第 1 篇　理论与背景

本篇将简单介绍软件开发过程和配置管理的基本原则、要素、方法和工具。着重探讨配置管理和软件开发过程的支持关系，并据此提出设计集成的开发过程管理和配置管理方案的必要性。

第 1 章 典型软件开发过程及主要模型

软件开发是一个复杂而有序的过程。随着软件功能的日益强大，软件开发也逐渐从一种个人或小团队的的艺术性脑力劳动上升为一种大型团队的工程化产业活动。熟悉某种编程语言已经远不能胜任大型软件开发项目的需求，很多时候还要求开发人员具有包括软件工程、开发过程和工具等方面的综合性技能，这对软件架构、设计人员和软件项目管理人员来说尤其重要。

本章从介绍典型的软件开发过程入手，详细阐述开发过程中每个环节的内涵与外延，帮助读者加深对软件开发过程的认知与了解。然后，选取若干常见的软件开发过程模型，包括瀑布模型、螺旋模型、RUP（Rational Unified Process，统一软件过程）模型、敏捷开发模型等，阐释各种模型的特点与精髓。各个模型之间并没有绝对的高、低、优劣之分，只有适用与否之别。本章让读者大致了解各种模型，在本书的随后章节中，将通过具体翔实的案例对当前主流的过程模型和相应工具进行详细介绍、演练，以供读者实践参考。

本章分为两节，第一节描述典型的软件开发过程，包括生命周期的六个阶段；第二节介绍几种主要的软件开发过程模型。

1.1　软件生命周期及主要过程

自第一台计算机 ENIAC（Electronic Numerical Integrator and Calculator，电子数字积分计算器）于 1946 年 2 月 14 日在美国宾夕法尼亚大学研制成功起，软件作为计算机系统的主要组成部分之一，一直是计算机科学的研究重点。美国著名软件工程专家 Barry W. Boehm 对软件有如下定义：软件是程序开发、使用并维护而需要的所有文档。由此不难看出，软件不仅仅是程序，而软件开发也不仅仅是编写代码。

1983 年，Barry W. Boehm 在其经典的论文“Seven Basic Principles of Software Engineering[18]”中提出，为了保证软件开发项目的成功，至少应贯彻以下 7 个基本原则。

- 用分阶段的生命周期计划进行管理。
- 实施持续验证，以尽早发现并解决问题。
- 维护有章可循的产品控制行为记录，以避免不断变化的需求带来混乱。
- 采用现代编程实践，使软件开发过程更为透明。
- 明确项目人员的责任分工，清晰描述软件结果。
- 项目参与人员少而精。
- 承诺持续提高软件开发过程，以适应新的变化。

本书将从多个角度、不同层次来阐述这些基本原则，并阐述如何在实际的软件开发项目中通过工具和最佳实践成功实现以上原则。本节先对其中一些原则进行简单介绍，权当引子。

项目开发应有规划，这一点易于理解。一个软件项目计划至少应包括：项目的整体描述、各阶段的主要目标和成果、项目控制计划（包括组织结构、责任分工、资源管理等）、产品控制计划（包括软件产品和配置管理中的活动、产品跟踪与发布等）、验证计划，以及操作与维护计划等。这其中的很多内容都包含在项目管理的范畴中，在业界巨头 IBM 内部，都有相应的培训课程，详细讲授项目管理的成功秘诀。

软件项目中存在的缺陷是无法避免的，因此，从软件计划阶段就应开始关注缺陷的修正。在软件开发生命周期中，晚一个阶段来修正缺陷会使成本成指数增长，即所谓的 1-10-100 规则。而且在缺陷修正时，几乎 80%的精力都将耗费在 20%左右的缺陷上，即所谓的 8-2 现象[19][20]。由此不难看出，为保证软件项目按质、按量、按时且在预算内完成，尽早发现并解决缺陷是极其重要的一环。通过持续集成，让用户或测试人员尽早地体验最新的开发版本，发现问题、修订问题，进行持续的验证，是行之有效的实践经验。

随着市场、技术等方面的变化，客户的需求也时有变化。这涉及大量文档和代码的变更。如果没有一套方法来实施产品控制，使大量的产品版本和文档有章可循、有案可查，那将不可避免地使开发人员在需求、变更和产品之间产生混淆，影响开发进度，甚至软件质量。

随着时代的变迁和开发团队的成长，软件开发也会出现一些新的特点，由此可能会使某些软件开发实践变得过时。因此，持续地总结软件开发的经验教训、尝试新的开发思想、采用新的最佳实践，对软件企业来说，也是一种立于不败之地的方式。

Boehm 的文章中所提出的这些基本原则，即便放在今天来看，也依然具有很大的借鉴价值，对现代软件开发过程仍然具有指导作用。事实上，国内相当多的中小软件开发企业至今也远未能在其软件开发项目中遵循以上的基本原则，而这也是目前国内软件行业存在大量作坊式开发，缺乏优秀软件产品问世的重要原因。对于软件行业的从业人员来说，如果连以上几条基本原则都无法正确理解并实践，那么就无法称之为合格的软件人才，而这也是当前大量软件从业人员无法突破自己，成为 IT 精英的主要瓶颈。

一般认为，典型的软件开发过程，也即软件生命周期，由 6 个阶段组成，即软件计划阶段、需求分析阶段、软件设计阶段、软件编码阶段、软件测试阶段和软件维护阶段。本节将结合笔者的软件开发实践，逐一介绍这 6 个阶段并简单介绍相关的工具。

1.1.1 软件计划阶段

软件计划阶段是整个软件开发过程的起始，古语云“失之毫厘，谬以千里”。软件计划中的任何偏差、瑕疵都有可能给后续的软件开发过程带来麻烦、返工甚至失败的厄运。软件计划也是一个不易掌握与操作的阶段，因为此时对软件的最终形式与内容尚未明确，存在许多不确定因素，需要进行大量调研和论证工作，从而为将来的软件开发定下基调。

严格来说，在软件计划阶段，首先要弄清楚该软件要解决什么问题，对该问题需进行明确的定义，以及为解决该问题，对软件在功能、性能等方面有怎样的需求。然后，调研用户需求、历史背景、实际情况和经济前景，从多个方面研究并论证该软件项目的可行性，并编写可行性研究报告，探讨解决问题的各种可能的方案。可行性研究主要关注 3 个方面[21]：经济可行性、技术可行性和法律可行性。最后要对开发投入、回报，以及开发进度等进行预估。

不难看出，软件开发的计划涵盖以下 4 个方面。

- 软件开发的范围：包括软件的功能、性能、接口、运行环境与维护条件等。
- 开发所需的资源：包括所需投入的硬件、软件和人力资源。
- 开发进度表：制订每个阶段的完成时间、主要目标，以及验收标准。

- 开发成本：根据所需的资源和开发进度等因素，给出大致的开发成本。

在软件计划阶段，要全面了解所有的问题是相当困难的，尤其是在日益信息化和扁平化的今天，用户提出的问题经常随瞬息万变的市场而发生变化。但是，明确并把握要解决的核心问题，仍然是软件开发项目得以成功的关键。大部分软件开发项目虽然都有软件计划这一过程，但常常因为沟通交流不够、经验不足而导致效果不佳，最后直接影响软件开发的进度甚至软件项目的成败。

在实际软件开发项目中，并非所有的用户都能清楚地描述其所面临的问题，这就给软件架构与系统分析人员带来了挑战。在这种情形下，通常需要双方加强沟通交流，及时交换各自的理解和看法，并通过简单、有效的方式记录双方已达成的共识，以备后期检验和修订。

软件计划阶段的时长不等，从数天到数月，有时甚至超过一年。但即便如此，在更多的情形下，软件项目都会在客户需求并不完全明确或确定的时候，进入下一阶段。因此，软件架构和系统分析人员应做好未来可能多次变更和修正的思想准备。

1.1.2　需求分析阶段

软件需求分析阶段的首要任务是回答这个问题——为了解决计划阶段确定的问题，软件系统必须提供什么功能。这一阶段要将软件计划阶段得到的初步需求描述进行精细化处理，生成具体的软件需求规格说明书以作为后续软件开发和双方交流的基础。因此，这是软件生命周期的决定性一环。

一般而言，用户清楚自己所遇到的问题，并有相应的需求，但通常很难完整、准确地描述出这些需求，也无法利用计算机领域的术语来规范定义这些需求。而软件架构或系统分析人员对用户所在领域的知识有限，也很难对用户的痛处和需求感同身受。因此在需求分析阶段，软件开发双方常常有“鸡同鸭讲”的痛苦经历，也对参与需求分析的软件人员的素质提出了更高的要求。

需求分析阶段就是要求双方密切配合，充分沟通，准确把握用户的核心问题，并对用户的所有需求进行分类整理、去粗取精，最后以双方均能理解的方式完整、详细、准确地记录下来。在现实的项目中，不确定因素不可忽视，加之外界条件不断变化，往往很难得到如此精细的需求描述，即便形成这样的文档，也很难保证一成不变。因此，软件开发双方在整个开发过程中，都应保持足够且通畅的交流，随时消除误解，修正偏差。

完成对用户需求的理解之后，软件架构或分析人员需要采用软件工程开发语言建立软件的逻辑模型，编写需求规格说明书并获得用户的认可。在这一阶段，常用的方法包括结构化分析方法、数据流图、数据字典和简要的算法描述等。

在需求分析阶段确定的软件逻辑模型是后续软件设计和软件编程阶段的基础，因此必

须完整、准确地涵盖用户的需求，经用户确认后才能进入下一个阶段。在实际的软件开发项目中，有些系统分析人员在需求尚未明确时，就急于投入大量精力进行系统的具体分析设计工作。一般而言，这不是一种良好的习惯和工程实践，建议读者尽量避免。但在有些情况下，当单纯的语言和文字交流不足以让双方取得对需求的共识时，可以通过快速搭建一个概念原型系统的方法来规范所描述的关键问题，并明确用户的真正需求。实践证明，这是一种有效的快速沟通的方式，这也是近来流行的敏捷开发过程中的一个重要策略。但是，建议读者在实践时，还是要在需求分析上多下工夫，尽早把握核心问题及主要分歧点，控制前期时间和精力的投入。

目前，市场上有若干用于需求分析、收集和管理的工具，其中以 IBM 的 Rational RequisitePro 最为优秀，而 IBM 在近期推出的基于 Jazz 平台的 Rational Requirement Composer 则是新一代的需求定义、配置与变更管理的平台。本书将在最后一篇中介绍 Rational Requirement Composer，以及可与之组合使用的其他 Jazz 平台产品。

1.1.3 软件设计阶段

软件设计阶段以软件需求规格说明书为基础，设计软件系统的体系结构，完成软件的具体设计方案。软件设计可以分为概要设计和详细设计两个阶段。实际上，软件设计的主要目标是将软件系统分解成若干便于编程的子系统或模块，再对各子系统或模块进行详细设计，并定义各个子系统或模块之间的接口。每个模块可以是数据、代码，或两者的结合，用于实现某个具体的功能。一般来说，软件设计的优劣取决于设计人员的能力和经验。

概要设计即结构设计，其主要目的是给出软件系统的整体模块架构。详细设计则要对各个模块进行具体的描述，包括模块的程序流程、关键算法、数据结构和数据存储等设计，并制订初步的测试方案。详细设计要给出软件模块的内部描述，即算法设计，其表现形式有多种，最为常用的是流程图和伪代码。设计阶段结束时，要提供软件设计说明书，包含模块结构图和详细的模块功能说明，作为后续阶段的参照依据。结构化的软件设计方法是常用的设计方法。

软件需求分析阶段和设计阶段密切相关，前者是后者的基础，进行需求分析时就有可能涉及软件的模块设计。另一方面，在进行软件设计时，若发现需求分析存在问题，也应及时反馈并进行重新分析和修正。

如何进行模块化设计，怎样的模块化设计是合理的？这些问题涉及一些设计准则。例如，减少模块之间的耦合、降低模块接口的复杂性、模块功能完整单一等。IBM 的 L.Constantine、E. Yourdon 等人提出的结构化设计方法曾是应用最广的一种设计方法，适用于任何软件系统的结构化设计。该方法通常与数据流分析技术配合使用，通过数据流分析得到用数据流图和数据字典描述的需求规格说明书。结构化设计方法则以数据流图为基础

得到软件的模块结构。这些设计思想在 IBM 的 Rational 软件设计建模产品中均有相应的体现。随着面向对象技术和语言的普及，采用面向对象的方法来进行模块化设计具有天生的优良特性，模块设计准则和面向对象思想也不谋而合，成为当前主要的设计方法之一。

需要指出的是，图形用户接口设计是现代软件设计的重点之一，良好的软件界面设计可以提高用户体验、实现软件价值并充分发挥软件本身的功能。一般专业的软件开发企业均有专门从事图形设计领域的人员参与界面开发。

目前有多种进行软件设计的工具，其中以 IBM 的 Rational Software Modeler（RSM）功能最为强大和广受欢迎，其实 RSM 就是读者非常熟悉的 Rational Rose 的下一代产品。RSM 提供了许多新的特性，几乎涵盖了对软件系统进行建模的各个方面。

1.1.4　软件编码阶段

软件编码是指把软件设计阶段获得的模块转换成计算机可以接受的程序。一个合格的程序员应充分了解所使用的开发语言和编程工具，以及该语言的良好编程风格，才能保证软件产品的开发质量。

软件编程所用到的程序语言多种多样，当前较为常用的包括 Java、C/C++、Ruby、Groovy，以及其他一些面向对象语言。在某些特定平台或系统上，还存有大量其他的编程语言。例如，在小型机 IBM i 系统上还存在大量的 RPG、CL、COBOL 等语言开发的应用。此外，例如 PERL、JavaScript、VBScript 等脚本语言也大有用武之地，在很多应用领域大行其道。伴随而来是开发工具的选择问题，目前以 Eclipse 为代表的集成开发环境成为与面向对象语言，它结合紧密、功能强大、简单易用且扩展性强，大大提高了软件开发的速度和质量。

软件编程阶段除了编程语言和开发环境的选择之外，还要考虑团队协作和版本控制工具的采用。多个编程人员同时开发一个产品时如何保证每个开发人员能够各种独立地进行编码，同时又能准确地把每个人的代码汇合在一起，这就需要配置管理或版本控制工具的支持。这种工具能够通过强制的检出、检入和分支合并机制实现开发人员之间的分工、合作，以维护复杂的代码版本关系。业界公认的最强大的配置管理工具是 IBM Rational ClearCase。它能提供便利的并行开发的编程环境，方便开发人员处理不同的文件版本，共享版本信息，无缝地与变更控制系统集成，也能与常用的 IDE 工具紧密集成，方便编程人员使用。本书后面章节将会重点介绍 ClearCase 在项目中的应用。此外，也会简单介绍 IBM Rational 最新的基于 Jazz 平台的团队开发协作系统 RTC（Rational Team Concert）。

1.1.5　软件测试阶段

软件测试阶段在整个软件生命周期中占据非常重要的地位，是保证软件质量的关键，

也是对需求分析、设计和编码的最后审核与修正。从软件系统的可靠性考虑，需要对软件进行尽可能完备的测试。因此，测试阶段投入的时间和人力占到整个软件开发项目的 40%也不足为怪。但对一般的软件开发项目而言，测试的目的还是以较小的代价发现尽可能多的错误。为达到这个目的，涉及软件测试的方法、测试工具的选择，以及测试用例的构造等技术。

软件测试有以下一些基本原则[21]。

- 设计测试用例时，要给出测试的预期结果。
- 开发和测试小组分离。
- 要设计非法输入的测试用例。
- 在修改程序之后，要进行回归测试，以避免引入新的错误。
- 在进行深入的测试时，要集中测试容易出错的程序段。

由此不难看出，测试是一项非常复杂、具有创造性的智力型工作。

常用的测试方法有黑盒法和白盒法。当已经知道产品所具备的功能时，可以用黑盒法测试软件的每个功能是否达到了设计的要求；如果已经知道了产品的内部逻辑，则可以使用白盒法来测试软件的内部逻辑是否符合设计要求。

黑盒法着眼于程序的外部表现特性与功能，而不考虑程序的内部逻辑结构和特性。因此，测试人员通常基于程序所提供的外部接口进行测试，确认输入/输出是否正常、功能是否如期执行。显然，受测试数据的限制，黑盒法很难进行完全的测试。

白盒法的测试对象是程序本身，通过测试程序内部的逻辑结构来发现软件中存在的各种类型的错误。在测试过程中，需要在程序的不同点设置检查点，比对实际测试结果是否和设计的状态一致。在复杂的应用软件中，白盒法要遍历所有的逻辑路径，往往是不现实的。因此，测试用例设计的关键是以最少量的用例来覆盖尽可能多的内部程序逻辑结构。

根据测试重点的不同，可分为多种类型的测试，常见的有单元测试、模块测试、集成测试、功能测试、系统测试等。每种测试的操作步骤都大体包括制订测试计划、设计测试方案、编写测试用例、执行测试用例、汇报测试结果等事项。随着软件规模的日趋庞大、功能日益复杂，自动化测试正逐渐成为软件测试中最为重要的研究方向之一，相应的自动化测试框架和工具也层出不穷。例如，IBM 的自动化测试框架 STAF（Software Testing Automation Framework）及自动化测试工具 Rational Functional Tester 等。

1.1.6 软件维护阶段

软件维护是在完成软件分析、设计、编码和测试工作，并交付用户使用之后，对软件

产品所进行的后续开发活动。软件维护的范围非常广泛，既有改正性维护、适应性维护，也有完善性维护和预防性维护。所谓改正性维护是指在软件运行过程中发现、诊断并修正在软件开发阶段而引入的错误；适应性维护是指根据软件运行环境的改变，对软件进行适当修改，以适应新的环境要求。为了满足用户在软件使用过程中所提出来的新功能要求或对现有功能的改进要求，就需要进行完善性维护工作。为了进一步提高软件的易维护性和可靠性而做的工作称为预防性维护。

众所周知，在整个软件生命周期中，软件维护阶段是最为耗时的，也是投入最大的，大多数软件开发企业将 40%～60%的经费用于维护，甚至更多[21]。在软件维护阶段开发人员需要解决开发阶段所遗留的各种问题，同时还要解决软件维护本身特有的问题，如多个发行版本之间代码的共享或迁移。良好的软件分析、设计和开发工作可以显著地降低维护费用的投入。从另一方面看，良好的软件维护工作，不仅可以最终达成软件开发的目的，保证软件的正常工作，满足用户的设计需求，而且还可能收集用户的新需求，扩展功能，提高性能，开始新一轮的软件开发过程。

在实际项目中，上述软件生命周期的各个阶段并非单一顺序地执行。在其间的任何阶段，根据项目的需要或由于外界环境的变化或技术本身的更新，都有可能反复并回到前面的阶段，甚至首尾连成环状，周而复始，形成复杂的开发流程。这也是软件开发过程中会有不同的过程模型的原因。

1.2　软件开发过程模型

软件生命周期主要由以上 6 个阶段组成，它们都属于软件开发活动。这些开发活动及其之间的关系所构成的框架就是软件开发过程模型。在实际项目中，选用何种模型是仁者见仁、智者见智的。需要根据项目本身的特点和开发目标，以及开发团队的实际情况进行选择。甚至在有些情况下，可能多种开发模型结合起来使用。

本节将介绍几种常见的开发过程模型、特点和适用范围，如瀑布模型、螺旋模型、RUP 模型，以及现在业界颇为流行的敏捷开发模型。

1.2.1　瀑布模型

1970 年，Winston W. Royce 博士在其经典论文“Managing the Development of Large Software Systems（大型软件系统开发的管理）[22]”中，总结了他多年来在与太空飞船计划相关的大型软件开发任务中的经验，第一次提出了瀑布模型（Waterfall Model）这一概念，如图 1-1 所示。

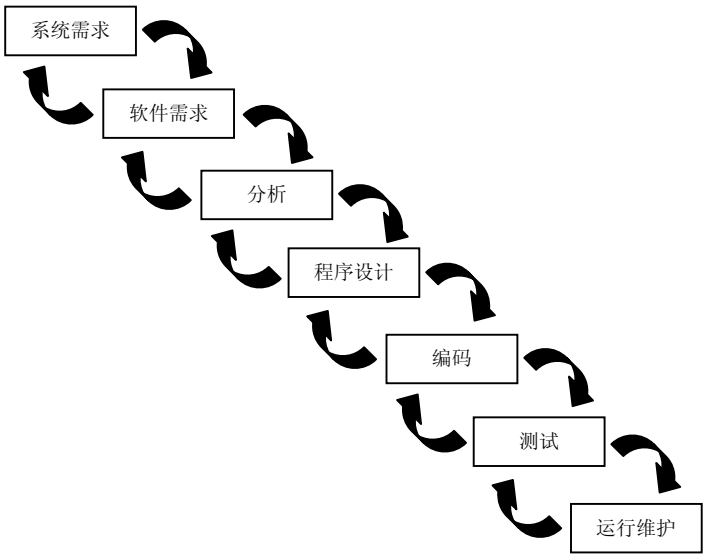

图 1-1　瀑布模型

瀑布模型是一个顺序化的软件开发过程。在这个模型中，开发过程基本按照前述的软件开发过程各个阶段自上而下、相互衔接地进行。每个阶段的活动都以文档作为驱动，需要大量的文档来记录活动状态与结果，因此瀑布模型是典型的文档驱动的开发模式。在开发过程的各个顺序阶段，从系统需求分析开始直到产品操作与维护，每个阶段的活动都可能产生对前溯阶段活动的循环反馈。即在每一阶段的结束，瀑布模型会对该阶段实施的工作进行评审，若得到确认，则继续下一项活动；否则返回前一阶段，甚至更前一阶段的活动进行修改。因此，一旦在某个阶段发现了问题，就应该返回问题被引入的所在前溯阶段并进行适当的修改，然后，开发过程再从此阶段继续流动到下一个阶段。

瀑布模型是最早出现的软件开发模型，在 20 世纪 70 年代被广泛采用，在软件工程领域占有重要的地位。它刻画了软件开发的基本框架和流程，每个阶段的活动接收上一阶段的工作结果作为输入，来完成该阶段的工作内容，并生成相应的输出，作为下一阶段的输入。

瀑布模型是美国国防部、美国国家航空航天局等机构的大型软件开发项目的实践经验的总结，促进了软件开发方法和开发工具的研究与应用，大大提高了大型软件开发项目的成功率和质量。瀑布模型对那些需求非常明确固定、要求完善文档支持并强调定期、定点检测的软件开发项目而言，有着天然的适用性。但瀑布模型的线性开发流程，以及繁重的文档化工作极大地限制了它在软件开发项目中的普及。随着世界变得更为国际化和扁平化，客户需求的变化更为频繁。虽然瀑布模型可以通过进入增量开发、并行开发、迭代等概念进行改良，但仍难以契合现代软件开发对快速响应客户需求、尽早让用户评价开发结果并给予反馈，以及强调软件本身而非文档的趋势。因此，连美国国防部的软件开发项目从 1994 年起也逐渐脱离了瀑布模型。尽管如此，瀑布模型的简单性、易实施、易管理的特性仍使之在不少软件开发项目中得以继续应用。

1.2.2　螺旋模型

由于瀑布模型具有的不足，从 20 世纪 80 年代起，有关提高软件过程模型的讨论就从未停止过。1988 年，Barry W. Boehm 发表了经典论文“A Spiral Model of Software Development and Enhancement（一种用于软件开发与增强的螺旋模型）[23]”。Boehm 在文中正式总结了螺旋模型的定义和特点，并结合作者在 TRW 国防系统部门的大型军用软件项目中使用螺旋模型的经验，给出了采用螺旋模型的最佳实践。

螺旋模型与其他模型相比，最大的区别在于前者是风险驱动（Risk-driven）的，而非传统的文档或代码驱动，同时它还吸收了其他模型的优点以克服面临的问题，因此适合大型复杂软件系统的开发。螺旋模型如图 1-2 所示[23]。

在图 1-2 所示的螺旋模型中，沿半径方向表示为完成至今为止的步骤所消耗的费用，沿角度方向表示螺旋中每个周期的完成进度。在图 1-2 中，螺线沿角度方向进行若干次迭代，4 个象限分别代表以下内容。

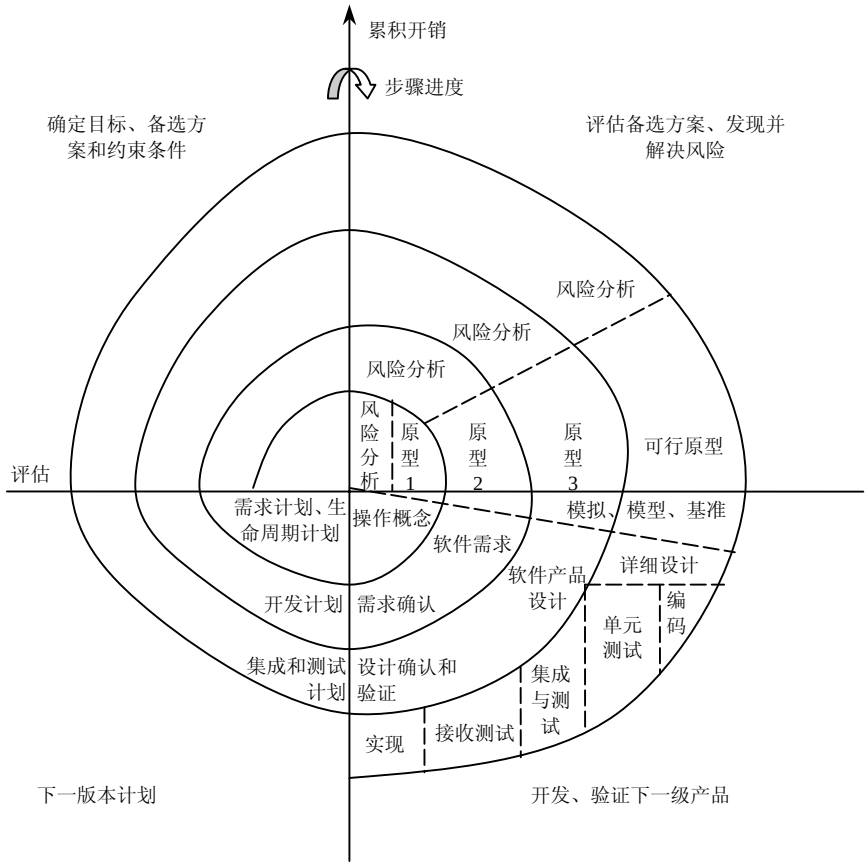

图 1-2　软件开发过程中的螺旋模型

- 第 II 象限代表确定软件过程的目标、备选方案和约束条件。
- 第 I 象限代表评估备选方案，定位并解决可能的风险。
- 第 IV 象限代表软件开发和验证。
- 第 III 象限代表客户评估，并制订下一个版本的开发计划。

一个典型的螺旋周期通常涉及产品的一部分，并都从确定这部分工作的目标开始，包括性能、功能等方面的要求。同时，提出多种备选的实现方案及相应的约束条件，例如费用、进度、接口等。之后，用参照目标和约束条件来评估备选方案，这通常将发现项目中存在的不确定性风险，因而需要提出有效的解决方法。当所有的风险都得到有效控制和解决时，下面的步骤就类似瀑布模型进行软件开发和验证工作了。

螺旋模型以风险驱动，强调备选方案和约束条件的作用，避免由于某些瓶颈问题而导致过多开发投入甚至项目失败，因而有利于软件重用和提高软件质量。但是要发现并解决存在的风险，并非易事，既需要经验丰富的开发人员，还需要平衡风险评价所投入的成本和收益，因而来自于大型软件产品开发实践的螺旋模型，也多用于大型软件开发项目。

1.2.3 RUP 模型

1．概述

20 世纪 70—90 年代，软件建模技术与软件开发过程成为 IT 业界的热门研究方向和新兴技术。1995 年 10 月，Grady Booch、Ivar Jacobson 和 James Rumbaugh 这 3 位方法学大师发布了统一方法（Unified Method），即后来被称为统一建模语言（Unified Modeling Language，UML）的 0.8 版本，并在 1997 年 9 月成为 OMG（Object Management Organization，对象管理组织）的正式标准，由 OMG 来全面负责 UML 的发展。

现代软件开发除了需要建模技术和语言之外，还需要一个先进的能够指导软件开发人员进行开发活动的开发过程方法学。1998 年，最早由 Ivar Jacobson 提出的 Rational Object Process（Rational 对象过程，ROP）被正式命名为 Rational Unified Process（Rational 统一过程，RUP），并且将 UML 作为其建模语言。由此 RUP 成为 IT 业界最为成熟和成功的软件开发过程。2002 年 12 月 6 日，IBM 收购了 Rational 软件公司，从此赋予了 RUP 新的生命力，通过加入 IBM 多年软件开发的最佳实践，形成了强大的 IRUP（IBM Rational Unified Process）架构。先进的理论和 IBM 的最佳实践相结合使得现代软件开发能够通过切实可行的指导来及早地发现并规避风险，通过统一建模、用例驱动、迭代开发、需求管理、变更控制来提高软件产品的质量。

2．RUP 的 4 个阶段

Rational 统一过程是一种迭代式的、以架构为核心的、以用例为驱动的软件开发方法，

用户可基于 RUP 进行定制裁减，以适合自身软件项目的需要。Rational 统一过程包含 4 个迭代开发阶段，如图 1-3 所示。

图 1-3　RUP 的 4 个阶段

在初始阶段（Inception），主要任务是了解需要构造什么，包括规划远景、概要需求、商业案例等，这一阶段一般不会涉及具体的需求细节；在细化阶段（Elaboration），需要知道如何来构造，包括基准架构、大部分需求细节，但一般不关心设计细节；构建阶段（Construction）是主要的产品开发阶段，要完成软件产品的开发、各种测试等；在最后一个交付阶段（Transition），将对开发完成的产品进行验收，并交付客户。每个阶段都结束于一个主要的里程碑（Milestone），如果该里程碑已完成且评估合格，即可进入下一阶段。

Rational 统一过程的各个不同阶段关注工作流中的不同活动，如图 1-4 所示。例如，在初始阶段更多地关注软件产品的商业建模和需求确定，而在细化阶段主要从事需求的分析和设计工作；代码实现、配置与变更管理等更多地体现在构建阶段，部署则是交付阶段的主要工作内容之一。

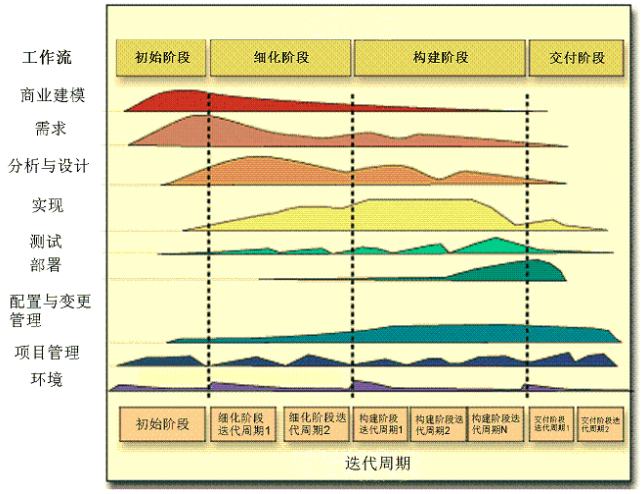

图 1-4　Rational 统一开发过程架构图

Rational 统一过程是一种迭代开发的模式。因此在理论上，Rational 统一过程的每个阶段都可以有若干个迭代周期，而每个迭代周期的结束都会产生某种形式的交付品，如设计文档、原型系统或者可执行的部分产品功能等，如图 1-5 所示。

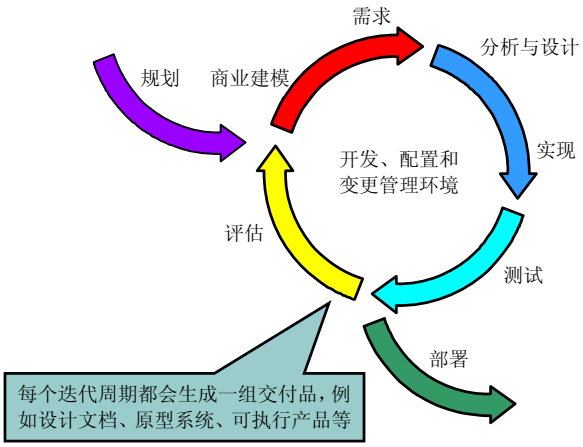

图 1-5　典型的迭代开发周期

3．最佳实践

Rational 统一过程有如下已经证明的最佳实践。

- 迭代开发：在软件开发过程中，用户需求经常改变，通过多个迭代周期，分解需求，使得高风险、高优先级的需求能够尽早实现，从而极大地减少软件开发项目的风险。每个迭代过程都会生成一组可交付客户的产品，使客户反馈能够及时地反映到产品中。
- 需求管理：Rational 统一过程借助用例来获取、组织和文档化用户的功能需求，并以此来驱动软件设计、功能实现和产品测试，使交付的系统能尽量满足用户的需求。
- 基于组件的体系结构：Rational 统一开发过程支持基于组件的软件开发方法。该方法关注于早期的开发和健壮可执行的体系结构的基线，以设计灵活的、便于修改和理解的弹性软件架构，以促进有效的软件重用，提高开发效率。
- 可视化建模：在软件产品开发中，Rational 统一开发过程借助工业级的建模标准语言统一建模语言（Unified Modeling Language, UML）来进行可视化建模软件体系架构，描述各组件的结构与行为。可视化抽象建模有助于加强开发团队的沟通交流，深入了解软件的不同层面，考查各组件的配合情况，确保实现与设计的一致性及代码的兼容性。
- 质量验证：软件产品的质量涉及多个层面，例如功能性、可靠性、易用性、系统性能、可扩展性等。Rational 统一开发过程统一规划、设计、实现、执行并评估这些验证活动，并通过迭代周期的可交付产品，引入客户反馈来持续进行质量验证和改善工作。
- 变更控制：Rational 统一开发过程具有良好的变更管理能力，描述了如何控制、跟踪和管理变更以完成迭代开发，拥抱客户的需求变更请求。

Rational 统一开发过程提供了一套可定制的软件产品，包括方法学指导、过程定义、文档模，以及示例工程等。

在总结这些最佳实践的同时，IBM 还提供了相当数量的完备产品，供用户选择以便在实际软件开发项目中采用这些最佳实践，包括开发流程定制与管理工具 Rational Method Composor、可视化建模工具 Rational Rose、配置与变更管理工具 Rational ClearCase 和 ClearQuest 等。读者可通过 IBM 网站及 IBM Rational 产品的介绍获得更多的相关信息。

1.2.4　敏捷开发

1．概述

从 20 世纪 90 年代中期起，针对传统的重量级软件开发方法过于笨重、管理过多而导致开发效率不高等问题，敏捷软件开发（Agile Software Development）作为一种轻量级的替代方法，逐渐成为一种主流的声音。2001 年 2 月 11～13 日，在美国犹他州的滑雪胜地 Snowbird，17 位各种新的轻量级软件开发方法论的代表人物在数日讨论后共同签署了“敏捷软件开发宣言（ManifestocforcAgilecSoftwarecDevelopment）”[24]。其涵盖了多种新的软件开发方法论，如极限编程（Extreme Programming）、Scrum、自适应软件开发（Adaptive Software Development）等。之后，其中一些人又组建了一个非营利性的组织敏捷联盟（Agile Alliance），来推广敏捷开发方法[25]。

2．重要思想及基本原则

在“敏捷软件开发宣言”中，明确指出了敏捷开发的一些重要思想，如下。

- 开发人员及其相互之间的交流重于开发流程和工具。
- 可工作的软件产品重于完善的文档。
- 客户协作重于合同谈判。
- 响应变更重于照章办事。

同时，宣言还列出了所遵循的 12 条基本原则，如下。

- 开发人员最重要的责任，是尽早并持续地交付有价值的软件产品以满足客户的需求。
- 即便在开发后期，也欢迎需求的变化。
- 经常交付可运行的软件产品，时间从几周到数月不等，且越短越好。
- 在项目全程，商业人员和开发人员都应每天共同工作。
- 围绕被激励的个人来构建项目，为开发人员准备所需的环境和支持，并给予信任。

- 在开发团队内最有效和直接的方式是面对面的交流。
- 将可运行的软件产品视为开发进度的主要衡量标准。
- 敏捷过程倡导张弛有度的开发，项目出资人、开发人员和用户应该维持一致步调。
- 持续关注技术上的优势和良好的设计，以加强敏捷度。
- 简单为要。
- 最佳体系架构、需求和设计来自于自我组织的团队。
- 开发团队定期反省如何更为有效，并对其行为做出相应的调整。

敏捷方法论推荐一种鼓励经常性检查和修订的项目管理过程，一种鼓励团队协作、自我组织和责任感的领导哲学，一套能快速交付高质量软件产品的工程最佳实践，以及一种能使软件开发与客户需求及公司目标相统一的商业模式。

3．开发方法

敏捷方法论包括一些非常著名的敏捷软件开发方法。

1）极限编程[26]

极限编程的一个基本思想就是认为没有任何一个流程是适合所有项目的，因此需要裁减实践经验以适合每个不同的项目。作为一种软件工程的方法论及一种敏捷软件开发方法，极限编程描述了项目关系人的一套日常实践，以体现并鼓励如下核心价值。

- 交流：在传统的软件开发方法论中，开发人员借助文档来交流系统需求；而极限编程更倾向简单的设计、通用的比喻、用户和程序员之间的协作、频繁的口头交流和反馈来达到以上目的。
- 简洁：极限编程鼓励从最简单的解决方案开始开发，之后逐渐加入额外的功能，即设计与编程关注满足现今的需求，而非将来。
- 反馈：极限编程从 3 个渠道来收集反馈，即来自系统、客户和开发团队；开发人员应对反馈有正确的认识和及时的反应。
- 勇气：极限编程鼓励开发人员勇敢做出一些决定，如永远只为现在的需求而设计和实现；又如敢于抛弃无用的代码，不论为这些代码花费了多少精力；再如，勇气有时也意味着坚持，程序员或许花费一整天试图解决某个问题而毫无所获，但只要坚持不懈，在第二天也许就能迎刃而解。
- 尊敬：开发成员相互之间应互相尊重，不因自身原因而影响团队的开发进度；开发人员也应对其工作保持足够的敬意，尽可能地给出最优设计与实现。

2）Crystal 方法论[27]

Alistair Cockburn 是 Crystal 方法论的倡导者，这是一套强调以人为本和自适应的超轻量级软件开发方法论，开发结果刚好满足需求即可。Crystal 认为自由是人类的本性，需要在开发纪律和工作效率上达成一种平衡，虽然开发效率低于极限编程，但更易于为开发人员所接受。Crystal 明确体现了以下特点。

- 经常性地发布可用代码供用户体验。
- 通过自我反省来提高开发效率。
- 更倾向于“渗透性的交流”。

3）Scrum[28]

Scrum 是一个迭代增加的软件开发过程，通常和敏捷软件开发共同使用。Scrum 不仅可用于软件开发的管理，也可用于软件维护团队或作为一种通用的项目管理方法。Scrum 最为人熟知的是每天的 Scrum 会议，时长 15～30 分钟，会上每个团队成员都要回答以下 3 个问题。

- 昨天我做了什么？
- 今天我计划做什么？
- 有没有什么困难使我无法完成既定的目标？

4）自适应软件开发[29]

作为一种软件开发流程，自适应软件开发起源于快速应用开发（Rapid Application Development），并由 Jim Highsmith 和 Sam Bayer 所倡导，体现了持续修正工作流程是一种常态的原则。

自适应软件开发通过一系列重复的推测、协作和学习周期来取代传统的瀑布式开发周期，这种动态的周期可为项目现状提供持续的学习和修订。自适应软件开发生命周期的特点就在于其以目标为核心、采用迭代、周期时长固定、风险驱动，以及拥抱改变。

5）动态系统开发方法论（Dynamic Systems Development Methodology, DSDM）[30]

动态系统开发方法论最早基于快速应用开发方法论，是一种迭代的累积式方法，强调用户的持续参与。其目标是在预算范围内及时地交付软件系统，同时在开发过程中为需求变更而相应做出调整。

作为快速应用开发的扩展，动态系统开发方法论侧重于开发进度和预算都很紧张的信息系统项目，其关注导致信息系统项目失败的最常见的原因，包括预算超标、进度超期、缺少用户参与及管理层的支持等。动态系统开发方法论包括以下内容。

- 3 个阶段：项目前阶段、项目生命周期阶段和项目后阶段。
- 项目生命周期阶段分成 5 个子阶段：可行性研究、商业研究、功能模型迭代、设计与构建迭代，以及实现。

敏捷方法论还涉及很多其他的开发方法，例如特征驱动开发（Feature-Driven Development, FDD）、上下文驱动测试（Context-Driven Testing, CDT）、Lean、实效编程（Pragmatic programming）等。受篇幅所限，本书不再进一步介绍，感兴趣的读者可阅读相关资料获取更多细节。

在敏捷开发中，软件项目被分配在若干开发阶段完成，项目起始时间内可包含多个固定时长的开发周期（Iteration 或 Sprint）。每个开发周期都有明确的开发任务，周期结束后，应向客户发布最新的可运行的软件产品，以供测试并收集反馈意见。

虽然敏捷开发相对传统开发模型有许多优势，但也存在一些缺点，这就意味着并非所有的团队和项目都适合采用敏捷开发模式。敏捷开发对团队中个人的能力有较高的要求，对项目经理的管理经验也是个挑战。因此，在一个成熟且个人能力较强的团队，采用敏捷开发相对而言就会有更便利的条件。另一方面，敏捷开发强调顺畅交流和及时沟通，如果开发和测试团队分布在多个不同的物理地点，尤其是跨越较大时区时，将对团队提出更高的要求，敏捷开发也会在这种情况下面临更大的挑战。

随着软件开发项目的不断复杂化和软件开发技术的不断改进，今后必定还有新的软件开发过程模型出现。软件开发项目的实践证明，没有一种万能的开发过程或模型可以解决软件开发中遇到的所有问题，因而需要根据项目和开发团队的实际情况，选择适合自身的开发模型和方法，并需要不断的学习和改进，才能完成既定的项目目标。

本章着重介绍了典型的软件开发过程，详细阐述了常见的软件开发过程模型，包括瀑布模型、螺旋模型、RUP（Rational Unified Process，统一软件过程）模型和敏捷开发模型，并分析了各个模型及其开发方法的特点与适用场合，可以看出各种不同模型是对软件开发阶段的各种活动的不同组合。本书后续章节将会从这些开发模型中提取出决定软件开发过程和软件项目成败的更本质的要素，并结合项目开发实例，为读者展现在实际项目如何通过工具来管理这些要素，自动化软件开发过程，以支持软件开发项目顺利进行。

第 2 篇　设计与实现

第 1 篇概述了软件配置管理和软件开发过程的基本原理和主要过程、要素，从人员、任务、方法和产品等方面阐明了软件配置管理对软件开发过程的重要支持作用。本篇将探讨如何构造一个完善的配置管理系统，以实现对软件开发过程各个要素的有力支持。

本篇以 IBM Rational 的变更和配置管理工具 ClearCase 和 ClearQuest 为基础，利用这两个工具所提供的扩展定制功能来实现前篇所述的功能需求，为广大 ClearCase 和 ClearQuest 用户提供一个系统定制的参考方案。由于 ClearCase 和 ClearQuest 本身的配置管理功能比较完善，因此本书的设计重点侧重在功能的关联性和过程的衔接性等方面，主要目标是加强过程支持，实现软件开发过程的可控性和可跟踪性。尽管系统的实现细节与底层工具紧密相关，本系统的设计思路和部分算法同样可供其他配置管理系统定制参考。