Software Engineering Theory and Practice , Second Edition

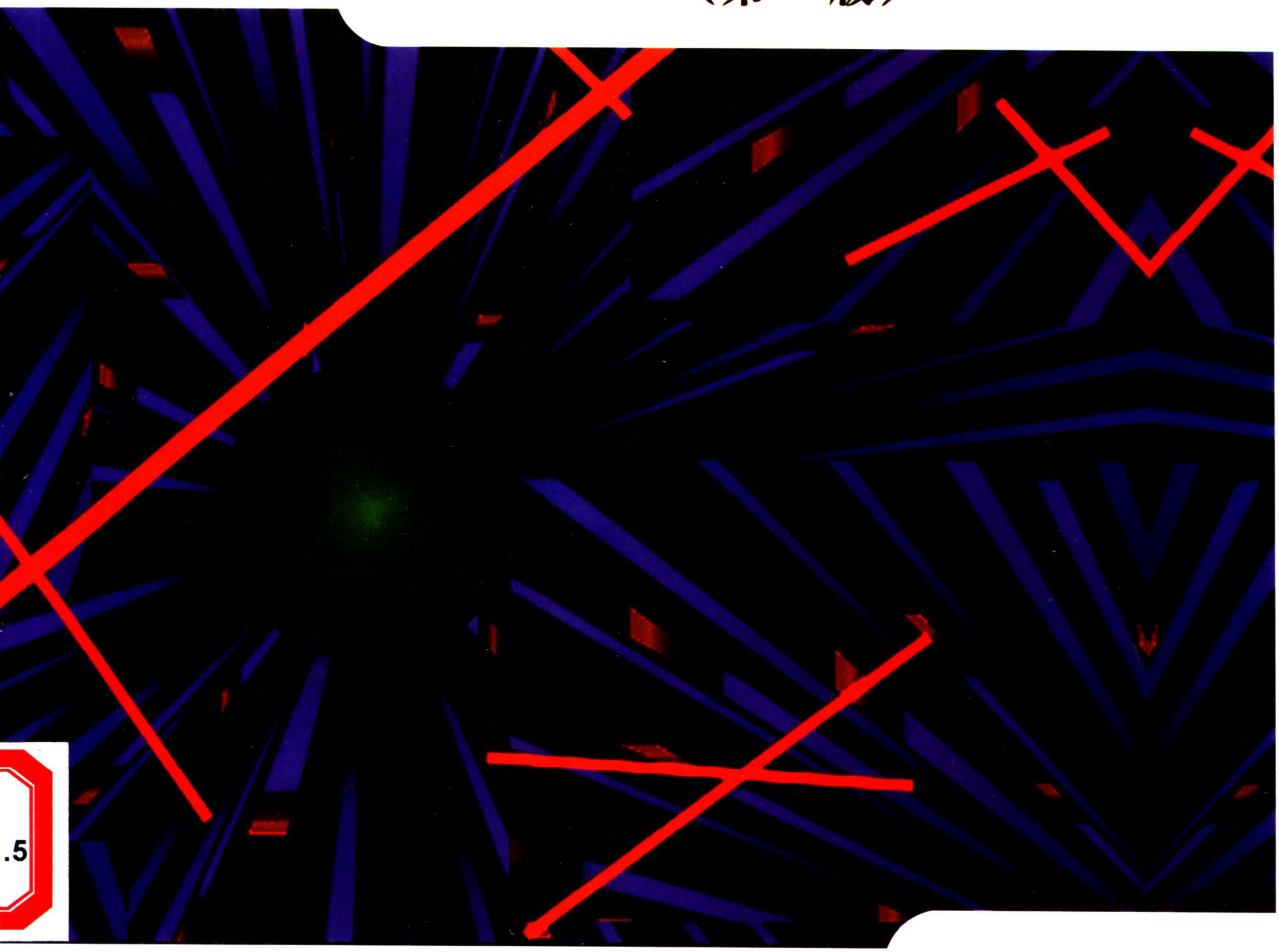
国外软件工程教材

PEARSON Prentice Hall

软件工程

理论与实践

（第 2 版）



[美] Shari Lawrence Pfleeger 著

吴丹　史争印　唐忆 译

清华大学出版社

软件工程 理论与实践（第2版）

Software Engineering Theory and Practice , Second Edition

"作者将其宝贵的实践经验融入到书本中，增强了理论知识的效果。本书的写作风格充分体现了作者的实力。"

Anita J. LaSalle 教授
美国大学

"这是市面上最好的书籍之一。它真实地涵盖了软件工程领域的理论和实践。"

Sung Shin
南达科他州大学

本书通过丰富的实例，将概念彻底地融入到了实际应用中。

Pfleeger 博士不断地将最先进的软件工程方法应用到两个个案研究（一个真实的信息系统和一个有大量需求的实时系统）中，将方法与实践彻底地结合。

新亮点：

- 第 6 章：考虑对象——大量扩展了对面向对象开发方法（已被许多公司贯彻的一种编程观点）的介绍。
- 第 14 章：软件工程的前景——讨论了软件工程中技术传播的重要性和决策的作用。
- 学期项目——这个实践性的项目将本书的概念融入到一个切实的例子中。

作者将概念应用到了真实的环境中——从宏观上讨论了开发小组，微观上讨论了单个的开发人员。

作者简介

Shari Lawrence Pfleeger 是 Systems/Software 公司（一家软件工程和技术的专业咨询公司）的总裁，她经常在马里兰大学的计算机科学系教授研究生课程和进修课程。Pfleeger 由于其在软件工程经验研究领域的工作成就而享有盛名，她著有大量相关领域的书籍和论文。Journal of Systems and Software 多次将 Pfleeger 评为当今顶级的软件工程研究人员。Pfleeger 现任 IEEE Transactions on Software Engineering 的助理编辑，她还是 Prentice Hall 的 Software Quality Institute 丛书的编委会成员。

ISBN 7-302-07009-1

9 787302 070092 >

定价：54.00 元

PEARSON Prentice Hall

www.PearsonEd.com

封面设计：杨月静
读者联系电话：（010）82896448，82896450
网　址：http://www.khp.com.cn
E-mail：khp@khp.com.cn

第 1 章　软件工程概述

本章将介绍

- 软件工程的含义
- 软件工程的历史轨迹
- 什么是“好的软件”
- 系统方法的重要性
- 从20世纪70年代以来软件工程发生了哪些变化

软件无处不在，有时我们想当然地认为它使我们的生活更加舒适、方便和高效。拿准备早餐面包这样的简单任务来说，烤面包机中的代码控制着烤好的面包片颜色的深浅，以及什么时候将烤好的面包片弹出来。程序控制并调节电能的输送，把电能送到各个房间，软件为我们使用的能源计账。实际上，我们可以使用自动程序支付电费、订购杂物、甚至可以去买一台新的烤面包机！今天，软件不但可以明确地、而且可以在后台虚拟地为我们生活的各个方面工作，包括影响我们健康和福利的关键系统。正因如此，软件工程比以往任何时候都更加重要。优质的软件工程实践必须确保软件给我们的生活提供积极的作用。

本书针对软件工程中的关键问题，介绍了已有的软件技术和工具，以及它们如何影响我们构建和使用的最终软件产品。我们不但强调理论而且强调实践：我们知道什么，以及如何把所知道的应用到一个典型的软件开发或软件维护项目中。我们还研究那些目前未知的，但有助于产品更加可靠、安全、有用、可用的问题。

首先我们考虑如何分析问题并提出解决方案。然后，研究计算机科学问题和工程问题的区别。我们的最终目标是给出高质量的软件的解决方案，并考虑那些对质量有益的特性。

我们还要了解软件系统开发人员已取得的成功。通过研究几个软件失败的例子，可以看看在控制软件开发质量的艺术方面，我们已经取得的进展以及将要取得的进展。

接着，讨论软件开发过程中涉及的人员。在描述完顾客、用户和开发人员的角色和责任之后，我们转而研究系统本身。我们看到，可以把系统看作是和一个活动集合相关联的一组对象，而且该组对象可用一个边界围住。换句话说，从工程师的角度考虑系统：开发一个系统很像建造一座房屋。定义了构造系统的步骤以后，就要讨论每个步骤中开发小组的角色了。

最后，我们讨论已经影响我们实践软件工程方式的一些变化。我们给出了Wasserman的8个观点，并和我们的软件实践结合成一个统一的整体。

1.1 什么是软件工程

作为软件工程师，我们要利用自己的计算机和计算技术知识来解决问题。通常情况下，我们正在处理的问题与一台计算机或已经存在的计算机系统有关。但是，有时候问题潜在的困难与计算机没有任何联系。因此，首先必须理解问题的本质。在有些特殊情况下，我们必须非常小心，不要把计算的机器或技术强加于我们遇到的每个问题。我们必须首先解决这个问题。然后，如果需要的话再把技术当作一个工具来实现解决方案。在本书的剩余部分中，假设经分析已知计算机系统对于解决手头特殊问题是必需的或是希望用到的。

问题求解

许多问题规模很大，有时处理起来很棘手，尤其是以前从未解决过的新问题。因此，研究它的时候首先要进行分解，也就是说，把问题分割成可以理解并能处理的小块。这样，我们就可以通过描述较小的问题和它们之间关系的集合来描述较大的问题。图1.1说明了如何进行分析。请注意，它们之间的相互关系（即图中的箭头，以及子问题的相对位置）与子问题本身同样是非常重要的。有时，正是这种相互关系而不是子问题的简单属性提供了解决大型问题的线索。

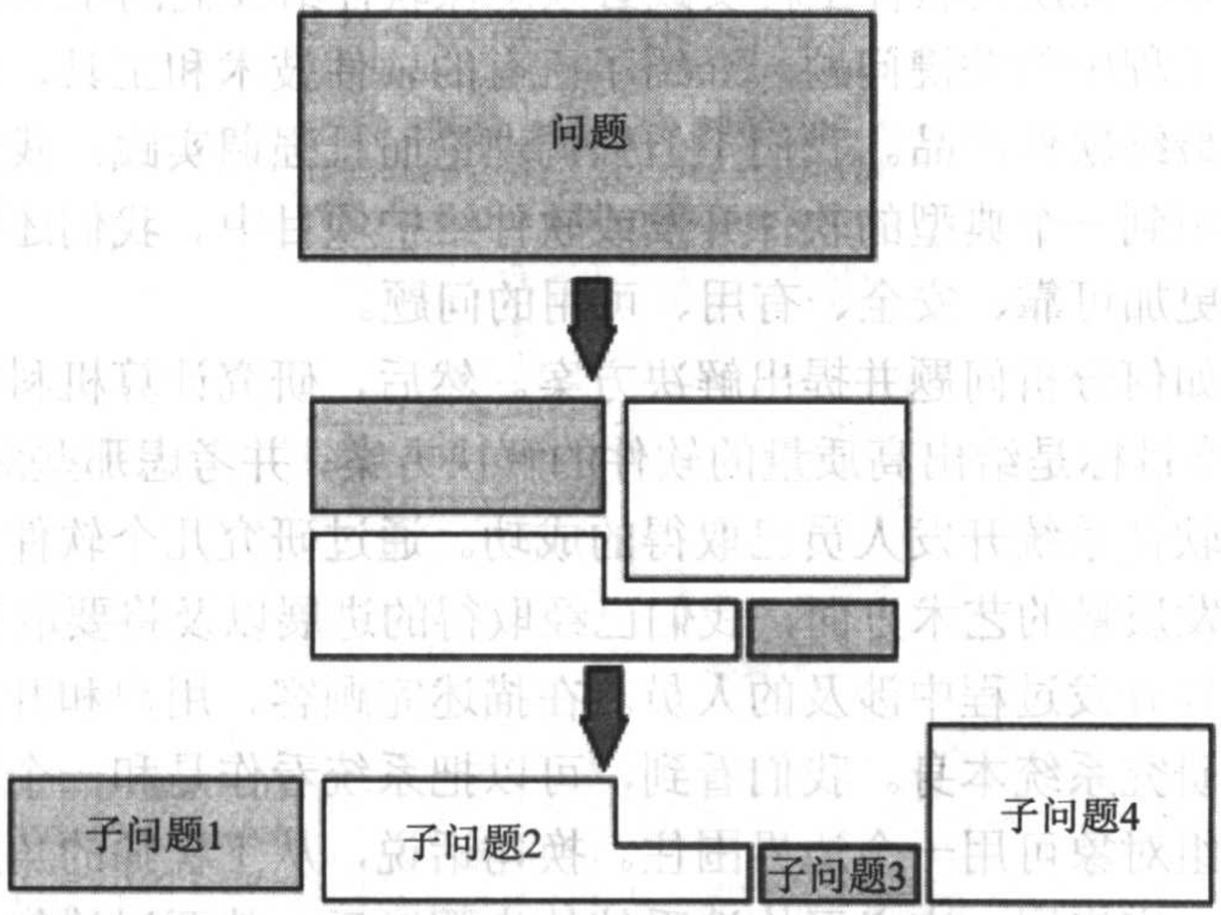


图 1.1　分析的过程

分析了这个问题以后，就必须根据描述问题不同方面的组件来构造解决方案。图1.2说明了这个逆向过程：合成是把小的构造块组合成一个大的结构。正如我们分析的，单个解决方案的合成与寻找解决方案本身同样具有挑战性。其原因可以用写一本小说的过程加以说明。字典中包含有写作过程中用到的所有可能词汇。但是，写作最困难的部分是决定如何把单词组成句子，以及如何把句子组成段落，乃至把章节组成一本完整的书。因此，任何解决问题的技术都应该有两部分：分析问题确定其本质，然后，基于分析合成解决方案。

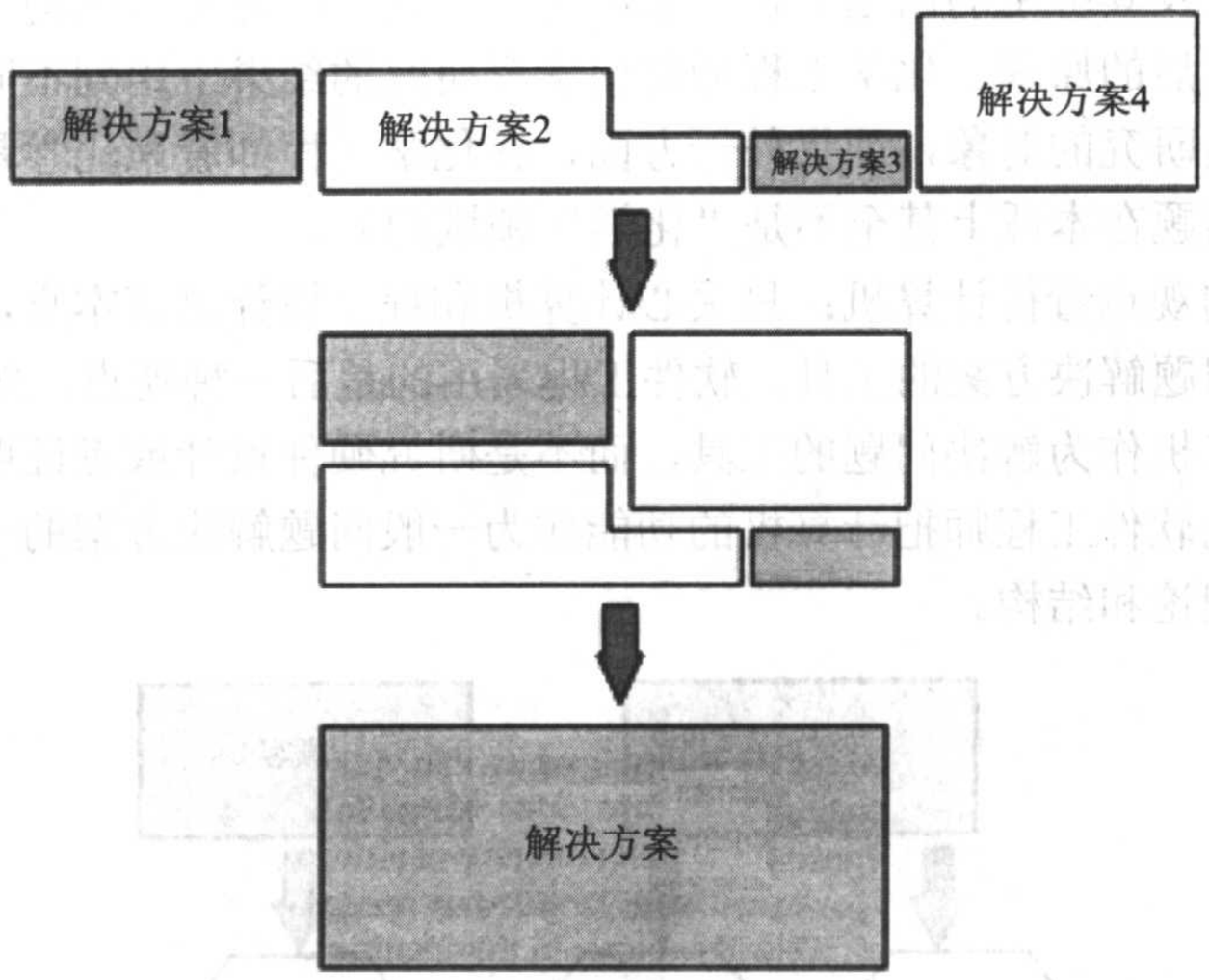


图 1.2　合成的过程

为了帮助我们解决问题，可以使用多种方法、工具、过程和范例。方法或技术是产生某种结果的形式化过程。例如，厨师可能严格按照一系列时间和顺序来把成分组合成调味汁，以使调味汁浓厚而不会结块或分离。准备调味汁的过程涉及到时间和成分，但并不依赖于所使用的烹调工具。

工具是用来更好地完成某件事情的设备或自动化系统。这个“更好地”是可以使我们更准确、更有效或更高产的工具，或者可以提高最终产品质量的工具。例如，我们使用打字机或键盘和打印机写信，这样写出的文本就比手写的容易阅读。再如，我们使用剪刀作为工具是因为用它裁纸要比直接用手更快更直。然而，要做好一件事情，工具并不总是必要的。比如，是烹调技术而不是厨师用的锅和羹匙使调味汁的味道更好。

过程就像食谱：即协调地产生特殊效果的工具和技术的组合。正如我们将在后面的章节中看到的那样，测试计划描述了测试过程；它们能够告知在什么情况下用哪种工具处理哪个数据集合，以便我们确定软件是否满足了需求。

最后，范例就像烹调风格；它代表软件开发时一种特定的方法或哲学。就像我们能够把法国烹饪与中国烹饪区别开来一样，我们也能够把范例区别开来，如面向对象的开发范例和基于过程的开发范例。一种范例并不绝对地比另一种范例好；每一种范例都有优缺点，可能在某种情况下一种范例要比另外一种更合适。

软件工程师使用工具、技术、过程和范例来改进软件产品的质量。他们的目标就是使用高效的、高产的途径产生解决问题的有效方案。下面的章节将重点介绍支持我们所说的开发和维护活动的特殊方法。关于工具和技术的最新链接列在本书的WWW主页中。

软件工程师适合做什么

为了理解为什么软件工程师适合计算机科学领域，让我们讨论另外一个学科的例子。

考虑有关化学研究及其应用的问题。化学家研究化学物质：它们的结构、它们的相互作用以及在它们行为背后的原理。化学工程师把化学家研究的结果应用到不同的问题之中。化学在化学家看来是研究的对象，而另外一方面，在化学工程师看来化学是用来解决一般问题的工具（这个问题在本质上甚至不是“化学”领域的）。

可以用类似的观点看待计算机：只关心计算机和程序设计语言本身，或把它们看作是用来设计和实现问题解决方案的工具。软件工程采用的是后一种观点，如图1.3所示。软件工程师侧重把计算机作为解决问题的工具，而不是研究硬件设计或者证明算法理论。本章后面的部分将讨论软件工程师把计算机的功能作为一般问题解决方案的一部分，而不是作为计算机本身的理论和结构。

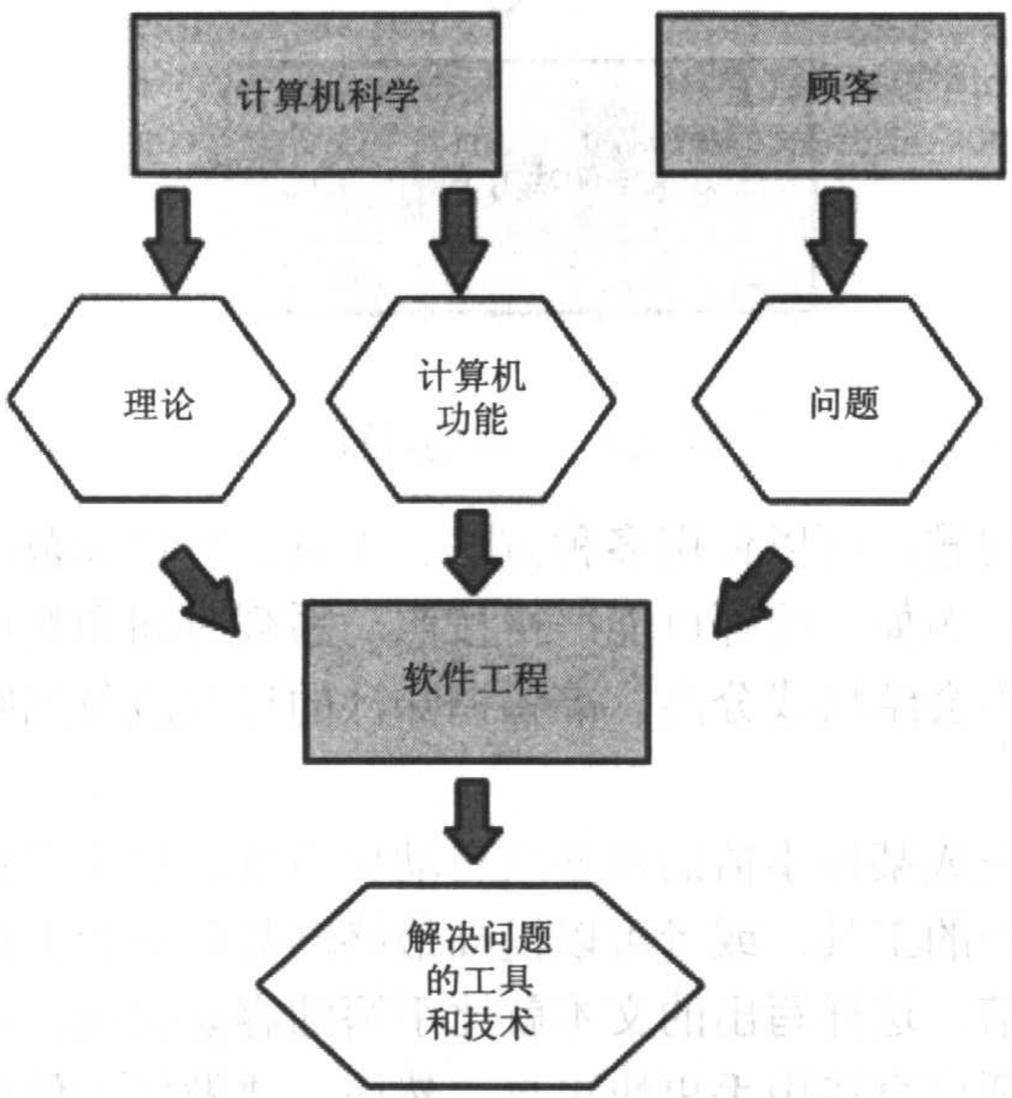


图 1.3　计算机科学和软件工程之间的关系

1.2　软件工程取得了哪些进展

编写软件是艺术也是科学，作为一名计算机学科的学生，理解为什么这样是很重要的。计算机科学家和软件工程研究者研究计算机的工作原理，并建立使计算机的生产率和效率更高的理论。当然他们也设计计算机系统并编写程序在这些系统上运行，这是涉及到艺术感、天才和技巧的实践。在一个特定的系统上完成某个特殊的任务可能有很多方法，但是其中一些方法要比其他的更好。某种方法可能更高效、更精确、更易于修改、更容易使用或更便于理解。任何黑客都能够编写代码使其运作，但是要编写健壮的、易于理解和维护的并且能以最有效、最可行的方式完成工作的代码，必须具备专业软件工程师的技巧和思路。因此，软件工程是关于设计和开发高质量软件的科学。

在研究要产生高质量软件系统需要什么之前，让我们回顾一下已经取得的成就。用户对现有软件系统很满意吗？回想文字处理、表格处理、电子邮件或先进电话等出现之前的

生活，软件已经使我们比以往任何时候更快、更有效地完成任务。软件支撑了在医学、农业、交通和其他许多领域的生活维持或生命救治方面的改进。另外，软件使我们能做以前从未做过的事情，如显微外科手术、多媒体教育、机器人技术等。

然而，软件并非没有问题，系统功能常常令人失望。很多人都听说过系统几乎不能工作的例子。大家也都写过有缺陷的程序：代码尽管已足以“及格”或足以证明某个方法的可行性，但它仍然有缺陷。很显然，当开发要交付给客户的系统时，这种行为是不能接受的。

课堂教学项目中的错误和大型软件系统中的错误之间有着天壤之别。事实上，软件的缺陷和开发无缺陷软件的困难是科学文献和日常工作中经常讨论的问题。有些缺陷只是比较烦人；而有的缺陷则要花费大量时间和金钱；甚至还有一些可能会威胁生命。附注1.1解释了缺陷、错误和故障之间的关系。让我们来看几个故障的例子并了解出错原因。

附注1.1　描述错误的术语

通常，我们在软件中谈到的“错误”（Bug）有很多意思，它依赖于上下文。错误可能是需求说明的错误、代码中的语法错误或引起系统崩溃的原因（该原因目前未知）。IEEE已经提出了描述软件产品（IEEE 1983）中“错误”的标准术语（见IEEE 标准729）。

缺陷（fault）是人进行软件开发活动中人为出错（称为错误（error））造成的。例如，一个设计者可能误解需求，并得出了与需求分析员和用户的实际需求不符的设计。这个设计缺陷是一种错误的代码，它能导致其他缺陷，如：不正确的代码或用户手册中不正确的描述等。因此，单个错误可能产生多个缺陷，而且任何开发或维护过程中都可能存在缺陷。

故障（或称失效，Failure）是相对于系统指定行为的偏离。它可能会在系统交付前后、测试过程或者在操作和维护的过程中被发现。既然需求文档可能包含错误，因此故障意味着系统即使按说明执行，也可能没有按照需要来执行。

因此，缺陷是系统的内部观点，就像从开发者的角度看待问题那样；而故障是外部观点：即用户所看到的问题。并非每个缺陷都对应相应的故障：例如，如果永远不执行错误的代码或者永远不进入特定的状态，那么缺陷将永远不会使代码产生故障。图1.4给出了故障的起源。

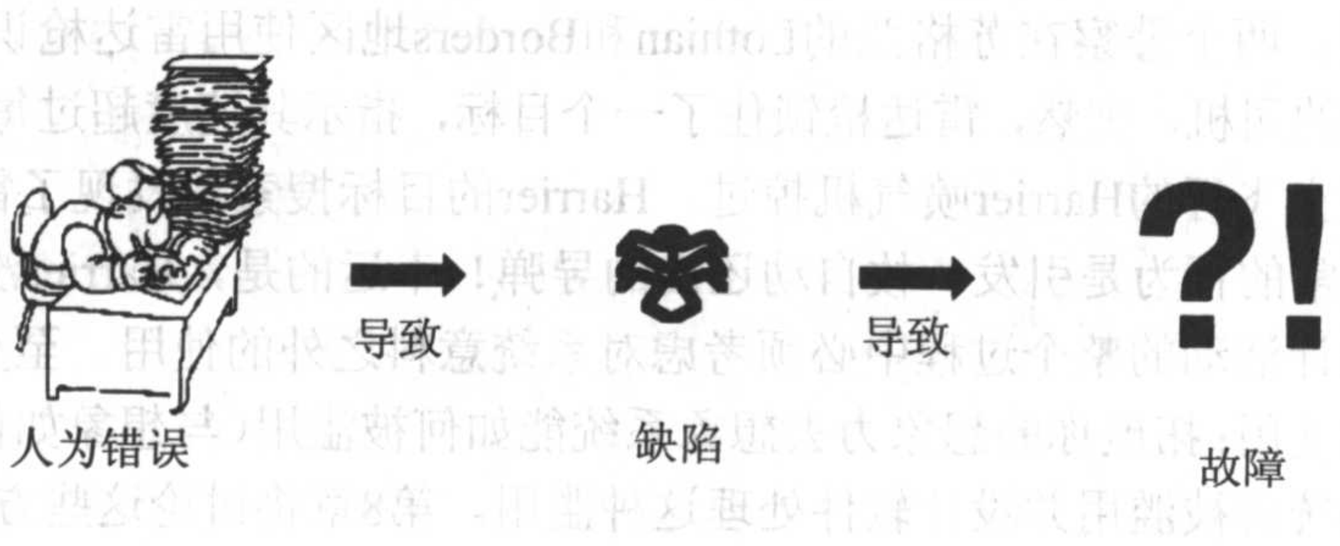


图 1.4　人为错误如何引起故障

20世纪80年代早期，美国国内税收服务处（Internal Revenue Service，IRS）让Sperry公

司建立一套联邦税收表格自动处理系统。根据华盛顿邮报的报道，“系统被证明不适合当前的工作量，花费几乎是预算的两倍，必须立即更换”（Sawyer 1985）。1985年，还需要再加9千万美元来改进Sperry公司最初价值1.03亿的设备。另外，因为出现的问题防碍了IRS按时返还纳税者的税款，IRS被迫偿付4.02千万美元的利息以及2.23千万美元的工资给加班职员。1996年，情况并没有改善。洛杉矶时报在3月29日报道说，仍然没有更新IRS计算机系统的成熟计划，目前只有一个6000页的技术文档。国会议员Jim Lightfoot把这个项目称为“40亿美元的彻底失败，原因是没有充分计划就错误行事”（Vartabedian 1996）。我们将在第2章了解到为什么项目计划对高质量的软件产品来说是至关重要的。

多年以来，公众在日常生活中接受了软件不存在问题的灌输。但里根总统提出的战略防御行动（Strategic Defense Initiative，SDI）增加了公众对开发无缺陷软件系统的困难的认识。流行的报纸和杂志报道（比如Jacky 1985, Parnas 1985, Rensburger 1985）表示出了对计算机科学研究人员的怀疑。并且在20年后的今天，当要求美国国会划拨资金建立一个类似的系统时，许多计算机科学家和软件工程师仍然认为，没有办法编写和测试确保其充分可靠性的软件。

例如，许多软件工程师认为反弹道导弹系统至少需要1千万行代码；有些人估计竟高达1亿行。形成对照的是，支持美国航天飞机的软件只包含300万行代码，包括控制发射和飞行的地面控制计算机；1985年，航天飞机上只有10万行代码（Rensburger 1985）。因此，一个反导弹软件系统将需要数量巨大的代码测试。另外，可靠性约束问题是无法测试的。这一点可以用安全攸关的（Safety-critical）软件的概念来说明。一般地，我们说某些事情是安全攸关（即这些事情的故障会对生命或健康造成威胁），则其可靠性应当至少是10^{-9}。正如我们将在第9章看到的那样，这意味着系统运行10^9小时其失效不能超过一次。要观察这个数量级的可靠性，就不得不使这个系统运行至少10^9小时验证它没有失效，但是10^9小时超过114 000年——作为一个测试周期来说它实在是太长了。

我们也将在第9章看到，如果不能适当地设计软件或进行编码，原本有用的技术可能会致命。例如，当Therac-25（一种射线疗法和X射线机器）发生故障并使几个病人死亡时，医学界变得惊恐万状。软件设计者没有估计到会有不按标准使用几个方向键的情况；结果，当需要低剂量的射线束时，软件却保持高剂量的设置并发出了极为集中的射线束（Leveson and Turner 1993）。

类似地，*Pilot*杂志和Risk Forum（*Pilot* 1996）报道了一些意料之外的软件使用及其严重后果的例子。两个警察在苏格兰的Lothian和Borders地区使用雷达枪识别在Berwickshire旷野超速行驶的司机。突然，雷达枪锁住了一个目标，指示其速度超过每小时300英里。几秒后，一架低空飞行的Harrier喷气机掠过。Harrier的目标搜索器发现了雷达并认为它属于“敌人”，正常的行为是引发一枚自动还击的导弹！幸运的是，Harrier没有配备武器。

在软件设计活动的整个过程中必须考虑对系统意料之外的使用。至少有两种方式处理这些非正常的使用：拓展你的想象力去想象系统能如何被滥用（与想象如何正确使用一样），还可以假定系统将被滥用并设计软件处理这种滥用。第8章将讨论这些方法。

尽管许多供应商致力于设计零缺陷的软件，事实上大多数软件产品都做不到。市场压力促使软件开发人员尽快交付产品，几乎没有时间进行完整的测试。典型情况是，测试小组只能测试那些最有可能用到的功能，或那些最有可能危及用户或激怒用户的功能。正因

如此，许多用户安装代码的第一个版本时都很谨慎，他们知道要等到第二个版本才会解决这些“Bugs”。此外，修正已知的错误有时很困难，以至于重写整个系统要比改动现有代码更为容易。我们将在第11章中探讨软件维护中涉及的问题。

虽然现实中有些软件非常成功并被全面接受，但是在软件质量方面仍然有很大的改进余地。例如，劣质产品可能耗资巨大，一个错误未被检测到的时间越长，纠正它们所需要的费用越高。据估计，项目最初分析阶段纠正错误所需的费用仅仅是在把系统交付给客户后纠正类似错误所需费用的1/10。不幸的是，大多数的错误无法在早期发现。测试和维护阶段发现并改正错误的费用有一半是来自系统生命周期的早期阶段的错误。在第12、13章我们将了解到评价开发活动效率的方法以及对尽可能早地发现错误的过程的改进。

我们提出的简单而有效的技术之一是使用评审和审查。许多学生习惯于自己开发和测试软件。但是他们的测试并没有他们想象的那么有效。例如，Fagan研究了过去他们检测错误的方法。他发现通过测试数据运行程序进行测试时，仅能找出系统开发阶段所留错误的1/5。然而，同事评审（即由同事检查和评论彼此的设计和代码）能发现其余4/5的错误（Fagan 1986）。这样，通过同事评审你的工作，软件质量得到了大幅度的提高。在后面的章节中，我们将学到更多关于如何在每个主要开发步骤之后，利用评审和审查的过程尽可能早地发现并修复错误的知识。并且，我们将在第13章了解到如何改进审查过程。

1.3　什么是好的软件

正如制造商寻找各种方法确保他们生产的产品的质量一样，软件工程师也必须寻找各种途径确保他们的产品具有可接受的质量和用途。因此，好的软件工程必须总是含有开发优质软件的策略。但是在设计策略之前，必须理解优质软件是什么意思。附注1.2说明了不同的观点是如何影响“质量”的含义的。在本节中，我们研究好的软件和差的软件的区别是什么。

附注1.2　质量的观点

Garvin（1984）描述了不同的人是如何认识质量的。他从5种不同的角度对质量进行了描述：

- 超越的观点。质量是可以认识而不能定义的。
- 用户的观点。质量是恰好达到目的。
- 制造的观点。质量是与需求说明的一致。
- 产品的观点。质量是与产品的内在特征相联系的。
- 基于价值的观点。质量取决于顾客愿意支付的金额。

超越的观点很像柏拉图对于理想的描述或亚里士多德形式的概念。换句话说，就像每个真实的桌子是接近理想的桌子一样，我们可以把软件质量认为是我们努力的理想；然而，我们可能永远也不能完全实现它。

超越的观点是无形的，与用户更为具体的观点形成对照。评估产品特性时（如

检测缺陷密度或可靠性），为了理解产品的全面质量，我们采用用户的观点。

制造的观点是在开发过程中以及交付后观察产品质量。特别地，它检查第一次开发的产品是否正确，以避免花费大量重复劳动去改正交付的错误。因此，制造的观点是过程的观点，它提倡与良好的过程相一致。然而，几乎没有什么证据能说明与过程一致而得到的产品会有较少的错误或故障；过程可能确实会得到高质量的产品，但是也可能使生产低质量的产品变成一种习惯。我们将在第12章讨论其中的一些问题。

用户和制造的观点都是从外部来看待产品，但是，产品的观点是从内部观察并评估产品的内部特性。软件度量专家常常提倡这种观点，他们认为好的内部质量指标会得到好的外部质量，如可靠性和可维护性。然而，需要进行更多的研究去验证这些观点并确定质量的哪些方面会影响实际产品的使用。为了把产品的观点和用户的观点联系起来，我们可能不得不开发一些模型。

顾客或销售商常采用用户的质量观点。研究者有时会采用产品的观点，而开发团队使用制造的观点。如果不能明确观点之间的区别，那么混乱和误解将导致错误的决策和质量较差的产品。基于价值的观点把这些不同的质量观点结合起来。通过把质量与顾客期望的付出等同起来，我们能够了解到价格和质量之间的平衡点，而且在它们发生冲突的时候，我们就能控制这些冲突。类似地，购买者把产品的价格与潜在的利益进行比较，认为质量就是金钱。

Kitchenham和Pfleeger（1996）在*IEEE Software*杂志质量问题专刊的引言中探讨了这个问题的答案。他们注意到环境有助于确定答案。文字处理软件中可容忍的错误可能在安全要求较高的或关键性任务系统中是不可接受的。因此，我们必须至少从3方面考虑质量：产品的质量，生产产品的过程的质量以及在使用产品的商业环境中产品的质量。

产品质量

我们可以要求人们把对整体质量起作用的软件特性列出来，但是，很可能从我们询问的每一个人员中得到的答案都有所不同。之所以出现这种差异，是因为特性的重要性取决于谁分析这个软件。如果软件用比较易于学习和使用的方式完成了用户想要的任务，用户就认为软件具有高质量。然而，有时质量和功能是交织在一起的；如果有些软件很难学习和使用，但是它的功能值得进行这些付出，则仍认为它具有较好的质量。

我们试图测定软件质量，这样就可以把一个产品与其他产品进行比较。为此，要识别哪些系统特性对整体质量起作用。这样，测定软件质量时，由用户评估这些外部特性，如错误的数目和类型。例如，可以把错误定义为小错误、主要错误以及灾难性错误，并且我们期望所发生的任何错误都是小错误。

软件必须既由那些设计和编写代码的人员评价，也要由那些维护程序的人员评判。这些实践者倾向于检查产品的内部特性，有时甚至会在产品移交给用户之前进行评判。特别是，实践者常常把错误的数量和类型作为产品质量（或质量低劣）的依据。例如，开发者跟踪在需求说明、设计和代码检测中错误的数量，并把它们作为最终产品的质量指标。

正因如此，我们常常建立模型，把用户的外部观点和软件开发者的内部观点联系起来。

图1.5是一个早期质量模型的例子，它是由McCall和他的同事建立的，说明了外部的质量因素（在图的左边）如何与产品质量标准（在图的右边）相联系。McCall给右边的每个标准分配一种度量，以说明各质量因素受该标准影响的程度（McCall、Richards和Walters 1977）。我们将在第12章介绍几个产品质量模型。

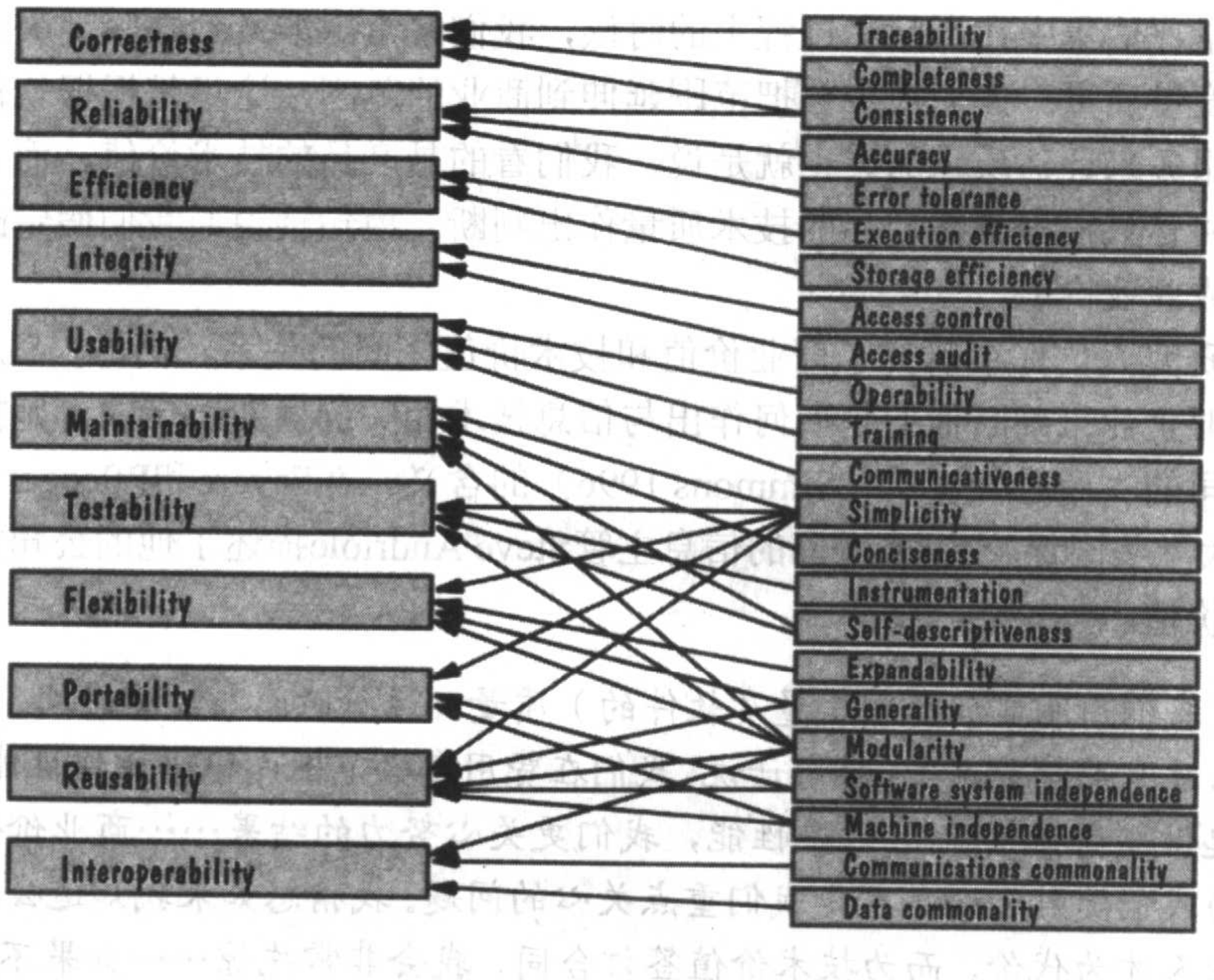


图 1.5　McCall 的质量模型

过程质量

有很多活动会影响最终的产品质量；如果任何一个活动出了差错，整个产品的质量就会受到影响。正因如此，许多软件工程师认为开发和维护过程的质量与产品质量同等重要。对过程建模的一个优点是可以仔细研究它并寻找方法来改进它。例如，我们可以提出如下问题：

- 在何时以及在何地可能发现某种特定类型的错误？
- 如何更早地在开发过程中发现错误？
- 如何建立容错机制以便把因错误导致失效的可能性降到最低？
- 是否有一些可选的活动能够让开发过程更有效以确保质量？

这些问题可以应用到整个开发过程或某个子过程（如配置管理、复用或测试）中；我们将在后面的章节中研究这些过程。

20世纪90年代，软件工程侧重于过程建模和过程改进。受Deming和Juran工作的启发，IBM等公司实现的能力成熟度模型（Capability Maturity Model，CMM）、ISO 9000、软件过程改进及能力确定（Software Process Improvement and Capability dEtermination，SPICE）说明：通过改进软件开发过程，可以改进最终产品的质量。在第2章，我们将看到如何识别

相关的过程活动，及其建模对中间产品及最后产品的影响。在第12章、13章将进一步讨论过程模型并改进框架。

商业环境中的质量

当质量评估集中在产品和过程上的时候，我们常常用涉及到缺陷、失效和时间的数学表达式来度量质量。人们很少会把范围延伸到商业的范围，这里是根据软件供应商所提供的产品和服务来评估质量的。也就是说，我们看的是产品的技术价值，而不是更广泛的商业价值，并且只根据最终产品的技术质量作出判断。换句话说，我们假定改进技术质量会自动转化成商业价值。

一些研究人员仔细研究了商业价值和技术价值之间的关系。例如，Simmons调查了很多澳大利亚企业来判断他们是如何作出与信息技术相关的商业决策的。她提出了一种构架来理解公司的“商业价值”（Simmons 1996）的含义。在Favaro和Pfleeger（1997）的报告中，一个大型美国保险公司Cigna的信息主管Steve Andriole描述了他的公司如何区分商业价值与技术价值：

> 我们用明显的标准度量（软件的）质量：运行时间与故障时间、维护费用、与修改有关的费用等。换句话说，我们在费用参数范围内根据操作性能管理开发。比起供应商提供费用-效率性能，我们更关心努力的结果……商业价值与技术价值与我们更贴近……也是我们重点关心的问题。我猜想如果我知道公司之间将以商业人才为代价，而为技术价值签订合同，我会非常吃惊……如果不存在（所期望的）明确的商业价值（可定量描述：可处理的要求数量等），那么我们不能启动这个系统项目。我们非常认真地经过“有目的性的”项目需求阶段，那时我们会问：“我们为什么会需要这个系统？”以及“我们为什么关心它？”

为了定量地或可理解地把技术价值和商业价值联系起来，人们已经进行了很多尝试。比如说，Humphrey、Snyder和Willis（1991）注意到，根据（将要在第12章中讨论的）CMM“成熟”度改进开发过程，Hughes Aircraft生产率提高了4倍，节省了数百万美元。类似地，Dion （1993）报道说Raytheon的生产率增加了2倍，并且过程改进中每1美元的投资都有7.7美元的回报。在俄克拉荷马州的Tikner空军基地，工作人员注意到其生产率提高了6.35倍（Lipke和Butler 1992）。

然而，Brodman和Johnson（1995）仔细研究了过程改进的商业价值。他们调查了33家从事某种过程改进活动的公司，并且仔细观察了若干关键事项。此外，Brodman和Johnson还询问了这些公司是如何定义投资回报（Return on Investment，ROI）的。投资回报这一概念在商业领域中有清楚的定义。他们注意到教科书中投资回报的定义来自金融界，根据为了其他目标放弃了什么来描述投资。也就是说，“投资必须不仅仅是返回原始资本，而且返回必须足够多，至少等于在其他地方这些资金所能挣来的利润，再加上风险金”（Putnam和Myers 1992）。通常情况下，商业社会使用下面三种模型之一评估投资回报：偿还模型，记账的回报率模型和折扣现金流模型。

但是，Brodman和Johnson（1995）发现，美国政府和美国工业界用来解释ROI的方式

彼此相差很大，并且都不同于标准商业学校的方式。政府根据美元来看待ROI，注重减少操作费用。期望节省美元并计算使用新技术的费用。政府投资也是用美元来表示，如引入新技术的费用或启动过程改进等的费用。

另一方面，工业界根据工作量而不是费用或美元来看待投资。也就是说，公司对节省时间和使用更少的人感兴趣，并且他们对投资回报的定义反映在减少工作量上。在对公司的调查中，投资回报包括如下术语：

- 培训
- 进度
- 风险
- 质量
- 生产率
- 过程
- 顾客
- 费用
- 商务

上述定义中的费用包括会议费用预算、提高费用性能并保持在预算范围之内，而不是靠减少操作费用或使项目或组织合理化（Streamlining）。图1.6总结了许多组织使用ROI定义的投资项目的频率。例如，那些被调查的组织中，大约有5%把质量小组的工作包括在ROI的工作量计算中，大约有35%在考虑投资规模时包含软件费用。

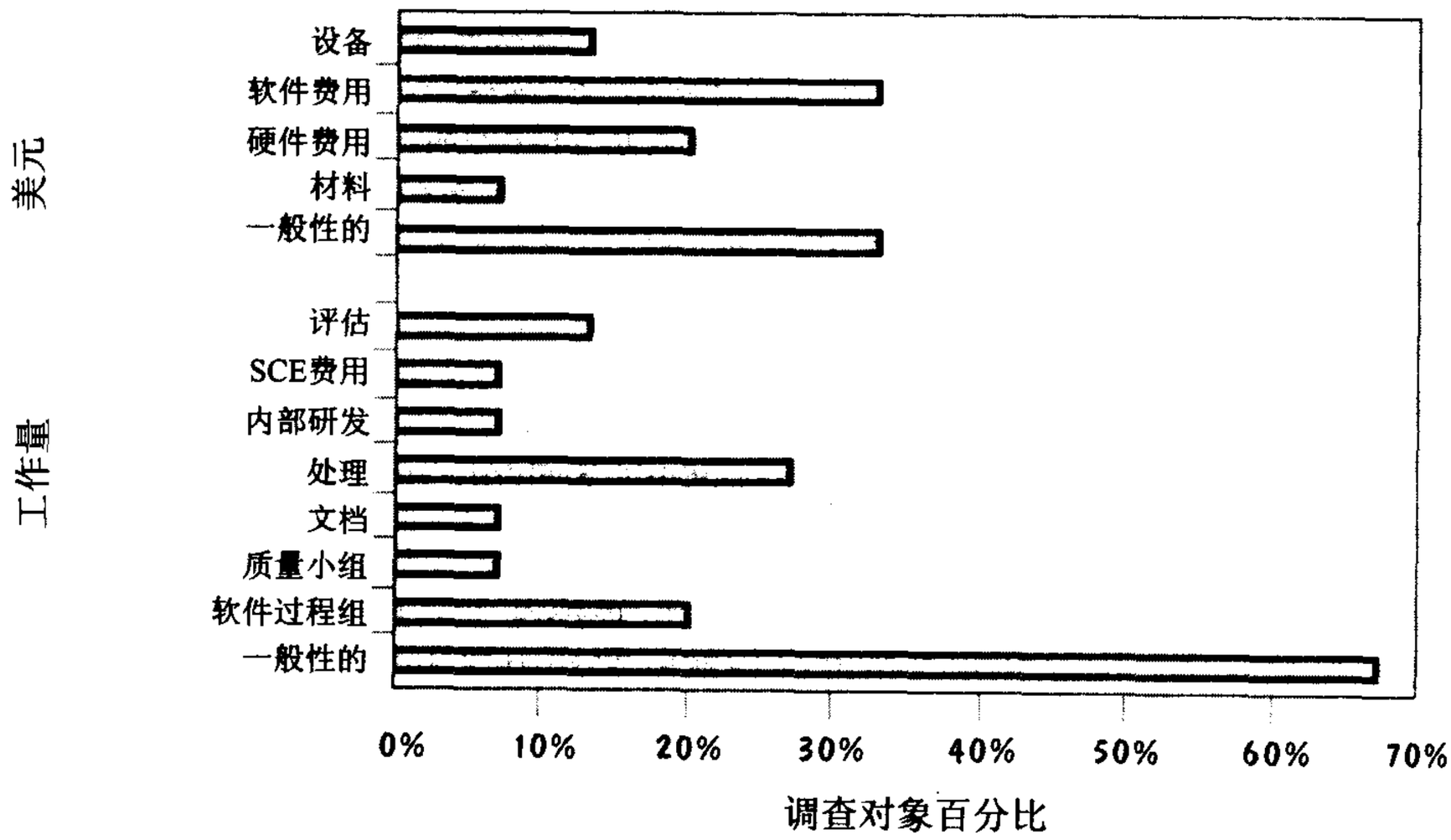


图 1.6　工业界对投资回报的定义中包含的术语

观点的不同带来了混乱，因为它意味着组织之间的ROI计算是不可比较的。但是，这些不同的观点也有好处。因缩短的进度、更高的质量、提高的生产率等因素节省的美元都返回给了政府而不是承包商。另一方面，承包商常常不仅追求更大的利润，也寻找竞争的

优势以及提高劳动能力；这样，承包商的投资回报是基于工作量而不是基于费用的。特别是，更精确的费用和进度估计意味着顾客的满意度和再合作的可能性。并且缩短投入市场的时间和提高产品质量也被认为是提供了商业价值。

尽管不同的ROI计算方法能够由每个组织调整，但令人担心的是：软件技术的ROI与金融的ROI是不同的。在很多时候，程序的成功必须报告给高层管理机构，很多管理与软件无关而与公司的主要业务（如通信或银行业等）相关。许多混淆是因为用相同的术语表示有着巨大差异的事物而产生的。这样，成功的标准不但要对软件项目和过程有意义，而且对支持更一般的业务实践也要有意义。我们将在第12章中更仔细地讨论这个问题，并使用几种常用的商业价值度量来选择技术。

1.4 谁来做软件工程

软件开发的一个关键因素是顾客与开发人员之间的交流；如果交流失败，那么系统也将失败。我们必须在构造帮助顾客解决问题的系统之前，理解顾客想要什么以及他们需要什么。为了做到这一点，让我们把注意力转移到在软件开发过程中涉及到的人员。

软件开发中工作人员的数量取决于项目的规模和难易程度。然而，不论涉及多少人，都可以区分他们在整个项目生命周期中的角色。这样，对于大的项目，一个人或一个小组必须被分配到所能标识的每一个角色；在小的项目中，一个人或一个小组可以同时担任若干角色。

通常情况下，参与项目的人员分为三类：顾客、用户和开发人员。顾客是支付开发软件系统费用的公司、组织或个人。开发人员是为顾客构造软件系统的公司、组织或个人。这种分类包括了所有协调并指导程序员和测试人员的管理者。用户是实际使用系统的人：要么坐在终端前，要么提交数据，要么阅读输出。虽然，对某些项目来说顾客、用户、开发人员是同一个人或同一组人，但通常是不同的人员集合。图1.7给出了这三类参与人员之间的基本关系。

顾客控制着资金，常常进行合同谈判并在验收报告上签字。然而，有时候顾客并不是用户。例如，假设Wittenberg Water Works与Gentle Systems, Inc.签定了一份合同，为公司建立一套计算机记账系统。Wittenberg的总裁可以向Gentle Systems, Inc.的代表精确地描述他们需要什么，并且她将要签定这份合同。然而，总裁并不直接使用这套记账系统；用户将是会计或记账的职员。因此开发人员准确理解顾客和用户想要什么以及需要什么是非常重要的。

另一方面，假设Wittenberg Water Works是一家大的公司，它有自己的计算机系统开发部门。这个部门可能觉得它需要一个自动化的工具来记录其项目的费用和进度。通过自己建立这个工具，这个部门既是用户、顾客同时也是开发人员。

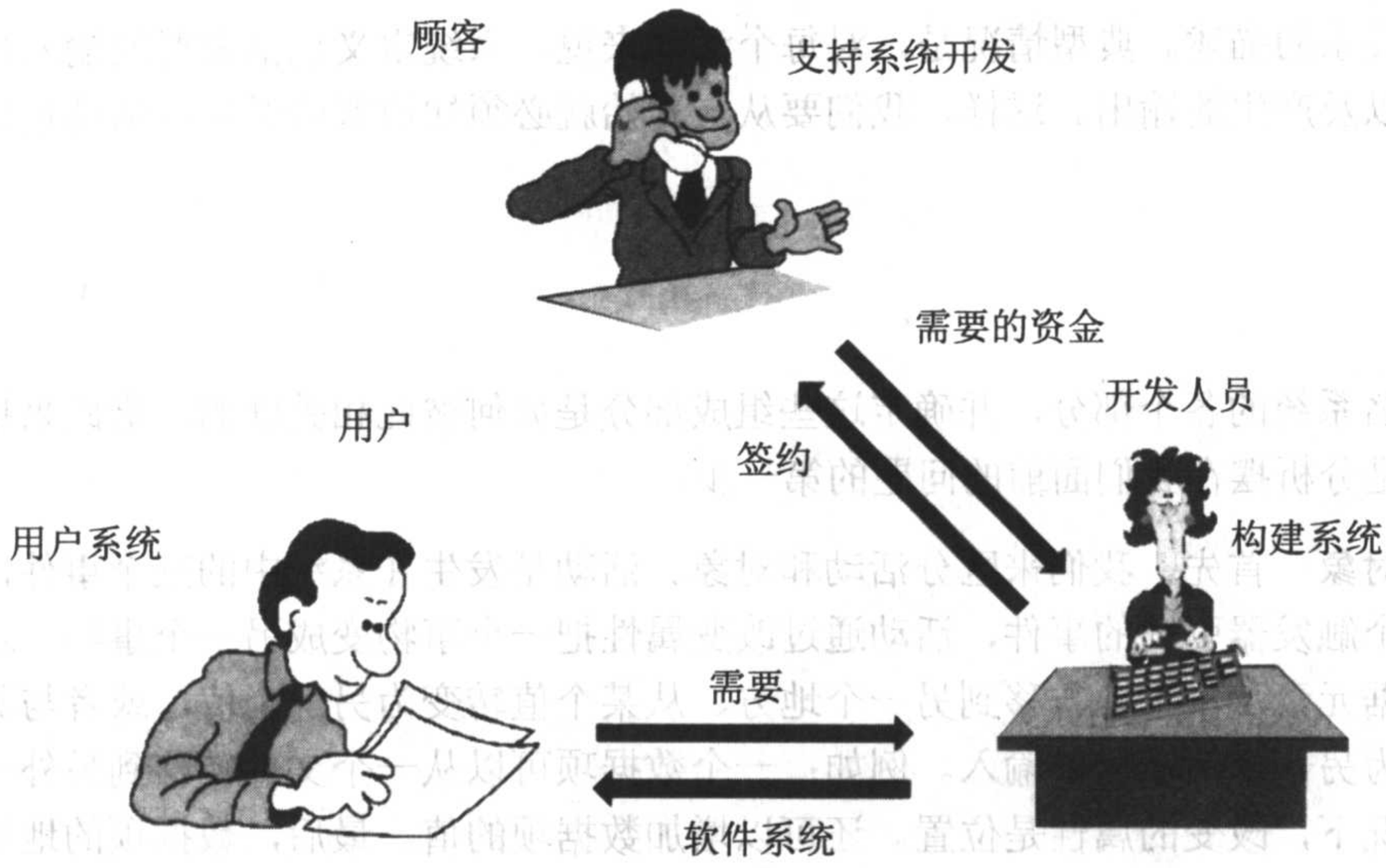


图 1.7　软件开发中的参与者

近几年，顾客、用户和开发人员之间的细微差别变得越来越复杂。顾客和用户涉及到开发过程的很多方面。顾客可以决定购买通用商业软件(Commercial Off-The-Shelf, COTS)，并合成到开发人员提供和支持的最终产品中。在这种情况下，顾客涉及到系统体系结构的决策，对开发过程有很多约束。类似地，开发人员可以另外选择称为转包商的开发人员，转包商开发子系统并把它交付给开发人员，使之包含在最终的产品中。转包商可以与主要的承包商并肩工作，也可以在一个不同的地方和主要开发人员协调工作，并在开发过程后期交付子系统。子系统可能是个全承包的系统，其代码是集成到一起的（不需要集成其他代码），或者它可能需要一个独立的集成过程连接主要系统和子系统。

因此，软件工程中的“系统”观点不仅对问题分析或解决方案的合成至关重要，对组织开发过程以及将适当的角色分配给参与者来说也是非常重要的。下一节我们将要了解好的软件工程实践中系统方法的作用。

1.5　系统方法

我们开发的项目并不存在于真空中。通常，我们要把硬件和软件放在一起，还必须与用户、其他软件任务、其他硬件、现有数据库（即仔细定义的数据集合和数据关系）、甚至其他的计算机系统进行交互。因此，了解项目的边界（即项目中包括什么不包括什么），从而为项目提供背景是非常重要的。例如：假设主管让你写一段程序为办公室的人员打印工资单。你必须知道你的程序是否只是简单地从另一个系统中读入工作时间并且打印结果，还是同时必须计算工资信息。同样地，你必须知道程序是否需要计算税率、补贴金以及津贴，或者这些项是否要随每份工资单提供。你真正要问的问题是：项目从哪里开始，到哪里结束？同样的问题可以应用于任何系统。一个系统是对象和活动的集合，另外还有对象

和活动之间关系的描述。典型情况是，对每个活动来说，系统定义包括需要的输入列表、采取的动作以及产生的输出。这样，我们要从一开始就必须知道哪些对象或活动包括在系统中。

系统元素

我们命名系统的各个部分，并确定这些组成部分是如何彼此相关联的，借此来描述一个系统。这是分析摆在我们面前的问题的第一步。

活动和对象 首先，我们来区分活动和对象。活动是发生在系统中的某个事件，通常描述成由某个触发器引发的事件，活动通过改变属性把一个事物变成另一个事物。这种转变意味着数据元素从一个地方移到另一个地方、从某个值转变为另一个值，或者与其他的数据相结合为另一个活动提供输入。例如：一个数据项可以从一个文件转移到另外一个文件。这种情况下，改变的属性是位置。还可以增加数据项的值。最后，数据项的地址与若干其他数据项的地址一起包含在参数列表中，从而能够通过调用其他程序处理所有数据。

活动中涉及的元素称为对象或实体。通常情况下，这些对象通过某种方式彼此联系。例如，对象能够排列在表格或矩阵中。对象常常组成记录，而记录以预先规定的格式排列。例如，雇员历史记录中的每个雇员都包含如下对象（也称字段）：

名字	邮政编码
姓	每小时的工资
街道地址	每小时的津贴
城市	应计休假
州	应计病假

记录中不但定义了每个字段，而且定义了每个字段的大小以及字段和其他命名字段之间的关系。这样，记录描述说明了每个字段的数据类型、记录中的开始位置和字段的长度。依次地，因为每个雇员都有一条记录，所有的记录就组成了文件，并且要指明文件特性（比如最大的记录数）。

有时，定义的对象稍有不同。与在一个大的记录中把每个项看成字段不同，对象被看成是独立的。对象的描述包括每个对象的属性列表，以及所有使用对象或影响对象的活动列表。例如：考虑“多边形”对象。对象描述可以说这个对象具有如下属性：边数以及每条边的长度。动作可包括计算面积和周长。甚至还可以有一个属性称为“多边形的类型”，这样，可以确定每个“多边形”的实例，比如是“菱形”或“长方形”等。类型本身也可以有对象描述；比如“长方形”可以由“正方形”和“非正方形”组成。我们将在第4章研究需求分析时讨论这些概念，并在第6章讨论面向对象的开发时作深入的探讨。

关系和系统边界 一旦定义了实体和活动，就要把实体和它们的活动进行匹配。要很清楚而且很详细地定义实体和活动之间的关系。实体的定义包括实体起源于何处的描述：有些项存在于现有的文件中；有些项在活动的过程中产生。实体的目标也是非常重要的。有些项仅仅被一个活动所使用，但是，可能会指定某些项作为其他系统的输入。也就是说，

在当前系统范围之外的活动会使用系统的某些项。因此，我们可以把系统看成是有边界或界限的。有些项穿过边界进入系统，而另一些是系统的产品并为其他系统所使用。

使用这些概念，我们能够把系统定义成一组事物的集合：实体的集合、活动的集合、实体和活动之间关系的描述以及系统边界的定义。系统的定义不但可以应用到计算机系统中，而且可以应用到任何系统中，在该系统中对象以某种方式与其他对象发生作用。

系统举例　为了解系统定义如何工作，以身体吸进氧气排出二氧化碳和水的器官——呼吸系统为例。我们可以很容易地定义它的边界：如果给身体的一个特殊器官命名，就能说出它是不是呼吸系统的一部分。氧气和二氧化碳分子都是实体或对象，它们按照可以明确定义的方式进出呼吸系统。我们也可以根据实体间的相互作用描述系统中的活动。如果必要的话，可以通过给出什么进入或离开了系统来描述这个系统；也可以用一个表格来描述其中涉及到的所有实体和活动。图1.8是呼吸系统。请注意每个活动都涉及到实体，并且可以通过描述哪些实体是输入、它们如何被处理以及输出的结果来定义活动。

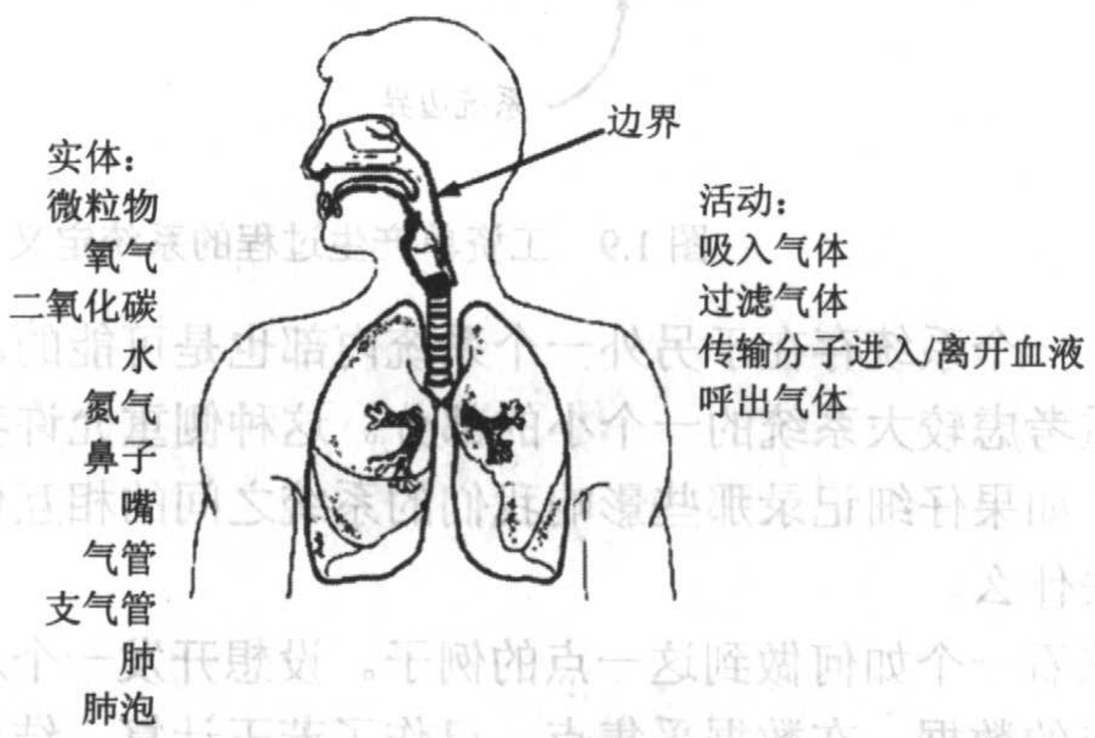


图 1.8　呼吸系统

我们还必须清晰地描述计算机系统，和未来的用户一起定义系统的边界：在什么地方开始和结束工作？另外，需要知道什么在系统的边界上，从而可以确定输入的开始和输出的目的地。例如，在打印工资单的系统中，支付信息可能来自公司的计算机。系统输出可能是发送到邮局的工资单的集合，邮局负责把它们送到正确的接收者。在图1.9给出的系统中，我们可以看到边界并且了解实体、活动和它们之间的关系。

相互关联的系统

边界的概念之所以重要，是因为很少有系统是与其他系统独立的。例如，呼吸系统必须与消化系统、循环系统、神经系统以及其他系统相互作用。呼吸系统没有神经系统就不能发挥作用；循环系统没有呼吸系统也不能正常工作。这种相互依赖是很复杂的（事实上，因为我们不能协调我们的生态系统，已经引起和加剧了许多环境问题）。然而，一旦描述了系统的边界，分清什么在系统内部、什么不在以及什么穿越边界就比较容易了。