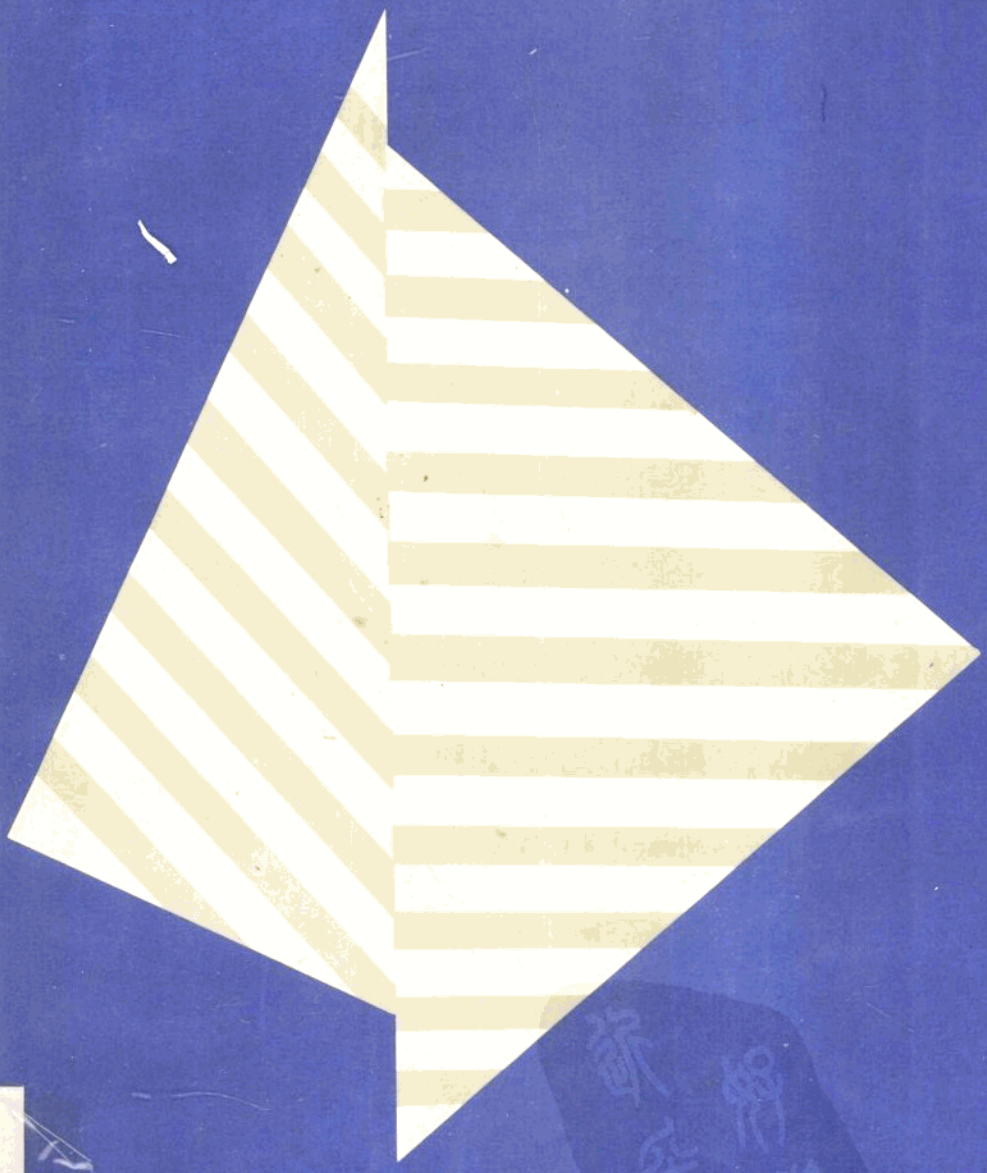
程序设计方法学教程

CHENGXUSHEJIFANGFAXUEJAOCHEN FANG

张幸儿 编著

CHENGXUSHEJIFANGFAXUEJAOCHEN FANG



南京大学出版社

程序设计方法学教程

张幸儿　编著

南京大学出版社
1992·南京

(苏)新登字第 011 号

程序设计方法学教程

张幸儿　编著

*

南京大学出版社出版

(南京大学校内,邮编 210008)

江苏省新华书店发行　　江苏地质印刷厂印刷

*

开本 787×1092 1/16　印张 12　字数 302 千

1992 年 7 月第 1 版　　1992 年 7 月第 1 次印刷

印数 1—2000

ISBN 7-305-01522-9/TP・43

定价:6.80 元

序　言

程序设计方法学自70年代随着结构程序设计概念的提出应运而生，它与软件工程学一起在克服软件危机中发挥了重大的积极影响，并对提高程序(软件)生产率始终保持着强有力的推动作用。

程序设计方法学研究程序设计的原理、原则与技术，以指导程序设计各阶段的工作。它向人们揭示了程序设计不只是手艺，更是一门科学。

“用系统的、有组织有纪律的方式研制程序”

“首先研制一个正确但可能功效低下的程序，然后用自动或半自动的手段将其变换成一个等价的、一般不易读但功效高的程序”

“程序的开发与程序正确性证明同时并举”

“控制结构描述的精化与数据结构描述的精化同时并举”

……

这些出自程序设计方法学的哲理给人们以深刻而有益的启迪。科学技术一旦为群众所掌握，便能变成巨大生产力。同样地，如果人们能自觉地应用程序设计方法学的原理、原则与技术于程序设计各阶段，无疑将促进软件的开发。

迄今程序设计方法学已成为计算机科学中内涵丰富而深刻的一个分支，涉及程序理论、研制技术、支持环境、工程规范和自动程序设计等课题。

本书胎自《程序设计方法学讲义》，该讲义在使用中深得学生好评。本书在该讲义基础上修改补充而成。显然，面面俱到是不可能也是不必要的。本书着重讨论程序设计方法学中最基本的成熟的方面，并在一定程度上反映国内外的当前工作，包括我们自己近年来的工作。

贯穿全书的主线是在程序设计方法学的原理、原则与技术指导下能以合理的代价在尽可能短的期间内研制出正确且高功效的程序。

本书的特点可概括如下：

1. 系统性强、层次分明、条理清晰、文字简炼、深入浅出、通俗易懂、便于自学。

2. 结合作者的理解与体会来阐述概念与讨论问题。读者能从程序设计方法学角度进一步深入理解与认识程序设计语言及编译实现中的有关方面。

3. 本书不仅有理论学习价值，也有现实实践指导意义。

作者殷切希望，本书不仅能有助于读者了解程序设计方法学，更能有助于读者程序的研制和为进一步开展计算机软件工作打下良好基础。

徐永森教授在百忙中抽空仔细审阅了全书，提出了许多宝贵意见与建议。本书也得到了软件工程教研室郑国梁教授及其他许多同志的关心与支持。在此一并向他们表示衷心的感谢。限于作者水平，错误与不妥之处仍在所难免，欢迎读者批评指正。

作者 1991年8月

于南京大学计算机科学系

目　录

序　言

第一章　引 言 …… 1

§1　软件的研制与程序设计方法学 …… 1

1.1　软件的研制 …… 1

1.2　大规模程序设计与软件危机 …… 3

1.3　程序设计方法学的意义、目标与内容 …… 4

§2　程序与程序设计的概念 …… 6

2.1　程序与程序设计语言 …… 6

2.2　程序设计风格 …… 7

2.2.1　冯·诺伊曼型程序设计 …… 7

2.2.2　函数型程序设计 …… 8

2.2.3　逻辑型程序设计 …… 11

2.3　程序的设计表示法 …… 16

2.3.1　流程图 …… 16

2.3.2　程序之设计语言 PDL …… 19

2.4　程序设计的基本手段 …… 21

习　题 …… 22

第二章　程序结构分析 …… 23

§1　程序的数据结构 …… 23

1.1　数据类型抽象 …… 23

1.2　类型扩充手段 …… 24

1.3　数据类型概念的进一步抽象 …… 28

§2　程序的控制结构 …… 30

2.1　基本控制成份的抽象 …… 30

2.2　基本控制结构 …… 30

2.3　其他的控制结构 …… 32

§3　程序的复杂性 …… 34

3.1　量度程序复杂性的必要性 …… 34

3.2　程序复杂性的若干量度技术 …… 35

3.2.1　控制结构复杂性 …… 35

3.2.2　整体复杂性 …… 37

3.2.3　程序复杂性量度的进一步改进 …… 39

3.3　简 评 …… 41

§4　结构化程序 …… 41

4.1　结构化程序及其特征 …… 41

4.2　结构化程序的阅读与书写 …… 46

4.2.1 结构化程序的阅读 …… 46
4.2.2 结构化程序的书写 …… 48
习 题 …… 50
第三章 程序的功能描述 …… 51
§1 概 况 …… 51
1.1 程序功能描述的需要 …… 51
1.2 若干概念 …… 52
§2 程序功能描述技术 …… 54
2.1 代数功能描述 …… 54
2.1.1 例 …… 54
2.1.2 名称的由来 …… 57
2.1.3 代数技术中要注意的几个问题 …… 58
2.1.4 代数功能描述之书写指导 …… 60
2.2 状态机功能描述 …… 61
2.2.1 例 …… 61
2.2.2 状态机功能描述与安全性的确认 …… 63
2.2.3 状态机技术的几个问题 …… 65
2.2.4 状态机功能描述之书写指导 …… 65
2.3 比 较 …… 67
习 题 …… 68
第四章 程序的正确性证明 …… 69
§1 前 言 …… 69
1.1 程序正确性证明的概念 …… 69
1.2 广义归纳法 …… 71
§2 程序正确性证明方法 …… 71
2.1 归纳断语法 …… 71
2.2 结构归纳法 …… 75
§3 程序完全正确性的证明 …… 78
§4 讨 论 …… 80
习 题 …… 81
第五章 程序设计的基本策略 …… 84
§1 大型程序之设计 …… 84
§2 程序的逐步精化 …… 84
2.1 设计方法与原则 …… 84
2.2 逐步精化之例 …… 89
2.3 讨 论 …… 90
§3 模块程序设计 …… 93
3.1 概 况 …… 93
3.2 模块程序设计之例 …… 94
3.3 与模块程序设计相关的一些问题 …… 99
3.3.1 模块定义好坏的衡量标准 …… 99

3.3.2 分块编译问题…… 100
3.3.3 现有非模块化程序设计语言与模块程序设计…… 101
3.3.4 模块库的建设…… 102
§4 关于结构程序设计 …… 102
4.1 概 况 …… 102
4.2 关于 goto 语句的讨论 …… 104
4.2.1 程序易读性与结构清晰性…… 104
4.2.2 功 效 …… 105
4.2.3 讨 论 …… 106
习 题…… 107
第六章 程序的设计技术…… 108
§1 递归程序设计技术 …… 108
1.1 递归的概念 …… 108
1.2 递归数据结构 …… 110
1.3 递归程序的设计及其正确性 …… 113
1.3.1 递归程序的设计…… 113
1.3.2 递归程序设计的正确性…… 113
1.4 化递归为迭代 …… 117
§2 其他设计技术——面向对象的设计技术 …… 119
2.1 基本思想与步骤 …… 119
2.2 例 …… 120
习 题…… 124
第七章 程序的形式推导…… 125
§1 自然演绎系统 …… 125
1.1 命 题 …… 125
1.2 推理规则 …… 128
1.3 自然演绎系统 …… 129
1.4 自然演绎系统证明之开发 …… 134
§2 最弱前置条件 …… 135
2.1 谓 词 …… 135
2.2 $WP(S,R)$的引进及其某些性质 …… 137
2.3 $WP(S,R)$之计算 …… 139
2.3.1 正文替换…… 139
2.3.2 若干简单情形 $WP(S,R)$的计算 …… 139
§3 程序设计语言控制成分的形式定义 …… 142
3.1 警卫与警卫命令 …… 142
3.2 选择结构的形式定义 …… 145
3.3 迭代结构的形式定义 …… 147
§4 程序的形式推导 …… 151
4.1 简单程序推导之例 …… 151
4.2 循环程序的推导 …… 153
4.2.1 基于循环不变式与界函数的程序推导…… 153

4.2.2 循环不变式的研制 …… 155
4.2.3 循环界函数的确定 …… 158
习 题 …… 159
第八章 程序的功效 …… 160
§1 概 述 …… 160
1.1 功效考虑 …… 160
1.2 实现高功效的途径 …… 160
§2 编译时刻优化 …… 161
2.1 合并常量运算 …… 162
2.2 消去公共子表达式 …… 162
2.3 外提不变表达式 …… 163
2.4 强度削减 …… 163
§3 化递归为迭代 …… 164
3.1 尾调用及其消去 …… 164
3.2 尾递归及其消去 …… 165
3.3 一般递归 …… 167
§4 其他方面的优化 …… 169
4.1 测试结构的简化处理 …… 169
4.2 应用于说明的优化 …… 172
第九章 程序设计工具 …… 173
§1 引 言 …… 173
1.1 程序设计工具与程序设计方法学 …… 173
1.2 程序设计工具的种类 …… 174
§2 程序设计工具简介 …… 175
2.1 设计描述工具 …… 175
2.2 编辑工具 …… 176
2.2.1 编辑程序 …… 176
2.2.2 语法制导的正文编辑程序 …… 176
2.3 格式化工具 …… 177
2.4 重构造工具 …… 177
2.5 翻译工具 …… 177
2.5.1 语言预处理程序 …… 178
2.5.2 翻译程序 …… 178
2.5.3 转换工具 …… 179
2.6 调试工具 …… 180
2.7 程序复杂性量度工具 …… 180
2.8 程序验证工具 …… 180
2.9 变换工具 …… 180
总复习思考题 …… 181
参考文献 …… 183

第一章 引 言

§1 软件的研制与程序设计方法学

1.1 软件的研制

通常对于某个应用领域中的实际问题，如果能找出其数学模型，并构造出能由计算机执行的相应算法，就能由计算机求出该问题的解案。例如，假定要用一块正方形铁皮做一个容积最大的上方开口的立方体形水桶，问应如何裁剪此铁皮？容积为多少？

为此，我们分析该问题，找出数学模型，确定算法，然后编制程序，尔后上机执行，发现错误时改正之，直到最终得到所需结果，整理成文而归档。计算机解题全过程如图 1-1 所示。

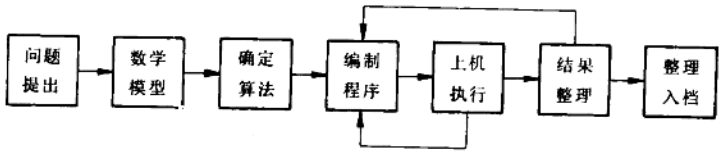


图 1-1

对于上述制桶问题，步骤如下。

问题提出：用一正方形铁皮做一个容积最大的立方体形水桶(上方开口)。

数学模型：这是求最大值问题。设铁皮边长为 a，水桶底面正方形边长为 x，则容积 $V=\frac{1}{2}(a-x)x^2$，问题归结为求解使 V 取最大值的 x；x 应满足 $\frac{dV}{dx}=0$。

确定算法：对于一般情况可以藉助于求多项式：

$$f(x)=a_0+a_1x+\cdots+a_nx^n$$

之导数 $\frac{df(x)}{dx}$ 和求多项式零点的算法求得 x 值，从而计算立方体的体积。现在 $\frac{dV}{dx}=\frac{1}{2}(2ax-3x^2)=0$，因此有 $x=\frac{2}{3}a$ 和 $V=\frac{2}{27}a^3$。

编制程序：使用 PASCAL 语言，可有下列程序：

```
program MAXVOLUME(input, output);
  var a, length,volume:real;
begin
  read(a);
  length:=2/3*a; volume:=2/27*a*a*a;
  writeln('length=',length,',V=',volume)
```

end.

假定 $a=100$(cm),可得到下列结果:length=66.67(cm)和 V(olume)=74074.07(cm^3)。

当用 a 的其他数值代入,可得到其他的相应结果。其正确性是明显的。

这是一个十分简单的例子,问题解答几乎垂手可得。一般情形下问题要复杂得多,尤其是编译程序、操作系统、大型管理系统或实时应用系统的研制,复杂程度更为显著。一般地,一个软件(程序)的研制要经历下列几个阶段,即计划阶段、研制阶段与运转维护阶段。

在计划阶段,管理阶层预测所需的经费、时间进度以及资源,判定一个软件研制课题是否可行。这时产生一个软件计划,并由课题管理人员进行复审。当可行时便进行软件需求分析和定义。软件研制人员与软件需求者共同商讨,根据系统成分间的接口关系和软件功能以及资源的限制,产生一个软件需求规格说明书。这个需求规格说明书陈述了系统要做什么,反映了用户的需求。这里,用户的需求应反映现实问题的实际需要,而软件需求规格说明书又应恰如其分地反映用户的需求。鉴于此规格说明书是其后阶段的基础,软件研制人员与软件需求者将共同对它进行复审。

已经知道要做什么,如何去做?这由研制阶段解决。

在研制阶段完成软件的研制。研制人员首先根据早先所定义的软件需求规格说明书,定义软件及其数据结构。软件往往由若干个部分(模块)组成,因此同时定义好各模块间的接口。根据一些设计准则,定性地评价设计质量。这初步设计应是完备的,且可顺着软件需求向前追溯的。

初步设计之后得到设计文件的初稿,便可进一步进行细节的研制,这时对各模块之过程方面作出详细的过程描述。当反复复审之后,把详细的过程描述也添加到设计文件中去。

在设计之后进行编码,选用某种合适的程序设计语言写出该软件的程序,不仅从细节设计描述对照检查编码的正确性,还从程序设计风格和清晰性等方面对编码进行复审。确认之后,把程序列表输出。

为了软件能交付使用,必须确信能正确工作,也即满足用户的需求,必须对软件进行测试。测试有三类,即单位测试:证实软件单个模块的功能性能;综合测试:把软件模块组装在一起进行功能与接口的测试;以及合法性测试:验证一切软件需求是否全部满足。每步测试中都制定有测试计划与例行步骤,对测试文件、测试实例与结果进行复审。

经过上述步骤,完成了软件的研制,便可准备把软件付诸使用。

在运转维护阶段,首先对整个软件构造进行复审,确信可获得一切文件且对于其后的维护任务是合适的。前面的简单例子无所谓需要维护。之所以需要维护,是因为一般地在先前的研制阶段产生的软件中往往潜伏有若干错误而不能发现,在使用过程中会暴露出来而需修改,或者由于用户提出新的要求而需添加新的功能,进而改进原有的软件。为了维护,应建立起维护责任和定义一个报错方案以及系统修改方案。当软件进行修改时不仅代码需修改,而且整个格局,包括设计阶段与研制阶段中所研制的一切文件资料都将被涉及。

一个软件(程序)不会像硬件那样因磨损等原因而报废,往往随硬件的淘汰或软件的更新而被淘汰。

概括起来,一个软件要经历这样一个全过程,即可行性分析、需求分析、需求定义、软件设计、编码实现、测试、维护与淘汰。

这整个过程反映了一个软件从无到有、直到最后被淘汰的全过程,因此称为软件的生命周

期或生命期。软件生命周期是一个软件课题的轮廓。

1.2 大规模程序设计与软件危机

随着计算机的发展，计算机的应用范围越来越广泛，软件的需要量急剧增加，同时软件的规模也日趋庞大。然而一个严酷的事实是：硬件的迅速发展使其成本猛烈下降，但是软件的成本并不因此而有所减少，相反地，软件的费用在整个计算机系统的支出中日益变得占据主要地位，这一情况越演越烈。更为严重的是，绝对费用额也在激增。据估算，美国国防部在 70 年代初，全部软件费用达到 34 亿美元之多，预测在 1990 年仅嵌入式计算机系统的软件费用总额将超过 320 亿美元。

更为遗憾的是，尽管软件费用在整个计算机系统的支出中所占比例日增，绝对费用额 也在激增，但软件质量却是低下的，软件可靠性不能保证，还往往不能按期完成。这种情况表明 70 年代所谓软件危机的存在。软件危机指的是在开发计算机软件中遇到的一系列问题，使得软件的研制周期长、费用高，而且质量低下，可靠性不能得到保证。导致软件危机的这些问题通常称为瓶颈口问题，不解决这些瓶颈口问题，软件的研制就必将受到极大的阻碍。这些问题不只表现在最终的软件功能上的不正确，事实上与如何研制软件以及如何维护其容量日益增加的现有软件有关的各种问题密切相关。

软件是逻辑的而不是物理的系统成分，不会“用坏”，但十分可能在研制期间引进的一些错误在测试期间未能察觉或暴露出来，因此，维护时经常涉及对设计进行校正与修改。这成为软件成本激增的主要原因之一。另方面，这些问题也因负有软件研制责任的人的失败而引起。在程序设计时代，甚至软件时代，软件的研制基本上都是在个别人或一组人的“手艺”基础上进行的。程序员或课题负责人几乎没有受到过软件开发新技术的正式训练，各个个别人的任务是根据他以往的经验来写程序。尽管某些人经过反复试验摸索出有秩序与高功效地研制软件（程序）的方法，但其他大多数人都以不良的习惯嗜好来研制，从而导致质量低下又不利于维护的拙劣软件。

程序是软件的本体，它是计算机系统中计算任务的处理对象和处理规则的描述。程序设计是设计、编制和调试程序的过程。通常往往认为，只需给出对象的一般陈述就足以开始编制程序，而程序一旦写出且对于若干精选的调试实例能工作，自己的任务也就完成了。也有的认为如果课题的进展落后于进度计划，只需追加一些程序员就能赶上去，等等。当然，这些往往事与愿违。

之所以这样，根本的一点是：程序设计比人们一般想象的远为复杂得多，其复杂程度超过了人类本身的智力能力范围。

当前存在两个概念，即小规模程序设计（programming in the small）与大规模程序设计（programming in the large）。它们分别同小型程序与大型程序相联系。一般地，小型程序是规模较小的，程序长度不是太长，例如几十行，至多几百行代码，个别人便能管理与控制其复杂性，因此能在较短时间内完成程序的编写，获得预期的结果。对这种程序的需求与目标等，一般总是明确的。

量变引起质变。大型程序本质上不同于小型程序。一个大型程序系统的需求会是模糊的，研制进程中的要求可能不断改变，甚至可能要求达到性能与可靠性这样的目标。诸如资源需求的预测、可行性估价等都会超出个别人的复杂性控制能力。一个大型程序可以长达几万行，甚

至几百万行代码，会由几十几百个模块组成，模块间要有复杂的接口，通常要由相当数量甚至几百个程序员工作几年才能完成。因此大规模程序设计具有程序大、复杂程度高、研制周期长与可靠性低等特点。一个大型程序系统甚至在交付使用后的几年中还会暴露出软件故障。1968年在西德加米施(Garmisch)召开的国际软件工程会议上普遍认识到了软件危机的存在，然而与软件危机有关的瓶颈口问题并非直到提出"软件危机"这一词语时才存在。事实上，它们的存在可追溯到早期软件的研制，只不过因程序规模日益庞大而导致矛盾急剧尖锐化才引起人们的重视。

软件危机的解决必须从各方面着手，需要一系列的解决办法与手段。例如在软件研制的一切阶段有合适的工具，有用于软件实现的强有力的积木块，有保证软件质量的自动技术等。根本上是采用软件工程的方法来研制软件，即用工程的方法作为产品来研制软件。软件工程技术把管理、控制与复审的各种方法跟分析、设计、编码、测试与维护的各项技术相结合。为了解决软件危机，对于作为软件本体的程序，必要的是对它的结构及程序设计，特别是大规模程序设计的原理、原则与技术进行研究。

1.3 程序设计方法学的意义、目标与内容

软件的发展大体上经历了三个阶段：程序设计时代、软件时代与软件工程时代。在当今的软件工程时代，软件不再是建立在个别人"手艺"基础上的、个体或"作坊"生产的"工艺品"，而是以工程的方式批量生产的产品。软件工程学强调了软件产品的生产性这个特征，围绕生产过程自动化与软件产品的可靠性开展了软件的生产方式、生产管理、产品设计方法、生产工具系统和产品质量保证等问题的研究。软件工程学的发展使得大大提高了软件生产率、提高了软件产品质量，又降低了软件生产成本。

软件的质量主要通过其本体程序的质量来体现，也就是说，软件质量的好坏极大程度上取决于程序质量的好坏。是程序的正确性、易读性、易维护性与功效决定了软件的可靠性、易读性、易维护性与功效。因此要研制一个可靠的、易读的、易维护的与高功效的软件实质上是研制一个正确的、易读的、易维护的与高功效的程序。在软件工程学形成与发展的同时，程序设计方法学应运而生并得到迅速发展，这是历史的必然。

程序设计方法学是软件工程学的一大支持，它的产生与发展有力地推动了计算机科学的发展，有着深远的意义。其意义大致可概括如下。

1) 揭示了程序设计不纯粹是一种技术性的"手艺"操作，它也有自身的一系列基本原理和方法，使程序设计升华为一门科学，改变了"手工艺"式程序编制的落后状态，使得能够适应软件工程的需要，作为软件产品的组成部分而系统地生产程序。

2) 应用程序设计方法学阐明的原理、原则与技术来研制程序，将使得能够确保所研制程序是正确的，从而增进程序(软件)的易维护性。

3) 程序设计方法学的发展使程序设计更加科学化、规范化与工程化，将大大有助于程序设计自动化，最终达到程序的自动生成，使人们从机械的繁重劳动中解脱出来。

4) 对程序设计方法学原理、原则与技术的研究无疑丰富了计算机科学，对计算机科学本身的发展是一个重大的推动。

5) 程序设计方法学的问世将影响计算机程序设计教育。程序设计方法学关心程序设计时应如何思考的问题。当学习程序设计时，不再仅仅学习一种程序设计语言，而是真正地学习程

序设计。程序设计方法学将成为每一个程序设计专门人才与软件工作者的一门必修课程。

总之，程序设计方法学将用以指导程序设计各阶段的工作，程序设计方法学的问世与发展有重大的理论意义和深远的影响，也有深刻的现实指导意义。

遗憾的是，关于程序设计方法学迄今尚无一个一致公认的确切定义。事实上程序设计方法学的含义与内容随着时间的演变、人们认识的深化而在不断充实、完善与发展。

大致上，程序设计方法学的目标如下。

1）研究能以合理的代价研制正确且易读易理解之程序的、系统而有条理并卓有成效的策略和方法；

2）确认与阐明解决程序设计问题及其他有关问题的各种工具与技术；

3）发现当程序设计时如何清晰地思考的规律。

程序设计方法学与软件工程学紧密相关，因为它们的目标都是试图采用系统的（工程的）方法使得能在较短的时间内以最低的费用生产正确的大型程序（软件）。两者的研究方面不乏复迭之处，但两者之间有着本质的区别。

软件工程学研究的对象一般是大型软件，研究重点是如何以工程方法生产作为产品的可靠软件；性能预测与评价、测试与调试以及维护问题的讨论往往占据大量篇幅。而程序设计方法学是以程序设计方法为其研究对象，重点研究程序设计的原理、原则与技术，讨论中占据大量篇幅的是程序的形式描述和程序设计的系统策略和方法；程序的正确性及其证明是较为关注的议题之一。

关于程序设计方法学应包括哪些内容尚无一统之说，一般地可以包括下列基本内容。

- 程序的性质与特性
- 程序的结构分析
- 程序的功能描述
- 程序语义的形式描述
- 程序的正确性证明
- 程序设计的基本策略与技术
- 程序的推导与综合
- 程序的研制与工具环境

概括起来，程序设计方法学的研究内容涉及程序设计理论、程序设计规范、程序研制技术、支持环境与自动程序设计等。它不仅研究各种具体的技术，而且着重研究各种具体技术的共性，涉及规范的全局性技术，以及这些技术的现实背景和理论基础。简而言之，程序设计方法学是研究程序设计的原理、原则和技术的学科。

这里要特别强调的是，程序设计方法学涉及的对象是大型程序，其目标是研究如何系统而有条理地研制大型程序。一般适用于研制小型程序的原理与技术并不一定能适用于大型程序的研制。然而，出于大型程序本身的复杂性，关于大型程序设计所阐述的一些原理与技术尚未能达到实用阶段。另方面，一般地教材中总是以小型甚至微型程序（仅几行代码）为例来阐明和解释。这不仅因篇幅所限，还由于下列事实：一个大型程序往往将由容易管理其复杂性的若干个部分组成。假定总计由 n 个部分组成，且每个部分的正确性概率全相等为 p，则整个程序正确性概率将为 P：$P<p^n$。要 P 是接近于 1 的有意义的值，便必须 p 充分接近于 1。这一事实表明，只有有效地研制小型程序，才有可能有效地研制大型程序或程序系统。

当然仅凭藉小规模程序设计的经验是远远不够的，有待于把程序设计方法学中的一些原理与技术应用到大规模程序设计课题中去。

程序设计包括顺序程序设计与并发程序设计。我们将主要讨论顺序程序设计问题。

§2 程序与程序设计的概念

2.1 程序与程序设计语言

“程序”这一词语，在人类社会生活中早已存在，譬如“大会程序”，它规定了大会依次要进行的工作。借用到计算机领域成为一个专门术语，程序指的是一系列的指令或语句，它同样地规定了依次要进行的一系列工作，只不过现在是由计算机去执行，更确切地说，由计算机执行组成程序的指令或语句来完成所期望的工作。

在计算机领域里，程序是计算机系统中计算任务的处理对象和处理规则的描述，它是计算机实现数据及其他信息处理的工具手段，是人与计算机的接口信息，人通过程序命令与操纵计算机进行预期完成的工作。为了要让计算机确切地理解程序的意义，对程序的书写应有较为精确的描述。

一个计算机程序总是联系于某种程序设计语言。程序设计语言是软件的重要方面，它用来书写计算机程序。可以是与具体计算机特性紧密相关的低级语言，如各种特定计算机的机器语言或汇编语言，也可以是接近于日常用语与数学表示法的高级程序设计语言，如 PASCAL、FORTRAN 与 COBOL 等，还可以是 ADA 这样的现代模块化语言等。用户可以按照应用领域实际问题的特定情况选择合适的程序设计语言。计算机直接或间接地执行用某种程序设计语言写的程序，加工处理某个特定的输入，而产生某些结果作为输出。因此一个程序 P 可以看作是把输入 I 映射到输出 O 的函数：

$$P(I)=O$$

一个程序是正确的，通常有两个含义。一是“书写”上正确，二是对于正确的输入，它能产生期望的输出，即含义上正确。

“书写”上正确，也即一串符号或字符符合定义相应程序设计语言的语法规则而构成了一个语法上正确的程序。语法规则通常用 BNF 表示法描述，也可以用语法图的形式来描述，甚而用口语方式描述；不论用哪种手段描述，总可以在关于程序设计语言的用户使用手册上找到。由于语法规则通常用 BNF 表示法描述，也即把程序设计语言处理作为用上下文无关文法来描述，从形式语言角度看，一个程序是相应上下文无关语言的句子，整个程序设计语言正是所有这样的句子组成的集合。只要语法上不正确，貌似程序，却不是程序！一个程序语法上的正确与否可由计算机（通过一个识别程序）机械而自动地识别。

一个程序执行的效果表达了这个程序的含义，一个程序的含义也就是程序的语义。尽管语法上是正确的，由于语义上不正确，一个程序依然会是不正确的。一个程序的语义自然地取决于相应程序设计语言各种成分的语义定义，也即取决于程序设计语言的语义。语义通常用非形式的口语描述方式来定义，为了精确而一致地定义程序设计语言的语义，有利于保证或验证程序的正确性，特别是自动生成程序，像语法那样，形式地定义语义是至为重要的。因而形成了计算机科学的又一重要分支——形式语义学。对此，本教程不作全面深入的讨论，仅部分地涉及

有关内容。语义形式化问题远较语法形式化问题困难得多，尽管已有不少工作，但总的说来还是探索性的工作。

人们要追求一个好的程序，不可避免地对程序的好坏进行比较。这就有一个衡量程序质量的标准问题。

一个程序很基本的好坏衡量标准是看它的研制费用与维护费用如何，一个好程序应是研制费用合理且维护费用少。但这并不反映出一个程序本身的内在好坏，从计算机上实际执行的角度看，应是正确的，能按照程序编写者的愿望那样地运行，达到预期的效果，而且执行所占时间尽可能短，所占内存空间也尽可能少。总而言之，正确的，而且时空效率都是高的。

作为表达算法、相互交流算法的程序，人们自然地期望程序是易读易理解的。一个程序也只有是易读易理解的，才容易检查或验证其正确性。在软件危机深重、人们注意力集中于可靠软件(正确程序)之研制的今天，易读易理解是尤其重要的。

当然作为软件产品之本体的程序，还应具备一定的灵活性，以适应各种不同的应用场合，例如，当需要时易于增添某些功能。易修改且灵活性强，自然地使得程序较易维护。

综合上述，一个好的程序应在下列几方面有较好的质量，即

- 正确性
- 易读性
- 易维护性
- 高功效性

一个程序是易读易理解的，这个程序就容易做到正确和易维护，而易读易理解的关键在于程序的简明、清晰和有良好的结构。然而这样的程序并不一定是高功效的，较好的是通过优化设施使易读易理解的程序也有高功效的程序代码。

为了尽可能不涉及具体的计算机特性等细节、易读易理解与易维护，通常使用高级程序设计语言。明显的是，程序是与程序设计语言紧密相关的，程序的质量在某种程度上取决于程序设计语言。程序设计语言的表示方式、定义形式与表达能力等等必然对程序的研制及其质量有相当的影响。程序设计语言的设计应该具备简明性、可靠性、可移植性与高功效性，应该考虑到程序设计方法学的原理、原则与技术。

为了能研制正确的程序，应用程序设计方法学的原理、原则与技术是势在必行。

2.2 程序设计风格

程序是可由计算机直接或间接地执行的指令或语句序列。如前所述，一个正确程序的得到一般地总要经历设计、编制与调试等阶段。程序的编制不仅反映了程序编制者对问题的理解，也反映了他的习惯、爱好与风格。例如，可以采用递归的方式也可以采用迭代的方式。特别是可以采用迥然不同的程序设计风格：冯·诺伊曼型风格、函数型风格与逻辑型风格。

2.2.1 冯·诺伊曼型程序设计

通常的程序设计都是冯·诺伊曼型的，这是因为早期 的计算机都是冯·诺伊曼型的。这种计算机包含了这样三部分，即中央处理机、存储器和把单个字在中央处理机与存储器间进行传输的连接管道。冯·诺伊曼型程序的一个基本工作是改变存储器的内容。依这种方式，程序中涉及的每一个数据的各个值对应于一个存储字，为了引用存储字中的值，必须对各个存储字命名；工作模式基本上是输入—处理—输出。

通常的程序设计语言程序都是冯·诺伊曼型程序的典型例子。这是众所熟知的，在此不拟作过多的讨论，事实上，今后的讨论仍将以冯·诺伊曼型程序设计为基础。这里要指出的是，这种风格的程序设计之基础是：对基本存储字命名、对它们赋值及重复基本动作。本质缺陷是使人们的思想局限于一次仅一个字，而不是考虑更大的概念性的单位，另外，在传输数据时不仅是关心数据本身，更多的是关心何处找到这个数据。传输的大部分不是有用的数据，而是数据的名。数据名与存储字紧密相关。

2.2.2 *函数型程序设计*

1）冯·诺伊曼型程序设计的缺陷

通常的冯·诺伊曼型程序设计语言使用变量来仿效计算机存储字，赋值语句则模拟计算机的取数、存数与算术运算，其他语句的存在是为了可能执行计算，这些计算必定是以赋值语句为基础的。因而人们总是不得不依一次一个字的方式来考虑问题。当需对数组元素求和或进行类似的工作时，必须重复地执行一些相同的语句（特别地包括赋值语句）多次，在每次重复时更改数组元素的下标值。这不能不招致冯·诺伊曼型程序设计语言的下列重大缺陷。

首先，一个程序设计语言可看作由两部分组成，一是基干部分，它包括了组成语言的基本成分，如赋值语句、循环语句和几乎所有其他的语句都是PASCAL基干的组成部分。二是可变部分。对这部分，基干部分预料其存在但并不指明其特殊性行。库函数与用户定义的过程都属可变部分。语言的基干部分描述了其固定的特征，并提供了可变部分的一般环境。假定一个语言有一个小的基干部分，却能提供大量的强有力的特性作为可变部分，那末这样一个基干部分无需改变自己就能支持很多不同的特性与风格。但冯·诺伊曼型程序设计语言因其一次一个字的程序设计风格，基干部分不可避免地是刻板而庞大的。

其次，在一个语言中提供强有力的可变部分的重要手段是组合形式，一般地能通过组合形式从现有的一些过程建立新的过程。但冯·诺伊曼型语言仅关于语句提供某些原始组合形式，例如for、while与if语句，还由于把表达式与语句分隔成两大部分和采用复杂的命名约定而阻碍了这些组合形式的充分利用。因此，冯·诺伊曼型语言可变部分几乎无扩充能力。

再次，冯·诺伊曼型语言的另一重要缺陷是缺乏有用的数学性质，因而难于关于程序的正确性进行推理。不仅几乎没有有助于对程序正确性进行推理的性质，相反却存在一些性质起着阻碍作用，例如全程变量与按引用调用参数等，尤其严重的是别名与副作用等。这体现在下列三方面。

（1）数学是静态的：函数 $f(x)$，对于相同的 f 与相同的 x，一一对应时其值总是相同的。但程序设计语言中过程概念无数学中的等价物，每次不同的调用往往会给出不同的结果。

（2）数学上变量不变，在其整个作用域内代表常值，例如 $x^2-2x+1=0$，其正整数解恒为 $x=1$，但程序设计语言中同一变量之值在计算过程中往往将不断变化。

（3）数学推理中相等这一概念起基础作用，相等的两个表达式表示有相同的值，各处可相互交换，因而可演绎。但程序设计语言中赋值语句使变量在其作用域内改变其值，打破了数学表示法的基本规则，不是引用透明的，而是引用模糊的。事实上，程序设计语言中的变量全然不是数学中的变量，而是冯·诺伊曼型计算机中存储字与寄存器的名字。

鉴于上述情况，考虑冯·诺伊曼型语言的替换语言是很自然的。

2）函数型程序设计概况

函数型程序设计风格的程序简单地是把对象映射到对象的无变量的函数。函数型程序设

计(简称 FP)的本质在于对函数进行组合以产生新的强有力的函数。

一个 FP 系统由下列几个成分组成。

(1) 对象集合:对象都是函数定义域或值域中所允许的成员。所有的原子都是对象,例如整数与大写字母串作为原子都是对象。一般地,序列$\langle x_1,\cdots,x_n\rangle$是对象,其中的$x_i(i=1,\cdots,n)$都是对象。

(2) 原始函数集合:原始函数是语言预定义的且可被应用。原始函数的例子将在后面给出。

(3) 函数模式集合:函数模式用来对现有函数进行组合以创建新的函数。

(4) 一个操作,即应用:函数应用是内在的机制,用来应用一个函数于它的自变量而产生一个对象。函数应用写成下形:$f:x$,这里把函数 f 应用于自变量 x。

(5) 定义集合:每个定义用来定义函数并对每个函数赋以一个名。

要注意的是,对象集合中还应包含元素$\perp$,它表示"底"或无定义。如果一个对象 X 是有$\perp$作为元素的序列,则 $X=\perp$。更且一切函数都是严格的,或说是保底的,即,$f:\perp=\perp$。

这里给出一些原始函数的例。注意,符号"$\equiv$"表示"等价于",而形式

$$P_1\rightarrow e_1;\cdots;P_n\rightarrow e_n;e_{n+1}$$

表示一个条件表达式,即当条件(称为断语)P_i 为真时取 e_i 值$(i=1,\cdots,n)$,否则取 e_{n+1}值。

例 1.1 恒等函数 id:$X\equiv X$

例 1.2 选择符函数

$$S:X\equiv X=\langle x_1,\cdots,x_n\rangle \ \ \&\ \ n\geqslant S\rightarrow x_i;\perp$$

这里 S 是正整数,但 S 表示函数符号:当 X 有定义且 $1\leqslant S\leqslant n$ 时,这函数取值 X 的第 S 个元素,否则取值$\perp$,表示无定义。因此,

$$3:\langle A,B,C,D\rangle=C$$

$$2:\langle E\rangle=\perp$$

例 1.3 尾函数

$$tl:X\equiv X=\langle x_1\rangle\rightarrow\varnothing;$$
$$X=\langle x_1,\cdots,x_n\rangle \ \ \&\ \ n\geqslant 2\rightarrow\langle x_2,\cdots,x_n\rangle;\perp$$

这里$\varnothing$表示空序列。

函数型程序设计藉助于函数模式从现有函数来创建新的函数。函数模式可以是复合、结构、条件、常值、插入与应用于一切等。这里仅给出将在后面例中出现的几个函数模式。

复合:$(f\circ g):X\equiv f:(g:X)$

结构:$[f_1,\cdots,f_n]:X\equiv\langle f_1:x_1,\cdots,f_n:x_n\rangle$

常值:$\overline{x}:y\equiv y=\perp\rightarrow\perp;x$

插入:$/f:X\equiv X=\langle x_1\rangle\rightarrow x_1;$
$$X=\langle x_1,\cdots,x_n\rangle \ \ \&\ \ n\geqslant 2\rightarrow f:\langle x_1,/f:\langle x_2,\cdots,x_n\rangle\rangle;\perp$$

应用于一切:

$$af:X\equiv X=\varnothing\rightarrow\varnothing;$$
$$X=\langle x_1,\cdots,x_n\rangle\rightarrow\langle f:x_1,\cdots,f:x_n\rangle;\perp$$

用来定义函数的定义取下列形式:

$$\underline{\text{def}}\ l\equiv r$$