软件工程基础

编著　赵一丁

北京邮电大学出版社
·北京·

内 容 简 介

本书全面系统地阐述了自动化领域软件工程的基本概念、基本技术和基本方法。全书共16章，主要内容包括：软件开发过程和软件开发模型、软件需求分析、软件测试和软件维护、软件工程管理、软件设计基础、UML建模语言、实时系统软件设计、工业自动化组态软件技术。组态软件技术的内容包括实时数据库设计和变量组态、图形界面设计、报表组态、设备驱动与管理、通讯组态、工控组态软件设计应用实例。附录包括软件文档格式、常用工控组态软件(力控PCAuto、西门子Win CC和MCGS)简介。

本书内容丰富，图文并茂，通俗易懂。本书可作为高等院校本科及专科自动化专业、测控专业、自控专业、机电专业、计算机相关专业的教材，也可供软件工程技术人员参考。

图书在版编目(CIP)数据

软件工程基础/赵一丁编著. —北京：北京邮电大学出版社，2006
ISBN 7-5635-1277-2

Ⅰ.软... Ⅱ.赵... Ⅲ.软件工程—高等学校—教材 Ⅳ.TP311.5

中国版本图书馆CIP数据核字(2006)第084839号

出版发行：北京邮电大学出版社
社　　址：北京市海淀区西土城路10号(100876)
北方营销中心：电话：010-62282185　传真：010-62283578
南方营销中心：电话：010-62282902　传真：010-62282735
E - mail：publish@bupt.edu.cn
经　　销：各地新华书店
印　　刷：
开　　本：787 mm×1 092 mm　1/16
印　　张：18.75
字　　数：439千字
印　　数：1—3 000册
版　　次：2006年9月第1版　2006年9月第1次印刷

ISBN 7-5635-1277-2/TP·238　　定价：28.00元

高等院校自动化新编系列教材

编 委 会

编 写 说 明

一本好的教材和一本好的书不同，一本好的书在于其内容的吸引力和情节的魅力，而一本好的教材不仅要对所介绍的科学知识表达清楚、准确，更重要的是在写作手法上能站在读者的立场上，帮助读者对教材的理解，形成知识链条，进而学会举一反三。基于这种考虑，在充分理解自动化专业培养目标和人才需求的前提下，我们规划了这套《高等院校自动化新编系列教材》。

本套系列教材共包括21册，在内容取舍划分上，认真分析了各门课程内容的相互关系和衔接，避免了不必要的重复，增加了一些新的内容。在知识结构设计上，保证专业知识完整性的同时，考虑了学生综合能力的培养，并为学生继续学习留有空间。在课程体系规划上，注意了前后知识的贯通，尽可能做到先开的课程为后续的课程提供基础和帮助，后续的课程为先开的课程提供应用的案例，以便于学生对自动化专业的理解。

《高等院校自动化新编系列教材》编委会

2005 年 8 月

前　言

软件工程作为一门有关软件开发的工程方法学，在软件开发中的指导意义与基础地位已经越来越多地得到了信息业界和自动化业界的高度重视。软件工程已经成为计算机及其相关专业的专业核心课程。

在自动化领域，随着嵌入式实时系统、工业控制计算机、PLC 和工业网络通讯应用的日益扩大，软件开发的规模也越来越大，对软件的可靠性和可维护性的要求也越来越高。采用软件工程的理论、方法和技术进行软件开发和软件项目管理，对自动化领域的软件开发同样是很重要的。

自动化领域的软件开发有其自身的特点。一是编程语言和开发工具多种多样。嵌入式系统的编程语言有汇编语言、C 和 C++ 语言、实时 Java (J2ME)；在通用 PC 计算机上的编程语言有 C++、Java、C#、Visual Basic 等；在 PLC 中采用的编程语言有梯形图、指令表等。开发工具有编程语言的开发工具和工控组态软件开发平台。二是对软件的可靠性要求很高。在许多自动控制系统中，软件缺陷可能会导致严重后果。三是自动化系统往往要与许多硬件设备连接，软件调试和软件测试更加复杂和困难。

在自动化领域，采用组态控制技术，以组态软件为软件平台进行软件开发越来越受到用户的青睐。组态软件能以灵活多样的组态方式（而不是编程方式）提供良好的用户开发界面和简捷的使用方法，其预先设置的各种软件模块可以非常容易地实现和完成工控层的各项功能，并能同时支持各种硬件厂家的计算机和 I/O 设备。组态软件与其支持的硬件设备一起，为用户提供自动化系统的软硬件整体解决方案。嵌入式软件开发也是组态控制技术应用的领域。

本书介绍了软件工程的基础知识、基本理论、基本方法和基本技术，重点介绍了自动化领域软件的开发方法和技术，包括实时系统软件设计和用工控组态软件平台进行软件开发。在内容的叙述上，以理论联系实际为原则，注重简明扼要，通俗易懂。由于软件工程是一门实践性很强的学科，书中提供了许多范例供读者参考。

本书共分 16 章。第 1 章软件工程的基本概念，第 2 章需求工程，第 3 章软件测试，第 4 章软件维护与软件进化，第 5 章软件工程管理，第 6 章软件设计基础，第 7 章 UML 建模语言，第 8 章实时软件设计基础，第 9 章组态软件

技术基础，第 10 章工控过程的图形画面设计，第 11 章实时数据库与变量组态，第 12 章输出报表组态，第 13 章通讯组态，第 14 章工控系统中的设备管理，第 15 章工控组态软件的通讯网络，第 16 章工控组态软件设计应用实例。每章后附有习题。

本书由赵一丁主编，聂钢主审。参加编写的有范立娜、赵玉倩、刘秀田、曹毅。本书第 1、8、9、14、15 章由赵一丁编写，第 3、4、5、6 章由范立娜编写，第 11、12、13、16 章由赵玉倩编写，第 2 章由刘秀田编写，第 7 章由曹毅编写。全书由赵一丁统稿，西安交通大学聂钢教授审阅全稿。聂教授对本书提出了许多宝贵意见和建议，对此表示衷心的感谢。在本书的编写过程中，我们还得到了汪晋宽教授和江南大学张秋菊教授的热情关心和指导，在此一起表示感谢。

本书参考教学时数为 56 学时，其中授课 46 学时，上机实习 10 学时。

限于编者水平，书中的缺点和错误在所难免，敬请广大读者批评指正。

编　者

2006 年 7 月

第 7 章　UML 建模语言

7.1　基于 UML 的软件开发过程

UML(Unified Modeling Language,统一建模语言)是一种建模语言,是第三代用来为面向对象开发系统的产品进行说明、可视化和编制文档的方法。它是由信息系统(IS,Information System)和面向对象领域的三位著名的方法学家 Grady Booch、James Rumbargh 和 Ivar Jacobson 提出的。UML 是一种建模语言而不是一种方法,因为 UML 并不包含对方法而言非常重要的对过程的定义这一部分。但是,UML 的设计者一直在将他们早先各自定义的过程进行合并,并于 1997 年将其合并后的过程命名为 Rational Objectory Process。后来他们又吸取了他人的经验,重新命名为 Rational Unified Process(简称为 Unified Process),并于 1998 年正式颁布。而软件开发涉及到多种因素,所以很难定义一种通用的过程,用以支持各种不同类型软件的开发。因此应当允许采用不同的过程。UML 设计者所定义的软件开发过程是一个公共的过程框架,对过程中应当具有的一些公共要素加以规范,同时给用户留出一定的空间,以便用户采用适用于自己项目的技术。

本节将就开发过程进行简要介绍,让读者了解使用 UML 方法时应当采用的典型过程。

7.1.1　过程概述

图 7-1 是开发过程的一个高层视图。这是一个迭代增量式的开发过程,采用这种方法,不是在项目结束时一次性提交软件,而是分块逐次开发和提交。构造阶段由多次开发组成。每一次开发都包含编码、测试和集成,所得产品应满足项目需求的某个子集,或提交给早期用户,或纯粹是内部提交。每一次迭代都包含了软件生命周期的所有阶段,即:分析、设计、实现和测试阶段。

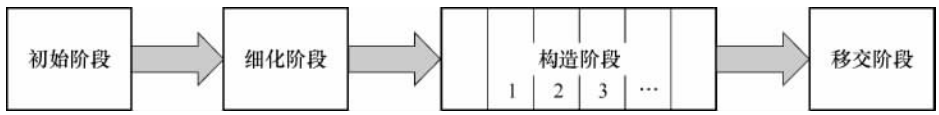

图 7-1　开发过程简图

为此,首先要做的工作是选择一些功能点,然后完成这些功能,随后再选择别的功能点,如此循环反复。显然,作这个计划需要时间。前两个阶段是初始阶段和细化阶段。在初始阶段需要考虑项目的效益,并确定项目的适用范围。这一阶段需要与项目出资方进行讨论;在细化阶段,需要收集更为详细的需求,进行高层分析和设计,并为构造阶段制订计划。

构造阶段是一个典型的迭代过程。其实,在每一个阶段,尤其在一个大的阶段中都存在迭代。

下面我们将分别介绍各个阶段的主要任务和目标。

7.1.2 初始阶段

初始阶段可以有多种形式。根据项目的大小进行区别,小项目时只用几句话即可讨论,大的项目则有可能需要几个月的时间来进行可行性分析。这一阶段需要考虑项目的商业属性,即粗略估计项目的费用和可能得到的效益。此外,还需要了解项目的范围,必要时需要作进一步分析以便确定项目的规模。

在这一阶段,应对需求有一个大概的了解。其中心的任务是要弄清这个项目的意义和价值,并确认是否正式启动这个项目。

7.1.3 细化阶段

在正式确认启动这个项目之后,就进入软件开发过程的细化阶段。在细化阶段,需要对问题有更详细的理解,包括:

- 实际要做什么
- 如何做
- 将采用什么技术

1. 风险分析与风险管理问题

这里,首先需要考虑的(也是最重要的)问题就是风险分析与风险管理。一般来说,风险可分为以下 4 类:

(1) 需求风险

系统的需求是什么? 最大的危险是建立了一个偏离用户需求的系统。因此在细化阶段,需要充分了解用户需求以及各需求的相对优先程度。

(2) 技术风险

为了了解你所面临的技术风险,不妨考虑以下问题:如果你将使用面向对象技术,那么你具备多少面向对象设计工作的经验?

(3) 技能风险

譬如,能否得到所需的技术人员和专家?

(4) 政策风险

是否存在一些政策性因素会影响项目的进行?

下面我们分别讨论各种风险的含义及其处理方法。虽然这些内容在软件开发的“细化阶段”进行阐述,但所讨论的内容对其他阶段也是普遍适用的。

2. 处理需求风险

需求风险是指项目的目标是否满足用户的需求方面所存在的风险。需求分析是项目开发的源头,自然有着非常重要的地位。在需求分析中,UML 采用的主要技术是用例分析技术,因此用例分析技术成为 UML 的核心。采用标准建模语言 UML 中的用例分析技术,是避免或是减少需求风险的主要途径。

一个用例是指用户为了达到目标而与系统进行的一次典型交互过程。例如：对于我们常用的字处理软件，用例可以是："将所选文本置为黑体"或"建立文档索引"等。细化阶段最重要的事情是列出系统的所有用例。在实际运作中，即使得不到所有用例，至少也应列出最重要的用例。因此，在这一阶段应安排开发人员与用户进行交流，以便收集用例。

由于用例不描述系统全貌，因此需求分析的另一项主要任务是要开发应用领域的概念模型。所谓领域的概念模型就是反映在商业运行过程中各种元素之间的联系。领域概念模型为对象模型的建立奠定了基础。

在 UML 推荐的方法中，应该采用不同的模型来描述系统开发的不同侧面。用域模型和用例来描述功能需求；用分析模型来描述针对某一具体应用时这些需求的内涵；用设计模型来描述系统工作的内部结构。设计模型是用来描述域对象的属性和用例中的行为。

在建立域模型时，有两个 UML 技术特别有用，它们是：

（1）类图

概念层上的类图有助于开发人员了解用户在其业务中使用的语言，因此，我们应当首先列出领域专家使用的概念，并画出这些概念之间的关系，经过适当整理，便可以得到建立在应用域上的类模型。

（2）活动图

通过描述工作流程来补充类图。运用活动图来进行这项工作的一个关键目的是鼓励发现系统中存在的并行过程。应当避免把实际的业务过程转换成不必要的串行工作。

领域建模有利于考察用例。当收集用例时，最好能请一个领域专家，并借助于概念层的类图和活动图，了解应用领域的需求和工作方式。最初的域模型只是一个框架，而非真正的高层模型。所谓"高层"是指抓住问题的本质而忽略各种细节的域模型。

域模型是随着对用例的了解逐渐建立起来的。当出现一个用例时，建模者应该分析这个用例，考虑它是否影响模型。如果影响，则应进一步分析这种影响是由于用例不妥还是域模型不妥所引起，然后做相应的调整。如果一个用例对于模型没有影响，则表明两者是一致的，但这个用例对完善这个领域模型不起作用，因此该用例可以暂时放在一边。

在细化阶段，根据国外不少顾问公司的经验，建模组应集中工作，直到建立起域模型。有时为了更直观准确地理解需求，有时还需要为复杂难懂的用例建立原型。原型法是理解复杂系统动态特性的一种极有价值的技术。但不是所有的用例都需要建立原型，大多数用例采用图形描述就够了。

3. 处理技术风险

在考虑技术风险时，最重要的工作是建立原型，尝试所选的技术方案是否可行。在此阶段，还应考虑体系结构的设计问题，考虑所需构件以及构造方法。这在设计分布系统时尤为重要。在这一阶段，还应考虑如何使设计更易于支持设计成分的维护和修改，这方面包括以下问题：

- 如果某一技术不能正常工作，将发生什么情况？
- 如果无法将两个构件连接起来，会发生什么情况？
- 如果某一部分出错，会出现什么情况？如何处理这种情况？

使用模型时，应认真考察用例中是否含有不利于设计的内容，为避免其中有华而不实的东西，应做深入的调查。在此过程中，需要采用大量 UML 技术来帮助刻画设计思想，并应写出需求文档，此时无需完全和精细，而是要简洁和准确。

4. 处理技能风险

技能风险是指项目实施者的素质是否满足项目要求方面所存在的风险。根据实际情况，采取自学、参加培训、组织一个学习小组一起学习和研讨等等方法都是很有效的，并且切实可行。在项目开发过程中，应当随时注意你缺乏技能或是经验的部分，并作出计划以便能取得所需要的经验。当然我们最终的目的是要提高项目实施者水平，避免或是减少风险。

综上所述，培训、聘请顾问、定期的学习与交流、有针对性的补缺等，都是提高项目实施者水平、避免或减少技能风险的行之有效的办法。

5. 政策风险

是否存在一些政策性文件会影响项目的进行？有时是需要加以考虑的。这里所指的政策性影响是广义的、多方面的，譬如：大到国际政治经济秩序、国家政策法规，小到开发方的发展规划或投资方的投资政策。此时会对项目的进展产生一些分歧意见。这时是需要重新讨论作出决策的。由于政策风险属于人文科学的范畴，在此不做进一步的阐述。下面回到软件开发细化阶段应该讨论的内容。

6. 基线体系结构

细化阶段最重要的结果之一是建立系统的基线体系结构。该体系结构是开发的基础，并作为下一阶段的蓝图。当然，在细化阶段建立的体系结构的细节会有一些变动，但不允许有太多的重大变更。该体系结构主要包括：

(1) 用例表，用于描述系统需求；

(2) 域模型，作为获取应用领域中的关键类的起点，反映了你对系统将要提供的业务和服务的理解；

(3) 技术平台，描述重要的实现技术以及技术间的协作和集成。

关于体系的稳定性，对不同的技术有不同的要求。

7. 细化阶段何时结束

根据国外研究报告所提供的数据，细化阶段通常需占项目总时间的五分之一。以下两个事件是细化阶段结束的标志：

(1) 开发人员能给出项目估算。最好对每个用例的估算能精确到人周。

(2) 考虑到所有风险，并制订了相应的对策和计划。下面我们对如何制订计划进一步讨论。

8. 计划

计划的实质是为构造阶段建立迭代开发序列，并将用例分派到各个迭代中。当所有用例都已分配到各次迭代中，并且各次迭代开发的开始日期都已确定下来之后，计划工作便可宣告完成。

从下面的讨论我们可以看到，用例是制订项目计划的基础。因此，计划阶段的第一步工作是对用例进行分类。下面的 3 条是对用例进行分类需要采用的技术。

(1) 用户应当为每一个用例标出其优先级。通常分为 3 级:哪些是必须首先实现的功能;哪些是在短期内可以没有的功能;哪些在较长的时间内可以没有的功能。

(2) 对于每一个用例,开发人员都应考虑相关的体系结构风险。这里,仍可采用 3 级分类:高风险;可能的风险;几乎不可能的风险。

(3) 开发人员还应评价自己对每个用例所需工作量的估算,称为进度风险。同样采用以下 3 级分类:确信自己对时间的估算;只能估计到人月;无法估算。

完成分类后,应估算每一个用例所要求的时间,精确到人周。在估算时,应考虑到分析、设计、编码、单元测试、集成以及编写文档所需要的时间,确保开发人员能充分理解所要求的用例。估算完成后,应考虑是否具备了作计划的条件。尤其要考察具有高进度风险的用例。如果大部分任务的进度计划都与那些具有高进度风险或者具有高体系结构风险的用例有关,则应做进一步的评审。只有在确定好用例并对用例进行正确分类之后,才能进入下一步工作。

计划阶段的第二步是确定每次迭代的开发周期。对于整个项目可以制订固定的迭代周期,以便能定期地提交产品。对于每次迭代,时间应足够长,以保证完成所要求的用例。具体的迭代周期长短应根据项目的大小、开发要求以及开发队伍的规模和经验水平等因素来确定。

至此便可考虑每次迭代开发能提供的工作量。在实践中,人们总结了很多经验估算方法。一种简单的计算方法是:首先假设每个开发人员的工作效率为 50%(因为一半时间要用于开发用例),将开发周期的长度乘以开发人员的个数,再乘以 1/2,其结果便是每次迭代开发所能提供的工作量。将所有用例所需的时间加起来,除以每次迭代的工作量,再加上 1 作为保险值,便得到完成该项目所需迭代次数的第一个估算值。一般整个项目的迭代次数不宜过多,以 3～5 次为宜。但是考虑到应当尽早开始集成和进行集成测试、提高开发的并行程度,开发组内部的迭代频度可能要大得多,并且在软件开发的不同阶段,频度上也会有很大的差异。当然,要想实现这种高度并行的开发过程,准确的计划和严格的管理是关键。

总之,在确定每次迭代的开发周期时,要权衡多方面的因素,最后对迭代周期、迭代次数和开发人员数目等主要因素作出合理的选择。再下一步是将用例分配到各次迭代中。具有高优先级、高体系结构风险和高进度风险的用例应尽早处理。不要把风险留到最后。

移交阶段所需的时间通常要占构造阶段所需时间的 10%～35%,主要用于调试和打包。如果对当前环境缺乏调试和打包经验,则可分配更长一点的时间。为应付意外事件,通常还需要根据风险程度,将构造时间增加 10%～25%。在制订交付计划时并不考虑这段时间,因为这是内部交付时间。在制订对外承诺的交付计划时应包括这段时间。

完成以上各项工作后,也就完成了计划的制订工作。该计划列出了每一次迭代所要完成的用例。由于该计划是开发者和用户共同协商完成的,我们将其称为“委托进度计划”。

由以上讨论可得,用例是制订项目计划的基础,这正是 UML 强调用例的另一重要原因。

7.1.4 构造阶段

构造阶段通过一系列迭代过程建造系统。每一次迭代开发都是一个小项目,需要对所要求的用例进行分析、设计、编码、测试和集成。完成一次迭代后应向用户演示,并完成系统测试,以表明所要求的用例已正确实现。

这种做法的目的是为了减少风险。如果困难被推到最后解决,则存在很大的潜在风险。有不少项目将测试和集成放在最后,而测试和集成是个大任务,常常比人们预计所需要的时间长得多。

一般来讲,测试分为单元测试和系统测试。单元测试的代码应由开发人员编写。这些测试应被归并成一些测试包,而且他们应测试到所有类的接口。系统测试的代码应由测试组开发,持黑盒子测试原理对系统进行测试,并力图发现所存在的错误。

构造阶段的迭代开发是增量式和重复进行的:

- 迭代开发在功能上是增量式的,每次迭代开发都是建立在已经开发的用例上的;
- 已有的程序代码在每次迭代时都可能会有部分修改,使其更为灵活,程序重组是迭代编码中非常有用的技术。

集成是一个连续的过程。首先,集成是每次迭代结束时的工作,但它不仅仅在每次迭代结束时才进行。开发应在每次重大改动时进行集成,每次集成完成后应重新进行全套的单元测试,从而确保回归测试的完整性。

1. 迭代开发和计划

对于每一次迭代开发,还可以做更为详细的计划。做计划时要考虑的一个重要方面是如何处理偏离计划的情况。由于这样的情况时有发生,因此应提前考虑如何处理这种情况。

采用迭代开发的重要目的是要确保不拖延开发进度。为此,有时需要在与有关各方协商并取得其同意的基础上,将某个相对次要的用例移入下一个迭代中去开发。这里要强调的是应当注重按时完成,提倡改掉拖延时间的不良习惯。

但是,如果发现有许多用例要拖后时,则应考虑重新制订计划,包括对尚未开发的各个用例所需的工作量重新进行估计。通过这些措施,开发人员对后续开发所需要的时间就会有更准确的估计。

2. 在构造阶段使用 UML

所有 UML 技术均可用于这个阶段。每当考虑要加入一个用例时,首先应当使用用例模型来确定工作范围,然后用概念层类图来刻画用例的一些概念,并考虑如何将这些概念与已有的软件结合起来。如果用例包含了许多工作流的因素,则可考虑使用活动图来描述。

在这个阶段使用这些技术时应和领域专家一起工作。如果分析工作没有和领域专家一起进行,其分析结果的可信度必然会大打折扣。

过度到设计阶段时将进一步细化,可使用规格说明层类图,并可使用交互图来描述实现用例时类之间的交互作用关系。

对于类图和交互图,既可以直接画,也可以先用 CRC 技术考察其行为,然后再画出类

图和交互图。无论采用这两种方法中的的哪一种，职责都是应首要考虑的因素。

包图可用于绘制系统的逻辑组成图，用来描述系统的逻辑组成部分以及各部分之间的依赖关系。应当采用一些工具来帮助识别这些依赖关系，以避免遗漏。这样的工具有 Perl Scripts，Java 等。由于 Java 直接支持包的概念，所以可以借用。配置图在这一阶段也很有用，可以显示高层的物理结构。

对于每一个包，应该采用规格说明层的观点来描述它的类图，因而不必画出每个类中的每一个操作，只需画出有助于理解的关键属性和操作以及类之间的联系。

类图可以看做是一种图形化的目录内容，它帮助建立了一张包含有对每个类的简要定义的索引表。通常可以通过对职责的描述来对每个类进行简要的定义。应当将有关职责的说明作为注释加到程序代码中，并采用特定的格式，以便能用适当的工具将其抽取出来组成一个文档。

如果一个类行为具有复杂的生命周期，则可画出其状态图。记住，这仅仅在其行为足够复杂时才是必要的。通常的情况是，虽然类之间的交互关系很复杂，但类的行为并不十分复杂，这时只需画出交互图。

一份好的文档至少应包括以下 3 个基本因素：

(1) 1～2 页的类图，描述少量类；

(2) 几幅交互图，描述类之间的协作情况；

(3) 一些文字说明将几幅图连结起来。

通常，还可以用文字编程的风格写下一些重要的代码。如果涉及的算法特别复杂，则应当考虑使用活动图。在构造阶段根据所需要表达的内容，可以从说明层或实现层的观点来设计和画类图，也可以同时采用这两种类图。

7.1.5　移交阶段

迭代开发的关键是规范化地进行整个开发过程，以保证开发组能够正常地交付已完成的代码。但是，有些事则不宜做得太早，譬如代码优化。

优化代码会降低系统的可读性和可扩展能力。最终系统应当足够快，以满足用户的要求，但需要把握好时机。如果做得过早，会给开发带来麻烦。所以，优化是一件应当留待开发后期再做的事情。

在移交阶段，不能再开发新的功能(除了个别小功能或非成基本的以外)，而只是集中精力进行纠错工作。产品的 β 版本和最终版本之间的这段时间是典型的移交阶段。

7.1.6　何时使用迭代式开发方法

要想使项目成功，就应该采用迭代式开发方法。也许这样说有点绝对了，但是，无论怎样，应该尽可能早地发现风险，并在开发过程中对这些风险进行有效的控制。反复迭代并不是没有管理，相反，它需要很好地进行计划和管理。总之，这种方法已经得到肯定，所有面向对象的书都认为应当采用这种方法。

7.2 基于 UML 的需求分析

在了解用户的需求之后，将需求用一种模型来表示，就是需求分析。人们在进行软件开发时，首先要做的就是了解需求，并进行需求分析。需求分析的根本任务是明确系统要完成的功能，建立可理解的模型。它是软件工程中的一个重要阶段，有着极其重要的地位。特别是大规模软件的需求非常复杂，而且用户的需求也会不断变化，没有好的需求分析就不可能编写出优秀的软件，所以好的需求分析是软件开发成败的关键。在以往的软件开发过程中，需求分析往往被当作一个可有可无的步骤而不受重视，甚至许多项目根本就不做需求分析，直接进入设计阶段。这极大地影响了软件的质量，降低了软件开发的效率，更严重的是系统功能不能满足用户需求，最终导致项目失败。

目前比较流行的分析方法是面向对象的方法，通过分析用户需求，用类、类之间的各种关系来表示整个系统。要建立一个准确完整的需求分析模型，通常情况下需要有经验丰富的领域专家提供领域知识，并对需求分析的结果的正确性进行验证，而后由建模专家针对系统的特点提出需求分析的指导原则、实施方案、需求建模的文档规范，对需求分析及建模的进度实施必要的控制等指导性工作，最后由建模工程师充分挖掘需求并提出一些初步的方案，形成最终的需求分析文档。

UML 提供了强大而全面的模型，支持从不同角度来考察系统，其中的用例图、类图、包图、状态图和序列图等模型均可用于需求分析，尤其是以用例为核心组织需求，已被证明为一种行之有效的需求分析方法。

基于 UML 进行需求分析，如图 7-2 所示，其过程通常由以下 3 个部分组成：

1. 分析系统，将系统合理划分

首先，对项目进行调研，依据项目的问题描述，通过分析，将系统合理分割成若干个子系统，将每个子系统用 UML 中的包表示；接着，分析系统中子系统之间的联系，借助 UML 建模工具将子系统之间的联系勾画出来，从而形成系统的概念层的包图。通过这种划分办法来降低系统需求分析的难度。

2. 对每个子系统进行系统分析，识别各个子系统的用例和角色

首先对各个子系统进行需求调研，依据与此子系统的业务流程图和数据流程图以及子系统中涉及的各级操作人员，通过分析，识别出系统中的所有用例和角色并分析它们之间的联系。最后借助 UML 建模工具画出各个子系统的用例图。

3. 针对用例图中每个用例，用活动图描述其业务关系

每个用例代表着系统要实现的功能，每个功能的实现有时可能包含着复杂的业务逻辑，要明确系统的需求，必须对系统的用例的业务关系有个清晰的描述。针对每个用例，分析与其相关的业务流程图和数据流程图，画出该用例的活动图，从而达到明确该用例的业务关系的目的。

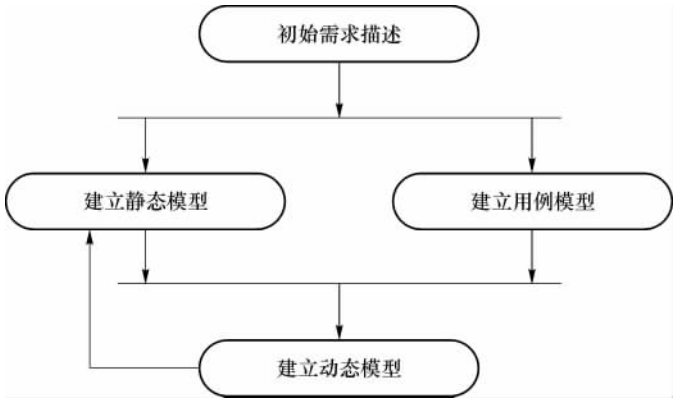

图 7-2　基于 UML 的需求分析过程

7.2.1　建立用例模型

一个典型的用例模型主要由以下模型元素构成：

(1) 参与者(Actor)

参与者是指存在于被定义系统外部并与该系统发生交互的人或其他系统，他们代表的是系统的使用者或使用环境。

(2) 用例(Use Case)

用例用于表示系统所提供的服务，它定义了系统是如何被参与者所使用的，它描述的是参与者为了使用系统所提供的某一完整功能而与系统之间发生的一段对话。

(3) 通讯关联(Communication Association)

通讯关联用于表示参与者和用例之间的对应关系，它表示参与者使用了系统中的哪些服务(用例)，或者说系统所提供的服务(用例)是被哪些参与者所使用的。

在用例建模的过程中，我们建议的步骤是先找出参与者，再根据参与者确定每个参与者相关的用例，最后再细化每一个用例的用例规约。

1. 寻找参与者

所谓的参与者是指所有存在于系统外部并与系统进行交互的人或其他系统。通俗地讲，参与者就是我们所要定义系统的使用者。寻找参与者可以从以下问题入手：

(1) 系统开发完成之后，有哪些人会使用这个系统？

(2) 系统需要从哪些人或其他系统中获得数据？

(3) 系统会为哪些人或其他系统提供数据？

(4) 系统会与哪些其他系统相关联？

(5) 系统是由谁来维护和管理的？

这些问题有助于我们抽象出系统的参与者。对于图书馆管理系统的例子，回答这些问题可以使我们准确找到参与者：读者通过本系统预约、取消预约、借书、还书，管理员负责处理和维护管理本系统。

2. 确定用例

找到参与者之后，我们就可以根据参与者来确定系统的用例，主要是看各参与者需要

系统提供什么样的服务，或者说参与者是如何使用系统的。寻找用例可以从以下问题入手(针对每一个参与者)：

(1) 参与者为什么要使用该系统？

(2) 参与者是否会在系统中创建、修改、删除、访问、存储数据？如果是的话，参与者又是如何来完成这些操作的？

(3) 参与者是否会将外部的某些事件通知给该系统？

(4) 系统是否会将内部的某些事件通知该参与者？

综合以上所述，图书馆管理系统的用例图可如图 7-3 所示。

在用例的抽取过程中必须注意：用例必须是由某一个主角触发而产生的活动，即每个用例至少应该涉及一个主角。如果存在与主角不进行交互的用例，就可以考虑将其并入其他用例，或者是检查该用例相对应的参与者是否被遗漏，如果是，则补上该参与者，反之，每个参与者也必须至少涉及到一个用例。如果发现有不与任何用例相关联的参与者存在，就应该考虑该参与者是如何与系统发生对话的，或者由参与者确定一个新的用例，或者该参与者是一个多余的模型元素，应该将其删除。

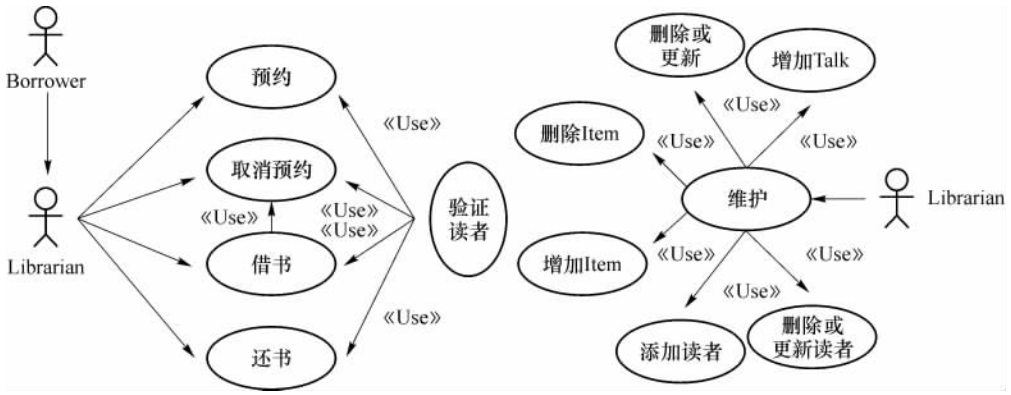

图 7-3 图书馆管理系统用例图

3. 检查用例模型

(1) 功能需求的完备性

现有的用例模型是否完整地描述了系统功能，这也是我们判断用例建模工作是否结束的标志。如果发现还有系统功能没有被记录在现有的用例模型中，那么我们就需要抽象一些新的用例来记录这些需求，或是将他们归纳在一些现有的用例之中。

(2) 模型是否易于理解

用例模型最大的优点就在于它易于被不同的涉众所理解，因而用例建模最主要的指导原则就是它的可理解性。用例的粒度、个数以及模型元素之间的关系复杂程度都应该由该指导原则决定。

(3) 是否存在不一致性

系统的用例模型是由多个系统分析员协同完成的，模型本身也是由多个工件所组成的，所以我们要特别注意不同工件之间是否存在前后矛盾或冲突的地方，避免在模型内部

产生不一致性。不一致性会直接影响到需求定义的准确性。

(4) 避免二义性语义

好的需求定义应该是无二义性的,即不同的人对于同一需求的理解应该是一致的。在用例规约的描述中,应该避免定义含义模糊的需求,即无二义性。

7.2.2　UML 静态建模机制

用例图是站在用户的角度上理解图书馆管理系统所要完成的功能。根据用例图进行领域概念分析,可以找出图书馆管理系统中概念性的类及它们之间的相互关系,进而画出类图,用于表示系统静态模型。寻找对象的基本规则是名词和名词组成为候选对象,动词是对象的服务,形容词可能暗示存在子类。当然,由于自然语义并不十分精确,所以不能机械地套用基本规则,还须作进一步的分析与调整。如果类很多,还可从功能的角度建立逻辑包,绘制包图。在面向对象分析阶段用 UML 进行静态建模时,建立用例图、类图就基本能够描述系统,如果需要,也可建立包图和对象图。而构件图和配置图是在设计、实现阶段所必需的。

对于一个所描述的系统,其类模型、对象模型以及它们之间的关系揭示了系统的内部结构,一定程度上也反映了系统的数据表达、组织和存储方式。类图描述了系统中的类及其相互之间的各种关系,其本质反映了系统中包含的各种对象的类型以及对象间的各种静态关系,并且在系统的整个生命周期都是有效的。

下面以一个小型图书管理系统为例,说明进行需求分析的过程,下面给出这个系统初始的需求描述:读者可以向图书馆借书,读者和被借的书都必须在系统中注册;图书馆对新买的书进行处理,流行的书可以买多本,过期或破损的旧书从系统中删除;管理员是图书馆的雇员,与读者交互,系统支持管理员的工作;读者可以预约不在馆的书,当预约的书还回来或买回来之后,通知读者,当读者借到书或者取消预约时,预约信息被删除;系统可以方便地对图书信息、读者信息、借书信息和预定信息进行创建、更新和删除。

分析需求陈述,利用名词策略,或者组织用户、领域专家和开发者召开头脑风暴会议,找出所有的需要处理的类及其主要属性以及类之间的关系。通过对需求描述的分析,从中可以发现以下类:书名(Title)、书(Item)、借书(Loan)、预约(Reservation)、读者信息(BorrowInfo)等,于是可以用 UML 类图表示这些类及其关系,如图 7-4 所示。

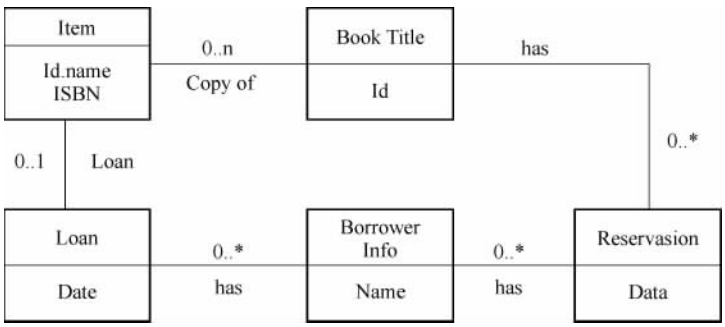

图 7-4　图书管理系统类图