UML 面向对象分析

吴 际 金茂忠 编著

北京航空航天大学出版社

内容简介

UML 在 1997 年被国际对象管理组织 OMG 采纳为面向对象建模语言的国际标准，这为 UML 在实践中的推广奠定了坚实的基础。本书主要集中于讨论如何使用 UML 进行面向对象分析，包括如何使用 UML 的静态建模机制和动态建模机制在系统分析阶段建立系统分析模型。面向对象分析本身既涉及到系统的静态组成层面，又关注系统的动态行为，因此本书在 UML 静态建模机制和动态建模机制的基础上，通过完整的实例研究，就一个系统如何使用 UML 进行面向对象分析展示讨论。

本书主要可供从事面向对象分析的广大开发人员参考，也可以作为计算机专业高年级本科生和研究生学习 UML 的教材和参考书。

图书在版编目(CIP)数据

UML 面向对象分析/吴际等编著. —北京：北京航空航天大学出版社，2001.12

ISBN 7-81077-116-7

Ⅰ. U… Ⅱ. 吴… Ⅲ. 面向对象语言，UML—程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2001)第 079985 号

UML 面向对象分析

吴 际 金茂忠 编著

责任编辑 刘宝俊

责任校对 戚 爽

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(100083) 发行部电话：82317024 传真：82328026

http://www.buaapress.com.cn

E-mail:pressell@publica.bj.cninfo.net

河北省涿州市新华印刷厂印装 各地书店经销

*

开本：787×1092 1/16 印张：12.5 字数：320 字

2002 年 1 月第 1 版 2002 年 1 月第 1 次印刷 印数：4000 册

ISBN 7-81077-116-7/TP·065 定价：17.00 元

前　言

随着20世纪90年代面向对象技术的广泛应用，人们发现仅仅在代码阶段使用面向对象的概念是不够的，需要在分析和设计阶段就使用面向对象技术，于是面向对象分析和面向对象设计技术迅速发展起来，并得到了工业界和研究学者们的一致认可。

UML在1997年被国际对象管理组织OMG采纳为面向对象建模语言的国际标准，这为UML在实践中的推广奠定了坚实的基础。实际上早在被采纳为国际标准之前，UML已经在工业界获得了广泛的引用，如Booch、OMT和OOSE方法。在UML被接纳为国际标准后，国内对UML的研究逐渐深入，并在实践中尝试使用它。北京航空航天大学是较早进行UML教学和科研实践的单位之一，在硕士研究生的学位课"软件工程"中用了很大的篇幅对UML作了系统的介绍，同时北京航空航天大学软件工程研究所开发了UML开发支撑环境的教学版UML_Designer。不仅如此，软件所的多个科研项目均采用UML和相应的支撑工具Rose进行系统的分析和设计。我们在实践过程中遇到了很多问题，如在用例图中识别多少用例合适，类图应该如何组织，如何建立活动图模型等，这些问题在UML的规格描述文本中根本找不到现成的答案。本书作者不仅参与了北京航空航天大学的UML教学和对外的UML培训活动，还亲自在多个项目中使用UML进行面向对象分析和设计，这些都为本书的编写和出版积累了大量宝贵的实践经验。

目前国内有关UML的出版物主要集中于介绍UML建模语言本身和UML建模环境的使用两方面，基本上没有回答(或很少触及)如何使用UML这个关键问题。本书正是基于这种现状从实践的角度来试图回答这个问题。限于笔者的经验和本书的篇幅，本书主要集中于讨论如何使用UML进行面向对象分析，包括如何使用UML的静态建模机制和动态建模机制在系统分析阶段建立系统分析模型。UML的静态建模机制主要包括用例图、类图和包图，而包主要用来实现模型的规模管理；UML的动态建模机制主要包括活动图、状态图和交互图。然而面向对象分析本身既涉及到系统的静态组成层面，又要关注系统的动态行为，因此本书在UML静态建模机制和动态建模机制的基础上通过两个大型的实例研究，就一个系统如何使用UML进行面向对象分析展开讨论。

本书在保持与UML1.1描述一致的基础上，尽量不去介绍UML各个模型层次复杂的语义和UML复杂的形式化描述，而是专注于对UML重要概念的把握和理解，包括用例、类、关联和状态等，并通过大量的例子阐述如何在实践中运用UML，这在本书的几乎各个章节都可以看到，因此具有很强的实践针对性是本书的重要特点。

本书采用循序渐进、由浅入深、从整体到局部再回到整体的方式(读完本书读者会发现这正好就是一种在实践中非常有效的UML建模方式)介绍如何在实践中使用UML。在前三章通过对软件、软件开发过程、需求分析、面向对象等重要概念的介绍为读者对后续章节的阅读进行概念的准备和温习；从第四章到第六章依次介绍UML的形成过程，基本概念和静态、动态建模机制，在第五章和第六章读者可以发现有大量的面向对象分析实例；第七章介绍面向对象分析模型的测试，期望通过本章使分析人员建立明确的软件质量概念；第八章则完全从实践的角度介绍Rose和UML_Designer，相信一定会对初次使用这些工具的读者有所帮助；最后本书用两章介绍两个完整的面向对象分析实例，包括建立用例模型、类模型、活动模型和交互

模型等。所选的两个实例在实践中都具有较强的代表性，其中一个有关信息管理，另一个则有关嵌入式系统开发。

北京航空航天大学的刘超教授针对书中的内容提出了很多有益的建议，北京航天部二院的汤铭端研究员对本书的编写给予了大量的支持并就书中的内容提出了很多富有建设性的意见，在此对他们表示最真挚的谢意。还要感谢李燕虹同志，感谢她一丝不苟的录入工作。

限于编者水平的局限性，书中不可避免存在一些问题和不足，因此真诚欢迎专家和广大读者批评指正。同时，由于 UML 本身是一个新的事物，其中的一些概念、术语在国内还没有达成一致的理解和中文对应词，我们根据国内普遍采纳的中文译法和对 UML 概念的理解选用了我们认为较为合适的译法。本书中的译法一定还有欠妥之处，愿与广大读者一起讨论、磋商。

北京航空航天大学软件工程研究所
吴　际　金茂忠
2001 年 9 月

目　录

第一章 绪 论

1.1 软 件

大约在1970年左右，很少有人能够描述计算机软件的含义。那时人们所理解的软件几乎仅限于一段编制十分精巧、紧凑、用于解某个数学方程的程序，而且这种理解也只限于少数的一些人，很多人可能根本就没有接触过软件这个名词。然而随着计算机趋向个人化，计算机软件的概念也为越来越多的人所了解。计算机软件的教科书在开篇就会告诉我们，软件是：①能以预期的性能执行预期功能的一段指令（或程序代码）；②便于程序操纵信息的数据结构；③记录了程序的操作和使用的文档。

简单说，可以认为软件就是程序和文档的总称。可见，文档已经被提升到非常重要的位置。因为人们认识到软件是人的劳动产品，因此软件的编制需要一定的周期。在软件编制过程中所有与软件相关的阶段成果都应该归结为软件。这些阶段成果往往表现为各种文档，如需求文档、设计文档、程序文档、测试文档、用户手册等。

同人类的任何劳动产品一样，软件也具有某些特征。通过这些特征可以对软件有更好的了解。

人们在谈论软件的特征时通常把软件和硬件及其他劳动产品做比较。一般来说软件有以下几个特征：

（1）软件的产生通常经历一个开发或工程过程，而不是一般意义下的制造过程。

（2）软件不会出现“磨损、老化”现象。

大部分软件都是为用户定制的，而不是从已有的部件装配起来的。

尽管软件开发和硬件开发具有一定程度上的相似性，如都有分析、设计和实现阶段，但是二者之间却有很多本质的不同。硬件的产生需要经历设计和制造两个阶段，而且制造成本往往都很高；而软件一旦研制成功以后，其投产过程则仅仅是一个拷贝的过程，相对开发成本来说，其成本几乎可以忽略不计。

硬件和其他一些劳动产品（如建筑物、劳动工具等）在使用过程中都会出现“磨损、老化”现象。一开始这些劳动产品可以很好地完成它们的职责，但是经过一段时间以后，这些劳动产品就变得故障重重而不得不被淘汰（人们形象地称这种特征为“浴盆曲线”特征）。可是软件的情况却不是这样，一开始的时候软件的故障率可能会比较高，但随着在使用过程中维护时间的推移，软件的故障率会降低并保持在一个平稳的水平线上。

硬件和其他一些劳动产品的生产可以形象地比喻为“搭积木”，即利用很多已经存在的部件等按照设计好的框架进行搭建。软件开发者从来就没有享受过这种“奢华”的待遇，他们往往不得不从底层做起，设计并实现所有部件。不过我们认为随着先进开发方法学的出现和构件技术以及软件可重用性的提高，某些特定领域的软件开发最终有可能采用“搭积木”方式来实现。

1.2 软件工程

"软件工程"一词首次出现于1968年北大西洋公约组织会议(NATO)上,当时人们争论的焦点是高级编程语言要不要跳转(goto)指令。以后的相当长一段时间内,人们对软件工程的内涵进行了深入的讨论,也给出了多种定义。其中Fritz Bauer的定义为后来的关于软件工程内涵的讨论奠定了基础。Fritz Bauer认为,软件工程就是为了经济地获得能在真实的机器上可靠、有效运行的软件而建立和使用的好的工程规范(principles)。

该描述指出了软件工程的目标和手段,具体在技术上应该有什么样的保证却没有提到。但这仍不失为一个好的定义,因为它建立了软件工程内涵的描述框架。1993年IEEE给出了一个十分详尽的软件工程定义:①应用系统的、规范的、量化的,也即应用工程化的方法来开发、操作和维护软件;②关于①中的方法的研究。

一般来说,软件工程的研究对象包括过程(Process)、方法(Method)和工具(Tool)。其中的过程研究主要包括软件开发过程和维护过程的划分、效率,过程活动的定义和过程的控制等;方法研究包括各种开发方法、质量保证方法、测试方法等;工具的研究主要包括开发和使用相应的软件工程辅助工具,以及研究辅助工具对于软件工程的影响等。

1.3 软件生命周期

软件生命周期(Software life cycle)又称为软件生存期,是指从软件编制计划开始到软件在使用中消退的过程。一般可分为三个阶段:制定计划、开发和运行维护。其中开发又可进一步分为需求分析、设计、编码和测试。

1.3.1 制定计划(Planning)

确定待开发软件系统总的目标,给出功能、性能、接口等方面的总的要求。系统分析员和用户合作研究完成该软件任务的可行性,探讨解决问题的方案,并制定有关可用的资源(如资金、人力等)以及成本和开发进度等方面的计划。计划和可行性报告通常需要提交有关部门审查。

1.3.2 开发(Development)

软件开发是软件生命周期中的一个重要环节,一般可分解为如下活动。

1.3.2.1 需求分析(Requirements analysis)

软件最终要为用户服务,用户对于软件功能、性能等方面的需求也就决定了软件的功能。因此,分析员需要在开发开始的时候,就要充分地了解和论证软件所应具有的功能和性能。当然,这需要用户的密切合作。真正地理解了用户对于软件的各项要求后,分析员以标准的格式和描述方法,将已经明确了的用户要求记录下来形成软件需求规格说明。

由于软件是人开发出来的产品,因此错误在所难免。分析员在指定需求说明的同时应该指定出相应的测试条目。

1.3.2.2 软件设计(Design)

软件需求分析明确了待开发的软件应具有的功能和性能,软件设计就是在此基础上把需求细化为描述处理进程和算法的伪代码。设计形成的软件具有确定的体系结构,具有明确的模块划分,同时并有一定的手段描述清楚模块内部的逻辑。同样,软件设计结果必须按标准格式的规范描述方法记录下来,形成软件设计规格说明。

1.3.2.3 软件编码(Coding)

在软件设计规格说明基础上,可以运用某种编程语言,如C、C++、Java和Ada等,进行程序编写,以程序的形式实现软件的设计。通常程序通过编译就形成了可在机器上直接执行的机器代码(或通过解释执行的中间形式代码)。程序的编写应严格按照设计规格说明进行,当然也不排除程序员的个人发挥。

1.3.2.4 软件测试(Testing)

软件开发是人的创作活动,因而出现错误是正常的,也是不可避免的,因此需要在开发的每个阶段后进行测试。测试可以针对某个文档(如需求规格说明和设计规格说明),主要以审查的形式进行;也可以针对源程序,以执行方式进行;也可以走查方式(walkthrough)进行。如果要执行程序来测试,则需要设计测试用例来驱动程序的运行,以期发现错误。可以看出,测试是软件质量保证(Software quality assurance)的重要手段。

1.3.3 运行维护(Maintenance)

在通过了相应的某些测试后,软件就可以交付用户使用了。然而,由于软件测试不可能穷尽软件的所有输入情况,因此软件就不可避免地带着未被发现的故障(fault)进入使用。随着用户使用软件时间的延长,那些隐藏的故障就会暴露出来,形成失效(failure);同时用户在使用过程中还可能会遇到新的情况,这时就会对软件提出新的功能和性能要求;另外由于软件运行的外部环境(硬件、软件配置)或数据环境(数据库、数据格式、数据存储介质)可能发生变化,为了使软件能够适应这种变化,就要修改软件。当上述三种情况任一种出现时都要对软件进行维护,它们对应的维护分别叫改正性维护、完善性维护和适应性维护。

1.4 软件开发过程模型

软件过程描述了软件开发的一组具有时间约束的活动,通常包括需求分析、设计、编码、测试等活动,软件过程模型则以模型化的手段来定义开发的框架。

1.4.1 瀑布模型(Waterfall model)

瀑布模型指出软件开发包括制定计划、需求分析、设计、编码、测试和运行及维护。这些活动自上而下,相互间有固定的次序,如同瀑布流水的逐级下落,如图1.1所示。

实际上,上述各项活动之间并非具有完全的线性关系,每项活动都具有如下特征:

(1) 从上一项活动(如果有的话)接受该项活动的工作对象,作为输入。

(2) 利用这一输入实施该项活动应完成的内容。

(3) 提交该项活动的工作成果,作为输出传给下一项活动。

(4) 对该项活动的工作进行评审,如果得到确认则进行下一个活动,否则返回前项活动,

甚至更前项活动。

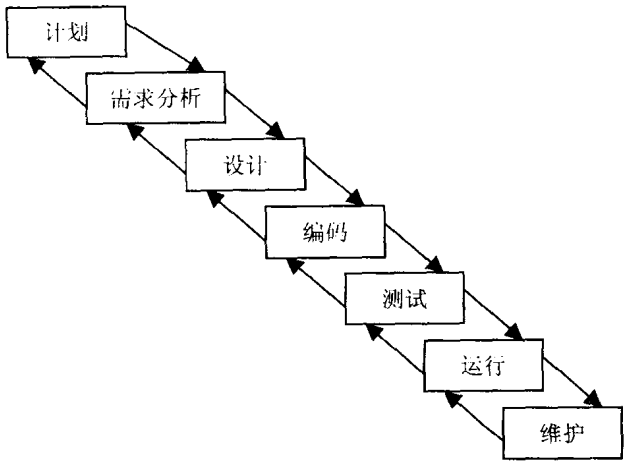

图 1.1　瀑布模型

1.4.2　螺旋模型(Spiral model)

螺旋模型将软件过程映射到坐标平面上，并把坐标平面划分为四个象限，分别表示制定计划、风险分析、实施工程和客户评估，如图 1.2 所示。

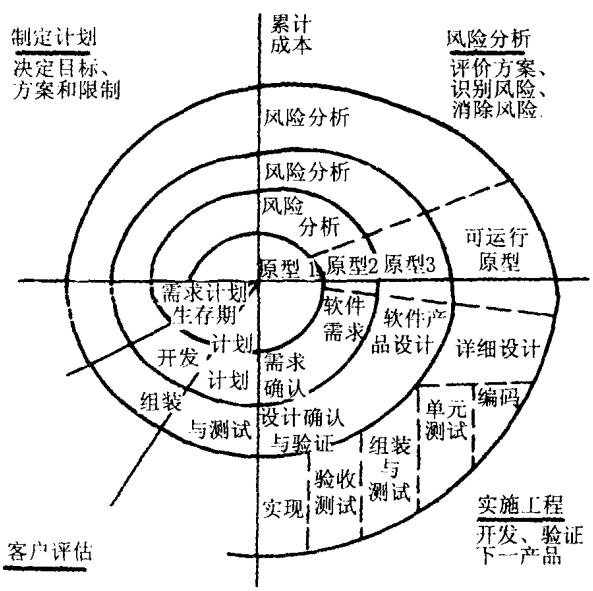

图 1.2　螺旋模型

(1) 制定计划：确定软件目标，选定实施方案，弄清项目开发的约束条件。

(2) 风险分析：分析所选方案，考虑如何识别和消除风险。

(3) 实施工程：实施软件开发。

(4) 客户评估：评价开发工作和工作产品，提出修改建议。

由图 1.2 中可看出，沿螺线自内而外旋转一圈就开发出一个新的软件版本(实质上是一个

原型)。螺线圈由内而外逐渐变大,软件原型也就更为完善,这是在客户评估基础上识别了更多的需求的结果。螺线一直外旋直至风险分析判断风险过大,以至开发方和用户无法承担,或者最终得到了一个用户所期望的系统。图 1.3 给出了螺旋模型的另一种视图。

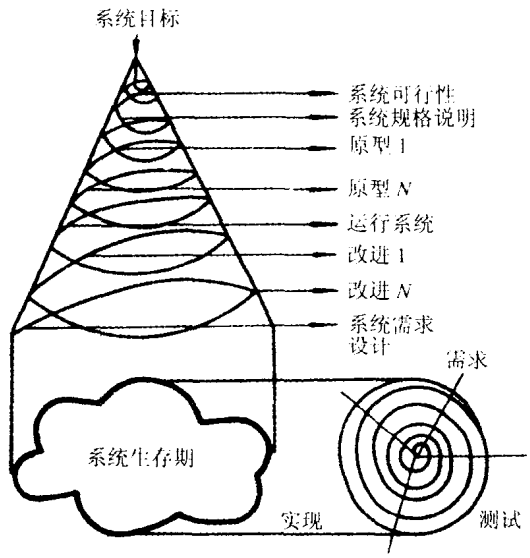

图 1.3 螺旋模型的另一种视图

1.4.3 喷泉模型(Fountain model)

喷泉模型体现了软件创建所应有的迭代和无间隙特征,如图 1.4 所示。这一模型表明了软件创建活动需多次重复,如在编码之前(实现之后)再次进行分析和设计,其间添加有关功能,使系统得以演化。可以看出每次重复都从分析开始,依次进行设计、实现……。应该指出,喷泉模型主要用于运行面向对象开发过程。

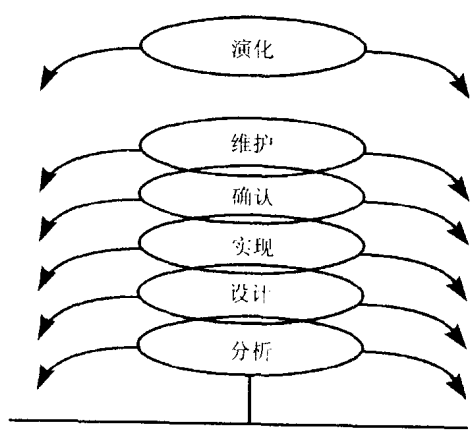

图 1.4 喷泉模型

1.4.4 增量模型(Incremental model)

增量模型的关键思想是自始至终有一个可以运行的软件版本。一开始实现软件的基本功

能，开发系统的一个初始子集。然后，根据这个软件版本逐步地添加另外一些功能，形成一个更为精细的版本。这样反复进行系统的增量开发，直至获得一个满意的产品。应该指出，增量模型实际指出了一个外围框架(infra framework)，它不限制增量开发过程中所使用的具体的过程模型和方法。

1.4.5 演化模型(Evolving model)

演化模型主要针对事先不能完整定义其需求的软件的开发。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的实现后能有效地提出反馈，以支持系统的最终设计和实现。软件开发人员根据用户提出的核心需求开发核心系统，并让用户试用这个核心系统；得到用户的有效反馈后，开发人员精化系统，增强系统的能力，然后再由用户试用。如此迭代下去，直至用户满意为止。演化模型中每一次的迭代实际上为整个系统增加一个可定义的、可管理的子集。

1.4.6 统一过程模型 RUP(Rational Unified Process)

统一过程模型由 Rational 公司提出，主要用来描述使用 UML 开发的过程。该过程有如下三个主要特点：

(1) 用例(Use-case)驱动的软件开发过程。

(2) 以体系结构(Architecture)为中心的过程。

(3) 迭代(Iterative)开发与增量(Incremental)开发相结合的过程。

统一软件开发过程使用用例来识别并描述需求。不仅如此，用例在整个开发过程中起着驱动的作用。在需求阶段，分析员使用用例建立需求模型；在设计阶段，设计人员根据用例进行设计，给出实现模型，在此过程中，开发人员还要经常把设计模型和实现模型与用例进行比较，评价；在测试阶段，用例是测试的依据，测试人员根据用例来设计测试用例(Testing cases)，从而验证软件是否正确地实现了用例。可见，统一开发过程从用例的建立开始，同时用例在整个过程中起着驱动的作用。但是用例的建立不是独立完成的，建立用例模型的时候要充分考虑体系结构的诸多因素，它们之间的关系是“鸡与鸡蛋”的关系。

体系结构体现了系统关键(或核心)的静态和动态特征，它对于软件就如同建筑物的结构(Architecture)对于建筑物一样。体系结构取决于由用户提出的业务领域的需要，这些需要在用例中得到具体描述。

一开始，体系结构师(Architect)只能建立一个较粗糙、不完整和大纲形式的体系结构，这时的体系结构不考虑某个具体的用例因素。然后，体系结构师根据已经由分析员建立的用例模型来更新体系结构。随着用例的不断被识别和变得稳定，更多的体系结构被识别。这反过来又将使更多的用例变得更稳定、成熟。

标准过程认为在实际中一个开发项目应该被划分为若干个迭代的子项目。在一次迭代过程中，开发者识别并描述相关的用例，根据选定的体系结构进行设计。同时开发者还要把设计按照构件的形式加以实现并需要验证开发出来的构件是否满足相应的用例，如果达到目标则进入下一个迭代过程。

迭代过程的划分通常考虑一组用例和那些重大的风险，这组用例会扩展当前开发出来的产品的可用性(Usability)。不同的迭代过程之间从开发出来的产品的角度考虑实际上形成了

增量。一个增量过程并不一定对应于一个迭代过程，它往往对应于几个迭代过程，即一个增量过程步骤可能由几个迭代过程步骤组成，如图 1.5 所示。

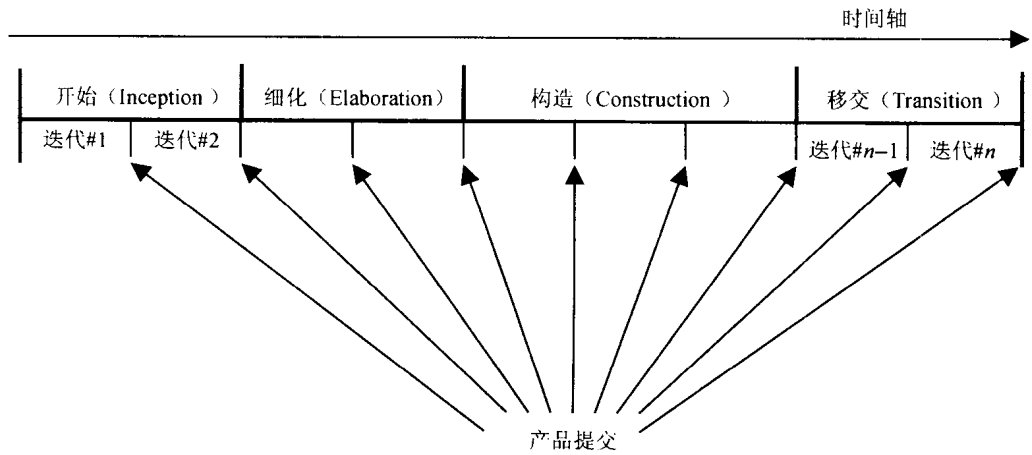

图 1.5　统一软件开发过程的阶段划分和迭代

本章小结

本章是读者阅读本书的知识准备和回顾，只有了解软件工程的一些基本概念才能够比较快地掌握 UML 中的一些概念。由于本书的重点是介绍 UML 的一些基本概念和应用 UML 进行面向对象分析时应该注意的问题，因此面向对象和需求分析概念的掌握就显得至关重要。本书的后续章节会陆续给予介绍。限于篇幅，本书在介绍这些概念时只能简单带过，读者可以参考其他一些著作，如 Pressman 的《软件工程实践》，Yourdon 的《面向对象分析》和《面向对象设计》等。

第二章　软件需求分析

在第一章我们指出软件需求分析在软件开发中起着至关重要的作用，它对于软件的质量往往具有决定性意义。本章主要介绍需求分析的概念、过程、需求分析文档格式和需求分析方法。

2.1　需求分析

需求分析是一组活动的总称，通常由系统分析员和软件用户共同完成。需求分析的任务是定义待开发的软件的功能、性能等指标，因此它通常包括这样一些活动：

(1) 了解用户的相关业务。分析员必须首先到用户的工作环境(或者待开发软件的预期使用环境)中去，通过用户的演示、讲解和有关文档了解相关业务，进行整理，并且还要和用户进行交流、协商。这是一个关键的活动，如果一个分析员还不能真正地理解用户的相关业务，那么他就无法知道用户到底想要软件帮他们做些什么。用户自然非常清楚他们自己的业务，但是他们往往不能正确、完整地表述他们对于软件的要求。不过如果某个软件实现的并不是他们想要的情况，他们会立刻发现。因此在了解用户的相关业务时，分析员还需要制定一些提问单和表格交给用户填写，以让他们说出那些用户自己无法说出的，但对软件又是重要的信息。

(2) 分析用户业务流程。分析员了解到的用户业务也许只是一些离散的业务活动，而业务流程是重要的信息，它往往指出了某些用于指导软件功能组织的信息。分析员将了解到的业务活动加以整理并按照这些活动所固有的次序形成业务流程。

(3) 了解用户对于软件的期望值。软件首先需要能正确地处理用户的业务，但是用户对于软件还有一些其他的要求，如使用的便利性、附带接管一些机械的日常活动(如每天的整理业务数据等)和一些性能要求等。用户对软件的这些需求有时虽然并不是直接和业务相关，但是对于用户而言却显得很重要。

(4) 整理用户要求。分析员需要独立地完成用户需求的整理并形成规范的需求规格说明。如果在整理过程中发现有疑点，就需要立刻和用户协商解决它；如果有些问题现在根本无法确定，那就只有暂时留在那儿(但是需要通知用户知道)，一旦时机成熟就要立刻解决它。

(5) 需求评审。分析员整理得到的需求分析规格说明必须要通过需求评审才能说明需求分析工作可以结束而进入设计(应该指出，在面向对象开发中，需求分析和设计之间并没有明确的界限，但通常也有相应的需求评审)。

2.2　需求分析过程

有些传统的开发模型要求将需求分析做充分后方进入设计；而有些则不这样，如快速原型法。这种方法要求首先给用户尽快做出软件原型演示，让用户指出哪里有错误，哪里缺了什么