第 1 章　软件工程概述

学习目标

- 了解软件的定义、软件工程的定义和软件工程学的知识体系。
- 理解软件的特点、软件的分类、软件工程的基本开发方法等。
- 掌握软件工程的基本内容与目标、软件工程的基本原则、软件生存周期和软件开发模型等。
- 了解软件开发工具的功能、特性和分类。

1.1　软　　件

软件是一种产品，也是开发和运行产品的载体。作为一种产品，它表达了由计算机硬件体现的计算潜能。不管它是驻留在设备中，还是在主机中，软件是一个信息转换器，能够产生、管理、获取、修改、显示或转换信息。这些信息可以很简单，也可以很复杂，如多媒体信息。作为开发运行产品的载体，软件是计算机工作和信息通信的基础，也是创建和控制其他程序的基础。

信息是 21 世纪最重要的产品，软件充分地体现了这一点。通过软件处理数据，凸显了数据的重要性；软件管理商业信息，更增强了商业竞争力。软件不仅提供了通往全球信息网络的途径，而且也提供了获取信息的多种手段。

1.1.1　软件的发展

计算机硬件的发展规律基本遵循了摩尔定律，即每 18 个月芯片的性能提高一倍。但软件的发展也非常迅速，在软件体系结构上，经历了从主机结构到文件服务器结构，从客户端/服务器到基于 Internet 的浏览器/服务器结构的变化；在程序语言上，经历了从机器语言到汇编语言，从高级程序语言到第四代语言的变化；在开发工具上，经历了从分离开发工具到集成的可视化开发工具，从简单的命令行调试器到多功能调试器等变化。

1. 程序设计阶段

计算机发展的早期阶段(20 世纪 50 年代初期至 20 世纪 60 年代中期)为程序设计阶段。在这个阶段硬件已经通用化，而软件的生产却是个体化。这时，由于程序规模小，几乎没有什么系统化的方法可遵循。对软件的开发没有任何管理方法，一旦计划推迟了或

者成本提高了，程序员才开始弥补。在通用的硬件已经非常普遍的时候，软件产品还处在初级阶段，对每一类应用均需自行再设计，应用范围很有限。设计往往仅是人们头脑中的一种模糊想法，而根本不存在文档。

2. 程序系统阶段

计算机系统发展的第二阶段(20 世纪 60 年代中期至 20 世纪 70 年代末期)为程序系统阶段。多道程序设计、多用户系统引入了人机交互的新概念。交互技术打开了计算机应用的新世界，硬件和软件配合达到了一个新层次，出现了实时系统和第一代数据库管理系统。这个阶段的另一个特点就是软件产品的使用和“软件作坊”的出现。被开发的软件可以在较宽广的范围中应用。主机和计算机上的程序能够有数百甚至上千的用户。

在软件的使用过程中，当发现错误或硬件环境发生变化时都需要修改软件，这些活动统称为软件维护。在软件维护上的花费以惊人的速度增长。更为严重的是，许多程序的个体化特性使得它们根本不能维护。“软件危机”出现了。

3. 软件工程阶段

计算机系统发展的第三阶段始于 20 世纪 70 年代中期并跨越了近 10 年，被称为软件工程阶段。在这一阶段，以软件的产品化、系列化、工程化、标准化为特征的软件产业发展起来，打破了软件生产的个体化特征，有了可以遵循的软件工程化的设计原则、方法和标准。在分布式系统中，各台计算机同时执行某些功能，并与其他计算机通信，极大地提高了计算机系统的复杂性。广域网、局域网、高带宽数字通信以及对“即时”数据访问需求的增加都对软件开发者提出了更高的要求。

4. 第四个阶段

计算机系统发展的第四阶段已经不再着重于单台计算机和计算机程序，而是针对计算机和软件的综合影响。由复杂操作系统控制的强大的桌面机、广域网络和局域网络，配以先进的软件应用已成为标准。计算机体系结构迅速地从集中的主机环境转变为分布的客户端/服务器环境。世界范围的信息网提供了一个基本结构，信息高速公路和网际空间连通已成为令人关注的热点问题。事实上，Internet 可以看做是能够被单个用户访问的软件，计算机正朝着社会信息化和软件产业化方向发展，从技术的软件工程阶段过渡到社会信息化的计算机系统阶段。随着第四阶段的进展，一些新技术开始涌现。面向对象技术将在许多领域中迅速取代传统软件开发方法。表 1-1 给出了计算机发展 4 个阶段的典型技术。

表 1-1 4 个阶段的典型技术

阶　段	第一阶段	第二阶段	第三阶段	第四阶段
典型技术	• 面向批处理 • 有限的分布 • 自定义软件	• 多用户 • 实时 • 数据库 • 软件产品	• 分布式系统 • 嵌入“智能” • 低成本硬件 • 消费者的影响	• 强大的桌面系统 • 面向对象技术 • 专家系统 • 并行计算 • 云计算

1.1.2 软件的定义

计算机系统通过运行程序来实现各种不同应用。各种不同功能的程序，包括用户为自己的特定目的编写的程序、检查和诊断机器系统的程序、支持用户应用程序运行的系统程序、管理和控制机器系统资源的程序等，通常称为软件。它是计算机系统中与硬件相互依存的另一部分，与硬件合为一体完成系统功能。软件定义如下：

(1) 在运行中能提供所希望的功能和性能的指令集(即程序)；

(2) 使程序能够正确运行的数据结构；

(3) 描述程序研制过程和方法所用的文档。

随着计算机应用的日益普及，软件变得越来越复杂，规模也越来越大，这就使得人与人、人与机器间相互沟通，保证软件开发与维护工作的顺利进行显得特别重要，因此，文档(即各种报告、说明、手册的总称)是不可缺少的。特别是在软件日益成为产品的今天，文档的作用就更加重要。

1.1.3 软件的特点

在计算机系统中软件是一个逻辑部件，而硬件是一个物理部件。因此，软件相对硬件而言有许多特点。为了能全面、正确地理解计算机软件及软件工程的重要性，必须了解软件的特点。软件的特点可归纳如下：

(1) 软件是一种逻辑实体，而不是具体的物理实体，因而它具有抽象性。这个特点使它与计算机硬件或其他工程对象有着明显的差别。人们可以把它记录在介质上，但却无法看到软件的形态，而必须通过测试、分析、思考、判断去了解它的功能、性能及其他特性。

(2) 软件是通过人们的智力活动，把知识与技术转化成信息的一种产品，是在研制、开发中被创造出来的。一旦某一软件项目研制成功，以后就可以大量地复制同一内容的副本，这就是软件的生产过程。也就是说，软件的研制成本远远大于其生产成本。软件故障往往是在开发时产生而在测试时没有被发现的问题，所以要保证软件的质量，必须着重于软件开发过程，加强管理。

(3) 在软件的运行和使用期间，没有硬件那样的机械磨损、老化问题。软件维护比硬件维护要复杂得多，与硬件的维修有着本质的差别，如图 1-1～图 1-3 所示。图 1-1 所示是硬件的故障率随时间变化的曲线，图 1-2 所示是在理想情况下软件故障率随时间变化的曲线，图 1-3 所示是软件的实际故障率曲线。

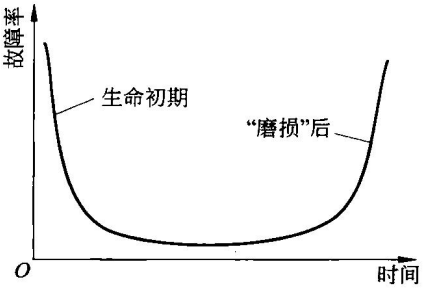

图 1-1 硬件的故障率随时间变化的曲线

(4) 软件的开发和运行经常受到计算机系统的限制，对计算机系统有着不同程度的依赖。为了解除这种依赖，在软件开发中提出了软件移植的问题，并且把软件的可移植性作为衡量软件质

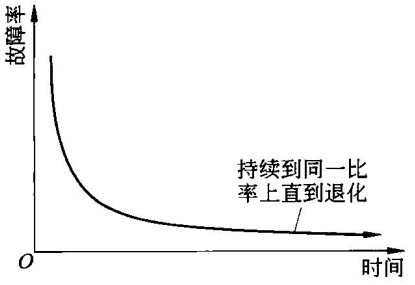

图 1-2 理想情况下的软件故障率随时间变化的曲线

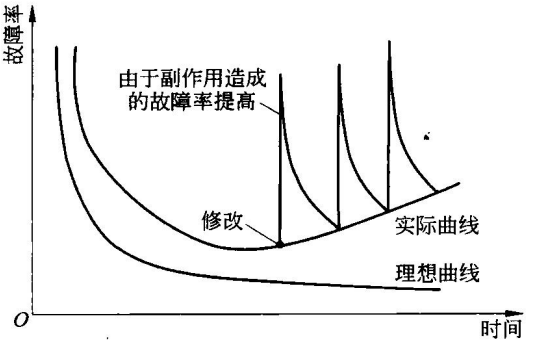

图 1-3 软件的实际故障率曲线

量的因素之一。

(5) 软件的开发尚未完全摆脱手工的开发方式。软件开发的效率受到很大的限制。因此,应促进软件技术的进展,提出和采用新的开发方法。例如,近年来出现的充分利用现有软件的复用技术、自动生成技术和其他一些有效的软件开发工具或软件开发环境,既方便了软件开发的质量控制,又提高了软件开发的效率。

(6) 软件的开发费用越来越高。软件的研制工作需要投入大量的、复杂的、高强度的脑力劳动,需要较高的成本。

(7) 软件的开发是一个复杂的过程,例如银行管理系统涉及安全等问题,因而管理是软件开发过程中必不可少的内容。

(8) 软件的生产过程较简单,软件生产过程就是复制过程。

1.1.4 软件的分类

人们在工作和学习中经常接触到各式各样的软件,那么这些数量众多的软件究竟分为哪些类型,这就要考虑对计算机软件进行分类的依据。但事实上,要给出一个科学的、统一的、严格的计算机分类标准是不现实的。而对软件的类型进行必要的划分,对于不同类型的工程对象采用不同的开发和维护方法是很有价值的,因此有必要从不同角度对计算机软件做适当的分类。

1. 基于软件功能的划分

(1) 系统软件。与计算机硬件紧密配合以使计算机各个部件与相关软件及数据协调、高效工作的软件。例如,操作系统、数据库管理系统等。系统软件在工作时频繁地与硬件交互,以便为用户服务,共享系统资源,在这中间伴随着复杂的进程管理和复杂的数据结构的处理。系统软件是计算机系统必不可少的重要组成部分。

(2) 支撑软件。协助用户开发软件的工具性软件,包括帮助程序人员开发软件产品的工具和帮助管理人员控制开发进程的工具。它可分为以下几类。

• 一般类型:包括文本编辑程序、文件格式化程序和程序库系统等。

- 支持需求分析：包括 PSL/PSA 问题描述语言、问题描述分析器、关系数据库系统、一致性检验程序等。
- 支持设计：包括图形软件包、结构化流程图绘图程序、设计分析程序、程序结构图编辑程序等。
- 支持实现：包括编译程序、交叉编译程序、预编译程序和连接编译程序等。
- 支持测试：包括静态分析程序、符号执行程序、模拟程序和测试覆盖检验程序等。
- 支持管理：包括 PERT 进度计划评审方法、绘图程序、标准检验程序和库管理程序等。

(3) 应用软件。在特定领域内开发，为特定目的服务的一类软件。现在几乎所有的国民经济领域都使用了计算机，为这些计算机应用领域服务的应用软件种类繁多。其中商业数据处理软件是占比例最大的一类，工程与科学计算软件大多属于数值计算问题。应用软件还包括计算机辅助设计/计算机辅助制造(CAD/CAM)、系统仿真、智能产品嵌入软件(如汽车油耗控制、仪表盘数字显示、刹车系统)及人工智能软件(如专家系统、模式识别)等。此外，在事务管理、办公自动化、中文信息处理、计算机辅助教学(CAI)等方面的软件也得到迅速发展，产生了惊人的生产效率和巨大的经济效益。

2. 基于软件工作方式划分

(1) 实时处理软件。指在事件或数据产生时，立即处理，并及时反馈信号，控制需要监测和控制过程的软件。主要包括数据采集、分析和输出三部分，其处理时间是应严格限定的，如果在任何时间超出了这一限制，都将造成事故。

(2) 分时软件。允许多个联机用户同时使用计算机的软件。系统把处理机时间轮流分配给各联机用户，使各用户都感到只有自己在使用计算机。

(3) 交互式软件。能实现人机通信的软件。这类软件接收用户给出的信息，但在时间上没有严格的限定，这种工作方式给予用户很大的灵活性。

(4) 批处理软件。把一组输入作业或一批数据以成批处理的方式一次运行，按顺序逐个处理的软件。

3. 基于软件规模的划分

根据开发软件所需的人力、时间以及完成的源程序行数，可划分为下述 6 种不同规模的软件。

(1) 微型软件。指一个人在几天之内完成的、程序不超过 500 行语句且仅供个人专用的软件。通常这类软件没有必要做严格的分析，也没有必要有完整的设计、测试资料。

(2) 小型软件。指一个人在半年之内完成的 2000 行以内的程序。这种程序通常没有与其他程序的接口，但需要按一定的标准化技术、正规的资料书写以及定期的系统审查，只是没有大型软件那样严格。

(3) 中型软件。5 个人以内在一年多时间内完成的 5000～50 000 行的程序。中型软件开始出现了软件人员之间、软件人员与用户之间的联系和协调的配合关系问题，因而计划、资料书写以及技术审查需要比较严格地进行。在开发中使用系统的软件工程方法是

完全必要的，这对提高软件产品质量和程序开发人员的工作效率起着重要的作用。

(4) 大型软件。指 5～10 个人在两年多的时间内完成的 50 000～100 000 行的程序。参加工作的软件人员需要进行二级管理。对于这样规模的软件，采用统一的标准、实行严格的审查是绝对必要的。由于软件的规模庞大以及问题的复杂性，往往在开发过程中会出现一些事先难以估计的不测事件。

(5) 甚大型软件。指 100～1000 人在 4～5 年时间里完成的具有 100 万行的程序。这种甚大型项目可能会划分成若干个子项目，每一个子项目都是一个大型软件。子项目之间具有复杂的接口。例如，实时处理系统、远程通信系统、多任务系统、大型操作系统、大型数据库管理系统通常有这样的规模。很显然，如果这类问题没有软件工程方法的支持，它的开发工作是不可想象的。

(6) 极大型软件。指 2000～5000 人在 10 年内完成的 1000 万行以内的程序。这类软件很少见，往往用于军事指挥、弹道导弹防御系统等。

可以看出，规模大、时间长、多人参加的软件项目，其开发工作必须要有软件工程的知识做指导，而规模小、时间短、参加人员少的软件项目的开发也要用到软件工程的概念和开发规范，其基本原则是一样的。

4. 基于软件失效的影响进行划分

工作在不同领域的软件，在运行中对可靠性也有不同的要求。事实上，随着计算机进入国民经济各个重要领域，软件的可靠性越来越重要。人们一般称这类软件为关键软件，其特点在于：

(1) 可靠性要求高。

(2) 常与完成重要功能的大系统的处理部件相连。

(3) 含有的程序可能对人员、公众、设备或设施的安全造成影响。还可能影响到环境的质量和国家的安全。

5. 基于软件服务对象的范围进行划分

软件工程项目完成后可以有两种情况：

(1) 定制软件。受某个特定客户(或少数客户)的委托，由一个或多个软件开发机构在合同的约束下开发出来的软件。

(2) 产品软件。由软件开发机构开发出来直接提供给市场，或是为众多用户服务的软件。

1.2 软件工程的内容与方法

由于微电子学技术的进步，计算机硬件的性能有了很大的进步，而且质量稳步提高。然而，计算机软件成本却不断上升，质量也不尽如人意，软件开发的生产率也远远不能满足计算机应用的要求。软件已经成为限制计算机系统进一步发展的关键因素。

更为严重的是，计算机系统发展早期所形成的一系列错误概念和做法已经严重地阻碍了计算机软件的开发，甚至有的大型软件根本无法维护，只能提前报废，造成大量人力、

物力的浪费，从而导致软件危机。为了研究解决软件危机的方法，计算机科学技术领域中一门新兴的学科逐步形成了，这就是计算机软件工程学。

1.2.1　软件危机与软件工程的定义

1. 软件危机

软件危机指的是软件开发和维护过程中遇到的一系列严重问题。软件危机包含下述两方面的问题：如何开发软件，怎样满足对软件日益增长的需求；如何维护数量不断膨胀的已有软件。具体地说，软件危机主要有下列表现：

(1) 产品不符合用户的实际需要。

(2) 软件开发生产率提高的速度远远不能满足客观需要，软件的生产率远远低于硬件生产率和计算机应用的增长速度，使人们不能充分利用现代计算机硬件提供的巨大潜力。

(3) 软件产品的质量差。软件可靠性和质量保证的定量概念刚刚出现不久，软件质量保证技术(审查、复审和测试)没有贯穿到软件开发的全过程中，这些都导致软件产品发生质量问题。

(4) 对软件开发成本和进度的估计常常不准确。实际成本比估计成本有可能高出一个数量级，实际进度比预期进度拖延几个月甚至几年。这种现象降低了软件开发者的信誉。而为了赶进度和节约成本所采取的一些权宜之计又往往降低了软件产品的质量，从而不可避免地会引起用户的不满。

(5) 软件的可维护性差。很多程序中的错误是难以改正的，实际上不能使这些程序适应硬件环境的改变，也不能根据用户的需要在原有程序中增加一些新的功能。没能实现软件的可重用，人们仍然在重复开发功能类似的软件。

(6) 软件文档资料通常既不完整也不合格。

(7) 软件的价格昂贵，软件成本在计算机系统总成本中所占的比例逐年上升。

2. 软件工程的定义

(1) 软件开发应是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目，必须充分吸取和借鉴人类长期以来从事各种工程项目所积累的行之有效的原理、概念、技术和方法，应该推广使用在实践中总结出来的软件开发的成功技术和方法，并且研究探索更好更有效的技术和方法，尽快消除在计算机系统早期发展阶段形成的一些错误概念和做法。将软件的生成在时间上分成若干阶段以便于分步而有计划地分工合作，在结构上简化若干逻辑模块。把软件作为工程产品来处理，按计划、分析、设计、实现、测试、维护的周期进行生产。采用工程化方法和途径来开发与维护软件。

(2) 应该开发和使用更好的软件工具。在软件开发的每个阶段都有许多烦琐重复的工作需要做，在适当的软件工具辅助下，开发人员可以把这类工作做得既快又好。如果把各个阶段使用的软件工具有机地集合成一个整体，支持软件开发的全过程，则称为软件工程支撑环境。

(3) 采取必要的管理措施。软件产品是把思维、概念、算法、组织、流程、效率、质量等

多方面问题融为一体的产品。但它本身是无形的，所以有不同于一般工程项目的管理。它必须通过人员组织管理、项目计划管理和配置管理等来保证软件按时高质量完成。

总之，为了解决软件危机，既要有技术措施（包括方法和工具），又要有必要的组织管理措施。软件工程正是从管理和技术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。

软件工程是指导计算机软件开发和维护的一门学科。它采用工程的概念、原理、技术和方法，把经过时间考验而证明是正确的管理技术和当前能够得到的最好的技术方法结合起来，用于开发与维护软件。

1968 年，在前联邦德国召开的国际会议上正式提出并使用了“软件工程”这个术语，运用工程学的基本原理和方法来组织和管理软件生产。后来还发展了与软件有关的心理学、生理学和经济学等方面的学科。在这期间，研究软件工程的专家学者们陆续提出了 100 多条关于软件工程的准则。这 100 多条软件工程准则可以概括为下述 6 条基本原则。

（1）用分阶段的生存周期计划严格管理。一个软件从定义到开发、使用和维护，直到最终被废弃，要经历一个漫长的时期，通常把软件经历的这个漫长时期称为生存周期。在软件开发与维护的漫长过程中，需要完成许多不同性质的工作。所以应把软件生存周期划分为若干个阶段，并相应地制定出可行的计划，然后按照这个计划对软件的开发与维护工作进行管理。不同层次的管理人员都必须严格按照计划各尽其职，绝不能受客户或上级人员的影响而擅自背离预定计划。

（2）坚持进行阶段评审。软件的质量保证工作不能等到编码阶段结束之后再进行。其理由是：

① 大部分错误是在编码之前造成的，根据统计，设计错误占软件错误的 63%，编码错误仅占软件错误的 37%。

② 错误发现与改正得越晚，所需付出的代价也越高。因此，在每个阶段都进行严格的评审，以便尽早发现在软件开发过程中所犯的错误是一条必须遵循的重要原则。

（3）实行严格的产品控制。在软件开发过程中不应随意改变需求，因为改变一项需求往往需要付出较高的代价。但是，在软件开发过程中改变需求又是难免的，由于外部环境的变化，相应地改变用户需求是客观需要。显然不能硬性禁止客户提出改变需求的要求，而只能依靠科学的产品控制技术来顺应这种要求。也就是说，当改变需求时，为了保持软件各个配置成分的一致性，必须实行严格的产品控制，其中主要是实行基准配置管理。所谓基准配置又称为基线配置，它们是经过阶段评审后的软件配置成分（各个阶段产生的文档或程序代码）。基准配置管理也称为变动控制，一切有关修改软件的建议，特别是涉及对基准配置的修改建议，都必须按照严格的规程进行评审，获得批准以后才能实施修改，绝对不能随意进行修改。

（4）采用现代程序设计技术。从提出软件工程的概念开始，人们一直把主要精力用于研究各种新的程序设计技术。20 世纪 60 年代末提出的结构程序设计技术已经成为绝大多数人公认的先进的程序设计技术。以后又进一步发展了各种结构分析（SA）与结构设计（SD）技术。实践表明，采用先进的技术既可提高软件开发的效率，又可提高软件维

护的效率。

(5) 应能清楚地审查结果。软件产品不同于一般的物理产品，它是看不见、摸不着的逻辑产品。软件开发人员(或开发小组)的工作进展情况可见性差，难以准确度量，从而使得软件产品的开发过程比难以评价和管理。为了提高软件开发过程的可见性，更好地进行管理，应该根据软件开发项目的总目标及完成期限，规定开发组织的责任和产品标准，从而使得所得到的结果能够清楚地被审查。

(6) 合理安排软件开发小组的人员。合理安排软件开发小组人员的原则是少而精。高素质的人员会大大提高软件的开发效率，且明显减少软件中的错误。此外，开发小组人员少，也可减小因讨论问题和交流情况而造成的通信开销。

最后要强调的是，必须不断灵活地改进软件工程实践。按照这个要求，就要积极主动采用新的软件技术，而且要注意不断总结经验。

1.2.2　软件工程的基本内容与目标

1. 软件工程的基本内容

从内容上划分，软件工程学可分为理论、结构、方法、工具、环境、管理和规范等。理论与结构是软件开发的技术基础，包括程序正确性证明理论、软件可靠性理论、软件成本估算模型、软件开发模型和模块划分原理等。软件开发技术包括软件开发方法学、软件工具和软件开发环境。良好的软件工具可促进方法的研制，而先进的软件开发方法能改进工具。软件工具的集成构成软件开发环境。管理技术是提高开发质量的保证，软件工程管理包括软件开发管理和软件经济管理，前者包括人员分配、制定计划、确定标准与配置，而后者的主要内容有成本估算和质量评价。

2. 软件工程学的目标

软件工程学研究的基本目标是：

(1) 定义良好的方法学。即面向计划、开发维护整个软件生存周期的方法学。

(2) 确定软件成分。记录软件生存周期每一步的软件文件资料，按步显示轨迹。

(3) 可预测结果。在生存周期中，每隔一定时间可以进行复审。

软件工程学的最终目的是以较少的投资获得易维护、易理解、可靠、高效率的软件产品。软件工程学是研究软件结构、软件设计与维护方法、软件工具与环境、软件工程标准与规范、软件开发技术与管理技术的相关理论。

1.2.3　软件的基本开发方法

软件工程中的开发方法就是指软件工程方法论。常用的软件开发方法有如下三种，即面向过程方法、面向对象方法和敏捷开发方法。

1. 面向过程方法

面向过程方法又称为结构化方法，这种方法主要包括面向过程需求分析、面向过程设计、面向过程编程、面向过程测试和面向过程管理等。面向过程方法的特点是程序的执行过程完全有程序控制，不由用户控制。其优点是简单实用，缺点是维护困难。

结构化方法开始于20世纪60年代，成熟于70年代，盛行于80年代。该方法强调自顶向下、逐步求精，编程时强调结构化程序设计。

2. 面向对象方法

面向对象方法开始于20世纪80年代，兴起于90年代。现已广泛应用。

面向对象方法主要包括面向对象需求分析、面向对象设计、面向对象编程、面向对象测试和面向过程管理等。面向对象方法的基本特点是将对象的属性和方法封装起来，形成信息系统的基本执行单位，再利用对象的继承特性，由基本执行单位派生出其他执行单位，从而产生许多新的对象。将这些离散对象通过事件或消息连接起来就形成了软件系统。面向对象方法的特点是程序的执行过程不由程序员控制，完全由用户交互控制。其优点是易于维护，缺点是较难掌握。

3. 敏捷开发方法

开发方法就是解决问题的过程，敏捷开发方法是来源于开发实践的总结。敏捷开发方法将整个软件开发活动划分成一系列短的迭代过程，每个迭代完成一定数目的性能。迭代的周期应该尽量短。更为重要的是，迭代应该是由测试和反馈驱动的。只有这样，才能为持续地改进提供一个良好的基础和安全网络。这里的每个迭代周期产出的都是一个经过验证的可用产品，只是可能功能不全，并且这是一个有意识的、持续的基于反馈的改进过程，而不是简单的选定。其实所有成功的项目开发活动都是接近这个标准的，只不过敏捷开发方法把它放在了最为重要的位置上。

为了有效地达到上面所说的效果，除了需要一些技术方面的技能外(如重构技能等)。还需要一个能够对上面这种形式的活动进行有效支撑的环境，这个环境应该是所有想取得成功的项目的基础，也就是一个持续集成环境。持续集成为有效地达到敏捷开发方法提供了基础。

1.2.4 软件工程的基本原则

为了开发出低成本、高质量的软件产品，软件工程学应遵守以下基本原则。

1. 分解

分解是人类分析解决复杂问题的重要手段和基本原则，其基本思想是从时间上或是从规模上将一个复杂抽象问题分成若干个较小的、相对独立的、容易求解的子问题，然后分别求解。例如，软件瀑布模型、结构化分析方法、结构化设计方法、Jackson方法、模块化设计等都运用了分解的原则。

2. 抽象和信息隐蔽

尽量将可变因素隐藏在一个模块内，将怎样做的细节隐藏在下层，而将做什么抽象到上一层做简化，从而保证模块的独立性。这就是软件设计独立性要遵守的基本原则。模块化和局部性的设计过程使用了抽象和信息隐蔽的原则。

3. 一致性

研究软件工程方法的目的之一就是要使开发过程标准化、统一化，使软件产品设计有

共同遵循的原则，要求软件文件格式一致，工作流程一致。

4. 确定性

软件开发过程要用确定的形式表达需求，表达的软件功能应该是可预测的，用可测试性、易维护性、易理解性和高效率等指标来具体度量软件质量。

组织实施软件工程项目，主要要达到的目标有开发成本较低，软件功能要达到用户要求并具有较好的性能，要有良好的可移植性，易于维护且维护费用较低，能按时完成并及时交付使用。

1.3　软件生存周期与软件开发模型

软件工程采用的生存周期方法就是从时间角度对软件的开发与维护这个复杂问题进行分解，将漫长的软件生存时期分为若干阶段，每个阶段都有其相对独立的任务，然后逐步完成各个阶段的任务。

1.3.1　软件生存周期

软件生存周期就是指从提出软件产品开始，直到该软件产品被淘汰的全过程。研究软件生存周期是为了更科学、更有效地组织和管理软件的生产，从而使软件产品更可靠、更经济。采用软件生存周期来使软件开发分阶段依次进行，前一个阶段任务的完成是后一个阶段任务的前提和基础，而后一个阶段通常是将前一个阶段提出的方案进一步具体化。每一个阶段的开始与结束都有严格的标准，每一个阶段结束之前都要接受严格的技术和管理评审。采用软件生存周期的划分方法，使每一阶段的任务相对独立，有利于简化问题且便于不同人员分工协作。而且其严格而科学的评审制度保证了软件的质量，提高了软件的可维护性，从而大大提高了软件开发的成功率和生产率。

软件生存周期一般可分为 8 个阶段：问题定义、软件项目可行性研究、需求分析、概要设计、详细设计、编码、测试、运行与维护。

在软件的研制和开发过程中：

(1) 要了解和分析用户的问题，以及经济、技术和时间等方面的可行性。

(2) 将用户的需求规范化、形式化，编写成需求说明书及初步的系统用户手册，提交评审。

(3) 将软件需求设计为软件过程描述，即设计人员将确定的各项需求转化成一个相应的体系结构。结构的每一组成部分都是意义明确的模块，每个模块都与某些需求相对应(概要设计)。然后对每个模块的具体任务进行具体的描述(详细设计)。

(4) 编代码，就是把过程描述编为机器可执行的代码。

(5) 测试，发现错误，进行改正。

(6) 维护，包括故障的排除以及为适应使用环境的变化和用户对软件提出新的要求所作的修改。

根据上述过程，可以将软件生存周期分为三个大的阶段，即计划阶段、开发阶段和维

护阶段。

1. 计划阶段

计划阶段又分为软件计划和需求分析两步。第一步，因为软件是计算机系统中一个子系统，这样不但要从确定的软件子系统出发，确定工作域，即确定软件总的目标、功能等，开发这样的软件系统需要哪些资源(人力和设备)，作出成本估算；而且还要求作出可行性分析，即在现有资源与技术的条件下能否实现这样的目标；最后要提出进度安排，并写出软件计划文档。上述问题都要进行管理评审。第二步，在管理评审通过以后，要确定系统定义和有效性标准(软件验收标准)，写出软件需求说明书。还要开发一个初步用户手册，进行技术评审。技术评审通过以后，再进行一次对软件计划的评审，因为这时对问题有了进一步的了解。所以对制定的计划需要进行多次修改，以尽量满足各种要求，然后再进入到开发阶段。

2. 开发阶段

开发阶段要经过三个步骤：设计、编码和测试。首先对软件进行结构设计，定义接口，建立数据结构，规定标记。接着对每个模块进行过程设计、编码和单元测试。最后进行组合测试和有效性测试，对每一个测试用例和结果都要进行评审。

3. 维护阶段

首先要做的工作就是配置评审，即检查软件文档和代码是否齐全、两者是否一致、是否可以维护等，然后要确定维护组织和职责，并定义表明系统错误和修改报告的格式。维护可分为改正性维护、完善性维护和适应性维护等。维护内容广泛，有人把维护看成是第二次开发。要适应环境的变化，就要扩充和改进，但不是建立新系统。维护的内容应该通知用户，要得到用户的认可。然后可进行修改，修改不只是代码修改，必须要有齐全的修改计划、详细过程以及测试等文档。

以上简要介绍了软件生存周期各个阶段的主要任务和评审标准。以后本书将围绕软件生存周期的各个阶段详细讲述其所要完成的任务和完成这些任务所需要的技术方法和辅助工具，以及软件开发和维护的主要管理技术。

1.3.2 软件开发模型

为了反映软件生存周期内各种工作应如何组织及软件生存周期各个阶段应如何衔接，需要用软件开发模型给出直观的图示表达。软件开发模型是软件工程思想的具体化，是实施于过程模型中的软件开发方法和工具，是在软件开发实践中总结出来的软件开发方法和步骤。总的来说，软件开发模型是跨越整个软件生存周期的系统开发、运作、维护所实施的全部工作和任务的结构框架。

1. 瀑布模型

瀑布模型即生存周期模型，由 B. M. Boehm 提出，是软件工程的基础模型。其核心思想是按工序将问题化简，将功能的实现与设计分开，便于分工协作。采用结构化的分析与设计方法，将逻辑实现与物理实现分开。瀑布模型规定了各项软件工程活动，包括制定开

发计划、进行需求分析和说明、软件设计、程序编码、测试及运行维护，并且规定了软件生存周期的各个阶段如同瀑布流水，逐级下落、自上而下、相互衔接的固定次序。如图 1-4 所示，模型中主要阶段映射为一些基本的开发活动，每项开发活动均应具有下述特征：

(1) 从上一项活动接收该项活动的工作对象，作为输入。

(2) 利用这一输入实施该项活动应完成的内容。

(3) 给出该项活动的工作成果，作为输出传给下一项活动。

(4) 对该项活动实施的工作进行评审，若其工作得到确认，则继续进行下一项活动；否则返回前项，甚至更前项的活动进行返工。

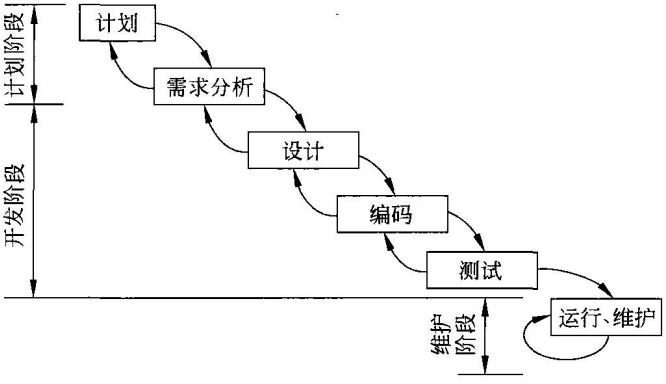

图 1-4　瀑布模型

瀑布模型为软件开发和软件维护提供了一种有效的管理图式。根据这一图式制定开发计划、进行成本预算、组织开发力量，以项目的阶段评审和文档控制为手段，有效地对整个开发过程进行指导，从而保证了软件产品及时交付，并达到预期的质量要求。与此同时，瀑布模型在大量的软件开发实践中也逐渐暴露出它的严重缺点。其中最为突出的缺点是该模型缺乏灵活性，特别是无法解决软件需求不明确或不准确的问题。这些问题的存在对软件开发带来严重影响，最终可能导致开发出的软件并不是用户真正需要的软件。并且，由于瀑布开发模型具有顺序性和依赖性，凡后一阶段出现的问题需要通过前一阶段的重新确认来解决，因此其代价十分高昂。而且，随着软件开发项目规模的日益庞大，由于瀑布模型不够灵活等缺点引发出的上述问题显得更为严重。软件开发需要人们合作完成，因此人员之间的通信和软件工具之间的联系以及开发工作之间的并行和串行等都是必要的，但瀑布模型中并没有体现出这一点。

2. 螺旋模型

为克服瀑布模型的不足，近年来已经提出了多种其他模型。对于复杂的大型软件，开发一个原型往往达不到要求。螺旋模型将瀑布模型与演化模型结合起来，并且加入的两种模型均忽略了风险分析，弥补了两者的不足。

“软件风险”是普遍存在于任何软件开发项目中的实际问题。对于不同的项目，其差别只是风险有大有小而已，在制定软件开发计划时，系统分析员必须回答：项目的需求是什么，需要投入多少资源以及如何安排开发进度等一系列问题。然而，若要他们当即给出

准确无误的回答是不容易的，甚至几乎是不可能的，但系统分析员又不可能完全回避这一问题。从经验的估计出发给出初步的设想便难免带来一定风险。实践表明，项目规模越大，问题越复杂，资源、成本和进度等因素的不确定性越大，承担项目所冒的风险也越大。总之，风险是软件开发不可忽视的潜在不利因素，它可能在不同程度上影响软件开发过程或软件产品的质量。软件风险分析的目标是在造成危害之前及时对风险进行识别、分析，采取对策，进而消除或减少风险的损害。螺旋模型沿着螺线旋转，如图 1-5 所示，在笛卡儿坐标的 4 个象限上分别表达了 4 个方面的活动，即：

（1）制定计划。确定软件目标，选定实施方案，弄清项目开发的限制条件。

（2）风险分析。分析所选方案，考虑如何识别和消除风险。

（3）实施工程。实施软件开发。

（4）客户评估。评价开发工作，提出修正建议。

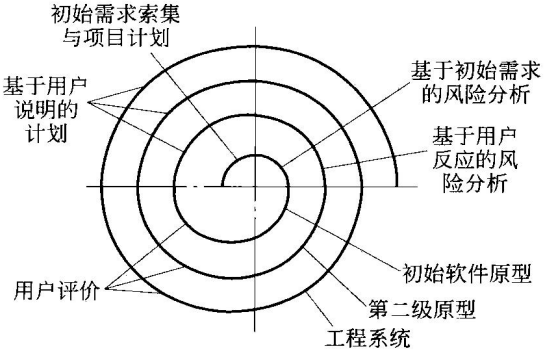

图 1-5　螺旋模型

沿螺线自内向外每旋转一圈便开发出一个更为完善的新的软件版本。例如，在第一圈确定了初步的目标、方案和限制条件以后，转入右上象限，对风险进行识别和分析。如果风险分析表明需求有不确定性，那么在右下的工程象限内所建的原型会帮助开发人员和客户考虑其他开发模型，并对需求做进一步修正。客户对工程成果做出评价之后，给出修正建议。在此基础上需再次计划，并进行风险分析。在每一圈螺线上做出风险分析的终点是能否继续下去的判断。假如风险过大，开发者和用户无法承受，项目有可能终止。多数情况下，沿螺线的活动会继续下去，自内向外，逐步延伸，最终得到所期望的系统。

如果软件开发人员对所开发项目的需求已有了较好的理解或较大的把握，则无需开发原型，可采用普通的瀑布模型，这在螺旋模型中可认为是单圈螺线。与此相反，如果对所开发项目需求理解较差，则需要开发原型，甚至需要不止一个原型的帮助，那就需要经历多圈螺线。在这种情况下，外圈的开发包含了更多的活动，也可能某些开发采用了不同的模型。

螺旋模型适合于大型软件的开发，应该说它是最为实际的方法，它吸收了软件工程“演化”概念，使得开发人员和客户对每个演化层出现的风险有所了解，继而做出应有的反映。螺旋模型的优越性比起其他模型来说是明显的，但并不是绝对的。要求许多客户接受和相信此方法并不容易。这个模型的使用需要开发人员具有相当丰富的风险评估经验

和专门知识。如果风险较大，又未能及时发现，势必造成重大损失。此外，螺旋模型是出现较晚的新模型，远没有瀑布模型普及，要让广大软件人员和用户充分肯定它，还有待于更多的实践。

3. 第四代技术模型

第四代技术(4GT)包含了一系列软件工具，它们的共同点是能使软件设计者在较高级别上说明软件的某些特征，然后软件工具根据说明自动生成源代码。在越高的级别上说明软件，就能越快地构造出程序。软件工程的第四代技术模型的应用关键在于软件描述的能力，它用一种特定的语言来完成或者以一种用户可以理解的问题描述方法来描述需解决的问题。

目前，支持第四代技术模型的软件开发环境及工具有数据库查询的非过程语言、报告生成器、数据操纵、屏幕交互及定义，以及代码生成、高级图形功能、电子表格功能。最初，上述的许多工具仅能用于特定应用领域，而今天，第四代技术环境已经扩展，能够满足许多软件应用领域的需要。

像其他模型一样，第四代技术模型也是从需求分析开始，在理想情况下，用户能够描述出需求，而且这些需求能被直接转换成可操作原型。但这是不现实的，因为用户可能不确定需要什么；在说明已知的事实时，可能出现二义性；可能不能够或是不愿意采用一个第四代技术工具可以理解的形式来说明信息。因此，其他模型中所描述的用户对话方式在第四代技术模型中仍是一个必要的组成部分。

对于较小型的应用软件，使用一个非过程的第四代语言有可能直接从需求分析过渡到实现。但对于较大的应用软件，就有必要制定一个系统的设计策略。对于较大项目，如果没有很好地设计，即使使用第四代技术也会产生不用任何方法来开发软件所遇到的同样问题，这些问题包括质量低、可维护性差、难以被用户接受等。

应用第四代技术的生成功能使得软件开发者能够以一种方式表示期望的输出，这种方式可以自动生成该输出的代码。很显然，相关信息的数据结构必须已经存在，且能够被第四代技术访问。

要将一个第四代技术模型生成的功能变成最终产品，开发者还必须进行测试，写出有意义的文档，并完成其他软件工程模型中同样要求的所有集成活动。此外，采用第四代技术开发的软件还必须考虑维护是否能够迅速实现。

像其他所有软件工程模型一样，第四代技术模型也有优点和缺点。其优点是极大地缩短了软件的开发时间，并显著地提高了构造软件的生产率。缺点是目前的第四代技术并不比程序设计语言更容易使用，而且这类工具生成的结果源代码是“低效的”，使用第四代技术开发的大型软件系统的可维护性是令人怀疑的。

综上所述，可以概括如下：

(1) 在过去十余年中，第四代技术模型发展很快，且目前已成为适用于多种不同应用领域的方法。与计算机辅助软件工程(CASE)工具和代码生成器结合起来，第四代技术为许多软件问题提供了可靠的解决方案。

(2) 从使用第四代技术模型的公司收集来的数据表明，在小型和中型的应用软件开发中，它使软件的开发时间大大缩短，且使小型应用软件的分析和设计的时间也缩短了。

(3) 在大型软件项目中使用第四代技术，需要同样的甚至更多的分析、设计和测试才能实际上节省时间，主要是通过编码量的减少来节省时间。

因此，第四代技术模型已经成为软件开发的一个重要方法。

4. 原型模型

原型模型如图 1-6 所示，从需求分析开始，软件开发者和用户在一起定义软件的总目标，说明需求，并规划出定义的区域，然后快速设计软件中对用户可见部分的表示。快速设计导致了原型的建造，原型由用户评估，并进一步求精待开发软件的需求。逐步调整原型使之满足用户需要，这个过程是迭代的。

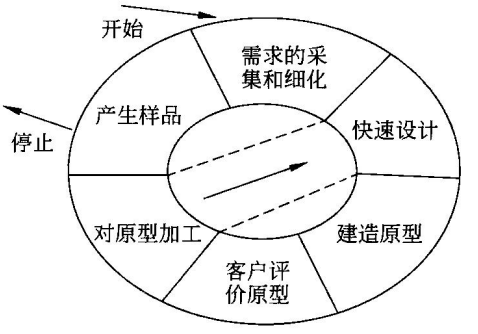

图 1-6 原型模型

(1) 原型模型的优点主要有：

- 在得到良好的需求定义上比传统生存周期法好得多，不仅可以处理模糊需求，而且开发者和用户可充分通信。
- 可作为培训环境，有利于用户培训和开发同步，开发过程也是学习过程。
- 给用户以机会更改原先设想的、不尽合理的最终系统。
- 可以低风险开发柔性较大的计算机系统。
- 使系统更易维护、对用户更友好。
- 使总的开发费用降低，时间缩短。

(2) 原型模型的缺点主要有：

- “模型效应”或“管中窥豹”。开发者在不熟悉的领域中易把次要部分当做主要框架，做出不切题的原型。
- 原型迭代不收敛于开发者预先的目标。为了消除错误，更改是必要的，但随着更改次数的增多，次要部分越来越大，“淹没”了主要部分。
- 原型过快收敛于需求集合，而忽略了一些基本点。
- 资源规划和管理较为困难，随时更新文档也带来麻烦。
- 长期在原型环境下开发，容易只注意到满意的原型，而“遗忘”用户环境和原型环境的差异。

(3) 原型模型的适用范围为：

- 特别适用于需求分析与定义规格说明。
- 设计人机界面。
- 充作同步培训工具。
- “一次性”的应用。
- 低风险引入新技术。

(4) 原型模型的不适用范围为：

- 嵌入式软件。
- 实时控制软件。

- 科技数值计算软件。

(5) 原型模型的开发步骤有以下 4 步：

① 弄清用户/设计者的基本信息需求。

目标：

- 讨论构造原型的过程。
- 写出一份简明的框架式说明性报告，反映用户/设计者的信息需求方面的基本看法和要求。
- 列出数据元素和它们之间的关系。
- 确定所需数据的可用性。
- 概括出业务原型的任务并估计其成本。
- 考虑业务原型的可能使用。

用户/设计者的基本责任是根据系统的输出来清晰地描述自己的基本需要。设计者和构造者共同负责系统范围的规定，确定数据的可用性。这个步骤的中心是设计者和构造者定义基本的信息需求。讨论的焦点是数据的提取、过程模拟。

② 开发初始原型系统。

目标：建立一个能运行的交互式应用系统来满足用户/设计者的基本信息需求。

在这一步骤中设计者没有责任，由构造者去负责建立一个初始原型，其中包括与设计者的需求及能力相适应的对话，还包括收集设计者对初始原型反映的设施。

主要工作：

- 逻辑设计所需的数据库。
- 构造数据变换或生成模块。
- 开发和安装原型数据库。
- 建立合适的菜单或语言对话，使用户输入输出接口友好。
- 装配或编写所需的应用程序模块。
- 把初始原型交付给用户/设计者，并且演示如何工作、确定是否满足设计者的基本需求、解释接口和特点、确定用户/设计者是否能很舒适地使用系统。

原则：

- 建立模型的速度是关键因素，而不是运行的效率。
- 初始原型必须满足用户/设计者的基本需求。
- 初始原型不求完善，它只响应设计者的基本已知需求。
- 设计者使用原型必须要很舒适。
- 装配和修改模块，构造者不应编写传统的程序。
- 构造者必须利用可用的技术。
- 用户与系统接口必须尽可能简单，使设计者在用初始原型工作时不至于受阻碍。

③ 用原型系统完善用户/设计者的需求。

目标：

- 让用户/设计者能获得有关系统的亲身经验，必须使之更好地理解实际的信息需求和最能满足这些需求的系统种类。