高等院校信息类专业校企衔接创新实践系列教材

Java 项目实战

企业案例分析与项目自主设计

□ 赵明 单洪 著

中南大学出版社
www.csupress.com.cn

Java 项目实战：企业案例分析与项目自主设计

赵 明 单 洪 著

中南大學出版社
www.csupress.com.cn

图书在版编目(CIP)数据

Java 项目实战:企业案例分析与项目自主设计/赵明,单洪著.
—长沙:中南大学出版社,2016.12
ISBN 978-7-5487-2291-5

Ⅰ.J... Ⅱ.①赵...②单... Ⅲ.JAVA 语言-程序设计
Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2016)第 127811 号

Java 项目实战——企业案例分析与项目自主设计

XIANGMU SHIZHAN——QIYE ANLI FENXI YU XIANGMU ZIZHU SHEJI

赵 明 单 洪 著

□责任编辑 谢贵良
□责任印制 易建国
□出版发行 中南大学出版社
社址:长沙市麓山南路 邮编:410083
发行科电话:0731-88876770 传真:0731-88710482
□印 装 长沙市宏发印刷有限公司

□开 本 787×1092 1/16 □印张 20 □字数 496 千字
□版 次 2016 年 12 月第 1 版 □印次 2016 年 12 月第 1 次印刷
□书 号 ISBN 978-7-5487-2291-5
□定 价 49.00 元

前 言

Spring 是一个开源框架，最早由 Rod Johnson 创建，并在《Expert One - on - One: J2EE Design and Development》这本著作中进行了介绍。Spring 是为了解决企业级应用开发的复杂性而创建的。使用 Spring 可以让简单的 JavaBean 实现之前只有 EJB 才能完成的事情。但 Spring 不仅仅局限于服务器端开发，任何 Java 应用都能在简单性、可测试性和松耦合等方面从 Spring 中获益。

Spring 可以做很多事情，但归根结底，支持 Spring 的仅仅是少许的基本概念。所有的理念都可以追溯到 Spring 最根本的使命上：简化 Java 开发。

虽然本书不会对 Spring 做过多的分析和介绍，但还是希望读者在学习过程中能够时时记住 Spring 框架的使命。因为在笔者与多位新手交流后发现，许多刚接触 Spring 框架的初学者都认为 Spring 很“难用”，配置“麻烦”。实际上，这是初学者刚接触 Spring 的 J2EE 开发时的一个错觉，造成这种错觉的原因大概有以下几种：

- 一次性接触的知识太多：很多新手都是在匆匆学习了 Java 的基础之后，就一头杀入了众多框架(Spring、Struts、Hibernate、Mybatis 等)学习中，试图开发出一个 J2EE 的应用来。这时候不光是接触到大量的框架，而且同时还会接触众多其他辅助知识点，如 HTML、JavaScript、CSS、JSP 等。众多新知识交织在一起，让初学者无所适从，错误频出。因为在这些知识点中，Spring 处于核心位置，是初学者无论如何都无法绕开的“坎”，因此很多初学者将碰到的问题都“归结”于 Spring 很“难用”了。
- 出现的错误难以调试：因为刚开始使用框架首先都是进行大量的配置，这个过程中经常会出现一些意料不到的问题(如大小写问题、拼写错误等)，这类问题如果没有较为丰富的经验，会让调试的过程变得异常痛苦。很多时候，初学者会将这类问题归结于框架(大多数是 Spring)很“难用”。

基于上述原因，笔者衷心希望读者在学习的过程中，不要对 Spring 失去信心。Spring 的简单之美会在痛苦的煎熬之后“突然”来临，读者要做的就是在不断的学习和思考中等待这一“幸福时刻”到来就好。

之所以花这么长的篇幅来说明 Spring 框架，是因为本书是一本介绍 Spring MVC 开发的书，而 Spring 又是此类开发的核心。其实，Java 本身并不难学，但是一旦进入到 J2EE 的开发，学习曲线陡然升高。这时如果没有外界的帮助和内心的执着，很容易中途放弃学习。这也是现在各种教学视频大受欢迎的重要原因，因为跟随教程，读者有很大可能能够真正将一个项目运行起来。但是依笔者的观点，视频教程仅仅只是让初学者能够“敲”出一个能运行的网站而已，一旦稍作变化，读者立马再次进入“煎熬模式”。而书本不一样，书本能够详细的讲述背后的原理和规则，而且便于查找。希望笔者在前面的说明，多少能够增强读者内心的一丝丝执着之念头。

和其他同类介绍 Spring MVC 的书不一样，本书最大的特色就是所有代码都来自于一个真实的项目，而且笔者在写书时并未对原有项目做任何修改，包括程序员编写的有瑕疵的代码(一些敏感的信息做了处理)。优雅的代码有之，丑陋的代码也依然存在，希望这种不完美，能够让读者提前感受到软件开发的真实。

本书在代码开发的组织上的编排，也是本书和其他书的不同之一。本书在描述开发过程时基本上是按照中小项目的开发顺序来组织的。即程序员是以“垂直”的方式，从模型设计，依次开发持久层、服务层、控制层和视图层，这样，开发完一个功能之后，就能马上对此功能进行完整流程的测试，更利于初学者排查问题(当然，更合理的做法是写完一部分代码，就对这段代码进行单元测试，但初学者一般要做到这一点比较困难，与其邯郸学步，不如直奔主题)。

当然，受限于笔者的水平，本书的瑕疵肯定存在，正如软件中永远无法根除的 Bug 一样。希望读者在碰到此类问题后，不要因此对本书的其他地方，甚至对笔者产生怀疑，也希望，不要影响读者感受到编程的快乐，更不要影响大家对于开发的追求。

祝大家学习愉快!!!

编　者

2016 年 7 月

目 录

第1章　项目开发方法概述

软件项目的开发，并不是将人力和代码简单地堆砌就能成功的，软件发展的历史上因此造成的损失已经不计其数。虽然本书主要是介绍一个软件项目的设计和开发整过程，但在设计和开发的过程中，作者还是希望学员能够有一些软件工程的基本概念和理解，使其在软件项目的开发过程能够更加顺利。因此，本书特地在开篇第一章就对软件工程的一些基本概念和方法进行了简单的介绍，希望学员能够在软件开发的过程上少走弯路。

1.1　软件工程概述

人类社会已经跨入了21世纪，计算机系统已经渗入到人类生活的各个领域，同时计算机软件已经发展成为当今世界最重要的技术领域。随着当今社会的发展，软件的规模和复杂度已经远远超出了个人能够完成的程度，精英单干已经被团队协作所代替，团队内部互相协作以及团队之间协同开发日益重要，研究此类问题则产生了一门重要的学科，即是软件工程。软件工程的研究领域包括软件的开发方法、软件的生命周期以及软件的工程实践等。

1.1.1　软件危机

20世纪60年代中期以前，是计算机系统发展的早期。软件是为每个项目的具体应用而专门编写的，实际上就是规模较小的程序，程序的编写者和使用者往往是同一个(或同一组)人。由于规模小，也没有什么系统化的方法，对软件开发工作更没有进行任何管理，使得软件设计往往只是在人们头脑中隐含进行的一个模糊过程，除了程序清单之外，根本没有其他文档资料保存下来。

从20世纪60年代中期到70年代中期，出现了“软件作坊”，产品软件得到广泛使用，软件数量急剧膨胀，出现的问题也越来越多。在程序运行时发现的错误必须设法改正；用户有了新的需求时必须相应地修改程序；硬件或操作系统更新时，通常需要修改程序以适应新的环境。上述种种软件维护工作，以令人吃惊的比例耗费资源。更严重的是，许多程序的个体化特性使得它们最终成为不可维护的。“软件危机”就这样开始出现了。

1.1.1.1　软件危机的表现

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。这些问题绝不仅仅是不能正常运行的软件才具有的。实际上，几乎所有软件都不同程度地存在这些问题。概括地说，软件危机包含下述两方面的问题：如何开发软件，以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。鉴于软件危机的长期性和症状不明显的特征，有人建议把软件危机更名为“软件萧条(depression)”或“软件困扰(affliction)”。不过“软件危

机”这个词强调了问题的严重性，而且也已为绝大多数软件工作者所熟悉，所以本书仍将沿用它。

具体来说，软件危机主要有以下一些典型表现。

1）对软件开发成本和进度的估计常常很不准确。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的现象并不罕见。这种现象降低了软件开发组织的信誉。而为了赶进度和节约成本所采取的一些权宜之计又往往损害了软件产品的质量，从而不可避免地会引起用户的不满。

2）用户对“已完成的”软件系统不满意的现象经常发生。软件开发人员常常在对用户要求只有模糊的了解，甚至对所要解决的问题还没有确切认识的情况下，就仓促上阵匆忙着手编写程序。软件开发人员和用户之间的信息交流往往很不充分，“闭门造车”必然导致最终的产品不符合用户的实际需要。

3）软件产品的质量往往靠不住。软件可靠性和质量保证的确切定量概念刚刚出现不久，软件质量保证技术还没有坚持不懈地应用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）软件常常是不可维护的。很多程序中的错误是非常难改正的，实际上不可能使这些程序适应新的硬件环境，也不能根据用户的需要在原有程序中增加一些新的功能。“可重用的软件”还是一个没有完全做到的、正在努力追求的目标，人们仍然在重复开发类似的或基本类似的软件。

5）软件通常没有适当的文档资料。计算机软件不仅仅是程序，还应该有一整套文档资料。这些文档资料应该是在软件开发过程中产生出来的，而且应该是“最新式的”（即和程序代码完全一致的）。软件开发组织的管理人员可以使用这些文档资料来管理和评价软件开发工程的进展状况；软件开发人员可以利用它们作为通信工具，在软件开发过程中准确地交流信息；对于软件维护人员而言，这些文档资料更是至关重要、必不可少的。缺乏必要的文档资料或者文档资料不合格，必然给软件开发和维护带来许多严重的困难和问题。

6）软件成本在计算机系统总成本中所占的比例逐年上升。由于微电子学技术的进步和生产自动化程度不断提高，硬件成本逐年下降，然而软件开发需要大量人力，软件成本随着通货膨胀以及软件规模和数量的不断扩大而持续上升。美国在1985年软件成本大约已占计算机系统总成本的90%。

7）软件开发生产率提高的速度，既跟不上硬件的发展速度，也远远跟不上计算机应用迅速普及深入的趋势。软件产品“供不应求”的现象，使人类不能充分利用现代计算机硬件提供的巨大潜力。

以上列举的仅仅是软件危机的一些明显的表现，与软件开发和维护有关的问题远远不止这些。

1.1.1.2 产生软件危机的原因

在软件开发和维护的过程中存在这么多严重问题，一方面与软件本身的特点有关，另一方面也和软件开发与维护的方法不正确有关。究其原因，大概有以下几个原因：

1）评估难：软件不同于硬件，它是计算机系统中的逻辑部件而不是物理部件。由于软件缺乏“可见性”，在写出程序代码并在计算机上试运行之前，软件开发过程的进展情况较难衡量，软件的质量也较难评价，因此，管理和控制软件开发过程相当困难。

2)规模大：软件不同于一般程序，它的一个显著特点是规模庞大，而且程序复杂性将随着程序规模的增加而呈指数上升。为了在预定时间内开发出规模庞大的软件，必须由许多人分工合作。然而，如何保证每个人完成的工作合在一起确实能构成一个高质量的大型软件系统，更是一个极端复杂困难的问题，不仅涉及许多技术问题，诸如分析方法、设计方法、形式说明方法、版本控制等，更重要的是必须有严格而科学的管理。

3)忽视需求：事实上，对用户要求没有完整准确的认识就匆忙着手编写程序是许多软件开发工程失败的主要原因之一。只有用户才真正了解他们自己的需要，但是许多用户在开始时并不能准确具体地叙述他们的需要，软件开发人员需要做大量深入细致的调查研究工作，反复多次地和用户交流信息，才能真正全面、准确、具体地了解用户的要求。对问题和目标的正确认识是解决任何问题的前提和出发点，软件开发同样也不例外。急于求成，仓促上阵，对用户要求没有正确认识就匆忙着手编写程序，这就如同不打好地基就盖高楼一样，最终必然垮台。事实上，越早开始编写程序，完成它所需要用的时间往往越长。

4)重编码：一个软件从定义、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的时期称为生命周期。软件开发最初的工作应是问题定义，也就是确定要求解决的问题是什么；然后要进行可行性研究，决定该问题是否存在一个可行的解决办法；接下来应该进行需求分析，也就是深入具体地了解用户的要求，在所要开发的系统(不妨称之为目标系统)必须做什么这个问题上和用户取得完全一致的看法。经过上述软件定义时期的准备工作才能进入开发时期，而在开发时期首先需要对软件进行设计(通常又分为概要设计和详细设计两个阶段)，然后才能进入编写程序的阶段，程序编写完之后还必须经过大量的测试工作(需要的工作量通常占软件开发全部工作量的40% ~50%)才能最终交付使用。所以，编写程序只是软件开发过程中的一个阶段，而且在典型的软件开发工程中，编写程序所需的工作量只占软件开发全部工作量的10% ~20%。

5)轻维护：许多软件产品的使用寿命长达10年甚至20年，在这样漫长的时期中不仅必须改正使用过程中发现的每一个潜伏的错误，而且当环境变化时(如硬件或系统软件更新换代)还必须相应地修改软件以适应新的环境，特别是必须经常改进或扩充原来的软件以满足用户不断变化的需要。所有这些改动都属于维护工作，而且是在软件已经完成之后进行的，因此，维护是极端艰巨复杂的工作，需要花费很大代价。统计数据表明，实际上用于软件维护的费用占软件总费用的55% ~70%。软件工程学的一个重要目标就是提高软件的可维护性，减少软件维护的代价。

6)测试介入晚：在软件开发的不同阶段进行修改需要付出的代价是很不相同的。在早期引入变动，涉及的面较少，因而代价也比较低；在开发的中期，软件配置的许多成分已经完成，引入一个变动要对所有已完成的配置成分都做相应的修改，不仅工作量大，而且逻辑上也更复杂，因此付出的代价剧增；在软件“已经完成”时再引入变动，当然需要付出更高的代价。根据美国一些软件公司的统计资料，在后期引入一个变动比在早期引入相同变动所需付出的代价高2 ~3个数量级。

7)缺文档：程序只是完整的软件产品的一个组成部分，一个软件产品必须由一个完整的配置组成，软件配置主要包括程序、文档、数据等成分。必须清除只重视程序而忽视软件配置其余成分的糊涂观念。

了解产生软件危机的原因，澄清错误认识，建立起关于软件开发和维护的正确概念，还

仅仅是解决软件危机的开始，全面解决软件危机需要一系列综合措施。概念，还仅仅是解决软件危机的开始，全面解决软件危机需要一系列综合措施。

1.1.1.3 消除软件危机的途径

为了消除软件危机，首先应该对计算机软件有一个正确的认识。软件是程序、数据及相关文档的完整集合。其中，程序是能够完成预定功能和性能的可执行的指令序列；数据是使程序能够适当地处理信息的数据结构；文档是开发、使用和维护程序所需要的图文资料。1983 年 IEEE(电气和电子工程师协会)为软件下的定义是：计算机程序、方法、规则、相关的文档资料以及在计算机上运行程序时所必需的数据。虽然表面上看来在这个定义中列出了软件的 5 个配置成分，但是，方法和规则通常是在文档中说明并在程序中实现的。

更重要的是，必须充分认识到软件开发不是某种个体劳动的神秘技巧，而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。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，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。

应该推广和使用在实践中总结出来的开发软件成功的技术和方法，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。

应该开发和使用更好的软件工具。正如机械工具可以“放大”人类的体力一样，软件工具可以“放大”人类的智力。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。

总之，为了消除软件危机，既要有技术措施(方法和工具)，又要有必要的组织管理措施。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。

1.1.2 软件工程

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。在这次会议上正式提出并使用了“软件工程”这个名词，一门新兴的工程学科就此诞生了。

1.1.2.1 什么是软件工程

概括地说，软件工程是指导计算机软件开发和维护的工程学科。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最好的技术方法结合起来，经济地开发出高质量的软件并有效地维护它，这就是软件工程。

1993 年 IEEE 给出了一个比较全面的定义。

软件工程是：①把系统化的、规范的、可度量的途径应用于软件开发、运行和维护的过程，也就是把工程化应用于软件中；②研究①中提到的途径。

1.1.2.2 软件工程的基本原理

自从 1968 年在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”。著名的软件工程专家 Barry W. Boehm 综合这些学者们的意见并总结了 TRW 公司多年开发软件的经

验，于 1983 年在一篇论文中提出了软件工程的 7 条基本原理。他认为这 7 条原理是确保软件产品质量和开发效率原理的最小集合。这 7 条原理是互相独立的，其中任意 6 条原理的组合都不能代替另一条原理，因此，它们是缺一不可的最小集合。然而这 7 条原理又是相当完备的，人们虽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 多条软件工程原理都可以由这 7 条原理的任意组合蕴含或派生。

下面简要介绍软件工程的 7 条基本原理。

1）用分阶段的生命周期计划严格管理：有人经统计发现，在不成功的软件项目中有一半左右是由于计划不周造成的，可见把建立完善的计划作为第 1 条基本原理是吸取了前人的教训而提出来的。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。这条基本原理意味着，应该把软件生命周期划分成若干个阶段，并相应地制定出切实可行的计划，然后严格按照计划对软件的开发与维护工作进行管理。Boehm 认为，在软件的整个生命周期中应该制定并严格执行 6 类计划，它们是项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划和运行维护计划。不同层次的管理人员都必须严格按照计划各尽其职地管理软件开发与维护工作，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。因为，第一，大部分错误是在编码之前造成的，如根据 Boehm 等人的统计，设计错误占软件错误的 63%，编码错误仅占 37%；第二，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一种客观需要，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。绝对不能谁想修改软件（包括尚在开发过程中的软件），就随意进行修改。

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。传统的结构程序设计技术，现在的面向对象技术，都是优于早期的程序设计技术。实践表明，采用先进的技术不仅可以提高软件开发和维护的效率，而且可以提高软件产品的质量。

5）结果应能清楚地审查：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一般产品的开发过程更难于评价和管理。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审查。

6）开发小组的人员应该少而精：这条基本原理的含义是，软件开发小组的组成人员的素质应该好，而人数则不宜过多。开发小组人员的素质和数量，是影响软件产品质量和开发效

率的重要因素。素质高的人员的开发效率比素质低的人员的开发效率可能高几倍至几十倍，而且素质高的人员所开发的软件中的错误明显少于素质低的人员所开发的软件中的错误。此外，随着开发小组人员数目的增加，因为交流情况讨论问题而造成的通信开销也急剧增加。当开发小组人员数为 N 时，可能的通信路径有 $N(N-1)/2$ 条，可见随着人数 N 的增大，通信开销将急剧增加。因此，组成少而精的开发小组是软件工程的一条基本原理。

7）承认不断改进软件工程实践的必要性：遵循上述 6 条基本原理，就能够按照当代软件工程基本原理实现软件的工程化生产。但是，仅有上述 6 条原理并不能保证软件开发与维护的过程能赶上时代前进的步伐，不能跟上技术的不断进步，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 7 条基本原理。按照这条原理，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，如收集进度和资源耗费数据，收集出错类型和问题报告数据等。这些数据不仅可以用来评价新的软件技术的效果，而且可以用来指明必须着重开发的软件工具和应该优先研究的技术。

1.1.2.3 软件工程包含的领域

IEEE 在 2004 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 10 个知识领域。

1. 软件需求（software requirements）

软件需求表达了为解决某些真实世界问题而施加在软件产品上的要求和约束。软件需求的主要类型包括：产品与过程，功能性与非功能性，突出的属性。软件需求知识领域涉及软件需求的抽取、分析、规格说明和确认。软件工业界一致认同的就是如果这些工作完成得不好，软件工程项目就很容易失败。

2. 软件设计（software design）

软件设计是一个过程和这个过程的结果，此过程对一个系统或组件定义架构（architecture，也叫体系结构）、组件、接口以及其他特征。软件设计作为过程看待时是一项软件工程生命周期的活动。在这项活动中分析软件需求以产生一个软件内部结构的描述，此描述将成为软件构建的基础。更精确地说，软件设计的结果必须描述软件架构（即软件如何分解成组件并组织起来）以及这些组件之间的接口，它必须详细地描述每个组件以便能构建这些组件。

软件设计在软件开发中起着重要作用：软件工程师为了实现一个解决方案要设计出各种的模型形成蓝图。我们可以分析和评估这些模型，以确定据此能否实现各种不同的需求；也可以检查和评估候选方案，并进行权衡取舍。最后，我们可以使用作为设计结果的模型来规划后续的开发活动，并且作为构造与测试的输入和起始点。

3. 软件构建（software construction）

软件构建指的是如何创建产生软件的详细步骤，这其中包括编码、验证、单元测试、集成测试和调试。

4. 软件测试（software testing）

测试是一个标识产品的缺陷和问题的活动。测试的目的是为了评估和改进产品质量。

软件测试通过使用有限的测试用例来动态地验证程序是否能达到预期的行为。有限的测试用例是从通常情况下有无限可能性的执行领域中适当地选取出来的。

5. 软件维护（software maintenance）

软件开发工作的结果就是交付一个满足用户需求的软件产品。软件产品一旦投入运行，产品的缺陷就会被逐渐地暴露出来，运行的环境会逐渐发生变化，新的用户需求也会不断地浮出水面。软件维护就是要针对这些问题而对软件产品进行相应地修改或演化，从而修正错误，改善性能或其他特征，以及使软件适应变化的环境。

6. 软件配置管理(software configuration management)

软件配置管理(software configuration management，SCM)是一项跟踪和控制软件变更的活动。

7. 软件工程管理(software engineering management)

软件工程管理是软件的开发和维护的管理活动，为了达到系统的、遵循规程的和可量化的目标，它包括计划、协调、度量、监控、控制和报表。

8. 软件工程过程(software engineering process)

可以在两个层次上分析软件工程过程领域。第1个层次包括软件生命周期过程中技术的和管理的活动，它们是在软件获取、开发、维护和退出运行中完成的。第2个层次是元层次，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. 软件工程工具和方法(software engineering tools and methods)

软件开发工具是用于辅助软件生命周期过程的基于计算机的工具，工具可以将重复并明确定义的动作自动化，减少了软件工程师的认知负担，使软件工程师可以集中在开发过程的创造性方面。另外，还可以通过工具来支持特定的软件工程方法，减少手工应用软件工程方法时相应的管理负担。工具的范围覆盖支持单个任务到囊括整个生命周期。

软件工程方法的目标是使软件工程活动系统化并最终更可能成功。方法通常提供一种符号和词汇表，为完成一组明确的任务提供流程，为检查过程和产品提供指南。方法的覆盖范围从单个生命周期阶段到整个生命周期。

尽管对于特定的工具有详尽的手册，对于新型工具有大量的论文，但是，关于软件工程工具的一般性技术读物却很少。一个困难是软件工具的高变化率，特定的细节有规则地变化着，使得难以提供具体的、最新的实例。

10. 软件质量(software quality)

什么是软件质量？多年以来，许多作者和组织对术语“质量”有着不同的定义。对于PhilCrosby，质量就是“遵从用户需求”。Watts Humphrey认为质量就是“达到适合使用的卓越层次”。IBM发明了术语“市场驱动的质量”，它基于达到全面的客户满意，关于组织质量的Baldrige准则使用了一个类似的短语“客户驱动的质量”，将客户满意作为主要的考虑。在ISO 9001—00中，质量被定义为“一组内在特征满足需求的程度”。

1.2　软件开发方法

随着人们对软件工程认识的不断加深，在软件开发的各个阶段，其过程、任务的结构框架形成了一些固定的模式，称之为软件开发模型。

软件开发模型能清晰、直观地表达软件开发全过程，明确规定了要完成的主要活动和任务，用来作为软件项目工作的基础。

每一种软件开发模型都有其优缺点，明白每种模型的特点以及适用场景，是软件项目开

发的基础。

1.2.1 瀑布模型

瀑布模型(Waterfall Model)是最早的软件开发模型，它将软件生存周期的各项活动规定为按固定顺序而连接的若干阶段工作，形如瀑布流水(见图 1－1 瀑布模型)，最终得到软件产品。

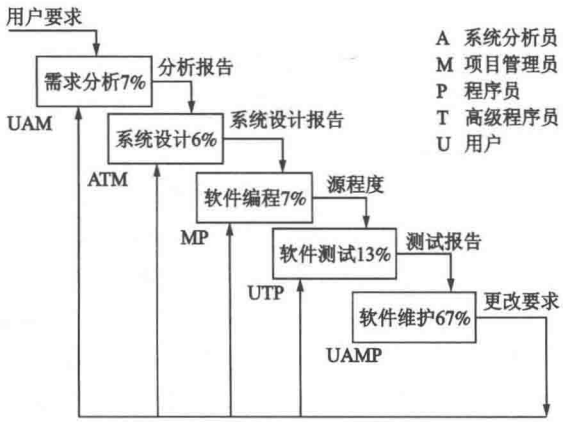

图 1－1 瀑布模型

瀑布模型的开发过程是通过设计一系列阶段顺序展开的，从系统需求分析开始直到产品发布和维护，每个阶段都会产生循环反馈，因此，如果有信息未被覆盖或者发现了问题，那么最好“返回”上一个阶段并进行适当的修改，项目开发进程从一个阶段“流动”到下一个阶段，这也是瀑布模型名称的由来。包括软件工程开发、企业项目开发、产品生产以及市场销售等构造瀑布模型。

瀑布模型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，即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。将软件生命周期划分为制定计划、需求分析、软件设计、程序编写、软件测试和运行维护等六个基本活动，并且规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。

瀑布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。其过程是从上一项活动接收该项活动的工作对象作为输入，利用这一输入实施该项活动应完成的内容给出该项活动的工作成果，并作为输出传给下一项活动。同时评审该项活动的实施，若确认，则继续下一项活动；否则返回前面，甚至更前面的活动。对于经常变化的项目而言，瀑布模型毫无价值。

随着时间的推移，瀑布模型的缺点越来越明显：

1)各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量。

2)由于开发模型是线性的，用户只有等到整个过程的末期才能见到开发成果，从而增加了开发风险。

3)早期的错误可能要等到开发后期的测试阶段才能发现，进而带来严重的后果。

4）通过过多的强制完成日期和里程碑来跟踪各个项目阶段。

5）瀑布模型的突出缺点是不适应用户需求的变化。

尽管瀑布模型招致了很多批评，但是它对很多类型的项目而言依然是有效的，如果正确使用，可以节省大量的时间和金钱。对于软件项目而言，是否使用这一模型主要取决于是否能理解客户的需求以及在项目的进程中这些需求的变化程度，对于经常变化的项目而言，瀑布模型毫无价值，对于这种情况，可以考虑其他的架构来进行项目管理。

在瀑布模型中，软件开发的各项活动严格按照线性方式进行，当前活动接受上一项活动的工作结果，实施完成所需的工作内容。当前活动的工作结果需要进行验证，如果验证通过，则该结果作为下一项活动的输入，继续进行下一项活动，否则返回修改。

1.2.2　快速原型模型

快速原型模型需要迅速建造一个可以运行的软件原型，以便理解和澄清问题，使开发人员与用户达成共识，最终在确定的客户需求基础上开发客户满意的软件产品。快速原型模型允许在需求分析阶段对软件的需求进行初步而非完全的分析和定义，快速设计开发出软件系统的原型，该原型向用户展示待开发软件的全部或部分功能和性能；用户对该原型进行测试评定，给出具体改进意见以丰富细化软件需求；开发人员据此对软件进行修改完善，直至用户满意认可之后，进行软件的完整实现及测试、维护。

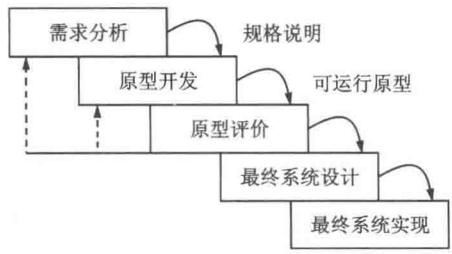

图 1－2　快速原型模型

快速原型模型又称原型模型，它是增量模型的另一种形式；它是在开发真实系统之前，构造一个原型，在该原型的基础上，逐渐完成整个系统的开发工作。快速原型模型的第一步是建造一个快速原型，实现客户或未来的用户与系统的交互，用户或客户对原型进行评价，进一步细化待开发软件的需求。通过逐步调整原型使其满足客户的要求，开发人员可以确定客户的真正需求是什么；第二步则在第一步的基础上开发客户满意的软件产品。

快速原型模型克服了瀑布模型的缺点，减少由于软件需求不明确带来的开发风险，适合预先不能确切定义需求的软件系统的开发。其缺点所选用的开发技术和工具不一定符合主流的发展；快速建立起来的系统结构加上连续的修改可能会导致产品质量低下。

快速原型是利用原型辅助软件开发的一种新思想。经过简单快速分析，快速实现一个原型，用户与开发者在试用原型过程中加强通信与反馈，通过反复评价和改进原型，减少误解，弥补漏洞，适应变化，最终提高软件质量。

传统的瀑布模型本质上是一种线性顺序模型，存在着比较明显的缺点，各阶段之间存在

着严格的顺序性和依赖性，特别是强调预先定义需求的重要性，在着手进行具体的开发工作之前，必须通过需求分析预先定义并“冻结”软件需求，然后再一步一步地实现这些需求。但是实际项目很少是遵循着这种线性顺序进行的。在系统建立之前很难只依靠分析就确定出一套完整、准确、一致和有效的用户需求，这种预先定义需求的方法更不能适应用户需求不断变化的情况。

传统的瀑布模型很难适应需求可变、模糊不定的软件系统的开发，而且在开发过程中，用户很难参与进去，只有到开发结束才能看到整个软件系统。这种理想的、线性的开发过程，缺乏灵活性，不适合实际的开发过程。

而快速原型模型的提出，可以较好地解决瀑布模型的局限性，通过建立原型，可以更好地和客户进行沟通，解决对一些模糊需求的澄清，并且对需求的变化有较强的适应能力。原型模型可以减少技术、应用的风险，缩短开发时间，减少费用，提高生产率，通过实际运行原型，提供了用户直接评价系统的方法，促使用户主动参与开发活动，加强了信息的反馈，促进了各类人员的协调交流，减少误解，能够适应需求的变化，最终有效提高软件系统的质量。

1.2.3 螺旋模型

螺旋模型(Spiral Model)采用一种周期性的方法来进行系统开发。这会导致开发出众多的中间版本。使用它，项目经理在早期就能够为客户实证某些概念。该模型是快速原型法，以进化的开发方式为中心，在每个项目阶段使用瀑布模型法。这种模型的每一个周期都包括需求定义、风险分析、工程实现和评审 4 个阶段，由这 4 个阶段进行迭代。软件开发过程每迭代一次，软件开发又前进一个层次。螺旋模型的开发活动如图 1 -3 所示：

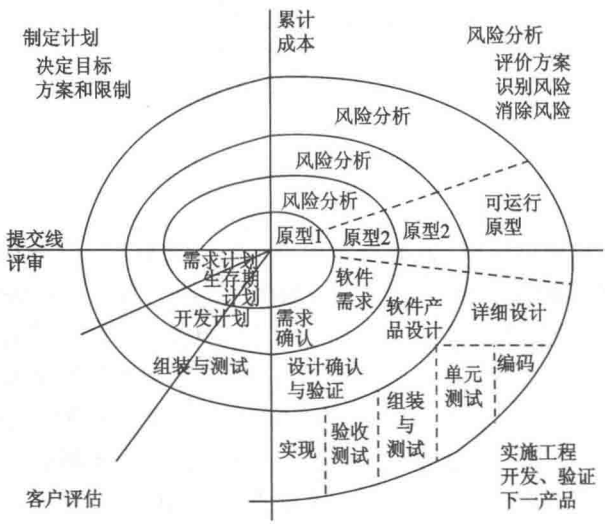

图 1 -3 螺旋模型

螺旋模型基本做法是在“瀑布模型”的每一个开发阶段前引入一个非常严格的风险识别、风险分析和风险控制，它把软件项目分解成一个个小项目。每个小项目都标识一个或多个主要风险，直到所有的主要风险因素都被确定。