# 第1章 绪 论

1969 年，美国 IBM 公司首次宣布除操作系统继续随计算机配送外，其余软件一律计价出售，由此开创了软件成为独立商品的先河。短短 30 多年间，计算机软件的重要性与日俱增。从 PC 到笔记本式计算机，从 Internet 到移动电话，从先进的武器到现代家电，计算机软件几乎无处不在，无时不在。世界上最大的软件公司 Microsoft 及其创始人，已分别成为全球知名度最高的企业和人物之一。在很多发达国家，软件产业已成为社会的支柱产业，软件工程师也成为最受青睐的一种职业。

随着软件产业的发展，计算机应用逐步渗透到社会生活的各个角落，使各行各业都发生了很大的变化。这同时也促使人们对软件的品种、数量、功能和质量等提出了越来越高的要求。然而，软件的规模越大、越复杂，软件开发越显得力不从心。于是，业界开始重视软件开发过程、方法、工具和环境的研究，软件工程应运而生。

本章介绍软件和软件工程的基本概念，包括软件、软件危机、软件工程学、三代软件工程及其应用等。章末讨论与软件工程教学有关的几种观点，也可视为本书的导读。

## 1.1 软件和软件危机

和计算机硬件一样，20 世纪 60 年代以来，软件也在规模、功能等方面得到了很大的发展，同时人们对软件质量的要求也越来越高。那么，究竟什么是软件，它有哪些主要特征呢？

### 1.1.1 软件的定义

众所周知，计算机硬件的发展基本上遵循了摩尔定律，即每 18 个月芯片的性能与速度均提高一倍。软件的发展也十分惊人，例如就体系结构而言，它经历了从主机结构到文件服务器结构，从客户/服务器系统到基于 Internet 的服务器/浏览器结构的体系结构等变化；从编码语言来讲，它经历了从机器代码到汇编代码，从高级程序设计语言到人工智能语言等变化；从开发工具来看，它经历了从分离的开发工具（如代码编辑器、中间代码生成器和连接器）到集成的可视化开发系统，从简单的命令行调试器到方便的多功能调试器等变化。

但是在过去 40 余年中，软件的基本定义却并未改变。有些初学者认为软件就是程序，这个理解是不完全的。这里引用著名的美国软件工程教材作者 R. S. Pressman 的定义：“软件

是能够完成预定功能和性能的可执行的计算机程序，包括使程序正常执行所需要的数据，以及有关描述程序操作和使用的文档。”简而言之，可以表述为“软件 = 程序 + 文档”。

程序是为了解决某个特定问题而用程序设计语言描述的适合计算机处理的语句序列。它是由软件开发人员设计和编码的，通常要经过编译程序，才能编译成可在计算机上执行的机器语言指令序列。程序执行时一般要输入一定的数据，同时也会输出运行的结果。而文档则是软件开发活动的记录，主要供人们阅读，既可用于专业人员和用户之间的通信和交流，也可以用于软件开发过程的管理和运行阶段的维护。为了提高软件开发的效率和方便软件产品的维护，现在软件开发人员越来越重视文档的作用及其标准化工作。我国国家标准局已参照国际标准，陆续颁布了《计算机软件开发规范》、《计算机软件需求说明编制指南》、《计算机软件测试文件编制规范》、《计算机软件配置管理计划规范》等文档规范。

### 1.1.2 软件的特征

要对软件有一个全面的理解，首先要了解软件的特征。当制造硬件时，生产的结果能转换成物理的形式。如果建造一台新的计算机，从设计图纸、生产部件（VLSI 芯片、线路板、面板等）到装配原型，每一步都将演化成物理的产品。而软件却是逻辑的而不是物理的，在开发、生产、维护和使用等方面，都同硬件具有完全不同的特征。

1．软件开发不同于硬件设计

与硬件设计相比，软件更依赖于开发人员的业务素质、智力，以及人员的组织、合作和管理，而硬件设计与人的关系相对小一些。对硬件而言，设计成本往往只占整个产品成本的一小部分，而软件开发的成本很难估算，通常占整个产品成本的大部分，这意味着对软件开发项目不能像硬件设计项目那样来管理。

2．软件生产不同于硬件制造

硬件设计完成后就投入批量制造，制造也是一个复杂的过程，其间仍可能引入质量问题；而软件成为产品之后，其制造只是简单的复制而已。

3．软件维护不同于硬件维修

硬件在运行初期有较高的故障率（主要来源于设计或制造的缺陷），在缺陷修正后的一段时间中，故障率将下降到一个较低和稳定的水平上。随着时间的推移，故障率会再次升高，这是因为硬件将受到磨损等损害，达到一定程度后就应该报废。软件是逻辑的而不是物理的，虽然不会磨损和老化，但在使用过程中的维护却比硬件复杂得多。如果软件内部的逻辑关系比较复杂或规模比较大，在维护过程中很可能产生新的错误。

### 1.1.3 软件危机

随着计算机应用的逐步扩大，软件需求量迅速增加，规模也日益增长。长达数万行、数十万乃至百万行以上的软件，已不鲜见。美国阿波罗登月计划的软件长 1 000 万代码行，航天飞机软件长达 4 000 万行，就是两个突出的例子。

软件规模的增长，带来了它的复杂度的增加。如果说编写一个数十到数百行的程序连初学者也不难完成，那么开发一个数万以至数百万行的软件，其复杂度将大大上升，即使是富有经验的开发人员，也难免顾此失彼。其结果是，大型软件的开发费用经常超出预算，完成时间也常常脱期。尤其糟糕的是，软件可靠性往往随规模的增长而下降，质量保证也越来越困难。

众所周知，任何计算机系统均由硬件、软件两部分组成。在计算机应用早期，软件仅包含少量规模不大的程序，应用部门花费在软件上的投资（成本）仅占很小的份额。随着应用的不断扩大，软件的花费越来越大，所占的百分比也越来越高。B. Boehm 在 1973 年发表的一篇文章中预期，到 1985 年，美国空军的软件费用将上升到计算机总费用的 90%（参阅图 1.1）。即在每 100 元用于计算机投资总额中，软件将花费 90 元。这一预期早已为实践所证实。

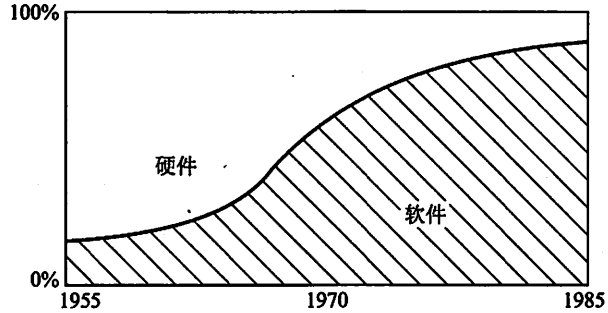



图 1.1 硬件/软件成本变化趋势

庞大的软件费用，加上软件质量的下降，对计算机应用的继续扩大构成了巨大的威胁。面对这种严峻的形势，软件界的有识之士发出了软件危机的警告。

以下再从维护和生产两个方面，进一步说明出现软件危机的原因。

① 软件维护费用急剧上升，直接威胁计算机应用的扩大。

根据一些大公司的统计，软件维护费用大约占软件总花费的 2/3，比开发费用高出一倍。一个大型软件，即使在开发时经过严格的测试与纠错，也不能保证运行中不再出现错误。维护的第一件事，就是纠正软件中遗留的错误，称为“纠错性维护”。在此后的运行过程中，还常常要为完善功能、适应环境变更等原因对软件进行修改，即所谓“完善性维护”和“适应性维护”（详见第 9 章）。不言而喻，软件的规模愈大，以上各种维护的成本必然愈高。

维护既耗费财力，也耗费人力，为了维护，要占用计算机厂家或软件公司许多软件人员，使他们不能参加新软件的开发。难怪有些文献把维护比作冰海中横在前进航道上的冰山，或直称之为维护墙（maintenance wall），将其视为软件生产和维护中难以逾越的障碍。

② 软件生产技术进步缓慢，是加剧软件危机的重要原因。

有人统计，硬件的性能价格比在过去40余年中增长了$10^6$。一种新器件的出现，其性能较旧器件提高，价格反而有所下降，这就是微电子技术创造的奇迹。软件则相形见绌。一方面，软件规模与复杂度增长了几个数量级，但生产方式长期未突破手工业的方式，创建新软件的能力提高得十分缓慢（参阅图1.2）；另一方面，很多在早期用“自由化”方法开发的、带有很强“个人化”特征的程序，因缺乏文档而根本不能维护，更加剧了供需之间的矛盾。结构化程序设计的出现，使许多产业界人士认识到必须把软件生产从个人化方式改变为工程化方式，从而促使了软件工程的诞生。

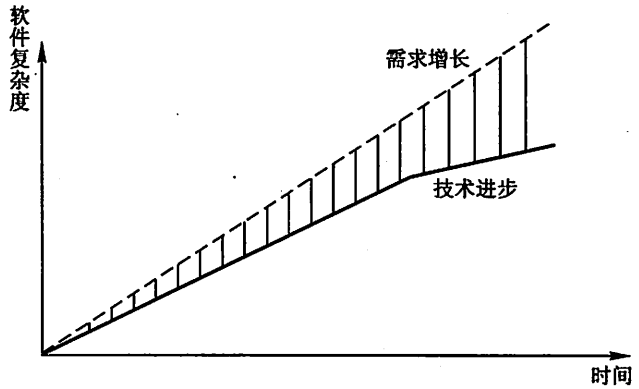



图1.2 软件技术进步落后于需求的增长

## 1.2 软件工程学的范畴

“软件工程”一词，首先是1968年北大西洋公约组织（NATO）在联邦德国召开的一次会议上提出的。它反映了软件人员认识到软件危机的出现，以及为谋求解决这一危机而做的一种努力。

人们曾从不同的角度，给软件工程下过各种定义。但是不论有多少种说法，它的中心思想都是把软件当作一种工业产品，要求“采用工程化的原理与方法对软件进行计划、开发和维护”。这样做的目的，不仅是为了实现按预期的进度和经费完成软件生产计划，也是为了提高软件的生产率与可靠性。

40年来，人们围绕着实现软件优质高产这个目标，从技术到管理做了大量的努力，形成了“软件工程学”这一计算机新学科。图1.3列举了它所包含的主要内容。

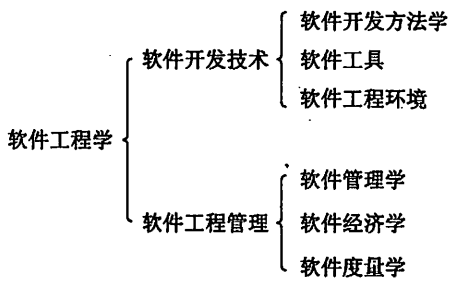



图1.3 软件工程学的范畴

### 1.2.1 软件开发方法学

软件的发展，大体上经历了程序、软件和软件产品 3 个阶段。早期的程序规模较小，随着系统程序的增加，人们把程序区分为系统程序和应用程序，并且通常将前者称为软件。但是，无论是软件或程序，在开发过程中都很少考虑到对它们的维护。只是当软件工程兴起之后，人们才把软件视为产品，强调软件的可维护性，同时确定了各个开发阶段必须完成的文档（documents）。

与上述 3 个发展阶段相对应，软件开发方法也发生了巨大变化。早期的程序设计基本上属于个人活动性质，开发人员各行其是，并无统一的方法可循。20 世纪 60 年代后期兴起的结构程序设计，使人们认识到采用结构化的方法来编写程序，不仅可以改善程序的清晰度，而且也能提高软件的可靠性与生产率。随后，人们又认识到编写程序仅是软件开发过程中的一个环节，有效的开发还应包括需求分析、软件设计、编码等多个阶段。把结构化的思想扩展到分析阶段和设计阶段，于是形成了结构化分析与结构化设计等传统的软件开发技术。与此同时，也出现了一些从不同基点出发的软件开发方法，如 Jackson 方法、LCP 方法等。尽管这些方法的具体内容各有不同，但它们都遵循结构化程序设计的原则，都对软件开发步骤和文档格式提出了规范化的要求。软件生产已经摆脱了过去随心所欲的个人化的状态，进入了有章可循的、向结构化和标准化迈进的工程化阶段。

20 世纪 80 年代出现的 Smalltalk、C++等语言，促进了面向对象程序设计的广泛流行。但是，人们继而发现，仅仅使用面向对象程序设计不会产生最好的效果。只有在软件开发的早期乃至全过程都采用面向对象技术，才能更好地发挥该技术的固有优势。于是，包括“面向对象需求分析—面向对象设计—面向对象编码”在内的软件开发方法学开始形成，并且逐步地取代了传统的软件开发方法，成为许多软件工程师的首选方法。面向对象技术还促进了软件复用技术的发展。复用加快了开发速度，开发又产生了更多的可复用软件构件，使软件复用最终成为软件开发方法学的一个重要组成部分。

### 1.2.2 软件工具

“工欲善其事，必先利其器”，人类早就认识到工具在生产过程中的重要作用。伴随着软件开发的发展，也研制出了众多“帮助开发软件的软件”，人们称之为软件工具（software tools）。它们对提高软件生产率，促进软件生产的自动化都有重要的作用。

设想在 PC 上用 Pascal 语言开发一个应用软件的过程。首先，要在编辑程序支持下把源程序输入计算机。然后调用 Pascal 编译程序，把源程序翻译成目标程序。如果发现错误，就重新调入编辑程序对源程序进行修改。编译通过后，再调用连接程序把所有通过编译的目标程序同与之有关的库程序连接起来，构成一个能在计算机上运行的可执行软件。在这里，编译程序、编辑程序、连接程序以及支持它们的计算机操作系统，都属于软件工具。离开了这些工具，软件开发就失去了支撑，将十分困难和低效，甚至不能工作。

以上提到的，仅是在编码阶段常用的一些软件工具。在开发的其余阶段，例如分析阶段、设计阶段和测试阶段，也研制了许多有效的工具。众多的工具组合起来，可以组成工具箱（tool box）或集成工具（integrated tool），供软件开发人员在不同的阶段按需选用。

### 1.2.3 软件工程环境

工具和方法，是软件开发技术的两大支柱，它们密切相关。当一种方法提出并证明有效后，往往随之研制出相应的工具，来帮助实现和推行这种方法。新方法在推行初期，总有人不愿接受和采用。若将新方法融入工具之中，使人们通过使用工具来了解新方法，就能更有效地促进新方法的推广。

方法与工具相结合，再加上配套的软、硬件支持就形成环境。创建适用的软件工程环境（software engineering environment，$SE^2$），一直是软件工程研究中的热门课题。

为了说明软件开发对环境的依赖，不妨回顾一下分时系统所产生的影响。在批处理时代，用户开发的程序是分批送入计算中心的计算机的，有了错误，就得下机修改。软件开发人员对自己编写的程序只能断续跟踪，思路经常被迫中断，效率难于提高。分时系统的使用，使开发人员能在自己的终端上跟踪程序的开发，仅此一点，就明显提高了开发的效率。近 30 年来出现的 UNIX 环境、Windows 环境，以及形式繁多的网络环境等，不仅反映了人们创造良好软件环境的努力，也把对软件工程环境的研究提升到一个新的高度。

### 1.2.4 软件工程管理

在工业生产中，即使有先进的设备与技术，管理不善的企业也不能获得良好的经济效益。不少软件在生产过程中不能按质、按时完成计划，管理混乱往往是其中的重要原因。可惜的是，至今软件管理尚未获得普遍的重视。

软件工程管理的目的，是为了按照进度及预算完成软件计划，实现预期的经济和社会效益。它包括成本估算、进度安排、人员组织和质量保证等多方面的内容，还涉及管理学、度量学和经济学等多个学科方面的知识。

显然，软件管理也可以借助计算机来实现。一些帮助管理人员估算成本、制定进度和生成报告等工具现在已研制出来。一个理想的软件工程环境，应该同时具备支持开发和支持管理两个方面的工具。

## 1.3 软件工程的发展

自 1968 年首次提出软件工程概念以来，已经 40 年了。在这一时期中，编程范型（programming paradigm）已经经历了 3 次演变，软件工程也从第一代发展到了第三代。本节将对此作概略的说明。

### 1.3.1 3种编程范型

计算机应用离不开编写程序。按不同的思路和方法来编写程序，就形成不同的编程范型。1956年，世界上第一个高级语言FORTRAN问世。50多年来，高级语言的编程范型大体经历了3次演变，即过程式编程范型、面向对象编程范型与基于构件技术的编程范型。

1．过程式编程范型

过程式编程范型遵循“程序 = 数据结构 + 算法”的思路，把程序理解为由一组被动的数据和一组能动的过程所构成。编程时，先设计数据结构，再围绕数据结构编写其算法过程。程序运行后，获得预期的计算结果或正确的操作，借以满足程序的功能需求。对于比较大的程序，必须先进行功能分解，把较小的功能编写为子程序，子程序再调用子子程序，直至最终将程序做成由一组组大小适中、层层调用的模块所构成的应用程序系统。

从20世纪50年代至80年代后期，过程式编程范型广泛流行了30余年。FORTRAN、Pascal和C，都是当时常用的面向过程编码语言（procedure-oriented programming language，POPL），它们集中展示了这种编程范型的特点。

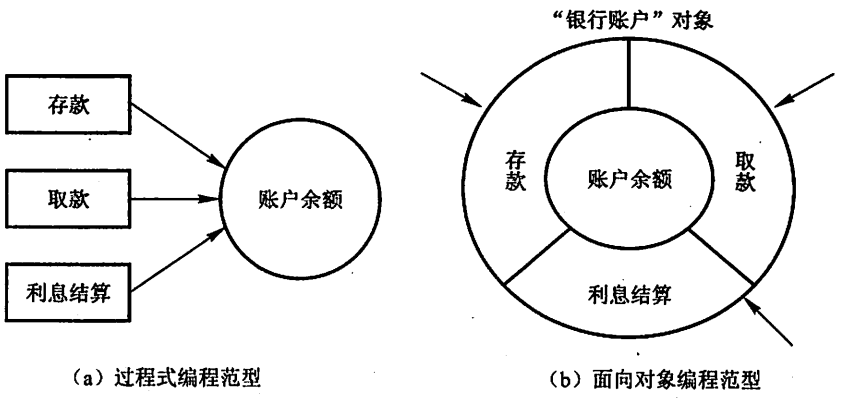
在客观事物中，实体的内部“状态”（一般用数据表示）和“运动”（施加于数据的操作）总是结合在一起的。可是在用POPL编码时，程序模型（称为解空间，solution domain）却被人为地构造为偏离客观实体本身的模型（称为问题空间，problem domain）。随着程序规模的扩大，这类编程范型的缺陷越来越明显，在错综复杂的调用下，即使功能可以满足，性能也不容易满足，使程序难于维护和移植。因此，这类范型通常只用于编写代码在50 000行以下、不会轻易更改的应用程序。

2．面向对象编程范型

在面向对象的程序设计中，数据及其操作被封装在一个个称为对象（object）的统一体中，对象之间则通过消息（message）相互联系，“对象+消息”的机制取代了“数据结构+算法”的思路，因而较好地实现了解空间与问题空间的一致性，从而为解决软件危机带来了新的希望。从面向过程程序设计到面向对象程序设计（object-oriented programming，OOP），是程序设计方法的又一次飞跃，目前正日益显露出其优越性。

为便于对照，请读者先看一个简单的例子——银行储蓄处理事务。这一事务包含一个数据（账户余额）和3个对数据的操作——存款、取款与利息结算（每年度一次）。图1.4显示了使用两类不同编程范型实现这一事务的模型。

在采用过程式编程范型中，如图1.4（a）所示，数据“账户余额”与施加在其上的操作是分离的，存款、取款与利息结算等3个过程模块分别将相应的操作施加于“账户余额”，使数据获得更新。图1.4（b）则采用面向对象编程范型，存款、取款与利息结算作为对象内部的3个事件过程，与“账户余额”一起封装在“银行账户”对象中，通过来自对象外部的3种不同的消息（如图1.4（b）中箭头所示），即可启动相应的操作。




（a）过程式编程范型　　（b）面向对象编程范型

图 1.4 用两类不同编程范型实现银行储蓄处理事务

两者对照，面向对象编程范型具有明显的优势。首先，在图 1.4（a）中，存款、取款与利息结算都是分离的单个模块，而在图 1.4（b）中，“银行账户”对象一般是由多个模块组成的一个独立的单元。而且由于解空间与问题空间的一致性，软件开发人员对客观世界建立的分析模型，实际上已为软件系统的设计准备好了基本的框架。因而在大型程序的开发中，采用面向对象编程范型可以有效地降低软件的复杂性，简化软件的开发。

面向对象编程范型的优势也体现在软件的维护上。假定图 1.4（a）中的“账户余额”在开发时使用了整型数，维护时要改变为浮点数，则该软件中凡与“账户余额”有关的程序均需修改。但如果采用图 1.4（b）中的面向对象编程范型，则上述改变将限于“银行账户”对象本身，与该对象以外的其他部分以及触发该对象操作的外部消息完全无关。这将使因修改程序而引发软件故障的机会大大减少，使得大型软件的维护更加容易和快捷。

3．基于构件技术的编程范型

正当过程式编程范型渐渐过时、逐步被方兴未艾的面向对象编程范型取而代之的时候，一种更先进的编程范型已悄悄地进入软件开发人员的视线，这就是基于构件技术的编程范型。

简言之，构件（component，有些文献翻译为组件）可以理解为标准化（或者规格化）的对象类（参见 2.3.3 节）。它本质上是一种通用的、可支持不同应用程序的组件，正如硬件中的标准件一样，插入不同的平台或环境后即可直接运行。

值得指出的是，基于构件的开发技术（component-based development，CBD）与面向对象技术其实是一脉相承的。它给软件开发人员带来了根本的变革，就是把面向特定应用的 OO 编程，扩展为面向整个“领域”（domain）的 CBD 编程，使查找与集成适合于所需领域的构件成为这一新编程范型的主要工作。由此可见，CBD 实际上是 OO 开发的延伸与归宿。现代的网络应用程序，几乎普遍采用基于构件技术的编程范型（参见第 10 章）。

4．3 种编程范型的比较

常用编程粒度的大小来比较 3 种编程范型的差异。

过程式编程范型：着眼于程序的过程和基本控制结构，粒度最小。

面向对象编程范型：着眼于程序中的对象，粒度比较大。

基于构件技术的编程范型：着眼于适合整个领域的类对象，粒度更大。

由上可见，编程范型的演变是伴随着编程粒度的扩大而推进的，这也标志着软件开发技术的不断成熟。随着 CBD 的规范化与市场化，软件开发现已进入一个新阶段，开辟了软件应用的新纪元。

### 1.3.2 3 代软件工程

1．从编程范型到软件开发过程

前已指出，程序编码与软件开发过程是内涵不同的两个概念，但二者又密切联系，相互对应。编写程序仅是软件开发过程的一部分内容，在整个软件开发过程中，通常包括需求分析、软件设计、程序编码、软件测试等多个阶段。但程序编码是建立在编程范型的基础之上的，有什么样的编程范型，就将对应什么样的软件开发过程。以过程式编程范型为例，它采用 POPL，遵循以“清晰第一、效率第二”为目标的结构化程序设计的思想与方法，因而在软件开发过程中，也相应地采用以结构化分析与结构化设计为代表的结构化开发模型。与此相似，面向对象编程范型将对应于以 OO 分析及 OO 设计为代表的面向对象软件开发模型；基于构件技术的编程范型将对应于以领域分析与领域设计为代表的构件集成模型（参见第 2.3.3 节），从而形成了软件工程的分代。

2．软件工程的分代

经过了 40 年的发展，软件工程已经历了以下 3 代：

（1）传统软件工程或经典软件工程

它以结构化程序设计为基础，又可区分为瀑布模型、原型模型等，其开发过程一般包括以下阶段：

结构化分析 → 结构化设计 → 面向过程的编码 → 软件测试

（2）面向对象软件工程

它以面向对象程序设计为基础，其开发过程可包括以下阶段：

OO 分析与对象抽取 → 对象详细设计 → 面向对象的编码与测试

（3）基于构件的软件工程

它以软件复用为目标、领域工程为基础，其开发过程一般包括以下阶段：

领域分析和测试计划定制 → 领域设计 → 建立可复用构件库 → 按“构件集成模型”查找与集成构件

图 1.5 以简明的图示对照 3 代软件工程的比较。

由图 1.5 可见，在面向对象软件工程中，为软件确定（或抽取）类/对象的工作通常提前到分析阶段进行，在设计阶段主要完成对象内部的详细设计。随着测试工具的大量应用，在详细设计一个对象时，常常先对其关键算法和重要接口进行临时的编码和测试。当验证确认

可行后再纳入设计，临时代码稍微修改即可转换为正式代码，因而编码和测试并无明显的先后。可见在面向对象的软件工程中，从一个阶段过渡到另一个阶段，比传统的软件工程更加平滑，从而也降低了开发过程中的故障率。

- 传统软件工程

  结构化分析 → 结构化设计 → 面向过程的编码 → 软件测试

- 面向对象软件工程

  OO分析与对象抽取 → 对象详细设计 → 面向对象的编码与测试

- 基于构件的软件工程

  领域分析和测试计划定制 → 领域设计 → 建立可复用构件库 → 按“构件集成模型”查找与集成构件

图 1.5 3代软件工程的简单比较

基于构件的软件工程是以大量的可复用构件（在应用软件中可达80%以上）和测试工具为后盾的。在领域分析阶段，不仅要定义领域的体系结构，还要定义初步的用户界面。这些都需要借助测试来判断能否满足用户的需求。因此在基于构件的软件工程中，测试将进一步提前到需求阶段。换言之，软件刚刚开发，需求初步定义之后，测试工作也就开始了。

## 1.4 软件工程的应用

软件工程已提出和宣传多年，但真正获得广泛应用的时间并不太长。本节将首先讨论它在不同规模软件开发中的应用，然后简述软件工程迄今取得的成就和今后发展展望。

### 1.4.1 在各种规模软件开发中的应用

软件开发在技术和管理两个方面的复杂程度，均与软件的规模密切相关。越是规模大的软件，越要在开发和维护中严格遵守软件工程的原则和方法。表1.1列出了软件规模的分类。以下将按照这一分类，分别说明软件工程方法在各种规模软件生产中的指导作用。

表 1.1 软件规模的分类表

| 分类 | 程序规模 | 子程序数 | 开发时间 | 开发人数 |
|---|---|---|---|---|
| 极小 | 500行以下 | 10~20 | 1~4周 | 1人 |
| 小 | 1K~2K行 | 25~50 | 1~6月 | 1人 |
| 中 | 5K~50K行 | 250~1 000 | 1~2年 | 2~5人 |
| 大 | 50K~100K行 | | 2~3年 | 5~20人 |
| 甚大 | 1M行 | | 4~5年 | 100~1 000人 |
| 极大 | 1M~10M行 | | 5~10年 | 2 000~5 000人 |

1．中、小型程序

包括表 1.1 中的前半部分：极小、小和中 3 类。

极小程序大都为个人软件，由个人开发和使用，且常常只用几个月就废弃了。这类程序一般不需要正式的分析和详细的设计文档，也不必制定完整的测试计划。但即使是这类极小的程序，如能在开发中做一点分析和系统设计，遵守结构化编码和合乎规范的测试方法，对提高程序质量和减少返工，仍会有不小的帮助。

小程序包括工程师们用于求解数值问题的科学计算程序，数据处理人员生成报表或完成数据操作所用的小型商业应用程序，以及大学生们在编译原理或操作系统等课程设计中编写的程序。这类程序的长度一般不超过 2 千行，与其他外部程序也没有什么联系。开发者通常仅有一人，无须或很少需要和用户或其他开发人员打交道。在开发这类程序时，应贯彻软件工程中的技术标准和表示方法(notations)，按标准编写文档，并系统地进行复审。当然，上述工作的正规程度不必像开发大程序时那样严格。

中规模程序包括汇编程序、编译程序、小型 MIS 系统、仓库系统以及用于过程控制的一些应用程序。这类程序可能与其他程序有少量联系，也可能没有。但是在开发过程中，开发人员与用户间或开发人员之间均存在一定的联系。所以在制定软件计划、编制文档、进行阶段复审等方面，正规化的要求都比较高。在开发中如能系统地应用软件工程的原理，对改进软件质量、提高开发人员生产率和满足用户需求，都将有很大帮助。

有些人以为，软件工程适用于开发大型软件，对开发规模小的软件就缺少用武之地了，这其实是误解。许多系统软件和大多数应用软件都属于中、小型软件。如上所说，它们的开发都需要软件工程作指导。

2．大型程序

包括表 1.1 中的后半部分：大、甚大和极大 3 类。

大型编译程序、小型分时系统、数据库软件包以及某些图形软件和实时控制系统等，都是大型软件的实例。它们的编码长度可达 5 万至 10 万行，且通常与其他程序或软件系统有种种联系。开发人员一般由几个开发小组（如 3 个小组、每组 5 人）组成，在组与组间、组内不同成员间、开发人员同管理人员及用户之间，都存在着大量的通信。在长达两三年的开发过程中，中途离开或增加部分开发人员，也是常有的事。

长达百万行的软件称为甚大型软件，常见于实时处理、远程通信和多任务处理等应用领域。例如，大型的操作系统和数据库系统，军事部门的指挥与控制系统，等等。IBM/360 系列的操作系统，就是一个拥有 100 万行源程序的甚大型软件，其开发历时 5 年，参加开发人员达 5 000 人之多。今天，典型的 PC 上 Windows 操作系统的规模已增长到上百万行，不少 Windows 应用软件都包含若干个子系统，每一子系统均构成一个大型软件。

再以 2007 年我国发射的“嫦娥一号”月球卫星为例，为了在长达一年的绕月飞行中，

对离地38万千米的月球卫星实现长期管理，北京飞行控制中心开发了一个软件，统一调度各相关测控站对卫星的跟踪与测量，并通过对遥测数据的分析和计算，判断卫星在太空中的姿态、位置和星载设备的工作状况。即使在卫星携带的“星际计算机”与地面失去联系时，也能通过“自主管理”维持卫星有效地工作。这一甚大型软件包含了实时处理、长期管理、轨道控制、数据存储、指挥显示、数字仿真、国际联网等7个子系统，与星际计算机一起构成“嫦娥一号”的“大脑”和“中枢神经”。整个飞行控制软件共有104万代码行，而为了确保这些软件正确运行，还另编了160万行的测试程序。

极大型软件通常由数个甚大型的子系统构成，常含有实时处理、远程通信、多任务处理以及分布处理等软件。空中交通管制系统、洲际导弹防御系统以及某些军事指挥和控制系统，都是极大型软件的实例。它们的源代码往往长达数百万至数千万行，开发周期可能长达10年，并要求有极高的软件可靠性。

毫无疑问，所有大型以上软件的生产必须自始至终采用软件工程的方法，严格遵守标准文档格式和正规的复审制度。而且，由于大型软件本身的复杂性，对它们的计划和分析很难做到一劳永逸。因此，要十分重视管理工作，坚决贯彻软件工程关于软件管理的要求，才能避免或减少混乱。

由上可见，从极小程序到极大程序，软件工程都有它的用武之地。软件生产中是否采用工程化的方法，其结果将明显不同。

### 1.4.2 软件工程的成就与发展展望

40年的发展，软件工程取得了巨大成就。但一般以为，软件工程并非解决软件危机的灵丹妙药。国内外的实践一致说明，软件生产率的提高，总是赶不上软件需求的增长。本节将首先回顾软件工程迄今取得的成就，然后对其今后发展作一展望。

1．令人兴奋的成就

1987年，美国计算机科学家Frederick P. Brooks博士（即上文提到过的IBM/360操作系统的项目经理，被人尊称为“IBM/360计算机系列之父”），曾在一篇题为“没有银弹”（No Silver Bullet）的著名论文中，把软件生产的困难区分为本质问题和非本质问题两大类，并且指出，软件工程只能解决软件开发中出现的非本质问题，对解决本质问题仍无能为力。他总结说，从提出软件工程以后的20年（1968—1987）间，全球的软件生产率一直以6%左右的速度稳步增长，即每隔12年软件生产率大约提高一倍，其成就确实令人兴奋。但直到这篇论文发表时，并未像神话传说那样找到制服“狼人”（比喻导致软件危机的本质性问题）的“银弹”。

Brooks在这篇文章中还预期，随着面向对象设计、程序正确性证明等技术的发展，某些非本质的难题将继续得到解决，但为了解决软件生产中固有的、本质上的困难，他建议应改变软件的生产策略，例如尽可能利用现成的软件（参见第10章），采用快速原型和增量开发技术（参见第2章）等软件开发模型等。他还把取得重大突破的希望寄托在出现能够创建诸如UNIX、Pascal、Smalltalk那样产品的伟大的设计者身上，主张把鼓励和培训伟大的设计者

作为提高软件生产率的一项最重要的目标。

2．今后发展的展望

第二、三代软件工程的出现，给克服软件危机再次带来了希望。随着面向对象编程粒度的增大，软件工程师在前进的征途上陆续谱写出一些新的华章。特别是构件开发的规范化与市场化，已经把软件开发推进到一个新阶段，出现了“开发伴随软件复用，开发为了软件复用”（development with reuse，development for reuse）以及“软件就是服务”（software is service）等新思想。这些突出的成就，是否表明基于构件的软件开发，可能最终促进 Brooks 要寻找的“新”的软件生产策略的出现呢？毋庸讳言，构件开发目前还面临着许多问题需要解决，但如果继续沿着这一方向前进，会不会由此找到解决“软件生产本质困难”的“银弹”呢？前方似乎已初露曙光，读者请拭目以待。

## 1.5 软件工程的教学：本书导读

经过 40 年的实践，软件工程的基本原则现已被产业界广泛接受。全国各高等学校在计算机专业普遍开设了“软件工程”课程，有些还在非计算机专业中也设置了与之相关的课程。本书主要定位于计算机专业的本科生。根据编者多年来的教学经验和体会，在这一层次的“软件工程”课程的教学中，特别要正确认识和处理好以下的几个关系。

1．3 代软件工程的相互关系

从 20 世纪 80 年代中期以来，面向对象软件工程发展迅速，其主要技术已基本成熟，成为软件开发的主流范型。因此本版的编写宗旨，将从第 2 版“并行介绍传统的和面向对象的软件工程”，转变为“重点介绍面向对象软件工程”，对第 2 版的内容进行较大幅度的修改。具体涉及如下内容：

① 第 1、2 章仍为综述，其基本轮廓不变，但内容向面向对象软件工程倾斜。

② 传统软件工程从原来的多章压缩为一章，重点讲述以结构化分析与结构化设计为代表的结构化程序设计技术。

③ 加强面向对象软件工程的内容，把原来第 2 版的第 6、7 两章扩充为“面向对象与 UML”（第 4 章）、“面向对象分析”（第 6 章）和“面向对象设计”（第 7 章）等 3 章，并在第 6、7 这两章分别给出实例。

④ 适当反映软件工程的近期进展。除第 2 版的“软件复用”仍设专章讨论外，另增加第 14 章“软件工程高级课题”，在其中简介“Web 工程”、“基于体系结构的软件开发”、“形式化的软件开发”等新发展。

值得指出，从面向过程到面向对象再到基于构件，3 代软件工程并非相互排斥，而是“你中有我，我中有你”。前已提到，构件开发其实是面向对象开发的延伸与归宿。在第一、二两代软件工程之间，也有许多原理与方法是通用的。例如“分析先于设计”（analysis before design）、“设计先于编码”（design before coding）、“使程序（的结构）适合于问题（的结构）”（make the program fit the problem ）等精辟的警句，就是在第一代软件工程时期已经提出，并

被第二、三两代软件工程继续沿用的著名原则。在开发方法方面，也可举出很多三代一脉相承的例子。例如，分析与设计都应该提倡建立模型；编码需遵循编码范型；要加强软件工具的开发，不断改进以工具集成为特点的软件工程环境等，在各代软件工程中都十分重视。

2．软件工程技术与软件工程管理的关系

技术与管理是软件生产中缺一不可的两个方面。没有科学的管理，再先进的开发技术也不能充分发挥作用。作为主要供大学本科学生使用的教材，把重点放在软件开发技术上无疑是适当的，但也有必要讲一点软件工程管理，特别是工程管理（第 11 章）与质量管理（第 12 章）。

管理离不开度量。“靠度量来管理”（management by measurement），已成为现代管理工作的一条重要原则。软件度量学（software metrics）和软件经济学（software economics）的诞生，就是这一原则在软件工程管理中的具体体现。在本书中，对项目度量和过程度量均将作简单介绍（第 12 章）。

3．形式化方法与非形式化方法的关系

软件工程是广大软件人员长期实践活动的科学归纳与总结，包含了他们行之有效的好方法与好经验；也包含了很多学者为实现更高的软件可靠性、更高的软件生产自动化程度所进行的一系列理论课题的研究成果。这里所说的形式化方法，就是软件工程的高级研究课题之一。它是一种基于数学的开发技术，主要采用数学的方法来描述系统的性质，例如程序变换和程序验证等。而非形式化方法（亦称欠形式化方法）则主要运用文本、图、表与符号来描述系统的模型，如结构化设计、面向对象设计和 UML 语言等。

虽然形式化方法的支持者宣称这种技术将引发软件开发技术的革命，但它的实现难度很大，进展十分缓慢。软件工程师一般而言都不具备它所要求的数学基础，这也限制了它的推广。有人认为，形式化的方法加上自动化的开发环境，可能是解决这一难题的出路。

作为大学本科学生的教材，本书主要讲述非形式化方法的软件开发技术。但为了扩展读者对软件工程的了解，在有关章节仍对形式化方法作简单的介绍，例如净室模型（第 2.4.2 节）、程序正确性证明（第 12.4 节）等。

4．小程序设计与大程序设计的关系

前面已提到，有些初学者认为软件工程仅适用于大型软件的开发，对开发小规模软件并无多少用处。在高校师生中澄清这一误解，对做好软件工程的教学十分重要。诚然，软件工程是在软件规模急剧增长所引起的软件危机中诞生的，但它的基本思想和许多方法，在不同规模的软件开发中都有自己的用武之地。D. Gries 说过，“只有学会有效地编写小程序的人，才能学习有效地编写大程序。”小程序设计是大程序设计的基础，两者都需要软件工程的指导。在数年前的一次软件工程教育国际会议上，有一批教师积极倡导用软件工程来统率计算机科学与工程教学，主张在高校中普遍建立软件工程系。编者在一次与青年学生的座谈中，有一位学生在谈及学习软件工程的收获时甚至说，软件工程所倡导的“Why（软件计划）—What（软件分析）—How（软件实现）”开发过程，加上在分析与设计活动中反复强调的建模（modeling）思想，不仅是有效的工程方法，而且对指导人们的日常生活和其他工作也有着重

要的意义。

还需指出，“软件工程”课程具有很强的实践性。除了在课堂教学中要强调实用，在教材中多举一些实例外，应该尽可能在课程中安排适量的课程设计，让学生实际演练，真正达到学以致用、学用结合的目的。

## 小 结

软件工程自 1968 年提出以来，在过去 40 年中，已发展成为用于指导软件生产工程化，覆盖软件开发方法学、软件工具与环境、软件工程管理等内容的一门新学科。随着程序设计从结构化程序设计发展到面向对象程序设计，软件工程也由传统的软件工程演变为面向对象的软件工程，现正向更新一代的基于构件的软件工程迈进。

通过长期的实践，软件工程研究人员积累了许多行之有效的原理与方法，已经为产业界广泛接受与应用。大多数国内外高校都把软件工程列为本科生的教学内容。许多高校开办了软件学院，将软件工程设置为主要课程。为了做好本科这一层次的软件工程教学，本章末提出了 4 个需要正确认识的关系：3 代软件工程的相互关系；软件工程技术与软件工程管理的关系；形式化方法与非形式化方法的关系；小程序设计与大程序设计的关系，可供读者特别是初学者参考。

## 习 题

1. 什么是软件危机？为什么会产生软件危机？
2. 什么是软件生产工程化？工程化生产方法与早期的程序设计方法主要差别在哪里？
3. 分别说明软件开发方法与开发工具、软件技术与软件管理的关系。
4. 试根据你的亲身实践，谈谈软件工具在软件开发中的作用。
5. 什么是软件工程环境？谈谈你对环境重要性的认识。
6. 何谓面向对象软件工程？简述它与传统软件工程的差别和联系。
7. 软件按规模可分成哪几类？简述软件工程在各种规模的软件开发中的作用。
8. 什么是形式化软件开发方法？实现这类开发的困难和出路在哪里？