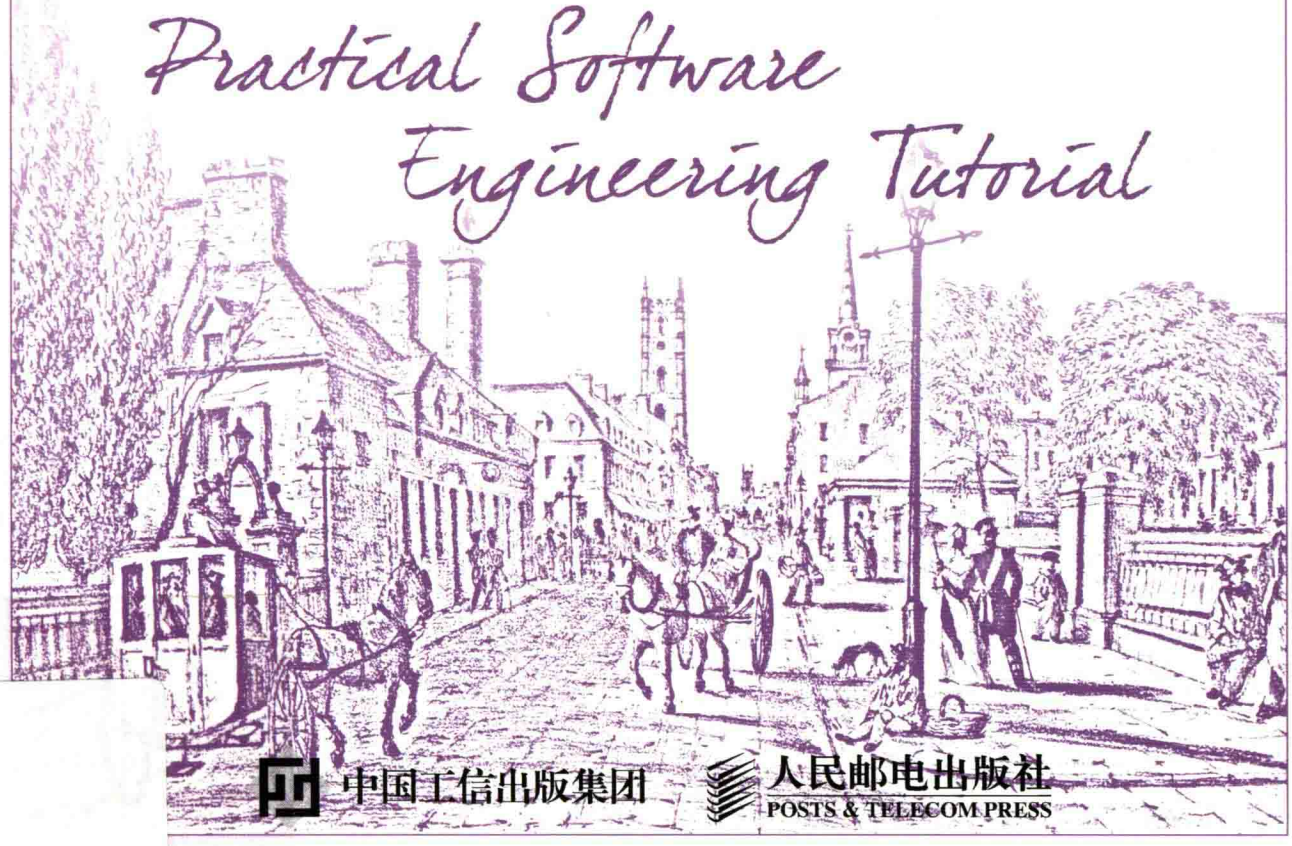
普通高等教育
软件工程 “十二五”规划教材

12th Five-Year Plan Textbooks
of Software Engineering

软件工程实用教程

朴勇 ◎ 编著

Practical Software Engineering Tutorial



中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

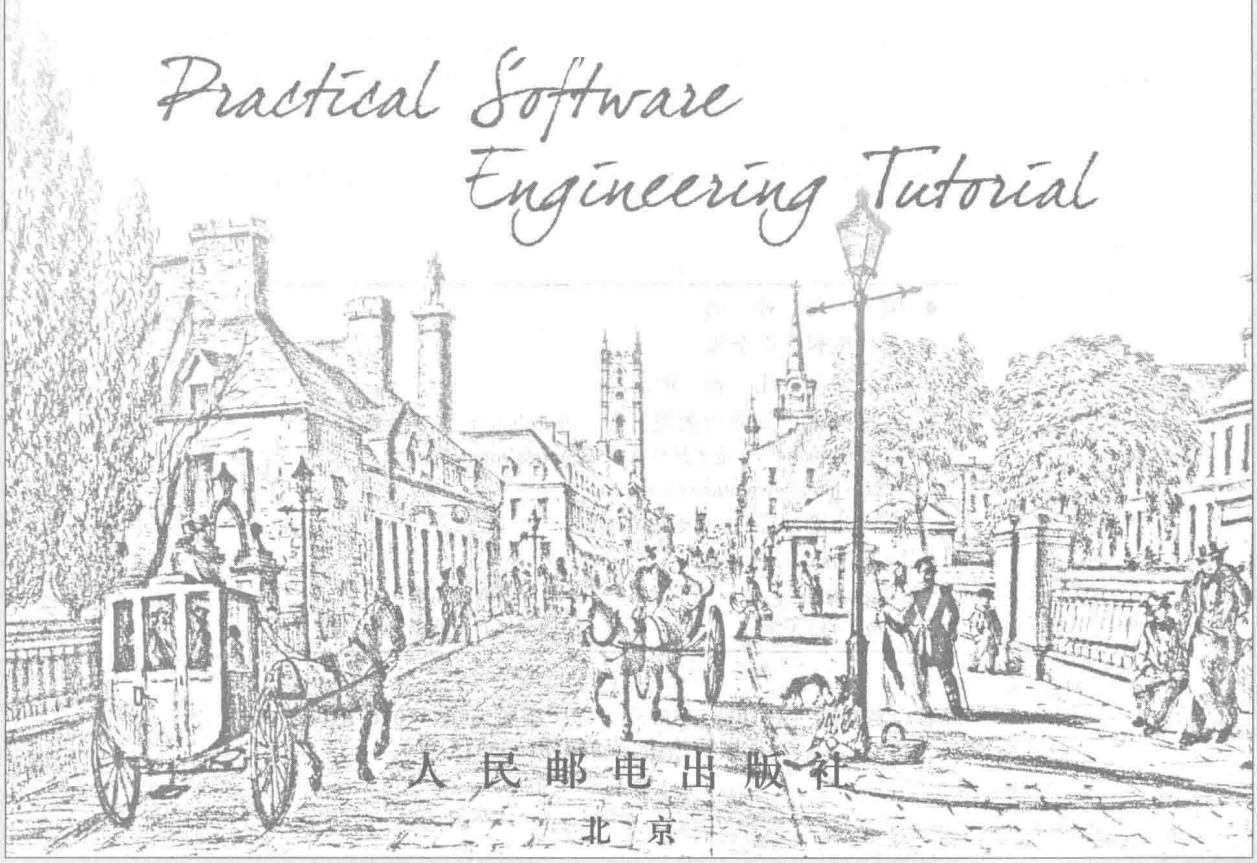
普通高等教育
软件工程 “十二五”规划教材

12th Five-Year Plan Textbooks
of Software Engineering

软件工程实用教程

朴勇 ◎ 编著

Practical Software Engineering Tutorial



人民邮电出版社
北京

图书在版编目（CIP）数据

软件工程实用教程 / 朴勇编著. -- 北京 : 人民邮电出版社, 2015.8
普通高等教育软件工程“十二五”规划教材
ISBN 978-7-115-39317-3

Ⅰ. ①软… Ⅱ. ①朴… Ⅲ. ①软件工程－高等学校－教材 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2015)第123811号

内 容 提 要

本书主要围绕软件的系统工程化开发过程，介绍相关的理论、方法、技术和工具。本书以面向对象的分析和设计为主线，遵循 UML 2 标准，以基本理论为出发点，介绍对软件开发的组织管理及业务流程分析；重点针对软件开发的生命周期，讨论需求分析、类的概要设计、代码生成之道、类的详细设计、设计优化、实现、交互设计、质量保证等重要环节；介绍软件开发环境，包括项目计划及跟踪。

本书内容丰富、循序渐进，注重软件工程理论与实践的结合，可作为高等院校计算机相关专业本科生和研究生的教材，也可作为从事软件开发的理论研究人员及工程技术人员的参考用书。

◆ 编　　著　朴　勇
责任编辑　许金霞
责任印制　沈　蓉　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市潮河印业有限公司印刷

◆ 开本：787×1092　1/16
印张：16.5　　2015 年 8 月第 1 版
字数：432 千字　　2015 年 8 月河北第 1 次印刷

定价：38.00 元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315

前　言

软件工程是软件学院开设的一门专业基础课程，主要介绍软件工程的基本原理、开发方法和开发工具，是软件开发的理论课程，同时也具有很强的实践性；另外，该课程涉及计算机、经济学、管理学、工程学、市场学等多个领域的知识，具有知识广泛的特点。

作者在多年教学实践中，深切体会到该课程讲授过程中的诸多问题，如理论知识的枯燥乏味，不容易将抽象的理论与实践联系起来；没有大项目案例的依托，体会不出软件工程的作用；软件开发经验介绍和感受的机会少等。如何突出重点，如何应用案例进行教学，如何使学生掌握必备的软件工程基础及应用这些都对软件工程课程的讲解提出了挑战。

本书突出实用的特点，一方面涵盖软件工程主要知识点，另一方面以案例为线索，以面向对象方法为主线，将分散的知识点连贯起来，便于读者理解和消化，为读者提供一条循序渐进的学习路线，可以为软件工程课程的教学提供较好的辅助。

本书具体内容包括：第 1 章概述了软件工程相关的概念、技术与方法；第 2 章围绕软件开发过程，对软件开发的组织管理及业务流程分析进行了说明；第 3～10 章主要围绕软件工程开发的生命周期，讨论了需求分析、类的概要设计、代码生成之道、类的详细设计、设计优化、交互设计、质量保证等内容；第 11 章介绍软件开发环境，包括项目计划及跟踪；第 12 章重点介绍版本控制系统。本书知识结构紧凑、面向软件学院教学实际、突出案例教学，在开发方法的介绍上通过具体案例贯穿，并尽量以程序代码的形式对相关的理论进行说明和阐释，从而达到理论联系实际的目的。

本书的写作得到了所在教学团队的大力支持和配合，在此向周勇、梁文新、陈鑫以及许真珍等老师表示感谢。另外，本书在写作过程中参考和引用了大量国内外同行的文献，在此对他们一并表示感谢。

限于作者的水平及精力，书中难免有不妥之处，敬请各位读者批评指正。

作　者

2015 年 2 月

目 录

第1章 软件工程概述

软件的开发与管理过程其实是一个优化问题，人们总是希望能在有限的资源条件下做到收益的最大化，如有限的预算、不断压缩的交付时限、软件工程师的数量及能力、各种风险的预期等，最终的目标是为了在保证产品质量的情况下尽量降低软件开发的成本，软件工程的引入就是力求达到这一目的。

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获得的最好的开发技术方法结合起来的学科。

本章主要介绍软件危机的产生以及软件工程的由来、软件工程包括的主要内容以及软件开发的主要方法及技术。

1.1 软件危机与软件工程

1.1.1 软件危机

软件工程的提出始于软件危机的出现。1968 年，北大西洋公约组织（NATO）在当时的联邦德国召开的国际学术会议上提出了软件危机一词，并同时提出软件工程的概念以解决软件危机。软件危机是指在软件开发及维护的过程中所遇到的一系列严重问题，这些问题的出现可能导致软件产品的寿命缩短，甚至夭折。

软件危机在 20 世纪 70 年代表现得尤其严重，具体表现有：超预算、超期限、质量差、用户不满意、开发过程无法有效介入和管理、代码难以维护等。人们逐渐认识到软件开发是一项高难度、高风险的活动，因为它失败的可能性较大。软件危机的产生与软件本身的特点有很大的关系，其中最主要的就是软件的复杂性。

- 软件是逻辑的，而不是有形的物理文件，与硬件具有完全不同的特征。而且，软件的主要成本产生于设计与研制的过程，而不是在制造的环节。软件的制造过程可以理解为“复制”。
- 软件在使用过程中不会磨损，但会退化。因此，软件的维护，不能像维修硬件那样，进行简单的更换，软件维护就是修复不断发现缺陷。这个过程比较复杂，有时需经历新的开发过程，而且缺陷被发现得越晚，为之付出的代价也越高。
- 软件开发早期是一种艺术，但目前越来越趋于标准化，软件产业正向大规模制造和基于构件的方向前进。

- 软件同时也是一种逻辑实体，具有抽象性。软件可以被使用，但无法看到其本身的形态。软件产品是人类智慧的作品。
- 软件是复杂的，并且会越来越复杂。人类思想的复杂性导致了软件的复杂性，而且随着信息领域的发展，软件的规模会越来越大，也越具有复杂性。

另一方面，人们对软件往往有着过高的期望，认为软件无所不能，对软件的认识也比较模糊。比如，早期人们对软件的误解之一就是软件即程序，软件开发就是编写程序，编写程序就是软件开发的全部工作等。实际上，软件是由 3 部分组成的，即程序、数据和文档。程序是指能够运行的提供所希望的功能和性能的指令集。数据是指支持程序运行的数据。文档是指描述程序研制的过程、方法及使用的记录。随着对软件了解的深入，人们也认识到软件开发一般性的规律——变化是软件开发不变的主题，变化带来了诸多挑战。

- 软件开发各环节对于缺陷具有放大作用，一个小的问题如不及时识别和处理，经过几级放大后，会在后期带来“可观”的成本上的损失。
- 只有在早期发现问题，才会尽量减少损失，缺陷的遗漏具有“失之毫厘，谬以千里”的副作用，但一个难以解决的问题是用户的需求不可能一次性地确定下来；甚至有时用户对软件的需求也是模糊的，他们本身也需要一个不断学习的过程。

总之，软件危机的产生主要是由软件的复杂性、过高的期望及无处不在的变化导致的。人们逐渐认识到应对软件危机的必要性和寻找解决软件危机的途径，主要包括如下内容：

- 要对软件有正确的认识。
- 推广使用开发软件成功的技术和方法，研究探索更有效的技术和方法。
- 开发和使用更好的软件工具。
- 对于时间、人员、资源等，需要引入更加合理的管理措施。

1.1.2 软件工程知识体系

软件工程是从技术和管理两个方面更好地开发和维护计算机软件的一门学科。IEEE 对软件工程的定义是：将系统化、规范化、可量化的工程原则和方法应用于软件的开发、运行和维护及对其中方法的理论研究，其主要目标是高效开发高质量的软件，降低开发成本。

1999 年 5 月，ISO 和 IEC 的第一联合技术委员会启动了标准化项目——软件工程知识体系指南（Guide to the Software Engineering Body of Knowledge，SWEBOK）。SWEBOK 指南的目的是为软件工程学科的范围提供一致的确认。软件工程知识体系中包含了 10 个主要的知识域，如图 1.1 所示。

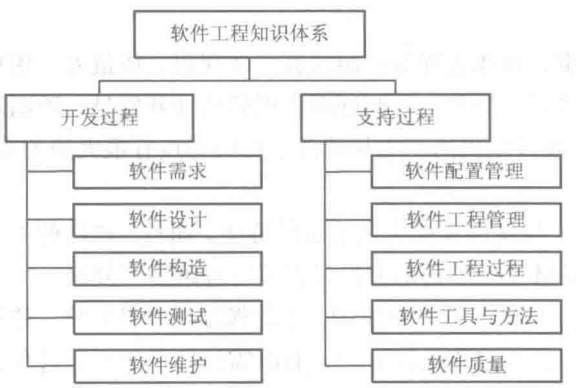


图 1.1　软件工程知识体系

软件需求（Software Requirements）是对业务领域中需要的特征的理解，包括软件需求基础、需求过程、需求获取、需求分析、需求规格说明、需求确认和实践考虑。

软件设计（Software Design）是定义系统或组件的体系结构、组成、接口和其他特征的过程及其输出结果，包括软件设计基础、软件设计关键问题、软件结构与体系结构、设计质量的分析与评价、软件设计表示、软件设计的策略与方法。

软件构造（Software Construction）指通过编码、验证、单元测试、集成测试和排错的组合，具体创建一个可以工作的、有意义的软件，包括软件构造基础、管理构造、实践考虑。

软件测试（Software Testing）是在有限的测试用例集合上，根据期望的行为对程序的行为进行的动态验证过程，包括软件测试基础和测试级别、测试技术、测试需求、与测试相关的度量、测试过程。

软件维护（Software Maintenance），即软件维护基础、软件维护的关键问题、维护过程、维护技术。软件一旦投入运行，就可能出现异常，运行环境可能发生改变，用户会提出新的需求。软件生命周期中的软件维护阶段从软件交付时开始，但是实际的维护活动出现得还要早。

软件配置管理（Software Configuration Management）是为了系统地控制配置的变更和维护在整个系统生命周期中的完整性和可追踪性，而标志软件在时间上不同点的配置的技术。软件配置管理包括软件配置过程管理、软件配置标志、软件配置控制、软件配置状态统计、软件配置审核、软件发行管理和交付。

软件工程管理（Software Engineering Management），包括启动和范围定义、软件项目计划、软件项目实施、评审与评价、关闭、软件工程度量。虽然度量是所有知识域的一个重要方面，但是这里涉及的是对程序的度量。

软件工程过程（Software Engineering Process）涉及软件工程过程本身的定义、实现、评定、度量、管理、变更和改进。软件工程过程包括过程实施与改变、过程定义、过程评定、过程和产品度量。

软件工程工具与方法（Software Engineering Tool and Method），包括软件工程工具、软件工程方法。

软件质量（Software Quality）包括软件质量基础、软件质量过程、实践考虑。是在处理跨越软件生命周期过程的软件质量的考虑。软件质量在软件工程中无处不在，其他知识领域也涉及质量问题。

作为开发与维护的指导，软件工程的基本原理包括：用分阶段的生命周期计划严格管理；坚持进行阶段评审；实行严格的产品控制；采用现代程序设计技术；结果应能清楚地审查；开发小组的人员应该少而精；承认不断改进软件工程实践的必要性。

1.2　系统工程与统一建模语言

1.2.1　系统工程

系统工程是为了更好地达到系统目标，对系统的构成要素、组织结构、信息流动和控制机构等进行分析与设计的技术。针对不同的领域，系统工程有着不同的实现方法，如商业过程工程（Business Process Engineering）、产品工程（Product Engineering）等。系统工程的目的是使人们能够确保在正确的时间使用正确的方法在做正确的事情。

系统工程是用定量和定性相结合的系统思想和方法处理大型复杂系统的问题。系统由一系列相关元素的合理组织能够完成既定的任务或目标，如构成计算机系统的元素可以是硬件、软件、人员等，软件子系统又可以是程序、数据、文档等。所以，系统很自然的一种构成方式就是层次方式。系统分析的常用方法也是层次分析方法，它将问题分解为不同的组成因素，并按照因素间

的相互关联影响以及隶属关系将因素按不同层次聚集组合，形成一个多层次的分析结构模型。产品工程的层次结构如图 1.2 所示。

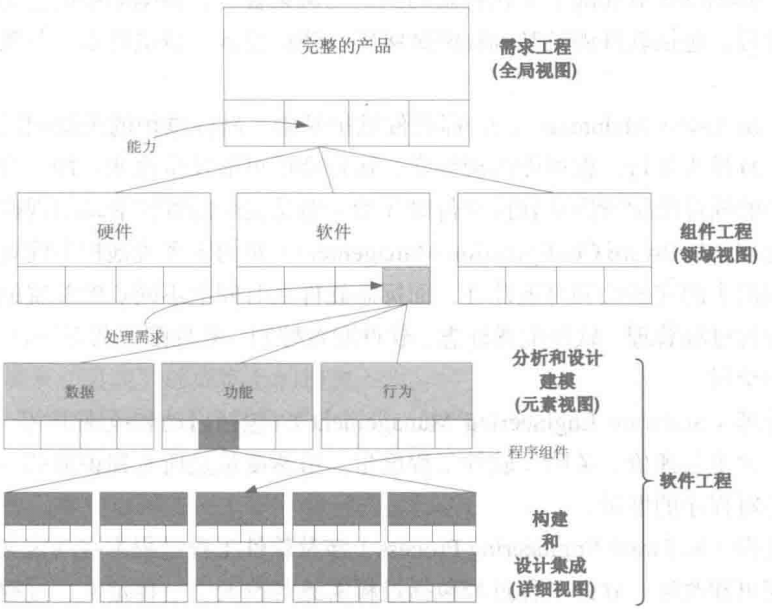


图 1.2　产品工程的层次结构[①]

模型的最高层表示领域目标，即层次分析要达到的总目标；中间层包括规范层和指标层，表示采取某一方案来实现预定总目标所涉及的中间环节；最底层表示要采用的解决问题的各种措施、策略、方案等。

模型能更好地体现出人们对系统的理解和驾驭能力。由于系统工程本身的层次特点，系统模型本质上也是层次化的。对应系统工程的不同层次，相应的模型会被创建和使用，如需求工程中的模型要能体现出对系统宏观上的理解，下层模型要能明确具体子系统的需求。随着对系统的理解，工程化的规范会被引入产生更细致的模型。

统一建模语言（UML）提供了一整套对系统建模的基础设施，包括模型的表示及建模的方法等，可以适用不同的系统层次。本书后续的章节中，我们会在软件工程领域中介绍相关的模型，包括传统的模型与面向对象模型。

1.2.2　统一建模语言

统一建模语言（Unified Model Language，UML）是继 20 世纪 80 年代末、90 年代初面向对象分析与设计方法（OOA&D）出现后面向对象领域又一研究与讨论的热点。UML 方法统一了 Booch、Rumbaugh（OMT）及 Jacobson 方法，并在此基础上进行了标准化，如今已经成为对象管理组织（Object Management Group）的标准之一。

统一建模语言顾名思义是一种语言，或者说是一种工具，而不是一种方法。UML 致力于分析设计的描述，其表述形式以图形的表现方式为主，但其描述形式本质上仍归为非正式（Informal）方式，这区别于其他一些形式化的正式描述方法。事实上，UML 是对 OOA&D 方法的分析设计结果的展现。

① 摘自《软件工程——实践者之路》

历史上 3 位在软件工程领域有着杰出贡献的人对 UML 的发展起着主要的推动作用，他们是 Grady Booch、James Rumbaugh 和 Ivar Jcobson，这 3 个人被誉为“三朋友”（The three amigos）。Grady Booch 是面向对象方法的最早倡导者之一，提出了面向对象软件工程的概念，他在早期与 Rational 软件公司研发 Ada 系统时做了大量工作，并由此提出了适合系统设计与构造的 Booch 方法。James Rumbaugh 一开始在 General Electric 公司曾经有一个研发团队，他们使用一种对象建模技术（OMT），这种方法中的表示符号独立于语言和模型并贯穿软件开发的各个阶段，实现了阶段间的平滑过渡，适合于以数据为中心的信息系统。Ivar Jacobson 早期在 Objectory 公司任职，在很多实际项目中积累了丰富的经验，提出了用例（Use case）的概念，进而提出了 OOSE 方法。OOSE 以用例为中心，进行系统需求的获取、分析以及高层设计等开发活动，适合支持商业工程的需求分析。

在 20 世纪 90 年代中期，以上 3 位分别代表着不同的学派，各自的理论也比较完善，但各有优缺点。其实，当时还有很多其他学派，如 Shlaer、Mellor、Coad 及 Yourdon 等，他们采用不同的符号进行类与对象的表示及关联，甚至出现了相似符号在不同模型中表示意义不尽相同的现象，分别自成体系，造成了混淆，不利于大规模的软件开发活动。面对几十种不同的建模语言，这些不同的面向对象的表示方法及相关的方法论形成了一种群雄争霸的时代，历史被称为是“方法的战争（Method Wars）”。

终于在 1995 年，Booch 与 Rumbaugh 合作，他们将其方法合并为统一方法，并公开发表了 0.8 版本，随后他们又联合 Jacobson，加入了他的用例思想。1996 年，3 人共同为他们的新方法 UML 努力，并命名为 UML 0.9。

但“方法的战争”并没有就此结束。1997 年 1 月，不同的组织和机构都向 OMG 提交了各自有关模型交换的草案，主要针对元模型与一些可选的表示。3 人所在的 Rational 公司也将此时的 UML 1.0 版本向 OMG 进行了提交，随后经过一段时间的工作和修改，取长补短，最终 OMG 选择了 UML 1.1 版本作为 OMG 的标准。从此，UML 又经过不断的修订，UML 1.3 在 1999 年成为当时的官方版本，也是在 UML 历史中最有意义的里程碑式的一个版本。2005 年 7 月，OMG 发布了 UML 2.0 版本，对 UML 1.x 版本进行了更新和扩充。目前，UML 也成为国际标准化组织（ISO）的标准之一。这些都决定了 UML 在软件开发建模领域中唯我独尊的地位。

一、统一建模语言功能

统一建模语言将软件开发中的语言表示与过程进行了分离，具有如下重要的功能：可视化（Visualization）、规格说明（Specification）、构造（Constructing）和文档化（Documenting）。下面分别对其进行说明。

1. 可视化

可视化能促进对问题的理解和解决，方便熟悉 UML 的设计师彼此交流和沟通。以此为基础，可以较容易地发现设计草图中可能的逻辑错误，保证软件能保质保量交付。

2. 规格说明

对一个系统的规格说明，应当通过一种通用的、精确的、没有歧义的通信机制进行。UML 适合这种说明工作。它使得在编码前，一些重要的决定得以表示和决定，使得开发人员达成共识，能对后续软件的开发过程进行指导，能够提高软件的开发质量，降低开发成本。

3. 构造

按照 UML 的语法规则，使用软件工具对模型进行解释和说明，将模型映射到某种计算机语言的实现，可大大加快系统建模和实现的速度。

在实现的过程中，根据设计合理调配资源识别可复用的组件，高效实现复用程度，降低开发成本。

4. 文档化

使用 UML 设计可以同时生成系统设计文档。这些专业化的设计文档资料可以帮助开发人员节省开发时间，快速熟悉及理解系统，起到人与系统之间的桥梁作用，达到事半功倍的效果。

二、统一建模语言的模型

UML 是提供给用户高层建模的方法，是针对用户需求的高层抽象，具体表现为描述组件的构成及联系，给人一种高屋建瓴的效果。代码也能够描述一种模型，但是具体的、底层的，如果其他人想要了解设计思想，必须要读懂代码，甚至可能要先去学习一门新的语言。另一种表示方法是使用自然语言的描述，但自然语言具有歧义及含糊不清的弱点。

UML 具有简单的表示法（Notation），而且其定义是规范和严谨的。其次，UML 的表示法要求必须简短，以便容易学习，而且其含义必须有明确的定义（在 UML 中，这种定义是通过元模型描述的）。

UML 2.0 的构成及与 UML 1.x 的比较：UML 1.x 明确了沟通设计、传达设计要点，以至捕获需求，映射到软件的解决方案。系统建模（不仅仅是软件建模）也能受益 UML。UML 2.0 经过体系的重建，克服 UML1.x 的过于复杂、脆弱和难以扩展等缺点，引进了一些新的图模型，以便扩展语言，使其能够支持最新的最佳实践。具体包括以下模型。

（1）用例图：用于表示系统与使用者（或其他外部系统）之间的交互，也有助于将需求映射到系统。

（2）活动图：用于表示系统中顺序和平行的活动。

（3）类图：用于表示类、接口及其间的关系。

（4）对象图：用于表示类图中定义的类的对象实例，其配置对系统很重要。

（5）顺序图：用于表示重要的对象之间的互动顺序。

（6）通信图：用于表示对象交互的方法和需要支持交互的连接。

（7）时序图：用于表示重点对象之间的交互时间安排。

（8）交互概况图：用于将顺序图、通信图和时序图收集到一起，以捕捉系统中发生的重要交互情况。

（9）组成结构图：用于表示类或组件的内部，可以在特定的上下文中描述类间的关系。

（10）组件图：用于表示系统内的重要组件和彼此间交互所用的接口。

（11）包图：用于表示类与组件群组的分级组织。

（12）状态机图：用于表示整个生命周期中对象的状态和可以改变状态的事件。

（13）部署图：用于表示系统最终怎样被部署到真实的世界中。

三、视图

描述系统的所有元素的集合及元素之间的关系，构成了模型。图是通往模型的窗口，特定的图会显示模型的某些部分，但不必显示模型的全部，图构成了 UML 的视图，每个视图捕获系统的一个方面。

有很多方法可以把 UML 的图分解为捕捉系统特定方面的透视图或视图，其中 Kruchten 的 4+1 视图定义如下：

1. 逻辑视图

逻辑视图（Logical View）描述系统组成部件的抽象表示，用来对系统由什么组成和各组成部

件之间如何结合进行建模，包括类图、包图等。

2. 进程视图

进程视图（Process View）描述系统内的进程结构，尤其在具体化系统内必须发生的内容时更为有益，包括顺序图、通信图、状态图等。

3. 开发视图

开发视图（Development View）是描述系统的部件如何组成模块和组件，可用来对系统体系结构内部的构成层次进行管理，包括组件图。

4. 物理视图

物理视图（Physics View）描述前 3 种视图描述的系统设计如何落实到真实世界的实体中。这种视图中的模型显示设计的抽象部分如何映射到最终部署的系统中，包括部署图。

5. 用例视图

用例视图（Use Case View）描述根据外部世界为系统建模时系统的功能性。这种视图描述系统应该做什么。所有，其他视图都依靠用例来引导和粘合。这也是这种模型称为“4+1”视图的原因，包括用例图、活动图等。

1.3　软件工程开发方法

软件工程在软件开发过程中引入了一整套相关的技术及规范，其主要包含方法、工具及过程 3 个要素。方法是完成软件开发各项任务的技术，主要回答“如何做”；工具是为方法的运用提供自动或半自动的软件支撑环境，回答“用什么做”；过程则是为了获得高质量的软件要完成的一系列任务的框架，规定完成各项任务的步骤，回答“如何控制、协调、保证质量”。随着软件工程的发展及技术的不断进步，软件工程开发方法从人们分析问题角度和方式的不同，可以笼统地分为传统方法和面向对象方法。

1.3.1　传统方法

传统的开发方法又叫做结构化的方法，其是一种静态的思想，将软件开发过程划分成若干个阶段，并规定每个阶段必须完成的任务，各阶段之间具有某种顺序性。传统方法体现出对于复杂问题“分而治之”的策略，但其主要问题是缺少灵活性，规范中缺少应对各种未预料变化的能力，而这些变化却是在实际开发中无法避免的。因此，当软件规模比较大，尤其是开发的早期需求比较模糊或者经常变化时，这种方法往往会导致软件开发不成功。即使开发成功，维护起来通常也比较困难，会增加系统的总成本。

1.3.2　面向对象方法

面向对象方法是一种动态的思想，其出发点和基本原则是尽可能模拟人类习惯的思维方式，将现实世界中的实体抽象为对象（Object），对象中同时封装了实体的静态属性和动态方法。面向对象分析设计的方式使得业务领域中实体及实体之间的关系与对象及其关系保持一致，做到了概念层与逻辑层的相互协调，更要强调的是各种逻辑关系在结构上的稳定性[①]，通过稳定的结构来提

① 这种稳定性可以通过 UML 的类图进行表达。

高应对各种变化的能力。因此，它的开发过程可以是一个主动多次迭代的演化过程，保证了开发阶段间的平滑（无缝）过渡，降低了模型的复杂性，提高了可理解性及应对各种变化的能力，从而简化了软件的开发和维护工作。

技术上，对象融合了数据及在数据之上的操作，所有的对象按照类（Class）进行划分，类与类之间可以构成“继承”的层次关系，对象之间互相联系是通过消息机制实现的，确保了对信息的封装性，使得对象之间更为独立。

同时，面向对象的分析过程既包含了由特殊到一般的归纳思维过程，也有由一般到特殊的演绎思维过程，而且对象是更为独立的实体，可以更好地进行“重用”。

1.3.3 理解两种开发方法

以上对传统方法和面向对象方法进行了介绍，下面通过一个具体的例子进行说明并辅助理解。后续章节还会对这个例子中涉及的具体方法和模型做更详细的叙述。

考虑日常生活中一个熟悉的应用场景“餐馆就餐”。传统的思维方式是一种过程化的方式，即将整个就餐过程划分成许多子过程，每个子过程对应不同的处理流程，数据在这些子过程中流动和处理，因此这种方法又叫做面向数据流的方法。图 1.3 形象地描述了运用两种不同的思维方式产生的不同分析方法。图中的下半部描述的是传统的过程化方法的分析图[①]。整个就餐过程划分为点菜、备料、烧菜、上菜 4 个子过程：首先顾客借助菜单在点菜过程中表达就餐意愿并记录在点菜清单中，后厨根据点菜清单进行菜品的备料和烧菜，上菜后需要对点菜清单做适当的标记。通过这样的分析过程，整个业务流程被描述和理解，并会在进一步设计中按照系统工程的层次方法对这些子过程进行更加具体化及技术化的展开。

面向对象的思维方式是以人类对现实世界的理解为出发点，对领域中的实体进行抽象，因此简化了分析的难度并增加了可理解性，同时保证了分析模型结构的稳定性，提高了应对变化的能力。图 1.3 中的上半部分是面向对象的分析方式，通过业务领域的理解抽象出 3 个主要的对象，即顾客、服务员和厨师。图 1.3 中标明了各个对象具有的服务能力，如服务员对象主要是对顾客提供点菜和上菜的服务。值得一提的是，厨师和顾客对象中都具有一个“品尝”的服务，但这个服务在两个对象中的作用是不同的。厨师的品尝服务前面有一个减号标识，表明这是一个私有服务——仅在本对象内使用（烧菜需要的动作）。顾客对象中的品尝则是一个公有服务。

对象之间的关系是保证整个分析图结构稳定的重要元素之一。图 1.3 中，顾客与服务员、服务员与厨师之间的箭头表明了对象之间具有的联系，这种联系在这个特定的业务领域（餐馆就餐）中是很自然的，与我们日常的就餐场景一致，而图 1.3 中顾客与厨师之间并没有任何联系，所以其具有较强的稳定性及可理解性。

如果用户此时对需求提出了新的修改建议：假设需要当前这个就餐系统能够“礼貌待客”。需求变化的响应对软件开发者来说一直是一个比较头疼的问题，应对需求变化的能力体现出的是设计师的素质和设计质量。针对用户具体的需求变化，传统的方法中必须首先要隔离出需要修订的位置——服务员与顾客之间的点菜和上菜两个过程需要做修改。不幸的是，上述隔离过程需要设计师理解整个业务流程，对于一些较大型的系统，往往这是很困难的一件事情。

面向对象过程在应对这一变化时是比较简单和直接的，由于其分析结果就是对现实世界的忠实反映，所以可以知道与礼貌待客直接相关的对象就是服务员，其他的对象根本不需要修改，因

① 此图的名字叫做数据流图（Data Flow Diagram，DFD）。

此也不需要去深入理解它们。由于对象的封装性，我们就可以直接把修改限定在一个特定的范围，模块化的优势也就体现出来。具体的，在本例中可以在服务员对象中添加一个私有的“问候”服务，并在其点菜和上菜服务中使用此服务即可，其他地方以及其他对象，尤其是对象之间的关系结构不用做任何变化，“隔离变化、应对变化、以不变应万变”正是面向对象方法优势的体现。

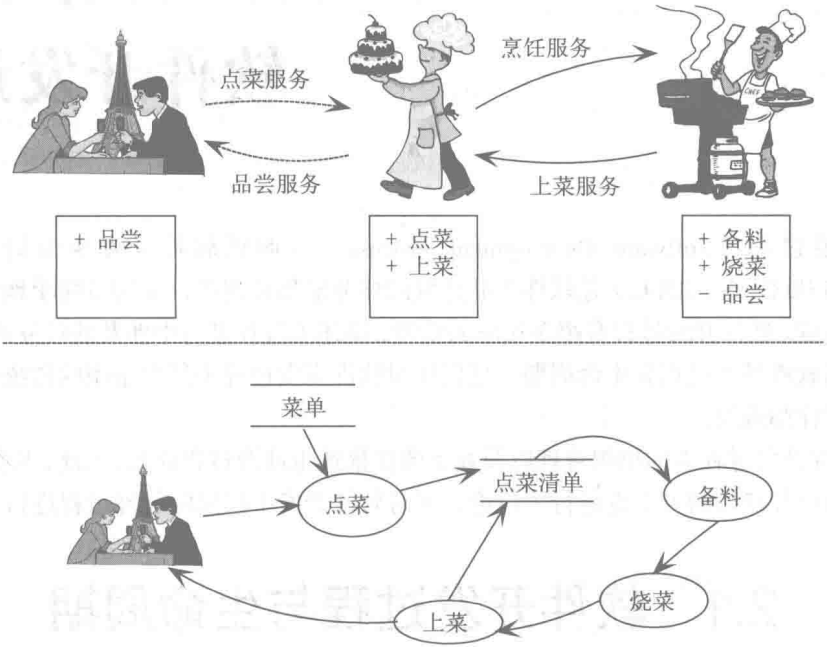


图 1.3　传统方法与面向对象方法的分析比较

以上简单介绍了传统方法和面向对象方法的特点及其比较。面向对象方法无论是在理念上，还是在实际开发过程中，都比传统过程化的方法具有优势，这也是为什么面向对象方法在工业界中越来越成为主流的开发方法。即使这样，也不能说我们可以完全摒弃传统的方法。作为面向对象方法的补充，传统方法中的一些经典的分析和设计的方法和过程目前仍然有着广泛的应用。

习　题

1. 软件工程主要包括那些内容？
2. 面向对象方法优于传统方法的根本在哪里？可否借助图 1.3 或其他实例给出自己的理解？
3. UML 包含哪些重要的模型，如何构成 4+1 视图？

第2章 软件开发过程

软件开发过程（Software Development Process）又叫做软件开发生命周期（Software Development Life Cycle, SDLC）是软件产品开发的任务框架和规范，又可以简单地称为软件生命周期及软件过程。软件开发过程有很多相应的模型，描述了过程进行中涉及的任务或活动的方法。本章首先介绍软件开发过程和生命周期，然后针对软件开发过程中经常采用的传统模型和敏捷模型分别进行了详细说明。

另外，软件开发过程中的组织管理以及业务流程描述也涉及过程建模，因此，本章对使用 UML 的活动图进行过程建模的方法也进行了讨论，并对软件开发中的风险管理过程进行了概括。

2.1 软件开发过程与生命周期

软件开发过程与生命周期模型相比，通常，生命周期模型更一般化（general），而软件开发过程模型则更具体化（specific）。比如，多种具体的软件开发过程都属于螺旋生命周期模型。国际标准 ISO/IEC 12207 是软件生命周期过程的国际标准，旨在提供一套软件开发与维护过程中涉及的各种任务定义的标准，如软件生命周期的选择、实现与监控等。

可重复的、可预测的过程能够提升软件生产的效率和质量。过程改进是对过程本身进行不断的提高，引入有效的开发经验，是质量保证的重要环节之一。软件过程小组在过程改进中具有重要作用，提供给软件开发人员统一的标准的开发原则，确保每个成员都在做正确的事情，步调一致，充分协调各开发人员、开发小组，通过过程控制的方法，保证软件产品的质量。

因此，软件生命周期是软件开发的宏观上的框架，软件过程则涉及软件开发的流程等管理细节，在框架稳定的前提下允许对软件过程进行裁剪。软件生命周期与软件过程如图 2.1 所示。

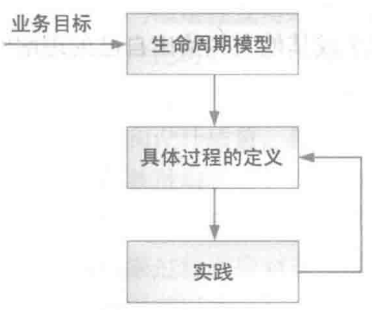


图 2.1 软件生命周期与软件过程

软件工程中的过程管理主要采用“分而治之”的思想，即将整个软件的生命周期划分成软件定义、软件开发和运行维护 3 个主要的时期，每个时期再细分为具体的阶段，分别对应明确的任务。这样做的目的是使规模大、结构复杂和管理复杂的软件开发变得容易控制和管理。通常，软件生命周期包括可行性分析与开发计划、需求分析、软件设计（概要设计和详细设计）、编码、软件测试、软件维护等阶段。