高等学校“十三五”规划教材

机电 MECHATRONICS
计算机 COMPUTER
电子 ELECTRONICS

面向对象分析设计与编程

主编 王燕 庞淑侠 胡文瑾

西安电子科技大学出版社
http://www.xduph.com

高等学校“十三五”规划教材

面向对象分析设计与编程

主　编　王　燕　庞淑侠　胡文瑾

西安电子科技大学出版社

内 容 简 介

本书从实用的角度出发，全面、详细地介绍了面向对象开发语言C++的基本知识以及利用UML进行面向对象分析和设计的方法，并利用一个综合性的案例，展示了利用UML进行软件建模的过程和步骤。

本书既可以作为计算机专业本科、研究生的面向对象技术教材，也可以作为软件技术培训教师、计算机软件领域的研究人员和工程技术人员的参考书。

图书在版编目(CIP)数据

面向对象分析设计与编程 / 王燕，庞淑侠，胡文瑾主编. —西安：西安电子科技大学出版社，2018.6

ISBN 978-7-5606-4938-2

Ⅰ. ① 面… Ⅱ. ① 王… ② 庞… ③ 胡… Ⅲ. ① 面向对象语言—程序设计 Ⅳ. ① TP312.8

中国版本图书馆CIP数据核字(2018)第105850号

策　　划　秦志峰

责任编辑　祝婷婷　秦志峰

出版发行　西安电子科技大学出版社(西安市太白南路2号)

电　　话　(029)88242885　88201467　　邮　　编　710071

网　　址　www.xduph.com　　电子邮箱　xdupfxb001@163.com

经　　销　新华书店

印刷单位　陕西天意印务有限责任公司

版　　次　2018年6月第1版　2018年6月第1次印刷

开　　本　787毫米×1092毫米　1/16　印　张　21

字　　数　499千字

印　　数　1～2000册

定　　价　49.00元

ISBN 978-7-5606-4938-2/TP

XDUP　5240001-1

如有印装问题可调换

前　言

在20世纪90年代，面向对象技术以其显著的优势成为计算机软件领域的主流技术，随后该技术在大多数领域得到了广泛的应用。

面向对象方法与技术起源于面向对象的编程语言，但面向对象开发技术的焦点不仅仅是编程阶段，还应该包括面向对象的其他阶段，即面向对象的分析和设计阶段。

为了体现面向对象分析设计与编程的全过程，本书全面介绍了面向对象分析设计的方法与面向对象程序设计的基本理论和技术，并且运用实例加深读者对理论知识的掌握和学习。全书思路清晰，结构严谨，在内容的叙述上按照软件设计的流程，由分析设计到编程再到实例，循序渐进，用语规范，在结构上特别注重前后内容的连贯性，做到了抓住关键、突出重点，体现了“理论性、技术性、实用性”的特色。

本书共分为三篇11章。第一篇为面向对象分析与设计，包括第1章到第4章，主要介绍面向对象分析和设计的方法和统一建模语言UML；第二篇为面向对象程序设计，包括第5章到第10章，主要介绍面向对象编程语言的基本概念、方法和面向对象编程的机制和思想。第三篇为面向对象建模实例，包括第11章，主要以图书管理系统为例，详细阐述了使用统一建模语言进行分析和设计以及编程的全过程。

本书由王燕、庞淑侠、胡文瑾主编，其中第1~4章由王燕老师编写，第5~10章由庞淑侠老师编写，第11章由胡文瑾老师编写。本书的最终出版得到了许多老师和同学的帮助，在此一并表示由衷的感谢。

尽管作者在写作过程中投入了大量的时间和精力，但由于水平有限，书中不足之处在所难免，恳请广大读者批评指正。

编　者

2018年2月

目　录

第一篇　面向对象分析与设计

第二篇 面向对象程序设计

第三篇 面向对象建模实例

第一篇

面向对象分析与设计

第1章 面向对象方法概述

面向对象是软件工程学的一个重要分支，也是当今软件开发的主流方法。自 20 世纪 40 年代计算机问世以来，计算机在各个领域得到了广泛的应用，使得计算机技术蓬勃发展。然而，长期以来计算机软件开发的低效率制约了计算机软件行业的发展。计算机业界努力探索和研究解决软件危机的途径，并提出了软件工程的思想和方法，极大地提高了软件开发的效率和软件的质量。当前软件工程的发展正面临着从传统的结构化范型到面向对象范型的转变，这需要有新的语言、新的系统和新的方法的支持，面向对象技术就是这种新范型的核心技术，在这一核心技术的支持下，面向对象的分析和设计方法已逐渐取代传统的方法，成为当今计算机软件工程学的主流方法。

1.1 软件生命周期和过程模型

软件是计算机系统中与硬件相互依存的部分，它包括程序、相关数据及其说明文档。其中程序是按照事先设计的算法要求执行的指令序列，数据是程序能正常操作的信息，文档是与程序开发、维护和使用有关的各种图文资料。人们对软件的认识经历了一个由浅到深的过程，从追求编程技巧到崇尚清晰好用、易于修改和扩充，其过程实际上就是软件设计方法的发展历史。

计算机科学的不断发展，使得软件的需求量不断增大，它的要求、复杂度、开发成本也越来越高，但软件的开发方法和技术却还停留在“小程序”、“个体化”的操作上面，致使软件设计犹如泥潭，大批的设计者深陷其中，人们称其为软件危机。虽然，“软件危机”的概念在 1968 年 NATO(北大西洋公约组织)的计算机科学家在联邦德国召开的国家学术会议上才第一次被提出，但其实它在计算机产生的那一天起就出现了，它是计算机科学发展进程的必然产物，只不过到后来这种现象日渐严重，已经影响到计算机产业的发展，因此才引起各界的关注。

软件危机产生的原因主要有两个：一是与软件本身的特点有关，二是与软件开发和维护的方法不正确有关。通过分析软件危机的表现和原因，再加上不断地实践和总结，人们终于认识到：按照工程化的原则和方法组织软件开发工作，是摆脱软件危机的一个主要出

路，这就是软件工程的概念。

软件工程是指导计算机软件开发和维护的工程学科，它采用工程的概念、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技术和当前能够得到的技术方法结合起来。在经历了几十年的不懈努力后，软件工程的理论已得到极大的丰富和完善，各种软件设计方法层出不穷，软件行业一片繁荣，从而也促进了计算机科学不断向前发展。

1.1.1　软件生命周期

软件生命周期(Software Life Cycle)是指软件产品从考虑其概念开始，到该软件产品不再使用为止的整个时期，一般包括概念阶段、分析与设计阶段、构造阶段、移交阶段等不同时期。通常在整个软件生命周期中贯穿了软件工程过程的六个基本活动，具体介绍如下。

1．制定计划(Planning)

确定待开发软件系统的总目标，给出它的功能、性能、可靠性以及接口等方面的要求；由系统分析员和用户合作，完成该项软件系统任务的可行性研究，探讨解决问题的可能方案，并对可利用的资源(计算机硬件、软件、人力等)、成本、可取得的效益、开发的进度做出估计，制定出完成开发任务的实施计划，连同可行性研究报告，提交管理部门审查。

2．需求分析和定义(Requirement Analysis and Definition)

对待开发软件提出的需求进行分析并给出详细的定义。软件人员和用户共同讨论决定哪些需求是可以满足的，并对其加以确切的描述，然后编写出软件需求说明书或系统功能说明书，以及初步的系统用户手册，提交管理机构评审。

3．软件设计(Software Design)

设计是软件工程的技术核心。在设计阶段，设计人员把已确定了的各项需求转换成一个相应的体系结构。体系结构中的每一个组成部分都是意义明确的模块，每个模块都和某些需求相对应，即概要设计；进而对每个模块要完成的工作进行具体的描述，即进行详细设计，为源程序编写打下基础。所有设计中的考虑都应以设计说明书的形式加以描述，以供后续工作使用并提交评审。

4．程序编写(Coding Programming)

把软件设计转换成计算机可以接受的程序代码，即写成以某一种特定程序设计语言表示的“源程序清单”，即程序编写。这一步工作也称为编码。自然，写出的程序应当是结构良好、清晰易读的，且与设计说明书相一致。

5．软件测试(Testing)

测试是保证软件质量的重要手段，其主要方式是在设计测试用例的基础上检验软件的各个组成部分。首先是进行单元测试，查找各模块在功能和结构上存在的问题并加以纠正；其次是进行组装测试，将已测试过的模块按一定顺序组装起来；最后按规定的各项需求，逐项进行有效性测试(或称确认测试)，确定已开发的软件是否合格，能否交付用户使用。

6．运行/维护(Running/Maintenance)

交付用户的软件投入正式使用，即进入运行/维护阶段。这一阶段持续到用户不再使用该软件为止。软件在运行中可能由于多方面的原因需要进行修改，原因可能有：运行中软

件出现错误需要修正；为了适应软件运行环境的变化需做适当变更；为了增强软件的功能需做变更。

1.1.2 软件过程模型

模型是实际事物、实际系统的抽象。它是针对所需要了解和解决的问题，抽取其主要因素和主要矛盾，忽略一些不影响基本性质的次要因素，形成的对实际系统的表示方法。模型的表示形式可以多种多样，可以是数学表达式、物理模型或图形文字描述等。总之，只要能回答所需研究问题的实际事物或系统的抽象表达式，都可以称为模型。由于模型省略了一些不必要的细节，所以对模型操作要比对原始系统操作更加容易。

软件过程模型是从一个特定角度提出的对软件过程的简化描述，是对软件开发实际过程的抽象，它包括构成软件过程的各种活动、软件工件(Artifact)以及参与角色等。从软件过程的三个组成成分可以将软件过程模型划分为以下三种类型。

(1) 工作流(Workflow)模型。工作流模型描述软件过程中各种活动的序列、输入和输出，以及各种活动之间的相互依赖性。它强调软件过程中活动的组织控制策略。

(2) 数据流(Dataflow)模型。数据流模型描述将软件需求变换成软件产品的整个过程中的活动，这些活动完成将输入工件变换成输出工件的功能。它强调软件过程中的工件的变换关系，对工件变换的具体实现措施并未加以限定。

(3) 角色/动作模型。角色/动作模型描述了参与软件过程的不同角色及其各自负责完成的动作，即根据参与角色的不同将软件过程应该完成的任务划分成不同的职能域(Function area)。它强调软件过程中角色的划分、角色之间的协作关系，并对角色的职责进行了具体的确定。

软件过程模型有时也称软件生命周期模型，即描述从软件需求定义开始直至软件使用后废弃为止，跨越整个生存期的软件开发、运行和维护所实施的全部过程、活动和任务的结构框架，同时描述生命周期不同阶段产生的软件工件，明确活动的执行角色等。

几十年来，软件生命周期模型得到了很大的发展，提出了一系列具体的模型以适应软件开发的需要，其中既有以瀑布模型为代表的传统软件生命周期模型，又有以敏捷建模为代表的新型软件生命周期模型。下面介绍几个经典的软件开发模型。

1. 瀑布模型

在软件开发早期，开发被简单地分成编写程序代码和修改程序代码两个阶段。在拿到项目后就根据需求开始编写程序，经调试通过后就拿给用户使用，这样项目就算结束了，而当应用中出现错误或者有新的要求时，则重新修改代码。这种小作坊式的软件开发方式有着明显的弊端，如缺乏统一的项目规划、不太重视需求的获取和分析、对软件的测试和维护考虑不周等，这些都会导致软件项目的失败，这种情况在软件项目规模增大时表现得特别明显。

汲取早期软件开发的教训，人们开始按照工程的管理方式将软件开发过程划分成不同的阶段，即制定计划、需求分析和定义、软件设计、程序编写、软件测试、运行/维护等。1970 年，W. Royce 在软件生命周期概念的基础上，提出了著名的“瀑布模型(Waterfall Model)”，它是最早出现的软件开发模型，直到 20 世纪 80 年代早期一直是唯一被广泛采用

的软件开发模型。顾名思义，瀑布模型就是将软件开发过程中的各项活动规定为按固定顺序连接的若干阶段工作，换句话说，它将软件开发过程划分为若干个互相区别又彼此联系的阶段，每个阶段中的工作都以上一个阶段工作的结果为依据，同时为下一个阶段的工作提供了前提。

简单地说，瀑布模型规定了软件按生命周期提出的六个基本工程活动，并且规定了它们自上而下、相互衔接的固定次序，如瀑布流水逐级下落，如图 1-1 所示。

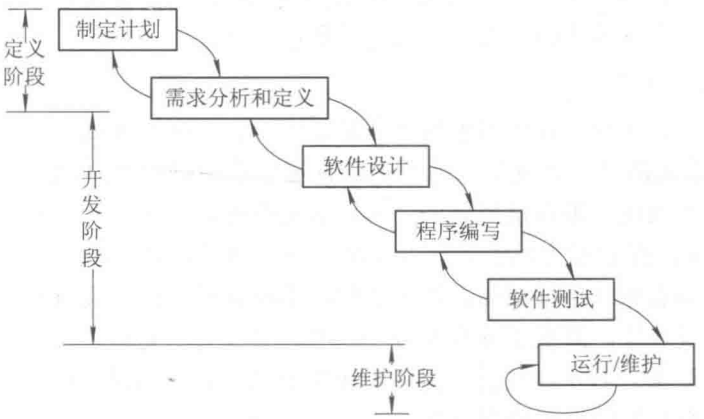

图 1-1　瀑布模型

1) 瀑布模型的特征

瀑布模型中的每一个开发活动都具有下列特征：

(1) 本活动的工作对象来自于上一项活动的输出，这些输出一般是代表本阶段活动结束的里程碑式的文档。一个阶段的输出文档能够成为该阶段的里程碑，必须经过严格的文档评审，以保证该阶段软件工件的质量。上一活动的阶段性文档输出到本阶段活动之前，必须进行“冻结”，防止文档的随意改动而造成对随后活动执行的影响。

(2) 根据本阶段的活动规程执行相应的任务。

(3) 产生本阶段活动的相关产出——软件工件，作为下一活动的输入。

(4) 对本阶段活动执行情况进行评审。如果活动执行得到确认，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。由于需要重新更改返工阶段在此前活动的产出文档，因此，需要对之前“冻结”的某些文档执行“解冻”操作，以便返工时进行修改。

2) 瀑布模型的优点

我国曾在 1988 年依据该开发模型制定并公布了“软件开发规范”国家标准，这对我国软件开发起到了较大的促进作用。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，它在软件开发早期为消除非结构化软件、降低软件复杂度、促进软件开发工程化起到了显著的作用，其优点体现在以下几点：

(1) 软件生命周期的阶段划分不仅降低了软件开发的复杂程度，而且提高了软件开发过程的透明性，便于将软件工程过程和软件管理过程有机地融合在一起，从而提高软件开发过程的可管理性。

(2) 推迟了软件实现，强调在软件实现前必须进行分析和设计工作。早期的软件开发

人员，或者没有软件工程实践经验的软件开发人员，接手软件项目时往往急于编写代码，缺乏分析与设计的基础性工作，最后导致代码被频繁、重复地改动，代码结构变得不清晰，甚至是混乱，不仅降低了工作效率，而且直接影响到软件的质量。

(3) 瀑布模型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，保证了阶段之间的正确衔接，能够及时发现并纠正开发过程中存在的缺陷，从而能够使产品达到预期的质量要求。由于是通过文档控制软件开发阶段进度的，所以，在正常情况下也能够保证软件产品的及时交付。当然，频繁出现的缺陷，特别是前期存在但潜伏到后期才被发现的缺陷，会导致不断的返工，从而导致进度拖延。

3) 瀑布模型的缺点

瀑布模型在软件工程实践应用中也逐渐暴露出一些严重的缺点：

(1) 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，这是瀑布模型最突出的缺点。因此，瀑布模型只适合于需求明确的软件项目。

(2) 模型的风险控制能力较弱。一方面是指只有当软件通过测试后才能可见、用户无法在开发过程中间看到软件半成品，增加了用户不满意的风险；而且软件开发人员只有到后期才能看到开发成果，降低了开发人员的信心。另一方面是指软件体系结构级别的风险只有在整体组装测试之后才能发现，同样，前期潜伏下来的错误也只能在固定的测试阶段才能被发现，这个时候的返工极有可能导致项目延期。

(3) 瀑布模型中的软件活动是文档驱动的，当各个阶段规定了过多的文档时，会极大地增加系统的工作量；而且当管理人员以文档的完成情况来评估项目完成进度时，往往会产生错误的结论，因为后期测试阶段发现的问题会导致返工，前期完成的文档只不过是一个未经返工修改的初稿而已，而一个瀑布模型应用不需返工的项目是很少见的。

随着软件项目规模和复杂性的不断扩大，项目需求的不稳定性变成司空见惯的情形，瀑布模型的上述缺点变得越来越严重，为了弥补瀑布模型的不足，后期又提出了多种其他的生命周期模型。

2. V 模型和 W 模型

瀑布模型将测试作为软件实现之后的一个独立阶段，使得在分析和设计阶段潜伏下来的错误得到纠正的时间大为推迟，造成较大的返工成本；而且体系结构级别的缺陷也只是在测试阶段才能被发现，使得瀑布模型驾驭风险的能力较低。

针对瀑布模型这个缺点，20 世纪 80 年代后期 Paul Rook 提出了 V 模型，如图 1-2 所示。

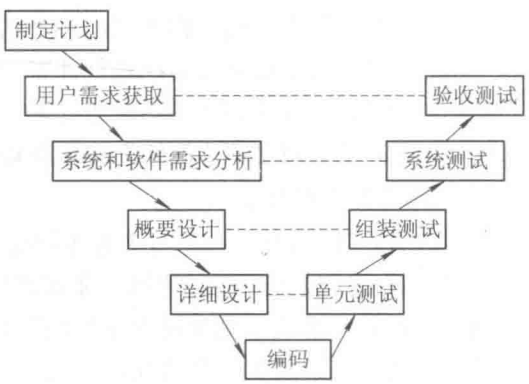

图 1-2　V 模型示意图

从图 1-2 中可以看出，V 模型的左半部分就是在测试阶段之前的瀑布模型，即 V 模型的开发阶段，右半部分是测试阶段。V 模型明确地划分了测试的级别，并将其与开发阶段的活动对应起来。

V 模型各测试部分的作用如下：

(1) 单元测试(Unit Test)。用来发现编码过程中可能存在的各种错误，并且验证编码是否和详细设计的要求相一致。单元测试与开发阶段的详细设计相对应。

(2) 组装测试(Integration Test)。用来检查软件各组成单元之间的接口是否存在缺陷，同时检查每个组成单元完成的功能是否和接口一致。组装测试与开发阶段的概要设计相对应。

(3) 系统测试(System Test)。检查系统功能、性能的质量特性是否达到系统要求的指标，同时检查软件系统和外围系统之间的接口是否存在缺陷。系统测试与开发阶段的系统需求分析相对应。

(4) 验收测试(Acceptance Test)。确认计算机系统是否满足用户需求或合同要求，它主要与开发阶段的用户需求获取相对应。

由于 V 模型只是在测试阶段对瀑布模型进行了改动，所以在美国很难被接受，但是在欧洲，特别是在英国却得到了广泛的认同和理解。V 模型虽然强调了测试阶段的重要性(对测试进行分级，并与开发阶段相对应)，但它却继承了瀑布模型的缺点，即将测试作为一个独立的阶段，所以并没有提高模型抵抗风险的能力。为了尽早发现分析与设计的缺陷，必须将测试广义化，即扩充确认(Validation)和验证(Verification)内容，并将广义的测试作为一个过程贯穿整个软件生命周期。基于这个出发点，Evolutif 公司在 V 模型的基础上提出了 W 模型，如图 1-3 所示。

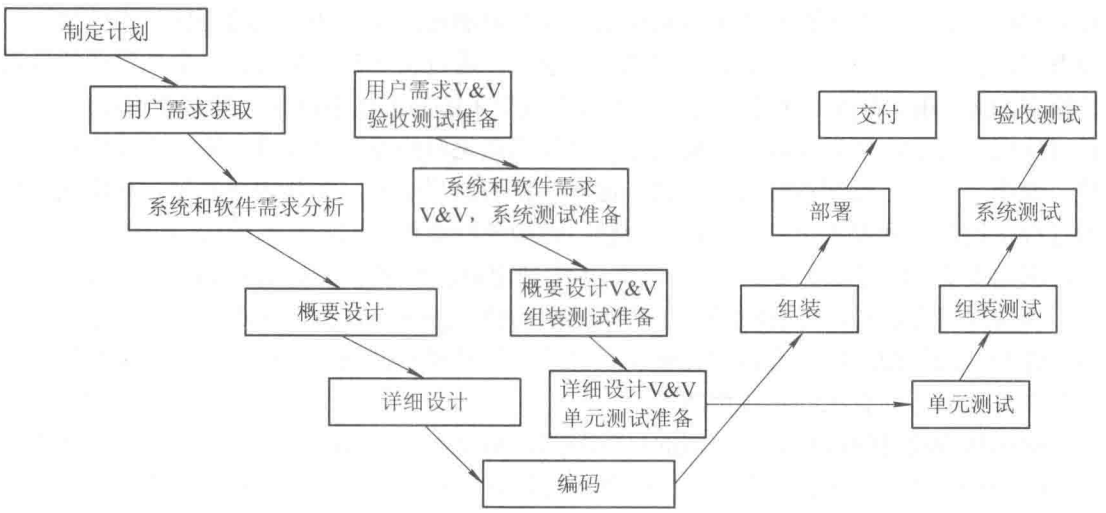

图 1-3　W 模型示意图

W 模型由两个 V 模型组成，分别代表测试与开发过程。图 1-3 明确表示出了测试与开发的并行关系。

W 模型强调，测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求和设计同样需要测试，也就是说，测试与开发是同步进行的。由于 W 模型扩展了测试的内容，增加了确认和验证活动，所以它有利于尽早地全面发现问题。例如，需求分析完成后，测试人员就应该参与到对需求的确认和验证活动中，以尽早找出分析中的缺陷；同时，对需求的测试也有利于及时了解项目的难度和测试风险，及早制定应对措施，这也会减少总体

测试时间，加快项目进度。W 模型也存在局限性，它并没有改变瀑布模型中需求、设计和编码等活动的串行关系，同时测试和开发活动也保持着一种线性的前后关系，上一阶段完全结束，才可正式开始下一阶段的工作，因此，W 模型仍然只适合于需求比较稳定的软件项目。

3. 原型方法

瀑布模型以阶段划分评审、文档控制等手段来保障软件项目的进度和质量，整个过程是文档驱动的，即上一个阶段没有完成不能进入下一个阶段，因此，当用户需求获取困难，很难一次性准确进行需求分析的时候，软件项目将无法进入到设计阶段，工期也会遥遥无期。尽管在瀑布模型及随后的 V 模型和 W 模型中通过加强评审和测试能够缓解上述问题，但仍然没有从根本上解决问题。

为了解决这些问题，逐渐形成了软件系统的原型建设思想。

原型通常是指模拟某种最终产品的原始模型，它在工程领域中被广泛应用。例如，一座大桥在开工建设之前需要建立很多原型，如风洞实验原型、抗震实验原型等，以检验大桥设计方案的可行性。在软件开发过程中，原型是软件的一个早期可运行的版本，它反映了最终系统的部分重要特性。

由于现实世界是不断变化的，而且变化的速度越来越快，因此，软件开发在需求获取、技术实现手段选择、应用环境适应等方面都出现了前所未有的困难，特别是对变化需求的控制和技术实现尤为突出。为了应对早期需求获取困难以及后期需求的变化，人们采取了原型方法构造软件系统。当获得一组基本需求后，通过快速分析构造出一个小型的软件系统原型，满足用户的基本要求。用户可在试用原型系统的过程中得到亲身感受和受到启发，做出反应和评价，然后开发人员根据用户的反馈意见对原型加以改进。随着不断构造、交付、使用、评价、反馈和修改，一轮一轮产生新的原型版本，如此周而复始，逐步减少分析过程中用户和开发人员之间的沟通误解，逐步使原本模糊的各种需求细节清晰起来。对于需求的变更，也可以在变更后的原型版本中做出适应性调整，从而提高最终产品的质量。

软件原型方法是在分析阶段为了明确需求而研究的方法和技术中产生的，但它也可面向软件开发的其他阶段，比如用在概要设计阶段以选择不同的软件体系结构，用在详细设计阶段以试验不同算法的实现性能等。

根据软件项目特点和运行原型的目的不同，原型主要分为以下三种不同的作用类型：

(1) 探索型。这种原型的目的是要弄清用户对目标系统的要求，确定所期望的特性，并探讨多种技术实现方案的可行性。它主要针对需求模糊、用户和开发者对项目开发都缺乏经验的情况。

(2) 实验型。这种原型用于大规模开发和实现之前，考核技术实现方案是否合适、分析和设计的规格说明是否可靠。

(3) 进化型。这种原型的目的是在构造系统的过程中能够适应需求的变化，通过不断地改进原型，逐步将原型进化成最终的系统。它将原型方法的思想扩展到软件开发的全过程，适用于需求变动的软件项目。

根据运用原型的目的和方式不同，在使用原型时可采取以下两种不同的策略：

(1) 废弃策略。先构造一个功能简单而且性能要求不高的原型系统，针对用户使用这

个原型系统后的评价和反馈，反复进行分析和改进，形成比较好的设计思想，据此设计出较完整、准确、一致、可靠的最终系统。系统构造完成后，原来的原型系统就被废弃不用。探索型和实验型原型属于这种策略。

(2) 追加策略。先构造一个功能简单而且性能要求不高的原型系统，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。它对应于进化型原型。

具体采用什么形式、什么策略的原型主要取决于软件项目的特点和开发者的素质，以及支持原型开发的工具和技术，要根据实际情况的特点加以决策。

在实际中，应用原型方法进行系统的分析和构造将带来以下好处：

(1) 原型方法有助于增进软件人员和用户对系统服务需求的理解，使比较含糊的具有不确定性的软件需求(主要是功能)明确化。对于系统用户来说，要他们想象最终系统是什么样的，或者描述当前有什么要求，都是很困难的。但是对他们来说，评价一个系统的原型要比写出规格说明容易很多。当用户看到原型的执行不是他们原来所想象的那样时，原型化方法允许并鼓励他们改变原来的要求。由于这种方法能在早期就明确用户的要求，因此可防止以后由于不能满足用户要求而造成的返工，从而避免了不必要的经济损失，缩短了开发周期。

(2) 原型方法提供了一种有力的学习手段。通过原型演示，用户可以亲身体验早期的开发过程，获得关于计算机和所开发系统的专门知识，对使用者培训有积极作用。软件开发者也可以获得用户对系统的确切要求，学习到应用领域的专业知识，使开发工作做得更好。

(3) 使用原型方法，可以容易地确定系统的性能，确认各项主要系统服务的可应用性，确认系统设计的可行性，确认系统作为产品的结果。因此，它可以作为理解和确认软件需求规格说明的工具。

(4) 软件原型的最终版本，有的可以原封不动地成为产品，有的略加修改就可以成为最终系统的一个组成部分，这样有利于建成最终系统。

当然，原型法也有一定的适用范围和局限性，主要表现在以下几个方面：

(1) 对于一个大型系统，如果不经过系统分析，则得到系统的整体划分模拟系统部件是很困难的。

(2) 对于大量运算的、逻辑性较强的程序模块，原型方法很难构造出该模块的原型来供人评价。

(3) 对于原有应用的业务流程、信息流程混乱的情况，原型构造与使用有一定的困难，即使构造出来了，也是对旧系统的模拟，很难达到新系统的目标。这种情况下需要先对业务流程和信息流程进行再造(Reengineering)，然后再考虑新系统的需求目标，利用原型逐步演进系统。

(4) 对于一个批处理系统，由于大部分活动是由内部处理的，因此应用原则方法会有一定的困难。

由于原型方法的应用具有一定的局限性，所以要根据软件项目的特点和应用原型的目的选择不同类型的原型方法。

1984 年 Boar 提出一系列影响原型方法选择的因素。如果是在需求分析阶段要使用原

型方法，则必须从系统结构、逻辑结构、用户特征、应用约束、项目管理和项目环境等多方面来考虑，以决定是否采用原型化方法。

(1) 系统结构：联机事务处理系统、相互关联的应用系统适合于应用原型方法，而批处理系统不适宜应用原型方法。

(2) 逻辑结构：有结构的系统(如运行支持系统、管理信息系统等)适合于应用原型方法，而基于大量算法和逻辑结构的系统不适宜应用原型方法。

(3) 用户特征：不满足于预先做系统定义说明，愿意为定义和修改原型投资，难于明确详细需求，愿意承担决策的责任，准备积极参与的用户是适合于使用原型的用户。

(4) 应用约束：对在线运行系统的补充，不能用原型方法。

(5) 项目管理：只有项目负责人愿意使用原型方法，才适合用原型的方法。

(6) 项目环境：需求说明技术应当根据每个项目的实际环境来选择。

当系统规模很大、要求复杂、系统服务不清晰时，在需求分析阶段先开发一个系统原型是很值得的。特别是当性能要求比较高时，在系统原型上先做一些试验也是很必要的。

4. 演化模型

演化模型(Evolutional Model)有时也译做进化模型，它是基于软件开发人员在应用瀑布模型的软件工程实践中体会出来的一种认识：在项目开发的初始阶段人们对软件需求的认识常常不够清晰，使得项目难于做到一次开发成功，出现返工在所难免。有人说，往往要“干两次”后开发出的软件才能较好地令用户满意。第一次只是试验开发，得到试验性的模型产品，其目标只是在于探索可行性，弄清软件需求；第二次则在此基础上获得较为满意的软件产品。

演化模型主要是针对需求不是很明确的软件项目，希望通过原型来逐步探索和理解用户需求。

按照原型应用策略的不同，演化模型也可以分为两类：

(1) 探索式演化模型。探索式演化模型的目标是与用户一起工作，共同探索系统需求，直到最后交付系统。这类开发是从需求较清楚的部分开始，根据用户的建议逐渐向系统中添加功能的。

探索式的演化模型也是“演化”本身的含义，它不强调按照瀑布模型那样严格划分阶段界限，即上一阶段没有结束不能进入下一阶段，而是针对部分明确的需求可以进行瀑布模型的过程活动，建立一个系统原型，对不明确的需求，希望用户在已经建立起来的原型基础上进行评价和反馈，逐步明晰需求。因此，最终的系统是在探索需求的原型上一步一步添加功能完成的。

(2) 抛弃式演化模型。抛弃式演化模型的目标是理解用户需求，然后给系统一个较好的需求定义。建立原型是为了帮助客户进一步明确原本含糊不清的需求，帮助开发人员理解客户需求的真实含义，帮助澄清客户和开发人员之间的沟通误解，从而得到一个正确、完整和一致的需求规格说明。这个时候的原型并不涉及系统核心功能的开发，更多的只是界面的模拟或者功能菜单的描述，甚至是拿同类产品运行作为原型演示。

相对于瀑布模型而言，演化模型的一个明显的好处就是可以处理需求不明确的软件项目，对于探索式的演化模型，能够在开发过程中逐步向用户展示软件“半成品”，降低系统