

## 软件和软件工程

## 要点浏览

**概念：**计算机软件是由专业人员开发并长期维护的软件产品。完整的软件产品包括：可以在各种不同规模及体系结构的计算机上运行的程序，程序运行过程中产生的各种结果，以及各种描述信息，这些信息可以以硬拷贝或是各种电子媒介形式存在。软件工程包括过程、各种方法（实践）以及各类工具，以协助专业人员构建高质量的计算机软件。

**人员及责任：**软件工程师开发软件并提供技术支持，产业界中几乎每个人都直接或间接地使用软件。

**重要性：**软件之所以重要是因为它在我们的生活中无所不在，并且日渐深入到商

业、文化和日常生活各个方面。软件工程之所以重要是因为软件工程使我们可以高效、高质量地构建复杂系统。

**步骤：**开发计算机软件就像开发任何成功的产品一样，需采用灵活、可适应的软件开发过程，完成可满足使用者需求的高质量软件产品。这就是软件工程化方法。

**工作产品：**从软件工程师的角度来看，最终产品是计算机软件，包括程序、内容（数据）和各种工作产品；而从用户的角度来看，最终产品是可以改善生活和工作质量的最终信息。

**质量保证措施：**阅读本书的后面部分，选择切合你所构建的软件特点的思想，并在实际工作中加以应用。

## 关键概念

应用领域

软件特点

框架活动

遗留软件

实践

原则

软件工程

软件神话

软件过程

普适性活动

Web应用

我谈话的对象具有大型软件公司高级主管的典型特征——45岁左右、鬓角微白、身材匀称而健壮，说话时眼睛似乎能够洞穿倾听者的内心。但是，他说的话却让我感到震惊：“软件已经死了。”

我惊奇地眨眨眼，随后笑道：“你在开玩笑，是吗？软件已成为世界进步的驱动力，也给你们的公司带来了巨大的利益。软件没有死！它还活着，并且在茁壮成长。”

他用力地摇了摇头说：“不，软件已经死了……至少就像我们曾经知道的那样。”

我倾身向前：“你继续讲！”

他敲着桌子以示强调，一边说道：“根据学院派观点——购买软件、拥有软件，并将使用和管理软件作为一项工作，这种模式已经走到了尽头。今天，随着Web 2.0和普适计算的发展，我们即将看到全然不同的新一代软件产品。软件通过网络交付使用，看起来就像驻留在每个用户的计算设备上运行一样……但实际上，软件可能运行在遥远的服务器上。”

我不得不同意：“因此，你的生活将变得更加简单。你们不必再为成千上万个用户交叉访问同一个应用系统的5个不同版本而担心。”

他笑道：“千真万确。我们仅在服务器上运行当前最新版本的软件。每当对软件做了更新或改正，我们为每个用户提供更新的软件功能和内容，每个用户都可以立刻拥有最新的版本！”

我做了个鬼脸，说道：“但如果你们犯了错误，每个用户也立刻受到影响。”

他轻声笑道：“是的，这就是我们加倍努力做好软件工程的原因。问题是我们必须‘快速’，因为在每个应用领域，市场都在加速发展变化。”

我坐直身体，把双手放在脑后说：“你知道人们怎么说……快速、正确或者廉价，你只能选择其中两个！”

他边起身边说道：“我会选择快速和正确。”

我也站起身来：“那么你确实需要软件工程。”

“我知道，”他准备离开，边走边说：“问题是我们必须意识到还需要新一代的高科技专家，并且确实如此。”

软件真的死了吗？如果是那样，你就不必阅读这本书了！



“创新观念  
和科技发现

是经济增长的推进  
器。”——《华尔  
街日报》

计算机软件仍然是世界舞台上最为重要的科技领域，并且是“意外效应法则”的一个最好的体现。50年前，没有人曾预料到软件科学会成为今天商业、科学和工程所必需的技术；软件促进了新科技的创新（例如基因工程和纳米技术）、现代科技的发展（例如通信），以及传统技术的根本转变（例如印刷业）；软件技术已经成为个人电脑革命的推动力量；消费者可以

很容易地在附近的商店购买到包装好的软件产品，软件还将由产品逐渐演化为服务，软件公司随需应变，通过Web浏览器发布即时更新功能；软件公司可以比几乎任何传统工业时代的公司更大、更有影响力；在大量应用软件的驱动下，互联网将迅速发展，并将对人们生活的诸多方面——从图书馆搜索、消费购物、政治演说到年轻人（或者不那么年轻的人）的约会习惯——引起革命性的变化。

没有人曾想到软件可嵌入到各种系统中：交通运输、医疗、通信、军事、工业、娱乐以及办公设备等不胜枚举。如果笃信“意外效应法则”的话，那么还有很多结果和影响是我们尚未预料到的。

没有人曾想到，随着时间的推移，将有数百万的电脑程序需要进行纠错、适应性调整和优化，这些维护工作将耗费比开发新软件更多的人力、物力。

随着软件重要性的日渐凸现，软件业界一直试图开发新的技术，使得高质量计算机程序的开发和维护更容易、更快捷、成本更低廉。某些技术主要针对特定应用领域（例如，网站设计和实现）；有些着眼于技术领域（例如，面向对象系统，面向方面的程序设计）；还有一些覆盖面很宽（例如，像Linux这样的操作系统）。然而，我们尚未开发出一种软件技术可以实现上述所有需求，而且未来能够产生这种技术的可能性很小。人们也尚未将其工作、享受、安全、娱乐、决策以及全部生活都完全依赖于计算机软件。这或许是正确的选择。

本书为需要构建正确软件的计算机软件工程师提供了一个框架。该框架包括过程、一系列方法以及我们称为软件工程的工具。

## 1.1 软件的本质



软件既是产品  
也是交付产品  
的载体。

现在的软件技术具有产品和产品交付载体的双重作用。作为一个产品，它显示了由计算机硬件体现的计算能力，更广泛地说，显示的是由一个可被本地硬件设备访问的计算机网络体现的计算潜力。无论是驻留在移动电话还是在大型计算机中，软件都扮演着信息转换的角色：产生、管理、获取、修改、显示或者传输各种不同的信息，简单如几个比特的传递或复杂如从多个独立的数据源获取的多媒体演示。而作为产品生产的载体，软件提供了计算机控制（操作系统）、信息通信（网络）以及应用程序开发和控制（软件工具和环境）的基础平台。

“软件是播撒梦想和收获噩梦的地方，是一片恶魔和神仙竞争的抽象而神秘的沼泽，是一个狼人和银弹共存的矛盾世界。”——Brad J.Cox

软件提供了我们这个时代最重要的产品——信息。它会转换个人数据（例如个人财务交易），使信息在一定范围内发挥更大的作用；它通过管理商业信息提升竞争力；它为世界范围的信息网络提供通路（比如因特网），并对各类格式的信息提供不同的查询方式。

在最近半个世纪里，计算机软件的作用发生了很大的变化。硬件性能的极大提高、计算机结构的巨大变化、内存和存储容量的大幅度增加、还有种类繁多的输入和输出方法都促使计算机系统的结构变得更加复杂，功能更加强大。如果系统开发成功，复杂的结构和功能可以产生惊人的效果，但是同时复杂性也给系统开发人员带来巨大的挑战。

现在，一个庞大的软件产业已经成为了工业经济中的主导因素。早期的独立程序员也已经被专业的软件开发团队所代替，团队中的不同专业技术人员可分别关注复杂的应用系统中某一个技术部分。然而同过去独立程序员一样，开发现代计算机系统时，软件开发人员依然面临同样的问题<sup>①</sup>：

- 为什么软件需要如此长的开发时间？
- 为什么开发成本居高不下？
- 为什么在将软件交付顾客使用之前，我们无法找到所有的错误？
- 为什么维护已有的程序要花费高昂的时间和人力代价？
- 为什么软件开发和维护的过程仍旧难以度量？

种种问题显示了业界对软件以及软件开发方式的关注，这种关注促使了业界对软件工程实践方法的采纳。

### 1.1.1 定义软件

今天，绝大多数专业人员和许多普通人认为他们已经大概了解软件。真的是这样吗？

来自教科书的关于软件的定义也许是：

软件是：（1）指令的集合（计算机程序），通过执行这些指令可以满足预期的特征、功能和性能需求；（2）数据结构，使得程序可以合理利用信息；（3）软件描述信息，它以硬拷贝和虚拟形式存在，用来描述程序操作和使用。

如何定义软件？

当然，还有更完整的解释。

但是一个非常形式化的定义可能并不能显著改善其可理解性，为了更好地理解“软件”的含义，有必要将软件和其他人工产品的特点加以区分。软件是逻辑的而非物理的系统元素。因此，软件和硬件具有完全不同的特性：

#### 1. 软件是设计开发的，而不是传统意义上生产制造的

虽然软件开发和硬件制造存在某些相似点，但二者根本不同：两者均可通过优秀的设计获得高品质产品，然而硬件在制造阶段可能会引入质量问题，这在软件中并不存在（或者易于纠正）；二者都依赖人，但是人员和工作成果之间的对应关系是完全不同的（见第24章）；它们都需要构建产品，但是构建方法不同。软件产品成本主要在于开发设计，因此不能像管理制造项目那样管理软件开发项目。

**KEY POINT**

软件是设计开发的，而非制造加工。

**KEY POINT**

软件不会磨损，但会退化。

① 在一本优秀的关于软件商业的论文集中，Tom DeMarco [DeM95]提出了相反的看法。他认为：“我们更应该总结使得当今的软件开发费用低廉的成功经验，而不是不停地质问为何软件开发成本高昂。这会有助于我们继续保持软件产业的杰出成就。”

## 2. 软件不会“磨损”



若希望降低软件退化, 需要改进软件的设计(第8~13章)。



软件工程方法的目的降低图1-2中向上变变的幅度及实际失效曲线的斜率。

图1-1描述了硬件的失效率, 该失效率是时间的函数。这个被称为“浴缸曲线”的关系图显示: 硬件在早期具有相对较高的失效率(这种失效通常来自设计或生产缺陷); 缺陷被逐个纠正之后, 失效率随之降低并在一段时间内保持平稳(理想情况下很低)。然而, 随着时间推移, 因为灰尘、震动、不当使用、温度超限以及其他环境问题所造成的硬件组件损耗累积的效果, 使得失效率再次提高。简而言之, 硬件开始“磨损”了。

而软件不会受引起硬件磨损的环境问题的影响。因此, 从理论上来说, 软件的失效率曲线应该呈现为图1-2的“理想曲线”。未知的缺陷将在程序的生命周期的前期造成高失效率。然而随着错误被纠正, 曲线将如图中所示, 趋于平缓。“理想曲线”只是软件实际失效模型的粗略简化。曲线的含义很明显——软件不会磨损, 但是软件退化的确存在。

这个似乎矛盾的现象用图1-2所示的“实际曲线”可以很好地解释。在完整的生存周期里<sup>①</sup>, 软件将会面临变更, 每次变更都可能引入新的错误, 使得失效率像“实际曲线”(图1-2)那样陡然上升。在曲线回到最初的稳定失效率状态前, 新的变更会引起曲线又一次上升。就这样, 最小的失效率点沿类似于斜线逐渐上升, 可以说, 不断的变更是软件退化的根本原因。

磨损的另一面同样说明了软硬件的不同。磨损的硬件部件可以用备用部件替换, 而软件却不存在备用部件。每个软件的缺陷都暗示了设计的缺陷或者在从设计转化到机器可执行代码的过程中产生的错误。因此, 软件维护要应对变更请求, 比硬件维护更为复杂。

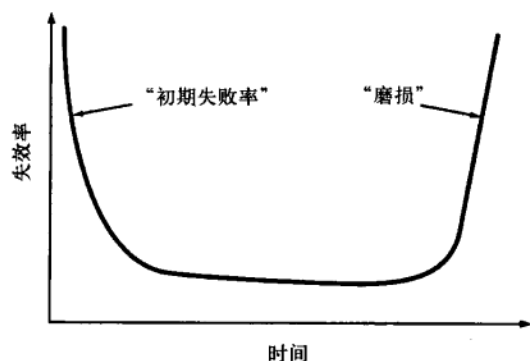


图1-1 硬件失效曲线图

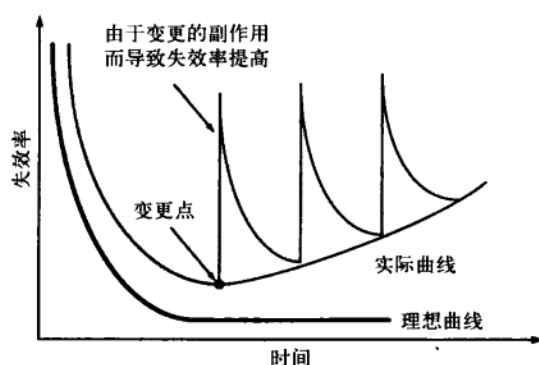


图1-2 软件失效曲线图

3. 虽然整个工业向着基于构件的构造模式发展, 然而大多数软件仍是根据实际的顾客需求定制的

工程学科的发展将产生一系列标准的设计器件。标准螺丝钉及可订购的集成电路只是机械工程师和电子工程师在设计新系统时所使用的上千种标准器件中的两种。可复用构件的使用可以使得工程师专心于设计中真正创新的部分, 也就是, 设计中真正新的内容。在硬件设计中, 构件复用是工程进程中通用的方法。而在软件设计中, 大规模的复用还刚刚开始尝试。

<sup>①</sup> 事实上, 从软件开发开始, 在第一个版本发布之前的很长时间, 会有许多利益相关者提出变更要求。

软件的构件应该设计并实现成可在不同程序中复用的组件。现代的可复用构件封装了数据和对数据的处理,使得软件工程师能够利用可复用的构件构造新的应用程序<sup>①</sup>。例如,现在的交互式用户界面就使用可复用构件构造图形窗口、下拉菜单和各种交互机制,构造用户界面所需要的数据结构和处理细节被封装在用于用户界面设计的可重用构件库中。

### 1.1.2 软件应用领域

#### WebRef

目前最大的共享软件/免费软件库之一: shareware.cnet.com.

今天,计算机软件可分为七个大类,软件工程师正面临持续的挑战。

**系统软件**——系统软件是一整套服务于其他程序的程序。某些系统软件(例如:编译器、编辑器、文件管理软件)处理复杂但确定的<sup>②</sup>信息结构。另一些系统应用程序(例如:操作系统构件、驱动程序、网络软件、远程通信处理器)主要处理的是不确定的数据。无论何种情况,系统软件多具有以下特点:和计算机硬件大量交互;多用户大量使用;需要调度、资源共享和复杂进程管理的同步操作;复杂的数据结构以及多种外部接口。

**应用软件**——解决特定业务需要的独立应用程序。这类应用软件处理商务或技术数据,以协助业务操作和管理或技术决策。除了传统数据处理的应用程序,应用软件也被用于业务功能的实时控制(例如:销售点的交易处理,实时制造过程控制)。

**工程/科学软件**——这类软件通常带着“数值计算”算法的特征,工程和科学软件涵盖了广泛的应用领域,从天文学到火山学,从自动应力分析到航天飞机轨道动力学,从分子生物学到自动制造业。不过,当今科学工程领域的应用软件已经不仅仅局限于传统的数值算法。计算机辅助设计、系统仿真和其他的交互性应用程序已经呈现出实时性甚至具有系统软件的特性。

**嵌入式软件**——嵌入式软件存在于某个产品或者系统中,可实现和控制面向最终使用者和系统本身的特性和功能。嵌入式软件可以执行有限但难于实现的功能(例如:微波炉的按键控制)或者提供重要的功能和控制能力(例如:汽车中的燃油控制、仪表板显示、刹车系统等汽车电子功能)。

**产品线软件**——产品为多个不同用户的使用提供特定功能。产品线软件关注有限的特定的专业市场(例如,库存控制产品)或者大众消费品市场(例如:文字处理、电子制表软件、电脑绘图、多媒体、娱乐、数据库管理、个人及公司财务应用)。

**Web应用软件**——叫做“Web应用”(WebApp),是一类以网络为中心的软件,其概念涵盖了宽泛的应用程序产品。最简单可以是一组超文本链接文件,仅仅用文本和有限的图形表达信息。然而,随着Web 2.0的出现,网络应用正在发展为复杂的计算环境,不仅为最终用户提供独立的特性、计算功能和内容信息,还和企业数据库和商务应用程序相结合。

“没有任何一台计算机具备基本常识。”——Marvin Minsky

**人工智能软件**——人工智能软件利用非数值算法解决计算和直接分析无法解决的复杂问题。这个领域的应用程序包括机器人、专家系统、模式识别(图像和语音)、人工神经网络、定理证明和博弈等。

全世界成百万的软件工程师在为以上各类软件项目努力地工作着。有时是建立一个新的系统;而有时只是对现有应用程序的纠错、适应性调整和升级。一个年轻的软件工程师所经手的项目比他自己的年龄还大是常有的事。对于我们上述讨论的各类软件,上一代的软件工程师都已经留下了遗留系统。我们希望这代工程师留下的遗留系统可以减轻未来工程师的负担。然而,新的挑战

① 第10章讨论了基于构件的软件工程。

② 软件的确定性是指系统的输入、处理和输出的顺序及时间是可以预测的;软件的不确定性是指系统的输入、处理和输出的顺序和时间是无法提前预测的。

(参见第31章)也已经逐渐显现出来:

**开放计算**——无线网络的快速发展也许将很快促成真正的普适计算、分布式计算的实现。软件工程师所面临的挑战将是开发系统和应用软件,以使得移动设备、个人电脑和企业应用可以通过大量的网络设施进行通信。

“你通常  
无法预  
测未来,但是  
可以时刻为未  
来做好准备。”  
——无名氏

**网络资源**——万维网已经快速发展为一个计算引擎和内容提供平台。软件工程师新的任务是构建一个简单(例如:个人理财规划)而智能的应用程序,为全世界的最终用户市场提供服务。

**开源软件**——开源软件就是将系统应用程序(例如:操作系统、数据库、开发环境)代码开放,使得很多人能够为软件开发做贡献,这种方式正在逐步成为一种趋势。软件工程师面临的挑战是开发可以自我描述的代码,而更重要的是,开发某种技术,以便于用户和开发人员都能够了解已经发生的改动,并且知道这些改动如何在软件中体现出来。

所有这些新的挑战毫无疑问将遵守“意外效应法则”,会产生现在无法预测的结果(对商务人员,软件工程师和最终用户)。然而,软件工程师可以做一些准备工作,采用一个足够灵活应变的过程,以适应在未来十年中必将发生的技术和业务的种种巨大变化。

### 1.1.3 遗留软件

成千上万的计算机程序都可以归于在前一小节中讨论的7大类应用领域。其中一些是当今最先进的软件——最近才对个人、产业界和政府发布。但是另外一些软件则年代较久,甚至过于久远。

这些旧的程序——通常称为遗留软件(legacy software)——从20世纪60年代起,就成为持续关注的焦点。Dayani-Fard和他的同事[Day99]这样描述遗留软件:

遗留软件系统……在几十年前开发,它们不断被修改以满足商业需要和计算平台的变化。这类系统的繁衍使得大型机构十分头痛,因为它们的维护代价高昂且系统演化风险较高。

Liu和他的同事[Liu98]进一步扩展了这个描述,指出:“许多遗留软件系统仍然支持核心的商业功能,是业务‘必不可少’的支撑。”因此,遗留软件具有生命周期长以及业务关键性的特点。

如果遇到  
质量低下  
的遗留软件项  
怎么办?

对遗留软  
件进行哪  
些类型的改  
变?



所有软件工程  
师都必须认识  
到,变化是不  
可避免的。不  
要反对变化。

然而不幸的是,遗留软件常常存在另一个特点——质量差<sup>①</sup>。通常,遗留系统的设计难以扩展,代码令人费解,文档混乱甚至根本没有,测试用例和结果从未归档,变更历史管理混乱等,有着数不清的问题。然而,这些系统仍然支撑“核心的应用,并且是业务必不可少的支撑”。该如何应对这种情况?

最合理的解释也许就是什么也不做,至少在不得不进行重大变更之前。如果遗留软件可以满足用户的需求并且可靠运行,那么它就没有失效,不需要修改。然而,随着时间的推移,遗留系统经常会由于下述原因发生演化:

- 软件需要进行适应性调整,从而可以满足新的计算环境或者技术的需求。
- 软件必须升级以实现新的商业需求。
- 软件必须扩展使之具有与更多新的系统和数据库的互操作能力。
- 软件架构必须进行改建使之能适应多样化的网络环境。

当这些变化发生时,遗留系统需要经过再工程(参见第29章)使之适应未来的多样性。当代软件工程的目标是“修改基于进化论理论建立的方法论”,即“软件系统不断经历变更,新的软件系统从旧系统中建立起来,并且……新旧所

<sup>①</sup> 所谓“质量差”是基于现代软件工程思想的,这个评判标准对遗留系统有些不公平,因为在遗留软件开发的年代里,现代软件工程的一些概念和原则可能还没有被人们完全理解。

有系统都必须具有互操作性和协作性” [Day99]。

## 1.2 WebApp的特性

“当我们看到任何程度的稳定性时，Web就会变得完全不同了。”——  
Louis Monier

WebApp与其他软件有哪些不同特性？

WWW的早期（大约从1990年到1995年），Web站点仅包含链接在一起的一些超文本文件，这些文件使用文本和有限的图形来表示信息。随着时间的推移，一些开发工具（例如，XML、Java）扩展了HTML的能力，使得Web工程师在向客户提供信息的同时也能提供计算能力。基于Web的系统和应用<sup>①</sup>（我们将这些总称为WebApp）诞生了。今天，WebApp已经发展成为成熟的计算工具，这些工具不仅可以为最终用户提供独立的功能，而且已经同公司数据库和业务应用集成在一起了。

如1.1.2节所述，WebApp是独特的软件类型中的一种，尽管可能存在质疑WebApp是否是不同的软件。Powell[Pow98]提出基于Web的系统和应用“涉及印刷出版和软件开发之间、市场和计算之间、内部通信和外部关系之间以及艺术和技术间的混合”。绝大多数WebApp具备下列属性：

**网络密集性**（network intensiveness）：WebApp驻留在网络上，服务于不同客户群体的需求。网络提供开放的访问和通信（如因特网）或者受限的访问和通信（如企业内联网）。

**并发性**（concurrency）：大量用户可能同时访问WebApp。很多情况下最终用户的使用模式存在很大差异。

**无法预知的负载量**（unpredictable load）：WebApp的用户数量每天都可能有数量级的变化。周一显示有100个用户使用系统，周四就可能会有10000个用户。

**性能**（Performance）：如果一位WebApp用户必须等待很长时间（访问、服务器端处理、客户端格式化显示），该用户就可能转向其他地方。

**可用性**（availability）：尽管期望百分之百的可用性是不切实际的，但是对于热门的WebApp，用户通常要求能够24/7/365（全天候）访问。澳大利亚或亚洲的用户可能在北美传统的应用软件脱机维护时间要求访问。

**数据驱动**（data driven）：许多WebApp的主要功能是使用超媒体向最终用户提供文本、图片、音频及视频内容。除此之外，WebApp还常被用做访问那些存储在Web应用环境之外的数据库中的信息（例如，电子商务或金融应用）。

**内容敏感性**（content sensitive）：内容的质量和艺术性仍然在很大程度上决定了WebApp的质量。

**持续演化**（content evolution）：传统的应用软件是随一系列规划好的时间间隔发布而演化的，而Web应用则持续地演化。对某些WebApp而言（特别是WebApp的内容），按分钟发布更新，或者对每个请求动态更新页面内容，这些都是司空见惯的事。

**即时性**（immediacy）：尽管即时性（也就是将软件尽快推向市场的迫切需要）是很多应用领域的特点，然而将WebApp投入市场可能只是几天或几周的事<sup>②</sup>。

**安全性**（security）：由于WebApp是通过网络访问来使用的，因此要限制访问的最终用户的数量，即使可能也非常困难。为了保护敏感的内容，并提供保密的数据传输模式，在支持

① 在本书中，Web应用（WebApp）这个术语包含了很多事物，从一个简单的帮助消费者计算汽车租赁费用的网页，到为商务旅行和度假提供全套旅游服务的复杂的Web站点。其中包括完整的Web站点、Web站点的专门功能以及Internet、Intranet或Extranet上的信息处理应用。

② 应用先进的工具，我们只需几小时就能完成复杂网页的开发。



WebApp的整个基础设施上和应用本身内部都必须实施较强的安全措施。

**美观性 (aesthetics):** 不可否认, WebApp的用户界面外观很有吸引力。是否能将产品或是思想成功地推向市场, 界面设计的美观和技术设计同样重要。

可以这样说, 1.1.2节中提到的其他应用类型可能具备上述某些特性, 但WebApp几乎具备了所有属性。

### 1.3 软件工程

要构建能够适应21世纪挑战的软件产品, 就必须认识到以下几个简单的事实:

#### KEY POINT

在制定解决方案之前要理解问题。

#### KEY POINT

设计是一项关键的软件工程活动。

#### KEY POINT

质量和可维护性都来自于好的设计。

“工程不仅仅是一个学科或知识体, 它是一个动词, 一个行为动词, 一个解决问题的方法。”——Scott Whitmire

如何定义软件工程?

- 软件已经深入到我们生活的各个方面, 其后果是, 对软件应用<sup>①</sup>所提供的特性和功能感兴趣的人们显著增多。当要开发一个新的应用领域或嵌入式系统时, 一定会听到很多不同的声音。很多时候, 每个人对发布的软件应该具备什么样的软件特性和功能似乎都有着些许不同的想法。因此, 在制定软件解决方案前, 必须尽力理解问题。
- 年复一年, 个人、企业和政府的信息技术需求日臻复杂。过去一个人可以构建的计算机程序, 现在需要由一个庞大的团队来共同实现。曾经运行在一个可预测、自包含的、特定计算环境下的复杂软件, 现在可以嵌入到消费类电子产品、医疗设备、武器系统等各种环境中执行。这些基于计算机的系统或产品的复杂性, 要求对所有系统元素之间的交互非常谨慎。因此, 设计已经成为关键活动。
- 个人、企业、政府在进行日常运作管理以及战略战术决策时越来越依靠软件。软件失效会给个人和企业带来诸多不便, 甚至是灾难性的失败。因此, 软件必须保证高质量。
- 随着特定应用感知价值的提升, 其用户群和软件寿命也会增加。随着用户群和使用时间的增加, 其适应性和可扩展性需求也会同时增加。因此, 软件需具备可维护性。

由这些简单事实可以得出一个结论: 各种形式、各个应用领域的软件都需要工程化。这也是本书的主题——软件工程。

尽管有很多作者都给出了各自软件工程的定义, 但Fritz Bauer[Nau69]在该主题会议上给出的定义仍是进一步开展讨论的基础:

(软件工程是) 建立和使用一套合理的工程原则, 以便经济地获得可靠的、可以在实际机器上高效运行的软件。

你也许试图在这个定义上增加点什么<sup>②</sup>。它没有提到软件质量的技术层面; 它也没有直接谈到用户满意度或按时交付产品的要求; 它忽略了测量和度量的重要性; 它也没有阐明有效的软件过程的重要性。但Bauer的定义给我们提供了一个基线。什么是可以应用到计算机软件开发中的“合理工程原则”? 我们如何“经济地”获得“可靠的”软件? 如何构建程序使其能够不是在一个而是在多个不同的“实际机器”上都能“高效运行”? 这些都是对软件工程师提出进一步挑战的问题。

IEEE[IEE93a]给出了一个更全面的软件工程的定义:

软件工程是: (1) 将系统化的、规范的、量化的方法应用于软件的开发、运行和维护, 即将

① 在本书的后续部分, 将这些人统称为“利益相关者”。

② 有关软件工程的其它定义, 参见 [www.answers.com/topic/software-engineering#wp\\_note-13](http://www.answers.com/topic/software-engineering#wp_note-13)。



工程化方法应用于软件。(2) 在(1)中所述方法的研究。

## KEY POINT

软件工程包括过程、方法和工具。

## WebRef

《Cross Talk》杂志提供了关于过程、方法和工具的许多具体的信息,网站地址是:  
www.s tsc. hill. af.mil.

然而,对于某个软件开发队伍来说可能是“系统化的、规范的、可量化的”方法,对于另外一个团队却可能是负担。因此,我们需要规范,也需要可适应性和灵活性。

软件工程是一种层次化的技术(如图1-3所示)。任何工程方法(包括软件工程)必须构建在质量承诺的基础之上。全面质量管理、六西格玛和类似的理念<sup>①</sup>促进了不断的过程改进文化,正是这种文化,最终引导人们开发更有效的软件工程方法。支持软件工程的根基在于质量关注点(quality focus)。

软件工程的基础是过程(process)层。软件过程将各个技术层次结合在一起,使得合理、及时地开发计算机软件成为可能。过程定义了一个框架,构建该框架是有效实施软件工程技术必不可少的。软件过程构成了软件项目管理控制的基础,建立了工作环境以便于应用技术方法、提交工作产品(模型、文档、数据、报告、表格等)、建立里程碑、保证质量及正确管理变更。

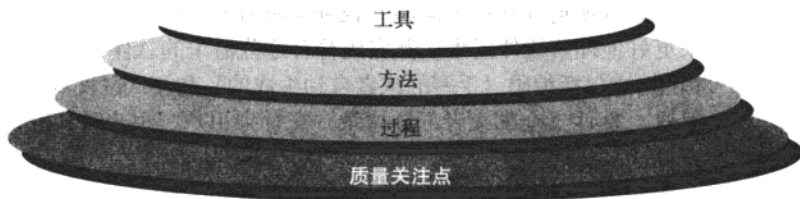


图1-3 软件工程层次图

软件工程方法(method)为构建软件提供技术上的解决方法(“如何做”)。方法覆盖面很广,包括沟通、需求分析、设计建模、编程、测试和技术支持。软件工程方法依赖于一组基本原则,这些原则涵盖了软件工程所有技术领域,包括建模和其他描述性技术等。

软件工程工具(tool)为过程和方法提供自动化或半自动化的支持。这些工具可以集成起来,使得一个工具产生的信息可被另外一个工具使用,这样就建立了软件开发的支撑系统,称为计算机辅助软件工程(computer-aided software engineering)。

## 1.4 软件过程

软件过程包含哪些要素?


过程定义了活动的时间、人员、工作内容和达到预期目标的途径。——Ivar Jacobson, Grandy Booch和James Rumbaugh

软件过程是工作产品构建时所执行的一系列活动、动作和任务的集合。活动(activity)主要实现宽泛的目标(如与利益相关者进行沟通),与应用领域、项目大小、结果复杂性或者实施软件工程的重要程度没有直接关系。动作(action)(如体系结构设计)包含了主要工作产品(如体系结构设计模型)生产过程中的一系列任务。任务(task)关注小而明确的目标,能够产生实际产品(如构建一个单元测试)。

在软件工程领域,过程不是对如何构建计算机软件的严格的规定,而是一种可适应性调整的方法,以便于工作人员(软件团队)可以挑选适合的工作动作和任务集合。其目标通常是及时、高质量地交付软件,以满足软件项目资助方和最终用户的需求。

<sup>①</sup> 质量管理和相关方法在本书第14章及第3部分进行讨论。

过程框架 (process framework) 定义了若干个框架活动 (framework activity), 为实现完整的软件工程过程建立了基础。这些活动可广泛应用于所有软件开发项目, 无论项目的规模和复杂性如何。此外, 过程框架还包含一些适用于整个软件过程的普适性活动 (umbrella activity)。一个通用的软件工程过程框架通常包含以下5个活动:

 五个最基本的过程框架活动是什么?

**沟通** 在技术工作开始之前, 和客户 (及其他利益相关者<sup>①</sup>) 的沟通与协作是极其重要的; 其目的是理解利益相关者的项目目标, 并收集需求以定义软件特性和功能。

**策划** 如果有地图, 任何复杂的旅程都可以变得简单。软件项目好比是一个复杂的旅程, 策划活动就是创建一个“地图”, 以指导团队的项目旅程, 这个地图称为软件项目计划, 它定义和描述了软件工程师工作, 包括需要执行的技术任务、可能的风险、资源需求、工作产品和工作进度计划。

**建模** 无论你是庭园设计家、桥梁建造者、航空工程师、木匠还是建筑师, 你每天的工作都离不开模型。你会画一张草图来辅助理解整个项目大的构想——体系结构、不同的构件如何结合, 以及其他一些特征。如果需要, 可以把草图不断细化, 以便更好地理解问题并找到解决方案。软件工程师也是如此, 利用模型来更好地理解软件需求, 并完成符合这些需求的软件设计。

**构建** 它包括编码 (手写的或者自动生成的) 和测试以发现编码中的错误。

**部署** 软件 (全部或者部分增量) 交付到用户, 用户对其进行评测并给出反馈意见。

上述五个通用框架活动既适用于简单小程序的开发, 也可用于大型网络应用程序的建造以及基于计算机的大型复杂系统工程。不同的应用案例中, 软件过程的细节可能差别很大, 但是框架活动都是一致的。

对许多项目来说, 随着项目的开展, 框架活动可以迭代应用。也就是说, 在项目的多次迭代过程中, 沟通、策划、建模、构建、部署等活动不断重复。每次项目迭代都会产生一个软件增量 (software increment), 每个软件增量实现了部分的软件特性和功能。随着每一次增量的产生, 软件逐渐完善。

软件工程过程框架活动由很多普适性活动来补充实现。通常, 这些普适性活动贯穿软件项目始终, 以帮助软件团队管理和控制项目进度、质量、变更和风险。典型的普适性活动包括:

**软件项目跟踪和控制**——项目组根据计划评估项目进度, 并且采取必要的措施保证项目按进度计划进行。


**风险管理**——对可能影响项目成果或者产品质量的风险进行评估。

**软件质量保证**——确定和执行软件质量保证的活动。

**技术评审**——评估软件工程产品, 尽量在错误传播到下一个活动之前, 发现并清除错误。

**测量**——定义和收集过程、项目和产品的度量, 以帮助团队在发布软件的时候满足利益相关者要求。同时, 测量还可与其他框架活动和普适性活动配合使用。

**软件配置管理**——在整个软件过程中, 管理变更所带来的影响。

 “爱因斯坦认为必然存在着一个对自然界简单的解释, 因为上帝既不专制也不喜怒无常。然而软件工程师却无法抱侥幸心理, 多数情况下, 他必须面对毫无规律的复杂性。”——Fred Brooks

## KEY POINT

普适性活动贯穿整个软件过程, 主要关注于项目管理、跟踪和控制。

① 利益相关者 (Stakeholder) 就是可在项目成功中分享利益的人, 包括业务经理、最终用户、软件工程师、支持人员等。Rob Thomsett曾开玩笑说“Stakeholder就是掌握巨额投资 (stake) 的人……如不照看好你的Stakeholder, 将失去投资”。

**可复用管理**——定义产品复用的标准（包括软件构件），并且建立构件复用机制。

**工作产品的准备和生产**——包括了生成产品（诸如建模、文档、日志、表格和列表等）所必需的活动。

## KEY POINT

对软件过程的普适性调整是项目成功的关键。

过程模型之间有哪些不同之处？

“我认为食谱只是指导方法，一个聪明的厨师每次都会变化出不同的特色。”——  
Madame Benoit

敏捷过程的特点是什么？

**WebRef**  
软件工程实践方面各种深刻的想法都可以在以下网址获得：  
[www.literateprogramming.com](http://www.literateprogramming.com)。

上述每一种普适性活动都将在本书后面详细讨论。

在本节前面部分曾提到，软件工程过程并不是教条的法则，要求软件团队机械地执行，而应该是灵活可适应的（根据软件所需解决的问题、项目特点、开发团队和组织文化等进行适应性调整）。因此，不同项目所采用的项目过程可能有很大不同。这些不同主要体现在以下几个方面：

- 活动、动作和任务的总体流程，以及相互依赖关系。
- 在每一个框架活动中，动作和任务细化的程度。
- 工作产品的定义和要求的程度。
- 质量保证活动应用的方式。
- 项目跟踪和控制活动应用的方式。
- 过程描述的详细程度和严谨程度。
- 客户和利益相关者对项目参与的程度。
- 软件团队所赋予的自主权。
- 队伍组织和角色明确程度。

本书第一部分将详细介绍软件过程。过程模型说明（prescriptive process model）（参见第2章）强调对过程活动和任务的详细定义、识别和应用。其目的是提高软件质量、项目的可管理性以及对于交付时间和项目费用的可预测性，并对软件工程师构建系统所必需的工作提供指导。但遗憾的是，这些目的往往并没有达到。如果只是教条地应用这些过程模型，而没有根据实际情况加以调整，那么，在构建基于计算机的系统的过程中，它将增加官僚作风，并给开发人员和利益相关者制造麻烦。

敏捷过程模型（agile process model）（参见第3章）强调项目的灵活性，并在一些基本原则的指导下，采用非正式的方式（支持者认为这并不会降低过程的有效性）执行软件过程。由于强调可操作性和可适应性，这些过程模型普遍具有敏捷的特征，对某些类型的项目很适用，尤其是Web应用开发。

## 1.5 软件工程实践

在1.4节中，介绍一种由一组活动组成的通用软件过程模型，建立了软件工程实践的框架。



你可能会觉得Polya的方法只是简单的常识，的确如此。但是令人惊奇的是，在软件世界里，很多常识常常不为人知。

通用的框架活动——沟通、策划、建模、构造和部署——和普适性活动构成了软件工程工作的体系结构的骨架。但是软件工程的实践如何融入该框架呢？在以下几节里，读者将会对应用于这些框架活动的基本概念和原则有一个基本了解<sup>①</sup>。

### 1.5.1 实践的精髓

在现代计算机发明之前，有一本经典著作《How to Solve It》，在本书中，George Polya[Pol45]列出了解决问题的本质，这也正是软件工程实践的精髓：

#### 1. 理解问题（沟通和分析）。


<sup>①</sup> 在本书后面对于特定软件工程方法和普适性活动的讨论中，你应该重读本章中的相关章节。

2. 计划解决方案（建模和软件设计）。
3. 实施计划（代码生成）。
4. 检查结果的正确性（测试和质量保证）。

在软件工程中，这些常识性步骤引发了一系列基本问题[改自Pol45]：

**理解问题。**虽然难于承认，但我们遇到的问题很多都源于我们的傲慢。我们只听了几秒钟就断言“好，我懂了，让我们开始解决这个问题吧”。不幸的是，理解一个问题不总是那么容易，需要花一点时间回答几个简单问题：

- 谁将从问题的解决中获益？也就是说，谁是利益相关者？
- 有哪些是未知的？哪些数据、功能、特征和行为是解决问题必需的？
- 问题可以划分吗？是否可以描述为更小、更容易理解的问题？
- 问题可以图形化描述吗？可以建立分析模型吗？

 “在任何问题的解决方案中都会有所发现。”——  
George Polya

**计划解决方案。**现在你理解了要解决的问题（或者你这样认为），并迫不及待地开始编码。在编码之前，稍稍慢下来做一点点设计：

- 以前曾经见过类似问题吗？在潜在的解决方案中，是否可以识别一些模式？是否已经有软件实现了所需要的数据、功能、特征和行为？
- 类似问题是否解决过？如果是，解决方案所包含元素是否可以复用？
- 可以定义子问题吗？如果可以，子问题是否已有解决方案？
- 能用一种可以很快实现的方式来描述解决方案吗？能构建出设计模型吗？

**实施计划。**前面所创建的设计勾画了所要构建的系统的路线图。也许存在没有想到的路径，也可能在实施过程中会发现更好的解决路径，但是这个计划可以保证在实施过程中不至于迷失方向。需要考虑的问题是：

- 解决方案和计划一致吗？源码是否可追溯到设计模型？
- 解决方案的每个组成部分是否可以证明正确？设计和代码是否经过评审？或者更好的算法是否经过正确性证明？

**检查结果。**你不能保证你的解决方案是最完美的，但是你可以保证设计足够的测试来发现尽可能多的错误。为此，需回答：

- 能否测试解决方案的每个部分？是否实现了合理的测试策略？
- 解决方案是否产生了与所要求的数据、功能、特征和行为一致的结果？是否按照项目共同利益者的需求进行了确认？

不足为奇，上述方法大多是常识。但实际上，有充足的理由说明，在软件工程中采用常识，将让你永远不会迷失方向。

### 1.5.2 一般原则

“原则”这个词在字典里的定义是“某种思想体系所需要的重要的根本规则或者假设”。在本书中，我们将讨论一些不同抽象层次上的原则。一部分关注软件工程的整体，另一部分考虑特定的、通用的框架活动（比如沟通），还有一些关注软件工程的动作（比如架构设计）或者技术任务（比如编制用例场景）。无论关注哪个层次，原则都可以帮助我们建立一种思维方式，进行扎实的软件工程实践。因此，原则非常重要。

David Hooker[Hoo96]提出了7个关注软件工程整体实践的原则，这里复述如下<sup>①</sup>：

① 这里的引用得到了作者的授权[Hoo96] Hooker定义这些原则的模式请参见：<http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment>。



在开始一个软件项目之前，应首先确保软件具有商业目标并且让用户体会到它的价值。

### 第1原则：存在价值

一个软件系统因能给用户提供服务而具有存在价值，所有的决定都应该基于这个思想。在确定系统需求之前，在关注系统功能之前，在决定硬件平台或者开发过程之前，问问你自己：这确实能为系统增加真正的价值吗？如果答案是不，那就坚决不做。所有的其他原则都以这条原则为基础。

### 第2原则：保持简洁

软件设计并不是一种随意的过程，在软件设计中需要考虑很多因素。所有的设计都应该尽可能简洁，但不是过于简化。这有助于构建更易于理解和易于维护的系统。这并不是说那些特征甚至是内部特征应该以“简练”为借口而取消。的确，优雅的设计通常也是简洁的设计，简练也不意味着“快速和粗糙”。事实上，它经常是经过大量思考和多次工作迭代才达到的，这样做的回报是所得到的软件更易于维护且存在更少错误。

### 第3原则：保持愿景

清晰的愿景是软件项目成功的基础。没有愿景，项目将会由于它有“两种或者更多种思想”而永远不能结束；如果缺乏概念的一致性，系统就好像是由许多不协调的设计补丁、错误的集成方式强行拼凑在一起……如果不能保持软件系统体系架构的愿景，将削弱甚至彻底破坏设计良好的系统。授权体系架构师，使其能够保持愿景，并保证系统实现始终与愿景保持一致，这对项目开发成功至关重要。

### 第4原则：关注使用者

有产业实力的软件系统不是在真空中开发和使用的。通常软件系统必定是由开发者以外的人员使用、维护和编制文档等，这就必须要让别人理解你的系统。因此，在需求说明、设计和实现时，经常要想到要让别人理解你所做的事情。对于任何一个软件产品，其工作产品都可能有很多读者。需求说明时应时刻想到用户；设计中始终想到实现；编码时想着那些要维护和扩展系统的人。一些人可能会被迫调试你所编写的代码，这使得他们成了你所编写代码的使用者，尽可能地使他们的工作简单化会大大提升系统的价值。

### 第5原则：面向未来

生命期持久的系统具有更高的价值。在现今的计算环境中，需求规格说明随时会改变，硬件平台几个月后就会淘汰，软件生命周期都是以月而不是以年来衡量。然而，真正具有“产业实力”的软件系统必须持久耐用。为了能成功地做到这一点，系统必须能适应这样那样的变化，能成功做到这一点的系统都是那些一开始就以这种路线设计的系统。永远不要把自己的设计局限于一隅，经常问问：“如果出现……应该怎样应对”，构建可以解决通用问题的系统，为各种可能的方案做好准备<sup>①</sup>，这很可能会提高整个系统的可复用性。

### 第6原则：计划复用

复用既省时又省力<sup>②</sup>。软件系统开发过程中，高水平的复用是很难实现的一个目标。代码和设计复用曾宣称是面向对象技术带来的主要好处，然而，这种投入回报不会自动实现。为达到面向对象（或是传统）程序设计技术所能够提供的复用性，需要有前瞻性的设计和计划。系

“简洁比所有巧妙的措词更加美妙。”——  
Alexander Pope  
(1688-1744)



如果软件有价值，其价值在其生命周期中将发生变化。因此，软件必须构建成可维护的。

① 把这个建议发挥到极致可能很危险，设计通用方案会带来性能损失，并降低特定解决方案的效率。

② 尽管对于准备在未来的项目中复用软件的人而言，这种说法是正确的，但对于设计和实现可复用构件的人来说，复用的代价会很昂贵。研究表明，设计和开发可复用构件比直接开发系统要增加25%~200%的成本，在有些情况下，这些费用差别很难核实。

统开发过程中各种层面，都有多种技术实现复用。提前做好复用计划，将降低开发费用，并增加可复用构件以及构件化系统的价值。

### 第7原则：认真思考

这最后一条规则可能是最容易被忽略的。在行动之前清晰定位、完整思考通常能产生更好的结果。仔细思考，可以提高做好事情的可能性，而且也能获得更多的知识，明确如何把事情做好。如果仔细思考过后，还是把事情做错了，那么，这就变成了很有价值的经验。思考的一个副作用是学习和了解本来一无所知的事情，成为研究答案的起点。当明确的思想应用在系统中，就产生了价值。使用前六条原则需要认真思考，这将带来巨大的潜在回报。

如果每位工程师、每个开发团队都能遵从Hooker这七条简单的原则，那么，开发复杂计算机软件系统时所遇到的许多困难都可以迎刃而解。

## 1.6 软件神话

“在缺少有意义的规范标准的情况下，像软件这样的新兴产业转而依靠民间传说。”——Tom DeMarco

### WebRef

软件项目经理网有助于人们澄清这些神话：[www.spmn.com](http://www.spmn.com)。

软件神话，即关于软件及其开发过程被人盲目相信的一些说法，可以追溯到计算技术发展的初期。神话具有一些特点，让人们觉得不可捉摸。例如，神话看起来是事实的合理描述（有时确实包含真实的成分），它们符合直觉，并且经常被那些知根知底、有经验的从业人员拿来宣传。

今天，大多数有见地的软件工程师已经意识到软件神话的本质——它实际上误导了管理者和从业人员对软件开发的态度，从而引发了严重的问题。然而，由于习惯和态度的根深蒂固，这一切难以改变，软件神话遗风犹在。

**管理神话。**承担软件职责的项目经理，像所有领域的经理一样，肩负着维持预算、保证进度和提高质量的压力。就像溺水人抓住稻草一样，软件经理经常依赖软件神话中的信条，只要它能够减轻以上的压力（即使是暂时性的）。

**神话：**我们已经有了本写满软件开发标准和规程的宝典。难道不能提供我们所需要的了解的所有信息吗？

**事实：**这本宝典也许的确已经存在，但它是否已在实际中采用？从业人员是否知道这本书的存在呢？它是否反映了软件工程的现状？是否全面？是否可以适应不同的应用环境？是否在缩短交付时间的同时还关注保证产品的质量？在很多情况下，问题的答案是否定的。

**神话：**如果我们未能按时完成计划，可以通过增加程序员人数而赶上进度。（即所谓的蒙古游牧概念）。



在项目开始之前，尽可能努力了解工作内容。也许难以明确所有细节，但你了解得越多，所面临的风险就越低。

**事实：**软件开发并不是像机器制造那样的机械过程。Brooks曾说过[Bro95]：“在软件工程中，为赶进度而增加人手，只能使进度更加延误。”初看起来，这种说法似乎与直觉不符。然而，当新人加入到一个软件项目中后，原有的开发人员必须要牺牲本来的开发时间对后来者进行培训，因此减少了本应用于高效开发的时间。只有有计划、有序的进行，增加人员对项目进度才有意义。

**神话：**如果决定将软件外包给第三方公司，就可以放手不管，完全交给第三方公司开发。

**事实：**如果开发团队不了解如何在内部管理和控制软件项目，那无一例外地将在外包项目中遇到困难。



每当你认为没有时间采用软件工程方法时，就再问问自己：“是否有时间重做整个软件？”

**客户神话。**软件产品的客户也许是隔壁的某个人，楼下的一个技术团队，市场/销售部门或者签订软件合同的某个外部公司。多数情况下，客户之所以相信所谓的软件神话，是因为项目经理和从业人员没有及时纠正他们的错误信息。软件神话导致客户错误的期望，最终导致对开发者的不满。

**神话：**有了对项目目标的大概了解，便足以开始编写程序，可以在之后的项目开发过程中逐步充实细节。

**事实：**虽然通常很难得到综合全面且稳定不变的需求描述，但是对项目目标模糊不清的描述将为项目实施带来灾难。要得到清晰的需求描述（经常是逐步变得清晰的），只能通过客户和开发人员之间保持持续有效的沟通。

**神话：**虽然软件需求不断变更，但是因为软件是弹性的，因此可以很容易地适应变更。

**事实：**软件需求的确在随时变更，但随变更引入的时机不同，变更所造成的影响也不同。如果需求变更提出得较早（比如在设计或者代码开发之前），则费用的影响较小<sup>①</sup>，但是，随着时间的推移，变更的代价也迅速增加——因为资源已经分配，设计框架已经建立，而变更更可能会引起剧变，需要添加额外的资源或者修改主要设计架构。

**从业者神话。**在50多年的编程文化的滋养下，软件开发人员依然深信着各种神话。在软件业发展早期，编程被视为一种艺术。旧有的方式和态度根深蒂固。

**神话：**当我们完成程序并将其交付使用之后，我们的任务就完成了。

**事实：**曾经有人说过，对于编程来说，开始得越早，耗费的时间就越长。业界的一些数据显示，60%~80%的工作耗费在软件首次交付顾客使用之后。

**神话：**直到程序开始运行，才能评估其质量。

**事实：**最有效的软件质量保证机制之一——技术评审，可以从项目启动就开始实行。软件评审（参见第15章）作为“质量过滤器”，已经证明可以比软件测试更为有效地发现多种类型的软件缺陷。

**神话：**对于一个成功的软件项目，可执行程序是唯一可交付的工作成果。

**事实：**软件配置包括很多内容，可执行程序只是其中之一。各样工作产品（如模型、文档、计划）是成功实施软件工程的基础，更重要的是，为软件技术支持提供了指导。

**神话：**软件工程将导致我们产生大量无用文档，并因此降低工作效率。

**事实：**软件工程并非以创建文档为目的，而是为了保证软件产品的开发质量。好的质量可以减少返工，从而加快交付时间。

很多软件专业人员已经认识到软件神话的谬误。然而遗憾的是，即使事实证明需要采用更好的方法，习惯性的态度和方法依然导致了错误的管理和技术行为。对于软件开发真实情况的正确理解是系统阐述如何使用软件工程方法解决实际问题的第一步。

## 1.7 这一切是如何开始的

每个软件工程项目都来自业务需求——对现有应用程序的纠错；改变遗留系统以适应新的业务环境；扩展现有应用程序功能和特性；或者开发某种新的产品、服务或系统。

在软件项目的初期，业务需求通常是在简短的谈话过程中，非正式地表达出来。以下的一段简短谈话就是一个典型的例子。

① 许多软件工程师采纳了“敏捷(Agile)”开发方法，通过增量的方式逐步纳入变更，以便控制变更的影响范围和成本。本书第3章讨论敏捷方法。



### 如何开始一个软件项目

**[场景]** CPI公司的会议室里。CPI是一个虚构的生产家用和商用消费产品的公司。

**[人物]** Mal Golden, 产品开发部高级经理; Lisa Perez, 营销经理; Lee Warren, 工程经理; Joe Camallen, 业务发展部执行副总裁。

**[对话]**

**Joe:** Lee, 我听说你们那帮家伙正在开发一个什么通用的无线盒?

**Lee:** 哦, 是的, 那是一个很棒的产品, 只有火柴盒大小。我们可以把它放在各种传感器上, 比如数码相机里, 总之任何东西里。采用802.11g无线网络协议, 可以通过无线连接获得它的输出。我们认为它可以带来全新一代产品。

**Joe:** Mal, 你觉得怎么样呢?

**Mal:** 我当然同意。事实上, 随着这一年来销售业绩的趋缓, 我们需要一些新的产品。Lisa和我已经做了一些市场调查, 我们都认为该系列产品具有很大的市场潜力。

**Joe:** 多大, 底线是多少?

**Mal** (避免直接承诺): Lisa, 和他谈谈我们的想法。

**Lisa:** 这是新一代的家庭管理产品, 我们称之为“SafeHome”。产品采用新型无线接口, 给家庭和小型商务使用者提供一个由电脑控制的系统—家庭安全、监视、应用和设备控制。例如, 你可以在回家的路上关闭家里的空调, 或者诸如此类的应用。

**Lee** (插话): Joe, 工程部已经作了相关的技术可行性研究。它可行且制造成本不高。大多数硬件可以在市场购买产品, 不过软件方面是个问题, 但并非做不到。

**Joe:** 有意思! 我想知道底线。

**Mal:** 在美国, 70%的家庭拥有电脑。如果我们定价合适, 这将成为一个十分成功的产品。到目前为止, 只有我们拥有这一无线控制盒技术。我们将在这个方向上保持两年的领先地位。收入嘛, 在第二年大约可达到3千万到4千万。

**Joe** (微笑): 我很感兴趣, 让我们继续讨论一下。

除了一带而过地涉及软件, 这段谈话中几乎没有提及软件开发项目。然而, 软件将是SafeHome产品线成败的关键。只有SafeHome软件成功, 该产品才能成功。只有嵌入其中的软件产品满足顾客的需求 (尽管还未明确说明), 产品才能被市场所接受。我们将在后面的几章中继续讨论 SafeHome中软件工程的话题。

## 1.8 小结

软件是以计算机为基础的系统和产品中的关键部分, 并且成为世界舞台上最重要的技术之一。在过去的50年里, 软件已经从解决特定问题和信息分析的工具发展为独立的产业。然而, 如何在有限的时间内, 利用有限的资金开发高质量的软件仍然是我们所面对的难题。

软件——程序、数据和描述信息——覆盖了科技和应用的很多领域。遗留软件仍旧给维护人员带来了特殊的挑战。

基于Web的系统和应用已经从简单的信息内容集合演化为能够展示复杂功能和多媒体信息

⊖ SafeHome项目将作为一个案例贯穿本书, 以便说明项目组在开发软件产品过程中的内部工作方式。公司、项目和人员都是虚构的, 但场景和问题是真实的。

的复杂系统。尽管Web应用具有独特的特性和需求，他们仍然属于软件范畴。

软件工程包含过程、方法和工具，这些工具使得快速构建高质量的复杂的计算机系统成为可能。软件过程包括五个框架活动：沟通、策划、建模、构建和部署，这些活动适用于所有软件项目。软件工程实践遵照一组核心原则，是一个解决问题的活动。

尽管我们关于构建软件所需的软件知识和技能增长了，但仍有大量的软件神话将管理者和从业人员诱入歧途。随着对软件工程理解的深化，你就会逐渐明白，为什么无论何时遇到这些神话，都要不遗余力地揭露。

## 习题与思考题

- 1.1 举出至少5个例子来说明“意外效应法则”在计算机软件方面的应用。
- 1.2 举例说明软件对社会的影响（包括正面影响和负面影响）。
- 1.3 针对1.1节提出的5个问题，请给出你的答案，并与同学讨论。
- 1.4 在交付最终用户之前，或者首个版本投入使用之后，许多应用程序都会有频繁的变更。为防止变更引起软件退化，请提出一些有效的解决措施。
- 1.5 思考1.1.2节中提到的7个软件分类。请问能否采用一个软件工程方法，应用于所有的软件分类？并就你的答案加以解释。
- 1.6 图1-3中，将软件工程三个层次放在了“质量关注点”这层之上。这意味着在整个开发组织内采用质量管理活动，如“全面质量管理”。仔细研究，并列出全面质量管理活动中关键原则的大纲。
- 1.7 软件工程对构建Web应用是否适用？如果适用，如何改进，以适应Web应用的独特特点？
- 1.8 随着软件的普及，由于程序错误所带来的公众风险已经成为一个愈加重要的问题。设想一个真实场景，由于软件错误而引起“世界末日”般重大危害（危害社会经济或是人类生命财产安全）。
- 1.9 用自己的话描述过程框架。当我们谈到框架活动适用于所有的项目时，是否意味着对于不同规模和复杂度的项目，可应用相同的工作任务？请解释。
- 1.10 普适性活动存在于整个软件过程中，你认为它们均匀分布于软件过程中，还是会集中在某个或者某些框架活动中？
- 1.11 在1.6节所列举的神话中，增加两种软件神话，同时指出与其相对应的真实情况。

## 推荐读物与阅读信息<sup>①</sup>

在数千本关于软件工程的书中，大多数讨论的是程序设计语言和软件应用，很少有涉及软件本身的。Pressman 和Herron（《Software Shock》，Dorset House, 1991）最早讨论了软件和专业开发方法的问题（针对门外汉）。Negroponte的畅销书《Being Digital》，Alfred A. Knopf, Inc., 1995）提供了关于信息论和其在21世纪发展和影响的观点。Demarco（《Why does Software Cost So Much?》，Dorset House, 1995）就软件和开发过程发表了一系列惊人且见解独到的论文。

Minasi在其著作中（《The Software Conspiracy: Why Software Companies Put Out Faulty Products, How They Can Hurt You, and What You Can Do》，McGraw-Hill, 2000）认为，现在由于软件缺陷引起的灾难将被消除，并提出了解决的方法。Compaine（《Digital Divide: Facing a Crisis or Creating a Myth》，MIT Press, 2001）认为，在本世纪的第一个十年里，是否能够访问到信息资源（如Web）的差别将越来越小。在Greenfield的著作（《Everyware: The Dawning Age of Ubiquitous Computing》，New Riders

① 在每章小结之后，“进一步阅读资料”小节简单介绍了本章相关资料，以便于读者扩展阅读和深入理解本章内容。针对本书，我们已经建立了网站<http://www.mhhe.com/pressman>。网站涉及软件工程的很多主题，并逐章列出了相关的软件工程网站资料信息以作为本书的补充，并给出了每一本书在Amazon.com的链接。

Publishing, 2006) 和Loke的著作(《Context-Aware Pervasive Systems:Architectures for a New Breed of Applications》, Auerbach, 2006) 中介绍了“普适性”(open-world)软件的概念, 并指出在无线网络环境中软件必须能够进行适应性调整, 以满足实时涌现的需求。

软件工程及软件过程的当前发展状况可以参阅期刊, 如《IEEE Software》、《IEEE Computer》、《CrossTalk》和《IEEE Transactions on Software Engineering》。行业期刊例如《Application Development Trends》和《Cutter IT Journal》通常包含一些关于软件工程的文章。每年, 由IEEE 和ACM资助的研讨会论文集《Proceeding of the International Conference on Software Engineering》, 都是对当年学术成果的总结, 并且在《ACM Transactions on Software Engineering and Methodology》、《ACM Software Engineering Notes》和《Annals of Software Engineering》等期刊上有进一步的深入讨论。当然在互联网上有很多关于软件工程和软件过程的网页。

近年出版了许多关于软件过程和软件工程的书籍, 有些是关于整个过程的概要介绍, 有些则深入讨论过程中的一些重要专题。下面是一些畅销书:

Abran, A.和J. Moore, 《SWEBOK: Guide to the Software Engineering Body of Knowledge》, IEEE, 2002.

Andersson, E., 等《Software Engineering for Internet Applications》, The MIT Press, 2006.

Christensen, M.和R. Thayer, 《A Project Manager's Guide to Software Engineering Best Practices》, IEEE-CS Press (Wiley), 2002.

Glass, R., 《Fact and Fallacies of Software Engineering》, Addison-Wesley, 2002.

Jacobson, I., 《Object-Oriented Software Engineering: A Use Case Driven Approach, 2nd ed.》, Addison-Wesley, 2008.

Jalote, P., 《An Integrated Approach to Software Engineering》, Springer, 2006.

Pfleeger, S., 《Software Engineering: Theory and Practice, 3rd ed.》, Prentice-Hall, 2005.

Schach, S., 《Object-Oriented and Classical Software Engineering, 7th ed.》, McGraw-Hill, 2006.

Sommerville, I., 《Software Engineering, 8th ed.》, Addison-Wesley, 2006.

Tsui, F., and O.karam, Essentials of Software Engineering, Jones&Bartlett Publishers, 2006.

在过去的几十年里, IEEE、ISO以及附属其下的标准化组织发布了大量软件工程标准。Moore(《The Road Map to Software Engineering: A Standards-Based Guide》, Wiley-IEEE Computer Society Press, 2006) 对相关标准进行了调查并指出了这些标准如何应用到实际工程中。

因特网上有很多有关软件工程和软件过程相关问题的信息资源。许多最新的软件过程相关资源可以参阅本书网站: <http://www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm>。