# 第 1 章　软件工程概述

当软件开发公司接到项目后，需要从无到有，以团队的形式，在规定的时间及有限的预算内，开发出满足用户需求的高质量的软件产品，使软件工程理论贯穿始终。


## 1.1　项目引入

国际国合办公项目，简称国合项目，是由计算机与软件学院软件工程系及外国语学院联合发起的，旨在提高国际合作交流的办公效率，同时方便学生与教师间的及时交流。

国合项目属于校内项目，预期成本 1 万元，项目周期 8 个月，主要由软件工程系项目负责人与外国语学院留学项目负责人对接。项目的开发目标是完成国际教育学院对国际合作项目的顺利推进，增强各相关部门之间的沟通与合作，方便学生及时了解和参加国际合作项目，提高国际合作业务的效率，实现一个规范化、流程化的国际合作办公平台。

## 1.2　项目分析

国合项目的顺序推进需要在项目开发的整个过程实现全程监控，才能实现在有限成本、有限时间内完成具有一定质量软件产品的目标。国合项目的开发周期为 8 个月，周期较长，需要合理安排时间，以免延误项目进度。整个项目面向的用户有系部教师、素质教师、学生、国合教师 4 个角色，每个角色具有不同的业务功能，考虑划分子系统。每个子系统从无到有进行开发，需要在清楚理解每个用户需求的基础上，有针对性地开发产品。在这个较长的过程中很难保证软件的质量，现阶段迫切需要一套有效的方法，实现从项目开发和项目管理两个角度进行软件项目的全程监控和指导。

## 1.3　需要解决的问题

- 如何开发和管理项目——采用软件工程方法论作为指导。
- 如何保证软件质量——软件工程的基本原理。

● 如何确认软件开发过程——软件开发过程模型。

要想开发出一个高质量的软件产品，选择合适的方法论尤为重要，软件工程是研究和应用如何以系统性的、规范化的、可定量的过程化方法去开发和维护软件，以及如何把经过时间考验而证明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。

## 1.4 软件工程的历史

在计算机系统发展早期，软件开发基本上沿用“软件作坊”式的个体化方法，这种方法在软件开发和维护过程中遇到了一系列严重问题：程序质量低下、错误频出、进度延误、费用剧增等问题导致了“软件危机”。1968年，北大西洋公约组织的计算机科学家在联邦德国召开国际会议讨论软件危机问题，正式提出并使用了“软件工程”这个名词，从此诞生了一门新兴的工程学科。

从20世纪40年代中期世界上第一台计算机出现以后，“程序”的概念就产生了。随后几十年中，计算机软件经历了3个发展阶段：程序设计阶段（约为20世纪50年代至60年代）；程序系统阶段（约为20世纪60年代至70年代）；软件工程阶段（约为20世纪70年代以后），如表1-1所示。

**表1-1　　计算机软件发展的3个阶段及其特点**

| 阶段 / 描述内容 | 程序设计 | 程序系统 | 软件工程 |
|---|---|---|---|
| 软件所指内容 | 程序 | 程序及说明书 | 程序、文档及数据 |
| 主要程序设计语言 | 汇编及机器语言 | 高级语言 | 软件语言* |
| 软件工作范围 | 程序编写 | 设计和测试 | 整个软件生存周期 |
| 需求者 | 程序设计者本人 | 少数用户 | 市场用户 |
| 开发软件的组织 | 个人 | 开发小组 | 开发小组及大、中型软件开发机构 |
| 软件规模 | 小型 | 中、小型 | 大、中、小型 |
| 决定质量的因素 | 个人程序设计技术 | 小组技术水平 | 管理水平 |
| 开发技术和手段 | 子程序、程序库 | 结构化程序设计 | 数据库、开发工具、工程化开发方法、标准和规范、网络和分布式开发、面向对象技术、软件过程与过程改进 |
| 维护责任者 | 程序设计者 | 开发小组 | 专职维护人员 |
| 硬件特征 | 价格高、存储容量小、工作可靠性差 | 降价，速度、存储容量及工作可靠性有明显提高 | 向超高速、大容量、微型化及网络化方向发展 |
| 软件特征 | 完全不受重视 | 软件技术的发展不能满足需求，出现软件危机 | 开发技术有进步，但未获突破性进展，价格高，未完全摆脱软件危机 |

* 软件语言包括需求定义语言、软件功能语言、软件设计语言、程序设计语言等。

软件发展最根本的变化体现在以下方面：

1.人们改变了对软件的看法。早在 20 世纪五六十年代，程序设计曾经被看作是一种自由发挥创造才能的技术领域。当时人们认为，只要能在计算机上得出正确的结果，程序的写法可以不受任何约束。随着计算机的使用日趋广泛，人们不断提出更高的要求（例如，要求程序易懂、易用、易于修改和扩充），于是程序便从按个人意图创造的“艺术品”转变为能被广大用户接受的工程化产品。

2.需求是软件发展的动力。早期为了满足自己的需要，程序开发者不拘风格地自由创作这种自给自足的生产方式是其初级阶段的表现。进入软件工程阶段后，软件开发的成果具有社会属性，它要在市场中流通以满足广大用户的需要。

3.软件工作的考虑范围从只顾程序的编写扩展到涉及整个软件生命周期。

4.随着计算机硬件技术的进步，要求软件能与之相适应。这个时期出现了“软件作坊”，它基本上仍然沿用早期的个体化软件开发方法，缺乏统一的管理和协调，导致许多开发项目由于软件质量问题造成巨大损失。同时随着产品的增加，软件开发力量不得不全部投入维护，没有能力继续开发新的应用系统，这就造成了计算机应用进一步发展的停滞，即 20 世纪 60 年代的“软件危机”现象。软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题，主要有以下一些表现形式。

（1）软件代价高。随着软件产业的发展，软件成本日益增长，而计算机硬件随着技术的进步、生产规模的扩大，价格却不断下降，造成了软件代价在计算机系统中所占的比例越来越大。20 世纪 50 年代，软件成本在整个计算机系统中所占的比例不大，为 10%～20%；到 20 世纪 60 年代中期已经增长到 50%左右；20 世纪 70 年代以后，软件代价高的问题不仅没有解决，反而进一步加深了。图 1-1 大体表示了一个计算机系统中，硬件和软件所占费用的比例。

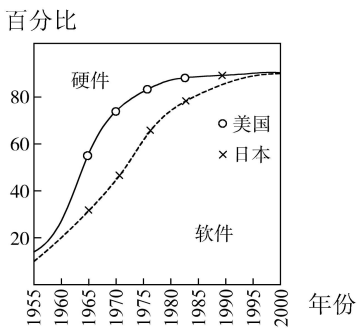



**图 1-1　软件与硬件费用之比**

（2）开发进度难以控制。软件是一种逻辑的系统元素。为了完成一个复杂的软件，常要建立庞大的逻辑体系。同样的算法可以由差别甚大的不同程序形式来实现，在研究大型系统时遇到的困难也是越来越多，而这些往往只存在于人的头脑之中。因此，软件的开发过程是极难加以控制的。

● 工作量估计困难。为软件开发制订进度很困难，通常对一个任务根据其复杂性、工作量及进度要求来安排人力，但这种工作量估算方式仅对各部分工作独立、互不干扰的工作适用，而软件系统整体各部分之间存在的任务合作与交流活动就会增加工作量的估算。由于软件系统结构复杂，各部分之间的附加联系极大，在拖延的软件项目上增加人力通常只会使其更难按期完成。这对于一般的工业产品来说是难以想象的。

● 质量差。软件的产品质量与其他商品的质量问题有着很大的不同。使用“软件作坊”开发软件的方法沿袭了早期形成的个体化方式，软件（程序）开发过程没有交互性，软件的规划、设计、测试和维护都只能由某一个人全部负责，只有程序清单而没有任何正式的软件规

划文档。这就使软件修改和维护十分困难，有时甚至变得不可能。此外，软件规模和数量的急剧增加，用户需求的不断变化，使软件的质量控制成为一个很难解决的问题，这是由软件所处的特殊地位造成的。

● 修改、维护困难。当软件系统变得庞大、问题变得复杂时，常常还会发生“纠正一个错误却带来更多新错误”的问题。此外，人们习惯于认为软件易于修改、容易扩充，因此在系统投入运行后为适应新环境，经常提出要求进行维护。这样产生的维护工作量将难以估算。1999 年美国的 Standish Group 对当年美国的软件项目的统计数字表明（如图 1-2 所示），只有 26％的软件项目是真正成功的，其余的项目全部是失败的或是有问题的，28％的项目是干脆失败的。这些存在问题的或是失败的项目带来的直接损失是 870 亿美金，占了美国当年全部 IT 投资（2550 亿美金）的近 40％；而由于这些项目所带来的间接损失是无法估量的，在全部这些项目中，平均超期 189％，平均超预算 222％，平均 27 个月滞后于最终用户的需求，更有 80％的资源被开销在对应用的维护上。

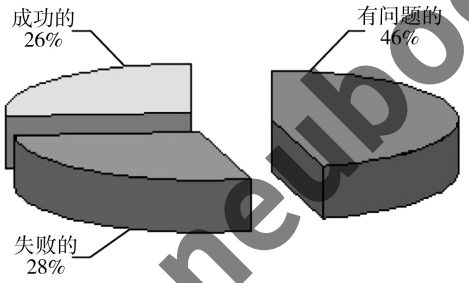



**图 1-2　Standish Group 小组报告**

总之，在软件整个生命周期中，错误发现得越晚，纠正错误所要付出的代价也就越大。其原因是：测试日趋复杂，修改的文件和文本需分发的范围更广，多次反复以前的测试，问题和修改信息传递的范围更大，涉及的人员更多。有资料表明，在数据处理方面，30％～80％的预算往往用于软件维护工作，这就直接造成了软件成本的提高。

产生软件危机的根本原因是软件面临的问题空间的复杂性。

软件的应用领域很广，面临的问题很复杂，所以涉及的处理技术也十分广泛，包括信息技术、网络技术、人机界面技术、人机会话环境技术等。另外，面临的问题空间往往还牵涉到管理体制、组织机构、内外部环境、用户水平、经济学、心理学等许多非技术问题。问题空间的复杂性决定了软件系统的复杂性。

产生软件危机的另一个重要原因是计算机硬件体系结构的发展速度滞后于软件应用的拓展速度。时至今日，硬件的体系结构基本未变。从五大组成部件来看，出现了图形扫描仪、光笔、绘图机等许多新式输入输出设备，多 CPU 的计算机在实时系统中得到应用，内外存的容量和存取速度有很大提高，但是，这些部件的变化都只是硬件功能的完善、性能的提高，属于改良性质的变化。而计算机的硬件体系结构仍属于冯·诺伊曼计算机，它的基本特征是：顺序地执行程序指令，按地址访问线性的存储空间，数据和指令在机内采用统一的表示形式，只能完成四则运算和一部分逻辑运算。冯氏计算机的初衷是为数值计算服务的，然而随着计算机应用领域的扩大，所面临的问题 90％以上是非数值计算。为了满足用户的需求，或在逻辑上构建许多的软件层次，每一软件层次都可以看作是一种语言的翻译器或解释

器，用这种方法来填补用户和裸机之间的鸿沟。简单地说，就是把解题过程分解成一系列能由冯氏计算机处理的四则运算和逻辑运算，这就使软件非常庞大，开发工作十分困难，软件的可靠性和可维护性很差。因此，可以说，正是因为把以科学计算为基础的冯氏计算机应用在非数值计算的数据处理中（会计信息系统属于此类处理），所以也把危机转嫁在软件上。

软件危机的产生，除了上述两个主要原因之外，还与人们在软件开发和维护中采用错误的方法有关。

软件系统的复杂性虽然给开发和维护带来了客观困难，但是人们在开发和使用计算机系统的长期实践中，也积累和总结了许多经验，如果坚持不懈地使用经过实践证明是正确的方法，许多困难是完全可以克服的。但是，目前相当多的开发人员对软件开发和维护还有不少糊涂的观念，在实践中或多或少地采用错误的技术和方法，表现为忽视软件的维护性等等。这些关于软件开发和维护的错误认识和做法是产生软件危机的第三个重要原因。

此外，造成软件危机还有如下一些原因。

（1）用户需求不明确，体现在 4 个方面：软件开发出来之前，用户自己不清楚其具体需求；用户对软件需求的描述不精确（有遗漏或者二义性）甚至有错误；软件开发过程中用户不停地提出修改要求（例如修改软件功能、界面、支撑环境等）；软件开发人员对用户的理解与用户本来愿望有差异。

（2）缺乏正确的理论指导，特别是缺乏有力的方法学和工具方面的支持。

（3）软件规模越来越大。

（4）软件复杂度越来越高。

（5）软件灵活性要求高。

影响软件生产率与质量的因素十分复杂，包括个人能力、团队联系、产品复杂度、合适的符号表达方式、可利用的时间地点以及其他因素（诸如技术水平、变更控制、采用的方法、所需要的可靠性、对问题的理解、需求稳定程度、设施及资源、相应的培训、管理水平、恰当的目标、期望的高低等）。

## 1.5　软件工程的基本概念

Fritz Bauer 曾经为软件工程下了如下定义："软件工程是为了经济地获得能够在实际机器上有效运行的可靠软件而建立和使用的一系列完善的工程化原则。"1993 年 IEEE 给出的定义为："软件工程是将系统化的、规范的、可度量的方法应用于软件的开发、运行、维护过程，即将工程化应用于软件中的方法的研究。"目前人们给出的一般定义是：软件工程是一门旨在生产无故障的、及时交付的、在预算之内的和满足用户需求的软件的学科。实质上，软件工程就是采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理方法和最先进的软件开发技术结合起来，应用到软件开发和维护过程中，来解决软件危机问题。

软件工程包括3个要素：方法、工具和过程，如图1-3所示。

软件工程方法为软件开发提供了“如何做”的技术。它包括多方面的任务，如项目计划与估算、软件系统需求分析、数据结构、系统总体结构的设计、算法过程的设计、编码、测试以及维护等。

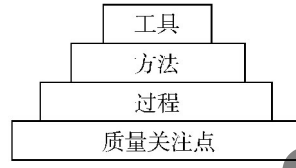



图1-3 软件工程层次

软件工具为软件工程方法提供了自动的或半自动的软件支撑环境。目前，已经推出了许多软件工具，这些软件工具集成起来，建立起称之为计算机辅助软件工程(CASE)的软件开发支撑系统。CASE将各种软件工具、开发机器和一个存放开发过程信息的工程数据库组合起来形成一个软件工程环境。

软件工程的过程则是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。过程定义了方法使用的顺序、要求交付的文档资料、为保证质量和协调变化所需要的管理，及软件开发各个阶段完成的里程碑。

## 1.6 软件工程的基本原理

自从1968年提出“软件工程”这一术语以来，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或信条。美国著名的软件工程专家Boehm综合这些专家的意见，并总结了TRW公司多年的开发软件的经验，于1983年提出了软件工程的七条基本原理。Boehm认为，这七条原理是确保软件产品质量和开发效率的原理的最小集合。它们是相互独立的，是缺一不可的最小集合；同时，它们又是相当完备的。人们当然不能用数学方法严格证明它们是一个完备的集合，但是可以证明，在此之前已经提出的100多条软件工程准则都可以由这七条原理的任意组合蕴含或派生。

**1. 用分阶段的生命周期计划严格管理**

这一条是吸取前人的教训而提出来的。统计表明，50%以上的失败项目是由于计划不周而造成的。在软件开发与维护的漫长生命周期中，需要完成许多性质各异的工作。这条原理意味着，应该把软件生命周期分成若干阶段，并相应制订出切实可行的计划，然后严格按照计划对软件的开发和维护进行管理。Boehm认为，在整个软件生命周期中应指定并严格执行6类计划：项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划、运行维护计划。

**2. 坚持进行阶段评审**

统计结果显示：大部分错误是在编码之前造成的，大约占63%；错误发现得越晚，改正它要付出的代价就越大，要差2～3个数量级。因此，软件的质量保证工作不能等到编码结束之后再进行，应坚持进行严格的阶段评审，以便尽早发现错误。

**3. 实行严格的产品控制**

开发人员最痛恨的事情之一就是改动需求。但是实践告诉我们，需求的改动往往是不可避免的。这就要求我们要采用科学的产品控制技术来顺应这种要求，也就是要采用变动控制，又叫基准配置管理。当需求变动时，其他各个阶段的文档或代码随之相应变动，以保

证软件的一致性。

**4. 采纳现代程序设计技术**

从六七十年代的结构化软件开发技术，到最近的面向对象技术，从第一、第二代语言，到第四代语言，人们已经充分认识到：采用先进的技术既可以提高软件开发的效率，又可以减少软件维护的成本。

**5. 结果应能清楚地审查**

软件是一种看不见、摸不着的逻辑产品。软件开发小组的工作进展情况可见性差，难于评价和管理。为更好地进行管理，应根据软件开发的总目标及完成期限，尽量明确地规定开发小组的责任和产品标准，从而使所得到的标准能清楚地审查。

**6. 开发小组的人员应少而精**

开发人员的素质和数量是影响软件质量和开发效率的重要因素，应该少而精。

这一条基于两点原因：高素质开发人员的效率比低素质开发人员的效率要高几倍到几十倍，开发工作中犯的错误也要少得多；当开发小组为 N 人时，可能的通讯信道为 N(N－1)/2，可见随着人数 N 的增大，通讯开销将急剧增大。

**7. 承认不断改进软件工程实践的必要性**

遵从上述六条基本原理，就能够较好地实现软件的工程化生产。但是，它们只是对现有的经验的总结和归纳，并不能保证赶上技术不断前进发展的步伐。因此，Boehm 提出应把承认不断改进软件工程实践的必要性作为软件工程的第七条原理。根据这条原理，不仅要积极采纳新的软件开发技术，还要注意不断总结经验，收集进度和消耗等数据，进行出错类型和问题报告统计。这些数据既可以用来评估新的软件技术的效果，也可以用来指明必须着重注意的问题和应该优先进行研究的工具和技术。

## 1.7　软件生命周期

软件的生命周期是一个孕育、诞生、成长、成熟和衰亡的生存过程，也就是所谓的软件定义、软件开发和运行维护三个时期，而每个时期又有所要完成的不同的基本任务。

我们用“网上选课系统”作为案例来分析各个时期的不同任务。

软件定义时期的主要任务是解决“做什么”的问题，通俗地讲就是做此项目的可行性报告及明确主要功能等。对于网上选课系统，在软件定义阶段要确定以下几个功能模块：管理员管理课程、教师、学生的增删改查和对教师、学生的权限授予等功能；教师对自己信息的修改和对自己课程的上传、修改、删除、查询等功能；学生对课程的选择、退选及查询等功能。针对此项目，从技术、经济、法律、成本、可获得的效益、开发的进度做出一系列的估算，制订出具体的实施计划。

软件开发时期的主要任务是解决“如何做”的问题，也就是如何完成此项目的过程，要解决每个构建所要完成的工作以及完成此工作的顺序。选择编写源程序的开发工具，把软件设计转换成计算机可以接受的程序代码。对于网上选课系统，在软件开发阶段，我们确定先

要进行管理员的模块编写，再进行教师模块的编写，进而进行学生模块的编写，另外也要确定运用某种软件开发工具，如Java、C语言等进行模块的开发等。在此阶段还要把前期的各个模块组装起来进行测试，保证按需求分析的要求完成软件功能的测试并对此进行确认，交与开发方运行测试。

运行维护时期的主要任务是使软件持久地满足用户的需要。对于网上选课系统，在运行维护阶段，主要针对客户在使用过程中系统出现的问题进行修改，保证系统的正常运行。同时也可以对用户提出的功能上的增强或完善性要求进行响应，以增强用户对系统的满意度。

完整的软件的生命周期是一个耗时长的工程。在软件工程生命周期的三个时期中，我们又可以基于各阶段任务相对独立，以及同一阶段任务性质相同的原则，将生命周期划分成如下几个阶段。

**1. 问题定义**

问题定义阶段必须回答的关键问题是：“要解决的问题是什么？”如果不知道是什么就试图解决这个问题，显然是盲目的，最终得到的结果很可能是毫无意义的。所以在此阶段，系统分析员需要与客户进行沟通，以明确问题的性质、工程目标及规模。

问题定义阶段是软件生命周期中最简短的阶段，一般只需要一天或更少的时间。

**2. 可行性分析**

此阶段是软件开发方与需求方共同讨论，主要确定软件的可行性。在这个阶段中我们需要从开发的技术、成本、效益等各个方面来衡量这个项目，进行可行性分析，形成可行性分析报告书，并以此为基础进行需求分析等后期的工作。

**3. 需求分析**

在此阶段主要解决的问题是“做什么”。在确定软件开发可行的情况下，对软件需要实现的各个功能进行详细分析，明确目标的功能需求和非功能需求，并建立分析模型，从功能、数据、行为等方面描述系统的静态特性和动态特性，对目标系统做进一步的细化，了解此系统的各种需求细节。在这个阶段实施时产生的需求分析说明书是今后开发过程中至关重要的一个文档，因此，需求分析阶段是整个生命周期中一个重要的阶段。随着系统的复杂性和规模的不断扩大，需求也会在整个软件开发过程中不断地发生变化，因此对需求的合理管理，是保护整个项目顺利进行的前提。

**4. 软件设计**

此阶段是整个软件开发的技术核心部分，解决“怎么做”的问题。主要是根据需求分析的结果，对整个软件系统进行设计，如系统框架设计、数据库设计等等。软件设计一般分为总体设计和详细设计。总体设计包括系统模块结构设计和计算机物理系统的配置方案设计。此过程中主要解决的是如何将一个系统划分成多个子系统，每个子系统如何划分成多个模块，如何确定子系统之间、模块之间传送的数据及其调用关系，如何评价并改进模块结构的质量等。计算机物理系统配置方案设计是要解决计算机软硬件系统的配置、通信网络系统的配置、机房设备的配置等问题。详细设计主要确定每个模块内部的详细执行过程，包括局部数据组织、控制流、每一步的具体加工要求等，一般来说，处理过程模块详细设计的难

度已不太大，关键是用一种合适的方式来描述每个模块的执行过程。

**5. 程序编码**

此阶段是选择合适的编程语言，将软件设计的结果转换成计算机可运行的程序代码。在程序编码中必须要制订统一、符合标准的编写规范，以保证程序的可读性、易维护性，提高程序的运行效率，且与设计相一致。

**6. 软件测试**

在软件开发完成后要经过严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。在测试过程中要建立详细的测试计划并严格按照测试计划进行测试，要根据需求规格说明书的要求，对必须实现的各项需求逐步进行确认，判定已开发的软件是否符合用户需求，能否交付用户使用。

**7. 软件运行和维护**

软件维护是软件生命周期中持续时间最长的阶段。在软件开发完成并投入使用后，由于多方面的原因，软件不能继续适应用户的要求，要延续软件的使用寿命，就必须对软件进行维护。

## 1.8　软件开发过程模型

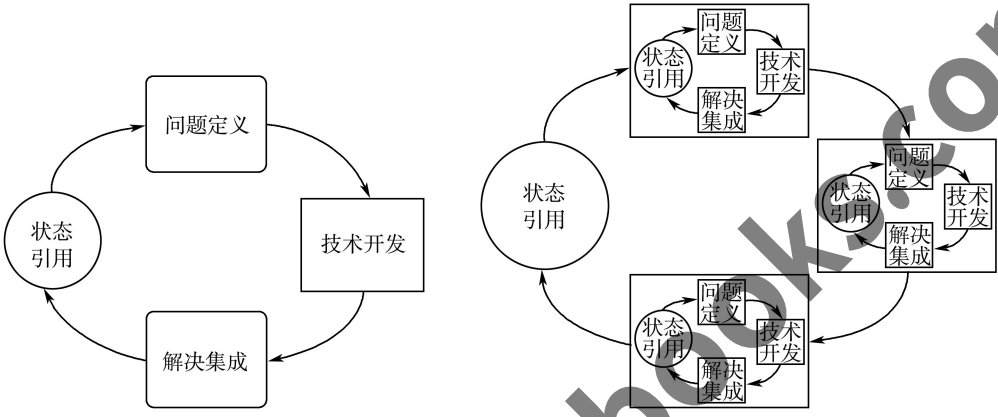
软件开发应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目，需要充分吸收和借鉴人类长期以来从事各种工程项目所积累的行之有效的管理、概念、技术和方法，特别要吸取几十年来人类从事计算机硬件研究和开发的经验教训。几十年的软件开发实践证明：按工程化的原则和方法组织管理软件开发工作是有效的，是摆脱软件危机的一个主要出路。为了解决软件危机，既要有技术措施（方法和工具），又要有必要的组织管理措施（例如软件质量管理、配置管理等）。软件工程正是从管理和技术两方面研究如何摆脱“软件危机”、如何更好地开发和维护计算机软件的一门新兴学科。

为了解决实际问题，软件工程师必须基于项目和应用的性质、采用的方法和工具以及需要控制和交付的产品，综合出一个开发策略。该策略包含过程、方法和工具三个层次以及定义阶段、开发阶段和支持阶段三个一般性阶段。这个策略常被称为过程模型或软件工程规范。

软件开发可以通过一个如图 1-4(a)所示的问题解决环进行刻画，环中包含 4 个不同阶段：状态引用、问题定义、技术开发和解决集成。状态引用表示事物的当前状态；问题定义标识要解决的特定问题；技术开发通过应用某些技术来解决问题；解决集成向需要解决方案的人提交结果。

上述问题解决环可以应用于软件工程的多个不同开发级别上，可以使用分形表示以提供关于过程的理想化视图，如图 1-4(b)所示。问题解决环的每一阶段又包含一个相同的问

题解决环，继续嵌套直到一个合理的边界（对于软件而言是代码行）。因为阶段内部和阶段之间的活动常常有交叉，很难清楚地划分出这4个阶段的活动，但是在某个细节的级别上它们同时共存，所以，在一个完整应用的分析和一小段代码的生成过程中可以递归地应用这4个阶段。




(a) 问题解决环的阶段　　(b) 问题解决环阶段中的阶段

**图 1-4　软件过程模型**

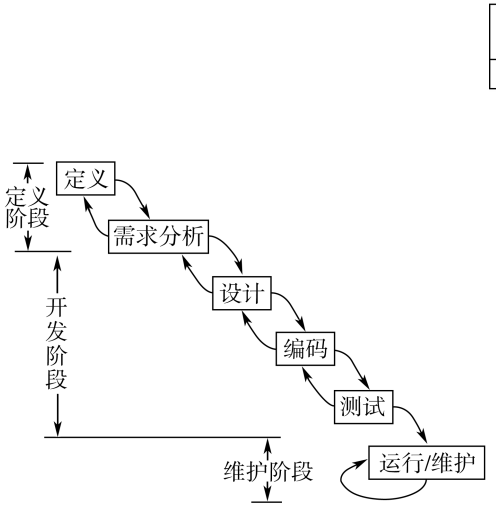
软件过程模型是从软件需求定义直至软件交付使用后报废为止，在这整个生存期中的系统开发、运行和维护所实施的全部过程、活动和任务的结构框架。到目前为止已经提出了多种模型，主要有线性顺序模型即传统的瀑布模型（Waterfall Model）、原型模型（Prototype Model）、螺旋模型（Spiral Model）、迭代开发与RUP等。模型的选择是基于软件的特点和应用的领域。下面介绍一些典型的过程模型。

## 1.8.1　瀑布模型

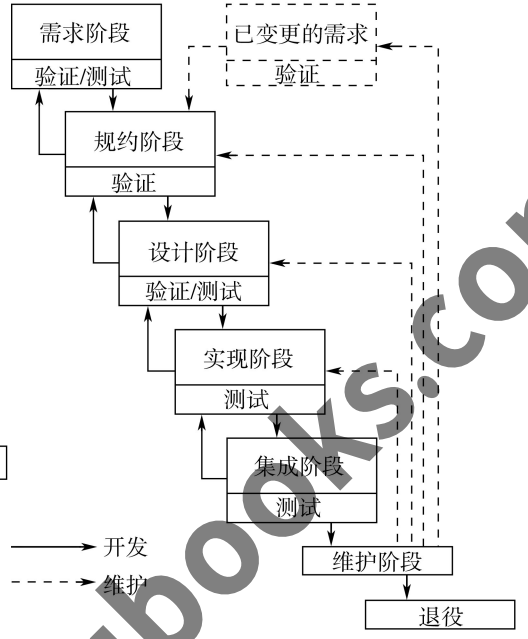
瀑布模型是最早出现的软件开发模型，在软件工程中占有重要的地位，它提供了软件开发的基本框架。瀑布模型提出了系统地按软件的生命周期顺序开发软件的方法，包括问题定义、需求分析、软件设计、编码、测试、运行和维护，如图1-5所示。各项活动自上而下、相互衔接，如同瀑布流水，逐级下落，体现了不可逆转性。

当然，软件开发的实践表明，上述各项开发活动之间并非完全是自上而下，呈现线性模式。实际上，要对每项活动实施的工作进行评审，若得到确认则继续下一项活动，在图1-5(b)中用向下的箭头表示；否则返回前面的活动进行返工，在图1-5(b)中用向上的箭头表示。

20多年来瀑布模型得到了广泛流行，一是由于它在支持开发结构化软件、控制并降低软件开发的复杂度、促进软件开发工程化方面起了显著作用；二是由于它为软件开发和维护提供了一种当时较为有效的管理模式，根据这一模式制订开发计划、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，有效地对整个软件开发过程进行指导，从而保证了软件产品及时交付，并达到预期的质量要求。我国曾在1988年依据该模型制订并公布了“软件开发规范”国家标准，对我国软件开发起到了较大的促进作用。




(a) 线性顺序模型（瀑布模型）表示一




(b) 线性顺序模型（瀑布模型）表示二

**图 1-5　瀑布模型的表示**

瀑布模型也被称为线性顺序模型、生命周期模型，它的优点表现在其强调开发的阶段性，强调早期计划和需求调查以及强调产品测试。但是在使用时有时会遇到如下一些问题：

1. 实际项目很少按照该模型给出的顺序进行。虽然线性模型允许迭代，但是间接的，在项目开发过程中可能会引起混乱。

2. 客户常常难以清楚地给出所有需求，而该模型却要求必须如此，所以它不能接受在许多项目的开始阶段自然存在的不确定性。

3. 客户必须有耐性，一直要等到项目开发周期的后期才能得到程序的运行版本，此时若发现大的错误，其后果可能是灾难性的。

4. 过分依赖于早期进行的需求调查，不能适应需求的变化；由于是单一流程，开发中的经验教训不能反馈应用于本产品的过程；风险往往迟至后期的开发阶段才显露，因而失去及早纠正的机会，项目开发往往失去控制。

除此之外，瀑布模型的线性特征还会导致“阻塞状态”，即某些项目团队成员不得不等待团队内的其他成员先完成其所依赖的任务，耽误的时间可能会超过在开发工作上的时间。

尽管存在以上问题，瀑布模型在软件工程中仍然占有肯定和重要的位置。它提供了一个模板，使分析、设计、编码、测试和支持的方法可以在此指导下应用。它适合以下类型的项目：

1. 需求简单清楚，并在项目初期就可以明确所有的需求。

2. 要求做好阶段审核和文档控制。

3. 不需要二次开发。

表 1-2 是瀑布模型中各个阶段的主要工作，以及相应的质量控制手段。

表 1-2 瀑布模型各阶段主要工作及质量控制手段

| 阶段 | 主要工作 | 应完成的文档 | 控制文档质量的手段 |
|---|---|---|---|
| 系统需求 | (1)调研用户需求及用户环境<br>(2)论证项目可行性<br>(3)制订项目初步计划 | (1)可行性报告<br>(2)项目初步开发计划 | (1)规范工作程序及编写文档<br>(2)对可行性报告及项目初步开发计划进行评审 |
| 需求分析 | (1)确定系统运行环境<br>(2)建立系统逻辑模型<br>(3)确定系统功能及性能要求<br>(4)编写需求规约、用户手册概要、测试计划<br>(5)确认项目开发计划 | (1)需求规约<br>(2)项目开发计划<br>(3)用户手册概要<br>(4)测试计划 | (1)在进行需求分析时采用成熟的技术与工具,如结构化分析<br>(2)规范工作程序及编写文档<br>(3)对已完成的4种文档进行评审 |
| 设计 | 概要设计：(1)建立系统总体结构,划分功能模块<br>(2)定义各功能模块接口<br>(3)数据库设计(如果需要)<br>(4)制订组装测试计划 | (1)概要的设计说明书<br>(2)数据库设计说明书(如果有)<br>(3)组装测试计划 | (1)在进行系统设计时采用先进的技术与工具,如结构化设计、结构图<br>(2)编写规范化工作程序及文档<br>(3)对已完成的文档进行评审 |
| | 详细设计：(1)设计各模块具体实现算法<br>(2)确定模块间详细接口<br>(3)制订模块测试方案 | (1)详细的设计说明书<br>(2)模块测试计划 | (1)设计时采用先进的技术与工具,如结构图SC<br>(2)规范工作程序及编写文档<br>(3)对已完成的文档进行评审 |
| 实现 | (1)编写程序源代码<br>(2)进行模块测试和调试<br>(3)编写用户手册 | (1)程序调试报告<br>(2)用户手册 | (1)在实现过程中采用先进的技术与工具,如结构图SC<br>(2)规范工作程序及编写文档<br>(3)对实现过程及已经完成的文档进行评审 |
| 测试 | 集成测试：(1)执行集成测试计划<br>(2)编写集成测试报告 | (1)系统的源程序清单<br>(2)集成测试报告 | (1)测试时采用先进的技术和工具<br>(2)规范工作程序及文档编写<br>(3)对测试工作及已经完成的文档进行评审 |
| | 验收测试：(1)测试整个软件系统(鲁棒性测试)<br>(2)试用户手册<br>(3)编写开发总结报告 | (1)确认测试报告<br>(2)用户手册<br>(3)开发工作总结 | |
| 维护 | (1)为纠正错误,完善应用而进行修改<br>(2)对修改进行配置管理<br>(3)编写故障报告和修改报告<br>(4)修订用户手册 | (1)故障报告<br>(2)修改报告 | (1)维护时采用先进的工具<br>(2)规范工作程序及编写文档<br>(3)配置管理<br>(4)对维护工作及已经完成的文档进行评审 |

## 1.8.2　原型模型

由于在项目开发的初始阶段人们对软件的需求认识常常不够清晰，因而使开发项目难于做到一次开发成功，出现返工在所难免。因此，可以先做试验开发，以探索可行性并弄清软件需求，在此基础上获得较为满意的软件产品。通常把第一次得到的试验性产品称为“原型(prototype)”，即把系统主要功能和接口通过快速开发制作为“软件样机”，以可视化的形式展现给用户，及时征求用户意见，从而明确无误地确定用户需求，同时也可用于征求内部意见，作为分析和设计的接口之一，以便于沟通。

原型实现模型的基本思想是：原型实现模型从需求采集开始，如图 1-6 所示；然后是“快速设计”，集中于软件中那些对用户/客户可见的部分的表示(如输入方式和输出格式)并最终导致原型的创建。这个过程是迭代的，原型由用户/客户评估并进一步精化待开发软件的需求，通过逐步调整以满足用户要求，同时也使开发者对将要做的事情有一个更好的理解。

原型实现模型的主要价值是可视化，强化沟通，降低风险，节省后期变更成本，提高项目成功率。一般来说，采用原型实现模型后可以改进需求质量；虽然先期投入了较多的时间，但可以显著减少后期变更的时间；原型投入的人力成本代价并不大，但可以节省后期成本；对于较大型的软件来说，原型系统可以成为开发团队的蓝图。另外，原型通过充分和客户交流，还可以提高客户的满意度。

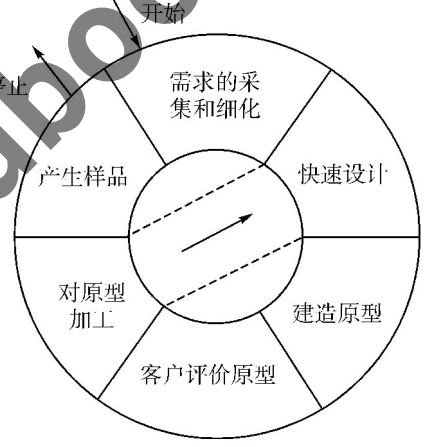



图 1-6　原型实现模型

对原型实现模型的基本要求有：体现主要的功能；提供基本的界面风格；展示比较模糊的部分，以便于确认或进一步明确，防患于未然；原型最好是可运行的，至少在各主要功能模块之间能够建立相互衔接。

根据运用原型的目的和方式不同，原型可分为以下两种不同的类型：

1. 抛弃型或丢弃型。先构造一个功能简单而且质量要求不高的模型系统，针对这个模型系统反复进行分析修改，形成比较好的设计思想，据此设计出更加完整、准确、一致、可靠的最终系统。系统构造完成后，原来的模型系统就被丢弃不用。这种类型通常是针对系统的某些功能进行实际验证，本质上仍然属于瀑布模型，只是以原型作为一种辅助的验证手段。

2. 演化型或追加型。先构造一个功能简单而且质量要求不高的模型系统，作为最终系统的核心，然后通过不断地扩充修改，逐步追加新要求，最后发展成为最终系统。软件的原型是最终系统的第一次演化。也就是说，首先进行需求调研和分析，然后选择一个优秀的开发工具快速开发出一个原型来请用户试用，用户经过试用提出修改建议，开发人员修改原型，再返回到用户进行试用，这个过程经过多次反复直到最终使用满意为止。

有人把抛弃型原型又细分为探索型和实验型。探索型原型的目的是要弄清对目标系统

的要求，确定所希望的特性，并探讨多种方案的可行性。它主要针对开发目标模糊、用户和开发者对项目都缺乏经验的情况。而实验型原型用于大规模开发和实现之前，考核方案是否合适，规约是否可靠。

一般小项目不采用抛弃型原型，否则成本和代价通常会偏高。演化型原型法主要针对事先不能完整定义需求的软件开发。用户可以给出待开发系统的核心需求，并且当看到核心需求实现后，能够有效地提出反馈，以支持系统的最终设计和实现。软件开发人员根据用户的需求，首先开发核心系统。当该核心系统投入运行，经过用户试用后，完成他们的工作，并提出精化系统、增强系统能力的需求。软件开发人员根据用户的反馈，实施开发的迭代过程。每一迭代过程均由需求、设计、编码、测试、集成等阶段组成，如图 1-7 所示。

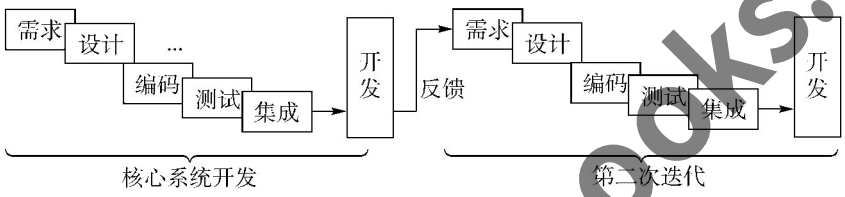



**图 1-7　演化模型**

使用演化模型具有以下好处：

1. 任何功能一经开发就能进入测试，以便验证是否符合产品需求。

2. 帮助导引出高质量的产品要求。如果一开始无法弄清楚所有的产品需求，也可以分批取得，而对于已提出的产品需求则可根据对现阶段原型的试用而做出修改。

3. 风险管理较少，可以在早期就获得项目进程数据，可据此对后续的开发循环做出比较切实的估算，提供机会去采取早期预防措施，增加项目成功的概率。

4. 有助于早期建立产品开发的配置管理、产品构建、自动化测试、缺陷跟踪、文档管理，均衡整个开发过程的负荷。

5. 开发中的经验教训能反馈应用于本产品的下一个循环过程，大大提高质量与效率。

6. 风险管理中若发现资金或时间已超出可承受的程度，则可以调整后续开发，或在一个适当时刻结束开发，但仍然要有一个具有部分功能的、可使用的产品。

7. 开发人员早日见到产品的雏形，可在心理上获得一种鼓舞。

8. 提高产品开发各过程的并行化程度。用户可以在新的一批功能开发测试后，立即参加验证，以提供有价值的反馈。此外，销售工作也有可能提前进行，因为可以在产品开发的中后期取得包含了主要功能的产品原型去向客户作展示和试用。

演化模型同时也存在一些不利之处：在一开始如果所有的产品需求没有完全弄清楚，会给总体设计带来困难并削弱产品设计的完整性，最终影响产品性能的优化及产品的可维护性；如果缺乏严格的过程管理，这个生命周期模型就很可能退化为一种原始的无计划的“试验—出错—改正”模式；心理上松懈，可能会认为虽然不能完成全部功能，但还是构造出了一个有部分功能的产品；如果不加控制地让用户接触开发中尚未测试稳定的功能，可能对开发人员和用户都会产生负面的影响。

理想情况下原型可以作为标识软件需求的一种机制，但它仍然存在问题：

1. 客户看到的似乎是软件的工程版本，但他们不知道原型只是拼凑起来的，不知道为了

使原型很快能够工作而没有考虑软件的总体质量和长期的可维护性。为了达到其高质量，程序员不得不反复修改原型使其成为其最终的工作产品，却放松了软件开发管理。

2. 开发者为使原型能够尽快工作，原型中常常会存在一些考虑欠成熟的方面，长时间后开发者可能已经习惯了这些选择，忘记了它们不合适的初始原因，最终这些不理想的选择就会成为系统的组成部分。

尽管存在以上问题，原型仍是软件工程的一个有效模型，关键是定义开始时的执行规则，即客户和开发商两方面必须达到一致：原型被建造仅是为了定义需求，之后就被抛弃（或至少部分被抛弃），实际软件在充分考虑了质量和可维护性之后才能被开发。

原型法在软件过程中的地位如图 1-8 所示。采用原型模型的一般过程如图 1-9 所示。

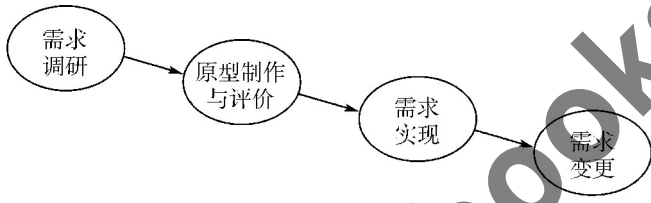



**图 1-8　软件原型的地位**

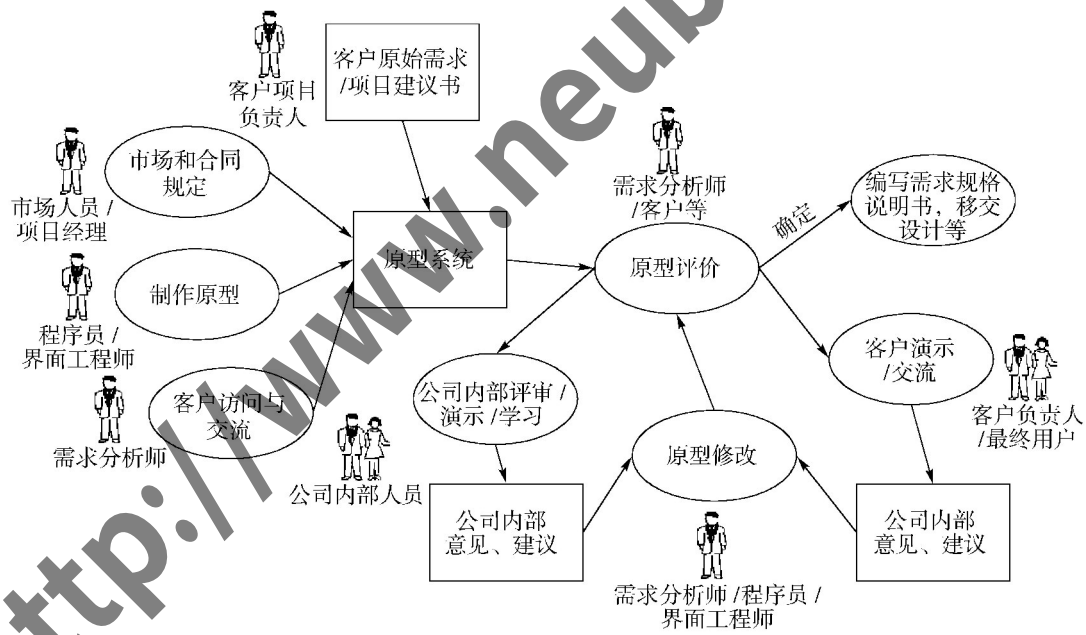



**图 1-9　原型模型的处理过程**

最后还要注意界面设计的引入。将界面风格在原型阶段就基本确定是一种优化的做法，可以避免后期开发时对界面进行统一调整所带来的不必要的成本花费。良好的界面可以使客户增加对系统的好感，这与系统功能的全面思考并不矛盾。

## 1.8.3　螺旋模型

1988 年美国 TRW 公司（B. W. Boehm）提出的螺旋模型是一种特殊的原型方法，适用于规模较大的复杂系统。它将原型实现的迭代特征与线性顺序模型中控制和系统化的方面结

合起来，并加入两者所忽略的风险分析，使软件增量版本的快速开发成为可能。软件项目风险的大小作为指引软件过程的一个重要因素，引入这一概念后可使软件开发被看作一种元模型，因为它能包容任何一个开发过程模型。在螺旋模型中，软件开发是一系列的增量发布：在早期的迭代过程中发布的增量可能是一个纸上的模型或者原型，在以后的迭代过程中逐步产生被开发系统的更加完善的版本。

螺旋模型被划分为若干个框架活动（或称任务区域），典型情况下沿着顺时针方向划分为3～6个任务区域。图1-10画出了包含6个任务区域的螺旋模型，在笛卡儿坐标的4个象限上分别表达了不同方面的活动，即：

1. 客户交流。确定需求、选择方案、设定约束条件。

2. 制订计划。定义资源、进度及其他相关项目信息所需的任务。

3. 风险分析。评估技术及管理的风险，制订控制风险措施的任务。

4. 实施工程。建立应用一个或多个表示所需要的任务。

5. 构造及发布。构造、测试、安装和提供用户支持（如文档和培训）所需要的任务。

6. 客户评估。对在工程阶段产生的或在安装阶段实现的软件表示的评估并获得客户反馈所需要的任务。

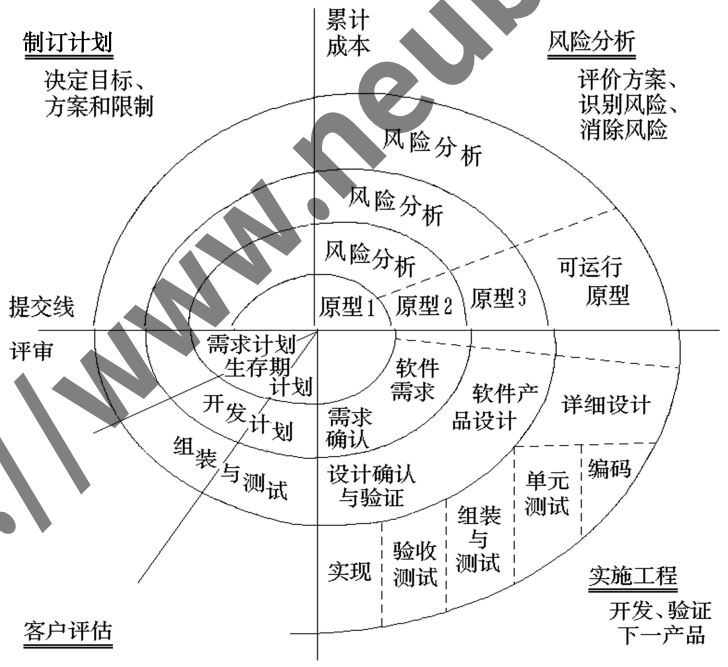



**图1-10　螺旋模型**

每一个区域都含有一系列适应待开发项目特点的工作任务，称为任务集合。对于较小的项目，工作任务的数目及其形式化程度均较低；对于较大的、关键的项目，每一个任务区域包含较多的工作任务以得到较高级别的形式化。

随着演化过程的开始，软件工程项目组按顺时针方向从核心开始沿螺旋移动，依次产生产品的规约、原型、软件更完善的版本。经过计划区域的每一圈都对项目计划进行调整，基于从客户评估得到的反馈调整费用和进度，并且项目管理者可以调整完成软件所需计划的

迭代次数。

螺旋模型在“瀑布模型”的每一个开发阶段之前，引入非常严格的风险识别、风险分析和风险控制，直到采取了消除风险的措施之后，才开始计划下一阶段的开发工作。否则，项目就很可能被取消。另外，如果有充足的把握判断遗留的风险已降低到一定的程度，项目管理人员可做出决定让余下的开发工作采用另外的生命周期模型。

对于大型系统及软件的开发，螺旋模型是一个很实用的方法。在软件过程的演化中，开发者和用户/客户能够更好地理解和对待每一个演化级别上的风险，所以，螺旋模型可以使用原型实现作为降低风险的手段，而且开发者在产品演化的任一阶段都可应用原型实现方法。螺旋模型在保持传统生命周期模型中系统的、阶段性的方法基础上，对其使用迭代框架，这就更真实地反映了现实世界，而且螺旋模型可以在项目的所有阶段直接考虑到技术风险，如果应用得当，就能够在风险出现之前降低它。因此，螺旋模型具有以下的优点：

1. 强调严格的全过程风险管理。

2. 强调各开发阶段的质量。

3. 提供机会检讨项目是否有价值继续下去。

但是，螺旋模型相对比较新，可能难以使用户/客户（尤其在合同情况下）相信演化方法是可行的，而且不像线性顺序模型或原型实现模型那样广泛应用，对其功效的完全确定还需要时间。此外，它需要非常严格的风险识别、风险分析和风险控制的专门技术，且其成功依赖于这种专门技术，这对风险管理的技术水平提出了很高的要求，还需要人员、资金和时间的较大投入。

### 1.8.4　迭代开发与 RUP

IBM Rational Unified Process(RUP)不仅仅是一个生命周期模型，也是一个支持环境（称为 RUP 平台），该开发环境帮助开发人员使用和遵从 RUP 生命周期。它以在线帮助、模板、指导等 HTML 或其他形式的文档提供帮助，是文档化的软件工程产品。RUP 支持环境是 IBM 软件工程套件中的重要组成部分，但作为一个生命周期模型，各个组织可根据自身的实际情况，以及项目规模对 RUP 进行裁剪和修改，因此，它可以应用于任何软件产品的开发。RUP 有三大特点：

1. 软件开发是一个迭代过程；

2. 软件开发是由用例驱动的；

3. 软件开发是以构架设计（Architectural Design）为中心的。

RUP 强调软件开发是一个迭代模型（Iterative Model），它定义了四个阶段（Phase）：初始（Inception）、细化（Elaboration）、构造（Construction）、交付（Transition）。其中每个阶段都有可能经历以上所提到的从商务需求分析开始的各个步骤，只是每个步骤的高峰期会发生在相应的阶段，例如开发实现的高峰期是发生在构造阶段。实际上这样的一个开发方法论是一个二维模型。这种迭代模型的实现在很大程度上提供了及早发现隐患和错误的机会，因此被现代大型信息技术项目所采用。

RUP 的另一大特征是用例驱动。用例是 RUP 方法论中一个非常重要的概念。简单地说，一个用例就是系统的一个功能。在系统分析和系统设计中，用例被用来将一个复杂的庞

大系统分割、定义成一个个小的单元，这个小的单元就是用例。然后以每个小的单元为对象进行开发。按照RUP过程模型的描述，用例贯穿整个软件开发的生命周期。在需求分析中，客户或用户对用例进行描述；在系统分布和系统设计过程中，设计师对用例进行分析；在开发实现过程中，开发编程人员对用例进行实现；在测试过程中，测试人员对用例进行检验。

RUP的第三大特征是它强调软件开发是以构架为中心的。构架设计(Architectural Design)是系统设计的一个重要组成部分。在构架设计过程中，设计师(Architect)必须完成对技术和运行平台的选取，整个项目的基础框架(Framework)的设计，完成对公共组件的设计，如审计(Auditing)系统、日志(Log)系统、错误处理(Exception Handling)系统、安全(Security)系统等。设计师必须对系统的可扩展性(Extensibility)、安全性(Security)、可维护性(Maintainability)、可延拓性(Scalability)、可重用性(Reusability)和性能(Performance)提出可行的解决方案。

RUP生命周期模型是一个二维的软件开发模型，如图1-11所示。纵轴代表核心工作流，是静态的一面，是软件周期活动和支持活动。横轴代表时间，显示过程动态的一面，它表示整个过程消耗的时间，用工作流程、周期、阶段、迭代、里程碑等名词描述。

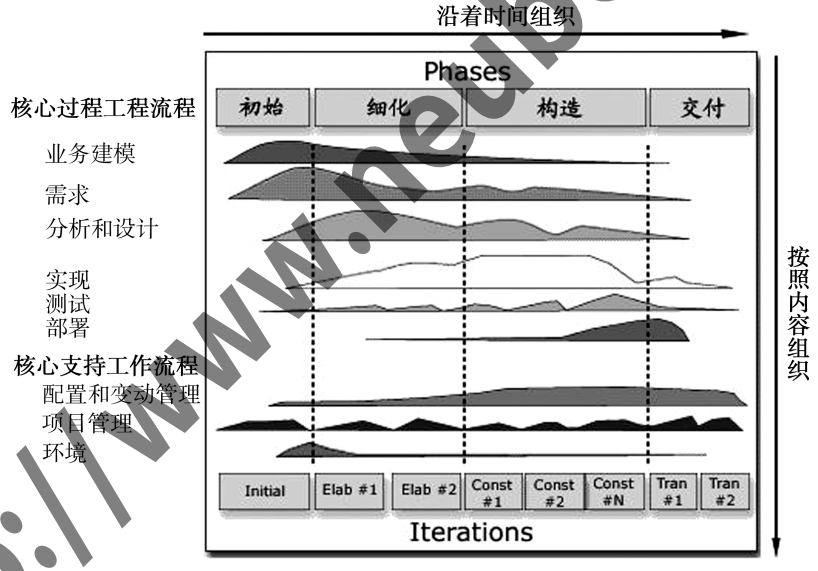



**图1-11 RUP过程模型**

纵轴表示的是在每次迭代过程中都要经历的工作流程(有一定顺序的活动)。核心过程工作流程中描述了软件生命周期过程中的各个阶段。核心支持工作流程则是一直贯穿于生命周期活动中的管理部分的内容。

1. 业务建模。理解待开发系统所在的机构及其商业运作，确保所有人员对它有共同的认识，评估待开发系统对结构的影响。

2. 需求。定义系统功能及用户界面，为项目预算及计划提供基础。

3. 分析与设计。把需求分析结果转换为分析与设计模型。

4. 实现。把设计模型转换为实现结果，并做单元测试，集成为可执行系统。

5. 测试。验证所有需求是否已经被正确实现，对软件质量提出改进意见。

6. 部署。打包、分发、安装软件，培训用户及销售人员。

7. 配置与变更管理。跟踪并维护系统开发过程中产生的所有制品的完整性和一致性。

8. 项目管理。为软件开发项目提供计划、人员分配、执行、监控等方面指导，为风险管理提供框架。

9. 环境。为软件开发机构提供软件开发环境。

从横轴来看，RUP 把软件开发生命周期划分为多个迭代，每个迭代生成产品的一个新版本。整个软件开发生命周期由 4 个连续阶段组成，它们分别是初始阶段、细化阶段、构造阶段、交付阶段。下面逐一为大家进行解释。

**1. 初始阶段**

初始阶段主要是定义最终产品视图和业务模型，确定系统范围。

RUP 不是瀑布模型，初始阶段作为 RUP 的第一个阶段不需要完成所有需求或建立可靠预算和计划。这些内容是在细化的过程中逐步完成的。大部分的需求分析是在细化阶段进行的，并且伴以具有产品品质的早期编程和测试。因此，大多数项目的初始阶段持续的时间相对较短，例如耗时一周或几周。

在初始阶段主要的工作包括如下内容：

- 简短的需求讨论会；
- 确定大多数参与者、目标和用例名称；
- 以摘要形式编写大多数用例；
- 以详细形式编写 10%～20%的用例；
- 确定大多数质量需求；
- 编写设想和补充性规格说明；
- 列出风险列表；
- 技术上的概念验证原型和其他调查；
- 面向用户界面的原型；
- 对购买/构建/复用构件的建议；
- 对候选的高层架构和构件给出建议；
- 第一次迭代的计划；
- 候选工具列表。

如果这个时期过长，那么往往是需求规格说明和计划过度的表现。用一句话来概括初始阶段要解决的主要问题：是否就项目设想基本达成一致，项目是否值得继续进行认真研究。

**2. 细化阶段**

细化阶段主要设计、确定系统的体系结构，制订工作计划及资源要求。

它是最初的一系列迭代，在这一阶段中，小组进行细致的调查、实现（编程和测试）核心架构、澄清大多数需求和应对高风险问题。在 RUP 中“风险”包含业务价值，因此早期工作包括实现那些被认为重要的场景，而不是专门针对技术风险。

细化阶段通常由两个或多个迭代组成，建议每次迭代的时间为 2～6 周。最好采用时间较短的迭代，除非开发团队规模庞大。每次迭代都是时间定量的，这意味着其结束日期是固

定的。

细化不是设计阶段，不是要完成所有模型的开发，也不是创建可以丢弃的原型，与之相反，该阶段产生的代码和设计是具有产品品质的最终系统的一部分。

用一句话来概括细化阶段：构建核心架构，解决高风险元素，定义大部分需求，以及预计总体进度和资源。

(1)关键思想和最佳实践

- 实行短时间定量、风险驱动的迭代；
- 及早开始编程；
- 对架构的核心和风险部分进行适应性的设计、实现和测试；
- 尽早、频繁、实际地测试；
- 基于来自测试、用户、开发者的反馈进行调整；
- 通过一系列讨论会，详细编写大部分用例和其他需求，每个细化迭代举行一次讨论会。

(2)迭代1结束时应该完成的任务

- 所有软件已经被充分测试；
- 客户定期地参与对已完成部分的评估，从而使开发人员获得对调整和澄清需求的反馈；
- 已经对(子)系统进行了完整的集成和固化，使其成为基线化的内部版本；
- 迭代计划会议；
- 对UI的可用性分析和工程也正在进行中；
- 数据库建模和实现也正在进行中；
- 举行另一个为期两天的需求讨论会。

**3. 构造阶段**

构造阶段构造产品并继续演进需求、体系结构、计划直至产品提交。

在此阶段将会涉及两个重要概念：(1)程序重构：指对程序中与新添功能相关的成分进行适当改造，使其在结构上完全适合新功能的加入。(2)模式：解决相似问题的不同解决方案。

此阶段要建立类图、交互图和配置图，如一个类具有复杂的生命周期，可绘制状态图；如算法特别复杂，可绘制活动图。

**4. 交付阶段**

交付阶段把产品提交给用户使用。

迭代式开发关键在于规范化地进行整个开发过程。在交付阶段，不能再开发新的功能(除了个别小功能或非常基本的以外)，而只是集中精力进行纠错工作，优化工作。

如果你在使用RUP的过程中，与下述一点或几点的看法一致，就说明你没有真正理解RUP。

- 在开始设计或实现之前试图定义大多数需求；
- 在编码之前试图做绝大部分需求分析和设计，将项目的主要测试和评估放到项目的最后；

- 在编程之前花费数日或数周进行 UML 建模；
- 认为“初始阶段＝需求阶段、细化阶段＝设计阶段、构造阶段＝实现阶段”；
- 认为细化的目的是完整仔细地定义模型，以能够在构造阶段将其转换成代码；
- 试图对项目从开始到结束制订详细计划，并试图预测所有迭代，以及每个迭代中可能发生的事情；
- 没有进行迭代开发；
- 迭代周期过长；
- 每次迭代都以产品发布作为结束（提交而非发布）；
- 细化阶段的目标是提交一个用之即抛弃的原型（应该是起始阶段使用原型）；
- 使用预见性的计划；
- 在细化阶段完成之前就完成构架文档。

RUP 是一个通用的过程模板，包含很多开发指南、制品、开发过程所涉及的角色说明，因为它非常庞大所以对具体的开发机构和项目，用 RUP 时还要做裁剪，也就是要对 RUP 进行配置。RUP 就像一个元过程，通过对 RUP 进行裁剪可以得到很多不同的开发过程，这些软件开发过程可以看作 RUP 的具体实例。RUP 裁剪可以分为以下几步：

1. 确定本项目需要哪些工作流。RUP 的 9 个核心工作流并不总是需要的，可以进行取舍。

2. 确定每个工作流需要哪些制品。

3. 确定 4 个阶段之间如何演进。确定阶段间演进要以风险控制为原则，决定每个阶段有哪些工作流，每个工作流执行到什么程度，制品有哪些，每个制品完成到什么程度。

4. 确定每个阶段内的迭代计划。规划 RUP 的 4 个阶段中每次迭代开发的内容。

5. 规划工作流的内部结构。工作流所涉及的角色、活动及制品，它的复杂程度与项目规模即角色多少有关。最后规划工作流的内部结构，通常用活动图的形式给出。

### 1.8.5　敏捷软件开发

从 1990 年代开始，一些新型软件开发方法逐渐引起广泛关注，敏捷软件开发就是其中的一种。敏捷软件开发又称敏捷开发，是一种应对快速变化的需求的一种软件开发能力。它们的具体名称、理念、过程、术语都不尽相同，相对于“非敏捷”，更强调程序员团队与业务专家之间的紧密协作、面对面的沟通（认为比书面的文档更有效）、频繁交付新的软件版本、紧凑而自我组织型的团队、能够很好地适应需求变化的代码编写和团队组织方法，也更注重软件开发中人的作用。

简言之，敏捷开发是一种以人为核心、迭代、循序渐进的开发方法。它不是一门技术，它是一种开发方法，也就是一种软件开发的流程，它会指导我们用规定的环节去一步一步完成项目的开发，而这种开发方式的主要驱动核心是人，它采用的是迭代式开发。

所谓“以人为核心”，是指敏捷开发不像瀑布开发模型那样以文档为驱动。在瀑布的整个开发过程中，要写大量的文档，把需求文档写出来后，开发人员都是根据文档进行开发的，一切以文档为依据。而敏捷开发只写有必要的文档，或尽量少写文档，它注重的是人与人之间面对面的交流，所以它强调以人为核心。

所谓"迭代",是指把一个复杂且开发周期很长的开发任务,分解为很多小周期可完成的任务,这样的一个周期就是一次迭代的过程;同时每一次迭代都可以生产或开发出一个可以交付的软件产品。

敏捷开发的具体方式主要包括 Scrum 和 XP,二者的区别在于:Scrum 偏重于过程,XP 则偏重于实践。

**1. 敏捷开发技术 1:Scrum**

Scrum 软件开发模型是敏捷开发的一种,在近几年内逐渐流行起来。Scrum 的英文意思是橄榄球运动的一个专业术语,表示"争球"的动作,把一个开发流程的名字取名为 Scrum,寓意为开发团队在开发一个项目时,大家像打橄榄球一样迅速、富有战斗激情、人人你争我抢地完成它。而 Scrum 就是这样的一个开发流程,运用该流程,能够看到团队高效的工作。

Scrum 的基本假设是:开发软件就像开发新产品,无法一开始就能定义软件产品最终的规程,过程中需要研发、创意、尝试错误,所以没有一种固定的流程可以保证专案成功。Scrum 将软件开发团队比拟成橄榄球队,有明确的最高目标,熟悉开发流程中所需具备的最佳典范与技术,具有高度自主权,紧密地沟通合作,以高度弹性解决各种挑战,确保每天、每个阶段都朝向目标,有明确的推进。

Scrum 开发流程通常以 30 天(或者更短的一段时间)为一个阶段,从客户提供新产品的需求规格开始,开发团队与客户于每一个阶段开始时挑选该完成的规格部分,开发团队必须尽力于 30 天后交付成果,团队每天用 15 分钟开会检查每个成员的进度与计划,了解所遭遇的困难并设法排除。

首先介绍有关 Scrum 的几个名词:

Backlog:可以预知的所有任务,包括功能性的和非功能性的所有任务。

Sprint:一次迭代开发的时间周期,一般最多以 30 天为一个周期,在这段时间内,开发团队需要完成一个制订的 Backlog,并且最终成果是一个增量的、可以交付的产品。

Sprint Backlog:一个 Sprint 周期内所需要完成的任务。

Scrum Master:负责监督整个 Scrum 进程,修订计划的一个团队成员。

Time-Box:一个用于开会的时间段。比如每个 Daily Scrum Meeting 的 Time-Box 为 15 分钟。

Sprint Planning Meeting:在启动每个 Sprint 前召开,一般为一天时间(8 小时),该会议需要制订的任务是:产品负责人和团队成员将 Backlog 分解成小的功能模块,决定在即将进行的 Sprint 里需要完成多少小功能模块,确定好这个 Product Backlog 的任务优先级。另外,该会议还需详细地讨论如何能够按照需求完成这些小功能模块。制订的这些模块的工作量以小时计算。

Daily Scrum Meeting:开发团队成员召开,一般为 15 分钟。每个开发成员需要向 Scrum Master 汇报三个项目:今天完成了什么?是否遇到了障碍?即将要做什么?通过该会议,团队成员可以相互了解项目进度。

Sprint Review Meeting:在每个 Sprint 结束后,这个 Team 将这个 Sprint 的工作成果演示给产品负责人和其他相关的人员。一般该会议为 4 小时。

Sprint Retrospective Meeting：对刚结束的 Sprint 进行总结。会议的参与人员为团队开发的内部人员。一般该会议为 3 小时。

Scrum 开发流程中的三大角色是：产品负责人(Product Owner)，主要负责确定产品的功能和达到要求的标准，指定软件的发布日期和交付的内容，同时有权力接受或拒绝开发团队的工作成果。流程管理员(Scrum Master)，主要负责整个 Scrum 流程在项目中的顺利实施和进行，以及清除挡在客户和开发工作之间的沟通障碍，使客户可以直接驱动开发。开发团队(Scrum Team)，主要负责软件产品在 Scrum 规定流程下进行开发工作，人数控制在 5～10人，每个成员可能负责不同的技术方面，但要求每个成员必须要有很强的自我管理能力，同时具有一定的表达能力；成员可以采用任何工作方式，只要能达到 Sprint 的目标。图 1-12 为 Scrum 开发模型的流程图。

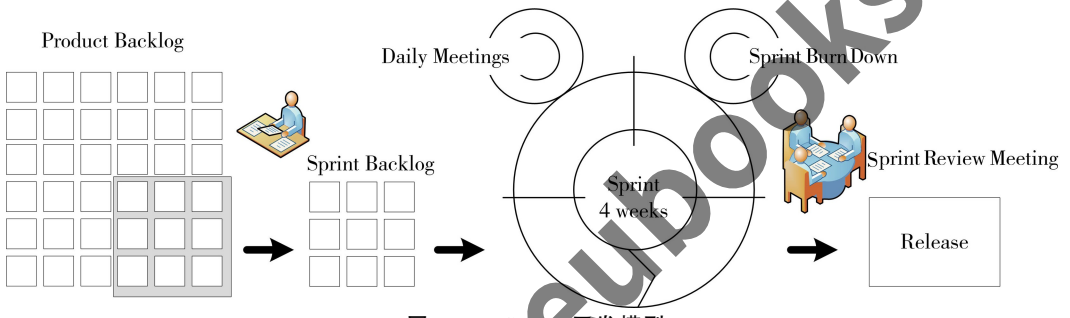



**图 1-12　Scrum 开发模型**

参照图 1-12，Scrum 的流程如下：

(1)确定一个 Product Backlog(按优先顺序排列的一个产品需求列表)，这是由产品负责人负责的。

(2)开发团队根据 Product Backlog 列表，做工作量的预估和安排。

(3)有了 Product Backlog 列表，需要通过 Sprint 计划会议来从中挑选出一个 Story 作为本次迭代完成的目标，这个目标的时间周期是 1～4 个星期，然后把这个 Story 进行细化，形成一个 Sprint Backlog。

(4)Sprint Backlog 是由开发团队去完成的，每个成员根据 Sprint Backlog 再细化成更小的任务(细到每个任务的工作量在 2 天内能完成)。

(5)在开发团队完成计划会议上选出的 Sprint Backlog 过程中，需要进行 Daily Scrum Meeting(每日站立会议)，每次会议控制在 15 分钟左右，每个人都必须发言，并且要向所有成员当面汇报你昨天完成了什么，并且向所有成员承诺你今天要完成什么，同时遇到不能解决的问题也可以提出，每个人回答完成后，要走到黑板前更新自己的 Sprint Burn Down (Sprint 燃尽图)。

(6)做到每日集成，也就是每天都要有一个可以成功编译、并且可以演示的版本；很多人可能还没有用过自动化的每日集成，其实 TFS 就有这个功能，它可以支持每次有成员进行签入操作的时候，在服务器上自动获取最新版本，然后在服务器中编译，如果通过则马上再执行单元测试代码，如果也全部通过，则将该版本发布，这时一次正式的签入操作才保存到 TFS 中，中间有任何失败，都会用邮件通知项目管理人员。

(7)当一个 Story 完成，也就是 Sprint Backlog 被完成，也就表示一次 Sprint 完成，这时，

我们要进行 Sprint Review Meeting(演示会议),也称为评审会议,产品负责人和客户都要参加(最好本公司老板也参加),每一个开发团队的成员都要向他们演示自己完成的软件产品(这个会议非常重要,一定不能取消)。

(8)最后就是 Sprint Retrospective Meeting(回顾会议),也称为总结会议,以轮流发言方式进行,每个人都要发言,总结并讨论改进的地方,放入下一轮 Sprint 的产品需求中。

**2. 敏捷开发技术 2:XP**

XP 是极限编程(Extreme Programming)的简称,由 KentBeck 在 1996 年提出。XP 一种"轻量型"的以编码为核心任务的灵活软件开发方法。与传统的软件开发方法不同,XP 摒弃了大多数重量型过程中的中间产物来提高软件开发速度,它是基于对影响软件开发速度的因素进行考查而发展起来的。这种开发方式适用于经常面临需求不明确或者需求快速变化的小到中型的软件开发团队。由于 XP 的迭代特性使其能够在开发过程中对需求的变化有很好的适应性,因而 XP 的目标便是:在最短的时间内将较为模糊、变化较大的用户需求转化为符合用户要求的软件产品。

XP 的内容包括四大价值观、五个原则和 12 个最佳实践:

(1)四大价值观

沟通:XP 方法论认为,如果小组成员之间无法做到持续的、无间断的交流,那么协作就无从谈起,从这个角度能够发现,通过文档、报表等人工制品进行交流面临巨大的局限性。因此,XP 组合了诸如对编程这样的最佳实践,鼓励大家进行口头交流,通过交流解决问题,提高效率。

简单:XP 方法论提倡在工作中秉承"够用就好"的思路,也就是尽量地简单化,只要今天够用就行,不考虑明天会发现的新问题。这一点看上去十分容易,但是要真正做到保持简单的工作其实很难。因为在传统的开发方法中,都要求大家对未来做一些预先规划,以便对今后可能发生的变化预留一些扩展的空间。正如对传统开发方法的认识一样,许多开发人员也会质疑 XP,保持系统的扩展性很重要,如果都保持简单,那么如何使系统能够有良好的扩展性呢?其实不然,保持简单的理由有两个:开发小组在开发时所做的规划,并无法保证其符合客户需要的,因此做的大部分工作都将落空,使开发过程中重复的、没有必要的工作增加,导致整体效率降低。另外,在 XP 中提倡时刻对代码进行重构,一直保持其良好的结构与可扩展性。也就是说,可扩展性和为明天设计并不是同一个概念,XP 是反对为明天考虑而工作,并不是说代码要失去扩展性。而且简单和沟通之间还有一种相对微妙的相互支持关系。当一个团队之间沟通越多,那么就越容易明白哪些工作需要做,哪些工作不需要做。另一方面,系统越简单,需要沟通的内容也就越少,沟通也将更加全面。

反馈:是什么原因使我们的客户、管理层这么不理解开发团队?为什么客户、管理层总是喜欢给我们一个死亡之旅?究其症结,就是开发的过程中缺乏必要的反馈。在许许多多项目中,当开发团队经历过了需求分析阶段之后,在相当长的一段时间内,是没有任何反馈信息的。整个开发过程对于客户和管理层而言就像一个黑盒子,进度完全是不可见的。

在项目的过程中,这样的现象不仅出现在开发团队与客户、管理层之间,还包括开发团队内部。这一切问题都需要我们更加注重反馈。反馈对于任何软件项目的成功都是至关重要的,而在 XP 方法论中则更进一步,通过持续、明确的反馈来暴露软件状态的问题。具体

而言就是：

● 在开发团队内部，通过提前编写单元测试代码，时时反馈代码的问题与进展。

● 在开发过程中，还应该加强集成工作，做到持续集成，使每一次增量都是一个可执行的工作版本，也就是逐渐使软件长大，整个过程中，应该通过向客户和管理层演示这些可运行的版本，以便及早地反馈，及早地发现问题。

同时，我们会发现，反馈与沟通也有着良好的配合，及时和良好的反馈有助于沟通。而简单的系统更有利于测试盒反馈。

勇气：在应用 XP 方法论时，我们每时每刻都在应对变化：由于沟通良好，因此会有更多需求变更的机会；由于时刻保持系统的简单，因此新的变化会带来一些重新开发的需要；由于反馈及时，因此会有更多中间打断你的思路的新需求。

总之这一切，使你立刻处于变化之中，这时就需要你有勇气来面对快速开发，面对可能的重新开发。也许你会觉得，为什么要让我们的开发变得如此零乱，但是其实这些变化若你不让它及早暴露，那么它就会迟一些出现，并不会因此消亡，因此，XP 方法论让它们早出现、早解决，是实现“小步快走”开发节奏的好办法。

XP 方法论要求开发人员穿上强大、自动测试的盔甲，勇往直前，在重构、编码规范的支持下，有目的地快速开发。

勇气可以来源于沟通，因为它使高风险、高回报的试验成为可能；勇气可以来源于简单，因为面对简单的系统，更容易鼓起勇气；勇气可以来源于反馈，因为你可以及时获得每一步前进的状态（自动测试），会使你更勇于重构代码。

（2）五个原则

快速反馈：及时地、快速地获取反馈，并将所学到的知识尽快地投入系统中去，是指开发人员应该通过较短的反馈循环，迅速地了解现在的产品是否满足了客户的需求。这也是对反馈这一价值观的进一步补充。

简单性假设：简单性假设原则是对简单这一价值观的进一步补充。这一原则要求开发人员将每个问题都看得十分容易解决，也就是说只为本次迭代考虑，不去想未来可能需要什么，相信具有将来必要时增加系统复杂性的能力，也就是号召大家出色地完成今天的任务。

逐步修改：就像开车打方向盘一样，不要一次做出很大的改变，那样将会使可控性变差，更适合的方法是进行微调。而在软件开发中，这样的道理同样适用，任何问题都应该通过一系列能够带来差异的微小改动来解决。

提倡更改：在软件开发过程中，最好的办法是在解决最重要问题时，保留最多选项的那个。也就是说，尽量为下一次修改做好准备。

四大价值观之外：在这四大价值观之下，隐藏着一个更深刻的东西，那就是尊重。因为这一切都建立在团队成员之间的相互关心、相互理解的基础之上。

（3）12 个最佳实践

计划游戏：计划游戏的主要思想就是先快速地制订一份概要的计划，然后随着项目细节的不断清晰，再逐步完善这份计划。计划游戏产生的结果是一套用户故事及后续的一两次迭代的概要计划。“客户负责业务决策，开发团队负责技术决策”是计划游戏获得成功的前提条件。也就是说，系统的范围、下一次迭代的发布时间、用户故事的优先级应该由客户决

定;而每个用户故事所需的开发时间、不同技术的成本、如何组建团队、每个用户故事的风险,以及具体的开发顺序应该由开发团队决定。

首先客户和开发人员坐在同一间屋子里,每个人都准备一支笔、一些用于记录用户故事的纸片,最好再准备一个白板,就可以开始了。

客户编写故事,即由客户谈论系统应该完成什么功能,然后用通俗的自然语言,使用自己的语汇,将其写在卡片上,这就是用户故事。

开发人员进行估算,即首先客户按优先级将用户故事分成必须要有、希望有、如果有更好三类,然后开发人员对每个用户故事进行估算,先从高优先级开始估算。如果在估算的时候,感到有一些故事太大,不容易进行估算,或者是估算的结果超过 2 人/周,那么就应该对其进行分解,拆成 2 个或者多个小故事。

确定迭代的周期:接下来就是确定本次迭代的时间周期,这可以根据实际的情况进行确定,不过最佳的迭代周期是 2～3 周。有了迭代的时间之后,再结合参与的开发人数,算出可以完成的工作量总数。然后根据估算的结果,与客户协商,挑出时间上够、优先级合适的用户故事组合,形成计划。

小型发布:XP 方法论秉承的是"持续集成,小步快走"的哲学,也就是说每一次发布的版本应该尽可能的小,当然前提条件是每个版本有足够的商业价值,值得发布。由于小型发布可以使集成更频繁,客户获得的中间结果也越频繁,反馈也就越频繁,客户就能够实时地了解项目的进展情况,从而提出更多的意见,以便在下一次迭代中计划进去,以实现更高的客户满意度。

隐喻:相对而言,隐喻这一个最佳实践是最令人费解的。什么是隐喻呢?根据词典中的解释是:"一种语言的表达手段,它用来暗示字面意义不相似的事物之间的相似之处"。那么这在软件开发中又有什么用呢?总结而言,常常用于四个方面。寻求共识:也就是鼓励开发人员在寻求问题共识时,可以借用一些沟通双方都比较熟悉的事物来做类比,从而帮助大家更好地理解解决方案的关键结构,也就是更好地理解系统是什么、能做什么。发明共享词汇:通过隐喻,有助于提出一个用来表示对象、对象间关系的通用名称。例如,策略模式(用来表示可以实现多种不同策略的设计模式)、工厂模式(用来表示可以按需"生产"出所需的设计模式)等。创新的武器:有的时候,可以借助其他东西来找到解决问题的新途径。例如,"我们可以将工作流看做一个生产线"。描述体系结构:体系结构是比较抽象的,引入隐喻能够大大减轻理解的复杂度。例如,管道体系结构就是指两个构件之间通过一条传递消息的"管道"进行通信。

当然,如果能够找到合适的隐喻是十分快乐的,但并不是每种情况都可以找到恰当的隐喻,没有必要强求。

简单设计:强调简单设计的价值观,引出了简单性假设原则,落到实处就是"简单设计"实践。这个实践看上去似乎很容易理解,但又经常被误解,许多批评者就指责 XP 忽略设计是不正确的。其实,XP 的简单设计实践并不是要忽略设计,而且认为设计不应该在编码之前一次性完成,因为那样只能建立在"情况不会发生变化"或者"我们可以预见所有的变化"之类的谎言的基础上的。

Kent Beck 概念中简单设计是这样的:能够通过所有的测试程序;没有包括任何重复的

代码；清楚地表现了程序员赋予的所有意图；包括尽可能少的类和方法。

测试先行：为了鼓励程序员愿意甚至喜欢在编写程序之前编写测试代码，XP 方法论提供了许多有说服力的理由：如果已经保持了简单的设计，那么编写测试代码根本不难；如果在结对编程，那么如果想出一个好的测试代码，你的伙伴一定行；当所有的测试都通过的时候，再也不会担心所写的代码存在其他隐患；当你的客户看到所有的测试都通过的时候，会对程序充满前所未有的信心；当需要进行重构时，测试代码会给你带来更大的勇气，因为你要测试是否重构成功只需要一个按钮。所以，测试先行是 XP 方法论中一个十分重要的最佳实践，并且其中所蕴含的知识与方法也十分丰富。

重构：重构是一种对代码进行改进而不影响功能实现的技术，XP 需要开发人员在发现代码有隐患时，有重构代码的勇气。重构的目的是降低变化引发的风险，使代码优化更加容易。通常重构发生在两种情况之下。实现某个特性之前，尝试改变现有的代码结构，以使实现新的特性更加容易。实现某个特性之后，检查刚刚写完的代码后，认真检查一下，看是否能够进行简化。

重构技术是对简单性设计的一个良好的补充，也是 XP 中重视“优质工作”的体现，这也是优秀的程序员必备的一项技能。

结对编程：结对编程即和搭档一起写程序。自从 20 世纪 60 年代，就有类似的实践在进行，长期以来的研究结果证明，结对编程的效率比单独编程更高。一开始虽然会牺牲一些速度，但慢慢的，开发速度会逐渐加快，究其原因，主要是结对编程大大降低了沟通的成本，提高了工作的质量，具体表现在：所有的设计决策确保不是由一个人做出的；系统的任何一个部分都肯定至少有 2 个人以上熟悉；几乎不可能有 2 个人都忽略的测试项或者其他任务。

结对编程技术被誉为 XP 保证工作质量、强调人文主义的一个典型的实践，应用得当还能够使开发团队之间的协作更加流畅、知识交流与共享更加频繁，团队的稳定性也会更加稳固。

集体代码所有制：XP 方法论鼓励团队进行结对编程，而且认为结对编程的组合应该动态地搭配，根据任务的不同、专业技能的不同进行最优组合。由于每个人都肯定会遇到不同的代码，因此代码的所有制就不再适合于私有，因为那样会给修改工作带来巨大的不便。也就是说，团队中的每个成员都拥有对代码进行改进的权利，每个人都拥有全部代码，也都需要对全部代码负责。同时，XP 强调代码是谁破坏的（也就是修改后发生问题），就应该由谁来修复。由于在 XP 中有一些与之匹配的最佳实践，因此无须担心采用集体代码所有制会让代码变得越来越乱。在 XP 项目中，集成工作是一件经常性的工作，因此当有人修改代码而带来了集成的问题，会在很快的时间内被发现。每一个类都会有一个测试代码，因此不论谁修改了代码，都需要运行这个测试代码，这样偶然性的破坏发生的概率将很小。每一个代码的修改都是通过了结对的两个程序员共同思考，因此通常做出的修改都是对系统有益的。集成代码所有制是 XP 与其他敏捷方法的一个较大不同，也是从另一个侧面体现了 XP 中蕴含的很深厚的编码情节。

持续集成：在前面谈到小型发布、重构、结对编程、集体代码所有制等最佳实践的时候，多次看到“持续集成”的身影，可以说持续集成是对这些最佳实践的基本支撑条件。持续集成与小型发布不同，小型发布是指在开发周期经常发布中间版本；而持续集成的含义则是要

求 XP 团队每天尽可能多次地做代码集成，每次都在确保系统运行的单元测试通过之后进行。这样，就可以及早地暴露、消除由于重构、集体代码所有制所引入的错误，从而减少解决问题的痛苦。要在开发过程中做到持续集成并不容易，首先需要养成这个习惯，而且集成工作往往是十分枯燥、繁琐的，因此适当地引入每日集成工具是十分必要的。

每周工作 40 小时：XP 方法论认为，加班最终会扼杀团队的积极性，导致项目失败，这也充分体现了 XP 方法关注人的因素比关注过程的因素更多一些。Kent Beck 认为开发人员即使能够工作更长的时间，他们也不该这样做，因为这样做会使他们更容易厌倦编程工作，从而产生一些影响他们效能的其他问题。因此，每周工作 40 小时是一种顺势行为，是一种规律。其实对于开发人员和管理者来说，违反这种规律是不值得的。但是，“每周工作 40 小时”中的 40 不是一个绝对数，它所代表的意思是团队应该保证按照“正常的时间”进行工作。那么如何做到这一点呢？首先，定义符合你团队情况的“正常工作时间”；其次，逐步将工作时间调整到“正常工作时间”；再次，除非你的时间计划一团糟，否则不应该向时间妥协；最后，鼓起勇气，制订一个合情合理的时间表。

现场客户：为了保证开发出来的结果与客户的预想接近，XP 方法论认为最重要的需要是将客户请到开发现场。就像计划游戏中提到过的，在 XP 项目中，应该时刻保证客户负责业务决策，开发团队负责技术决策。因此，在项目中有客户在现场明确用户故事，并做出相应的业务决策，对于 XP 项目而言有着十分重要的意义。现场客户在具体实施时，也不是一定需要客户一直和开发团队在一起，而是开发团队应该和客户能够随时沟通，可以是面谈，可以是在线聊天，可以是电话，当然面谈是必不可少的。其中的关键是当开发人员需要客户做出业务决策时、需要进一步了解业务细节时能够随时找到相应的客户。

不过，也有一些项目是可以不要现场客户参与的：当开发组织中已经有相关的领域专家时；当做一些探索性工作，而且客户也不知道他想要什么时（例如新产品、新解决方案的研究与开发）。

编码标准：XP 方法论认为拥有编码标准可以避免团队在一些与开发进度无关的细节问题上发生争论，这种争论会给重构、结对编程带来很大麻烦。不过，XP 方法论的编码标准的目的不是创建一个事无巨细的规则表，而是能够提供一个确保代码清晰，便于交流的指导方针。

## 1.9 项目实施

**1. RUP 过程和制品裁剪**

国合项目是一个自动化办公系统，具有一般信息管理类系统的普遍特点，采用 RUP 开发过程模型更适于项目本身，更加适于项目教学案例。在 RUP 的初始阶段，我们不仅关注需求的获取和描述，而且还需要从项目管理的角度为整个项目制订宏观上的规划。编写开

发案例就是对项目的实施方案所做的一个统筹安排。

在 RUP 中，一个开发案例是项目管理者如何为当前的项目定制这一开发过程的描述，它将描述如何根据需要采用 RUP。项目团队可以从繁杂的 RUP 过程中抽取适当的过程和制品用于自己的项目中，从而对 RUP 进行有效的裁剪。

**2. 角色职责分工**

开发案例的另一个重要部分就是解释项目中每个不同角色的职责。特别是对于一个小型团队来讲，很可能每个人担任了不止一个角色，所以仔细定义所有这些职责是很重要的。团队成员需要明白他们的职责，需要了解这个项目期望他们产生哪些制品，他们可以使用哪些制品以及这些制品本身的形式。

即使对一个使用通用开发案例的大型机构来讲，每个项目团队也都会针对自己的项目对开发案例进行裁减。项目团队应该着重于制品而不是活动。关注制品可以使团队成员将精力集中在要完成的任务上，明确本阶段的目标。因此在开发案例中最重要的载体就是用于描述每一个制品的表格。表 1-3 是在开发案例中制品表格的常见格式。

**表 1-3　　开发案例中制品表格的格式**

| 制品 | 如何使用 | | | | 审阅的详细情况 | 使用的工具 | 负责人 |
|---|---|---|---|---|---|---|---|
| | 初始阶段 | 细化阶段 | 构造阶段 | 移交阶段 | | | |
| | | | | | | | |

下面对在表 1-3 中出现的属性进行简单说明。

(1)制品

描述制品的名称，对 RUP 中某种制品的引用，或是在开发案例中定义的某种局部制品。

(2)如何使用

描述在生命周期中如何使用该制品。判定在每一个阶段中该制品是否被生成或者被显著地修改。这一字段可能的值包括：C——在本阶段生成；M——在本阶段被修改；空缺——不需要进行审阅。

(3)审阅的详细情况

定义对该制品的审阅等级以及审阅的过程。“正规”表示由客户或者相关涉众进行审阅和签署。“非正规”表示由一个或多个团队成员进行审阅，不需要进行签署。“空缺”表示不需要审阅。

(4)使用的工具

定义用于产生该制品的开发工具，有关用于开发和维护该制品的工具的详细参考。

(5)负责人

负责该制品的角色。描述由哪一个角色，例如项目经理或开发人员，来负责保证该制品的完成。

**3. 国合项目 RUP 开发案例裁剪和角色分工**

表 1-4 简单描述了国合项目的开发案例。

表 1-4 计划的简单开发案例

| 工作流程 | 制品 | 如何使用 | | | | 审阅的详细情况 | 使用的工具 | 负责人 |
|---|---|---|---|---|---|---|---|---|
| | | 初始阶段 | 细化阶段 | 构造阶段 | 移交阶段 | | | |
| 业务建模 | 领域模型 | | C | | | | VSS,MS Word&MS Visio | 系统分析师 |
| 需求 | 用例模型 | C | M | | | 正规的 | VSS,MS Word&MS Visio | 系统分析师 |
| | 补充性规格说明 | C | M | | | 正规的 | VSS,MS Word&MS Visio | 系统分析师 |
| | 词汇表 | C | M | | | | VSS&MS Word | 系统分析师 |
| | 用户界面原型 | | C | | | | DreamWeaver | 用户界面设计师 |
| | 前景 | C | M | | | 正规的 | VSS&MS Word | 系统分析师 |
| 分析和设计 | 设计模型 | | C | M | | 正规的 | VSS,MS Word&MS Visio | 软件架构师 |
| | 软件架构文档 | | C | M | | 正规的 | VSS&MS Word | 技术文档作者 |
| | 数据模型 | | C | M | | 正规的 | VSS& PowerDesigner | 数据设计师 |
| 实现 | … | | | | | | | |
| 项目管理 | … | | | | | | | |
| … | … | | | | | | | |

在初始阶段团队成员的主要任务是确定项目的总体范围,明确项目的前景,因此,在初始阶段关注的是前景的创建,但随着细化阶段迭代的不断进行,前景的内容也会不断进行改进和完善。同时,为了能够更好地获得系统的功能性需求,团队成员采用用例技术对需求进行描述。因此,用例模型的构建将是初始阶段的又一重要的任务,随后大家将会了解到,用例模型的构建仍需要在细化阶段进行不断的修改和补充。由于用例模型的重要性,因此需要对阶段性的内容进行正规的评审以保证质量。在构建模型的过程中,强调了工具环境的要求,一方面保证项目初期能够尽早建立一个团队的开发环境,另外一方面也强调了文档书写的一致性。

# 1.10 知识拓展

## 1.10.1 精益开发

“精益软件开发”一词起源于 Mary Poppendieck 和 Tom Poppendieck 写的一本同名书