ELECTRONIC ENGINEER
XIDIAN UNIVERSITY PRESS

Agile Software Development Methods and Practices

敏捷软件开发方法与实践

桑大勇　王 瑛　吴丽华　编著

西安电子科技大学出版社
XIDIAN UNIVERSITY PRESS

敏捷软件开发方法与实践

桑大勇　王　瑛　吴丽华　编著

西安电子科技大学出版社

内 容 简 介

本书第 1 章阐述了敏捷软件开发方法出现的历史背景、敏捷宣言、敏捷原则及最新动态；第 2 章介绍了常见的敏捷软件开发方法及其相互间的简单比较；在第 3 章至第 5 章中，作者结合自己的敏捷项目开发经验，融合其他方法，介绍了敏捷软件交付模型以及部分敏捷项目管理和开发实践；第 6 章从组织变革实施模型的角度分析了软件开发组织(全企业或企业中的一些部门)如何进行敏捷转型；第 7 章介绍了在分布式开发环境和团队中如何采用敏捷实践。

本书的目标读者包括软件行业从业人员、高等院校软件工程专业本科生和研究生以及对敏捷软件开发感兴趣的教学研究人员。

图书在版编目(CIP)数据

敏捷软件开发方法与实践/桑大勇，王瑛，吴丽华编著.

—西安：西安电子科技大学出版社，2010.5

ISBN 978-7-5606-2419-8

Ⅰ. ① 敏… Ⅱ. ① 桑… ② 王… ③ 吴… Ⅲ. ① 软件开发 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字(2010)第 056311 号

策　　划　臧延新

责任编辑　樊新玲　臧延新

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　陕西光大印务有限责任公司

版　　次　2010 年 5 月第 1 版　2010 年 5 月第 1 次印刷

开　　本　787 毫米×1092 毫米　1/16　印张　13.5

字　　数　300 千字

印　　数　1～2000 册

定　　价　22.00 元

ISBN 978-7-5606-2419-8/TP • 1209

XDUP 2711001-1

如有印装问题可调换

前　言

自 1968 年“软件危机”[Naur et al 1969]一词出现以来，软件产业从业者和学者一直在探讨如何将传统行业中的工程方法应用于软件行业，希望软件的开发过程以一种受控、可预测的方式进行，并因此出现了软件工程这一学科。40 多年来，软件从仅仅应用于国防军事和航空航天等高端领域，逐渐渗透到几乎所有的产业，甚至已经像水和空气一样成为人类日常生活和工作不可或缺的元素。相应地，软件开发领域相继出现了许多不同的开发过程和模型，从瀑布式模型、螺旋式模型，到 CMMI 软件能力成熟度评估和改进框架等。这些过程模型和框架无一例外地都是基于“完美的结果产生于精确控制的过程”这个理念，对软件开发生命周期中的计划与执行都十分重视，“按计划、不超预算、实现了预定的需求规格范围、产出物的质量可接受”，成了一种公认的、软件开发项目成功的标准。

对于最早利用计算机软件的国防军事和航空航天等复杂的、需要大量预先设计的应用领域来说，上述模型或项目成功标准依然成立。但是就数量上来说，在今天，更多的软件项目是服务于面对市场激烈竞争氛围的企业。能否快速响应市场的变化、调整自己的经营和管理方式，是决定一个企业能否生存和发展的根本因素，在这种情况下，已有的软件开发过程和模型就显得有些滞重，从而造成企业的信息化系统研发经常不能满足其日新月异的经营方式所需。为此，从 20 世纪 90 年代中后期开始，在企业应用软件开发的圈子里，陆续出现了一些轻量级的开发方法[Fowler 2004]，这些方法以企业业务价值最大化为目标，快速适应企业的业务变化，并尽量缩短企业信息系统从规划到初次投入使用的时间周期。在 2001 年的一次学术讨论会上，这些方法的创作者和拥护者们总结了这些方法的共性，发表了敏捷宣言，并将这些方法统一到“敏捷”这一面旗帜之下[Agilemanifesto 2001]。

最近十几年来，很多敏捷软件开发方法的成功案例，终于使之从“草根一族”渐渐走入了主流软件开发方法学的厅堂：CMM/CMMI 的核心发起单位——卡耐基-梅隆大学的 SEI，也专门有研究报告，以论证敏捷方法和 CMMI 的兼容性和互补性[Glazer et al 2008]；兼并了著名的软件工程工具厂商 Rational 的业界大鳄 IBM，也宣称推出自己的敏捷软件开发解决方案[IBM 2009]。国内敏捷软件开发的起步虽然较欧美晚，但在经济全球化的今天，敏捷热潮也逐渐从国外传到国内，尤其是随着业界领导厂商之一的 ThoughtWorks 公司进入中国市场，以及一贯致力于敏捷方法推广的 IT 专业媒体网站 InfoQ 中文站的开通，使得敏捷方法在中国也正以燎原之势快速地传播。通过其官方网站不难看出，国内最大的软件企业华为公司也已经开始采用这种方法[Huawei 2009]。

众所周知，日本丰田汽车的精益制造方法[Krafcik 1988]，是对美国福特公司所发明的、

曾经主宰汽车制造业的大批量、流水线生产方式的一种革命性的推进。精益的核心就是消除生产过程中的一切浪费，而敏捷软件开发方法由于大大减少了开发过程中在制品的数量及相关活动，加之所有的开发活动都是受客户的业务价值最大化驱动的，因此也极大地降低了不直接体现在最终产出物价值中的成本，与精益制造的想法不谋而合[Elssamadisy et al 2007]。

目前国内有关敏捷软件开发的书籍虽然也有一些，但多为译作或影印版本。作者衷心希望本书能有抛砖引玉之效果，引起中国软件业从业者和广大学者对敏捷软件开发给予更大的关注，并在不久的将来国内能够出现更多、更好的本土原创著作。受作者水平所限，本书中难免存在诸多不当之处，敬请读者向本书作者(sangdayong@gmail.com)指出。

本书由三位作者联合编著而成。海南师范大学信息与科学技术学院的吴丽华教授编写了本书的第 1 章，空军工程大学工程学院航空装备管理工程系的王瑛教授编写了本书的第 2 章，其余章节由桑大勇编写。

本书在完成的过程中得到了海南师范大学学术著作出版资助项目的支持；作者曾经向用友管理软件学院的李台元院长多次请教过企业信息化的方法论，对形成第 6 章有很大帮助。作者谨在此表示衷心的感谢。最后，还特别要感谢西安电子科技大学的蔡希尧教授，他对本书体系框架的形成给予了很多指导，提出了许多建设性的意见。

目　录

第 1 章　敏捷软件开发方法的历程

1.1　敏捷方法的出现

1.1.1　软件开发简史

软件对于我们今天的工作和生活来说，从某种意义上已经像空气和水一样，我们常常忽略它们的存在，直到偶然的机会当我们已经习以为常的软件突然失灵的时候才会发现，原来软件早就成为我们工作和生活中不可或缺的一部分了。电子计算机的历史本就不长，而软件重要性和地位的提高所经历的时间则更短。20 世纪 50 年代软件开始出现，但在 70 年代之前并没有引起重视，因为在新兴的计算机行业，软件此时所占的销售比重远低于计算机硬件。例如，1970 年美国所有软件公司的软件年度总销售额只有区区 5 亿美元，仅相当于当年计算机行业总销售额的 3.7%左右。然而，此后软件业进入了快速发展时期，美国软件公司的软件年度总销售额在 1979 年已达到 20 亿美元，1985 年达到 250 亿美元，而今天则已超过数千亿美元。

软件的诞生已经超过半个世纪，与软件急速提高的重要性相比，软件开发方法至今仍然远未成熟。软件开发者们依然经常在一个个濒临失败的软件项目中苦苦挣扎，即便开发者们内心深处知道这个世界上没有、而且永远也不可能有解决软件开发问题的“银弹[1]”[Brooks 1995]，可还是不断地将希望寄托在寻求新的开发方法上。图 1-1 大致列出了从 20 世纪 50 年代至今，软件开发方法的演变过程。图 1-1 不一定全面和准确，因为本书的目的不是研究软件开发方法的历史，而是试图让读者体会到，敏捷方法的出现是有其历史原因的，是在众多软件开发过程和项目实践的基础上应运而生的。

1．编码和改错

20 世纪 50 年代，大型计算机相继在研究机构和大学的实验室出现。这些计算机主要用于工程和自然科学计算，其用户只是实验室内的少数几个人。这个阶段的软件开发模型基本上就是“编码和改错”[Boehm 1988]，主要包括两个步骤：① 书写少量代码；② 修改代码中存在的问题。也就是说，基本过程就是先写代码，然后再考虑需求、设计、测试、维护等这些东西。

[1]：见本书第 6 页“没有银弹”部分的解释。

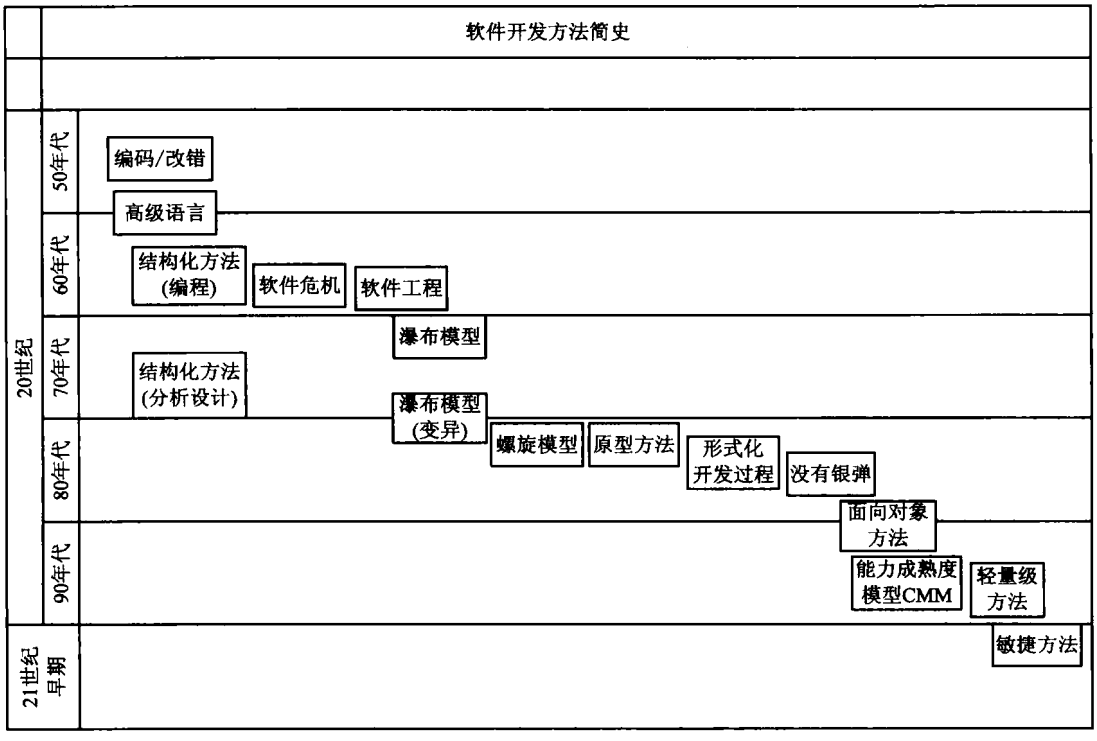

图 1-1 软件开发方法简史

2. 高级语言

20 世纪 50 年代，编程还是一项需要全身心投入、细致入微的复杂工作，程序员也十分钟爱书写出能够工作但极其晦涩难懂和诡异的代码；50 年代后期，计算机开始进入商业应用等公共领域，应用场合的增多意味着编制计算机程序的活动也渐渐增多，程序员成了一个新兴的职业。为了提高编码效率，作为形式化符号系统的高级编程语言(或称高级语言)开始出现。IBM 发布了第一个广为人知的高级语言 FORTRAN，随后 1958 年出现了 ALGOL，而在 1962 年美国国防部(Department of Defense，DoD)将 Cobol 作为商用编程语言发布。

3. 结构化方法

1965 年 Dijkstra 撰写了一篇著名的文章[Dijkstra 1972]，宣称编程不是一种工艺，而是一门学科；同年 Hoare 发表了一篇关于数据结构的重要论文[Hoare 1972]，这些论著深深地影响了 PASCAL 等新兴的结构化编程语言。结构化编程方法能更加容易地将软件系统划分成一个个独立的较小部分，让程序的正确性验证也变得容易起来。

结构化编程方法将程序划分成独立的组成部分，可以在一定程度上封装某一段程序的变化对其余部分的影响，即实现软件系统变化的局部性。后来，陆续出现了一些按顺序化阶段进行的软件开发过程(如瀑布模型)，为了对编码阶段之前各阶段(如分析和设计阶段)中所可能发生的变化进行封装，又出现了结构化分析与设计方法，如 DeMarco 的 SADT(结构化分析与设计技术)[DeMarco 1978]。

4. 软件危机

20 世纪 60 年代后期，随着计算机应用的日益广泛，许多软件开发项目出现了问题，如超过预算和预期交付时间，造成财产损失，有些项目甚至遭遇失去生命的严重后果[Leveson 1995]。软件开发界开始寻求降低开发风险、改善软件质量和开发生产力的办法。但研究表明[Lycett et al 2003]，仍然有 80%的软件项目延迟交付且超出预算，另外有 40%左右的项目失败或被废弃。即便那些已经交付的软件项目，可能也需要持续投入高昂的维护和打补丁的费用。

Standish 集团 2001 年的研究结果表明[Standish 2001]，软件项目的成功率只有 28%，其余项目要么失败，要么“受到质疑”(即超出预算、超出估计时间、发布的特性比最初确定的特性少等)。图 1-2 展示了各种软件项目所占的比例。因此，有些人坚持认为 20 世纪 60 年代出现的软件危机至今仍然存在。

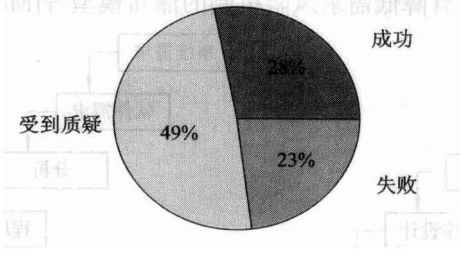

图 1-2　软件项目分类比例[Standish 2001]

5. 软件工程

自从 1968 年 NATO 的一次研讨会使用“软件工程”一词作为会议名称[Naur et al 1969]以后，这个词汇开始在业界频繁出现。那个时候使用这个“煽动性[2]”词汇的目的，是为了吸引人们注意软件开发中遇到的问题，并强烈地表达了一种观点：就像传统的各种工程学分支一样，软件制造也应当建立在理论基础和实践准则之上。之所以说“软件工程”是煽动性词汇，是因为在那次会议上并没有定义与这个词汇相关的任何关键概念，而只描述了一种目标。这种情况很长时间没有改观，例如 1990 年 IEEE 标准[IEEE 1990]对“软件工程”的解释也还只是一种目标，而不是一种定义。

软件工程

(1) 使用一种系统的、有准则可循的、量化的方法，开发、操作和维护软件，即把工程思想应用于软件。

(2) 有关(1)中所述方法的研究。

软件工程是为了解决软件危机问题，主要目的是降低开发风险，提高软件质量和开发生产率。但自那时起软件工程界就存在一种争论，即软件危机现在还是否存在？如果答案是“存在”，则说明软件开发不是一种工程学科，软件危机并没有如预期的那样得到解决；

[2]：文献[Naur et al 1969]中使用的英文原词是 provocative。

如果答案是“不存在”，说明软件开发是一种工程，可构成这个工程学科的内容又是什么呢？这仍然是一个没有公认答案的问题。

6. 瀑布模型

瀑布模型指一种将软件项目的进展过程用一系列按顺序执行的阶段来划分的开发过程，因其过程形如瀑布而得名，如图 1-3[Royce 1970]所示。该模型由文档驱动，注重项目的计划和预测，阶段与阶段之间没有时间重叠。每个阶段结束时，通过对该阶段所产生的文档进行评审，来决定项目是否可以进入下一阶段，如果评审没有通过，则项目应当保持在当前阶段。

最初提出的瀑布模型如图 1-3(a)所示，其中包含了迭代的步骤。但或许是因为概念更加简单的缘故，图 1-3(b)所示的简化版本却在业界得到了更多的应用。相对于 Royce 的“纯瀑布”模型，后来还出现了一些“变异瀑布”模型，如带有重叠阶段的瀑布模型(Sashimi)、带有子项目的瀑布模型和带有降低需求风险机制的瀑布模型等[McConnel 1996]。

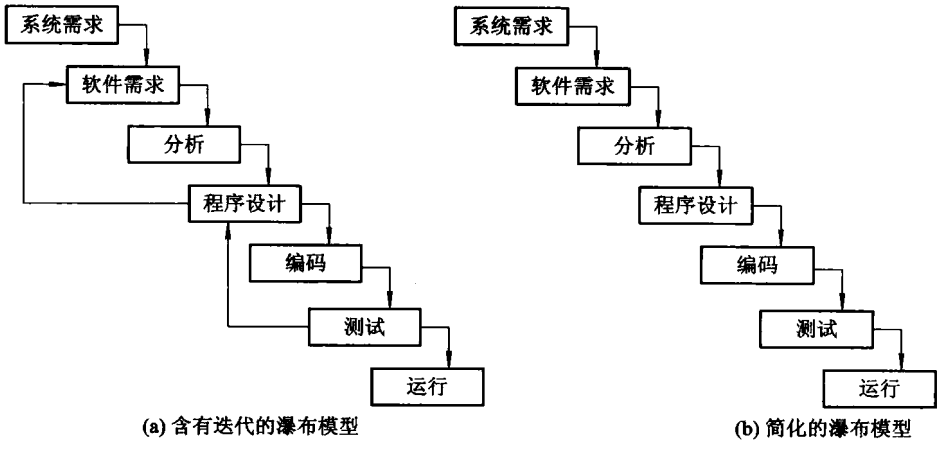

图 1-3 瀑布模型

尽管瀑布模型的缺点饱受批评，但至今仍被某些特定的软件项目管理层所垂青，其原因是瀑布模型易于解释，且让人感觉它是一种可控制、“按顺序、可预测、可靠、可度量、带有文档驱动的简单里程碑的”[Larman 2004]过程。

7. 螺旋模型

Boehm 对瀑布模型进行了改进，提出了螺旋模型[Boehm 1988]，如图 1-4 所示。螺旋模型是一种面向风险管理的生命周期模型，其核心思想是用原型和其他手段将风险最小化。该模型将软件项目分割成一系列的、以迭代方式实施的迷你项目，而每个迷你项目用来应对一个或多个主要风险，直到所有的风险都被处理后，再进入瀑布式开发过程。如果某一迭代(即某一个迷你项目)识别出来的风险无法被克服，就可以及时决策(如终止项目)，节约时间和资金。

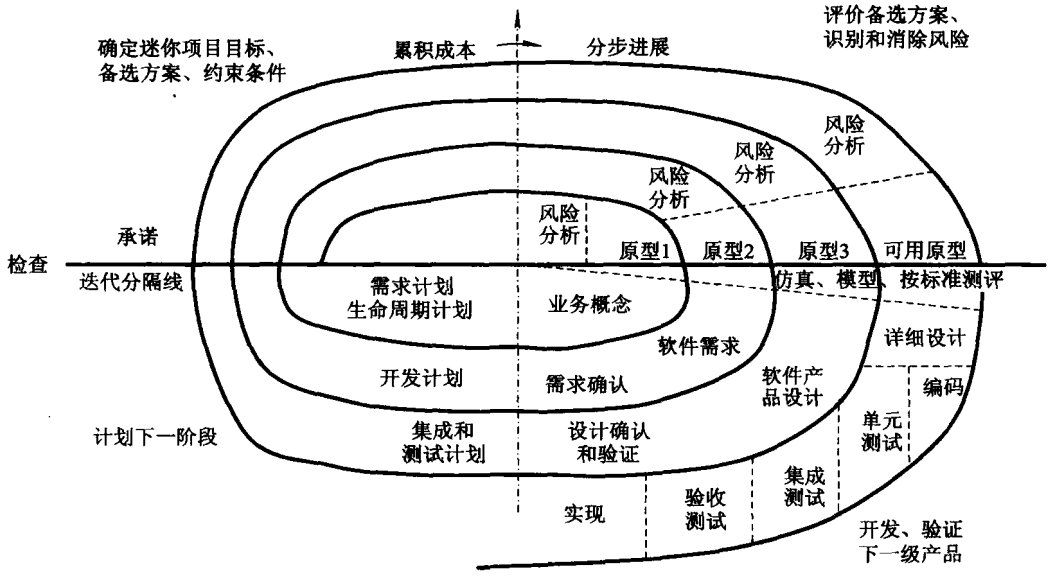

图 1-4　螺旋模型

这里所说的风险范围很广，可以是需求理解不透彻、架构理解不深刻、潜在的系统性能问题、底层技术问题等[McConnell 1996]。

8. 原型方法

原型方法[Boehm 1988]强调用户参与，如图 1-5[McConnell 1996]所示。开发人员使用应用生成器或专用的原型制作工具，从不同侧面就系统最重要的部分制作系统原型，然后给客户和最终用户展示，并与他们交流，以降低需求的不确定性。如果客户和最终用户对原型有不满意的地方，则他们可以与开发人员一起对原型进行精化，直到满意为止。一旦原型被接受，则要么将其作为最终产品的实现基础，要么将原型抛弃(即原型仅仅作为确认需求的工具)，开发人员使用另外的开发语言从头构建软件产品。

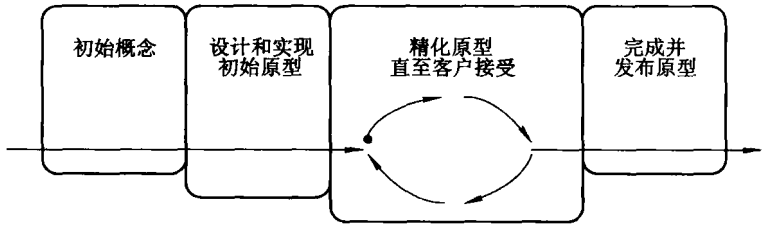

图 1-5　原型方法

9. 形式化开发过程

形式化开发过程通过自动化来增强软件的可信赖性，可用于软件规格的说明、转换和验证工作。形式化方法的拥护者们期望在理想情况下，一旦确定了形式化的系统需求，剩余的开发工作可以由一系列自动化的转换过程来完成，转换结果(所生成的软件系统)的正确

性能够得到保障。已经开发的相关工具包括设计语言(或称规格语言)、转换系统和验证器。

形式化方法主要在一些安全性至关重要的小型程序上获得了成功。但是在软件产业中，尽管付出了大量努力，但形式化方法被接受的程度仍然没有达到预期。对于大项目来说，不论是需求还是设计通常都无法用简洁的数学模型来描述。

10. 没有银弹

在西方民间传说中的所有恐怖妖怪中，最可怕的是人狼，因为它们可以完全出乎意料地从熟悉的面孔变成可怕的怪物，而只有银弹才可以杀死它们。为了对付人狼，人们在寻找可以消灭它们的银弹。

1986 年，F.P.Brooks 发表了论文《没有银弹：软件工程的根本和次要问题》[Brooks 1995]，预言今后十年内没有任何编程技巧能够给软件的生产率带来数量级上的提高，即对于软件开发这类具有一些“人狼”特征的项目来说，没有克敌制胜的银弹。

F.P.Brooks 将软件开发活动分为根本活动和次要活动两种类型。根本活动的目的是构造由抽象软件实体构成的复杂概念结构；次要活动的目的是使用编程语言表达这些抽象实体，并在空间和时间限制内将它们映射成机器语言。F.P.Brooks 认为软件生产率在此前所取得的巨大进步来自对后天障碍的突破，例如硬件的限制、笨拙的编程语言、机器时间的缺乏等，而这些突破所提高的只是次要活动的效率。相对于根本活动而言，如果软件工程师在次要活动上花费的时间和精力达不到 90%，则即使将全部次要活动的时间缩减到零，也不会给生产率带来数量级(即 10 倍以上)的提高。

在这篇文章中，F.P.Brooks 还用自己的经验，证实了 Mills 很久以前就提出的一种观点[Mills 1971]，即不应当“构建”一个软件系统，而应当在可运行系统(即便功能还不完备)上进行增量开发。采用这种增量式开发方法，可以比采用严格的软件工程方法更加方便地构建更为复杂的系统。这个观点也被后来的敏捷软件开发方法所采用。

11. 面向对象方法

结构化方法没有很好地解决软件系统开发中的三个问题：易于变化、易于重用和易于演进：

(1) 结构化方法关注的重点是处理过程，通过功能分解所封装的对象也还是处理过程，而处理过程仅仅是在数据集上进行的加工。然而，当系统需要变化时，往往是数据和过程都需要变化。因此结构化方法没有封装数据结构，即不能对数据变化进行封装。

(2) 早在 1968 年就有人预见到软件重用的潜力，可一直没有得到重视，结构化方法也并没有很好地解决软件重用问题。

(3) 结构化方法中在不同的阶段使用不同的建模方式，这些方式之间没有自然的延续性，系统演进过程中每次进行阶段更替都要经过一种模型转换的过程。转换成本很高，也很难保证转换不失真。

面向对象方法与技术主要就是要解决上述三个问题：

(1) 面向对象方法关注的是对象，对象是数据和加工过程(或称操作)的统一体。这种方

法用类来同时封装数据结构和数据加工过程的变化，用继承机制来封装问题领域中相似但不相同的概念间的部分属性变化，用多态机制封装多个规格相同的加工过程在实现上的变化。

(2) 通过类库或者应用程序框架实现程序代码，尤其是 GUI 代码的重用，通过设计模式实现面向对象设计的重用。

(3) 在分析、设计和编码阶段采用一致的概念模型和建模元素，从而使得从分析、设计到代码实现有可能完成无缝转换。

面向对象语言中很多概念的起源，可以追溯到 1967 年的 Simula 语言。Simula 中引入了类和继承；1972 年 Parnas Module 引入了信息隐藏；1974 年 CLU 引入了抽象数据类型；1981 年 Smalltalk-80 引入了类库，标志着面向对象编程语言(OOPL)的完善。

如同结构化方法一样，面向对象方法也是首先在程序设计中出现，然后才依次被延伸到设计和分析阶段。面向对象分析与设计始于 1986 年 G.Booch 提出的 OOD(面向对象设计)方法，后来又相继出现了 Shlaer/Mellor 方法(1988)、Coad/Yordon 方法(1991)、J.Rumbaugh 的 OMT 方法(1991)、I.Jacobson 的 OOSE 方法、GoF 的设计模式(1993—1994)、CBSE(基于组件的软件工程，1997)、UML(统一建模语言，1997)、MDA(模型驱动架构，2002)、UML2(2004)等。

12. 能力成熟度模型 CMM

到了 20 世纪 80 年代中期，软件开发似乎还是处于危机状态。在许多开发组织中，软件开发的进度、成本和产品总体质量仍然无法预测。美国联邦政府开始寻求一种对其软件承包商的能力和成熟度进行评估的办法，政府相信成熟的开发组织能够精确地预测软件开发的进度、成本和质量。1986 年软件工程研究所(Software Engineering Institute，SEI)开始开发一种过程能力框架作为能力评估方法，并对希望进行过程改进的组织提供帮助。SEI 这项工作的结果是 1993 年推出的软件能力成熟度模型(SW-CMM)，以及近 10 年后，即在 2002 年推出的能力成熟度集成模型(CMMI) [Chrissis 2003]。

CMMI 的发布，坚定地将软件开发同工程化产品生产归为一类[Chrissis 2003]。然而在 SEI 开发 CMMI 的同时，有人开始质疑软件工程能否达到人们所赋予它的目标，或者说开始质疑软件开发到底是不是一门工程学科[Mahoney 2004]。

13. 轻量级方法

随着许多小型组织内对软件需求的扩张，他们需要价格低廉的软件解决方案，从而促成了一些更加简单、快速、易用的软件开发方法的出现。从 20 世纪 90 年代中期开始，作为严格的软件工程方法的替代方法，这些方法开始引起业界的注意。这些新方法的共同特点是，通过面对面的交流和频繁的交付，来促成程序员团队与业务专家之间的紧密协作。

这些方法从原型方法进化而来。同传统的软件开发工程化过程相比，这些方法没有繁琐、沉重的过程管理负担，因此被统称为“轻量级方法”。传统的软件开发工程化过程相应地称做重量级方法或过程。为了提供日益增长的、数量巨大的小型软件系统，轻量级方法试图简化许多软件工程过程域，包括需求收集和可靠性测试[Wikipedia 2009a]。

14. 敏捷方法

2001 年 2 月，17 位知名的、发明或推崇轻量级方法的软件过程方法学家们，在参加一次旨在探索更好的软件开发方法的峰会时，发现这些“轻量级”方法之所以有效的原因，不是因为其过程简单，而是有一些共同的“敏捷”特征，于是便将这些方法统称为敏捷方法并成立了敏捷联盟。比较为人所知的敏捷软件开发方法，有极限编程(Extreme Programming，XP)、Scrum、特性驱动开发(Feature Driven Development，FDD)、动态系统开发方法(Dynamic System Development Method，DSDM)、适应性软件开发(Adaptive Software Development，ASD)、Crystal 和精益软件开发(Lean Software Development，LD)、敏捷统一过程(Agile Unified Process，AUP)等。本书将在第 2 章中对其中的一些方法进行简单的介绍。

“敏捷”的本意是指灵活的、有响应的，因而 Anderson 认为敏捷方法意味着“在永恒变化的环境中能够生存，并和成功相伴”[Anderson 2004]。Highsmith 则将敏捷定义为“为了在动荡的商务环境中获取利润而既能够创造变化，又能够响应变化的能力”[Highsmith 2002]。Cockburn 则补充说敏捷“是有效的和可机动的”[Cockburn 2002]。以上这些定义表明，敏捷的组织拥抱变化，行动灵活，并可在结构的刚性和柔韧性之间达到平衡。

1.1.2 敏捷方法是历史的必然

纵观软件开发方法的发展史，开发方法的演进遵循了一条从“计划和预测”发展到“反馈和适应”的历程，如图 1-6 所示。敏捷方法有时候被看做是计划驱动、讲求纪律的传统开发方法的对立面，这是一种误导，似乎敏捷方法就是没有计划、不讲纪律的方法。更为准确的区别是传统的方法更加靠近“预测性”一端，而敏捷方法更加靠近“适应性”一端[Boehm et al 2004]。

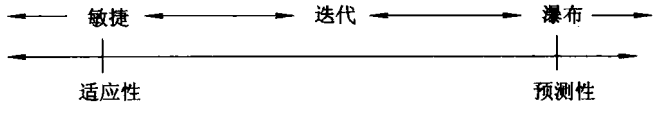

图 1-6 软件开发方法的持续演进过程

适应性方法关注的焦点是快速适应真实世界的变化。当项目的需要发生变化时，采用适应性方法的团队(以下简称适应性团队)也要做出变化。适应性团队很难准确地描述未来将要发生什么，将来的时间越远，适应性团队对其预测就越模糊。适应性团队能够准确汇报下一周要完成的任务，但只能知道下一个月计划要开发的特性，并不知道为此又需要完成哪些任务。如果问及他们六个月以后的那次发布包括什么内容，适应性团队可能只能讲出那次发布的目标描述、所期望实现的业务价值或预期发生的成本。

与此相反，预测性方法关注的焦点是对未来所进行的详细计划。采用预测性方法的团队(以下简称预测性团队)可以准确地告诉你在整个开发周期内所计划的特性和具体任务。预测性团队要想改变项目方向将非常困难，通常情况下项目计划已经根据最初的项目目标做过优化，