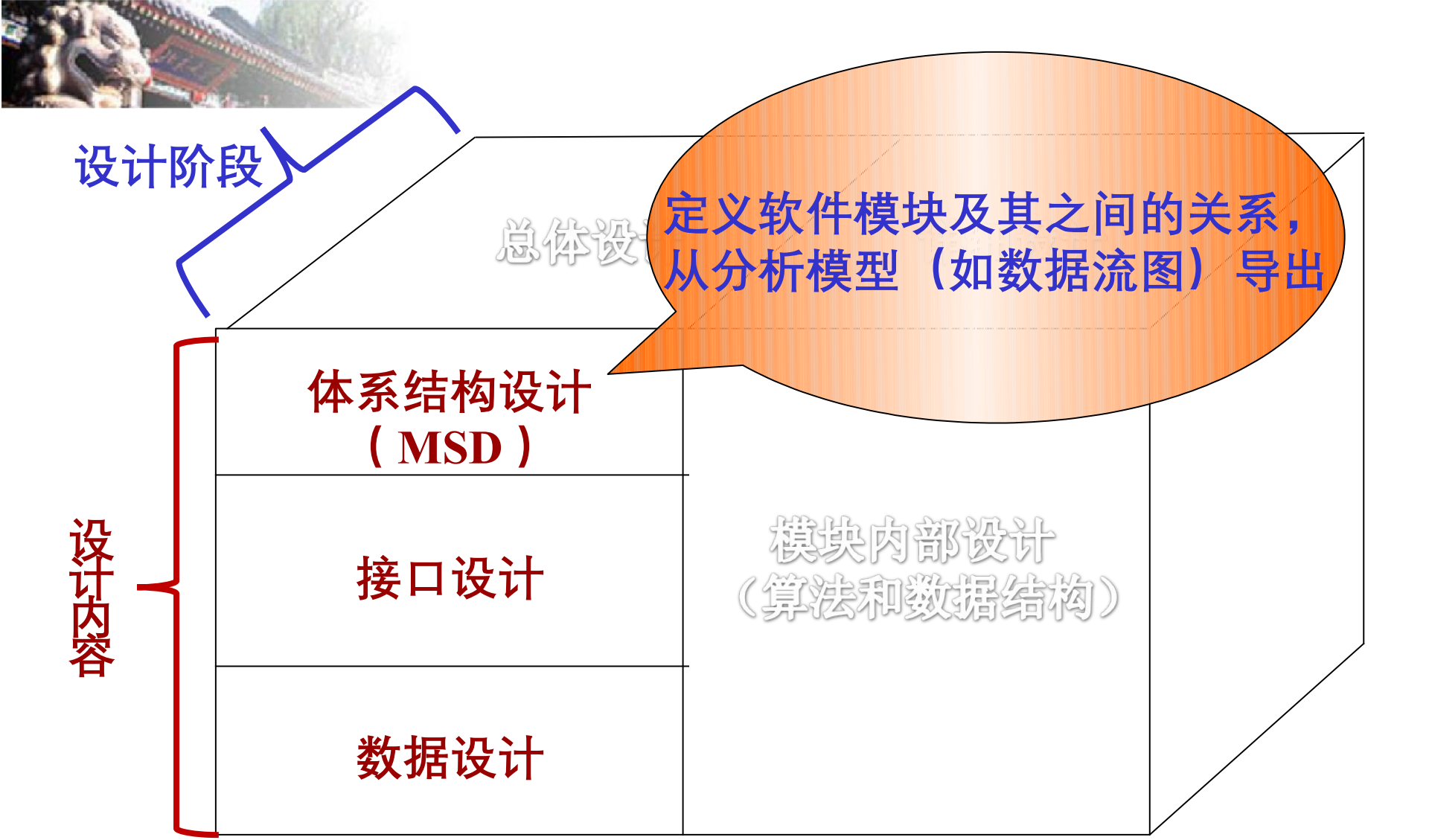


图 1　设计阶段和设计内容

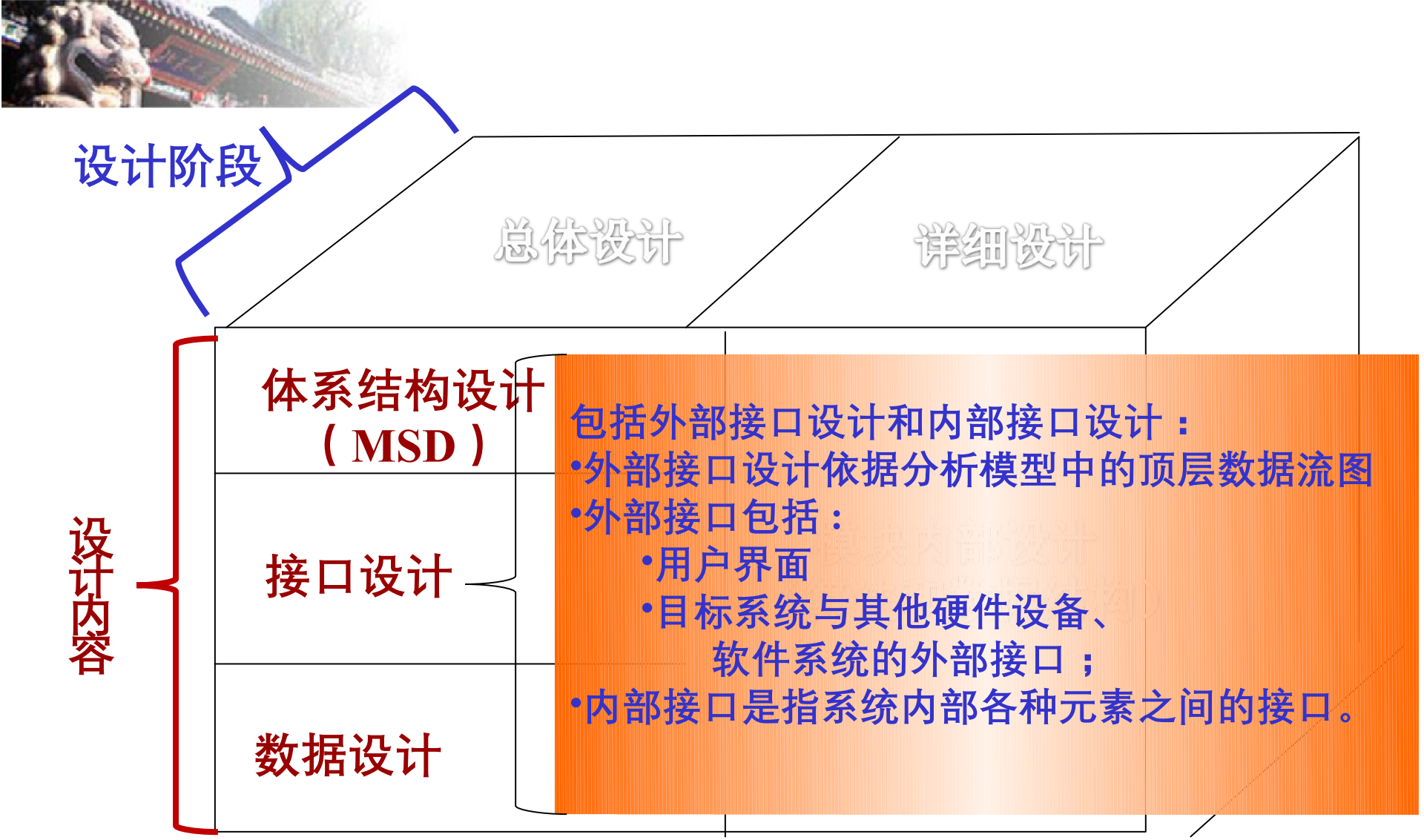


图 1　设计阶段和设计内容

图 1　设计阶段和设计内容

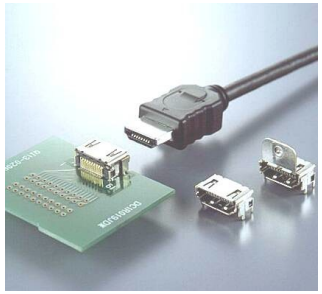
一、总体设计

总体设计分为三个阶段：

第一阶段：初始设计。在对给定的数据流图进行复审和精化的基础上，将其转化为初始的模块结构图。根据穿越系统边界的数据流初步确定系统与外部的接口。

第二阶段：精化设计。依据模块“高内聚低耦合”的原则，精化初始的模块结构图，并设计其中的全局数据结构和每一模块的接口。

第三阶段：设计复审阶段，对前两个阶段得到的高层软件结构进行复审，必要时还可能需要对软件结构做一些精化工作。



1、接口设计

1.1 接口设计的分类：

接口设计主要包括3个方面：

（1）模块或软件构件间的接口设计；

（2）软件与其他软硬件系统之间的接口设计；

（3）软件与人（用户）之间的交互设计。

系统的接口设计（包括用户界面设计及与其他系统的接口设计）是由穿过系统边界的数据流定义的。

在最终的系统中，数据流将成为用户界面中的表单、报表或与其他系统进行交互的文件或通信。

1.2 人机交互界面

在设计阶段，必须根据需求把交互细节加入到用户界面设计中，包括人机交互所必须的实际显示和输入。

人机交互界面是给用户使用的，为了设计好人机交互界面，设计者需要了解以下信息：

（1）用户界面应具有的特性？

（2）使用软件的用户是什么人？

（3）用户怎样学习与新的计算机系统进行交互？

（4）用户需要完成哪些工作？

- 用户界面应具备的特性
  - 可使用性：是用户界面设计最重要的目标．包括使用简单、界面一致、拥有 help 帮助功能、快速的系统响应和低的系统成本、具有容错能力等。
  - 灵活性：考虑到用户的特点、能力和知识水平，应该使用户接口满足不同用户的要求。因此，对不同的用户，应有不同的界面形式，但不同的界面形式不应影响任务的完成。
  - 可靠性：用户界面的可靠性是指无故障使用的间隔时间。用户界面应能保证用户正确、可靠地使用系统，保证有关程序和数据的安全性。

- 用户类型
  - 外行型：对计算机系统认知很少或毫无了解
  - 初学型：对计算机有一定经验，对系统的认识不足或经验很少，需要很多界面支持。
  - 熟练型：对一个系统有很多经验，需要较少的界面支持，但不能处理意外错误。
  - 专家型：了解系统的内部构造，需要为他们提供能够修改和扩充系统能力的复杂界面。

- **界面设计类型**
  - 如果从用户与计算机交互的角度来看，用户界面设计类型主要有问题描述语言、数据表格、图形、菜单、对话、窗口等。在选用界面形式的时候，应该考虑每种类型的优点和限制，可以从以下几个方面来考察，进行选择：
    - 使用的难易程度
    - 学习的难易程度
    - 操作速度
    - 复杂程度：该界面提供了什么功能、能否用新的方式组合这些功能以增强界面的功能
    - 控制：人机交互时，由计算机还是由人发起和控制对话。
    - 开发的难易程度：该界面设计是否有难度、开发工作量有多大。

一个界面的设计通常使用一种以上的设计类型，每种类型与一个或一组任务相匹配。

- 设计详细的交互遵循的原则：
  - 一致性
  - 操作步骤少
  - 不要“哑播放”
  - 提供 Undo 功能
  - 减少人脑的记忆负担
  - 提高学习效率

2、数据设计

在设计阶段必须对要存储的数据及其格式进行设计。

2.1 文件设计

以下几种情况适合于选择文件存储：

- 数据量较大的非结构化数据，如多媒体信息；
- 数据量大，信息松散，如历史记录、档案文件等；
- 非关系层次化数据。如系统配置文件
- 对数据的存取速度要求极高的情况
- 临时存放的数据

- 文件设计的主要工作就是根据使用要求、处理方式、存储的信息量、数据的活动性以及所提供的设备条件等确定文件类型，选择文件媒体，决定文件组织方法，设计文件记录格式，并估算文件的容量。

2.2 数据库设计

在结构化设计中，很容易将结构化分析阶段建立的数据字典和实体 - 关系模型映射到关系数据库中。

- 数据对象的映射
- 关系的映射

二、 详细设计层

详细设计的任务：定义每一模块

-- 主要引入了关于三种动作控制结构的术语 / 符号

三种控制结构：顺序，选择和循环

这三种结构在表达系统行为方面是完备的

- 结构化程序设计的概念：

设

•结构化程序设计的提出：

•1966 年， C.Bohm 和 G.Jacopini 证明只用三种基本控制结构就能实现单入口和单出口的程序

•1968 年， Dijkstra 的短文“ Goto Statement considered harmful” 引发了程序结构设计的讨论，明确了结构化程序设计思想 .

第一种表达 - 伪码（类程序设计语言 PDL,Program Design language ）

顺序

begin s1;s2;…sn end;

选择

if 条件表达式 then s1

else s2;

循环

while 条件表达式 do s;

伪码是一种混合语言。外部采用形式语言定义控制结构和数据结构，内部使用自然语言。

例如：

Begin

输入一元二次方程的系数

a,b,c;

if b↑2-4ac≥o then 计算两实根 else 输出无实根

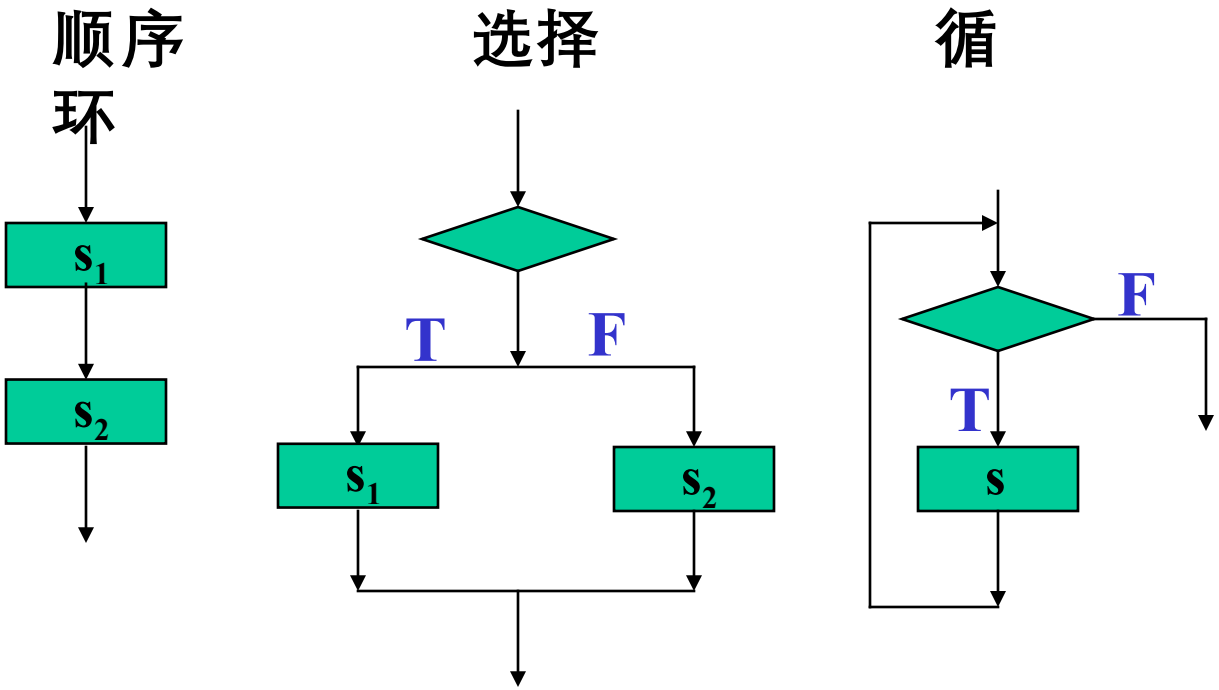
;

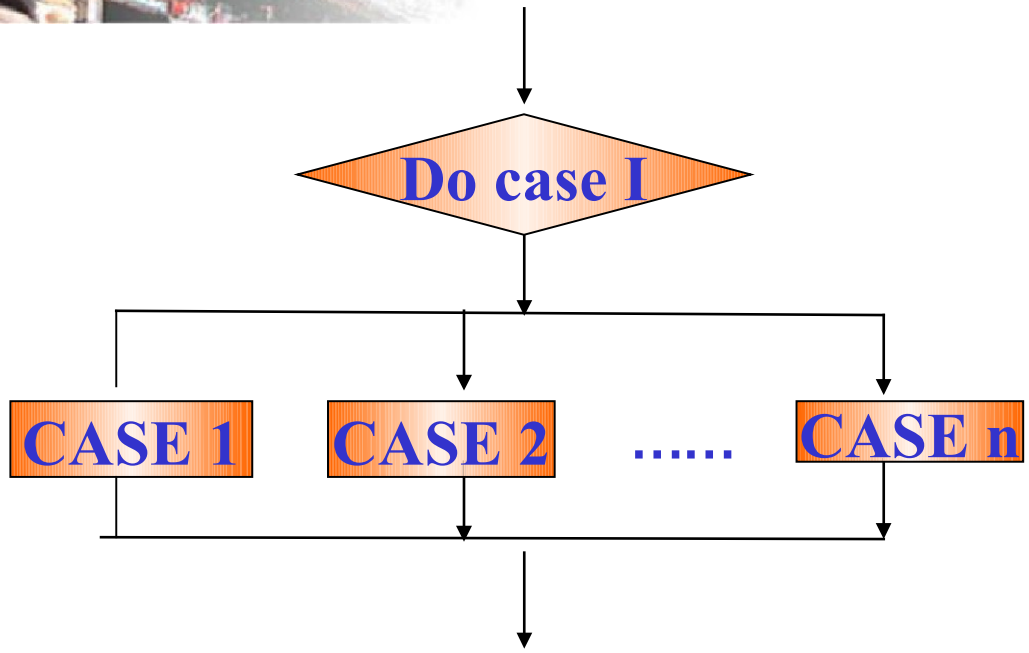
优点：PDL 不仅可以作为设计工具，而且可作为注释工具，直接插在源程序中间，以保持文档和程序的一致性，提高了文档的质量．

缺点：１．不如图形工具那样形象直观．

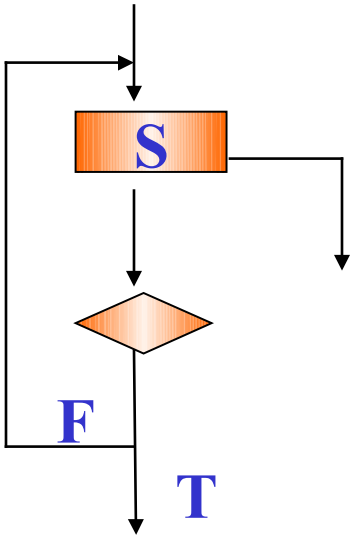
２．当描述复杂的条件组合与动作间的对应关系时，不如判定表和判定树那样清晰简单．

第二种表达 - 程序流程图（程序框图）





多分支结构



Repeat-until 循环结构

优点：对控制流程的描绘很直观，便于初学者掌握．

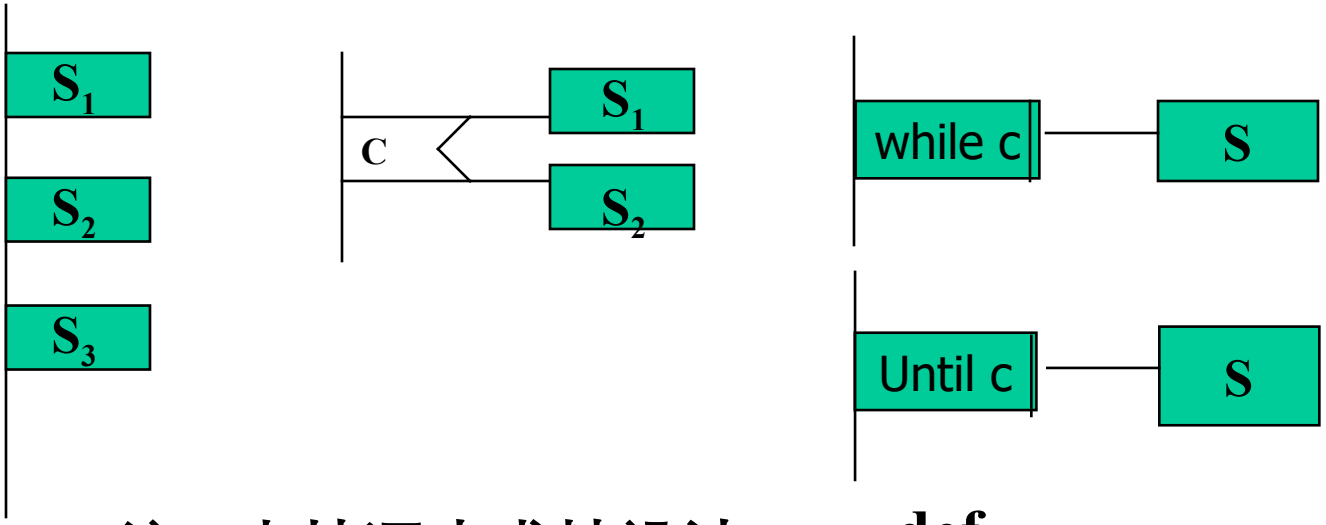
缺点：１．不是一种逐步求精的工具，程序员过早地考虑程序的控制流程，而不是全局结构．

２．所表达的控制流，可以不受约束随意转移．

３．不易表示数据结构．

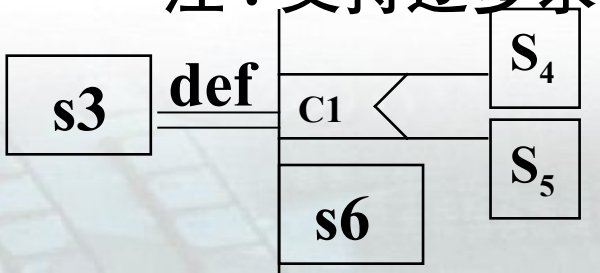
第三种表达－ PAD 图 (Problem Analysis Diagram)

顺序：　　选择 ：　　循环：



注：支持逐步求精设计：def def

例如：



优点：

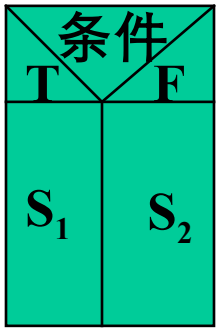
１．支持自顶向下逐步求精的结构化详细设计，可使用“ def”符号逐步增加细节．

２．PAD 图最左边的竖线是程序的主线，随着程序层次的增加，逐步向右延伸，每增加一个层次，图形向右扩展一条竖线，从而使 PAD 图所表现的处理逻辑易读、易懂和易记。

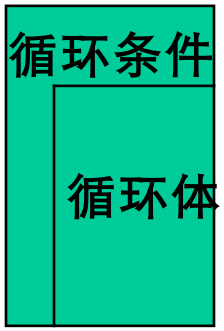
第四种表达 -N-S 图

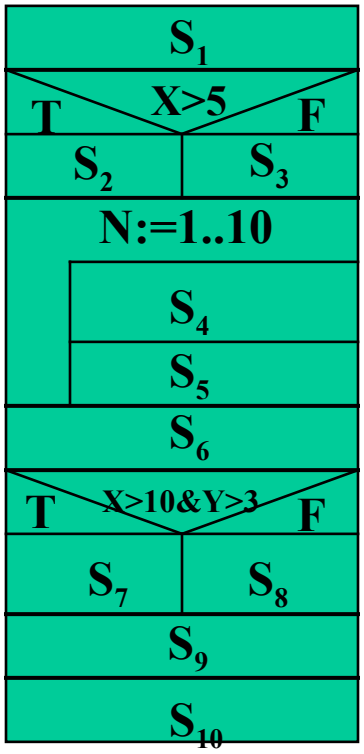
顺序：

选择 ：



循环：





支持逐步求精设计举例

优点：支持自顶向下逐步求精的结构化详细设计，并且严格限制了控制从一个处理到另一个处理的转移 .

第五种判定表和判定树

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、 PDL 都不易清楚描述，这时可以选择判断表来表达复杂的条件组合与应做的动作之间的对应关系 .

判定树是判定表的变种，也能清晰地表达复杂的条件组合与应做的动作之间的对应关系，形式简单，但简洁性不如判定表，数据元素的同一个值往往需要重复写多次，而且越接近树的叶断重复次数越多 .

3 软件设计规约（软件设计说明书）

3.1 什么是软件设计规约

软件设计规约对软件的组织或其组成部分的内部结构的描述，满足系统需求规约所指定的全部功能及性能要求。

3.2 软件设计规约的组成

软件设计规约通常有**概要设计规约**和**详细设计规约**，分别为相应设计过程的输出文档。

概要设计规约指明软件的组织结构，其主要内容包括：

(1) 系统环境

- 硬件、软件接口与人机界面
- 外部定义的数据库
- 与设计有关的限定条件

(2) 设计描述

- 数据流和主要数据结构
- 软件模块的结构
- 模块之间的接口

(3) 对每个模块的描述

- 处理过程外部行为
- 界面定义
- 数据结构
- 必要的注释

(4) 文件结构和全局数据

- 文件的逻辑结构、记录描述以及访问方式
- 交叉引用信息

此外，还应包括有关软件测试方面的要求和说明。

软件概要设计是面向软件开发者的文档，主要作为软件项目管理人员、系统分析人员与设计人员之间交流的媒体。

详细设计规约是对软件各组成部分内部属性的描述，它是概要设计的细化。即在概要设计规约的基础上，增加以下内容：

① 各处理过程的算法

② 算法所涉及的全部数据结构的描述，特别地，对主要数据结构往往包括与算法实现有关的描述

软件设计规约主要作为软件设计人员与程序员之间交流的媒体。

3.3 设计规约格式

1．引言

1．1 编写目的

说明编写本软件设计说明书的目的。

1．2 背景说明

(1) 给出待开发的软件产品的名称；

(2) 说明本项目的提出者、开发者及用户；

(3) 说明该软件产品将做什么，如有必要，说明不做什么。

1．3 术语定义

列出本文档中所用的专门术语的定义和外文首字母组词的原词组。

1．4 参考资料

列出本文档中所引用的全部资料，包括标题、文档编号、版本号、出版日期及出版单位等，必要时注明资料来源。

2．总体设计

2．1 需求规定

说明对本软件的主要输入、输出、处理的功能及性能要求。

2．2 运行环境

简要说明对本软件运行的软件、硬件环境和支持环境的要求。

2．3 处理流程

说明本软件的处理流程，尽量使用图、文、表的形式。

2．4 软件结构

在 DFD 图的基础上，用模块结构图来说明各层模块的划分及其相互关系，划分原则上应细到程序级（即程序单元），每个单元必须执行单独一个功能（即单元不能再分了）。

3．运行设计

3．1 运行模块的组合

说明对系统施加不同的外界运行控制时所引起的各种不同的运行模块的组合，说明每种运行所经历的内部模块和支持软件。

3．2 运行控制

说明各运行控制方式、方法和具体的操作步骤。

4．系统出错处理

4．1 出错信息简要说明每种可能的出错或故障情况出现时，系统输出信息的格式和含义。

4．2 出错处理方法及补救措施

说明故障出现后可采取的措施，包括：

(1) 后备技术。当原始系统数据万一丢失时启用的副本的建立和启动的技术，如周期性的信息转储；

(2) 性能降级。使用另一个效率稍低的系统或方法（如手工操作、数据的人工记录等），以求得到所需结果的某些部分；

(3) 恢复和再启动。用建立恢复点等技术，使软件再开始运行。

5．模块设计说明

以填写模块说明表的形式，对每个模块给出下述内容：

(1) 模块的一般说明，包括名称、编号、设计者、所在文件、所在库、调用本模块的模块名和本模块调用的其他模块名；

(2) 功能概述；

(3) 处理描述，使用伪码描述本模块的算法、计算公式及步骤；

(4) 引用格式；

(5) 返回值；

(6) 内部接口，说明本软件内部各模块间的接口关系，包括：

(a) 名称，

(b) 意义，

(c) 数据类型，

(d) 有效范围，

(e)I／O 标志；

(7) 外部接口，说明本软件同其他软件及硬件间的接口关系，包括：

(a) 名称，

(b) 意义，

(C) 数据类型，

(d) 有效范围，

(e)I／O 标志，

(f) 格式，指输入或输出数据的语法规则和有关约定，

(g) 媒体；

(8) 用户接口，说明将向用户提供的命令和命令的语法结构，以及软件的回答信息，包括：

(a) 名称，

(b) 意义，

(C) 数据类型，

(d) 有效范围，

(e)I／O 标志，

(f) 格式，指输入或输出数据的语法规则和有关约定，

附:模块说明表

模块说明表　　制表日期:　　年　月　日

模块名:	模块编号:	设计者:
模块所在文件:	模块所在库:	
调用本块的模块名:		
本模块调用的其他模块名:		
功能概述:		
处理描述:		
引用格式:		
返回值:		

续表

	名　称	意　义	数据类型	数值范围	I/O 标志
内部接口					

	名　称	意　义	数据类型	I/O 标志	格　式	媒　体	
外部接口							
用户接口							

4 软件设计评审

4.1 设计评审（design review）

设计评审（Design Review），就是对设计文档的评审。对于软件设计来说，评审与其技术设计方法本身是一样重要，评审对于研制项目的成功而言是绝对必要的。对设计进行评审是为了尽早发现软件的欠缺，尽可能把这些缺欠在进入下一阶段工作之前，予以纠正，从而避免后期付出更多的代价。

4.2 设计评审方法

目前存在着两种不同的设计评审方法：

- 非正式评审
- 正式技术评审

4.3 . 软件设计评审的指南

- 概要设计评审和详细设计评审应该分开进行，不允许合并为一次复审
  - 概要设计评审评价从需求到设计数据和体系结构的变换
  - 详细设计评审，通常叫详细设计走查（walkthrough），注重算法过程的正确性
- 建立一个议事日程并遵循它
- 评审设计文档，不评审设计者
- 评审中提出的问题应详细记录，但不要谋求当场解决。

- 限制参与人数和坚持充分准备
  - 除软件开发人员外，概要设计评审必须有用户代表参加，必要时还可邀请有关领域的专家到会
  - 详细设计评审一般不邀请用户和其他领域的代表。
- 为设计文档开发一个检查表，以帮助评审人员集中在重要问题上
- 为了提高评审的效率，所有评审的参加者应接受一定的正规的培训
- 评审结束前，应作出本次评审能否通过的结论

4.4 评审检查表

概要设计评审检查表如下：

- 软件体系结构是否反映了软件需求？
- 达到高的模块化吗？模块功能独立吗？
- 模块与外部系统元素接口定义了吗？
- 数据结构与软件需求一致吗？
- 考虑了可维护性吗？
- 是否直接评价了质量因素？

详细设计评审检查表如下：

- 算法能完成所要求的功能吗？
- 算法逻辑正确吗？
- 接口与体系结构设计一致吗？
- 逻辑的复杂性合理吗？
- 是否规定了错误处理和反故障处理？
- 正确地定义了局部数据结构吗？
- 都使用了结构化变成构造吗？
- 设计的细节适用于实现语言吗？
- 用的是哪个操作系统或语言独立性质？
- 考虑到可维护性吗？

结构化方法总结

1） 结构化方法的世界观

结构化方法看待世界的基本观点：一切系统都是由信息流构成的 (其中包含一些必要的数据变换), 每一个信息流都有自己的起点 - 数据源，有自己的归宿 - 数据潭，有驱动信息流动的加工，因此所谓信息处理主要表现为信息的流动。

2） 基于的基本原理 / 原则

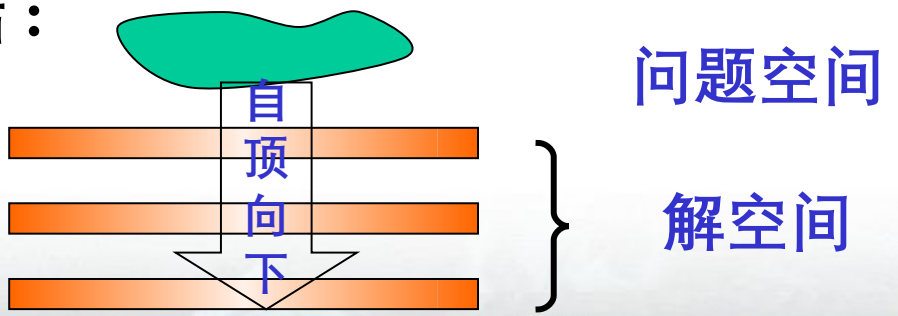
- ♦ 自顶向下功能分解
- ♦数据抽象
- ♦ 功能 / 过程抽象
- ♦模块化；

……

3）结构化方法是一种系统化的软件系统建模方法，从测试的角度看，结构化方法是一种特定的建立验证和确认所需标尺的方法学，包括结构化分析和结构化设计。

结构化方法的抽象层，包括：

- ♦ 需求分析层
- ♦ 设计层
- ♦ 实现层



4）该方法的组成

紧紧围绕“自顶向下”“过程抽象”“数据抽象”和“模块化”等基本原理 / 原则

给出了　完备的符号

可操作的过程

易理解的表示工具

并提供了　控制信息组织复杂性的机制，例如

逐层分解，数据打包等

以支持将问题空间的一个问题映射为解空间的一个解

5）问题

- ♦捕获的“过程”和“数据” 恰恰是客观事物的易变性质
- ♦解的结构没有保持原系统的结构

从而：造成　维护，验证上的困难。