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(例如，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)做到收益的最大化。在保证产品质量的前提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个目的。软件工程追求软件开发的精益化，其本质是规避浪费，确保在合适的时间去做合适的事情，达到效率的最大化，这是精益思想在软件工程中的体现。

软件工程是研究和应用如何以系统性、规范化、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且被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及软件工程的由来、软件工程包括的主要内容、软件工程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遇到的一系列严重问题，这些问题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。

(1)软件是逻辑层面上的，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，而不是制造的环节，因为软件的制造过程可以理解为“复制”。

(2)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的缺陷。这个过程比较复杂，有时需要经历新的开发过程，而且缺陷被发现得越晚，为之付出的代价就越高。

(3)软件开发早期是一门艺术，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。

(4)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。

(5)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。随着信息领域的发展，软件的规模会越来越大，也会越来越复杂。

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。例如，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。实际上，软件是由三部分组成的，即程序、数据和文档。程序是指能够运行的、能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的一般性规律——变化，变化是软件开发不变的主题，变化带来了诸多挑战。

(1)软件开发各环节对缺陷具有放大作用，一个小的问题如果不及时识别和处理，经过几级放大后，会在后期带来“可观”的成本上的损失。

(2)只有早期发现问题，才会尽量减少损失，但一个难以解决的问题是，用户的需求不可能一次确定下来；甚至有时候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于软件的复杂性、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，寻找解决软件危机的途径，主要包括以下内容。

(1)要对软件有正确的认识。

(2)推广使用软件开发成功的技术和方法，研究探索更有效的技术和方法。

(3)开发和使用更好的软件工具。

(4)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

软件工程是从技术和管理两个方面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(国际标准化组织)和 IEC(国际电工委员会)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。软件工程知识体系包含两个部分，10 个主要的知识域，如图 1.1 所示。

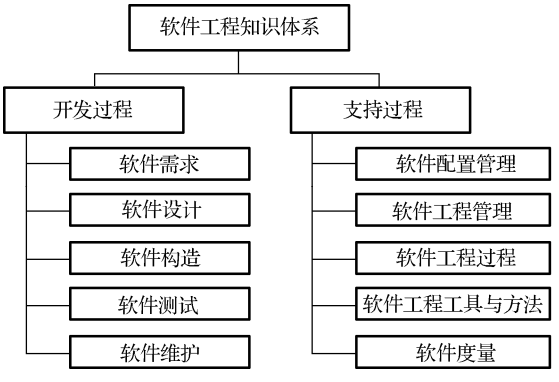

图 1.1　软件工程知识体系

软件需求(Software Requirement)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，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)是通过编码、验证、测试和排错的组合，具体创建一个可以工作的、有意义的软件的过程，包括软件构造基础、管理构造和实践考虑。

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

软件维护(Software Maintenance)，包括软件维护基础、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比其还要早。

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，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、软件工程度量。

软件工程过程(Software Engineering Process)涉及软件工程过程本身的定义、实现、评估、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、过程评估、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、软件质量过程和实践考虑，是在处理跨越软件生命周期过程时的软件质量的考虑。软件质量在软件工程中无处不在，其他知识域中也涉及质量问题。

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 UML

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确保在正确的时间使用正确的方法做正确的事情。

系统工程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。根据一系列相关元素的合理组织,系统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。例如,产品工程的层次结构模型如图 1.2 所示。

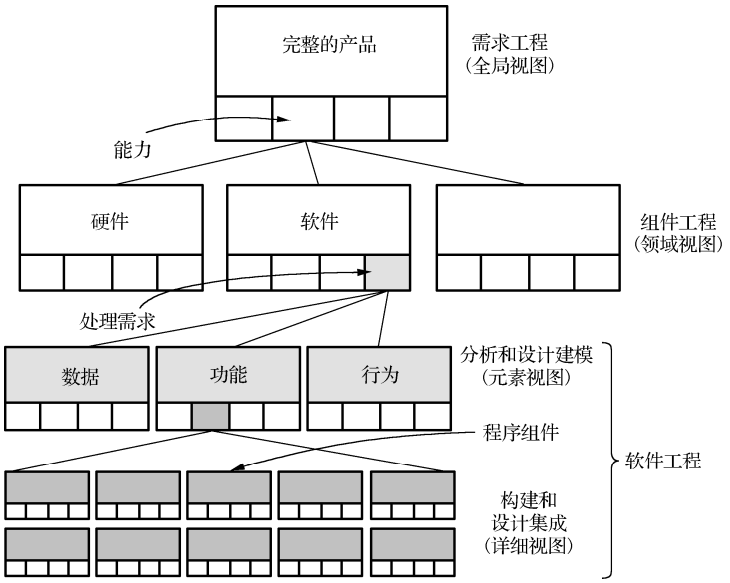

图 1.2　产品工程的层次结构模型

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;最底层表示要选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。

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,面向对象领域的又一个研究与讨论的热点。UML 方法统一了 Booch、OMT 及 OOSE 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,OMG)的标准之一。

顾名思义,UML 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，其表述形式以图形方式为主，但其描述形式本质上仍归为非正式(Informal)的方式，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。

历史上，三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，这三个人被誉为“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，适用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，以上三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的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表示方法及相关的方法论形成了一个群雄争霸的时代，历史上称为“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，在方法中加入了用例思想。1996 年，三人共同将他们的新方法命名为 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。三人所在的 Rational 公司也将此时的 UML 1.0 版本提交给了 OMG，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 年成为了当时的官方版本，其也是在 UML 历史上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 也成为了 ISO 的标准之一。这些都决定了 UML 在软件开发建模领域中的地位。

UML 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)，下面分别进行说明。

1. 可视化

可视化能帮助开发者理解和解决问题，方便熟悉 UML 的开发者彼此交流和沟通。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。它使得编码前的一些重要的决定得以表示，使得开发者达成共识，能够对后续软件的开发过程进行指导，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来实现，大大加快系统建模和实现速度。

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

4. 文档化

使用 UML 可以同时生成系统设计文档。这些专业化的设计文档可以帮助开发者节省开发时间，快速熟悉和理解系统，起到人与系统之间的桥梁作用，达到事半功倍的效果。

UML 是提供给用户的高层建模的方法，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。还有一种表示方法是使用自然语言描述，但自然语言具有歧义和含糊不清的缺点。

UML 具有简单的表示法(Notation)，而且其定义是规范和严谨的。UML 的表示法简短，容易学习，而且其含义有明确的定义(在 UML 中，这种定义是通过元模型描述的)。

UML 2.0 的构成及其与 UML 1.x 的比较如下。

UML 1.x 明确了沟通设计、传达设计的要点，以捕获需求，并将需求映射到软件的解决方案。UML 2.0 经过体系的重建，克服了 UML 1.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。

UML 2.0 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，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并行的活动。

(3)类图：用于表示类、接口及其之间的关系。

(4)对象图：用于表示类图中定义的类的对象实例，其配置是对系统的模拟。

(5)顺序图：用于表示重要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：用于表示类与组件集合的分级组织。

(12)状态图：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。

1.3 系统开发的解空间

在面向对象分析和设计方法中，因为 UML 是描述面向对象模型的标准化图形语言和表示法，本书将使用 UML 来完成面向对象分析和设计。

现代的面向对象分析和设计方法是基于模型的，综合使用用例建模、静态建模、动态建模和架构建模来描述软件需求、分析和设计模型，构成了系统开发的解空间。在用例建模中，系统的功能性需求按照用例和参与者进行定义；静态建模提供了系统的结构化视图，类按照其属性及其与其他类的关系进行定义；动态建模提供了系统的行为视图，对象交互用于表示对象之间的通信和协作，用来实现用例；架构建模是系统的核心，使用包、子系统和构件结构来描述系统的框架结构及框架中各个部分的连接关系。

本书描述的一种基于 UML 的软件建模和架构设计方法，如图 1.3 所示。用例作为整个分析设计的驱动，然后基于软件架构的理论和方法构造出软件的总体架构，以此为核心，进一步使用面向对象的静态和动态建模方法，完成以类为单元封装体和以构件为复合封装体的分析设计，最终生成代码，实现系统。整个过程是一种基于用例的高度迭代的软件开发过程。

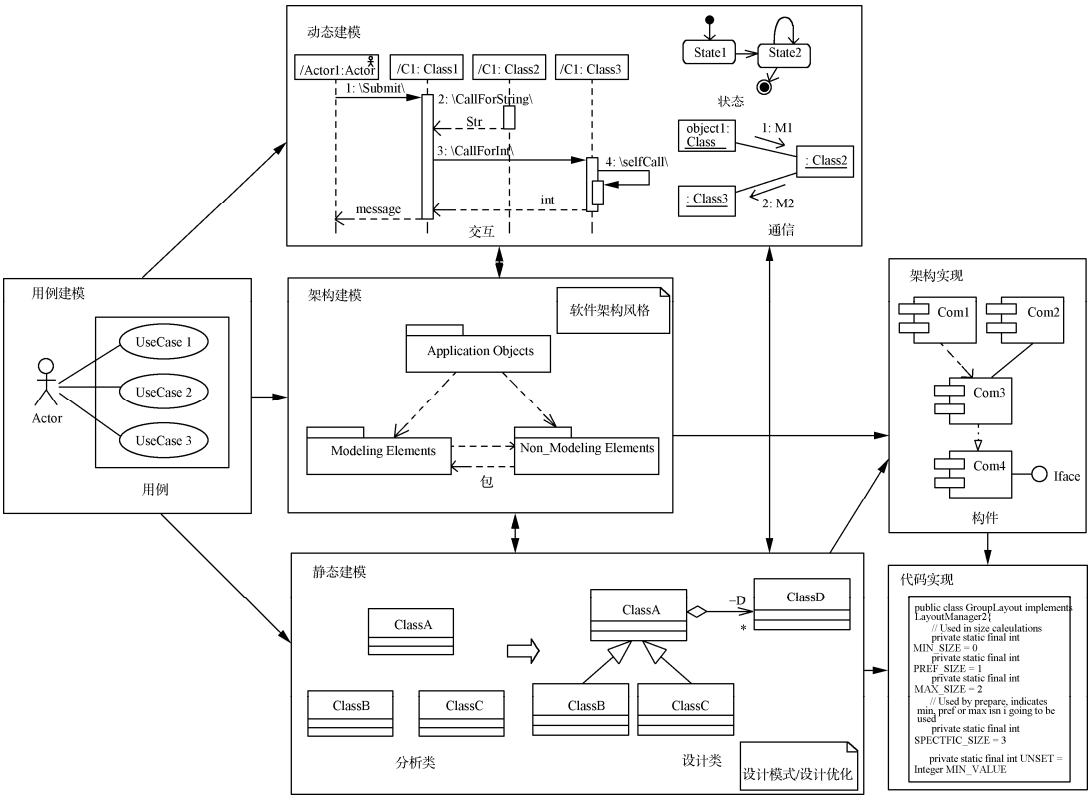

图 1.3　基于 UML 的软件建模和架构设计方法

1. 用例建模

在用例建模阶段，所开发的模型使用参与者和用例描述了系统的功能性需求，每个用例要开发一个叙述性描述的用例规约，在此过程中，用户的输入和主动参与是必不可少的。

2. 分析和设计

在分析和设计阶段，要迭代式地进行系统的静态建模和动态建模。静态建模定义了问题域中类之间的结构、关系，这些类及其关系描绘在类图中。然后，动态建模实现来自用例建模的用例，以显示每个用例中参与的对象及对象间是如何交互的，对象和它们之间的交互描绘在通信图或者顺序图中。在动态建模中，使用状态图来定义与状态相关的对象。

3. 架构建模

在架构建模阶段，要设计系统的软件体系结构。在此阶段中，模型被映射到一个运行环境中。问题域的分析模型被映射到设计模型的解空间中，此阶段提供包和子系统的组织准则来将系统组织为子系统。子系统被视为聚合或者复合对象，当分布式的子系统中使用相互通信的可配置构件进行设计时，要给予特别的考虑。

接着设计每个子系统。对于顺序系统，重点放在信息隐藏、类和继承的面向对象概念。对于并发系统，例如实时、客户端/服务器(C/S)或分布式应用，除了考虑面向对象概念，还需要考虑并发任务的概念。

4. 实现

在实现阶段，要将软件架构模型映射到可部署运行实现的模型解空间中。在此阶段中，包和子系统映射成组件，而其中的类则使用代码生成方法和优化策略生成可运行和部署的面向对象结构的代码。

1.4 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，主要包含方法、工具及过程三个基本要素。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，主要回答“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，主要回答“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，根据人们分析问题角度和方式的不同，软件工程开发方法可以笼统地分为传统方法(结构化方法)和面向对象方法。

1.4.1 传统方法

传统方法又称结构化方法，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。传统方法体现出对复杂问题“分而治之”的策略，但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，增加系统的总成本。

1.4.2 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

① 这种稳定性可以通过 UML 的类图进行表达。

的结构来提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，类是对象的“抽象”；类与类之间可以构成“继承”的层次关系；对象之间的互相联系是通过消息机制实现的，确保了对信息的“封装”，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”。

1.4.3 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。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，因此这种方法又称面向数据流的方法。图 1.4 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。图中的下半部分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 1.4 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 个主要的类，即顾客、服务员和厨师。图中标明了各个类具有的服务能力，如服务员主要是对顾客提供点菜和上菜的服务。值得一提的是，厨师和顾客类中都具有一个“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短横线标识，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)。顾客类中的品尝则是一个公有服务。

抽象类之间的关系是保证整个分析结构稳定的重要元素之一，图中顾客与服务员、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，而顾客与厨师之间并没有任何联系，符合该业务的一般规则，并且是一种固定的业务模式，所以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：需要当前这个就餐系统能够“礼貌待客”。需求变化的响应对于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筛选出业务变更的位置——服务员与顾客之间的点菜和上菜这两个过程。不幸的是，上述隔离过程需要设计及维护人员熟悉和理解整个业务流程，对于一些较大型的系统，往往这是很困难的一件事情。

① 此图的名字叫做数据流图(Data Flow Diagram，DFD)。

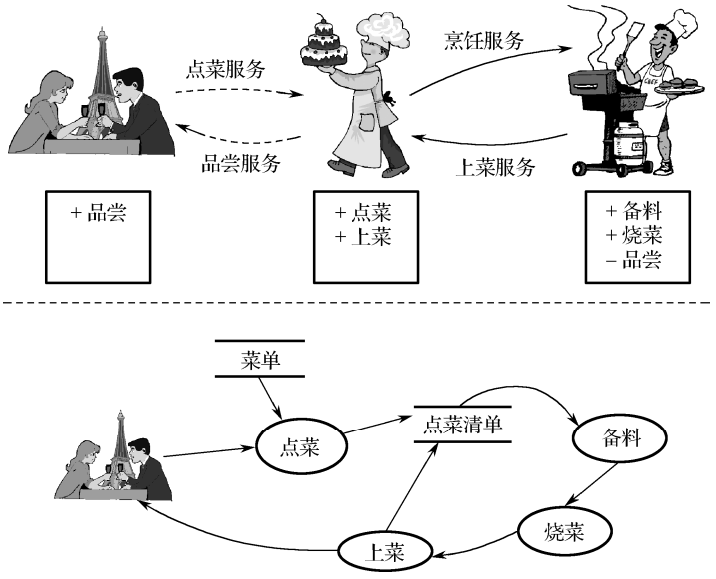

图 1.4　传统方法与面向对象方法的分析比较

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，因此也不需要去深入理解它们。由于类的封装性，我们可以直接把修改限定在一个特定的范围内，模块化的优势也就体现出来了。具体地，在本例中，可以在服务员类中添加一个私有的“问候”服务，并在该类内的点菜和上菜服务中使用此服务即可，其他地方及其他对象，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。

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，都比传统方法更具有优势，这也是面向对象方法在工业界越来越成为主流的原因。即使这样，也不能说可以完全摒弃传统方法。作为面向对象方法的补充，这些传统的、经典的分析和设计方法和过程目前仍然有着广泛的应用。

1.5　习题

1．软件工程主要包括哪些内容？

2．面向对象分析方法优于传统方法的根本原因是什么？可否借助图 1.4 或其他实例给出自己的理解？

3．UML 包含哪些重要的模型？它们在系统开发的解空间中作用如何？