第 1 章　软件工程概述

1.1　软　　件

1.1.1　软件的概念及特点

人们通常把各种不同功能的程序，包括系统程序、应用程序、用户自己编写的程序等称为软件。然而，计算机的应用日益普及，软件日益复杂，规模日益增大，人们意识到软件并不仅仅等于程序。程序是人们为了完成特定的功能而编制的一组指令集，它由计算机的语言描述，并且能在计算机系统上执行。而软件不仅包括程序，还包括程序的处理对象——数据，以及与程序开发、维护和使用关的图文资料（文档）。例如，用户购买的 Windows 10 操作系统这个软件，它不仅包含可执行的程序，还有一些支持的数据（都放在光盘中），并且还包含纸质的用户手册等文档。Roger S. Pressman 对软件给出了这样的定义：计算机软件是由专业人员开发并长期维护的软件产品。完整的软件产品包括在各种不同容量和体系结构计算机上的可执行的程序，运行过程中产生的各种结果，以及以硬复制和电子表格等多种方式存在的软件文档。

软件具有以下几个特点：

（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。

（2）软件的生产与硬件不同，它没有明显的制造过程。要提高软件的质量，必须在软件开发方面下功夫。

（3）在软件的运行和使用期间，不会出现硬件中所出现的机械磨损、老化问题。然而，它存在退化问题，必须对其进行多次修改与维护，直至其退役。例如，早期的 DOS 操作系统，就是进行了多次修改与维护，实在难以与 Windows 操作系统匹敌而退役了。图 1-1 和图 1-2 分别展示了硬件的失效率和使用时间的关系，以及软件的失效率和使用时间的关系。

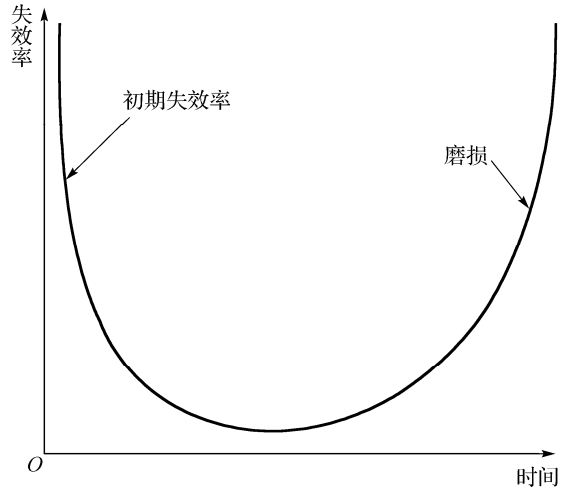

图 1-1　硬件的失效率和使用时间的关系

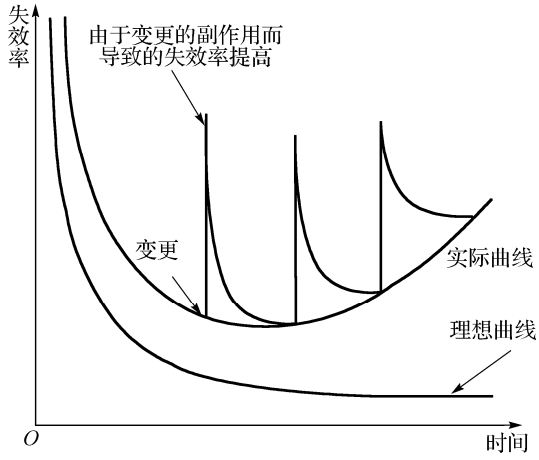

图 1-2　软件的失效率和使用时间的关系

（4）计算机的开发与运行常常受到计算机系统的制约，它对计算机系统有着不同程度的依赖性。如有专门针对 PC 的游戏，也有针对苹果计算机的游戏。为了解除这种依赖性，在软件开发中提出了软件移植的问题。

（5）软件的开发至今尚未完全摆脱人工的开发方式。

（6）软件本身是复杂的。软件的复杂性可能来自它所反映的实际问题的复杂性，也可能来自程序逻辑结构的复杂性。

（7）软件成本相当昂贵。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，它的成本是比较高的。

（8）相当多的软件工作涉及社会因素。许多软件的开发和运行涉及机构、体制及管理方式等问题，它们直接决定项目的成败。

1.1.2　软件的分类

随着计算机软件复杂性的增加，在某种程度上，人们很难对软件给出一个通用的分类，但是人们可以从不同的角度对软件进行分类。按照功能的不同，软件可以分为系统软件、支撑软件和应用软件三类。系统软件是居于计算机系统中最靠近硬件的一层，为其他程序提供最低层系统服务，它与具体的应用领域无关，如编译程序和操作系统等。支撑软件以系统软件为基础，以提高系统性能为主要目标，支撑应用软件的开发与运行，主要包括环境数据库、各种接口软件和工具组。应用软件是提供特定应用服务的软件，如字处理程序等。系统软件、支撑软件和应用软件之间既有分工又有合作，是不可以截然分开的。

基于规模的不同，软件可以划分为微型、小型、中型、大型和超大型软件。一般情况下，微型软件只需要一名开发人员，在 4 周以内完成开发，并且代码量不超过 500 行；这类软件一般仅供个人专用，没有严格的分析、设计和测试资料；例如，某个学生为完软件工程课程的一个作业而编制的程序就属于微型软件。小型软件开发周期可以持续到半年，代码量一般控制在 5000 行以内；这类软件通常没有预留与其他软件的接口，但是需要遵循一定的标准，附有正规的文档资料；例如，某个学生团队为完成软件工程课程的大作业（学期项目）而编制的程序就属于小型软件。中型软件的开发人员控制在 10 人以内，要求在 2 年以内开发 5000～50000 行代码；这种软件的开发不仅需要完整的计划、文档及审查，还需要开发人员之间、开发人员和用户之间的交流与合作；例如，某个软件公司为某个客户开发的办公自动化系统（OA）而编制的程序就属于中型软件。大型软件是由 10～100 名开发人员在 1～3 年的时间内开发的，具有 50000～100000 行（甚至上百万行）代码的软件产品；在这种规模的软件开发中，必须有统一的标准、严格的审查制度及有效的项目管理；例如，某个软件公司开发的某款多人在线的网络游戏就属于大型软件。超大型软件往往涉及上百名甚至上千名成员以上的开发团队，开发周期可以持续到 3 年以上，甚至 5 年；这种大规模的软件项目通常被划分为若干个小的子项目，由不同的团队开发；例如，微软公司开发的 Windows 10 操作系统就属于超大型软件。

根据软件服务对象的不同，软件还可以分为通用软件和定制软件。通用软件是由特定的软件开发机构开发、面向市场公开销售的独立运行的软件系统，如操作系统、文档处理系统和图片处理系统等。定制软件通常是面向特定的用户需求、由软件开发机构在合同的约束下开发的软件，如为企业定制的办公系统、交通管理系统和飞机导航系统等。

按照工作方式，计算机软件还可以划分为实时软件、分时软件、交互式软件和批处理软件。

软件的分类示意图如图 1-3 所示。

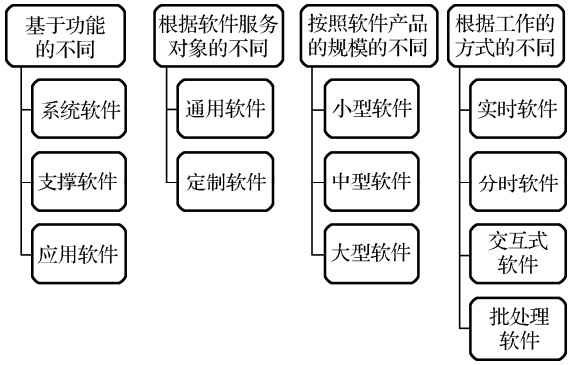

图 1-3　软件的分类示意图

1.2　软 件 危 机

1.2.1　软件危机的表现与原因

软件危机是指人们在开发软件和维护软件过程中所遇到的一系列的问题。在 20 世纪 60 年代中期，随着软件规模的扩大、复杂性的增加、功能的增强，使得高质量的软件开发变得越来越困难。在软件开发的过程中，会经常出现一些不能按时完成任务、产品质量得不到保证、工作效率低下和开发经费严重超支等现象。这些情况逐渐使人们意识到软件危机的存在及其重要性。计算机软件的开发、维护和应用过程中普遍出现的这些严重的问题的主要表现如下。

- 开发出来的软件产品不能满足用户的需求，即产品的功能或特性与需求不符。这主要是由于开发人员与用户之间不能充分有效地交流造成的，使得开发人员对用户需求的理解存在着差异。
- 相比越来越廉价的硬件，软件代价过高。
- 软件质量难以得到保证，且难以发挥硬件潜能。开发团队缺少完善的软件质量评审体系以及科学的软件测试规程，使得最终的软件产品存在着诸多缺陷。
- 难以准确估计软件开发、维护的费用以及开发周期。软件产品往往不能在预算范围之内，按照计划完成开发。很多情况下，软件产品的开发周期或经费会大大超出预算；
- 难于控制开发风险，开发速度赶不上市场变化。
- 软件产品修改维护困难，集成遗留系统更困难。
- 软件文档不完备，并且存在着文档内容与软件产品不符的情况。软件文档是计算机软件的重要组成部分，它为在软件开发人员之间以及开发人员与用户之间信息的共享提供了重要的平台。软件文档的不完整和不一致的问题会给软件的开发和维护等工作带来很多麻烦。

这些问题严重影响了软件产业的发展，制约着计算机应用。为了形象地描述软件危机，OS/360 经常被作为一个典型的案例。20 世纪 60 年代初期，IBM 公司组织了 OS/360 操作系统的开发，这是一个超大型的软件项目，它使用了 1000 人左右的程序员。在经历了数十年的开发之后，极度复杂的软件项目甚至产生了一套不包括在原始设计方案之中的工作系统。Fred

Brooks 是这个项目的管理者，他在自己的著作《人月神话》中曾经承认，自己犯了一个价值数百万美元的错误。

软件危机的出现和日益严重的趋势充分暴露了软件产业在早期的发展过程中存在的各种各样的问题。可以说，人们对软件产品认识的不足以及对软件开发的内在规律理解的偏差是软件危机出现的本质原因。具体来说，软件危机出现的原因可以概括为以下几点。

- 忽视软件开发前期的需求分析。
- 开发过程缺乏统一的、规范化的方法论的指导。软件开发是一项复杂的工程，人们需要用科学的工程化的思想来组织和指导软件开发的各个阶段。而这种工程学的视角正是很多软件开发人员所没有的，他们往往简单地认为软件开发就是程序设计。
- 文档资料不齐全或不准确。软件文档的重要性没有得到软件开发人员和用户的足够重视。软件文档是软件开发团队成员之间交流和沟通的重要平台，还是软件开发项目管理的重要工具。如果人们不能充分地重视软件文档的价值，则势必会给软件开发带来很多不便。
- 忽视与用户之间、开发组成员之间的交流。
- 忽视测试的重要性。
- 不重视维护或由于上述原因造成维护工作的困难。由于软件的抽象性和复杂性使得软件在运行之前，对开发过程的进展情况很难估计。再加上软件错误的隐蔽性和改正的复杂性，都使得软件开发和维护在客观上比较困难。
- 从事软件开发的专业人员对这个产业认识不充分，缺乏经验。软件产业相对于其他工业产业而言，是一个比较年轻、发展不成熟的产业，人们在对它的认识上缺乏深刻性。
- 没有完善的质量保证体系。完善的质量保证体系的建立需要有严格的评审制度，同时还需要有科学的软件测试技术及质量维护技术。软件的质量得不到保证，使得开发出来的软件产品往往不能满足人们的需求，同时人们还可能需要花费大量的时间、资金和精力去修复软件的缺陷，从而导致了软件质量的下降和开发预算超支等后果。

1.2.2　软件危机的启示

软件危机给我们的最大启示是，使我们更加深刻地认识到软件的特性以及软件产品开发的内在规律。

- 软件产品是复杂的人造系统，具有复杂性、不可见性和易变性，难以处理。
- 个人或小组在开发小型软件时使用到的非常有效的编程技术和过程，在开发大型、复杂系统时难以发挥同样的作用。
- 从本质上讲，软件开发的创造性成分很大、发挥的余地也很大，很接近于艺术。它介于艺术与工程之间的某一点，并逐步向工程一端漂移，但很难发展到完全的工程。
- 计算机和软件技术的快速发展，提高了用户对软件的期望，促进了软件产品的演化，为软件产品提出了新的、更多的需求，难以在可接受的开发进度内保证软件的质量。
- 几乎所有的软件项目都是新的，而且是不断变化的。项目需求在开发过程中会发生变化，而且很多原来预想不到的问题会出现，对设计和实现手段进行适当的调整是不可避免的。
- “人月神化”现象——生产力与人数并不成正比。

为了解决软件危机，人们开始尝试着用工程化的思想去指导软件开发，于是软件工程诞生了。

1.3　软 件 工 程

1.3.1　软件工程的概念

1968 年，在北大西洋公约组织举行的一次学术会议上，人们首次提出了软件工程这个概念。当时，该组织的科学委员们在开会讨论软件的可靠性与软件危机的问题时，提出了“软件工程”的概念，并将其定义为“为了经济地获得可靠的和能在实际机器上高效运行的软件，而建立和使用的健全的工程规则”。这个定义肯定了工程化的思想在软件工程中的重要性，但是并没有提到软件产品的特殊性。

随着 40 多年的发展，软件工程已经成为一门独立的学科，人们对软件工程也逐渐有了更全面、更科学的认识。

IEEE 对软件工程的定义为：（1）将系统化、严格约束的、可量化的方法应用于软件的开发、运行和维护，即将工程化应用于软件。（2）对（1）中所述方法的研究。

具体来说，软件工程是以借鉴传统工程的原则、方法，以提高质量，降低成本为目的指导计算机软件开发和维护的工程学科。它是一种层次化的技术，如图 1-4 所示。

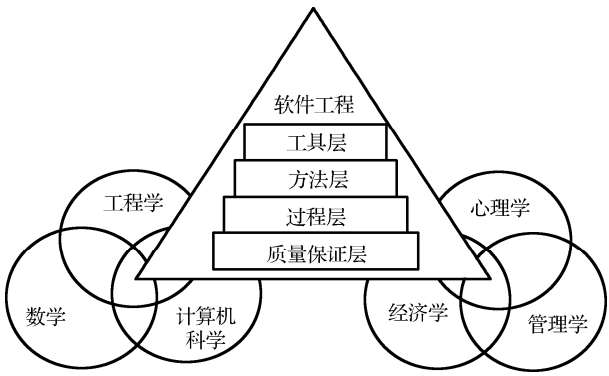

图 1-4　软件工程层次图

软件工程的根基就在于对质量的关注；软件工程的基础是过程层，它定义了一组关键过程区域的框架，使得软件能够被合理和及时的开发；软件工程的方法提供了建造软件在技术上需要“做什么”，它覆盖了一系列的任务，包括需求分析、设计、编程、测试和支持等；软件工程的工具对过程和方法提供了自动的或半自动的支持。而软件工程本身是一个交叉学科，涉及多种学科领域的相关知识，包括工程学、数学、计算机科学、经济学、管理学、心理学等。

软件工程以关注质量为目标，其中过程、方法和工具是软件工程的三要素。

1.3.2　软件工程研究的内容

软件工程研究的内容主要包括以下两个部分：

- 软件开发技术。主要研究软件开发方法、软件开发过程、软件开发工具和环境。
- 软件开发过程管理。主要研究软件工程经济学和软件管理学。

必须强调的是，随着人们对软件系统研究的逐渐深入，软件工程所研究的内容也在不断地更新和发展。

1.3.3 软件工程目标和原则

软件工程要达到的基本目标包括：

- 达到要求的软件功能。
- 取得较好的软件性能。
- 开发出高质量的软件。
- 付出较低的开发成本。
- 需要较低的维护费用。
- 能按时完成开发工作，及时交付使用。

为了达到上述目标，在软件工程设计、工程支持以及工程管理在软件开发过程中必须遵循一些基本原则。著名软件工程专家 B.Boehm 综合了有关专家和学者的意见并总结了多年来开发软件的经验，提出了软件工程的 7 条基本原则。

（1）用分阶段的生命周期计划进行严格的管理

将软件的生命周期划分为多个阶段，对各个阶段实行严格的项目管理。软件开发是一个漫长的过程，人们可以根据软件的特点或目标，把整个软件的开发周期划分为多个阶段，并为每个阶段制定分阶段的计划及验收标准，这样有益于对整个软件开发过程进行管理。在传统的软件工程中，软件开发的生命周期可以划分为可行性研究、需求分析、软件设计、软件实现、软件测试、产品验收和交付等阶段。

（2）坚持进行阶段评审

严格地贯彻与实施阶段评审制度可以帮助软件开发人员及时地发现错误并将其改正。在软件开发的过程中，错误发现得越晚，修复错误所要付出的代价就会越大。实施阶段评审，只有在本阶段的工作通过评审后，才能进入下一阶段的工作。

（3）实行严格的产品控制

在软件开发的过程中，用户需求很可能在不断地发生着变化。有些时候，即使用户需求没有改变，软件开发人员受到经验的限制以及与客户交流不充分的影响，也很难做到一次性获取到全部的正确的需求。可见，需求分析的工作应该贯穿到整个软件开发的生命周期内。在软件开发的整个过程中，需求的改变是不可避免的。当需求更新时，为了保证软件各个配置项的一致性，实施严格的版本控制是非常必要的。

（4）采用现代程序设计技术

现代的程序设计技术，比如面向对象，可以使开发出来的软件产品更易维护和修改，同时还能缩短开发的时间，并且更符合人们的思维逻辑。

（5）软件工程结果应能清楚地审查

虽然软件产品的可见性比较差，但是它的功能和质量应该能够被准确地审查和度量，这样才能有利于有效的项目管理。一般软件产品包括可以执行的源代码、一系列相应的文档和资源数据等。

（6）开发小组的人员应该少而精

开发小组成员的人数少有利于组内成员充分地交流，这是高效团队管理的重要因素。而高素质的开发小组成员是影响软件产品的质量和开发效率的重要因素。

（7）承认不断改进软件工程实践的必要性

随着计算机科学技术的发展，软件从业人员应该不断地总结经验并且主动学习新的软件技术，只有这样才能不落后于时代。

B.Boehm 指出，遵循前 6 条基本原则，能够实现软件的工程化生产；按照第 7 条原则，不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验。

1.3.4　软件工程知识体系

IEEE 在 2014 年发布的《软件工程知识体系指南》中将软件工程知识体系划分为以下 15 个知识领域。

（1）软件需求（software requirements）。软件需求涉及软件需求的获取、分析、规格说明和确认。

（2）软件设计（software design）。软件设计定义了一个系统或组件的体系结构、组件、接口和其他特征的过程以及这个过程的结果。

（3）软件构建（software construction）。软件构建是指通过编码、验证、单元测试、集成测试和调试的组合，详细地创建可工作的和有意义的软件。

（4）软件测试（software testing）。软件测试是为评价、改进产品的质量、标识产品的缺陷和问题而进行的活动。

（5）软件维护（software maintenance）。软件维护是指由于一个问题或改进的需要而修改代码和相关文档，进而修正现有的软件产品并保留其完整性的过程。

（6）软件配置管理（software configuration management）。软件配置管理是一个支持性的软件生命周期过程，它是为了系统地控制配置变更，在软件系统的整个生命周期中维持配置的完整性和可追踪性，而标识系统在不同时间点上的配置的学科。

（7）软件工程管理（software engineering management）。软件工程的管理活动建立在组织和内部基础结构管理、项目管理、度量程序的计划制订和控制三个层次上。

（8）软件工程过程（software engineering process）。软件工程过程涉及软件生命周期过程本身的定义、实现、评估、管理、变更和改进。

（9）软件工程模型和方法（software engineering models and methods）。软件工程模型特指在软件的生产与使用、退役等各个过程中的参考模型的总称，如需求开发模型、架构设计模型等都属于软件工程模型的范畴；软件开发方法，主要讨论软件开发各种方法及其工作模型。

（10）软件质量（software quality）。软件质量特征涉及多个方面，保证软件产品的质量是软件工程的重要目标。

（11）软件工程职业实践（software engineering professional practice）。软件工程职业实践涉及软件工程师应履行其实践承诺，使软件的需求分析、规格说明、设计、开发、测试和维护成为一项有益和受人尊敬的职业；还包括团队精神和沟通技巧等内容。

（12）软件工程经济学（software engineering economics）。软件工程经济学是研究为实现特

定功能需求的软件工程项目而提出的在技术方案、生产（开发）过程、产品或服务等方面所做的经济服务与论证，计算与比较的一门系统方法论学科。

（13）计算基础（computing foundations）。计算基础涉及解决问题的技巧、抽象、编程基础、编程语言的基础知识、调试工具和技术、数据结构和表示、算法和复杂度、系统的基本概念、计算机的组织结构、编译基础知识、操作系统基础知识、数据库基础知识和数据管理、网络通讯基础知识、并行和分布式计算、基本的用户人为因素、基本的开发人员人为因素和安全的软件开发和维护等方面的内容。

（14）数学基础（mathematical foundations）。数学基础涉及集合、关系和函数，基本的逻辑、证明技巧、计算的基础知识、图和树、离散概率、有限状态机、语法，数值精度、准确性和错误，数论和代数结构等方面的内容。

（15）工程基础（engineering foundations）。工程基础涉及实验方法和实验技术、统计分析、度量、工程设计，建模、模拟和建立原型，标准和影响因素分析等方面的内容。

软件工程知识体系的提出，让软件工程的内容更加清晰，也使得其作为一个学科的定义和界限更加分明。

1.3.5　软件工程的发展

随着软件项目的规模和难度逐渐增大，以个人能力为基础的软件开发所具有的弊端体检体现，随之出现了著名的“软件危机”。NATO（北约）科学委员会在这种情况之下，在 1968 和 1969 年召开了两次里程碑似的“软件工程会议”，许多顶尖级的研究员和工程师参加了这次会议，真正意义上的“软件工程”就此诞生。

20 世纪 70 年代，人们开始采用与 60 年代的“编码和组装”相反的过程，先做系统需求分析，然后再设计，最后再编码，并把 50 年代硬件工程技术最好的方面和改进的软件方向的技术加以总结，提出了“瀑布模型”。需要指出的是，瀑布模型本身在提出时，是一个支持迭代和反复的模型。然后为了更方便地对软件进行约束，瀑布模型总是被解释为一种纯顺序化的模型，另外对瀑布模型的固定过程标准的解释也加深了这种误解。

另一方面，Bohm-Jacoponi 提出了“goto 语句是有害的”论点，并提出所有程序都可以转换为三种逻辑即顺序、分支、循环来实现，奠定了结构化编程的基础。随后，很多种结构化软件开发方法被提出，极大地改善了软件质量，提高软件开发效率。数据结构和算法理论迅速发展，取得了很多重要成就；形式方法和程序证明技术也成为人们关注的发展焦点。

然而，随着形式化模型和连续化的瀑布模型所带来的问题大幅度增加，对于一个缺乏经验的团体来说，采用形式化的方法，软件在可靠性和有用性上要达到要求十分困难。瀑布模型在文档编写时消耗很大，而且速度慢，使用起来代价大。

伴随先前 20 世纪 70 年代开发的一些“最佳实践”，80 年代开始了一系列工作以处理 70 年代遗留问题，并且开始改进软件工程的生产效率和可测量性。COCOMO 模型、CMM 模型等被提出，软件体系结构相关研究和技术日益成熟，关系数据库被提出。

在软件工具方面，除了 20 世纪 70 年代已经出现的软件需求和设计工具外，其他领域一些重要的工具也得到了改进，比如测试工具和配置管理工具。工具集和集成开发支持环境先后出现，最终人们将范围扩展到了计算机辅助软件工程（CASE），软件开发的效率进一步得到提高。

在其他方面，也出现了一些潜在的提高软件生产率的方法，如专家系统、高级程序语言、面向对象、强大的工作站以及可视化编程等。Brooks 在 1986 年 IFIP 发表的著名论文《没有银弹》中对所有的这些方法发表了看法。他提出软件开发面临 4 个方面核心挑战：高等级的软件复杂度、一致性、可变性和不可视性。关于如何解决这些挑战，他严重质疑了将技术说成是软件解决方案的“银弹”的观点。Brooks 的解决这些核心挑战的候选方案包括：良好的设计者、快速原型、演化开发和通过复用降低工作量。

20 世纪 90 年代，面向对象方法的强劲势头得以持续。这些方法通过设计模式、软件体系结构和体系结构描述语言以及 UML 技术的发展得到了加强。同时，Internet 的继续扩展和 WWW 的出现增强了面向对象的方法以及市场竞争环境下软件的危险性。

软件作为竞争鉴别器重要性的增强以及缩短软件推向市场时间的需要，引发了从顺序的瀑布模型向其他模型的转变潮流，这类模型应该强调并行的工程性的需求、设计和编码，产品和过程以及软件和系统。软件复用成为软件开发中重要的内容，开源文化出露头角，可用性以及人机交互也成为软件开发中的重要指标。

20 世纪 90 年代末，出现了许多的敏捷方法，如自适应软件开发、水晶项目开发、动态系统开发、极限编程、特征驱动开发、Scrum 等。这些主要的敏捷方法的创始人在 2001 年聚集一堂并发表了敏捷开发宣言。

在 21 世纪，对快速应用开发追求的趋势仍在继续，在信息技术、组织、竞争对策以及环境等方面的变革步伐也正在加快。云计算、大数据、物联网、人工智能和机器学习、移动互联网、三维打印、可穿戴式技术、虚拟现实、增强现实、社交媒体、无人驾驶汽车和飞机等技术不断涌现。这种快速的变革步伐引发了软件开发领域越来越多的困难和挫折，更多的软件开发过程、方法和工具也相继出现，软件工程在持续的机遇与挑战中不断发展。“大规模计算”、“自治和生化计算机”、“模型驱动体系结构”、“构件化软件开发”等新领域都可能成为未来软件工程发展的主要方向。

1.4　软件过程概述

软件的诞生和生命周期是一个过程，我们总体上称这个过程为软件过程。软件过程是为了开发出软件产品，或者是为了完成软件工程项目而需要完成的有关软件工程的活动，每项活动又可以分为一系列的工程任务。任何一个软件开发组织，都可以规定自己的软件过程，所有这些过程共同构成了软件过程。为获得高质量的软件产品，软件过程必须科学、有效。由于软件项目千差万别，不可能找到一个适用于所有软件项目的任务集合，因此科学、有效的软件过程应该定义一组适合于所承担的项目特点的任务集合。通常，一个任务集合包括一组软件工程任务、里程碑和应该交付的产品。事实上，软件工程过程是一个软件开发组织针对某一类软件产品为自己规定的工作步骤，它应当是科学的、合理的，否则必将影响到软件产品的质量。

过程定义了运用方法的顺序，应该交付的文档资料，为保证软件质量和协调变化所需要采取的管理措施，以及标志软件开发各个阶段任务完成的里程碑。通常，使用生命周期模型简洁地描述软件过程。生命周期模型规定了把生命周期划分为哪些阶段及各个阶段的执行顺序，因此也称为过程模型。

1.5　软件生命周期

1.5.1　软件生命周期的概念

任何事物都有一个从产生到消亡的过程，事物从其孕育开始，经过诞生、成长、成熟、衰退，到最终灭亡，就经历了一个完整的生命周期。生命周期是世界上任何事物都具备的普遍特征，软件产品也不例外。作为一种工业化的产品，软件产品的生命周期是指从设计该产品的构想开始，到软件需求的确定、软件设计、软件实现、产品测试与验收、投入使用以及产品版本的不断更新，到最终该产品被市场淘汰的全过程。

软件产品概念的提出有利于人们更科学、更有效地组织和管理软件生产。软件生命周期这个概念从时间的角度将软件的开发和维护的复杂过程分解为若干个阶段，每个阶段都完成特定的相对独立的任务。由于每个阶段的任务相对于总任务难度会大幅度降低，在资源分配、时间把握和项目管理上都会比较容易控制。合理地划分软件生命周期的各个阶段，使各个阶段之间既相互区别又相互联系，为每个阶段赋予特定的任务，这些都是软件开发项目成功的重要因素。

1.5.2　传统软件生命周期的各个阶段

对软件生命周期的划分必须依据特定的软件开发项目所采用的软件开发模型。软件开发模型相当于软件生命周期中所有工作和任务的总体框架，它不仅反映了软件开发的组织方式，还映射了不同阶段之间的过渡和衔接关系。采用不同模型开发的软件产品，其生命周期也有所不同。但是，在传统的软件工程中，软件产品的生命周期一般可以划分为 6 个阶段，如图 1-5 所示。

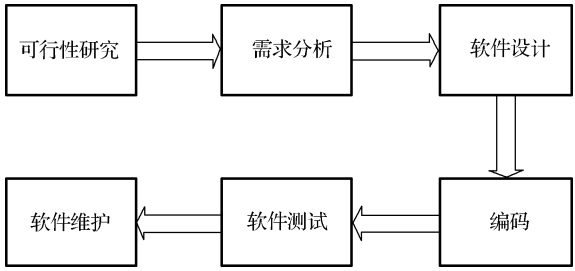

图 1-5　传统软件生命周期的各个阶段

1．可行性研究

可行性研究阶段为后续的软件开发做必要的准备工作。它首先要确定待开发的软件产品所要解决的问题。软件分析人员与用户之间需要充分地交流与合作，才能对待开发软件产品的目标达成一致。同时，在可行性研究阶段，开发人员还应该确定总体的开发策略与开发方式，并对开发所需要的资金、时间和各种资源做出合理的估计。精确的预估需要建立在开发人员对用户需求的充分了解以及自身丰富经验的基础上。此外，在可行性研究阶段，还需要对开发软件产品进行可行性分析，并制订初步的开发计划。可行性分析是为了在技术、经济、操作或社会等多个方面寻求可行的解决方案，并对各个方案进行比较，完成可行性分析报告。

2．需求分析

需求是指为了解决用户提出的问题，目标系统需要做什么。需求分析是一个很复杂的过程，需求分析是否准确和成功，直接关系到后续的软件开发的成败。在需求分析阶段，开发人员首先要通过各种途径对需求进行获取。要得到正确和详尽的需求，开发人员与用户之间的交流与沟通是非常重要的。得到需求后，开发人员需要对原始的需求进行抽象与概括，从功能、性能、界面和接口等诸多方面对需求进行详细的描述，并最终反映到软件需求规格说明书中。

3．软件设计

软件设计是指在需求分析的基础上，软件开发人员通过制订设计方案，把需求文档中描述的功能可操作化。设计可以分为概要设计和详细设计两个阶段。概要设计旨在建立系统的总体结构，从总体上对软件的结构、接口和全局数据结构等给出数据说明。详细设计关注每个模块的内部实现细节，为后续的编码工作提供最直接的依据。

4．编码

在编码阶段，开发人员根据设计阶段制订出的设计方案，编写程序代码。简单地说，编码的过程就是把详细设计文档中对每个模块实现过程的算法描述转换为能用某种程序设计语言来实现的程序。在规范的软件开发过程中，编码必须遵守一定的标准，这样有助于团队开发，同时能够提高代码的质量。

5．软件测试

软件测试是保证软件质量的关键步骤。软件测试的目的是发现软件产品中存在的软件缺陷，进而保证软件产品的质量。在软件开发的实践中，软件缺陷的产生是必然的。软件缺陷发现得越晚，修复缺陷所需的成本就越高，损失也就越大。为了尽早发现软件缺陷，必须有效地进行软件测试。按照测试点的不同，测试可以分为单元测试、集成测试、系统测试和验收测试。

6．软件维护

在软件产品被交付后，其生命周期还在继续。在使用的过程中，用户还会不断地发现产品中所隐藏的各种各样的错误。同时，随着用户需求的增长或改变，以及市场环境的变化，软件产品的功能需要不断更新，版本需要不断升级。所以，在使用软件产品的过程中，软件开发人员需要对产品进行维护，以保证软件产品的正常运行。一般来讲，软件产品的质量越高，进行维护的工作量就会越小。

1.6　软件过程模型

在现实生活中，人们处理问题时经常采用建模的方法。在软件工程中，人们通过建立抽象的软件开发模型（也称为软件过程模型或软件生命周期模型），把软件生命周期中的各个活动或步骤安排到一个框架中，将软件开发的全过程清晰且直观地表达出来。可以说，软件开发模型是软件工程思想的具体化，它反映了软件在其生命周期中各阶段之间的衔接和过渡关系以及软件开发的组织方式，是人们在软件开发实践中总结出来的软件开发方法和步骤。

软件开发模型的内在特征有以下 4 点。

（1）描述了主要的开发阶段。

（2）定义了每个阶段要完成的主要任务和活动。

（3）规范了每个阶段的输入和输出。

（4）提供了一个框架，把必要的活动映射到这个框架中。

40 多年来，软件工程领域中出现了多种不同的软件开发模型，它们具有不同的特征，适应于不同特点的软件开发项目。

常见的软件开发模型有很多种，这里主要介绍瀑布模型、快速原型模型、增量模型、螺旋模型、喷泉模型、基于组件的开发模型、统一软件开发过程模型以及敏捷模型与极限编程。

1.6.1　瀑布模型

瀑布模型是在 20 世纪 80 年代之前最受推崇的软件开发模型，它是一种线性的开发模型，具有不可回溯性。开发人员必须等前一阶段的任务完成后，才能开始进行后一阶段的工作，并且前一阶段的输出往往就是后一阶段的输入。由于它的不可回溯性，如果在软件生命周期的后期发现并要改正前期的错误，那么需要付出很高的代价。传统的瀑布模型是文档驱动的，如图 1-6 所示。

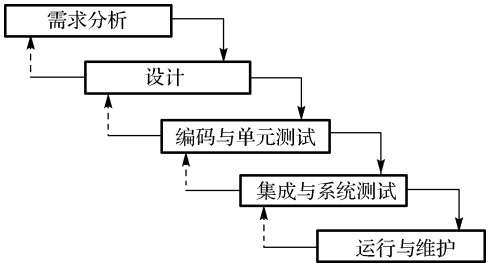

图 1-6　瀑布模型

瀑布模型的优点是过程模型简单，执行容易；缺点是无法适应变更。瀑布模型适应于具有以下特征的软件开发项目。

（1）在软件开发的过程中，需求不发生或发生很少变化，并且开发人员可以一次性获取到全部需求。否则，由于瀑布模型较差的可回溯性，在后续阶段中需求经常性的变更需要付出高昂的代价。

（2）软件开发人员具有丰富的经验，对软件应用领域很熟悉。

（3）软件项目的风险较低。瀑布模型不具有完善的风险控制机制。

1.6.2　快速原型模型

快速原型的基本思想是快速建立一个能反映用户主要需求的原型系统，让用户在计算机上试用它，通过实践来了解目标系统的概貌。通常，用户试用原型系统之后会提出许多修改意见，开发人员按照用户的意见快速地修改原型系统，然后再次请用户试用……反反复复地改进，直到原型系统满足用户的要求。

软件产品一旦交付给用户使用之后，维护便开始了。根据用户使用过程中的反馈，可能需要返回到收集需求阶段，如图 1-7 中虚线箭头所示（图中实线箭头表示开发过程，虚线箭头表示维护过程）。

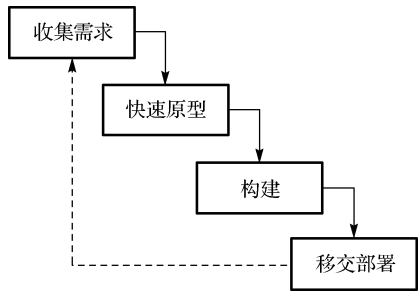

图 1-7　快速原型模型

快速原型的本质是“快速”。开发人员应该尽可能快地建造出原型系统，以加速软件开发过程，节约软件开发成本。原型的用途是获知用户的真正需求，一旦需求确定了，原型将被抛弃。因此，原型系统的内部结构并不重要，重要的是，必须迅速地构建原型然后根据用户意见迅速地修改原型。UNIX Shell 和超文本都是广泛使用的快速原型语言。快速原型模型是伴随着第四代语言（PowerBuilder、Informix-4GL 等）和强有力的可视化编程工具（Visual Basic、Delphi 等）的出现而成为一种流行的开发模式。

快速原型模型适用于具有以下特征的软件开发项目。

（1）已有产品或产品的原型（样品），只需客户化的工程项目。

（2）简单而熟悉的行业或领域。

（3）有快速原型开发工具。

（4）进行产品移植或升级。

1.6.3　增量模型

增量模型是把待开发的软件系统模块化，将每个模块作为一个增量组件，从而分批次地分析、设计、编码和测试这些增量组件。运用增量模型的软件开发过程是递增式的过程。相对于瀑布模型而言，采用增量模型进行开发，开发人员不需要一次性地把整个软件产品提交给用户，而是可以分批次地进行提交。

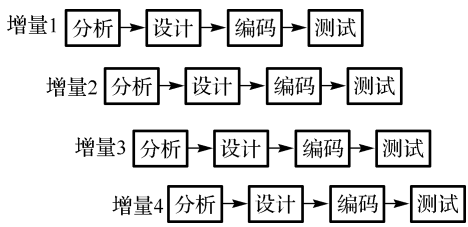

图 1-8　增量模型的示意图

一般情况下，开发人员会首先实现提供基本核心功能的增量组件，创建一个具备基本功能的子系统，然后再对其进行完善。增量模型的示意图如图 1-8 所示。

增量模型的最大特点就是将待开发的软件系统模块化和组件化。基于这个特点，增量模型具有以下优点。

- 将待开发的软件系统模块化，可以分批次地提交软件产品，使用户可以及时了解软件项目的进展。
- 以组件为单位进行开发降低了软件开发的风险。一个开发周期内的错误不会影响到整个软件系统。
- 开发顺序灵活。开发人员可以对构件的实现顺序进行优先级排序，先完成需求稳定的核心组件。当组件的优先级发生变化时，还能及时地对实现顺序进行调整。

增量模型的缺点是要求待开发的软件系统可以被模块化。如果待开发的软件系统很难被模块化，那么将会给增量开发带来很多麻烦。

增量模型适用于具有以下特征的软件开发项目。

- 软件产品可以分批次地进行交付。
- 待开发的软件系统能够被模块化。
- 软件开发人员对应用领域不熟悉，难以一次性地进行系统开发。
- 项目管理人员把握全局的水平较高。

1.6.4　螺旋模型

螺旋模型是一种用于风险较大的大型软件项目开发的过程模型。该模型将瀑布模型与快速原型模型结合起来，并且加入了这两种模型忽略了的风险分析。它把开发过程分为制订计划、风险分析、实施工程和客户评估 4 种活动。制订计划就是要确定软件系统的目标，了解各种资源限制，并选定合适的开发方案。风险分析旨在对所选方案进行评价，识别潜在的风险，并制定消除风险的机制。实施工程的活动中渗透了瀑布模型的各个阶段，开发人员对下一版本的软件产品进行开发和验证。客户评估是获取客户意见的重要活动。螺旋模型的示意图如图 1-9 所示。

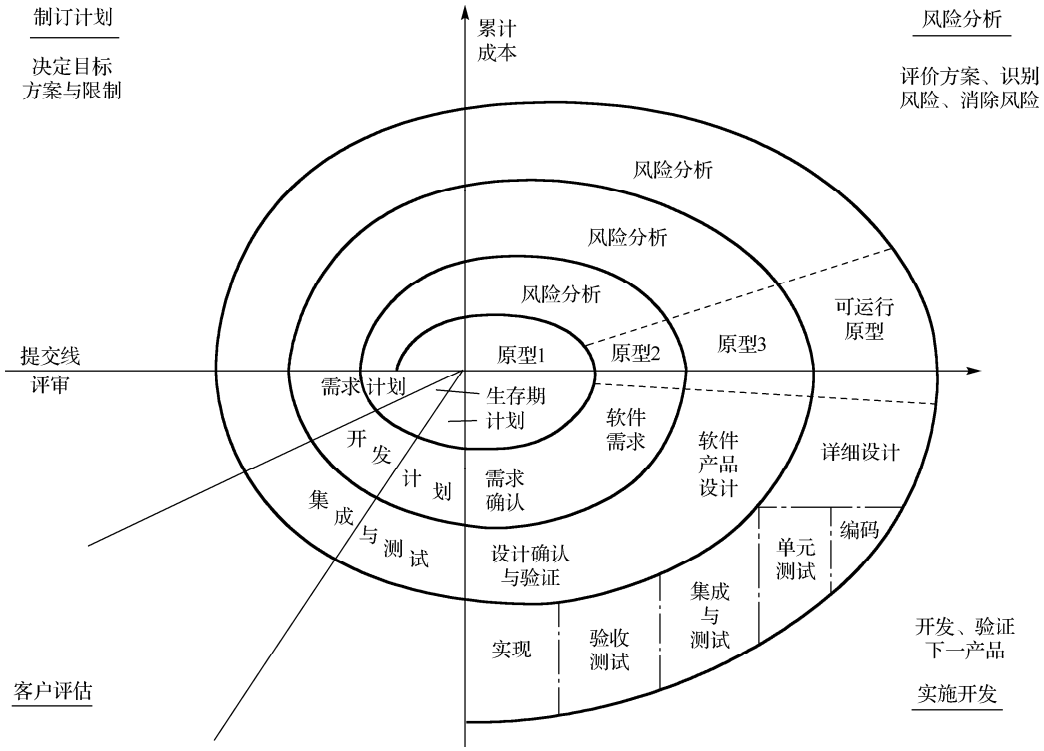

图 1-9　螺旋模型的示意图

螺旋模型适应于风险较大的大型软件项目的开发。它的优点是将风险分析扩展到各个阶段中，大幅度降低了软件开发的风险。但是，这种模型的控制和管理较为复杂，可操作性不强，对项目管理人员的要求较高。

1.6.5　喷泉模型

喷泉模型是一种过程模型，同时也支持面向对象开发。喷泉模型的示意图如图 1-10 所示。在分析阶段，定义类和对象之间的关系，建立对象-关系和对象-行为模型。在设计阶段，从实现的角度对分析阶段模型进行修改或扩展。在编码阶段，使用面向对象的编程语言和方法实现设计模型。在面向对象的方法中，分析模型和设计模型采用相同的符号标示体系，各阶段之间没有明显的界限，而且常常重复、迭代地进行。

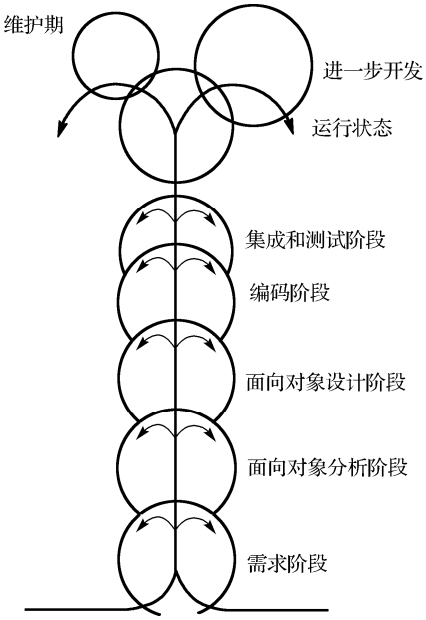

图 1-10　喷泉模型的示意图

“喷泉”一词体现了面向对象方法的迭代和无间隙性。迭代是指各阶段需要多次重复，例如，分析和设计阶段常常需要多次、重复进行，以更好地实现需求。无间隙性是指各个阶段之间没有明显的界限，并常常在时间上互相交叉，并行进行。

喷泉模型主要用于面向对象的软件项目，软件的某个部分通常被重复多次，相关对象在每次迭代中随之加入渐进的软件成分。

1.6.6　基于组件的开发模型

基于组件的开发模型使用现有的组件以及系统框架进行产品开发，由于现有组件大多已经历实际应用的反复检验，因此其可靠性相对新研发组件高出很多。

实际上，从最简单的应用程序到极度复杂的操作系统，现在的新产品开发很少完全从零开发，都或多或少地使用了现有的组件或系统开发框架，比如大型游戏的开发常常使用现有的图形引擎、声音引擎以及场景管理模块等。使用现有的组件开发新产品不仅极大地提高了产品开发效率，同时由于组件常常是经历了时间考验的，因此产品的质量也得到了提高。

基于组件开发模型的示意图如图 1-11 所示，在确定需求之后，开发人员开始从现有的组件库中筛选合适的组件，并对组件功能进行分析。组件库可能是组织内部开发的，也可能是商业授权组件，后者常常需要支付费用并且不能任意修改和传播，但也有一些开源组织（如著名的 GNU）或自由开发人员提供免费并可自由修改和传播的组件。在对组件分析之后，开发人员可能适当修改需求来适应现有组件，也可能修改组件或寻找新的组件。组件筛选完成之后，开发人员需要根据需求设计或使用现有的成熟开发框架复用这些组件，一些无法利用现有组件的地方，则需要进行单独的开发，新开发的组件在经历时间考验之后也会加入到组件库中。最后将所有组件集成在一起，进行系统测试。

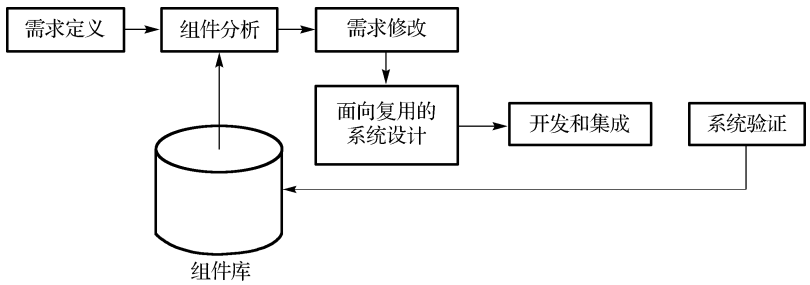

图 1-11　基于组件开发模型的示意图

基于组件的开发模型充分地体现了软件复用的思想，降低了开发成本和风险，并加快了产品开发。随着技术的发展，现在的软件系统越来越庞大，完全从零开发已近乎不可能，基于现有组件或系统开发已成为一种趋势。

1.6.7　统一软件开发过程模型

统一软件开发过程（Rational Unified Process，RUP）模型是基于 UML（统一建模语言）的一种面向对象软件开发模型。它解决了螺旋模型的可操作性问题，采用迭代和增量递进的开发策略，并以用例驱动为特点，集中了多个软件开发模型的优点。RUP 模型是迭代模型的一种。RUP 模型的示意图如图 1-12 所示。

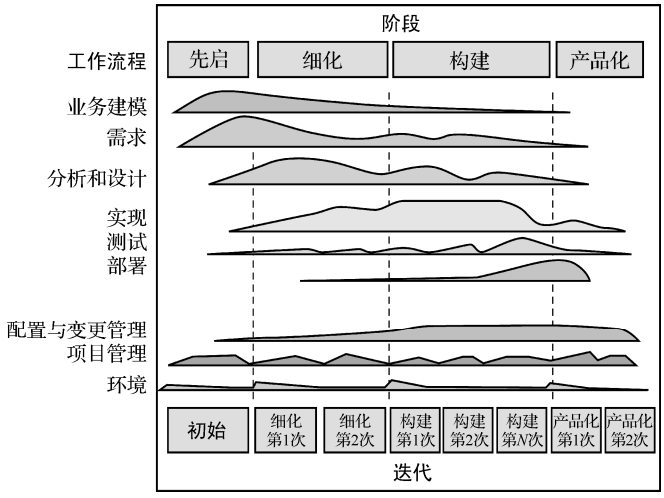

图 1-12　RUP 模型的示意图

图 1-12 中的纵轴以工作的内容为组织方式，表现了软件开发的工作流程。工作流程就是指一系列的活动，这些活动产生的结果是可见的价值。工作流程可以分为核心工作流程和核心支持工作流程。其中，核心工作流程是在整个项目中与主要关注领域相关的活动的集合。在每个迭代的软件生命周期中，核心工作流程有业务建模、需求、分析和设计、实现、测试和部署。配置与变更管理、项目管理和环境属于核心支持工作流程，它们为核心工作流程的实施提供支持。

图 1-12 中的横轴以时间为组织方式，表现了软件开发的 4 个阶段：先启、细化、构建和产品化。每个阶段中都可能包含若干次迭代。先启阶段的任务是估算项目的成本和效益，确定项目的规模、功能和架构，估计和安排项目的进度；细化阶段的主要目标是建立软件系统的架构，如建立用例模型、静态模型、动态模型和实现模型；构建阶段的任务是通过一系列的迭代过程，增量式地构建和实现用例；产品化阶段的任务是试用产品并改正试用中发现的错误，以及制作产品的最终版本，安装产品、完善用户手册并培训用户等。这 4 个阶段按照顺序依次进行，每个阶段结束时都有一个主要里程碑。实际上，可以把每个阶段看成两个主要里程碑之间的时间跨度。在每个阶段结束时都要进行阶段评估，确保该阶段目标已被实现，从而进入下一个阶段。阶段与里程碑的关系如图 1-13 所示。

统一软件开发过程模型是基于迭代思想的软件开发模型。在传统的瀑布模型中，项目的组织方法是使其按顺序一次性地完成每个工作流程。通常，在项目前期出现的问题可能推迟到后期才会被发现，这不仅增大了软件开发的成本，还严重影响了软件开发的进度。采用迭代的软件工程思想可以多次执行各个工作流程，有利于更好地理解需求、设计出合理的系统

架构，并最终交付一系列渐趋完善的成果。可以说，迭代是一次完整地经过所有工作流程的过程，从图 1-13 中可以看到，每个阶段都包含了一次或多次的迭代。

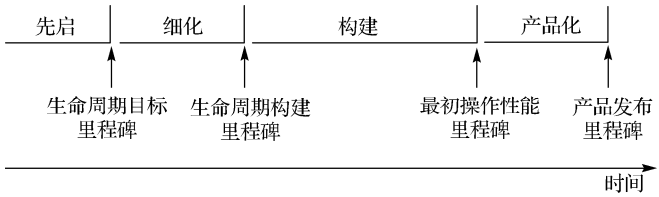

图 1-13　统一软件开发过程的项目阶段和里程碑

基于统一软件开发过程模型所构造的软件系统，是由软件构件建造而成的。这些软件构件定义了明确的接口，相互连接成整个系统。在构造软件系统时，RUP 采用架构优先的策略。软件架构概念包含了系统中最重要的静态结构和动态特征，架构体现了系统的总体设计。架构优先开发的原则是 RUP 开发过程中至关重要的主题。

统一软件开发过程模型适用的范围极为广泛，但是对开发人员的素质要求较高。

1.6.8　敏捷过程与极限编程

1．敏捷过程概述

随着计算机技术的迅猛发展和全球化进程的加快，软件需求常常发生变化，强烈的市场竞争要求更快速地开发软件，同时软件也能够以更快的速度更新。传统的方法在开发时效上时常面临挑战，因此，强调快捷、小文档、轻量级的敏捷开发方法开始流行。如今，“敏捷”已经成为一个非常时尚的名词。敏捷方法是一种轻量级的软件工程方法，相对于传统的软件工程方法，它更强调软件开发过程中各种变化的必然性，通过团队成员之间充分的交流与沟通以及合理的机制来有效地响应变化。

敏捷开发开始于“敏捷软件开发宣言”。在 2001 年 2 月，17 位软件开发方法学家在美国犹他州召开了长达两天的会议，制订并签署了“敏捷软件开发宣言”，该宣言给出了 4 个价值观。

（1）个体与交互高于过程和工具

这并不是否定过程与工具的重要性，而是更加强调人与人的沟通在软件开发中的作用。因为软件开发过程最终还是要由人来实施的，只有涉及软件开发过程的各方面人员（需求人员、设计师、程序员、测试人员、客户和项目经理等）充分地沟通和交流，才能保证最终的软件产品符合客户的需求。如果只是具有良好的开发过程和先进的过程工具，而开发人员本身技能很差，又不能很好地沟通，那么软件产品最终一样会遭到失败。

（2）可运行软件高于详尽的文档

对用户来说，更多地会通过直接运行程序而不是阅读大量的使用文档来了解软件的功能。因此，敏捷软件开发强调不断地、快速地向用户提交可运行程序，虽然不一定是完整程序，来让用户了解软件以及得到用户的认可。重要文档仍然是不可缺少的，能帮助用户更精准、全面地了解软件的功能，但软件开发的主要目标是开发出可执行的软件。

（3）与客户协作高于合同（契约）谈判

大量实践表明，在软件开发的前期，很少有客户能够精确完整地表达他们的需求，即便是那些已经确定下来的需求，也常常会在开发过程中改变。因此，靠合同谈判的方式将需求

确定下来非常困难。对于开发人员来说，客户的部分需求变更甚至会导致软件的大范围重构，而通过深入分析客户需求之后，有时还会发现通过适当调整需求就可以避免做出重大调整。而对于前者的情况，开发团队往往通过和客户谈判，撰写精确的需求合同来限制需求变更。但这会导致最终的软件产品功能与客户需求之间存在难以避免的差异，也会导致客户的满意度降低。因此，敏捷软件开发强调与客户的协作，通过密切的沟通合作而不是合同契约来确定用户的需求。

（4）对变更及时响应高于遵循计划

任何的软件开发都需要制订一个详细的开发计划，确定各任务活动的先后顺序以及大致日期。然而，随着项目的进展，需求、业务环境、技术、团队等都有可能发生变化，任务的优先顺序和时间有时面临必须调整，所以，必须保证项目计划能够很好地适应这种难以预料的变化，并能够根据变化修订计划。比如，在软件开发的后期，如果团队人员流失，那么若时间允许，适当后延计划比补充新的开发人员进入项目的风险更小。

发表“敏捷软件开发宣言”的 17 位软件开发人员组成了敏捷软件开发联盟（Agile software development alliance），简称“敏捷联盟”。他们当中有极限编程的发明者 Kent Beck、Scrum 的发明者 Jeff Sutherland 和 Crystal 的发明者 Alistair Cockburn。“敏捷联盟”为了帮助希望使用敏捷方法来进行软件开发的人们定义了 12 条原则。

（1）通过尽早和持续交付有价值的软件来让客户满意。

（2）需求变更可以发生在整个软件的开发过程中，即使在开发后期，我们也欢迎客户对于需求的变更。敏捷过程利用变更为客户创造竞争优势。

（3）经常交付可工作的软件。交付的时间间隔越短越好，最好是 2～3 周一次。

（4）在整个的软件开发周期中，业务人员和开发人员应该天天在一起工作。

（5）围绕受激励的个人构建项目，给他们提供所需的环境和支持，并且信任他们能够完成工作。

（6）在团队的内部，最有效果和效率的信息传递方法是面对面交谈。

（7）可工作的软件是进度的首要度量标准。

（8）敏捷过程提倡可持续的开发速度。责任人、开发人员和用户应该能够保持一种长期稳定的开发速度。

（9）不断地关注优秀的技能和好的设计会增强敏捷能力。

（10）尽量使工作简单化。

（11）好的架构、需求和设计来源于自组织团队。

（12）每隔一定时间，团队应该反省如何才能有效地工作，并相应地调整自己的行为。

2．极限编程

敏捷模型包括多种实践方法，比如极限编程（eXtreme Programming，XP）、自适应软件开发（Adaptive Software Development，ASD）、动态系统开发方法（Dynamic System Development Method，DSDM）、Scrum、Crystal 和特征驱动开发（Feature Driven Development，FDD）等。这里只介绍极限编程的相关内容。

极限编程是一种实践性较强的规范化的软件开发方法，它强调用户需求和团队工作。利用极限编程方法进行软件开发实践的工程师，即使在开发周期的末期，也可以很快地响应用户需求。在团队工作中，项目经理、用户以及开发人员都有责任为提高软件产品的质量而努