HZ BOOKS

高等院校计算机教材系列

面向对象技术与UML

刘振安 董兰芳 刘燕君 编著

机械工业出版社
China Machine Press

高等院校计算机教材系列

面向对象技术与UML

刘振安 董兰芳 刘燕君 编著

机械工业出版社
China Machine Press

本书介绍基于UML建模语言描述的面向对象的分析与设计过程，内容包括软件开发过程、面向对象的基本概念、基于UML语言进行分析和设计的流程，并结合实际的工程要求，介绍了软件体系结构的模式以及持久对象的概念。

本书通俗易懂、概念清楚、实用性强，可以作为软件工程硕士、研究生、高年级本科生的教材，也可以作为自学或培训教材以及工程技术人员的参考书。

图书在版编目（CIP）数据

面向对象技术与UML／刘振安，董兰芳，刘燕君编著. －北京：机械工业出版社，2007.4

（高等院校计算机教材系列）

ISBN 978-7-111-20912-6

Ⅰ.面…　Ⅱ.①刘…　②董…　③刘…　Ⅲ.面向对象语言，UML－程序设计－高等学校－教材　Ⅳ.TP312

中国版本图书馆CIP数据核字（2007）第023619号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：杨庆燕

北京京北制版厂印刷·新华书店北京发行所发行

2007年4月第1版第1次印刷

184 mm×260 mm·13.5印张

定价：22.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

前　言

20世纪70年代中后期，由于软件开发失败的几率很高，人们开始了软件工程的研究，将工程上的管理思想应用于软件开发，以应对软件危机。人们希望建立成套的管理方法，强调软件的模块性、抽象性、易维护性、可修改性、可移植性，将软件“做什么”和“怎么做”分开。在很长的一段时间里，尽管各种程序设计方法学应运而生，但是仍然不能从根本上解决或根除软件危机。

通过对已有的软件开发方法进行分析和总结，人们发现以往的分析、设计或实现方法存在着以下的问题：

1）要求开发人员按计算机的结构去思考，而不是按要解决的问题的结构去思考。

2）开发员必须在机器模型（解空间）和实际问题模型（问题空间）之间进行对应。

这种基于过程开发的最终程序，不仅接口和功能容易变，过程执行顺序也容易变，而且数据也极容易变。总之，这种方法充满着变数。

人们经过进一步的研究，发现相对于过程而言，对象是稳定的。我们把问题空间中的事物和它们在解空间中的表示称为“对象”。

基于过程的方法按计算机的结构建模，而基于对象的方法是对问题本身建模。当阅读用基于对象的方法来描述解决方案的代码时，也就相当于阅读表达该问题的文字。面向对象的方法允许开发人员用问题本身的术语来描述问题，而不是用要运行解决方案的计算机的术语来描述问题。

模型是专业人员用来与项目风险承担者和其他开发人员进行工作交流的，是对分析与设计结果的描述。采用统一的语言对分析与设计产生的软件模型进行描述具有重要的意义。面向对象的建模语言UML产生于20世纪90年代，并获得了工业界、科技界和应用界的广泛支持，成为面向对象和可视化建模语言事实上的工业标准。它代表了面向对象方法的软件开发技术的发展方向，具有巨大的应用前景。当前，有软件工程相关专业的高等院校已纷纷开设了基于UML的面向对象的分析与设计课程。

本书在中国科学技术大学软件工程硕士和高年级本科生的教学基础上，参阅国内外多种最新教材和资料编写而成。本书结合软件开发过程中所涉及的内容逐步展开，既结合理论，又接近工程实际需要，目的是引导学生训练自己分析问题和使用UML解决问题的能力，并养成良好的分析与设计习惯。本书的主要特点如下：

1）重点放在UML语言的基本特征上，涵盖UML语言的重要基础知识。

2）介绍面向对象最常用的概念和方法，而不是面面俱到。

3）结合设计实例，介绍实现设计的主要过程。

4）注意介绍目前最新的研究成果。

本书适合作为软件工程硕士、研究生、高年级本科生的教材，也可以作为广大工程技术人员的参考书或培训教材。参加本书编写的还有周军和徐浩等。由于作者水平所限，书中难免存在不妥之处，恳请广大读者批评指正。

作　者

2007年于中国科技大学

目　录

第1章　软件开发过程概述

显然，软件的种类和规模并不相同，所以不同的软件其开发过程也有所区别。本章旨在让读者了解软件开发的典型过程。一般来说，软件开发的过程可以分为需求分析、设计、实现、测试、运行和维护几个阶段，对于一个复杂的系统来说，需要多次重复这些过程。

UML 设计者定义了软件开发过程中公共的过程框架，同时给用户留出一定的自由空间，以便用户采用适合自己项目的技术。对于一个软件来说，无论采用何种过程，都可用 UML 来记录最终的分析和设计结果。

1.1　软件开发基本概念

软件开发涉及多种因素，如软件的种类（通信系统、办公自动化系统、嵌入式软件）、规模（个人开发、小组开发、超过 100 人的群体开发）等，所以很难定义一种通用的软件开发过程，以支持各种类型软件的开发。也就是说，在软件开发过程中，应当视具体的情况，采用不同的过程。

本节将介绍软件开发过程中的基本概念和主要流程，但不深入讨论开发过程中的各种问题。UML 并不包含对软件开发过程的定义，它是一种建模语言而不是一种方法。软件开发中，你可以选用任何适合项目类型的过程。但无论采用何种过程，都可用 UML 来记录最终的分析和设计结果。

本章旨在让读者了解软件开发的典型过程。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自由空间，以便用户采用适合自己项目的技术。

1.1.1　基本概念

软件是计算机系统中与硬件相互依存的一部分，它是包括程序、数据及其相关文档的完整集合。程序是按事先设计的功能和性能要求执行的机器指令序列，数据是使程序能正常运行的相关信息，而文档则是与程序开发、维护和使用相关的图文材料。

一般来讲，软件具有以下特点：

1）软件是一种逻辑实体，不是具体的物理实体，它具有抽象性。

2）对软件的质量控制，必须立足于软件开发的过程。

3）在软件的运行和使用期间，不存在硬件那样的磨损、老化问题。

4）软件的开发和运行往往受到计算机系统的限制，对计算机系统有不同程度的依赖性。

迄今为止，软件开发尚未完全摆脱手工编程的方式，其原因如下：

1）软件本身是复杂的。

2）软件的成本相当昂贵。

3）相当多的软件工作会涉及社会因素。

很多软件的开发过程很复杂，并且在开发过程中会产生以下问题：

1）对软件开发成本和进度的估计不准确。

2）用户不满意。

3）软件质量不高，可靠性差。

4）软件常常不可维护，难以改正错误。

5）缺乏适当的文档资料。

6）软件成本在系统总成本中的比例逐年上升。

7）软件开发速度跟不上计算机发展速度。

为了保证软件能够成功开发，人们试图从以下的两个方面着手进行改善。

1）使用更好的软件开发方法和开发工具。

2）进行组织管理，软件开发不是某项采用神秘技巧的个体劳动，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。

实践证明，采用先进的技术既可提高软件开发的效率，又可提高软件维护的效率。在20世纪80年代之前，采用的技术有结构化分析和设计技术，自20世纪90年代以来，面向对象分析和设计技术得到了广泛的应用。本书后续的章节将重点介绍基于统一建模语言（UML）的面向对象的分析与设计技术。在管理方面，人们也积极探索，提出了软件工程的概念，建立了软件过程成熟度模型CMM和CMMI，这将在本章的稍后小节中加以介绍。

软件开发的过程可以分为需求分析、设计、实现、测试、维护阶段。对于一个复杂的系统来说，要多次使用上述过程，所以首先介绍迭代增量式的开发过程。

1.1.2 迭代增量式的开发过程

图1-1显示了统一过程（Unified Process）中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软件。开发过程分为四个阶段：初始阶段、细化阶段、构造阶段和移交阶段。统一过程的核心思想是：首先选择一些功能点，然后完成这些功能，随后再选择其他的功能点，如此循环往复。

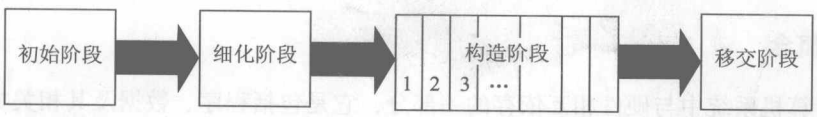

图1-1 开发过程简图

在初始阶段，需要考虑项目的效益，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，并为构造阶段制定计划。

构造阶段由多次开发组成，每一次开发都包含编码、测试和集成工作，所得产品应满足项目需求的某一子集，或提交给早期用户，或纯粹是内部提交。每一次迭代都包含软件生命周期的所有阶段，即分析、设计、实现和测试阶段。

运用这种迭代开发过程时，还是会有一些工作要放到移交阶段进行，如β测试、性能调试和用户培训等。

不同项目有不同的要求，“规范”较严的项目需要交付许多正式的文档，举行多次正式会议，要求进行多次正式签字。“规范”较宽的项目可能只需要在初始阶段与项目出资方进行一小时左右的交谈并制定一个表格式的计划。显然，项目越大，“规范”要求越严。但是无论项目的

规模多大，要求多严，都需要一个基本的规范，只是表现的形式有所不同。在实践中，规范的严宽应当适度，开发过程应当尽量简单明确，避免含糊不清、随意变动，同时也要简化不必要的工作。

可以看出，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。有很多人认为，要想使项目成功，就应该采用迭代式开发方法。这种说法也许有点太绝对，但确实需要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地控制。反复迭代并不意味着没有管理，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，所有面向对象的书籍都认为应当采用这种方法。

在 Jacobson 等人所著的《The Unified Software Development Process》一书中，作者进一步将统一过程描述为基于建模语言 UML 的、以体系结构为中心的、用例驱动与风险驱动相结合的、迭代的软件开发过程。这个开发过程包含初始、细化、构造和移交四个阶段，所有开发工作都围绕需求捕获、分析、设计、实现和测试 5 个核心工作来组织。

1.1.3 其他开发模型

如前所述，面向对象方法是一种运用对象、类、继承、封装、分类、聚合、关联、消息传递、多态性等概念来构造系统的软件开发方法，而且面向对象分析与面向对象设计可适应多种软件生命周期模型，开发方法也愈来愈成熟。

有人主张先按瀑布模型进行面向对象的分析，然后进行面向对象的设计；也可以按螺旋模型或增量模型，交替地进行面向对象的分析和面向对象的设计。不过更能体现二者之间关系特点的是近几年提出的喷泉模型。这种模型用两个交叉的水泡表示面向对象的分析和面向对象的设计，说明两者没有严格的边界，它们是连续的、无缝的，允许有一定的相交（一些工作既可看作是面向对象的分析，也可看作是面向对象的设计），也允许从面向对象设计回到面向对象分析。

图 1-2 是螺线（Spiral）模型的示意图。螺线模型的开发路径是原型开发→分析→设计→实现→维护→分析，周而复始，每一周期逐步求精，逼近目标系统。每个周期包括同样的步骤来用于项目的每个部分和每个层次的求精。它能较早注意到对已有软件的重用，并能对软件的扩展和需求变化做出快速的反映。

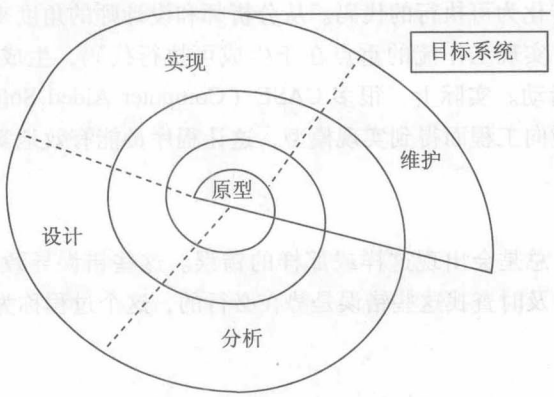

图 1-2 软件开发的螺线模型

开发者可以根据自己的实际情况，选择适合自己的开发模型。下面将重点介绍软件开发的主要过程。

1.2 需求分析

软件系统的分析指理解特定问题域的需求并制定书面文件。问题域（Problem Domain）是开发的整个业务范围，包括计划、分析、设计以及最终作为自动信息系统实现的某个商业功能。

系统设计是根据分析阶段所得的书面需求设计问题域的合适解决方案。

在进行软件的分析和设计之前，必须对这个系统要尽力达到的目标作一些规定，这就是软件需求过程的目的，即揭示系统应该做什么并达成一致，而且使用一种特定语言来表达。为系统应该做什么创建非常高级的规格说明的工作称为需求工程。

对于任何给定的系统，存在很多不同的利益相关人，这些人可以是不同类型的用户、维护工程师、支持人员、销售人员、管理人员等。需求工程要抽取系统中这些利益相关人的需求，并排出他们的优先级。这是一个协商过程，因为其中常常存在必须平衡相互冲突的需求。例如，某个团体可能想要添加很多用户，这可能在已有的数据库和通信基础结构上产生不现实的负荷。现在，这类冲突非常普遍，因为越来越多的公司通过因特网向巨大的用户群开放它们的部分系统。

需求分为功能性需求和非功能性需求。功能性需求是关于系统做什么的描述，非功能性需求是关于系统约束的描述（性能、可靠性等）。

分析的目标是产生分析模型，该模型关注系统需要做什么，但把系统如何做的细节留到设计阶段。

实际上，需求分析时还要考虑各种风险问题，风险分析将在1.7节中叙述。

1.3 设计

当分析活动越来越完善的时候，建模的焦点就转向设计。分析时的焦点是创建系统的逻辑模型；设计的目的是说明如何才能完全实现这些功能。需求来自于问题域，分析是从问题相关者的角度对问题域进行探究。设计包括整合解域的技术解决方案以提供实际可实现的系统模型。

1.4 实现

实现是将设计模型转化为可执行的代码。从分析师和设计师的角度来看，如有必要，实现的目的就是生成实现模型。实现工作流的重点在于生成可执行代码，生成实现模型只是它的副产品，而没有清晰的建模活动。实际上，很多CASE（Computer Aided Software Engineering）工具允许人们对源代码进行逆向工程而得到实现模型，这让程序员能有效地实现建模。

1.5 测试

软件在开发过程中，总是会出现这样或那样的错误。这些错误导致软件不能得到预期的效果。因此在软件开发中，及时查找这些错误是势在必行的，这个过程称为测试。

1.5.1 测试目的

Grenford J. Myers在《The Art of Software Testing》一书中描述了测试目的：

1）软件测试是为了发现错误而执行程序的过程。

2）测试是为了证明程序有错误，而不是证明程序无错误。

3）一个好的测试用例在于它能发现至今未发现的错误。

4）一个成功的测试是发现至今未发现的错误的测试。

这种观点提醒人们，测试要以查找错误为中心，而不是为了演示软件的正确功能。但是仅凭字面意思理解这一观点可能会产生误解，认为发现错误是软件测试的唯一目的，查找不出错误的测试就是没有价值的，但事实并非如此。

首先，测试并不仅仅是为了找出错误，而是通过分析错误产生的原因和错误的分布特征，帮助项目管理者发现当前所采用的软件过程的缺陷，以便改正。同时，这种分析也有助于设计出有针对性的检测方法，改善测试的有效性。

其次，没有发现错误的测试也是有价值的，完整的测试是评定测试质量的一种方法。详细而严谨的可靠性增长模型可以证明这一点。例如，Bev Littlewood 发现一个经过测试而正常运行了 n 小时的系统有继续正常运行 n 小时的概率。

1.5.2 测试集

正确性是程序最重要的属性。即使是很小的程序，要采用严格的数学证明方法来证明其正确性，也是非常困难的，因此只好求助于程序测试过程来实施这项工作。程序测试是指在目标计算机上利用输入数据（也称为测试数据）运行该程序，将运行结果与所期望的结果进行比较。如果两种结果不同，就可判定程序存在问题。然而，即使两种结果相同，也不能够断定程序就是正确的，因为对于其他的测试数据，可能会得到不同的结果。如果使用了许多组测试数据都能得到相同的结果，则可增加对程序正确性的信心。不过，要想通过使用所有可能的测试数据来验证程序是否正确是不现实的，因为对于大多数实际的程序，可能的测试数据的数量过大，不可能进行穷尽测试。实际用来测试的输入数据空间的子集称为测试集。

1.5.3 设计测试集的方法

在设计测试数据的时候，应当牢记，测试的目标是去找出错误。如果利用测试数据找不到错误，人们就可以进一步相信程序的正确性。目标是采用尽量少的测试数据来发现尽量多的错误。为了弄清楚对于一个给定的测试数据，程序是否存在错误，首先必须知道对于该测试数据，程序的正确结果应是什么。

一般来说，应该从两个方面来选择测试数据：这个数据能够发现错误的程度，并且能验证采用这个数据时程序的正确性。

设计测试数据的技术有三种：白盒测试法、黑盒测试法和灰盒测试法。第三种方法是前两种的混合。针对不同的测试，在选择测试用例时有很大差别。利用黑盒法时，考虑的是程序的功能，而不是实际的代码。利用白盒法时，通过检查程序代码来设计测试数据，以便使测试数据的执行结果能很好地覆盖程序的语句以及执行路径。

1.6 维护

软件维护是指在软件交付使用之后，为了改正错误或满足新的需要而修改软件的过程。维护的目的是保证软件系统能持续地与用户环境、数据处理操作、政府或其他有关部门的请求取得协调。

软件的维护可以分为四种类型。

1. 正确性维护

正确性维护的目的是改正在系统开发阶段已发生的，而系统测试阶段尚未发现的错误。一般来说，这类故障是由于遇到了以前从未有过的某种输入数据的组合，或者是系统的硬件和软件有了不正确的界面而引起的。

2. 适应性维护

适应性维护是为适应软件的外界环境变化而进行的修改。

3. 完善性维护

完善性维护是为扩充系统的功能和改善系统性能而进行的修改。

4. 预防性维护

预防性维护是为减少或避免以后可能需要的前三类维护而对软件配置进行的工作。

1.7 风险分析

在软件开发中存在着各种风险，清楚地认识这些风险，将有助于软件的开发。一般来讲，风险可分为需求风险、技术风险、技能风险和政策风险四种类型。

1. 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，有着非常重要的地位。要规避需求风险就要仔细分析系统的需求是什么。软件最大的危险是建立了一套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。

在需求分析中，UML 采用的主要是用例分析技术，用例分析技术是 UML 的核心。采用 UML 中的用例分析技术，是避免或减少需求风险的主要途径。

UML 强调要保证获得应用领域中的关键对象类和保证关键用例的正确性，并建立可扩展的、可重用的体系结构。然后，可以逐步加入其他的用例，并在迭代式开发过程中分阶段实现这些用例，并不要求一次完成整个系统。

在初始阶段尤其要注意尽量少地采用各种符号，也不必考虑严密性，而是建议在图上作大量的非形式化的注释。总之，无需考虑每一个细节，而是集中考虑带有风险的重要议题。在这个过程中，通常会画出大量相互之间没有明显联系的图，暂时不必考虑它们之间的一致性和相互关系。这个过程非常有助于开发人员理解用户需求。在理解的基础上，更容易识别出不同用户所要求的用例。

细化阶段最重要的工作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。用例不必很详细，通常用一段文字或几句话描述就足够了，但应保证不仅用户能理解其含义，而且开发人员也能明白其内涵。

2. 技术风险

技术风险是指所选的技术方案是否可行方面存在的风险。

在考虑技术风险时，最重要的工作是建立原型系统，尝试技术方案是否可行。例如，如果采用 Java 语言和关系数据库，则应做以下工作：

1）取得 Java 语言相关的支持工具。

2）建立域模型早期版本中的一个简单组成部分，以了解工具能完成哪些工作，有哪些限制？如果准备采用一种新的开发工具，新旧程序如何衔接？新、老数据文件的格式如何保持兼容等。

3）建立数据库，并和 Java 代码关联。

4）按照上述步骤再试几个工具，并进行比较，最后选择最适合易用的工具。

5）要慎重地采用新技术。例如，如果采用 ODBC 技术，则要仔细分析其优缺点，看看是否存在一些致命的限制，例如处理大量数据时其响应速度能否达到设计要求。

将各个组件集成也是非常重要的工作。例如，开发者可能很熟悉 Java，也了解关系数据库，但是没有将这两者结合的经验，这也存在风险。这就是要在开发过程的早期阶段考虑所有组件，并将其组合在一起的原因。

另外，还应考虑体系结构的设计问题，考虑所需组件以及构造方法。这在设计分布式系统时尤为重要。在这一阶段，还应考虑如何使设计对于设计成分的修改和维护来说更加容易，这包括以下问题：

1）如果某一技术不能正常工作，将发生什么情况？

2）如果无法将两个组件连接起来，会发生什么情况？

3）如果某一部分出错，会出现什么情况？如何处理这种情况？

总而言之，要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，应作深入的调查。在此过程中，需要采用大量 UML 技术来帮助刻画设计思想，并写出需求文档，此时无需完全和精细，而是要保证简洁和准确。建议使用类图和交互图来描述组件间的通信，使用包图来描述组件的高层结构，使用配置图来描述系统功能的分配。包图就像一个容器，可以用来组织模型中的相关元素，例如，将一个大系统分解为多个小系统，以便更容易理解系统构成。配置图用于定义系统中软硬件的物理体系结构，可以显示连接的类型及部件之间的依赖性。关于它们的详细介绍，请见 2.4 节。

3. 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存在的风险。

例如，不少公司在几乎毫无面向对象技术方面经验的情况下，就确定采用面向对象技术来实施一些重要的项目，但是却不愿出资进行有关面向对象技术方面的培训，这是非常错误的做法。要知道，项目每拖延一天都是需要花费费用的。

进行培训是避免出错的有效方法，因为它可以将前人的经验传授给参与培训的人员，否则如果项目出现差错需要返工或修补，则将耗费大量的时间和资金，同时影响项目的进度和质量。尽管事前培训和出错后补救都需要资金，但是如果不进行培训，项目需要更多的人才能完成，因而也就耗费更多的成本。因此，根据实际情况，采取自学、参加培训、组织学习小组和研讨等方法都是很有效的，也是较切实可行的方法。

在参加培训时必须注意如下两点：

1）不能只读书，要更加注重实际运用的练习。

2）普通的正规培训课程仅仅告诉人们需要了解什么，而不能传授做项目时的核心技能。短期培训是有用的，但这仅仅是一个开端。如能聘请有实践经验的专家作项目的顾问，并请有丰富经验的开发人员共同工作一段时间，由他们指导如何做，并在检查工作的同时传授技巧，对人员进行培训，则是一种最佳选择。

在项目开发过程中，应当随时注意发现自己缺乏的技能或经验，并做出计划以便能取得所需要的经验。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。

4. 政策风险

有时也需要考虑是否存在将会影响项目进行的一些政策性因素。这里所指的政策性影响是广义的、多方面的，例如，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，政策的变化会对项目的进展产生一些分歧意见。这虽然是正常的现象，但在产生不同意见后，应该重新讨论，按照一定原则作出新的决策。必要时还可以向一些政策方面的专家进行咨询。

由于政策风险属于人文科学的范畴，不属于 UML 语言教程讨论的范围，在此就不再作进一步阐述。

1.8 面向对象的方法

为了保证软件能够成功开发，人们首先考虑使用更好的软件开发方法和开发工具。

1.8.1 面向对象的软件开发方法

总结基于过程的软件开发方法，人们发现以往使用的基于过程的方法在分析、设计或实现方法时，会存在以下的问题：

1）要求开发人员按计算机的结构去思考，而不是按要解决的问题的结构去思考。

2）开发员必须在机器模型（解空间）和实际问题模型（问题空间）之间进行对应。

使用这种基于过程的方法最终产生程序中的各种成分：接口容易变，功能容易变，过程执行顺序容易变，数据也极容易变。总之，基于过程的方法充满着变数。

通过进一步的研究，人们还发现，相对于过程，对象是稳定的。称现实世界中客观存在的事物为对象，例如整数是一个对象，平面上的点是一个对象，河流湖泊都是对象。复杂的对象可以由简单的对象组成，例如火车站对象又包含售票处、行李房、信号灯、站台、铁轨和通信设施等对象。这些对象各自又由许多对象组成，对象各自完成特定的功能。总之，世界万物皆对象。面向对象的方法就是以对象代表求解问题的中心环节，追求的是现实问题空间与软件系统解空间的近似和直接模拟。这就改变了原来计算机程序的分析、设计和实现的过程与方法之间的脱节和跳跃状态，从而使人们对复杂系统的认识过程与系统的程序设计实现过程尽可能地一致。

基于过程的方法是以函数过程和数据结构为中心进行建模，而基于对象的方法则是对求解问题的本身进行建模。所以，当人们研究基于对象的方法所描述的解决方案的代码时，也就相当于阅读表达该问题的文字描述。面向对象的方法允许开发人员用问题本身的术语来描述问题，而不是用要运行解决方案的计算机的术语来描述问题。每个对象看上去就像一台小计算机，它有状态，有可以执行的运算。对象实际上是功能抽象和数据抽象的统一。可见，面向对象不仅仅作为一种技术，更作为一种方法论贯穿于软件设计的各个阶段。

其实，软件开发是一个对给定问题求解的过程。从认识论的角度看，软件开发可以归为两项主要活动：认识与描述。软件开发者将被开发的整个业务范围称作“问题域”，“认识”就是在所要处理的问题域范围内，通过人的思维，对该问题域客观存在的事物以及对所要解决的问题产生正确的认识和理解，包括弄清事物的属性、行为及其彼此之间的关系并找出解决问题的方法。

因为人类的任何思维活动都是借助于他们所熟悉的某种自然语言进行的，所以开发人员对问题域的认识是人类的一种思维活动。例如，有些人讲话很流利，但考虑问题时却很糊涂，这说明人们还需要具有正确的思维方法。尤其是在软件开发过程中，要求人们对问题域的理解，要比日常生活中对它的理解更深刻、更准确。这需要许多以软件专业知识为背景的思维方法。

“描述”是指用一种语言把人们对问题域中事物的认识、对问题及其解决方法的认识描述出来。最终的描述必须使用一种能够被机器读得懂的语言，即编程语言来完成。

人们借助自然语言所产生的对问题域的认识远远不能被机器理解和执行，而机器能够理解的编程语言又很不符合人的思维习惯。人们习惯使用的语言和计算机能够理解并执行的编程语言之间存在着很大的差距，这种差距称为“语言的鸿沟”。程序设计语言发展的趋势就是尽量缩短这道鸿沟。图1-3给出随着语言发展鸿沟变窄的示意图。

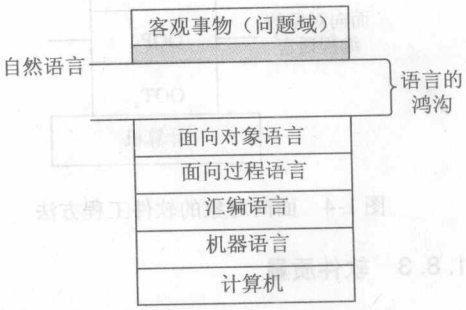

图1-3 语言的发展使鸿沟变窄

由于人的认识差距，所以问题域与自然语言之间也有缝隙。机器语言与自然语言的鸿沟最宽，随着编程语言由低级向高级的发展，它们与自然语言之间的鸿沟在逐渐变窄。

1.8.2 面向对象的软件工程思想

在20世纪70年代中后期，因为软件开发失败的几率很高，人们开始了软件工程的研究，将工程上的管理思想应用于软件开发，以减少软件危机。人们希望建立成套的管理方法，强调软件的模块性、抽象性、易维护性、可修改性、可移植性，将软件“做什么”和“怎么做”分开，在很长的一段时间里，尽管各种程序设计方法学应运而生，但是仍然不能从根本上解决并根除软件危机。

面向对象是认识论和方法学的一个基本原则。人对客观世界的认识和判断，常常采用由一般到特殊（演绎法）和由特殊到一般（归纳法）两种方法，这实际上是对认识判断的问题域对象进行分解和归类的过程。

面向对象分析（Object-Oriented Analysis，OOA）是面向对象软件工程方法的第一个环节，它包括一套概念原则、过程步骤和归类的过程。OOA的任务是采用面向对象方法，把对问题域和系统的认识理解，正确地抽象为规范的对象（包括类，继承层次）和消息连接关系，形成面向对象模型，为后续的面向对象设计（Object-Oriented Design，OOD）和面向对象编程（Object-Oriented Program，OOP）提供指导。而且OOA与OOD能够自然地过渡和结合，也是面向对象方法值得称道的一个优点。OOA和OOD的区别主要是前者与系统的问题域有关，后者与系统的实现更加密切。一般来讲，只有在基本掌握了OOP的主要表达方法的基础上，才能从系统分析入手进行面向对象的软件设计。这也是目前研究的热门话题，即面向对象的软件工程方法。如图1-4所示，这种方法从OOA到OOD，再到OOP和OOT（面向对象测试），都是紧密衔接的，填平了语言之间的鸿沟。

如图1-5所示，建立在面向过程基础上的传统的软件工程方法虽然在语言的鸿沟上铺设了一段平坦的路，但这段路并不连续，尤其是存在着需求分析和总体设计之间的鸿沟，会造成致命的弱点。