信息学竞赛程序设计方法

——基于类的程序设计方法和技巧

张世禄　陈毅清　著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

信息学竞赛程序设计方法

——基于类的程序设计方法和技巧

张世禄　陈毅清　著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书按程序基本结构将程序设计算法分为递推算法、迭代法、尝试法和点阵关系算法4类，通过90余例具体介绍了基于类的程序设计方法和技巧、低复杂度程序的设计方法和技巧以及各类问题程序的设计方法和技巧。书中3/4的例题其算法和程序都有新颖独到之处，第6章和第7章中的不少算例为本书特有，绝大多数算例可作为信息学程序竞赛试题。

本书既可作为计算机学科程序设计教师、研究生的参考书或教学用书，也可作为中学信息学程序竞赛的教学参考书或教学用书。

图书在版编目（CIP）数据

信息学竞赛程序设计方法：基于类的程序设计方法和技巧 / 张世禄，陈毅清著. —北京：电子工业出版社，2007.9

ISBN 978-7-121-05041-1

Ⅰ. 信… Ⅱ. ①张…②陈… Ⅲ. 程序设计 Ⅳ. TP311.1

中国版本图书馆CIP数据核字（2007）第143015号

责任编辑：张贵芹
印　　刷：北京牛山世兴印刷厂
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编　100036
开　　本：787×1 092　1/16　印张：14.5　字数：368千字
印　　次：2007年9月第1次印刷
定　　价：28.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

结构化程序设计方法的理论基础是20世纪60年代末软件界关于goto语句论战的共识。结构化程序设计语言问世之后使该设计方法更为完善，不过，该设计方法流行不到20年。随着面向对象的计算机语言的流行，结构化程序设计方法越来越受到冷落。

面向对象的计算机语言问世之后，软件工程和计算机语言教材里出现了面向对象的分析和设计方法。对象的范畴很广，既有静态特性又有动态特性。在第三代语言和第四代语言中，对象并没有本质区别，它在程序中不是独立的实体，独立且有意义的是函数或过程，函数或过程在第四代语言中都可用类来表示，类和程序设计方法有关。第四代语言和第三代语言的本质区别在于在第四代语言里增设了类。

实际上，在第四代语言出现之前，软件中各程序并不是独立的，它们是一个有机整体，不允许随便增加或修改任一程序的功能，因为牵一发就会动全身，这也是用第三代语言开发的程序的维护费用比编码费用高得多的原因。

由于第四代计算机语言都拥有类，程序设计方法应以类为核心。类具有继承性、多态性和封闭性，这样组成软件的各程序可变得相对独立，从而可以提高代码重用率，降低程序设计难度和软件维护难度。

1．基于类的程序设计方法的理论和物质基础

（1）程序虽然有千千万万，但其基本结构只有三类：顺序结构、分支结构和循环结构。

（2）按数学问题、物理问题分类，算法也有千千万万，但是按程序算法分类只有四类：

① 递推算法（含递归算法）；

② 迭代算法；

③ 点阵关系算法；

④ 尝试算法。

（3）第四代计算机语言都拥有类，程序设计实际上就是将非计算机语言表述的算法翻译成计算机语言表述的算法。基于类的程序设计方法就是利用第四代语言的类结合程序结构的类、融合算法的类并继承原有设计方法的一种新的设计方法。

（4）20世纪末21世纪初出现了业务基础软件平台，平台使实现技术和业务资源分离，大大提高了软件产品的成功率，软件产业已不再是不成熟产业。

2．基于类的程序设计方法的设计过程

（1）算法分析。

除了科技计算外，大多数服务、管理软件中所给出的算法都不是用数学公式表述的。信息学中竞赛程序的算法更不是现成的，且常常是不完整的，需要在不违背题意的前提下增添约束条件，需要通过实例或从所给图表中抽象出数学本质才能归纳出算法，并将算法按上面四类进行分类。

（2）算法设计。

算法分析所归纳出的算法或者是自然语言表述的，或者是用图表表述的，即使是用数学语言表述的，也常不带计算过程和计算条件。算法设计实际上是将不带计算过程和计算条件的所有算法改写成带计算过程和计算条件的计算公式。因为带计算过程和计算条件的计算公式和计算机语言中的语句有一对一的关系，较容易翻译成计算机语言。

（3）将带计算过程和计算条件的数学公式翻译成计算机语言，再加上说明和输入、输出语句，则设计出了程序。

磨刀不误砍柴工，既作算法分析又作算法设计非但不是多余的，而且常常是必要的。

3．基于类的程序设计方法的设计原则

（1）设计低复杂度程序

程序设计路径众多，高复杂度程序千千万万，但低复杂度程序往往是有限的，且常常是唯一的，设计低复杂度程序难度也小。

（2）尽量设计通用程序

只有通用程序才容易和第四代语言中的类挂钩，低复杂度程序容易以通用程序的形式出现。

（3）对于个性化突出的问题则编写个性化程序

用户希望服务软件或管理软件能与时俱进，因此，必定会要求设计一些个性化突出的程序。这些程序常常不能被编写成通用程序，所以只能被编写为个性化突出的程序。

上述第 2 点和第 3 点是一个问题的两个方面，二者并不矛盾。

程序设计能力和设计者的抽象、归纳能力有关。所谓抽象，是指通过少量的实例找出其中数与数之间的关系和规律。而归纳，则是指将这些规律写成带计算过程和计算条件的数学公式。

程序设计难度越大，程序设计方法和技巧越重要。在我国信息学程序设计竞赛中的程序设计难度通常都不小，这本专著最适宜作参加信息学程序竞赛的教师和学生的参考书，也可作为计算机专业本科、研究生及其教师的参考书。

全书共 7 章，通过 90 余例介绍程序设计方法和技巧，约 1/10 的算例、算法和程序是其他文献中未出现过的，3/4 以上的例子虽在相关文献中有所讨论，但不及本书中的算法和程序新颖、独到。书中的 2、3、4、5 章，每章各属一类，第 6 章是尝试法的特例，第 7 章属迭代法，而每一节就是一子类。书中不是介绍一个程序的设计方法，而是介绍一类程序的设计方法。书中的所有程序结果都由计算机给出。由于程序是由带计算过程和计算条件的算式直接翻译出来的，因此只要算式和算法正确，则程序就是正确的，不须再证明。

考虑到本书可能作为中学生程序竞赛的参考书，所以书中所有程序都用 C 语言编写而未用 C++。用 C 和 PASCAL 语言所编程序都是分程序结构，因此学过 PASCAL 语言的中学生都看得懂这些程序。

书中第 1 章的部分内容和第 2 章由陈毅清编写，其余各章由张世禄编写。

作　者

2007.5

目　录

第 1 章　程序设计方法

1946 年，第一台计算机在美国宾夕法尼亚州诞生，人类从此步入了计算机时代。与此同时，世界上第一个机器语言也问世。当时，计算机只用于科学计算，只能按照人们用机器语言编写的程序算题，若无科学家开发的机器语言，计算机只是一堆破铜烂铁。

用机器语言编写程序难度很大，程序中的运算符和操作数都由数字组成。早期机器语言仅包含 0 和 1 两个数字，后来扩展成 0～7 八个数字。由于能使用机器语言编写程序的软件人员甚少，加上当时的软件生产链是用户直接通过数学语言同软件人员沟通，程序设计采用小农方式，各用各的方法。程序和数据捆绑在一起，程序之间是完全独立的，并且当时机器功能极差且硬件维护工作量大，程序设计任务重，无暇考虑程序设计方法，至少没有公开讨论程序设计方法。

1.1　结构化程序设计方法

20 世纪 50 年代末，一批高级语言问世之后，程序设计的难度大大降低了，一大批科技工作者加入了程序设计大军，从而软件产业得以形成。计算机由单一的科学计算扩展到过程控制、实时处理以及管理和服务。随着时间的推移，管理和服务软件的比例也越来越高。自“软件”这个名词出现后，程序已不再相互独立，而是一个有机整体，程序设计方法被提到了议事日程。

过去，用机器语言或汇编语言编写的程序的价值常以程序所含指令条数计算。至今仍有人认为，程序愈长、程序复杂度愈高，软件的价值就愈高，软件开发人员的水平也愈高。显然，这些观点和看法是片面的。

应该提倡和推广什么样的程序设计方法呢？应设计什么样的程序呢？

goto 语句大论战

20 世纪五六十年代，程序设计方法受小农方式和作坊方式的影响，程序设计各行其是。下面是从一本教科书中所选的例子。

【例 1.1.1】选三个数的最小者。

请分析下面程序的功能：

```
main()
 {
   float a,b,c,small;
   scanf("%f%f%f",&a,&b,&c);
   if(a<b) goto L3;
   if(b<c) goto L5;
```

```
    small=c;
    goto L9;
    L3: if(a<c) goto L7;
    small=c;
    goto L9;
    L5: small=b;
    goto L9;
    L7: small=a;
    L9: printf("%f",small);
}
```

程序运行结果：

输入：3.0 7.0 5.0

输出：3.000000

上面程序中的输入语句和输出语句是本书中新加的，原书中的语言不是 C 语言，本书将其改成了 C 语言。

不看上面题目，弄清楚该程序功能需要数分钟，对于刚学语言的人可能需要数小时。若仿照上面程序设计途径设计求十个数的最小值，一般人即使用 1 天也很难编出正确程序。

1968 年，荷兰教授 E.W.Dijkstra 发表了一篇关于程序设计中使用 goto 语句的公开信，公开信发表后立刻引起了软件界的一场大论战。经过数年论战，软件界对 goto 语句在程序中的使用才得到了共识：

（1）goto 语句会使程序的静态与动态产生差异，程序不易测试，限制了代码优化；

（2）尽量不用 goto 语句；

（3）为了提高效率可以用 goto 语句。

goto 语句的论战实际上是程序设计方法的一场论战，论战所取得的共识为以后的结构化程序设计方法打下了基础。

这里需说明的是，共识中的第二句应将“不用 goto 语句”改为“不滥用 goto 语句”更为恰当。

goto 语句虽然一般都翻译成“无条件转移语句”，但程序中的所有 goto 语句都是有条件使用的，只不过一种是显含条件的，即跟在 if 或 else 后面的 goto 语句；另外一种是隐含条件的 goto 语句，而且程序中若有一个隐含条件的 goto 语句，一定有一个显含条件的 goto 语句。程序中多次出现隐含条件的 goto 语句，称为“滥用 goto 语句”。滥用 goto 语句会使程序混乱，破坏程序的可读性。前面的例子就是滥用 goto 语句的典型。

后面书中介绍了不定重循环，即为了实现 N 重循环，必须使用 goto 语句，也就是说不仅仅是为了提高效率才用 goto 语句。

经过 goto 语句论战之后，不少软件人员寻求避免出现过多 goto 语句的方法，不久就出现了结构化程序设计方法。结构化程序设计方法的全称是“自顶向下逐步求精结构化程序设计方法”。自顶向下决定了程序结构以及运行顺序，逐步求精揭示了模块分解以及程序的编写和调试过程和方法。

按结构化程序设计方法，前面例子所对应的程序为：

```
main()
 {
   float a,b,c,small;
   scanf("%f%f%f",&a,&b,&c);
   small=a;
   if(b<small) small=b;
   if(c<small) small=c;
   printf("%f",small);
 }
```

程序运行结果：

输入：5.0 3.0 8.0

输出：3.000000

按照下一节介绍的程序复杂度的定量算法，这个程序的复杂度比前面同例程序的复杂变低，两个程序的可读性和编程难度差异都很大，两个程序的通用性更是不可同日而语。现假设将题目改为求 a，b，c，d 的最小值，按这一程序只需添加一句“if（d<small）small=d；”即可。而对于前面的程序则不知道要增加多少句。

结构化程序设计方法所使用的语言仍然是第三代计算机语言，第三代语言没有“类”结构，自顶向下逐步求精设计方法无疑是一种好方法，这种方法被直接用于软件开发中，开发软件所使用的瀑布式模式就是自顶向下的翻版。

为了减少 goto 语句，（系统）软件专家开发出了结构化程序设计语言。早期的 BASIC 语言和 FORTRAN 语言都只有 for 型语句（FORTRAN 语言中为 DO 语句），对于循环次数不确定的循环只能通过条件语句和 goto 语句实现。为了减少 goto 语句，增加 while 型循环语句是必要的。PASCAL 语言被认为是标准的结构化程序设计语言（注：ALGOL 语言也有当型循环和重复—直到型循环语句）。和 ALGOL 语言不同，PASCAL 语言以及现行的第四代语言除了 goto 语句外还有其他类型的转移语句，例如 break 语句、loop 语句、exit 语句。不带语句体的 break 和 exit 语句与带语句体标号的 goto 语句比较，就可读性而言，显然前者差一些，而且功能也低得多。对于不定重循环（即循环次数太多或重数是一变数）而言，不用 goto 语句是无法实现的，而 goto 语句是可以代替 break 和 exit 语句的。D.E.Knuth 的观点“有些情形，我主张废除转向语句，有些情形我主张引进转移语句”是科学的。

1.2　程序复杂度定量算法

程序复杂程度关系着程序编写难度，即程序复杂程度（以后简称程序复杂度）关系着程序设计难度。通常程序的复杂度愈高设计难度愈大。程序复杂度用什么量度呢？很早以前就有不少人研究程序复杂度的定量算法，研究定量算法的目的在于衡量软件人员的价值和软件价值。

不过须指出：

（1）程序是用计算机语言编写的，和自然语言一样，虽然所有语言都有一定的规律，但也常常有例外，加上用数学公式衡量由语言组成的程序复杂度本身就只能是定性的，所以，所有定量算法实际上都是定性算法，都有意义，但不可能完美。

（2）对于解决同一问题的程序的复杂度，复杂度低者程序价值高，软件人员水平高；而不是程序复杂度高，程序价值高，软件人员水平高。对于不同问题的程序，其可比性比较差，若都是最低复杂度程序，则高者价值高，软件人员水平也高。

对于好的程序复杂度，定量算法应具有以下两点：

（1）可操作性强；

（2）公正合理。

目前较好的算法有以下几种。

1.2.1 Halsted 法

Halsted 法也称长度法。长度法是早期的程序复杂度的定量算法。在第三代计算机语言刚出现之时，没有显示器；外存储器虽有磁鼓，但磁鼓造价高，存储量也不大；当时还有磁带，磁带容量虽大，但使用不便，程序和数据通常都用穿孔机存放在纸带或卡片中。用程序长度作为衡量程序复杂度是顺理成章的。

1．长度法计算公式

设 N_1 为程序中出现的运算符（包括关键字和运算符）的总个数，N_2 为程序中出现的操作数（变量和常量）的个数，则程序的复杂度为：

$$N = N_1 + N_2 \qquad (1\text{-}2\text{-}1)$$

2．长度法的可操作性

式（1-2-1）算法简单，式中只有两个参数和一个加号，N_1 和 N_2 意义清楚，程序编好后，N_1 和 N_2 都是定数，可以人工统计，也可编一个程序由计算机统计，可操作性较好，不会出现二意性。

3．长度法的公正性和合理性

程序是运算符和操作数的集合。运算符中的关键字除 goto 是由两个英文单词组成之外，其余都是一个英文单词。在程序中英文单词本身的长度没有意义，仅是它的个数和含义与程序复杂度相关，真正的运算符仍然是其个数和含义重要。至于操作数，当然个数至关重要。

程序中运算符和操作数不是随机产生的，也不是按长度比出现的，因此用式（1-2-1）衡量程序复杂度虽然简单公正、可操作性好，有一定道理，但显得片面、说服力不强。

附带提及 Halsted 还给出了一个经验公式计算程序的长度（符号个数），经不少人验证，经验公式计算结果和实际长度相对误差不超过 7%，这也是人们称 Halsted 法为长度法的原因。此外，他还给出程序中错误个数和程序长度之间的关系函数。在无显示器时代，录入错误和程序长度之间成比例是必然的，但计算错误程序的复杂度没有意义，故这里不介绍其公式。

1.2.2　McCabe 法

长度法只考虑程序长度而不考虑程序结构，也就是说不考虑关键字和运算符的含义，不少人认为该法未抓住程序复杂度的本质。

McCabe 给出了另外一个计算公式：

$$V(G)=m-n+1 \tag{1-2-2}$$

式（1-2-2）中，$V(G)$ 表示初始框节点和结束框节点相连的程序图中基本环（线性无关环）的个数，m 表示程序图中有向弧条数，n 表示节点个数，数字 1 是首节点和尾节点相连的结果（首尾节点相连必然会增加一个线性无关环）。

由于式（1-2-2）表示程序图中线性无关环个数，因此 McCabe 法称为环数法。

1．McCabe 法的可操作性

要使用式（1-2-2），首先需绘制程序流程图，绘制程序流程图后还要将其改为程序图，绘制完程序图后还要人工数有向弧条数 m 和节点数 n。

当前，软件人员在编写程序之前绘制程序流程图的情况不多了。程序流程图绘制后还要将其改成程序图，程序图和流程图都不是唯一的。由于图中有向弧和节点不受几何约束，规律极差，因此 m 和 n 的值都只能用人工统计，其可操作性不如长度法。

2．McCabe 法的合理性

程序有千千万万，但组成程序的基本结构只有三个：循环结构、分支结构和顺序结构。只有循环结构和分支结构才产生环。单由顺序结构组成的程序谈不上设计二字。显然，环是引起程序复杂度增加的关键。

从式（1-2-2）和图论知识可得出：

（1）n 个一重循环和一个 n 重循环对 $V(G)$ 的贡献一样；

（2）n 个 if 语句和一个嵌套 n 重 if 语句的语句对 $V(G)$ 的贡献一样；

（3）循环语句和条件语句对 $V(G)$ 的贡献一样，for 型循环和 while 型循环一样；

（4）goto 语句不影响 $V(G)$。

由此可见，式（1-2-2）考虑得不全面。

1.2.3　计算实例

现在将本章给出的两个程序分别用式（1-2-1）和式（1-2-2）来计算其复杂度。

按式（1-2-1）计算，第 1 个程序中 N=55；第 2 个程序中，N=22（不考虑说明和输入输出，() 只算一个符号）。

为了用式（1-2-2）计算程序复杂度，先绘制两个程序的流程图和程序图（未连首尾框）：

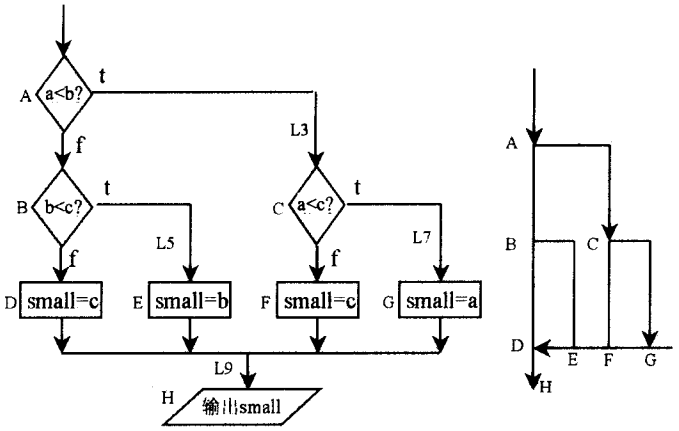

图 1-1 （a）例 1 程序流程图和程序图

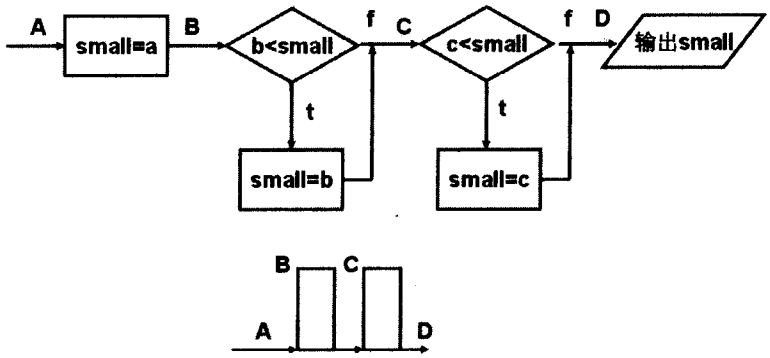

图 1-1 （b）例 1 程序流程图和程序图

按式（1-2-2）计算，图 1-1（a）的线性无关环为 4 个（首框尾框相连应增加一个线性无关环），图 1-1（b）为 3 个。

由此看来，两个算法（定性）结果相同，且和常理一致。

1.2.4 环数法改进算法

程序复杂度的定量算法意义并不大，原因有三点：

（1）程序要解决的问题千千万万，每个问题可用多个程序计算，无法找到一个完美的数学模型定量计算程序复杂度，只能定性计算。

（2）确定已编出的程序复杂度虽能衡量软件人员的水平，衡量软件的价值，但对软件本身意义不大。

（3）对程序复杂度定量计算只有一个意义，那就是为程序设计方法指出一个设计方向：即设计低复杂度程序。

基于以上三条，介绍更多的程序复杂度的定量算法意义并不大。鉴于以上两个算法都有

一定的合理性，但可操作性都不佳，长度法数运算符和操作数个数，程序较小，问题不大，对于大程序则耗费时间太多且数字大。环形法所得最后结果数字虽小，但准备工作和中间过程太长。这里不再介绍新算法，只根据图论知识和计算机语言特点给出环形法改进算法，其改进也只是改变计算环数的途径。

【定理】程序图中线性无关环个数满足

$$V(G)=N=N_1+N_2+1 \tag{1-2-3}$$

式（1-2-3）中 N_1 为程序中 if 的总个数，N_2 为循环语句的总重数。

【证明】由图论知识按式（1-2-2）可导出，在程序中只要增加一个 if 语句或者增加一个循环语句，则程序图中增加的有向弧条数和增加的节点数之差均为 1，则定理得到证明。

对于 if 语句，其程序流程图为：

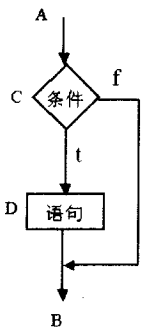

图 1-2a　不带 else 的 if 语句流程图

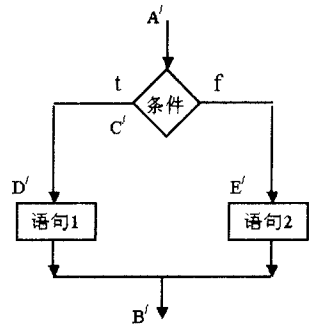

图 1-2b　带 else 的 if 语句流程图

下面是这两个流程图所对应的程序图：

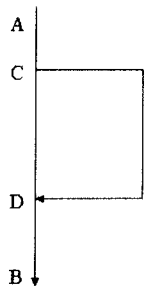

图 1-2c　不带 else 的 if 语句的程序图

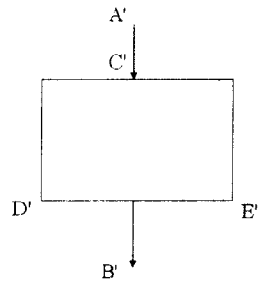

图 1-2d　带 else 的 if 语句的程序图

图 1-2c 中有 4 条有向弧、4 个节点，A 和 B 是原有的，AB 原来就是一段有向弧，也就是说增加了 3 条弧、2 个节点。对于图 1-2d 共有 5 条有向弧、5 个节点，新增加的节点数为 3，考虑到 AB 是原有的有向弧，故增加的有向弧为 4，两者之差仍为 1。

对于循环语句也可以作同样证明。

证毕。

程序复杂度算法是用作定性分析的（找不到完美算法），只要合理就行，关键是可操作

性好。显然，只统计程序中的 if 个数和循环重数，其可操作性比原算法好得多，比长度法的可操作性也好得多。虽然还有一些定量算法，但这里只介绍这两种。

由于式（1-2-3）中 1 无关紧要，故以后所说程序复杂度都不考虑 1，也就是说都不将首框和末框相连。

1.3 四代计算机语言和业务基础软件平台

计算机学科是发展最为迅速的学科，和其他学科不一样，计算机学科直接拥有两大产业——硬件产业和软件产业。正因为和产业直接挂钩，所以这门学科才发展得很快。

就软件学科而言，短短 60 年就推出了四代计算机语言，四代计算机语言又有什么特点呢？

所有的计算机语言都是程序设计工具，也是开发软件的工具，计算机语言的特点都应和程序及程序设计联系，都应和工具性能和特点相联系。

1.3.1 机器语言

第一代计算机语言被称为机器语言。机器语言离不开机器，不同的机器配置不同的机器语言，这也是机器语言名称的由来。用机器语言书写的程序其特点是：

（1）程序由一系列指令组成，指令中无论是数字还是运算府都由数字 0～1 或 0～7 组成，可读性极差。

（2）机器语言中每一条指令都只有一个简单的功能，几乎条条指令都要和地址（直接地址和间接地址）与寄存器打交道。500 条指令的程序是小程序，1 000 条指令的程序也不算大程序，程序设计难度极大。所有计算和处理都要软件人员亲自用指令实现，因此机器语言也称为手编语言。

（3）采用小农式生产方式生产，程序之间完全独立。机器语言程序主要用于科学计算，当时只有程序尚未有软件。

（4）用机器语言所编写的程序计算机能直接识别。

1.3.2 第二代计算机语言

用机器语言编写程序难度太大，程序修改也非常困难，能熟练用机器语言编写程序的人很少，因此，现在很少有人用机器语言编写程序。

为了降低编程难度，出现了汇编语言。汇编语言现在被称为第二代计算机语言。第二代计算机语言的特点是：

（1）第二代计算机语言里出现了标识符。标识符大多数直接取自英文单词，少数也和英文单词有关，都能看出其含义，因此对应程序可读性提高了。

（2）程序虽也由指令组成，但用汇编语言编写的程序中的指令实际上是机器语言中几个指令的集合，指令不再和寄存器打交道，因而代码得到了重用。

可读性提高和代码重用使得程序设计难度自然降低。

1.3.3　第三代计算机语言

20 世纪 50 年代后期出现了计算机高级语言，由于高级语言是继机器语言、汇编语言之后出现的计算机语言，故高级语言也称为第三代计算机语言。

用第三代计算机语言所编写的程序有以下特点。

（1）程序由类似自然语言的语句组成，可读性大大提高。

（2）每个语句是多条指令的结合，任何一个程序中的任何一条语句编译以后都变成若干条目标程序（指令），进一步提高了代码重用率。

（3）用高级语言编写的程序多是软件的一部分，软件是程序的有机集合。程序已不再是独立的，程序之间的关系是确定的，不允许轻易变动。

第三代计算机语言问世之后由于可读性提高了，代码重用率也提高了，因而程序设计难度大大降低了，一大批科技人员加入了程序设计大军并壮大了软件产业。

用第三代计算机语言编写出程序并生成软件后，程序之间相当于用铆钉或焊条连接起来，要修改程序或增加一个程序的功能必须考虑其他程序。能否有语言，用它编写的程序不会因为其他程序功能变异而影响该程序的功能，且可随意增加新功能而原功能不受影响呢？

1.3.4　第四代计算机语言

20 世纪 80 年代后期出现了面向对象的计算机语言，因为这代语言是第三代计算机语言出现以后的语言，故称为第四代计算机语言。

第四代计算机语言有以下特点：

（1）用第四代语言编写的程序的可读性和第三代语言所编写程序的可读性相同，都比第一、二代语言的可读性好。

（2）所有第四代计算机语言都拥有“类”结构。用第四代语言开发的软件的程序既有机相连又相对独立。“类”都具有封闭性、继承性和多态性，可按要求任意增加功能而不影响原功能和其他程序的功能，从而进一步提高了代码的重用率。

（3）对于类，由于增加功能不影响原有功能，从而降低了软件维护难度。对于用第三代语言开发的软件，维护软件所耗费的时间和财力并不比开发软件少。

降低软件维护的难度和程序设计难度一直是软件产业研究的重点，故这里只按以上两点区分第三代语言和第四代语言。

1.3.5　业务基础软件平台

自 C 语言问世后，尤其是 C++和 Java 投入使用后，工具性软件的开发难度大大降低，不少软件厂商致力于各类软件平台的开发，形成了软件平台产业。所有软件平台都是应用软件的开发工具。

计算机语言历经了四代，软件平台则分为三层：

第一层次平台是操作系统平台。操作系统平台诞生在 C 语言之前，是最基层平台，其主要功能是管理和分配软硬件资源，从而降低程序设计难度，提高了软硬件效率。操作系统平

台是跨对象、跨业务的平台，现行所有软件都必须使用操作系统平台。

第二层次平台软件为软件基础架构平台。各种中间件软件都属软件基础架构平台，不同的软件基础架构平台功能不同，但其宗旨都是降低程序设计难度和软件开发难度。

第三层次平台是业务基础软件平台，业务基础软件平台的特点是：

（1）业务基础软件平台是一个极其庞大的系统，它由操作系统平台和多个软件基础架构平台组成，是一个专为开发某类软件的工具软件。

（2）业务资源与实现技术分离，大幅度提高了代码重用率，大大降低了开发难度，也大大降低了软件维护难度。

业务资源是随用户需求变化最频繁的部分。通过分离业务部分与实现部分，使资源变动时不影响底层的实现技术。运行环境的独立，保证了应用能跨实现技术。

传统软件是将运行程序与业务逻辑一起编程并将之固定下来。应用技术与实现技术分离相当于将电视机和 VCD（或 DVD）以及光碟分开制造（或者将留声机和唱片分开制造），而将应用与实现捆绑在一起一并实现则相当于制造八音盒。

（3）形成了新的软件生产链，改变了软件生命周期。

在业务基础软件平台出现之前，软件生产链是软件厂商为用户（生产）开发软件并在一定时间内为用户维护软件。维护的含义是修改软件试运行期间所出现的错误。

业务基础软件平台出现之后，会有一些合作伙伴加盟平台开发商，合作伙伴实际上就是几个专业人员的技术组合。合作伙伴在很快熟悉了平台的功能以及应用软件的开发步骤后，为用户开发软件。由于实现技术是现成的，合作伙伴采用“业务模型驱动的方法体系和工具”生产软件架构和模式。

维护则由合作伙伴和用户中的软件人员或单独由用户软件人员完成。维护的内容不再是修改软件中的错误而是增加软件功能，维护时间也是由一段时间变成无期限。

“光碟”生产技术和成本都远比“电视机和 DVD”的低，因此合作伙伴和用户中的软件人员都很容易掌握软件开发技术和软件维护技术。

1.3.6 软件危机和软件产业前景

20 世纪 60 年代，正当软件产业界红火之时，突然发生了软件危机。产生软件危机的主要原因被认为是“作坊式生产”，但后来这种观点很快被否认。因为用机器语言作科学计算之时其生产方式还是小农生产方式，但当时并未产生软件危机。为了解决软件危机，北大西洋公约组织中的软件人员提出了“软件工程”的概念。但是软件工程虽缓解了软件危机，且让软件产业在风雨中仍向前发展，却仍未从根本上解决软件危机。

20 世纪 90 年代中期，对软件产业有三次重要的分析：

（1）Patterns of Software Systems Failure and Success, Capers Jones, 1996

（2）Chaos, Standish Group, 1995

（3）Report of the Defense Science Board Task Force on Acquiring Defense Software Commercially, Defense Science Board, 1994

Jones 的书中对软件产业的现状进行了详细的描述。Jones 分析了系统软件、信息系统、通信软件、外包软件、军用软件以及消费软件等 6 个软件领域中的几千个项目，总结了这些项目成功或失败的根本原因。三个分析的结论都相同，可概括为以下三点：