

软件工程期中复习提纲

第一章 软件工程学概述	- 1 -
一、概念	- 1 -
软件	- 1 -
软件危机 (书 P1)	- 1 -
软件工程 (书 P5)	- 1 -
生命周期方法学 (书 P11)	- 1 -
二、软件危机	- 2 -
软件危机产生的原因	- 2 -
消除软件危机的途径	- 2 -
三、软件生命周期	- 2 -
三个时期 (书 P11)	- 2 -
八个阶段 (书 P12)	- 2 -
四、软件过程模型	- 2 -
软件过程模型的种类 (书 P14)	- 2 -
统一软件开发过程 (RUP) (书 P22)	- 4 -
第二章 可行性研究	- 5 -
一、可行性研究任务 (书 P35)	- 5 -
任务和目的	- 5 -
可行性研究的内容	- 5 -
二、可行性研究过程 (书 P36)	- 5 -
三、系统流程图 (书 P38)	- 6 -
概念	- 6 -
符号 (书 P39)	- 6 -
四、数据流图 (书 P40)	- 7 -
概念	- 7 -
符号 (书 P41)	- 7 -
指导规则	- 7 -
数据流图的层次结构	- 8 -
五、数据字典 (书 P47)	- 9 -
概念	- 9 -
内容 (书 P47)	- 9 -
方法和符号 (书 P47)	- 9 -
构造准则	- 9 -
六、成本/效益分析 (书 P49)	- 10 -
概念	- 10 -
成本估计 (人力成本估计) (书 P50)	- 10 -
成本/效益分析方法 (书 P51)	- 10 -
第三章 需求分析	- 12 -
一、需求分析的任务 (书 P56)	- 12 -
基本任务	- 12 -
具体任务 (书 P56)	- 13 -
二、需求分析的过程和原则	- 13 -
三、分析建模与规格说明 (书 P62)	- 14 -
分析建模	- 14 -

需求分析建模之结构化分析.....	- 15 -
软件需求规格说明 (SRS)	- 15 -
四、实体-联系图 (E-R 模型) (书 P62)	- 16 -
概念.....	- 16 -
符号 (书 P64)	- 16 -
五、数据规范化 (书 P64)	- 16 -
目的.....	- 17 -
方式.....	- 17 -
范式.....	- 17 -
六、状态转换图 (书 P65)	- 17 -
概念.....	- 17 -
状态.....	- 18 -
事件.....	- 18 -
符号 (书 P66)	- 18 -
七、状态转换图 (书 P67)	- 18 -
层次方框图.....	- 18 -
Warnier 图.....	- 18 -
IPO 图.....	- 19 -
一种改进的 IPO 图 (也称为 IPO 表)	- 19 -
八、加工说明 (需求规格说明) (?)	- 19 -
定义.....	- 19 -
内容.....	- 19 -
描述方法.....	- 19 -
九、验证软件需求 (书 P70)	- 19 -
第五章 总体设计	- 20 -
一、结构化设计方法的概念.....	- 20 -
二、结构化设计原理 (书 P94)	- 21 -
1. 模块化.....	- 21 -
2. 抽象.....	- 21 -
3. 逐步求精.....	- 21 -
4. 信息隐藏和局部化.....	- 22 -
5. 模块独立性.....	- 22 -
三、结构化设计原理——模块独立性 (书 P97)	- 22 -
耦合.....	- 22 -
内聚.....	- 23 -
四、设计原理与启发规则 (书 P99)	- 23 -
1. 改进软件结构提高模块独立性.....	- 23 -
2. 模块规模应该适中.....	- 24 -
3. 深度、宽度、扇出和扇入应适当.....	- 24 -
4. 模块作用域应在控制域内.....	- 25 -
5. 降低模块接口复杂程度.....	- 26 -
6. 设计单入口、单出口模块.....	- 26 -
7. 模块功能可预测.....	- 26 -
五、描绘软件结构的图形工具 (书 P102)	- 26 -
层次图和 HIPO 图.....	- 26 -
结构图.....	- 26 -
六、面向数据流的设计方法 (书 P104)	- 26 -
概念.....	- 26 -

信息流类型.....	- 26 -
变换分析 (书 P105)	- 27 -
事务分析 (书 P111)	- 28 -
第六章 详细设计	- 29 -
一、详细设计的概念 (书 P117)	- 29 -
二、结构化程序设计 (书 P117)	- 29 -
三、人机界面设计 (书 P119)	- 29 -
设计问题.....	- 29 -
设计指南 (书 P112)	- 30 -
四、过程设计工具 (书 P124)	- 31 -
过程设计.....	- 31 -
程序流程图 (书 P124)	- 31 -
盒图 (N-S 图) (书 P125)	- 32 -
PAD 图 (书 P126)	- 33 -
判定表 (书 P127)	- 34 -
判定树 (书 P128)	- 34 -
结构化语言 (伪码 PDL) (书 P128)	- 34 -
五、面向数据结构的设计方法 (书 P129)	- 34 -
概念.....	- 34 -
Jackson 方法 (书 P130)	- 34 -
六、程序复杂程度的定量度量 (书 P136)	- 35 -
概念.....	- 35 -
McCabe 方法 (书 P137)	- 35 -
Halstead 方法 (书 P139)	- 36 -

第一章 软件工程概述

一、概念

软件

软件是计算机系统中与硬件相互依存的另一部分，它包括程序、数据及相关文档的完整集合。其中，程序是按事先设计的功能和性能要求执行的指令序列；数据是使程序能正常操纵信息的数据结构；文档是与程序开发、维护和使用有关的图文材料。

软件危机（书 P1）

软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。

软件工程（书 P5）

软件工程是采用工程的概念、原理、技术和方法来指导软件开发和维护的工程学科。

软件工程有 4 个关键元素：方法（method）、语言（languages）、工具（tools）和过程（procedures）。

方法：提供如何构造软件的技术。例如，结构化开发方法、原型化开发方法、面向对象的开发方法等。

语言：支持软件的分析设计和实现。例如，传统的编程语言、规格说明语言和设计语言，近年来原型开发语言也得到了发展。

工具：为方法和语言提供自动化或半自动化的支持。软件工具种类繁多，涉及面广。

过程：过程如同黏结剂（glue）。它把方法、语言和工具黏结在一起，它能使软件开发理性化和适时化。

软件工程基本原则之一，应该根据软件项目的性质和实际情况等因数，选择合适的软件开发模型。

生命周期方法学（书 P11）

生命周期方法学是软件工程的传统途径。

从时间角度对软件开发和维护的复杂问题进行分解，划分为若干个阶段，每个阶段有相对独立的任务，是在阶段结束时进行技术审查和管理复审，最后产生相应的文档资料。

软件生命周期由软件定义、软件开发和运行维护（也称为软件维护）三个时期组成，每个时期又进一步划分成若干个阶段（八个阶段）。

软件生命周期每个阶段的基本任务：

1. 问题定义
2. 可行性研究
3. 需求分析：需求规格说明书（SRS）
4. 总体设计（概要设计）
5. 详细设计（模块设计）
6. 编码和单元测试

7. 综合测试
8. 软件维护

二、软件危机

软件危机产生的原因

客观上与软件自身的特点有关(逻辑实体、手工开发、复杂度高、成本昂贵):

1. 软件开发进展情况较难衡量
 2. 软件开发质量难以评价
 3. 管理和控制软件开发过程相当困难
 4. 软件没有“磨损”概念, 软件维护通常意味着改进或修改原来的设计
- 主观上与开发、维护方法不正确有关(忽视用户需求, 轻视软件维护):
1. 程序员具有编程能力, 但对软件开发这一过程缺乏足够的工程化思维
 2. 对软件开发的一些认识的误区: 软件神话
 3. 没有将“软件产品研发”与“程序编码”区分清楚
 4. 忽视需求分析、轻视软件维护

消除软件危机的途径

技术措施: 方法和工具。

组织管理措施: 从管理角度进行审查、控制。

软件工程正是从**技术和管理两方面**研究如何更好地开发和维护计算机软件的一门新兴学科。

三、软件生命周期

三个时期 (书 P11)

1. 软件定义
2. 软件开发
3. 软件维护

八个阶段 (书 P12)

1. 问题定义
2. 可行性研究
3. 需求分析: 需求规格说明书 (SRS)
4. 总体设计 (概要设计)
5. 详细设计 (模块设计)
6. 编码和单元测试
7. 综合测试
8. 软件维护

四、软件过程模型

软件过程模型的种类 (书 P14)

- 瀑布模型 (书 P15)

核心：将功能的实现与设计分开

优点（文档驱动）：

可强迫开发人员采用规范的方法（例如，结构化技术）；

严格地规定了每个阶段必须提交的文档；

要求每个阶段交出的所有产品都必须经过质量保证小组的仔细验证。

存在的问题：

过于理想化：人在工作过程中不可能不犯错误。在设计阶段可能发生规格说明文档中的错误。而设计上的缺陷或错误可能在实现过程中显现出来。在综合测试阶段将发现需求分析、设计或编码阶段的许多错误。

局限性：

项目开始阶段，开发人员和用户对项目的描述往往是不全面的，可能影响到后面各个阶段。

瀑布模型是**以文档形式驱动的**，系统修改维护难度较大。

开发过程中，**事先选择的技术或需求迅速发生变化**，需要返回到前面某阶段进行修改。

➤ 原型模型（书 P16）

快速原型模型：在用户**不能给出完整、准确的需求说明**，或者开发者**不能确定算法的有效性、操作系统的适应性或人机交互的形式等**许多情况下，可以根据用户的一组基本需求，快速建造一个**原型（可运行的软件）**，然后进行评估，进一步精化、调整原型，使其满足用户的要求，也使开发者对将要做的事情有更好的理解。

存在的问题：

1. 为了使原型尽快的工作，没有考虑软件的总体质量和长期的可维护性。
2. 为了演示，可能采用不合适的操作系统、编程语言、效率低的算法，这些不理想的选择成了系统的组成部分。
3. 开发过程不便于管理。

有效的使用原型模式：

建造原型仅是为了定义需求，之后就被抛弃（或被部分抛弃），实际的软件在充分考虑了质量和可维护性之后才被开发。

➤ 螺旋模型（书 P19）

特点：

风险驱动，需要相当丰富的风险评估经验和专门知识，否则风险更大。

主要适用于内部开发的大规模软件项目，随着过程的进展演化，开发者和用户能够更好的识别和对待每一个演化级别上的风险。

随着迭代次数的增加，工作量加大，软件开发成本增加。

优点：

1. 对可选方案和约束条件的强调有利于已有软件的重用，也有助于把软件质量作为软件开发的一个重要目标。
2. 减少了过多测试或测试不足。
3. 维护和开发之间并没有本质区别。

➤ 喷泉模型（书 P21）

特点：主要用于支持面向对象开发过程体现了软件创建所固有的迭代和无间隙的特征。

➤ 智能模型

统一软件开发过程（RUP）（书 P22）

是由 Rational 公司的 Booch、Jacobson、Rumbaugh 提出的软件过程模型，也称 **RUP**（Rational Unified Process）。RUP 重复一系列周期，每个周期由一个交付给用户的产品结束。

每个周期划分为**初始、细化、构造和移交四个阶段**，每个阶段围绕着**五个核心工作流（需求、分析、设计、实现、测试）**分别迭代。

初始阶段：进行问题定义，确定目标，评估其可行性，降低关键风险。

细化阶段：制定项目计划、配置各类资源、建立系统架构（包括各类视图）。

构造阶段：开发整个产品，并确保产品可移交给用户。

移交阶段：产品发布、安装、用户培训。

在每个阶段的每次迭代的最后，用例模型、分析模型、设计模型、实现模型都会增量，每个阶段结束的里程碑处，管理层做出是否继续、进度、预算、是否给下一阶段提供资助等决定。

不同阶段工作流的侧重点不同，前两阶段大部分工作集中在需求、分析和架构设计上；在构造阶段，重点转移到详细设计、实现和测试上。

第二章 可行性研究

一、可行性研究任务（书 P35）

任务和目的

目的：

用最小的代价在尽可能短的时间内确定问题是否能够解决（不是解决问题，而是确定问题是否值得去解决）。

任务：

可行性研究的主要任务是“了解客户的要求及现实环境，从技术、经济和社会因素等三方面研究并论证本软件项目的可行性，编写可行性研究报告，制定初步项目开发计划。”

根本任务：对以后的行动方针提出建议。

可行性研究的内容

1. 技术可行性

度量一个特定技术信息系统解决方案的实用性及技术资源的可用性。

考虑的问题：

- （1）开发风险分析
- （2）资源分析
- （3）相关技术的发展（现有技术能否实现新系统，技术难点、建议采用技术的先进性）

2. 经济可行性

度量系统解决方案的性能价格比。

考虑的问题：

- （1）成本/效益分析（开发、运行的成本/效益）
 - 1）有形成本、效益
 - 2）无形成本、效益
- （2）价值和成本的关系
 - 1）质量与价值、成本的关系
 - 2）价值/成本的均衡

3. 操作可行性

考虑的问题：

- （1）用户使用可能性
- （2）时间进度可行性
- （3）组织和文化上的可行性
- 4. 社会可行性（法律可行性）
- 5. 抉择

二、可行性研究过程（书 P36）

- 1. 复查系统规模和目标
- 2. 研究目前正在使用的系统
- 3. 导出新系统的高层逻辑模型

4. 重新定义问题
5. 导出和评价供选择的解法
6. 推荐行动方针
7. 草拟开发计划
8. 书写文档并提交审查

编写《可行性研究报告》

三、系统流程图（书 P38）

概念

系统流程图是概括地描绘物理系统的传统工具。

它的基本思想是用图形符号以黑盒子形式描绘组成系统的每个部件（程序，文档，数据库，人工过程等）。

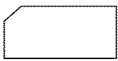
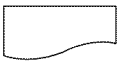

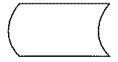






系统流程图表达的是数据在系统各部件之间流动的情况，而不是对数据进行加工处理的控制过程，因此尽管系统流程图的某些符号和程序流程图的符号形式相同，但是它却是物理数据流图而不是程序流程图。

符号（书 P39）

基本符号：

符 号	名 称	说 明
	处理	能改变数据值或数据位置的加工或部件，例如程序、处理机、人工加工等都是处理
	输入输出	表示输入或输出（或既输入又输出），是一个广义的不指明具体设备的符号
	连接	指出转到图的另一部分或从图的另一部分转来，通常在同一页上
	换页连接	指出转到另一页图上或由另一页图转来
	数据流	用来连接其他符号，指明数据流动方向

系统符号：

符 号	名 称	说 明
	穿孔卡片	表示用穿孔卡片输入或输出，也可表示一个穿孔卡片文件
	文档	通常表示打印输出，也可表示用打印终端输入数据
	磁带	磁带输入输出，或表示一个磁带文件
	联机存储	表示任何种类的联机存储，包括磁盘、磁鼓、软盘和海量存储器件等
	磁盘	磁盘输入输出，也可表示存储在磁盘上的文件或数据库
	磁鼓	磁鼓输入输出，也可表示存储在磁鼓上的文件或数据库
	显示	CRT 终端或类似的显示部件，可用于输入或输出，也可既输入又输出
	人工输入	人工输入数据的脱机处理，例如填写表格
	人工操作	人工完成的处理，例如会计在工资支票上签名
	辅助操作	使用设备进行的脱机操作
	通信链路	通过远程通信线路或链路传送数据

四、数据流图（书 P40）

概念

一种图形化技术，它描绘信息流和数据从输入移动到输出的过程中所经受的变换。

在数据流图中没有任何具体的物理部件，它只是描绘数据在软件中流动和被处理的逻辑过程，是系统逻辑功能的图形表示（用途）。

设计数据流图时只需考虑系统必须完成的基本逻辑功能，完全不需要考虑怎样具体地实现这些功能，所以它也是今后进行软件设计的很好的出发点。

符号（书 P41）

指导规则

1. 第一层 DFD 应当是基本系统模型
2. 注意父图和子图的平衡，维护信息的连续性
3. 区分局部文件和局部外部项
4. 掌握分解的速度，上快下慢
5. 遵守加工编号原则

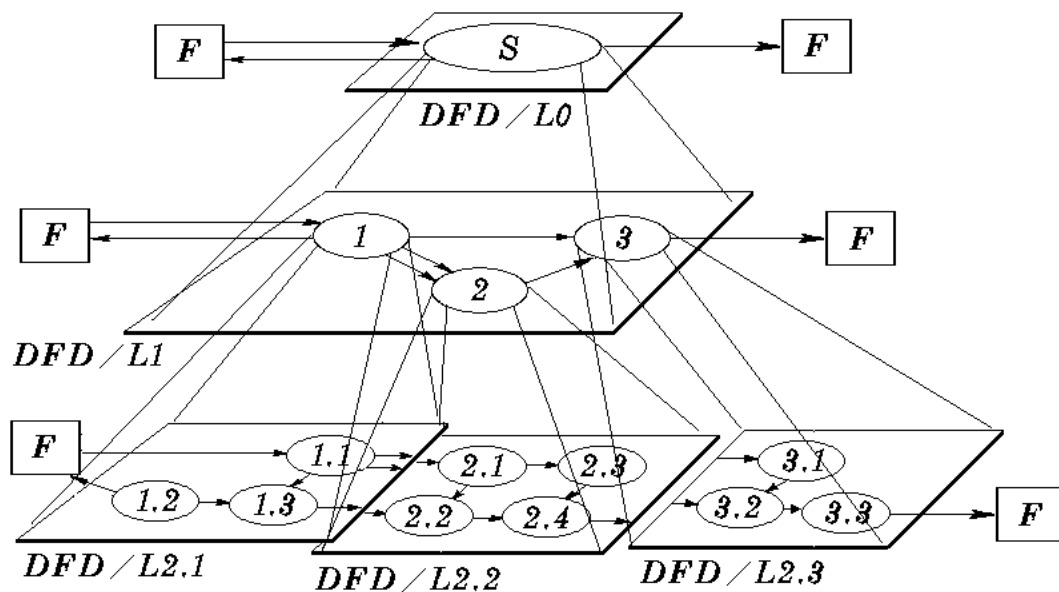
数据流图的层次结构

为了表达数据处理过程的数据加工情况，需要采用层次结构的数据流图。按照系统的层次结构进行逐步分解，并以分层的数据流图反映这种结构关系，能清楚地表达和容易理解整个系统。

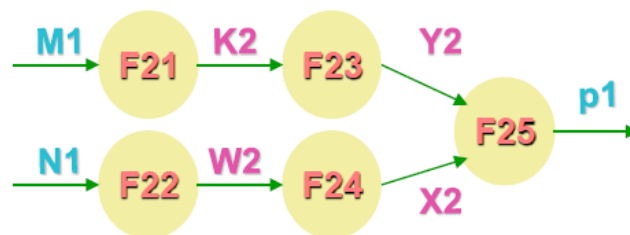
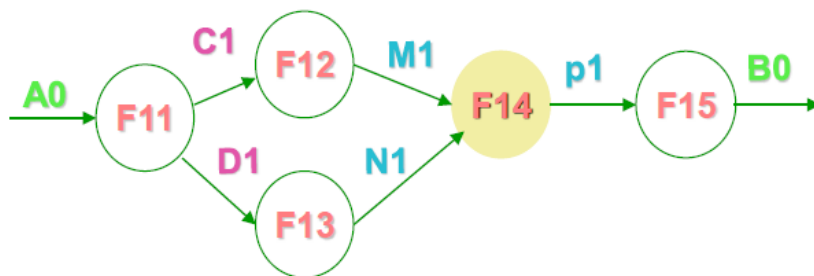
在多层数据流图中，顶层流图仅包含一个加工，它代表被开发系统。它的输入流是该系统的输入数据，输出流是系统所输出数据。

底层流图是指其加工不需再做分解的数据流图，它处在最底层。

中间层流图则表示对其上层父图的细化。它的每一加工可能继续细化，形成子图。



第 n 层



第 n+2 层

分层 DFD 图的优点：

便于实现：采用逐步细化的扩展方法，可避免一次引入过多的细节，有利于控制问题的复杂度。

便于使用：用一组图代替一张总图，方便用户及软件开发人员阅读。

五、数据字典（书 P47）

概念

数据字典的任务是：对于数据流图中出现的所有被命名的图形元素在字典中作为一个词条加以定义，使得每一个图形元素的名字都有一个确切的解释。

数据字典是关于数据的信息的集合，是对 DFD 中的所有元素定义的集合。数据流图和数据字典共同构成系统的逻辑模型。没有数据字典数据流图就不严格，没有数据流图数据字典也难于发挥作用。

内容（书 P47）

一般说来，数据字典应该由对下列 4 类元素的定义组成：

- （1）数据流
- （2）数据流分量（即数据元素）
- （3）数据存储
- （4）处理

方法和符号（书 P47）

构造准则

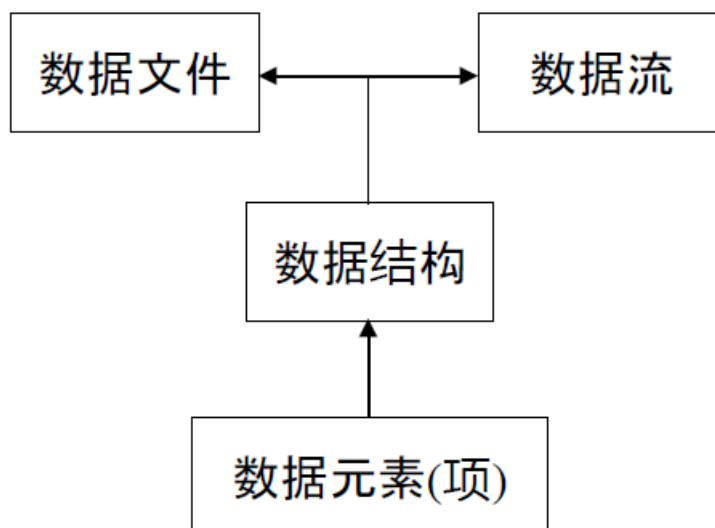
（1）数据流图中出现的名字都应编制一个数据条目。而且只能有一个条目，不能重复定义。

（2）应按自底向上的方式对数据进行定义，先定义数据元素，后定义数据结构，再定义数据流或数据文件。

（3）定义时所用的词汇都应是有明确的含义，只能有一种理解。

（4）条目应有序编排。

数据组成的层次关系：



六、成本/效益分析（书 P49）

概念

目的：从经济角度分析开发一个特定的新系统是否划算，从而帮助客户组织的负责人正确地作出是否投资于这项开发工程的决定。

成本估计（人力成本估计）（书 P50）

软件开发成本主要表现为人力消耗（乘以平均工资则得到开发费用）。

估算技术：

1. 代码行技术

根据经验和历史数据估计实现一个功能需要的源程序行数，用每行代码的平均成本乘以行数就可以确定软件的成本。

每行代码的平均成本主要取决于软件的复杂程度和工资水平。

代码行技术是比较简单的定量估算方法。当有以往开发类似工程的历史数据可供参考时，这个方法是非常有效的。

2. 任务分解技术

首先把软件开发工程分解为若干个相对独立的任务。

再分别估计每个单独的开发任务的成本，最后累加起来得出软件开发工程的总成本。

估计每个任务的成本时，通常先估计完成该项任务需要用的人力（以人月为单位），再乘以每人每月的平均工资而得出每个任务的成本。

3. 自动估计成本技术

采用自动估计成本的软件工具可以减轻人的劳动，并且使得估计的结果更客观。但是采用这种技术必须有长期搜集的大量历史数据为基础，并且需要有良好的数据库系统支持。

成本/效益分析方法（书 P51）

成本/效益分析的第一步是估计开发成本、运行费用和新系统将带来的经济效益，然后从经济角度判断这个系统是否值得投资。

运行费用取决于系统的操作费用（操作员人数，工作时间，消耗的物资等）和维护费用。

系统的经济效益等于因使用新系统而增加的收入加上使用新系统可以节省的运行费用。

1. 货币的时间价值

通常用利率表示货币的时间价值。设年利率为 i ，现已存入 P 元，则 n 年后可得钱数为 $F=P(1+i)^n$ ，这就是 P 元钱在 n 年后的价值。反之，若 n 年后能收入 F 元，那么这些钱现在的价值是 $P=F/(1+i)^n$ 。

2. 投资回收期

投资回收期是衡量一个开发工程价值的经济指标。投资回收期就是积累的经济效益等于最初的投资所需要的时间。投资回收期越短，就能越快获得利润。

3. 纯收入

工程的纯收入是衡量工程价值的另一项经济指标。纯收入就是在整个生存周期之内系统的累计经济效益（折合成现在值）与投资之差。

如果纯收入小于零，那么显然这项工程不值得投资。只有当纯收入大于零，才能考虑投资。

4. 投资回收率 ROI

投资回报率=利润/总量*100%

衡量投资效益的大小是衡量工程效益时的最重要的参考依据。

第三章 需求分析

需求分析的重要性：

开发软件系统最困难的部分就是准确说明开发什么，最困难的概念性工作是编写出详细需求，包括所有面向用户、面向机器和其它软件系统的接口。

此工作一旦做错，将会给系统带来极大损害，并且以后对它修改也极为困难。

需求分析的定义：

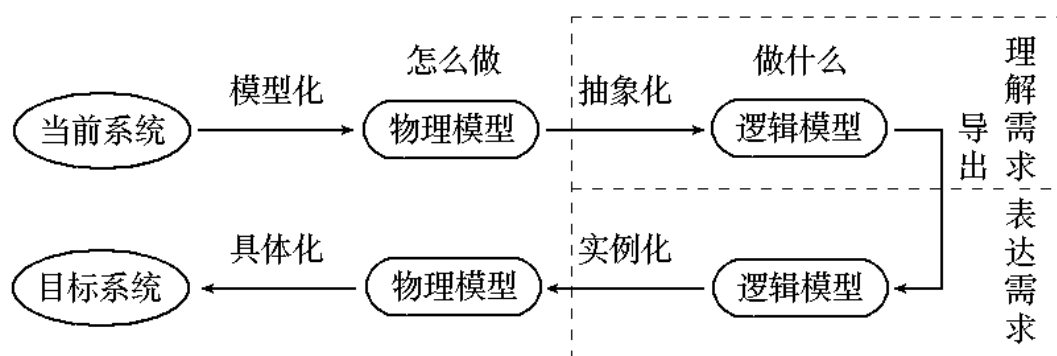
需求分析是软件定义时期的最后一个阶段，它的基本任务不是确定系统怎样完成它的工作，而是确定系统必须完成哪些工作，也就是对目标系统提出完整、准确、清晰、具体的要求。

并在在需求分析阶段结束之前，由系统分析员写出**软件需求规格说明书**，以书面形式准确地描述软件需求。

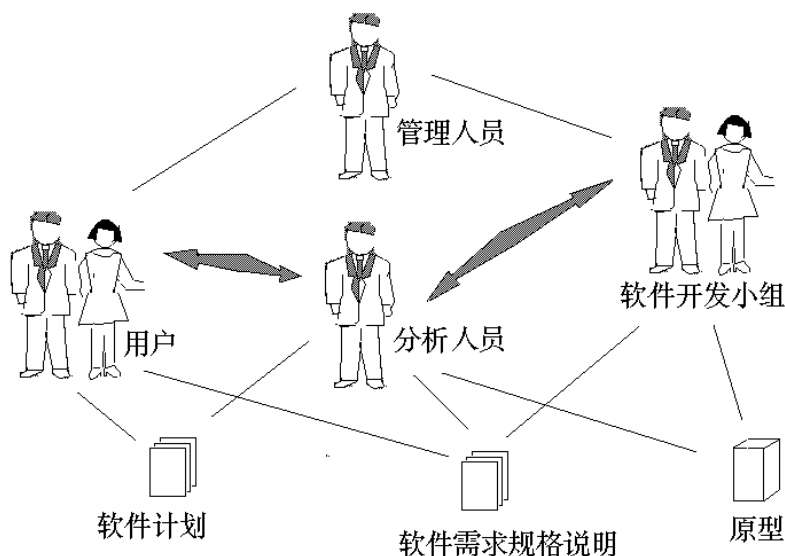
一、需求分析的任务（书 P56）

基本任务

需求分析的任务就是借助于当前系统的逻辑模型导出目标系统的逻辑模型，解决目标系统“**系统必须做什么**？”的问题，确定**目标系统功能和性能**。



在分析软件需求和书写软件需求规格说明书的过程中，分析员和用户都起着关键的、必不可少的作用。



具体任务（书 P56）

1. 确定对系统的综合要求

- (1) 功能要求
- (2) 性能要求
- (3) 可靠性和可用性需求
- (4) 运行要求
- (5) 未来可能的扩充要求

软件需求的组成：

业务需求：反映组织机构和客户对系统、产品高层次的目标要求。

用户需求：从用户使用的角度给出需求的描述。

系统需求：从系统的角度描述要提供的服务以及所受到的约束。

功能性需求：描述系统应该做什么，即为用户和其它系统完成的功能、提供的服务。

非功能性需求：产品必须具备的属性或品质。

设计约束：设计与实现必须遵循的标准、约束条件。如运行平台、协议、选择的技术、编程语言和工具等。

软件需求的描述：

结构化语言、面向对象、PDL

图形化表示

数学描述（形式化语言描述）

2. 分析系统的数据要求：

建立概念模型：E-R 图（概念模型）。

形象描绘数据结构：层次方框图、Warnier 图、IPO 图

数据结构规范化

3. 导出系统的逻辑模型：

数据流图，数据字典，加工处理说明书，处理算法等描述目标系统的逻辑模型等。

4. 需求分析评审：

目的：发现需求分析的错误和缺陷，修改开发计划

5. 修正系统开发计划：

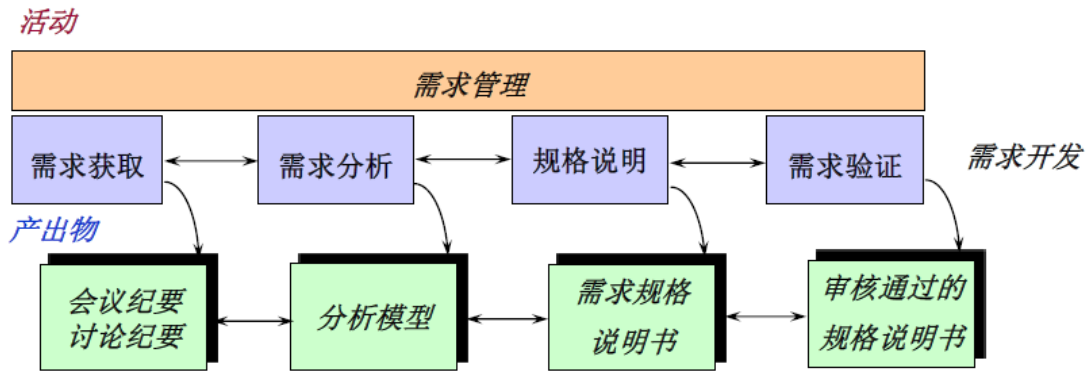
重估成本和进度

5. 开发原型系统：

使用户对目标系统有一个更直接、更具体的概念，从而能更准确提出用户需求（关键的困难在于成本）。

二、需求分析的过程和原则

过程：



需求获取是开发人员与客户或用户一起对应用领域进行调查研究，收集系统需求的过程。

需求分析是将获取到的需求准确的理解、求精，并将其转化为完整的需求定义（包括建模），进而生成需求规约的过程。

需求分析的原则：

必须能够表达和理解问题的数据域和功能域。

必须按自顶向下，逐层分解的方式对问题进行分解和不断细化（细化数据流程图）。

要给出系统的逻辑视图和物理视图。

三、分析建模与规格说明（书 P62）

分析建模

1. 分析建模

模型：

为了理解事物而对事物做出的一种抽象，是对事物的一种无歧义的书面描述。通常，由一组图形符号和组织这些符号的规则组成。

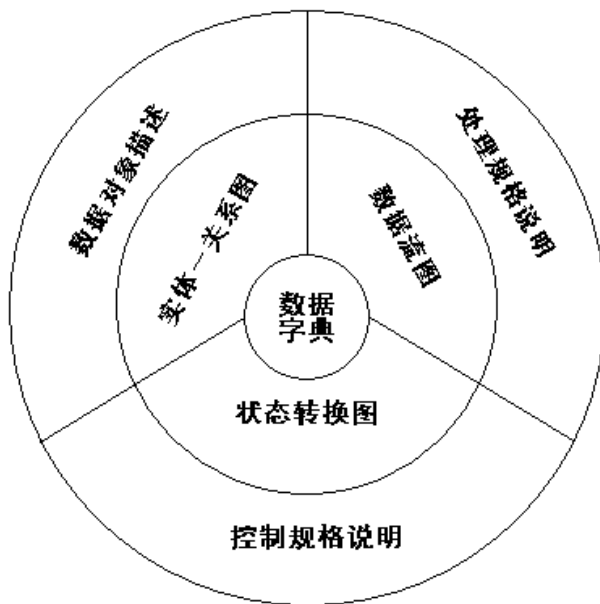
建模方法：

结构化分析（Structured Analysis, SA），70 年代末由 DeMarco 等人提出。该方法不是被所有的使用者一致地使用的单一方法，众多科学家对其进行了扩充，因此它是发展了超过 30 年的一个混合物。

面向对象分析。

需求分析建模之结构化分析

结构化分析模型：



具体步骤：

模型核心（数据字典）：描述软件使用和产生的所有数据对象。

数据模型（E-R 图表达）：描述数据对象间关系，图中数据对象属性用“数据对象描述”表达。

功能模型（DFD 表达）：描绘数据在软件中移动、变换及相应功能，图中功能用“处理规格说明”表达。

行为模型（状态转换图）：描绘系统状态和在不同状态间转换方式。图中软件控制附加信息用“控制规格说明”表达。

规格说明：书写软件需求规格说明，作为分析阶段最终成果。

复审

方法：

- (1) 信息域，数据模型（ER 图）
- (2) 软件功能，功能模型（数据流图）
- (3) 软件行为，行为模型（状态/活动图）
- (4) 分层细化

软件需求规格说明（SRS）

概念：

软件需求规格说明（Software Requirement Specification SRS）通常用自然语言+模型，完整、准确、具体地描述系统的数据要求、功能需求、性能需求、可靠性和可用性要求、出错处理需求、接口需求、约束、逆向需求以及将来可能提出的要求。

软件需求规格说明书，是需求分析阶段得出的最主要的文档。

作用：

- (1) 技术合同书。
- (2) 设计和编码阶段的基础。
- (3) 测试和验收目标系统的依据。

格式：

- (1) CMMI 文档格式
- (2) 国际标准化组织的文档参考格式
- (3) 国家标准的文档参考格式

其它文档：

- (1) 确认测试计划，验收标准和手续。
- (2) 初步用户手册

软件需求规格说明书组成：

- 引言

- 功能描述
 - ER 图
 - 数据描述
 - 数据流图
 - 数据字典
 - ◆ 数据结构
 - IPO 表
- 性能描述
- 质量保证
- 其他
- 参考文献

四、实体-联系图（E-R 模型）（书 P62）

概念

E-R 图是用来建立数据模型的工具。

数据模型是一种面向问题的数据模型，是按照用户的观点对数据建立的模型。它描述了从用户角度看到的数据，反映了用户的现实环境，而且与在软件系统中的实现方法无关。

数据模型中包含 3 种相互关联的信息：数据对象（**实体**）、数据对象的**属性**及数据对象彼此间相互连接的**关系（联系）**。

（1）数据对象（实体）：

数据对象是对软件必须理解的复合信息的抽象。

复合信息是指具有一系列不同性质或属性的事物，可以由一组属性来定义的实体都可以被认为是数据对象。

数据对象彼此间是有关联的。

（2）属性

属性定义了数据对象的**性质**。

必须把一个或多个属性定义为“标识符”，也就是说，当我们希望找到数据对象的一个实例时，用标识符属性作为“关键字”（通常简称为“键”）。应该根据对所要解决的问题的理解，来确定特定数据对象的一组合适的属性。

（3）关系（联系）

数据对象彼此之间相互连接的方式称为联系，也称为关系。

联系可分为以下 3 种类型：

A. 一对一联系（1：1）

B. 一对多联系（1：N）

C. 多对多联系（M：N）

联系也可能有属性。

符号（书 P64）

五、数据规范化（书 P64）

目的

- 消除数据冗余，即消除表格中数据的重复；
- 消除多义性，使关系中的属性含义清楚、单一；
- 使关系的“概念”单一化，让每个数据项只是一个简单的数或字符串，而不是一个组项或重复组；
- 方便操作，使数据的插入、删除与修改操作可行并方便；
- 使关系模式更灵活，易于实现接近自然语言的查询方式。

方式

规范化将数据的逻辑结构归结为满足一定条件的二维表（关系）。即：

1. 表格中每个信息项必须是一个不可分割的数据项，不可是组项。
2. 表格中每一列（列表示属性）中所有信息项必须是同一类型，各列的名字（属性名）互异，列的次序任意。
3. 表格中各行（行表示元组）互不相同，行的次序任意。

范式

通常用“范式（Normal Forms）”定义消除数据冗余的程度。第一范式（1NF）数据冗余程度最大，第三范式（3NF）数据冗余程度最小。

范式的特性：

1. 范式级别越高，存储同样数据就需要分解成更多张表，因此，“存储自身”的过程也就越复杂。
2. 随着范式级别的提高，数据的存储结构与基于问题域的结构间的匹配程度也随之下降，因此，在需求变化时数据的稳定性较差。
3. 范式级别提高则需要访问的表增多，因此性能（速度）将下降。

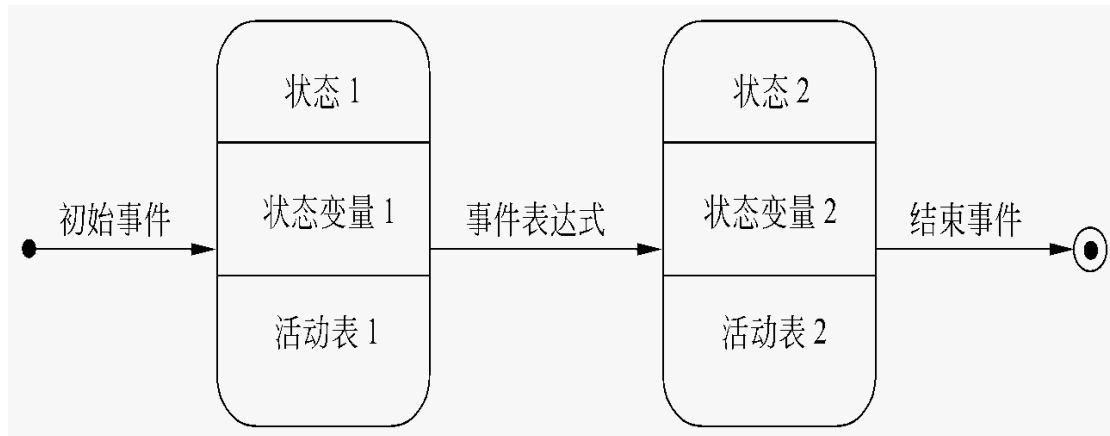
范式的分类：

- 1-NF：所有属性都是原子值，即不出现“表中有表”
- 2-NF：在 1-NF 基础上，每个非关键字都由整个关键字决定，而非依赖于关键字的一部分，即没有非主属性对码（可以理解为主键）的部分函数依赖。也就是全部都是完全函数依赖，部分函数依赖通常是因为有两个主属性。
- 3-NF：在 2-NF 基础上，非关键字之间无从属关系，即不存在非主属性对码的传递函数依赖。

六、状态转换图（书 P65）

概念

状态转换图（简称为状态图）通过描绘系统的状态及引起系统状态转换的事件，来表示系统的行为。此外，状态图还指明了作为特定事件的结果系统将做哪些动作（例如，处理数据）。



状态

状态是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式。状态规定了系统对事件的响应方式。系统对事件的响应，既可以是做一个（或一系列）动作，也可以是仅仅改变系统本身的状态，还可以是既改变状态又做动作。

状态分为：初态（即初始状态）、终态（即最终状态）和中间状态。一张状态图中只能有一个初态，而终态则可以有 0 至多个。

事件

事件是在某个特定时刻发生的事情，它是对引起系统做动作或（和）从一个状态转换到另一个状态的外界事件的抽象。例如，用户移动或点击鼠标等都是事件。

简而言之，事件就是引起系统做动作或（和）转换状态的控制信息。

符号（书 P66）

七、状态转换图（书 P67）

层次方框图

层次方框图用树形结构的一系列多层次的矩形框描绘数据的层次结构。

树形结构的顶层是一个单独的矩形框，它代表完整的数据结构，下面的各层矩形框代表这个数据的子集，最底层的各个框代表组成这个数据的实际数据元素（不能再分割的元素）。

随着结构的精细化，层次方框图对数据结构也描绘得越来越详细，这种模式非常适合于需求分析阶段的需要。系统分析员从对顶层信息的分类开始，沿图中每条路径反复细化，直到确定了数据结构的全部细节时为止。

Warnier 图

法国计算机科学家 Warnier 提出了表示信息层次结构的另外一种图形工具。

Warnier 图也用树形结构描绘信息，但是这种图形工具比层次方框图提供了更丰富的描绘手段。

用 Warnier 图可以表明信息的逻辑组织。

它可以指出一类信息或一个信息元素是**重复**出现的，也可以表示特定信息在某一类信息中是**有条件**地出现的。

重复和条件约束是说明软件处理过程的基础，所以很容易把 Warnier 图转变成软件设计的工具。

IP0 图

左边的框中列出有关的输入数据。

中间的框内列出主要的处理，处理框中列出处理的次序暗示了执行的顺序，但是用这些基本符号还不足以精确描述执行处理的详细情况。

在右边的框内列出产生的输出数据。

在 IP0 图中还用类似向量符号的粗大箭头清楚地指出数据通信的情况。

一种改进的 IP0 图（也称为 IP0 表）

在需求分析阶段可以使用 IP0 表简略地描述系统的主要算法（即数据流图中各个处理的基本算法）。

需求分析阶段，IP0 表中的许多附加信息暂时还不具备，但在设计阶段可以进一步补充修正这些图，作为设计阶段的文档。

这正是在需求分析阶段用 IP0 表作为描述算法的工具的重要优点。

八、加工说明（需求规格说明）（?）

定义

是对 DFD 中的加工所做的描述，包括：输入数据、加工逻辑、输出数据等。

内容

- 加工名称
- 加工编号
- 输入数据流
- 输出数据流
- 加工逻辑
- 执行次数

描述方法

- 结构化语言（PDL）
- 判定表
- 判定树

九、验证软件需求（书 P70）

验证软件需求的正确性，一般应从 4 个方面进行：

- （1）**一致性**：所有需求必须一致，不能互相矛盾。
- （2）**完整性**：需求必须完整，包含用户需要的所有功能和性能。
- （3）**现实性**：指定需求用现有的软、硬件技术基本上可以实现。
- （4）**有效性**：必须证明需求是正确有效的，确实能解决用户面对的问题。

第五章 总体设计

软件设计分为结构设计和详细设计两个阶段。

结构设计（总体设计/概要设计）：

制定系统实现方案和设计规范，合理确定软件系统的整体模块结构及接口关系。

详细设计（模块设计）：

详细规定每个模块功能的实现算法。

软件结构设计的方法：

结构化设计方法（面向数据流设计方法）

与结构化分析相配合的设计方法。

基本内容是遵循自顶向下的系统化分解的思想，应用一组转化规则将软件的数据流图逐步转化为一组程序模块的层次结构。

面向数据结构的设计方法

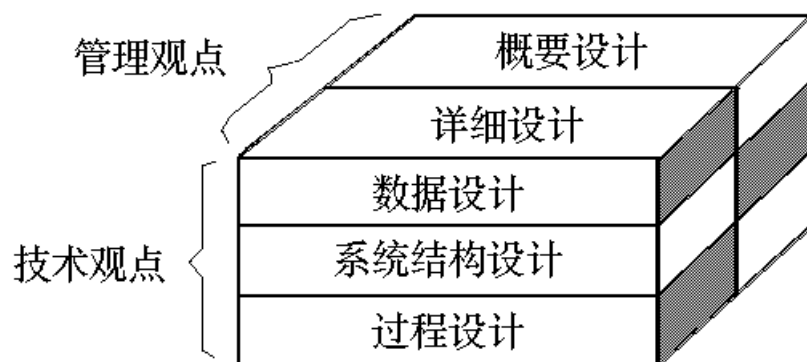
基于处理数据的控制结构与数据结构在格式上非常相似的原理。

基本内容是用表达工具描述数据结构的格式，然后应用一组规则将数据结构格式转换为程序的控制结构格式。

面向对象的设计方法

与面向对象分析方法相匹配的一种设计方法。

一、结构化设计方法的概念



从工程管理角度结构化设计分两步：

概要设计：将软件需求转化为数据结构和软件系统结构。

详细设计：过程设计，通过对结构细化，得到软件详细数据结构和算法。

从技术角度结构化设计分两步：

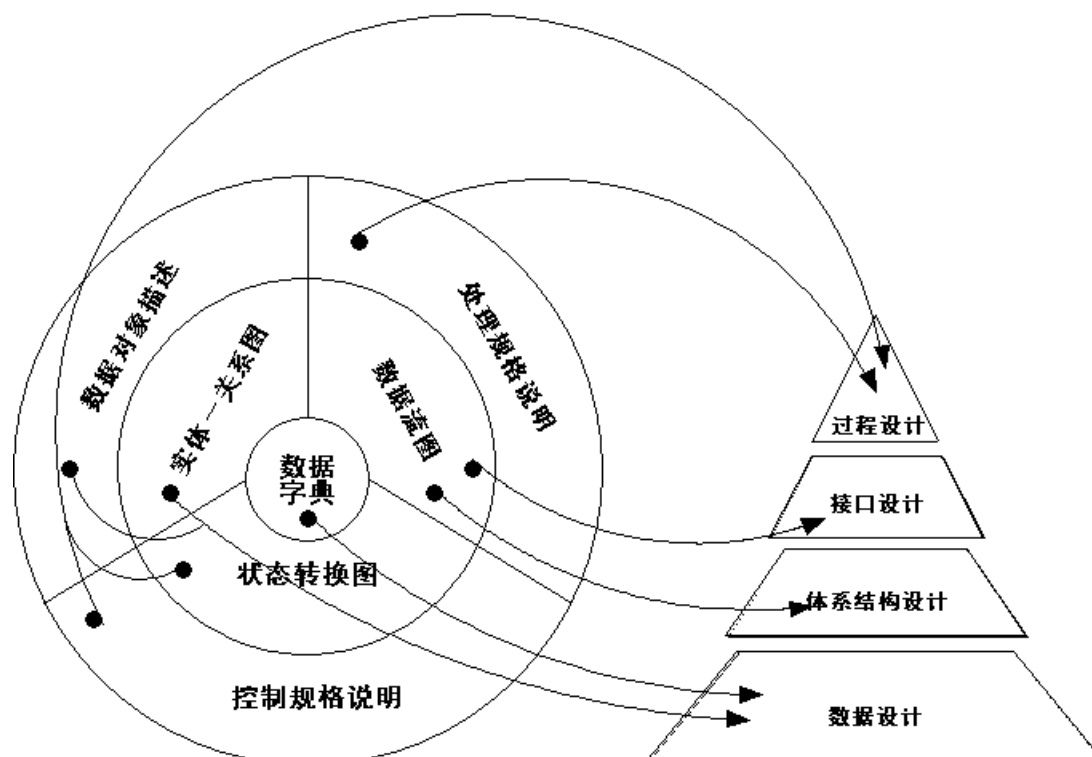
根据用信息域表示的软件需求，以及功能和性能需求，进行数据设计、系统结构设计和过程设计。

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。

结构化设计与分析关系图：



数据设计：数据模型及核心数据字典转变为数据结构。

体系结构设计：功能模型中数据流图转变成计算机模块框架。

接口设计：功能模型中数据流图转变成软件内部、软件与协作系统间、软件与用户间通信方式。

过程设计：行为模型及功能模型中的“处理规格说明”转换成软件构件过程描述。

二、结构化设计原理（书 P94）

1. 模块化

模块化是把程序按适当的原则把软件划分为一个个较小的、相关而又相对独立的模块，每个模块完成一个子功能，把这些模块集中起来组成一个整体，可以完成指定的功能，满足问题的要求。

模块（module）：“模块”又称“构件”，一般指用一个名字调用的相邻程序元素序列。

2. 抽象

抽出事物的本质特性，暂不考虑细节。

3. 逐步求精

求精是指为了能集中精力解决主要问题，尽量推迟对细节问题的考虑，实际上是一个细化过程，与抽象是互补的概念。

抽象使得设计者能够说明过程和数据，同时却忽略底层细节；求精帮助设计者在设计过程中揭示底层细节。

4. 信息隐藏和局部化

信息隐藏:

每个模块的实现细节对于其他模块来说是隐藏的。也就是说，模块中所包含的信息是不允许其他不需要这些信息的模块访问的。

每个客户只能通过接口来了解该模块，而所有的实现都隐蔽起来。

局部化:

只把一些关系密切的软件元素物理地址放得彼此靠近。如模块中的局部元素是局部化的一个例子。

局部化有助于实现信息隐藏。

5. 模块独立性

如下:

三、结构化设计原理——模块独立性（书 P97）

模块具有独立功能，且和其他模块之间没有过多的相互作用。即每个模块完成一个相对独立的特定子功能，且和其他模块之间的关系很简单。是软件划分模块时要遵守的准则，也是判断模块构造是否合理标准。

优点:

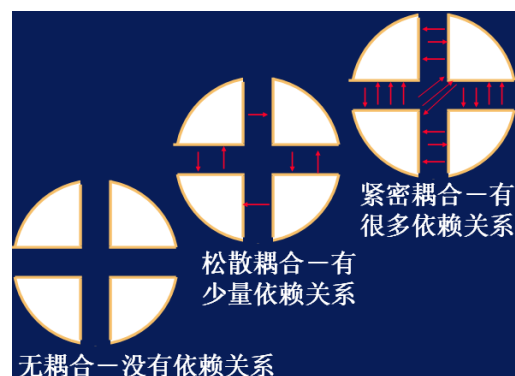
1. 容易分工合作;
2. 容易测试和维护，修改工作量较小，错误传播范围小，扩充功能容易。

两个定性度量标准:

耦合和内聚（在设计软件时追求**低耦合、高内聚**）。模块的独立性越高，其块内联系越紧密（内聚性强），块间联系越弱（耦合性越弱）。

耦合

定义：耦合是软件结构中不同模块间相对独立性（即互相连接的紧密程度）的度量，它取决模块间接口复杂程度和通过接口的数据。模块间连接越紧密，联系越多，耦合性越强。在设计时应追求尽可能**松散耦合**的系统。

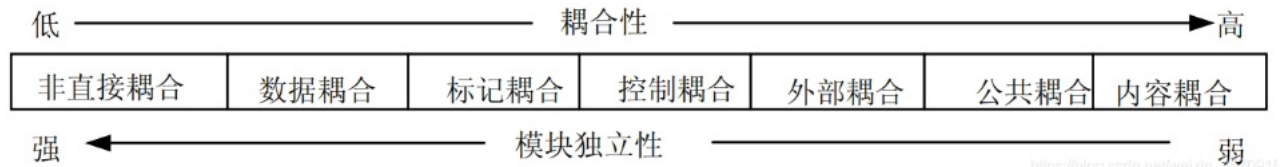


耦合的分类（耦合度从低到高）:

- 非直接耦合
两个模块分别能独立地工作不需要另一模块存在。
- 数据耦合
两模块通过参数交换数据信息。
- 控制耦合
两模块通过参数交换控制信息（包括数字形式）。
- 公共环境耦合
两个或多个模块通过一公共数据环境作用。
两种可能:
(1) 一模块送数据，另一模块取，等价数据耦合。
(2) 两模块既往公共环境送又从里面取，介于数据耦合和控制耦合之

间。

- 内容耦合
 - (1) 一模块访问另一模块内部数据;
 - (2) 一模块不通过正常入口转到另一模块内部;
 - (3) 两模块有部分程序代码重叠 (汇编程序);
 - (4) 一模块有多个入口。



原则:

尽量使用数据耦合, 少用控制耦合, 限制公共环境耦合, 完全不用内容耦合。

内聚

定义: 内聚是模块功能强度 (即一个模块内部各个元素彼此结合的紧密程度) 的度量。模块内部各元素之间联系越紧密, 内聚性越强。

内聚的分类 (内聚度从高到低):

- 功能内聚

一模块中各部分是完成某一功能必不可少组成部分。
- 顺序内聚

模块内处理元素同某功能密切相关, 顺序执行。
- 通信内聚

一模块内各功能部分都使用相同输入数据, 或产生相同输出数据。
- 过程内聚

模块内处理元素相关, 特定次序执行。如把流程图中循环部分、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

多为多功能模块, 要求所有功能在同一时间内执行。如初始化模块和终止模块及紧急故障处理模块。
- 逻辑内聚

一模块完成功能在逻辑上属相同相似一类。
- 偶然内聚

模块内各部分间没有联系, 即使有也很松散。

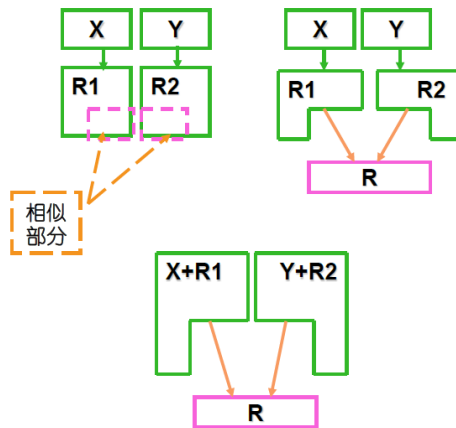
功能内聚 (10 分) > 顺序内聚 (9 分) > 通信内聚 (7 分) > 过程内聚 (5 分) > 时间内聚 (3 分) > 逻辑内聚 (1 分) > 偶然内聚 (0 分)

四、设计原理与启发规则 (书 P99)

1. 改进软件结构提高模块独立性

通过模块分解或合并, 力求降低耦合提高内聚。

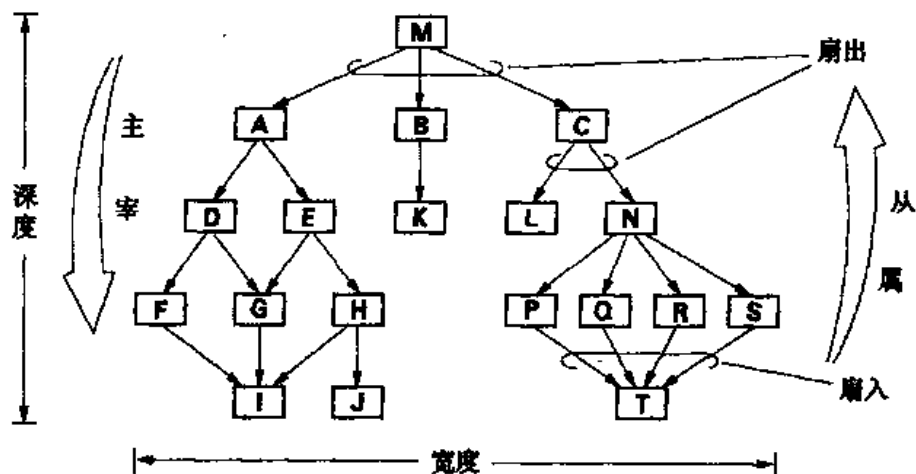
改善软件结构示意图:



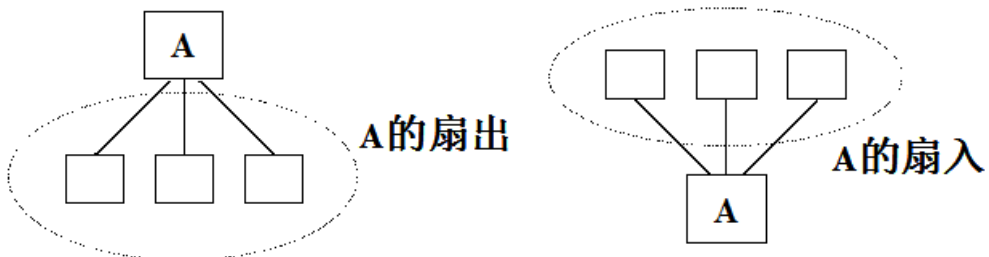
2. 模块规模应该适中

过大分解不充分，但进一步分解不应降低模块独立性。
 过小开销大于有效操作，模块数目过多系统接口复杂。
 通常语句行数在 50~100（一页纸），最多不超过 500 行。

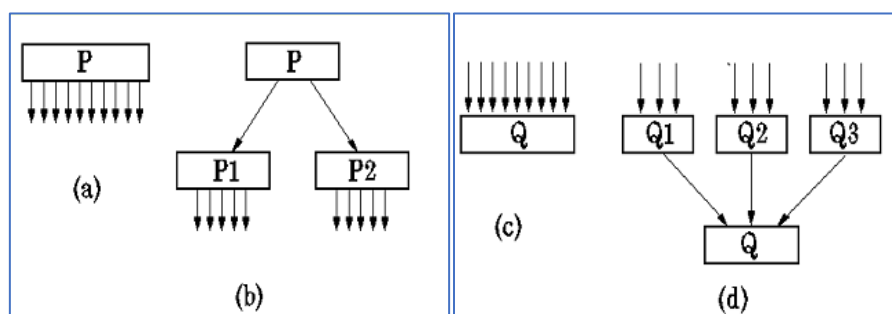
3. 深度、宽度、扇出和扇入应适当



深度：软件结构控制层数，标志一系统大小和复杂程度。
 宽度：软件结构同一层模块数最大值，越大系统越复杂。
 扇出：模块直接控制（调用）模块数。过大，模块复杂。理想扇出：3—9。
 扇入：有多少上级模块直接调用它，越大共享该模块上级模块越多。



扇出扇入改善：



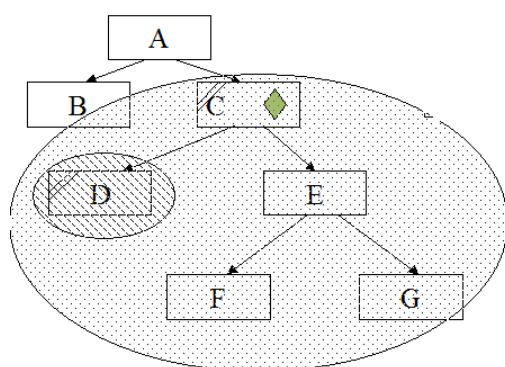
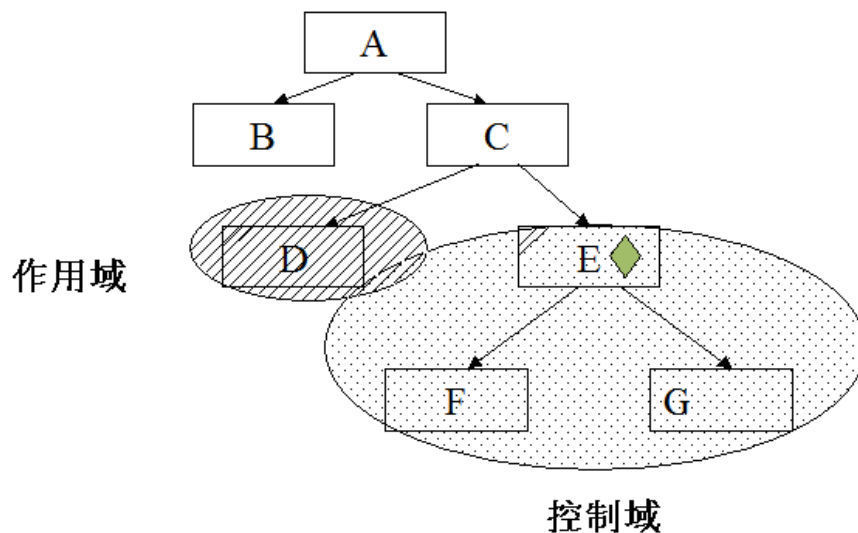
4. 模块作用域应在控制域内

作用域：受该模块内判定影响的所有模块集合。

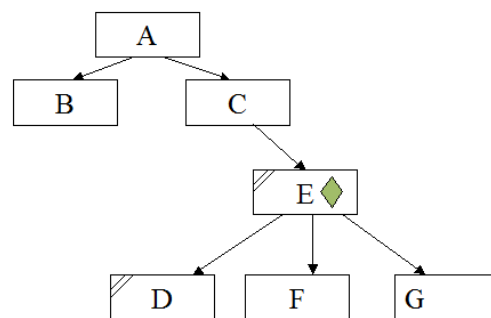
控制域：模块本身及所有直接或间接从属它的模块集合。

若模块作用域不在控制域内，会增大模块间控制耦合。

改善作用域不在控制域的软件示例：



改进方案一：判定点上移



改进方案二：将作用域不在控制域内的模块下移

5. 降低模块接口复杂程度

模块接口复杂是软件发生错误一主要原因。应使信息传递简单且和模块功能一致。

6. 设计单入口、单出口模块

避免内容耦合。

7. 模块功能可预测

输入数据相同，产生同样输出。

模块功能防止过分受限。

五、描绘软件结构的图形工具（书 P102）

层次图和 HIPO 图

结构图

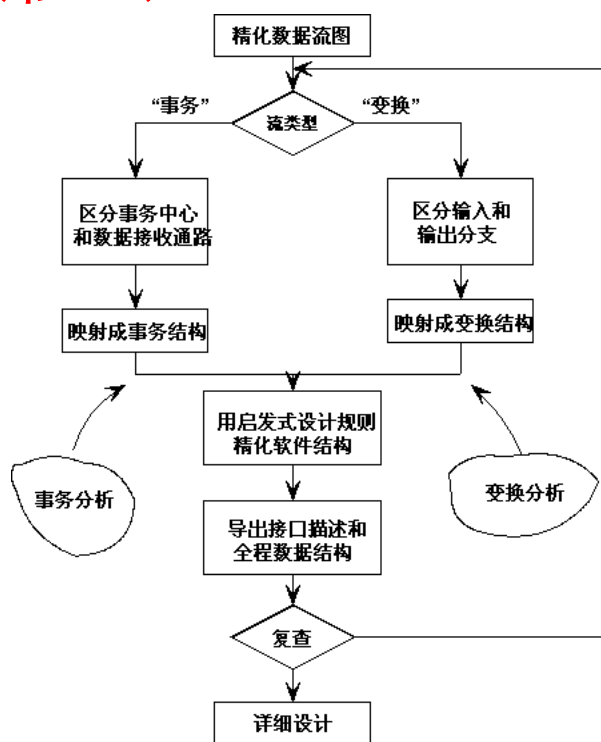
六、面向数据流的设计方法（书 P104）

概念

面向数据流的设计要解决的任务，就是将软件需求分析阶段生成的逻辑模型数据流图映射（Mapping）表达软件系统结构的软件结构图。

结构化设计属于面向数据流的设计方法。

面向数据流的设计遵循自顶向下的系统化分解的思想，应用一组转化规则将软件的数据流图逐步转化为一组程序模块的层次结构。

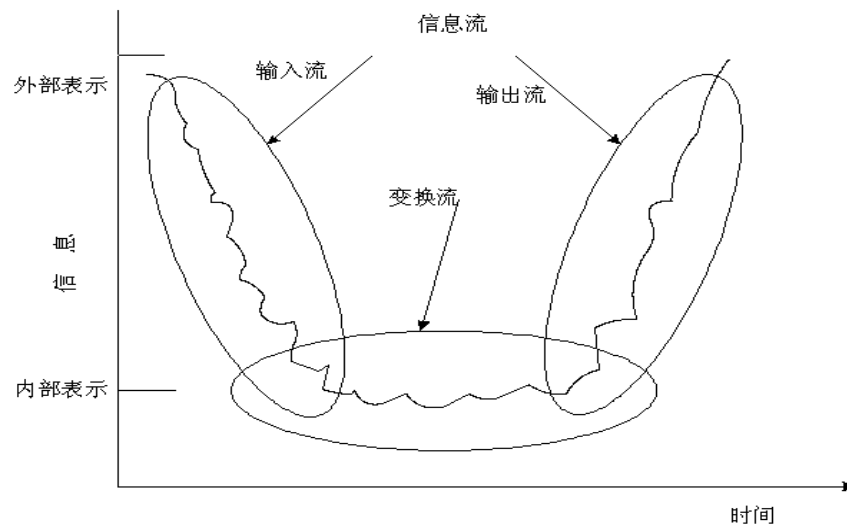


面向数据流设计过程

信息流类型

1. 变换流

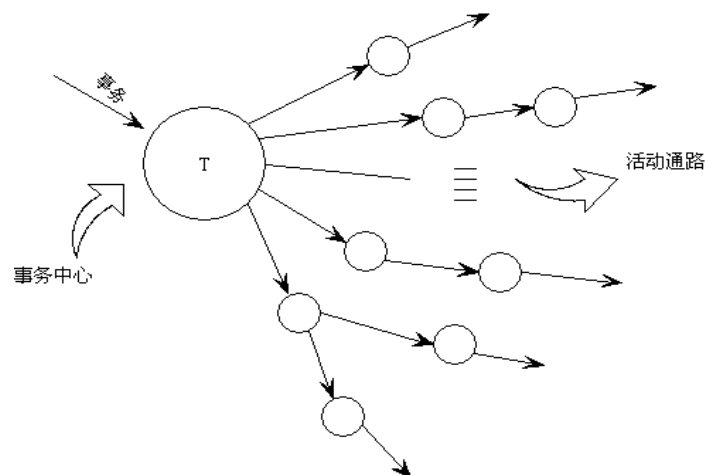
信息沿输入通路进入系统，由外部形式变换成内部形式，通过变换中心加工处理后再沿输出通路变换成外部形式离开软件系统。



2. 事务流

信息沿输入通路到一处理，由处理根据输入信息类型在若干动作序列中选一个执行。处理称事务中心，它完成以下任务：

- (1) 接收输入信息（又称事务）；
- (2) 分析每个事务确定类型；
- (3) 根据事务类型选取一活动通路。



变换分析（书 P105）

将具有变换流特点的数据流图映射成软件结构。

步骤：

1. 复查基本系统模型

确保系统输入和输出数据符合实际。

2. 复查并精化数据流图

使数据流图中每个处理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

没有明显事务中心为变换型。

4. 找出变换中心

确定数据流（输入和输出流）边界，孤立出事务中心。

5. 完成一级分解

6. 完成第二级分解

按照数据流图的结构逐步转换上层结构中的每个模块，构造其下层模块。

第二级转换就是把数据流图中的每个加工转换成软件结构中一个适当的模块。

7. 对初步软件结构精化

事务分析（书 P111）

信息流有**明显事务特点（事务中心）**，采用事务分析方法。

软件结构：**一接收分支和一发送分支**。

第六章 详细设计

一、详细设计的概念（书 P117）

结构设计确定了软件系统的总体框架，确定了系统模块的划分，而它仍不能直接进入程序编码。要从结构设计过度到编码的任务就是详细设计。

详细设计描述（详细设计阶段）可直接翻译为语言程序（编码阶段）

详细设计，又称过程设计或程序设计，其主要任务就是要严格依据软件需求规格说明书中关于功能的需求信息，选择并设计每一个模块的实现算法及其过程的详细描述。它不同于编码或编写程序。

所谓模块的算法，就是实现模块中规定功能的详细求解过程的定义，也就是模块功能“如何做”的问题。

目标：

早期人们追求的是算法的精致和效率，能够满足较少的存储需要量和较短执行时间这两项要求的算法，就认为是好算法。

但是程序的实践证明，算法的逻辑结构及可读性或可理解性应当成为比效率更重要的因素。软件危机中的一项重要内容就是软件的可维护性极差，而影响可维护性的重要前提就是算法的可读性。

因此，详细设计的目标是：在确保算法的正确性和效率的前提下，算法应当具有较好的可读性和可理解性以及容易测试及维护的特点。

任务：

详细设计向上承接软件总体结构设计阶段对系统功能分解并划分模块的结果，向下又为编码设计阶段构造源程序起到启下的基础作用，其任务有：

- ①模块的算法过程的设计。
- ②模块内部数据结构的设计。
- ③接口的设计。
- ④模块测试用例的设计。
- ⑤建立详细设计文档并复审。

二、结构化程序设计（书 P117）

是一种设计程序的技术，它采用自定向下，逐步求精的设计方法和单入口，单出口的控制结构。

三、人机界面设计（书 P119）

设计问题

1. 系统响应时间

从用户完成某控制动作，到软件给出预期响应。

两个重要属性：**长度和易变性**。

长度：

过长用户感到不安、沮丧。用户觉得系统立即响应时间范围 0.1-1 秒，超出 1 秒会让用户注意到延迟。

过短迫使用户加快操作节奏，导致出错。

解决方案：

1-10 秒鼠标显示成为沙漏

10 到 18 秒由微帮助来显示处理进度

18 秒以上显示处理窗口或显示进度条

易变性:

易变性指响应时间相对平均响应时间的偏差, 越低越好, 否则会让用户误认为系统工作异常。

2. 用户帮助措施

手册和联机帮助 (不离开用户界面)。

联机帮助两类: 集成帮助和附加帮助。

集成帮助设计在软件里面, 附加帮助系统建成后加到软件中, 前者可用性更强。

3. 出错信息处理

出错信息和警告信息, 是出现问题时交互式系统给出的“坏消息”。

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:

(1) 信息应该用用户可以理解的术语描述问题。

(2) 信息应该提供有助于从错误中恢复的建设性意见。提供清楚、易理解报错信息 (出错位置、原因);

(3) 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), 以使用户检查是否出现了这些问题, 并在确实出现问题时及时解决。

(4) 信息应该伴随着听觉上或视觉上的提示, 例如在显示信息时同时发出警告铃声, 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。

(5) 信息不能带有指责色彩, 也就是说, 不能责怪用户。

4. 命令交互

建议保留命令交互方式:

控制序列: Ctrl-C (拷贝)、Ctrl-H (帮助)、Ctrl-P (打印)、功能键: F1 (帮助) 等。

键入命令。

命令宏机制: 用户定义名字代表一个常用命令序列。

设计指南 (书 P112)

1. 一般交互指南

(1) 保持人机界面菜单选择、命令输入、数据显示风格一致;

(2) 提供有意义信息反馈: 双向通信;

(3) 破坏性动作前要确认: 删除、覆盖;

(4) 允许取消大多数操作;

(5) 减少两次操作之间必须的记忆量;

(6) 提高对话、移动和思考的效率;

(7) 允许犯错误: 保护不受致命错误破坏;

(8) 按功能对动作分类, 设计屏幕布局;

(9) 提供帮助措施;

(10) 用简单的动词或动词短语作为命令名。

2. 信息显示指南

(1) 显示与当前工作有关信息;

(2) 简单易懂方式表示数据: 图形、图表;

- (3) 使用一致标记、标准缩写和可预知颜色；
- (4) 产生有意义出错信息；
- (5) 使用模拟的方式显示信息等。

3. 数据输入指南

- (1) 减少用户输入动作：鼠标选择、滑动标尺等；
- (2) 使当前不适用命令不起作用（例如选项变灰不可点击）；
- (3) 交互灵活：保留各种输入方式；
- (4) 让用户控制交流；
- (5) 对所有输入都提供帮助；
- (6) 消除冗余输入：数据单位、整钱后键入.00、提供缺省值等。

四、过程设计工具（书 P124）

过程设计

过程设计的任务：

确定模块算法
 确定模块使用数据结构
 确定模块接口（系统外部接口、用户界面、内部模块间接口细节、输入数据和输出数据）

结构化程序设计：

结构化程序设计技术是过程设计一个关键技术。

经典定义：

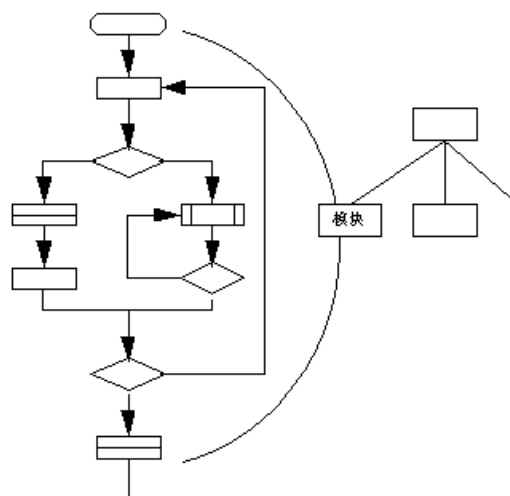
程序代码通过顺序、选择、循环三种控制结构连接，单入口单出口。

扩展定义：

可限制使用 GOTO 语句、DO_UNTIL 和 DO_CASE

修正定义：

LEAVE 和 BREAK，可从循环中转移出来。



程序流程图（书 P124）

程序流程图又称为程序框图，它是历史最悠久、使用最广泛的过程设计工具，它是人们对解决问题的方法，思路式算法的一种描述。

优点：

对控制流程描绘直观，便于初学者掌握。

缺点：

- (1) 不是逐步求精好工具，过早考虑控制流程而非非整体结构；
- (2) 用箭头代表控制流，程序员随意转移控制；
- (3) 不易表示数据结构和调用关系。

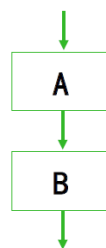
程序流程图的标准化图符：



程序流程图的表示方法：

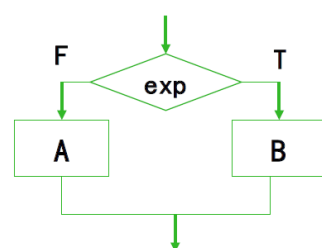
1. 顺序型

几个连续的加工依次序排列



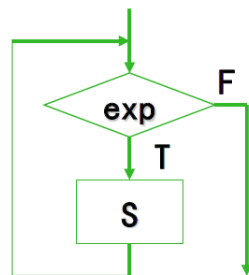
2. 选择型

由某个判定的取值决定选择两个加工中一个。



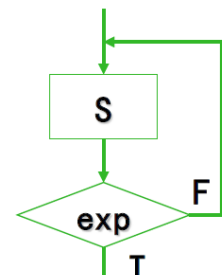
3. 当型循环型

当循环控制条件成立时，重复执行特定的加工。



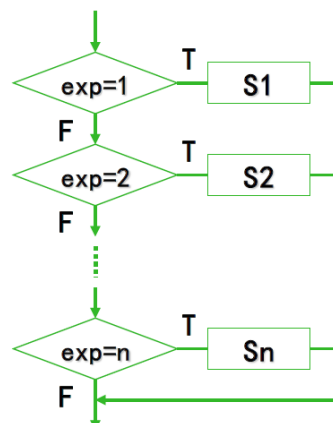
4. 直到型循环型

重复执行特定的加工，直到循环控制条件成立时。



5. 多情况选择型

列出多种加工情况根据控制变量的取值，选择执行其一。



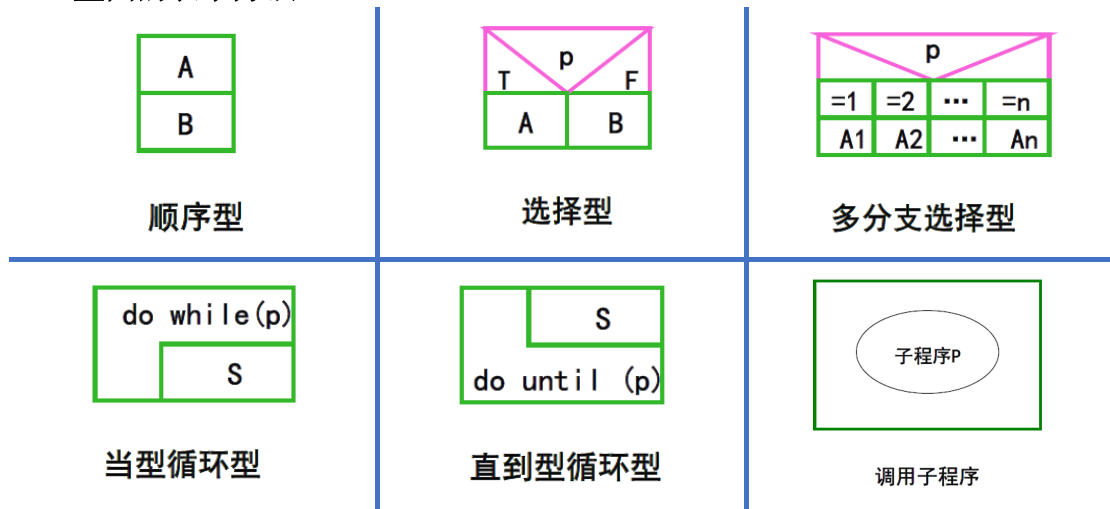
盒图（N-S 图）（书 P125）

特点：

(1) 功能域（一特定控制结构的作用域）明确；

- (2) 不可能任意转移控制；
- (3) 容易确定局部和全程数据的作用域；
- (4) 容易表现嵌套关系，也可表示模块的层次结构。

盒图的表示方法：

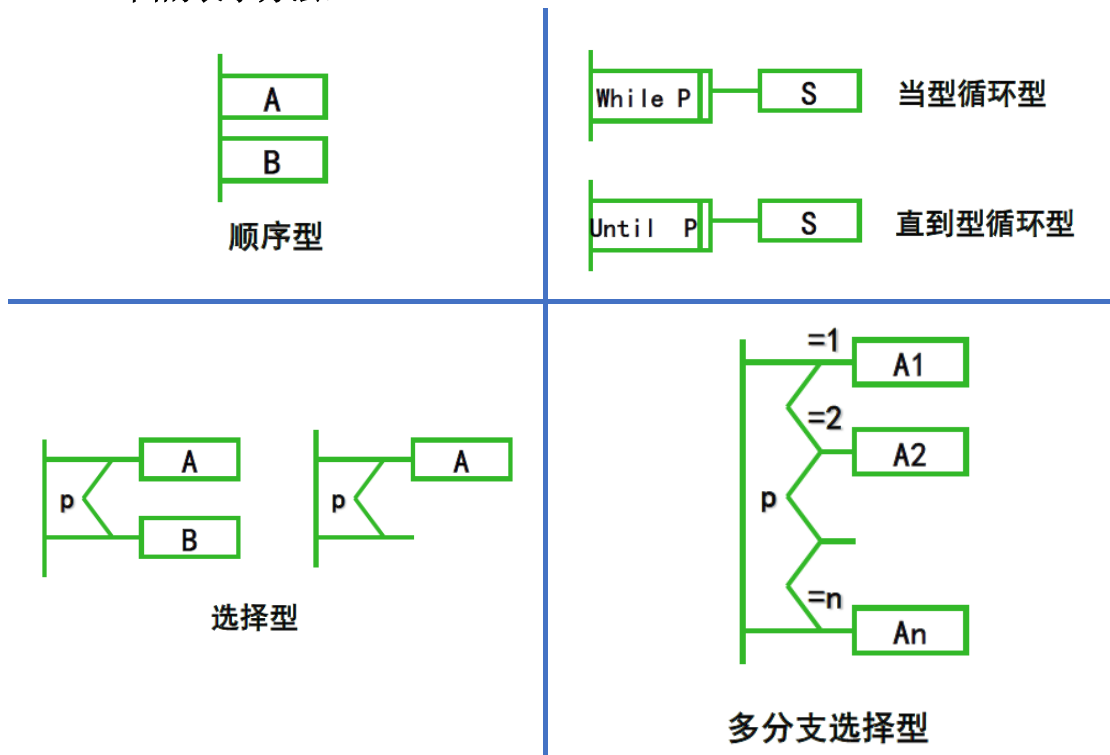


PAD 图（书 P126）

优点：

- (1) 使用 PAD 图设计的程序必然是结构化程序；
- (2) PAD 图描绘的程序结构十分清晰；
- (3) 用 PAD 图表现程序逻辑，易读、易懂、易记；
- (4) 容易将 PAD 图转换成高级语言源程序；
- (5) 支持自顶向下逐步求精。

PAD 图的表示方法：



判定表（书 P127）

概念：

判定表采用表格化的形式，能清晰表示复杂的条件组合与应做动作间对应关系。它包含四个部分：

左上部列出所有条件；

左下部所有可能做的动作；

右上部表示各种条件组合的一矩阵；

右下部是和每种条件组合相对应的动作。

判定树（书 P128）

概念：

是判定表的图形表示，其适用场合与判定表相同，表示复杂条件组合与应做动作间对应关系。

优点：

形式简单，易看出含义，易于掌握和使用。

缺点：

简洁性不如判定表，相同数据元素重复写多遍，越接近叶端重复次数越多。

结构化语言（伪码 PDL）（书 P128）

概念：

PDL 又称过程设计语言、伪码，它是一种介于自然语言与程序设计语言之间的语言，用正文形式表示数据和处理过程。即具有结构化程序的清晰易读的优点，又具有自然语言的灵活性，不受程序设计语言那样严格的语法约束。

PDL 具有严格关键字外部语法，定义控制结构和数据结构；

PDL 表示实际操作和条件的内部语法灵活自由，适应各种工程项目需要。

五、面向数据结构的设计方法（书 P129）

概念

数据结构既影响程序的结构也影响程序的处理过程，可从数据结构导出程序的处理过程，适合详细设计。

面向数据结构设计方法两种：Jackson 和 Warnier 方法

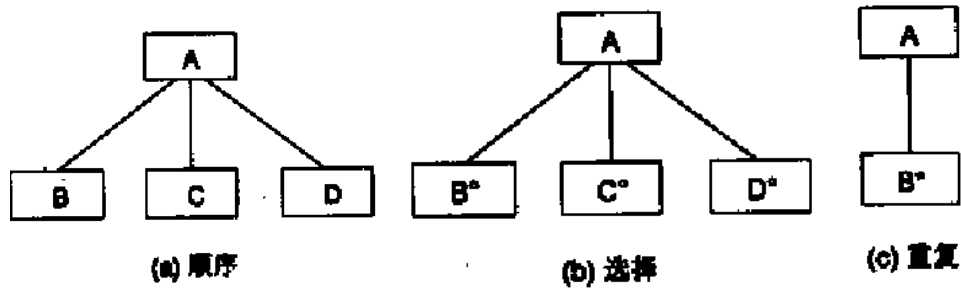
Jackson 方法（书 P130）

步骤：

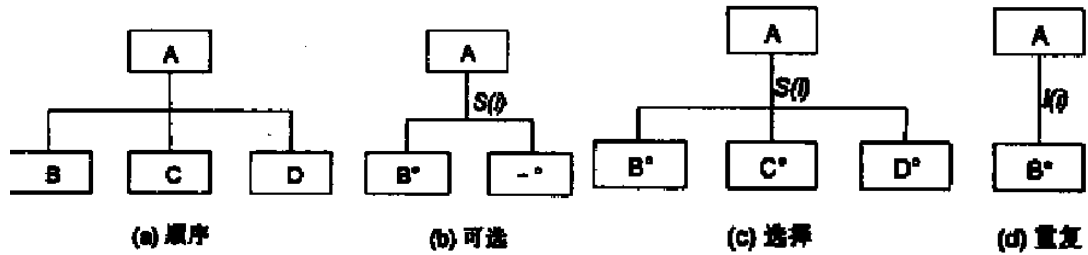
1. 确定输入数据和输出数据逻辑结构，用 Jackson 图表达；
2. 确定输入结构和输出结构中有对应关系（因果）的单元；
3. 描绘数据结构的 Jackson 图导出描绘程序结构 Jackson 图；
4. 列出所有操作和条件，分配到 Jackson 图中；
5. 用伪码表示。

表示方法：

描述数据结构：顺序、选择、重复



改进：直线，选择和循环结束条件



六、程序复杂程度的定量度量（书 P136）

概念

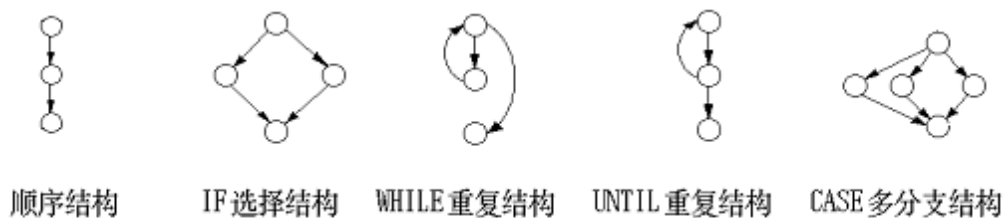
定量度量程序复杂程度的方法很有价值：把程序的复杂程度乘以适当常数即可估算出软件中错误的数量以及软件开发需要用的工作量，定量度量的结果可以用来比较两个不同的设计或两个不同算法的优劣；程序的定量的复杂程度可以作为模块规模的精确限度。

使用得比较广泛的程序复杂程度的定量度量方法是 McCabe 方法和 Halstead 方法。

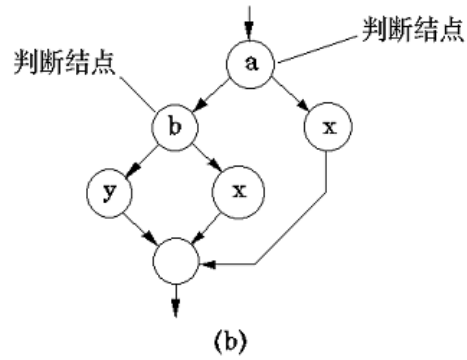
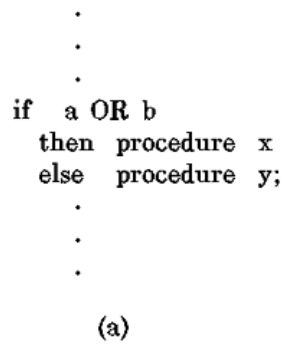
McCabe 方法（书 P137）

步骤：

1. 根据过程设计结果画出相应流程图描述程序控制流，基本图形符号如下图所示：



举例：



2. 计算流图的环形复杂度

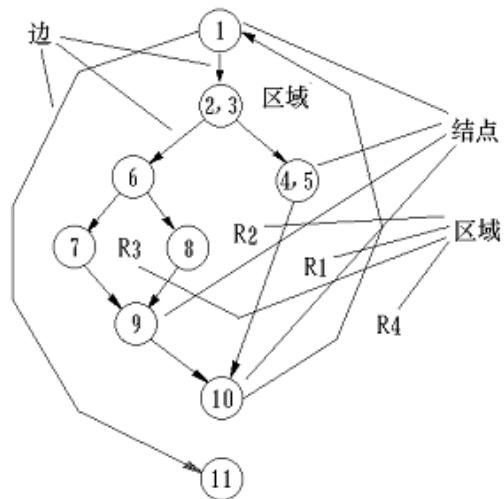
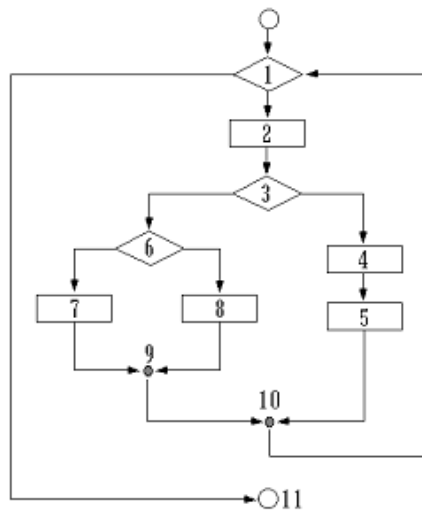
三种方法:

$$\square V(G) = \text{区域数}$$

$$\square V(G) = E - N + 2 \quad (E \text{ 为流图中边数, } N \text{ 为流图中节点数})$$

$$V(G) = P + 1 \quad (P \text{ 为判定点数})$$

举例:



$$V(G) = 4 \quad (\text{区域数})$$

$$V(G) = 11 (\text{边数}) - 9 (\text{结点数}) + 2 = 4$$

$$V(G) = 3 (\text{判定结点数}) + 1 = 4$$

Halstead 方法 (书 P139)