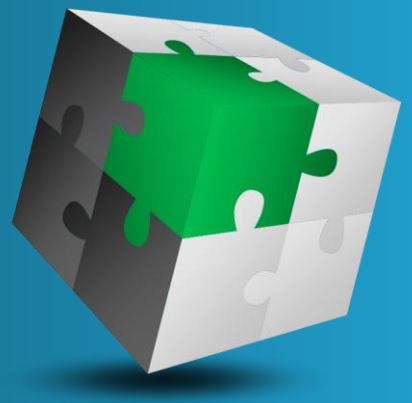
软件工程

Software Engineering



第五章 详细设计与编码实现

穆海伦

计算机学院 智能与软件研究所

E-mail: helen_se@163.com QQ: 1055874556 PH: 13750802617(612617)

1 详细设计的目标

2 结构化程序设计

3 面向对象的详细设计任务和原则

4 面向对象的详细设计与实现

5 编码

引言

软件系统的构造与实现，在现在普遍采用面向对象技术和开发工具的软件工程中已经不是特别受到关注的环节。组件和框架可以提供更专业的、更大颗粒的系统构成元素和构成框架，系统设计为构建一个完整的应用系统奠定了整体的体系结构，而完善的开发工具和开发平台已经能够提供很好的细节实现，可以给予开发者非常周到的编码、调试和开发支持。因此，在详细设计和实现阶段要做的事情，就是按照设计和规划，用这些材料，去“堆”起一个真实的系统。

另一方面，程序实现是编码工程师的任务，系统设计师对具体的实现细节并不需要给予更多的关注。

1 详细设计的目标

2 结构化程序设计

3 面向对象的详细设计任务和原则

4 面向对象的详细设计与实现

5 编码

1 详细设计的目标

软件系统的概要设计确定了系统的总体结构，而详细设计则是对概要设计结果的进一步细化，是对面向编码实现的目标的精确描述，其结果是可以直接交给编码工程师翻译成计算机代码的、简明易懂的详细规格说明。

概要设计和详细设计的区别

概要设计针对需求，因此概要设计的目标体现在：

- 概要设计对需求的完整实现；
- 概要设计与需求的一致性；
- 概要设计向需求的反向可追踪性；
- 概要设计对系统结构设计的逻辑性、合理性与可扩展性。

详细设计针对实现，因此详细设计的目标体现在：

- 设计应符合组织既定的标准；
- 设计结果对下一阶段的编码是可用的。

详细设计的程度

由于详细设计直接提供编码实现，所以详细设计要详细到什么程度，在软件企业中的差别是非常大的。这与企业本身的规模和组织形式、项目的大小、软件过程的规范程度、项目性质等都有很密切的关系。

在组织内，一般会对详细设计的程度做出规定。通过这种规定，明确详细设计与代码实现的界面，同时也是编码标准化的工作基础。

一般地，详细设计设计的具体程度包括如下内容：

- 算法过程的设计：选择合适的方法定义、描述每个处理过程的具体算法；
- 数据结构设计：对于处理过程中涉及的数据类型进行详细定义；
- 数据库物理设计：对数据库的数据存储格式、方法和安排做出定义；
- 信息编码设计：定义统一的信息编码规范，确保系统信息编码的唯一性、灵活性、简洁性、一致性、实用性和稳定性；
- 测试用例设计：设计具体测试（单元测试与集成测试）用例的测试数据和预期结果；
- 其他设计要求，如：与外部的接口等；
- 编写详细设计说明书。

说明：在对以上部分做出定义和书面的描述后，就可以任务完成了详细设计的任务，编码工程师就可以根据详细设计，进行编码实现。

1 详细设计的目标

2 结构化程序设计

3 面向对象的详细设计任务和原则

4 面向对象的详细设计与实现

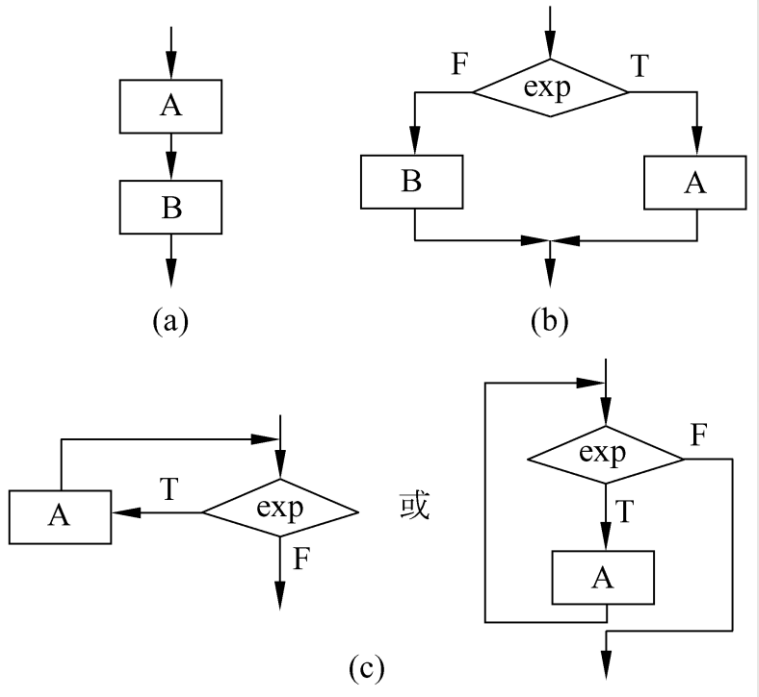
5 编码

2.1 结构化程序设计的定义

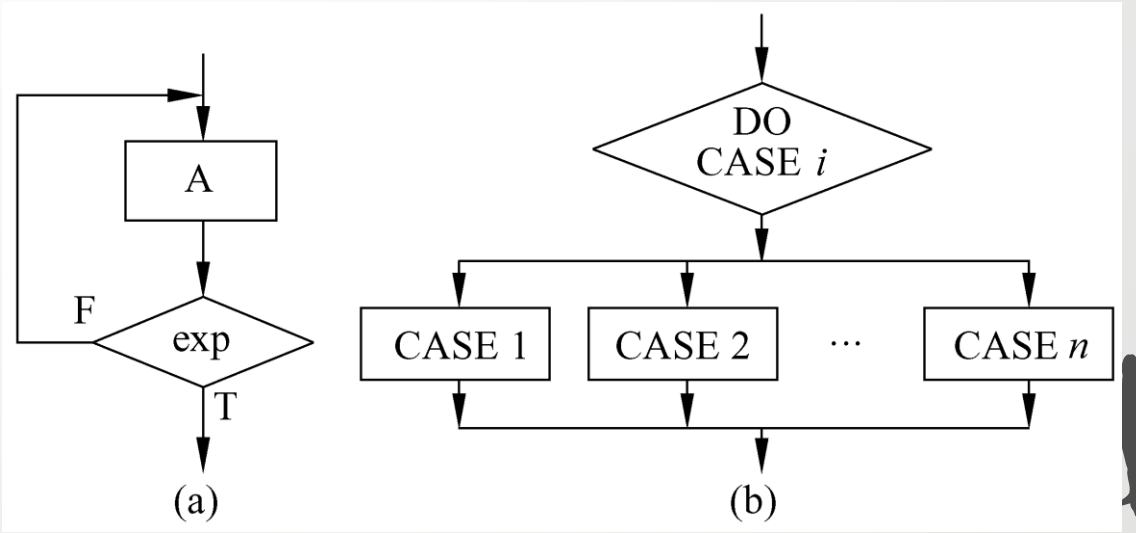
是一种设计程序的技术，它采用自定向下，逐步求精的设计方法和单入口，单出口的控制结构。

结构化程序设计的基本控制结构

- 三种基本控制结构：
  “顺序”、“选择”和“循环”；



- 补充控制结构：
  DO－UNTIL循环结构
  DO－CASE多分支选择结构



2.2 详细设计的描述工具

详细设计的几种常见描述工具：

- 程序流程图；
- N_S图（盒图）；
- PAD (problem analysis diagram)图（问题分析图）；
- 判定表；
- 判定树；
- 过程设计语言（PDL）。

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。

它的主要优点：

- 对控制流程的描绘很直观，便于初学者掌握。

它的主要缺点：

- (1) 程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。
- (2) 程序流程图中用箭头代表控制流，因此程序员不受任何约束，可以完全不顾结构程序设计的精神，随意转移控制。
- (3) 程序流程图不易表示数据结构。

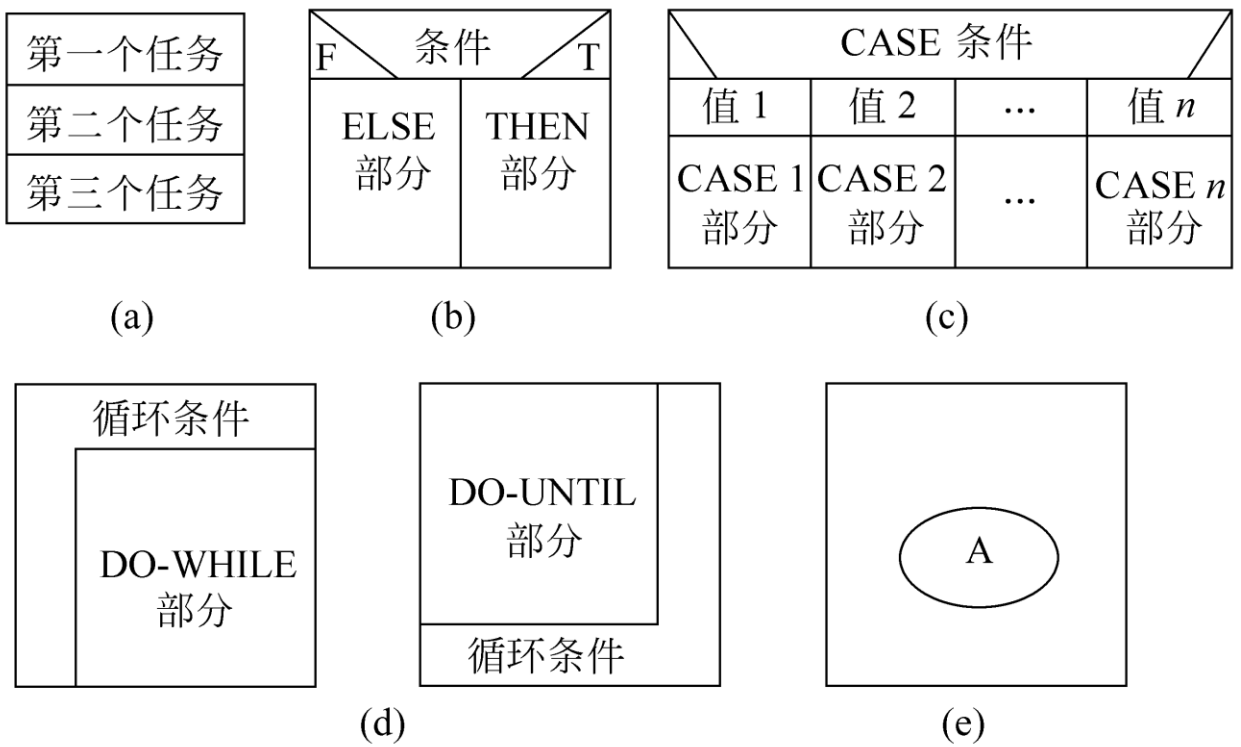
2) 盒图(N–S图)

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提出了盒图，又称为N-S图。

它的基本控制结构表示：

它有下述特点：

- (1) 功能域(即，一个特定控制结构的作用域)明确，可以从盒图上一眼就看出来。
- (2) 不可能任意转移控制。
- (3) 很容易确定局部和全程数据的作用域。
- (4) 很容易表现嵌套关系，也可以表示模块的层次结构。
- 盒图没有箭头，因此不允许随意转移控制。坚持使用盒图作为详细设计的工具，可以使程序员逐步养成用结构化的方式思考问题和解决问题的习惯。

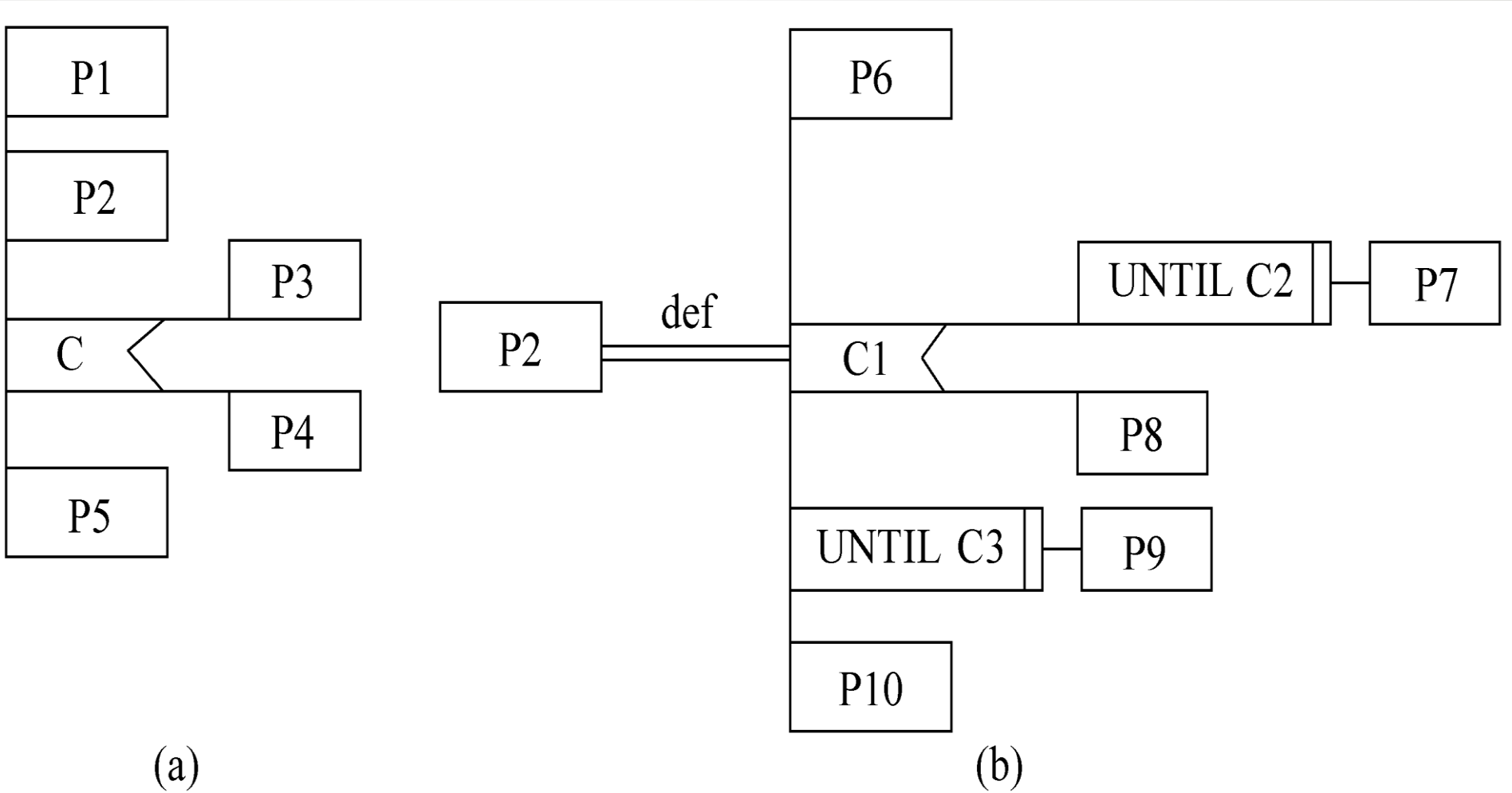


3) 盒图(N–S图)

PAD是问题分析图(problem analysis diagram)的英文缩写，自1973年由日本日立公司发明以后，已得到一定程度的推广。它用二维树形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。

它的主要优点

- (1) 使用表示结构
程序。
- (2) PAD图所描绘
，即第一层结构。
个层次，图形向右
数。
- (3) 用PAD图表现
图形，程序从图中
序执行，遍历所有
- (4) 容易将PAD图
成，从而可省去人
- (5) 即可用于表示
- (6) PAD图的符号
以定义一个抽象的
，直至完成详细设



4) 过程设计语言（PDL）

过程设计语言（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；另一方面，PDL表示实际操作和条件的内部语法通常又是灵活自由的，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。

PDL应该具有下述特点：

- (1) 关键字的固定语法，它提供了结构化控制结构、数据说明和模块化的特点。为了使结构清晰和可读性好，通常在所有可能嵌套使用的控制结构的头和尾都有关键字，例如，if…fi(或endif)等等。
- (2) 自然语言的自由语法，它描述处理特点。
- (3) 数据说明的手段。应该既包括简单的数据结构(例如纯量和数组)，又包括复杂的数据结构(例如，链表或层次的数据结构)。
- (4) 模块定义和调用的技术，应该提供各种接口描述模式。

PDL作为一种设计工具有如下一些优点：

- (1) 可以作为注释直接插在源程序中间。这样做能促使维护人员在修改程序代码的同时也相应地修改PDL注释，因此有助于保持文档和程序的一致性，提高了文档的质量。
- (2) 可以使用普通的正文编辑程序或文字处理系统，很方便地完成PDL的书写和编辑工作。
- (3) 已经有自动处理程序存在，而且可以自动由PDL生成程序代码。

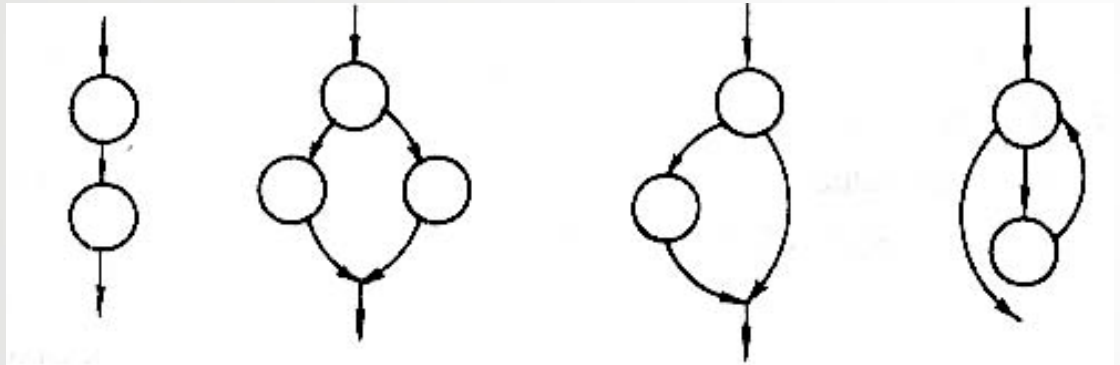
PDL的缺点是不如图形工具形象直观，描述复杂的条件组合与动作间的对应关系时，不如判定表清晰简单。

2.3 McCabe环域复杂度

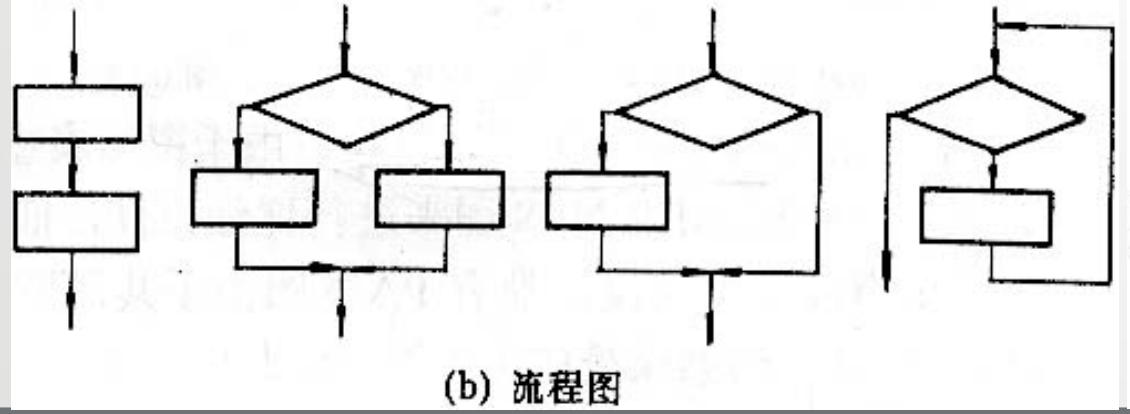
McCabe方法根据程序控制流的复杂程度定量度量程序的复杂程度，这样度量出的结果称为程序的环形复杂度。

为了突出表示程序的控制流，人们通常使用流图(也称为程序图)。

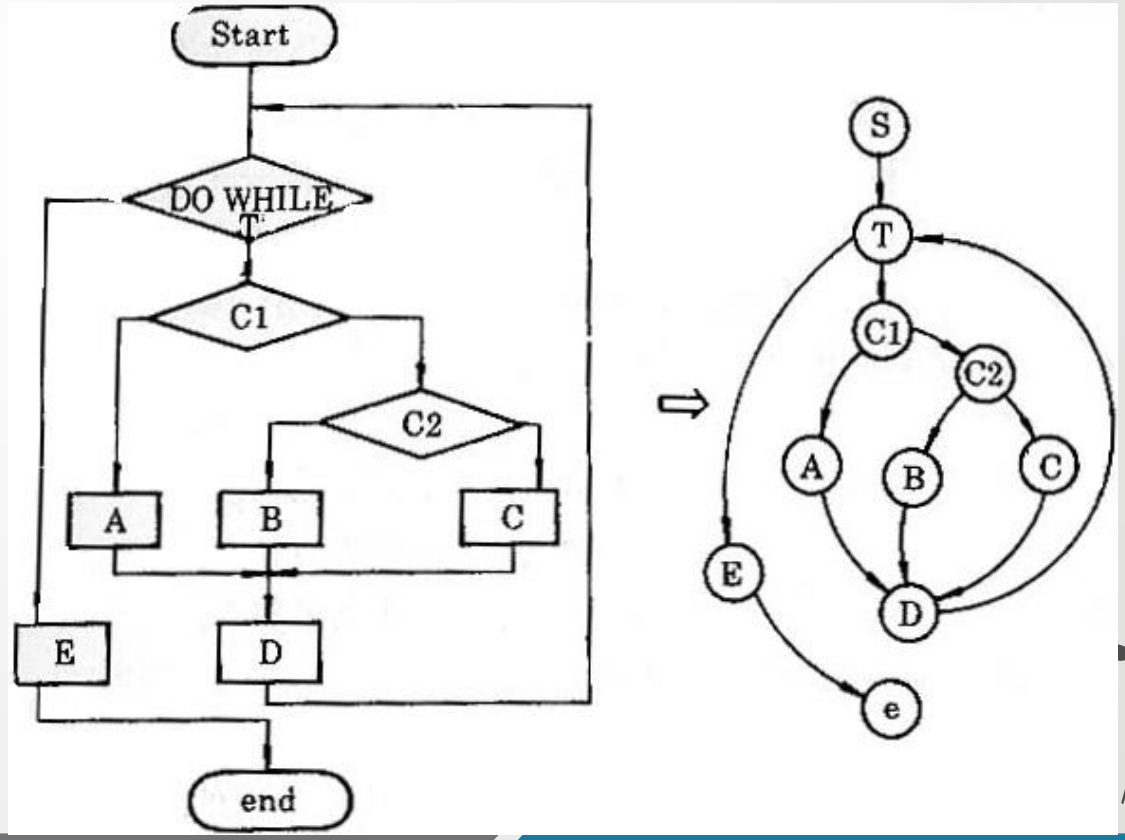
所谓流图实质上是“退化了的”程序流程图，它仅仅描绘程序的控制流程，完全不表现对数据的具体操作以及分支或循环的具体条件。



(a) 程序图



(b) 流程图



McCabe环域复杂度的计算方法

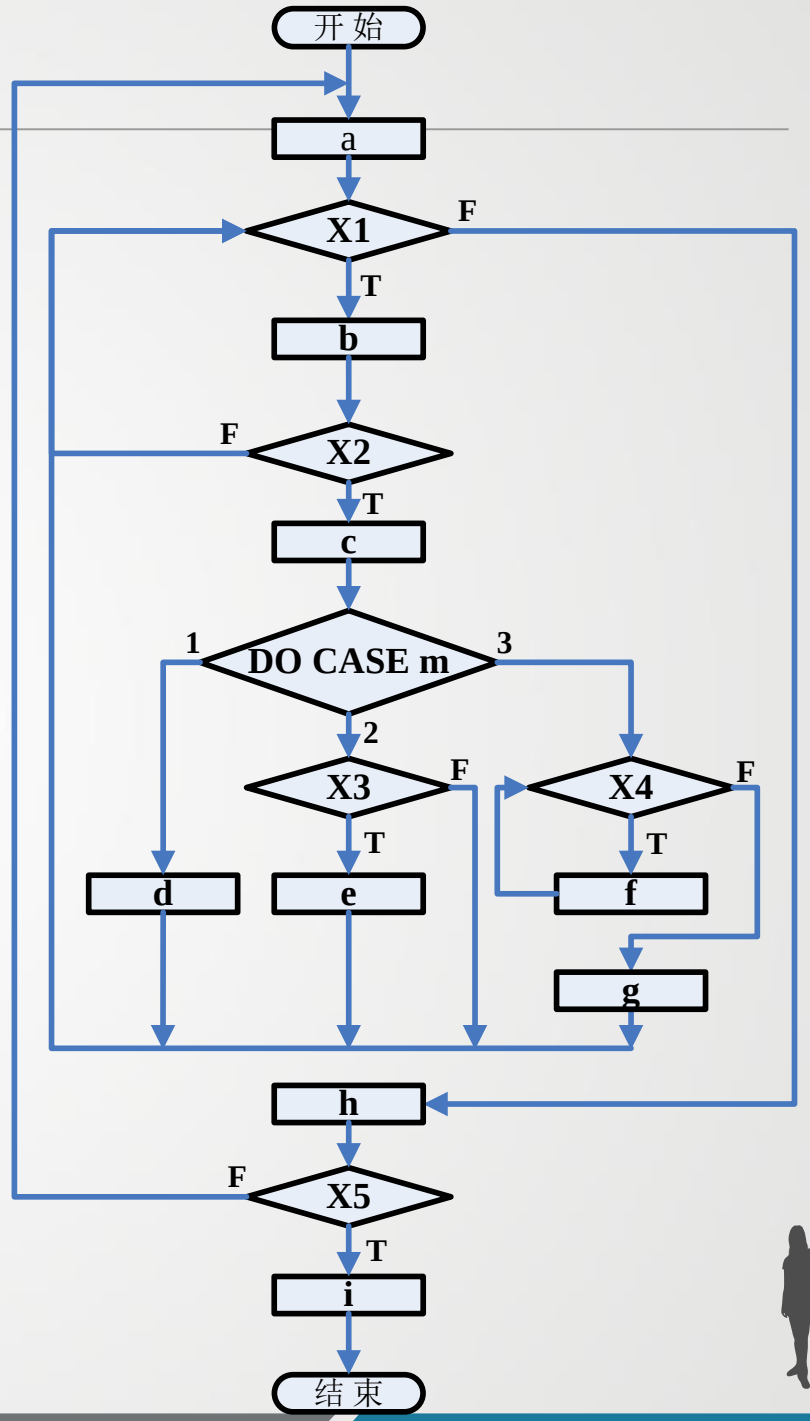
环形复杂度定量度量程序的逻辑复杂度。有了描绘程序控制流的流图之后，可以用下述3种方法中的任何一种来计算环形复杂度。

- (1) 流图中的区域数等于环形复杂度。
- (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是流图中判定结点的数目。

2.4 课堂练习

右图为某程序模块的程序流程图，试：

- 写出相应的伪码；
- 画出相应的N_S图、PAD图和程序图；
- 计算其环域复杂度。



3.1 结构化设计方法和面向对象设计方法的比较

在设计方法上，不论是传统的、还是面向对象的方法，首先都需要进行粗框的、整体的、总体的设计，然后才是精细的、具体的设计，这个基本思路是一样的。概要设计是将分析模型映射为具体的软件结构，而详细设计则是把概要设计具体化。

- 结构化方法的数据设计，是从需求分析阶段得到的数据模型和数据字典出发，设计出相应的数据结构；面向对象方法的对象设计则是把数据作为类的一个属性，设计合适的数据结构来表示这个属性；
- 结构化方法的体系结构设计师从业务流、数据流图出发，对业务处理流和相应的数据流进行分析，得出软件的层次化的模块结构图。典型的结构是树型的“菜单”结构，数据文件或数据库是模块之间最主要的信息连接纽带。而面向对象方法的子系统设计则是从分析模型划分子系统，在考虑通信、并发、部署、重用等问题的基础上，建立系统层次结构；
- 结构化方法的接口设计描述系统内部、系统与系统之间以及系统与用户之间如何通信，接口包括了信息交互和特定的行为，因此，数据流和控制是接口设计的基本对象。而面向对象方法的接口设计主要是消息设计；
- 结构化方法的过程设计是从需求分析阶段获得的过程规格说明、控制规格说明和状态图出发，得到系统各个功能的过程化描述。而面向对象方法的方法设计是从系统行为模型和功能模型出发，得到各个类的方法以及实现细节的描述。

3.2 面向对象设计的设计原则

在面向对象的详细设计阶段，追求如下内容：

- 可重用性；
- 可扩展性；
- 健壮性；
- 协作性；

1) 可重用性

在详细设计阶段的可重用性是代码级的可重用。代码级的可重用体现在本项目内部的代码重用（内部重用）和新老项目之间的代码重用（外部重用）。

为实现可重用的设计原则，应包括如下内容：

- 重用方法：
  - 内部重用方法：找出设计中的相同或相似部分，应用继承机制重用；
  - 外部重用方法：经过长期的积累，以体系结构为核心，获得重用可能。
- 设计原则：
  - 方法的内聚：类的一种方法只能完成一种功能，如有不相关的方法，应进行分解；
  - 减少代码的规模：代码过长，应进行分解；
  - 保持对外接口的一致性：相同或相似的方法，应保持名称、参数、返回值和条件的基本一致；
  - 分离策略和实现：把判断、选择和具体实现分离，前者检查状态、做成选择，它依赖具体的应用逻辑，后者则针对数据进行确定的操作，不做成选择，甚至出错只报告，也不做出处理；
  - 方法覆盖所有操作数据：对同一个数据的相同操作，应在统一的方法内实现；
  - 加强封装性：只操作对象内部的数据，避免操作全局数据；
  - 减少方法的耦合性；
  - 多利用继承机制：继承是实现共享和提高重用的主要方法。

2) 可扩展性

可扩展是软件质量的重要指标，面向对象的继承和多态性使系统具有很好的可扩展可能。

为实现可扩展的设计原则，应包括如下内容：

- 封装数据：类的内部数据必须是隐蔽的，其他类只有通过方法，才能访问该数据；
- 封装方法内部的数据结构：内部结构只为方法实现所用，外部不可直接使用；
- 避免情况分支语句：情况分支语句可用来测试对象内部的属性，但不能用来根据对象类型选择相应的行为。因为如果这样，在增加新类时，将不得不修改原有代码；
- 区分公有和私有方法：
  - 公有方法是对象的对外接口，其他对象只能使用公有方法访问该对象。公有方法通常不能草率地被修改和删除，否则会导致对整个系统进行全面修改；
  - 私有方法是对象内部的方法，通常用来辅助实现公有方法，对外是不可见的，因此，相对修改和变化的灵活性可以大一些；
  - 区分公有和私有方法的目的是保证建立一个系统内部和外部的合适边界。

3) 健壮性

健壮性是指在错误的输入或对象状态时仍保持方法不会出错的能力，当然，健壮是相对的，是与系统开销相平衡的。

为实现健壮性的设计原则，应包括如下内容：

- 友好地对待用户的输入错误，采用提示和辅导操作；
- 把握代码优化时机；
- 选择合适的实现方法；
- 检查参数的合法性。

4) 协作性

系统开发支持更好的项目协作。

为实现协作性的设计原则，应包括如下内容：

- 在程序设计开始之前，进行周密的考虑；
- 尽量使代码易于理解；
- 在对象模型中使用相同的名称；
- 把类打包成模块；
- 对类进行详细的文档化；
- 公开公共的设计说明。

4 面向对象设计的详细设计与实现

面向对象的详细设计和实现阶段的任务是消息设计和方法设计。方法设计和实现涉及具体的业务逻辑，是对具体处理的实现，因此是比较容易理解的。

类的实现—消息设计

实现类的时候会遇到类间的关系。一个类的实现常常在某些方面依赖于其他类的实例。

- 类级关系可以是应用级关系的实现，也可以是类内属性的实现。这些关系包括：消息、组装和继承；
- 在应用程序中，应用级关系大多是以类的实例之间的消息连接方式实现通信的。在消息的参数表中指定消息的接收者（一个类的实例）；
- 还可以通过参数表相接收者提供信息；
- 消息指定一个属于接收者的服务，这个服务必须对应到该类共有界面规定的行为；

类的实现—方法设计

实现类的另一种方案是先开发一个比较小且比较简单的类，作为开发比较大且较复杂的类的基础，即从简单到复杂的方案。在这种方案中，类的开发是分层的。一个类建立在一些即有的类的基础上，而这些即有的类又是建立在其他即有的类的基础上。

类的实现方法如下：

- 软件库；
- 重用；
- 断言；
- 调试；
- 错误处理；
- 内建错误处理；
- 用户定义的错误处理；
- 多重实现；
- 应用的实现。

1 详细设计的目标

2 结构化程序设计

3 面向对象的详细设计任务和原则

4 面向对象的详细设计与实现

5 编码

5 编码

所谓编码就是把软件设计结果翻译成用某种程序设计语言书写的程序。作为软件工程过程的一个阶段，编码是对设计的进一步具体化，因此，程序的质量主要取决于软件设计的质量。但是，所选用的程序设计语言的特点及编码风格也将对程序的可靠性、可读性、可测试性和可维护性产生深远的影响。

编码风格要求

源程序代码的逻辑简明清晰、易读易懂是好程序的一个重要标准，为了做到这一点，应该遵循下述规则：

- 1.实现源程序的文档化；
  - 符号名（即标识符）的命名；
  - 程序进行适当的注解；
  - 程序的视觉组织：使用标准的，统一的格式书写源程序，有助于改进可读性。
- 2.数据说明；
  - 数据说明的次序应该标准化。有次序就容易查阅，因此能够加速测试、调试和维护的过程。
- 3.语句结构；
  - 语句构造应力求简单、直接、不能为了片面的追求效率而使语句复杂化。
- 4.输入和输出；
  - 输入和输出的方式和格式应尽可能方便用户的使用，尽量做到对用户友善。

程序设计语言的选择

适宜的程序设计语言能使根据设计去完成编码时困难最少，可以减少需要的程序测试量，并且可以得出更容易阅读和更容易维护的程序。由于软件系统的绝大部分成本用在生命周期的测试和维护阶段，所以容易测试和容易维护是极端重要的。

程序设计语言选择的实用标准如下：

- (1) 系统用户的要求。如果所开发的系统由用户负责维护，用户通常要求用他们熟悉的语言书写程序。
- (2) 可以使用的编译程序。运行目标系统的环境中可以提供的编译程序往往限制了可以选用的语言的范围。
- (3) 可以得到的软件工具。如果某种语言有支持程序开发的软件工具可以利用，则目标系统的实现和验证都变得比较容易。
- (4) 工程规模。如果工程规模很庞大，现有的语言又不完全适用，那么设计并实现一种供这个工程项目专用的程序设计语言，可能是一个正确的选择。
- (5) 程序员的知识。虽然对于有经验的程序员来说，学习一种新语言并不困难，但是要完全掌握一种新语言却需要实践。如果和其他标准不矛盾，那么应该选择一种已经为程序员所熟悉的语言。
- (6) 软件可移植性要求。如果目标系统将在几台不同的计算机上运行，或者预期的使用寿命很长，那么选择一种标准化程度高、程序可移植性好的语言就是很重要的。
- (7) 软件的应用领域。所谓的通用程序设计语言实际上并不是对所有应用领域都同样适用。因此，选择语言时应该充分考虑目标系统的应用范围。

作业

- 构件模型
- 类设计（细化）
- 交互设计（细化）

本章结束

ANY QUESTION