HZ BOOKS
华章科技

软件工程技术丛书

软件系统架构与开发环境

Software architecture and development environment

郑建德 编著

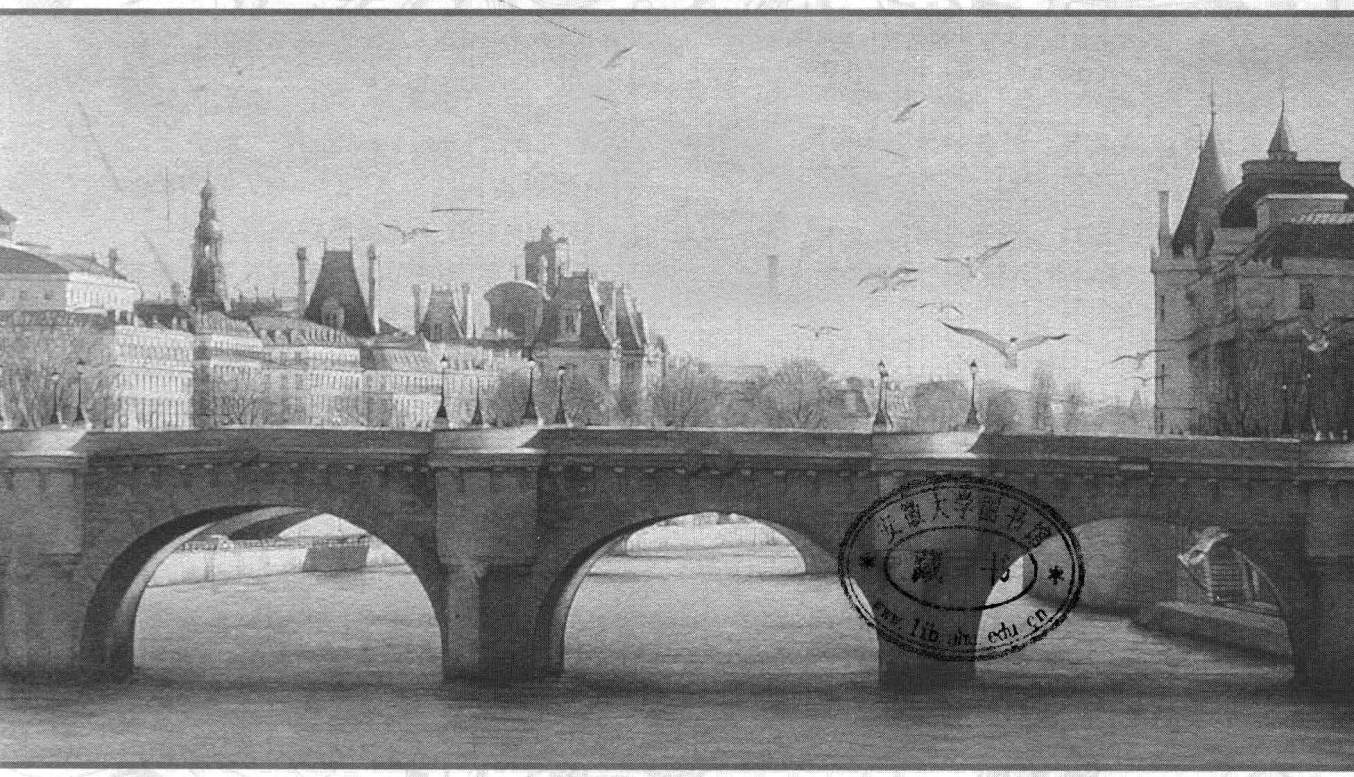
机械工业出版社
China Machine Press

软 件 工 程 技 术 丛 书

软件系统架构与开发环境

Software architecture and development environment

郑建德 编著



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

软件系统架构与开发环境 / 郑建德编著 . —北京：机械工业出版社，2013.9
（软件工程技术丛书）

ISBN 978-7-111-44002-4

Ⅰ. 软… Ⅱ. 郑… Ⅲ. 软件工程 Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字（2013）第 214919 号

近年来，新技术、新工具层出不穷，令工程技术人员目不暇接，在浩瀚的技术文档中摸索时容易陷入“见木不见林”的困境。为帮助读者解决上述困惑，本书作者对 20 多年来的行业工作经验进行了总结，透彻地分析了软件架构设计中的一些基本原理是如何在新的软硬件环境中重组、翻新的，清晰地揭示了软件架构设计与开发环境的关系实质。

本书介绍软件架构、软件开发模式、软件工程等基本技术原理，重点说明如何将一些流行的平台技术应用在软件系统架构设计中，以及 C-S 架构中的常用协议、中立化的信息交换语言，同时还介绍了中间件技术在分布式软件架构中的应用。本书内容包罗万象，分析透彻全面，同时还针对各技术点提供了 60 多个精练的演示程序，这些示例程序都经过作者精心调试、编译，可以直接运行，完整的源代码包含在本书附赠的光盘中。

本书是作者多年实践经验与教学经验的结晶，对软件开发技术及管理人员有很高的参考价值，同时也可供计算机及相关专业高年级本科生和研究生作为教材使用。

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：刘立卿
藁城市京瑞印刷有限公司印刷
2013 年 11 月第 1 版第 1 次印刷
186mm × 240mm • 19.5 印张
标准书号：ISBN 978-7-111-44002-4
　　ISBN 978-7-89405-098-4（光盘）
定　　价：59.00 元（附光盘）

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066　　投稿热线：（010）88379604
购书热线：（010）68326294；88379649；68995259　　读者信箱：hzjsj@hzbook.com

前　言

本书根据作者在厦门大学计算机系讲授八年的同名课程讲义编纂而成，其中系统架构方面的内容与2011年版《高等学校软件工程专业规范》中的相关内容不谋而合，涵盖了其核心课程“大型软件系统设计与体系结构”的教学大纲；软件开发环境方面的内容涉及一系列软件开发原理、技术和工具。作者1988年博士研究生毕业后一直在IT行业工作，2003年起担任厦门大学计算机系教授。本书实践性较强，除了用于教材之外，也非常适合作为软件工程师以及其他涉及软件技术的管理和技术人员的参考书。

本书共分五章，其中：

第1章介绍软件架构、框架、组件、对象、切面以及软件开发模式的知识，还介绍了软件工程流程、软件工具模型、CMM概要，以及软件配置管理系统和UML工具。

第2章介绍Java开发平台（含SE、EE、ME及Android）和.NET开发平台（含VC++）的关键技术，侧重介绍了Java的反射、序列化、线程、集合类、异常处理技术、VC++/MFC的对象反射特性、对象序列化、消息映射、文档－视图应用框架、动态链接库、.NET的CLR概念、Windows Forms/Web Forms框架以及Silverlight技术概要等，并从面向对象技术的角度对VC++/.NET与Java开发平台进行了比较。

第3章比较严谨地介绍了HTTP/WebDAV、JDBC以及LDAP协议，以它们为例分析标准协议在分布式架构中的作用，并以LDAP协议的应用为基础，介绍在Java EE平台上扮演重要角色的JNDI技术及其应用。

第4章从中立化信息交换语言的角度介绍ASN.1、HTML、XML（包括DTD与Schema），以及处理XML文档所用的SAX和DOM解析器模型，并传授基于JAXP的编程技术。

第5章介绍中间件技术，主要是Java EE中间件、CORBA中间件和Web Services中间件技术。本章关于Java EE的内容包括RMI、EJB、JMS、JDBC（数据源）、JTA/JTS等技术以及它们所依据的传统中间件模型；关于CORBA的内容包括ORB、IDL、IOR、COS、OMA；关于Web Services的内容包括WSDL和SOAP协议以及XFire、CFX和Axis框架。

本书的主要指导思想是第1章中介绍的一个软件工程工具模型。按照这个模型所表达的理念，对软件工具仅仅知其然是不够的，还应该知其所以然，即要了解这些工具背后的方法论，以及它们所遵循的基本准则。一个完整的软件专业教学体系可以通过数门课程帮助学生积累相关知识，但在普通计算机专业教学体系中很难做到这一点。作者八年前开设“软件系统架构与开发环境”课程，以及这次编写本书的目的都是想尝试解决上述问题。为了把多方面的知识汇集到一本教材中，剖析、简化与剪裁是必不可少的，并要花费不少心血才能把它

们有机地组织在一起，形成一个“知识架构”。了解以上知识架构可以避免在浩瀚的软件技术文档中摸索时陷入“见木不见林”的困境。近年来，新的软件技术层出不穷，几乎让人应接不暇。但是，除了少数创新点之外，它们大多是传统技术在新的硬件、软件环境中的重组或翻新。熟悉以上知识架构也有助于我们吸收和消化这些“新技术”。

为了构建一个简明扼要的知识架构，本书采用了多个方法。其中之一是从不同的角度来观察不同的知识点。学习没有捷径，却有“佳径”，循着正确的途径去学习往往可以收到事半功倍的效果。本书采用的第二个方法是利用一系列短小精练的演示程序来解释关键的知识点。这些演示程序都是作者创作或改编的，其清单列在正文后面，是本书的重要组成部分。这些程序的完整源代码包含在本书附赠的光盘中，作者对这些程序逐一进行过调试，它们都可以直接编译并运行。

本书或多或少得益于作者的实践经验与教学经验。尽管如此，作为一种尝试，本书难免有不成熟或不准确之处，敬请读者指正。

郑建德

致　谢

首先，我要借此机会感谢我的2009级和2010级研究生郑杭杰、林尚青、郑智强、陈腾、钟慧虹、许云源、傅希鸣，他们帮我把课件整理成为正文草稿，并帮我查找大量资料以充实其内容，从而减轻了我的工作量。我还要借此机会感谢机械工业出版社的温莉芳总编、刘立卿编辑以及我的同事赵志琢教授。我向来对著书之类的事敬畏有加。虽然这本教材属于编著，且又有项目的推动，但如果没有他们的鼓励与支持，我仍然很可能知难而退。

最后，我还要借此机会感谢我的妻子刘盈夏和我的女儿。这本书是我在厦门大学任教十年留下的印迹。我之所以能够踏踏实实地在高校担任这么多年的全职教师，全赖妻子在生活上的照顾与精神上的扶持，也要感谢女儿在海外奋斗自强免去了老父的后顾之忧。

郑建德

2013年7月

目　录

第①章

软件系统架构与软件工程

1.1 计算机及其软件系统

1.1.1 系统及其基本特性

我们从一般系统概念谈起。

“系统”（system）一词的渊源可追溯到柏拉图的《斐利布篇》、亚里士多德的《政治学》和欧几里得的《几何原本》，其原意是“总体”（total）、“群体”（crowd）或“联合体”（union），后多用于指为了某个目标而建立的集合体。本书主要讨论计算机软件系统，但了解几个关于系统的一般性概念有助于我们理解相关的方法论。

众所周知，科学技术是生产力，但具有讽刺意味的是，科学技术也是人类历史上最大的破坏力，即战争机器的组成部分，而且往往是在战争需求的推动下突飞猛进。系统科学也不例外，其发展就得益于美国国防部（DoD）的武器型号研究。20 世纪 70 年代出现的美军 FM-770-78 型号文档中给出的系统的定义是：一种由设备、技术与技能构成的，能够执行或支持某项战斗任务的综合体。经过半个世纪的沧桑，系统的意思已经泛化，但其相关词汇仍然多少带有一些军事色彩。

从某种意义上讲，系统并不是现实事物，而只是它们的模型。本书采用的系统模型具备如下几个特性，其中最重要的是其结构性。

1）可分解性。系统由元素（构件）组成，元素本身也可以是一个系统，即子系统，故系统一般可以递阶分解。图 1-1 以一个简化的 PC 机（硬件）系统为例，对此加以说明。

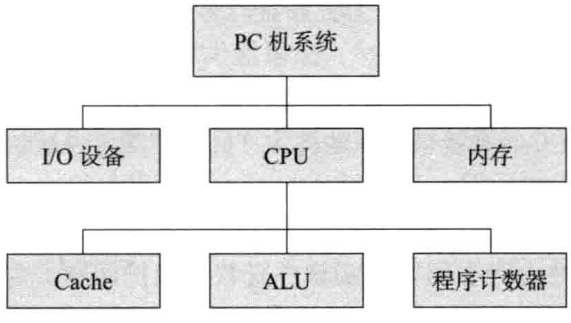


图 1-1　PC 机系统的递阶分解

2）结构性。结构性来源于可分解性。系统不是元素的堆积，而是其有机组成。元素之间的相互关联形成结构。由于元素可递阶分解，故系统结构一般具有相对性。

3）相对独立性。这包含两层意义。第一，系统与其环境之间具有明确的边界，这使得系统有一定的独立性；第二，系统并不孤立，每个系统都有一定的生存环境，即常说的上下文（context），系统的功能与性能都可因上下文的改变而改变。

4）集成性。系统的集成性表现在系统具有总体属性。系统的总体属性不是其元素属性的总和，而是后者集成后产生的新属性。好比车轮、车把等装配成一辆自行车后可以作为交通工具来使用，这个属性是任何单个元素都不具备的。总体属性与系统的元素及系统结构都有关，一般可分为功能属性和非功能属性两类。功能属性描述系统胜任特定任务的能力，实际上是特定用户需求的某种映射；非功能属性则是对功能属性的限定。好比自行车都可以作为交通工具，但其结实程度可能不同，可适应的骑行路面以及可承载重量也未必一样。常见的非功能属性有可靠性、可伸缩性和可维护性等。系统的非功能属性与系统结构之间的关系尤其密切。

1.1.2　分布式计算机系统

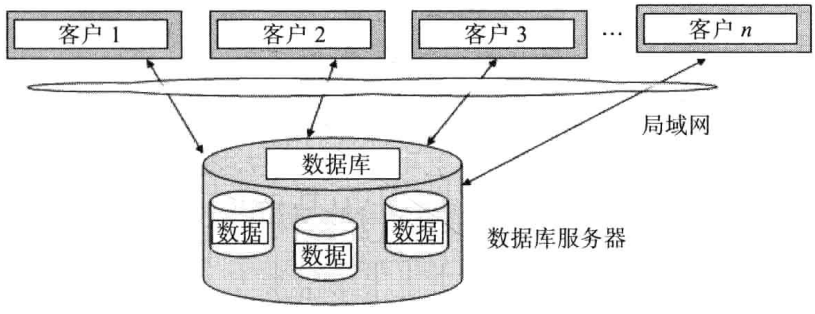
我们迅速把关注范围缩小为计算机系统，并把关注点放到分布式计算机系统上，利用它们显而易见的结构性切入主题。以下先简单介绍几个相关概念，它们的形成历史对于现代软件技术的发展具有比较深远的影响。

第一，集中式系统与分布式系统。计算机系统可分为集中式系统和分布式系统。前者的特点是资源及其用户在一个空间位置上，如早期的主机分时系统（也称为 Mainframe 系统）。在这类系统中，多个用户通过哑终端来使用孤立的主机，其中哑终端只有输入和显示功能，所有的计算（信息处理）都由主机实现。当前广泛使用的主要是分布式系统。它由若干自治的计算机联网组成，其特点是这些计算机能够协同工作，以实现一个统一的目标。与集中式系统相比，分布式系统有两大优势：其一是效率高，这主要得益于任务在不同计算机之间的合理分配；其二是可靠性高，这是因为用户和资源都可分布在多个空间位置上，从而克服单点依赖问题。因此，从 20 世纪 90 年代起，伴随着计算机网络技术的发展，原来分散的单机应用程序迅速向分布式应用系统发展。

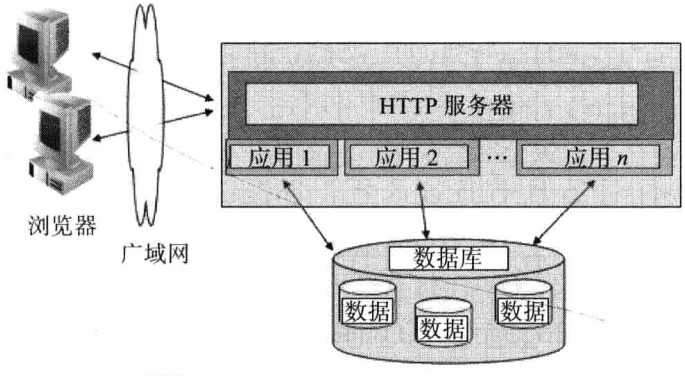
第二，C-S 与 B-S 应用系统。客户机 – 服务器（C-S）应用系统是最先发展起来的分布式应用系统。在 C-S 系统中，计算机可分为服务器与客户机两端。服务器提供共享资源（包括计算资源），客户机作为服务器的“前台”，除了提供用户界面之外，还可以提供本地计算能力，从多个方面降低整个系统的开销，提高其性能。随着用户需求的不断增长，客户机一端的逻辑往往日趋复杂，故 C-S 系统被形象地称为“胖客户端”系统。

到了 20 世纪 90 年代中后期，互联网（Internet）的兴起给 C-S 系统带来了两个方面的问题：其一是因为各种网络通信原因而产生的性能问题和可靠性问题；其二是由于客户大范围分布与客户端多样性而产生的客户端软件安装和维护问题。为了解决上述问题，人们推出了以万维网（World Wide Web，WWW）技术家族为基础的浏览器 –Web 服务器（B-S）应用系统。B-S 应用系统实际上是 C-S 应用系统的一种特例，如图 1-2 所示，它把原来在

客户端一侧的应用逻辑和显示功能分开，将应用逻辑挪到另一侧的 Web 服务器上。浏览器只为用户提供工作界面和少部分可以自动下载并直接运行的小应用，主要业务逻辑在服务器端实现。由于 B-S 应用系统实现了客户端的“零维护”，因此还可进一步降低系统运行和维护成本。



a）C-S 应用系统



b）B-S 应用系统

图 1-2 从 C-S 到 B-S 的演化

第三，企业应用系统。企业应用系统原来特指在信息管理系统基础上发展起来的用于企业内部资源管理和运营支持等的分布式系统，如企业资源规划（ERP）系统、运营支撑系统（OSS）。这类系统与早期 C-S 系统的一个重要区别在于除了客户机和各个部门的专用服务器之外，前者往往还引入了一种公共服务器，在其上运行一种称之为中间件的第三方软件，从而使整个系统形成三层乃至多层结构，如图 1-3 所示。关于“中间件”的概念我们在后面还要专门进行讨论，这里不再赘述。顺便提一下，随着相关技术的扩散，“企业应用”一词也在一般计算机技术领域中出现。例如在 Java EE 开发平台上，带有 ear 后缀的“企业应用组件”实际上是一种通用的软件模块。

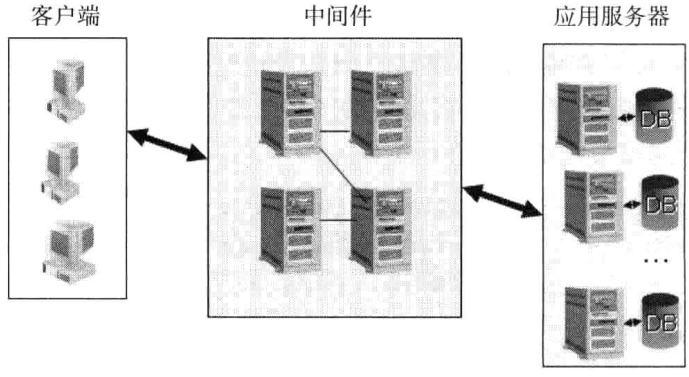


图 1-3　企业应用系统的三层结构

1.1.3　RM-ODP 及其视点模型

我们在本节中要介绍的另一个经典知识点是 ISO 的开放分布式处理参考模型（Reference Model of Open Distributed Processing），一般简称 RM-ODP。在计算机领域中工作的人员一般都熟知作为计算机网络基础的 OSI 七层参考模型（简称 OSI-RM），它实际上与 RM-ODP 紧密相关，可视为后者的通信技术模型。

分布式计算机系统的发展首先遇到的障碍是网络平台的异构问题，即由不同计算机厂商推出的网络产品、计算机软硬件产品在各个层面上出现的互连、互通与互操作问题。为了解决上述问题，就需要确立一种可遵循的、不依赖于特定厂家及其产品（包括操作系统、计算机硬件以及网络基础设施）的网络应用体系结构，即所谓的开放分布式处理结构。RM-ODP 正是在上述背景下产生的。本书想让读者了解的主要是 RM-ODP 关于分布式软件系统架构的视点模型，它与目前常用的软件系统架构视图之间或多或少存在某种渊源关系。

任何分布式应用系统都有许多不同类型的用户，由于它们所担任的角色不同，对系统的观察角度有所不同，它们所关心的问题的侧重点也有所不同，因此产生不同的视点，这些视点代表了原始系统在不同侧面上的抽象。开放分布式系统是一个庞大而复杂的系统，引入视点的概念有助于对其分而治之，从多个角度描述上述网络应用体系结构。RM-ODP 共引入五个视点，以下给出其粗略描述：

1）企业视点（enterprise viewpoint）。企业视点着眼于系统的应用场景与总体目标，主要关注：①系统要解决哪些问题，要实现哪些功能；②使用该系统的角色及其行为（包括行为所受约束）；③系统中的可识别对象，以及这些对象与角色之间的关系（哪些角色使用哪些对象）。

2）计算视点（computational viewpoint）。计算视点对系统的功能进行分解并分配到上述可识别对象中。这些对象通过封装隐藏了内部信息，它们之间只通过接口进行交互，可以在不影响其他对象的前提下安装、升级或替换。

3）信息视点（information viewpoint）。信息视点关注的是信息的语义及信息处理过程。它描述系统中的信息元素（包括类型、结构和处理形式），并进行信息流建模。

4）工程视点（engineering viewpoint）。工程视点主要提供系统主体结构与资源分布的抽象描述，并在此基础上定义各种对象的运行环境（上下文）。工程视点描述的系统主体结构可称为技术中性架构（Technical Neutral Architecture, TNA）。

5）技术视点（technology viewpoint）。技术视点关注的主要是实现系统主体结构所需要的具体技术元素，包括计算机网络、计算机硬件机型与配置、操作系统、数据库系统、程序开发语言等。相对于工程视点中的 TNA，技术视点描述的系统主体结构可称为 TSA（Technical Specific Architecture，技术特定架构）。

1.1.4 软件系统及其质量属性

我们下一步的任务是把计算机系统这个模型拆解成硬件系统和软件系统，然后撇开前者来讨论后者。顾名思义，计算机软件系统即计算机系统中的软件所构成的子系统。软件主要指运行在硬件机器上的程序代码及其相关数据，它们是计算机系统的组成部分，但与硬件机器之间具有相对独立性。我们都知道一个计算机系统中可以集成多个软件“包”，而每个软件包往往可以被集成到不同的计算机系统中。软件系统的上述相对独立性是我们建立软件系统这个模型的基本依据。根据我们解决特定问题的需要，一个计算机系统中往往可以拆出多个软件系统。

1. 软件系统的质量属性

在既定的运行平台上，软件系统对整个系统目标的支持能力称为软件质量。它可以用一系列系统总体属性来描述。一个软件系统的运行平台可以分为硬件平台和软件平台，前者通常指主机硬件和网络基础设施等，后者指运行在同一个硬件平台上的其他相关软件系统，如操作系统。它们一起构成所述软件系统的运行环境。

我们在前面谈到，系统的属性可分为功能属性与非功能属性，功能属性描述系统胜任特定任务的能力，非功能属性是对功能属性的限定。对于大型软件系统而言，非功能属性的重要性不亚于功能属性。常用于描述软件系统质量的非功能属性包括：正确性、可用性、有效性、可靠性、鲁棒性、可伸缩性、可维护性、可理解性、可验证性、互操作性、可移植性、可复用性。其中：

- 正确性（correctness）：正确性相对于功能需求而言，满足了功能需求定义的软件都是正确的。
- 可用性（usability）：可用性即软件产品对用户“友好”的程度，很大程度上取决于用户界面的设计。
- 有效性（efficiency）：有效性指软件产品有效地使用资源的能力，资源包括内存、CPU 时间、通信带宽等。
- 可靠性（reliability）：可靠性用于度量软件产品的可信赖程度，一般定义为一段时间内产品出现故障的概率。
- 鲁棒性（robustness）：鲁棒性描述软件产品承受意外冲击（如操作错误、硬件故障等）的能力。

- 可伸缩性（scalability）：可伸缩性指软件适应负载变化与用户增减的能力。
- 可维护性（maintainability）：可维护性指软件系统维护的难易程度。软件的维护又可分为正确性维护、适应性维护和完善性维护。正确性维护，主要是消除软件中残留的错误；适应性维护，主要是根据环境的改变对软件进行必要的调整；完善性维护，目的是满足用户不断提升的需求。
- 可理解性（understandability）：可理解性指系统具有清晰的结构，能直接反映问题的需求。可理解性有助于控制软件系统的复杂性，并支持软件的维护、移植和复用。
- 可验证性（verifiability）：可验证性指验证软件属性的难易程度。
- 互操作性（interoperability）：互操作性指一个软件系统与其他软件系统共存和协作的能力。
- 可移植性（portability）：可移植性指现有的软件产品经适度修改可适应新环境或满足新需求的能力。
- 可复用性（reusability）：可复用性指软件的全部或部分能够复用的能力。软件的复用指在两次或多次不同的软件开发过程中重复使用相同或相近的软件元素（软件代码、软件结构、软件设计文档甚至领域知识等）。常见的软件复用包括代码复用和结构复用。软件复用不仅能提高开发速度、降低开发成本，还可稳定软件质量，是软件开发能力成熟的一个重要标志。

我们可以从使用、维护和开发三个角度来整理上述软件质量属性，如图 1-4 所示。

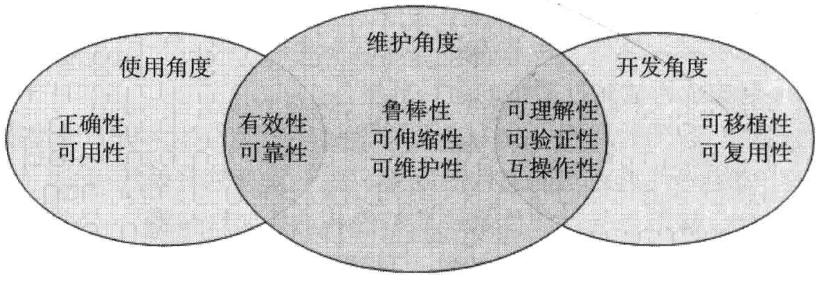


图 1-4　软件质量的主要属性

2. 软件质量的两对视图

软件系统的质量属性有两对视图，即外部视图 – 内部视图和产品视图 – 流程视图。

外部视图即用户可见的视图。从外部视图获得的软件质量称为外部质量，通常与用户提供的软件运行环境以及软件的使用时间有关。内部视图是开发人员可见的视图。从内部视图获得的软件质量称为内部质量，它可以从一个更可信的角度反映软件的“优劣”。一般地说，内部质量决定外部质量。

产品视图包括对软件产品（含其文档）进行安装、使用并观测其运行效果等所取得的一系列系统属性指标。从产品视图获得的软件质量称为产品质量，它反映的主要是软件开发目的是否达到。流程视图指通过了解产品的开发流程对其质量进行的评估。从流程视图获得的软件质量称为流程质量，它关注的主要是软件开发过程是否能够达到某个水平。一般情况

下，客户能够了解的产品质量信息多于外部质量信息，但不及内部质量信息，因此，透明的流程质量至关重要。

对流程质量进行管控可以间接实现对内部质量的管控，从而影响甚至决定产品质量。这也是我们后面要谈到的 CMM 之所以备受重视的一个基本原因。

3. 软件组成的一个参考模型

组成计算机软件系统的要素不是其物理形态（载体形态），而是凝聚其中的智力与知识。因此，软件产品的价值及其评价体系都与其他产品有所区别。

传统的计算机软件组成要素除了程序和数据之外，还包括文档。其中程序是对计算机的处理对象和处理规则的描述，是软件开发人员根据用户需求开发的、用程序语言描述的、适合计算机执行的指令序列；数据包括初始化参数、输入数据及数据格式，往往既是程序的处理对象，也是程序的运行依据；文档包括软件设计、开发、使用和维护过程中产生的全部文本材料，是程序的资源说明。除了上述三个要素之外，后面要讨论的流程和架构（主体结构）实际上也已成为现代计算机软件中两个不可缺少且具有相对独立性的要素。考虑到它们对于提高计算机软件的质量所具有的重要意义，以及它们在本教材中所扮演的重要角色，我们在结束本节之前，特对传统的计算机软件组成概念加以扩充，提出如图 1-5 所示的计算机软件组成参考模型。本书的其他章节将围绕着其中的流程与架构两个扩充要素展开。读者将看到，这两个要素之间也有千丝万缕的联系。

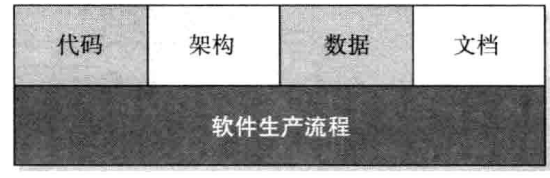


图 1-5 扩充后的软件组成参考模型

1.2 软件系统的流程要素

这一节讨论软件系统的流程（也称为过程）要素，以及它对于优势软件架构的形成所具有的重要意义。

1.2.1 系统工程概要

20 世纪 40 年代，美国贝尔实验室研制电话通信网络时，将研制工作分为规划、研究、开发、应用和通用工程五个阶段，同时期美国研制原子弹的曼哈顿计划也应用了类似的原理进行协调。一般认为系统工程（system engineering）概念就是在这个时期产生的。作者认为，应用系统工程方法而取得重大成果的两个更经典的例子是美国的登月火箭阿波罗计划和北欧跨国电网协调方案。

系统工程的概念定义也随着时间的推移不断演变，1978 年钱学森给出的系统工程的定义是："系统工程是组织管理的技术。把极其复杂的研制对象称为系统，即由相互作用和相互依赖的若干组成部分结合成具有特定功能的有机整体，而且这个系统本身又是它所从属的一个更大系统的组成部分。系统工程则是组织管理这种系统的规划、研究、设计、制造、试验和使用的科学方法，是一种对所有系统都具有普遍意义的科学方法。"2004 年国际系统工程协会（INCOSE）给出的定义是："系统工程是一种实现成功系统的跨学科的工业方法和手段。"

在本书中，我们把为了实现特定的目标，在一定的准则和规范指导下的系统开发过程统称为系统工程。

系统工程涉及的内容比较复杂，可以包括生命周期模型、形式化流程、配置管理、各种运筹学工具（PERT、GERT 等）以及各种风险评估技术等，但其核心思想却很简单，引用亚历山大的名言就是分而治之（divide and conquer），即先把系统分解为适当的模块并分别进行开发或采购，再把各个模块集成起来，从而完成整个系统的开发。

以下简单介绍一下系统工程的结构化生命周期模型，作为我们后面讨论软件工程结构化生命周期模型的基础。结构化生命周期模型也称为瀑布模型（waterfall model），它从提出需求直到系统报废为止，刻画了一个工程系统工程从起始到完成的整个周期，是系统工程中最常用的生命周期模型。本书采用的瀑布模型包括需求分析、系统设计、子系统开发、系统集成、系统安装、系统演化和系统退役共七个阶段，如图 1-6 所示。

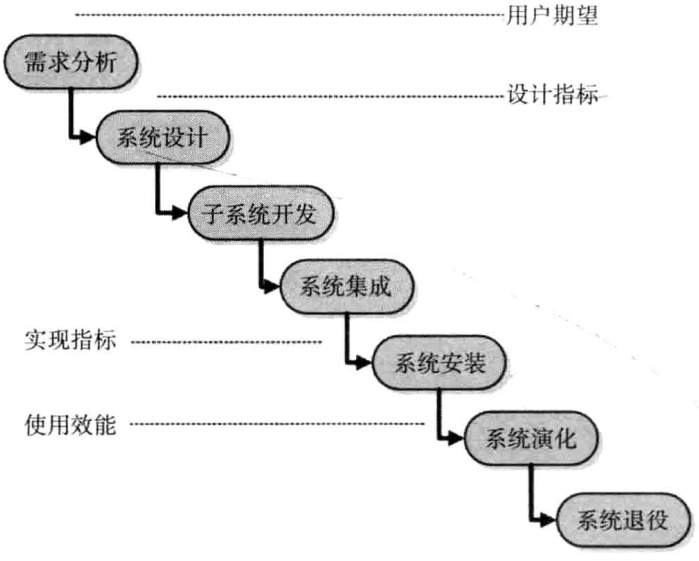


图 1-6　系统工程生命周期模型的瀑布模型

1）需求分析（requirements analysis）阶段。需求分析也称为需求定义。需求定义的内容包括对功能需求和非功能需求的定义，后者包括前面介绍软件质量属性时谈到的有效性、可靠性、鲁棒性、可伸缩性、可维护性等。需求定义也可以采用“减法”形式，即给出不符合需求的系统特征以及不可接受的系统行为。

2）系统设计（system design）阶段。如上所述，系统工程的核心思想是分而治之，系统设计阶段完成的主要任务就是系统的分解。这一阶段的任务还包括子系统及其接口的定义以及系统的总体需求在各子系统中的分配。

3）子系统开发（sub-system development）阶段。子系统开发阶段可包括多个并行的子系统开发项目，一般需要引入关键活动路径识别等方法对各子系统的开发时序进行规划，从而实现对整个系统开发进度的控制。在很多情况下，通过 COTS（Commercial Off-The-Shelf，现货软件）采购有利于加快开发进度，节省开发经费。

4）系统集成（system integration）阶段。在这个阶段中，要根据系统的设计方案，把由上一阶段获得的子系统合并，组成一个完整的系统，并进行联调与测试。系统集成阶段有时也称为系统的场外联试阶段。

5）系统安装（system installation）阶段。许多系统经过集成联调和测试后，需要拆分运到现场，并在现场（即系统真正发挥效用的环境中）重新安装就位。这个过程就是系统安装阶段，有时称为系统的现场联试阶段。在这个阶段中，往往会暴露一些未预见或不可预见的问题，包括物理不相容性问题、界面协调性问题以及数据转换问题等。这些问题一般是由于设计阶段对系统运行环境所作的假设不正确而引起，多数可以采取弥补措施加以解决。不过，若在这个阶段出现无法解决的问题，则有可能导致前功尽弃，即整个系统开发完全失败或部分失败。这是结构化生命周期模型的一个重要弱点。

6）系统演化（system evolution）阶段。引入系统演化阶段的目的是延长系统的生命周期，提高系统的效益。大型系统往往需要巨额投资，一般都具有较长的预期生命周期。但随着需求的变化和其他新系统的引入，系统与其运行环境之间的耦合可能出现一些无法预测的问题，因而需要进行必要的修改，这就是系统演化。

系统演化过程是有限的。一方面，演化代价将随着系统寿命期的延长和原始设计资料的流失而迅速上升；另一方面，系统经过反复修改之后，其基本结构逐渐被破坏，继续保留其残余部分将失去意义。

7）系统退役（system decommissioning）阶段。系统退役阶段指一个系统停止服务，并从其工作环境中被去除的过程。系统退役阶段也有许多重要的工作需要完成，例如：处理系统中可能污染环境的材料，回收系统中可以循环利用的部件甚至模块，销毁敏感数据等。

1.2.2 软件工程概要

软件工程的兴起主要源于20世纪70年代前后由于大量软件开发项目失败而出现的所谓“软件危机”。为了应对上述危机，北大西洋公约组织（NATO）在1968年举办了首次软件工程学术会议，在该会议中建议“软件开发应该是类似工程的活动”，并提出“软件工程”的概念。IEEE于1983年把软件工程界定为“开发、运行、维护和修复软件的系统方法”，并于1993年进一步将其定义为“把系统化、规范化、可度量的途径应用于软件开发、运行和维护的过程”。关于软件工程的定义似乎一直在演变，但目前比较普遍的观点是软件工程由方法论（方法和技术）、工具和流程三要素组成。软件工程中的各种方法论是完成软件工程项目的基本手段；软件工具能够自动或半自动地支持软件的开发、管理和文档的生成；而通过流程则可以对软件开发的质量、进度、成本等进行评估、管理和控制。

软件工程实际上是把系统工程的一部分即关于软件开发的部分独立出来自成体系，其主要目的当然是强调软件开发的特殊性，提升软件开发的重要性（不再附属于硬件开发），其基础则是软件开发的相对独立性。软件工程继承了系统工程的基本思想，即分而治之的思想，也继承了系统工程的许多方法和技术，包括结构化生命周期模型（即软件生命周期模型）、配置管理（即软件配置管理）等。理所当然，软件工程引入形式化流程，特别是贯穿于软件开发各个环节的软件工程流程（Software Engineering Process，也称为软件生产流程）。本节的