

建 模

在本书这一部分，读者将学到构建高质量需求模型和设计模型的基本原则、概念和方法。在接下来的章节中，将涉及如下问题：

- 指导软件工程实践有哪些概念和原则？
- 什么是需求工程？什么是能导致良好需求分析的基本概念？
- 如何创建需求模型？它包含哪些元素？
- 好的设计包含哪些元素？
- 体系结构设计如何为其他的设计活动建立一个架构，使用什么模型？
- 如何设计高质量软件部件？
- 在设计用户接口时使用什么概念、模型及方法？
- 什么是基于模式的设计？
- 使用什么特别策略和方法设计WebApp？

一旦解决了以上问题，准备工作也就已经完成了，可以马上开始软件工程应用实践了。

指导实践的原则

要点浏览

概念：软件工程实践是软件计划和开发时需要考虑的方方面面，包括概念、原则、方法和工具等。指导实践的原则成为软件工程实施的基础。

人员：软件工程实践由实践者（软件工程师）和软件项目经理共同完成多种软件工程任务。

重要性：软件过程为每个开发计算机系统或产品的人提供了成功抵达目的地的路线图。实践为你提供了沿路驾驶的细节，它会告诉你哪里有桥、哪里有路障、哪里有岔路，它帮助你理解一些必须理解的并且必须遵循的快速安全驾驶概念和原则，它指示你如何驾驶、在哪里应该减速、在

哪里应该加速。在软件工程中，实践就是要将软件由想法转化为现实时你天天应该做的事情。

步骤：不管选择哪种过程模型，都必须运用实践三要素：概念、原则和方法。实践的第四个要素是工具，工具为方法的应用提供支持。

工作产品：实践贯穿于整个技术活动中。这些技术活动开发出由所选软件过程模型定义的所有工作产品。

质量保证措施：首先，要深刻理解目前工作所用到的概念和原则（例如设计）。然后，确信你已经选定了一个合适方法；确信你已经理解了如何运用这种方法以及使用适合此任务的自动工具，并且坚信需要一些技术来保证工作产品的质量。

关键概念

核心原则

指导原则：

编码

沟通

部署

设计

建模

计划

需求

测试

在一本探讨软件工程师生活和思想的书中，Ellen Ullman[Ull97]通过一个生活片段描述了重重压力下软件工作者的思索：

我对于时间已经没有概念。在这间办公室里，没有窗户也没有时钟，只有红色LED（发光二极管）微波显示器在闪烁，它不断地闪现着12:00, 12:00, 12:00, 12:00。Joel和我已经折腾了好几天，有一个bug，一个很难处理的bug。这红色的闪烁脉冲就像我们的大脑，一直在以相同的速率闪烁着……

我们在做什么？……具体是什么我现在也不知道。我们可能是在帮助可怜的病人，或者是在分布式数据库协议上调整一组底层例程来验证比特流——这些我并不关心。我应该关心，在其他时间——在此之后，或许当我们从这间到处都是电脑的房间中出来的时候，我会非常关心为什么、为了谁以及为了什么目标而开发软件。但是现在不。我已经穿过了一层隔膜，在这里真实世界及其用处都已不再重要，我是软件工程师……

这段文字展现了软件工程实践的黑暗画面，而这种情况很可能和本书大多数读者都有关系。

计算机软件开发人员日常从事的艺术、工艺或者规范性活动^①，就是软件工程。那什么是软件工程“实践”？一般来讲，实践就是软件工程师每天使用的概念、原则、方法和开发工具的集合。实践使得项目经理可以管理软件项目，保证软件工程师开发计算机程序。实践利用由

① 一些作者主张使用这些词语中的一个，而排除其他词语。事实上，软件工程包括所有这三个含义。

必要技术和管理组成的软件过程模型，保证开发工作顺利开展。实践将一些杂乱的容易被忽视的方法转化为更具组织性、更高效并且更容易获得成功的重要东西。

软件工程实践的各个不同方面将在本书的剩余部分介绍。本章的重点是指导软件工程实践的原则和概念。

4.1 软件工程知识

十年前编辑出版的《IEEE Software》期刊中，Steve McConnell[McC99]发表了以下评论：

许多软件实践者认为软件工程知识无疑与特定技术知识：Java、Perl、html、C++、Linux、Windows NT等相同。这些特定技术知识是用来进行计算机程序设计的。如果某人派你去编写一个C++程序，你就得掌握关于C++的知识来完成你的编程工作。

经常听到人们说，软件开发知识的半衰期为3年：现在你需要知道的那些知识，在三年内会有一半将过时。在技术相关的知识领域内，这种说法可能是正确的。但还有另一种软件开发知识——一种我认为是“软件工程原则”——并没有3年半衰期的说法。这些软件工程原则可以为专业程序设计人员在其整个职业生涯内提供服务。

McConnell 继续论述了软件工程知识体（大约在2000年）已经演变为“稳定的核心”，他估计该知识体提供了大约“开发一个复杂系统所需的75%知识”。但是这个“稳定的核心”里面都有些什么呢？

如McConnell 所述，核心原则——指导软件工程师工作的基本概念——现在提供从软件工程模型、方法及工具中得来的，可以应用和评价的基本原理。

4.2 核心原则



“理论上

理论与实践没有区别，而事实上，差别是存在的。”
——Jan van de Snepscheut

软件工程是以一系列核心原则作指导的，这些核心原则为应用具有重大意义的软件过程以及执行有效的软件工程方法提供了帮助。在过程级上，核心原则建立了哲学基础从而指导软件开发团队执行框架活动和普适性活动、引导过程流以及生产一系列软件工程产品。在实践级，核心原则建立了一系列价值和规则，为分析问题、设计解决方案、实现和测试解决方案以及最终在用户社区部署软件提供指导。

在第1章，我曾提出一系列跨越软件工程过程和实践的通用原则：（1）为最终用户提供价值，（2）保持简洁，（3）维护可见的东西（产品和计划），（4）认识（必须理解）到别人消费你生产的产品，（5）面向未来，（6）计划复用，以及（7）认真思考！虽然这些通用原则非常重要，但是以这样高度抽象的语言来表述，使它们有时很难转化为日常的软件工程实践。在以下的部分，我将更为具体地着眼于指导过程和实践的核心原则。

4.2.1 指导过程的原则

在本书的第一部分，我讨论了软件过程的重要性，描述了提供给软件工程的许多不同的过程模型。没有论及模型是线性的或是迭代的、传统的或敏捷的，这可以用适用于所有过程模型的普通过程框架来描述。以下一组核心原则能够适用于框架，并延伸至每一个软件过程。

原则1：敏捷。你所选择的过程模型是否是传统的或敏捷的，敏捷开发的基本原则会提供给你判断的方法。你所做的工作的每一方面都应着重于活动的经济性——保持你的技术方法尽可能简单，保持你的工作产品尽可能简洁，



每一个计划和每一个团队都是独立的。这就意味着，必须使过程尽可能地适应于需求。



“事实是
你总是
知道该做的正
确事情。困难
的是去做。”

—— General
H.Norman Sch-
warzkopf

无论何时尽可能根据具体情况做出决定。

原则2：每一步都关注质量。每个过程活动、动作及任务的出口条件应关注所生产的工作产品质量。

原则3：做好适应的准备。过程不是信仰体验，其中没有信条。当需要的时候，就让你的方法适应于由问题、人员以及项目本身施加的限制。

原则4：建立一个有效的团队。软件工程过程和实践是重要的，但最根本的还是人。必须建立一个彼此信任和尊重的自组织团队。

原则5：建立沟通和协调机制。项目失败是由于遗漏重要信息，以及（或者）利益相关者未能尽力去创建一个成功的最终产品。这些属于管理的问题，必须设法解决。

原则6：管理变更。管理变更的方法可以是正式的或非正式的，但是必须建立一种机制，来管理变更要求的提出、变更的评估、变更的批准以及变更实施的方式。

原则7：评估风险。在进行软件开发时，会做错很多事。建立应变计划是非常重要的。

原则8：创造能给别人带来价值的工作产品。仅仅创建那些能为其他过程活动、动作或任务提供价值的工作产品。每一个工作产品都作为软件工程实践的一部分传递给别人。需求功能和特征表会传递给开发设计的人员，设计会传递给编码的人，等等。一定要确保工作产品所传达的是必要信息不会模棱两可或遗失。

本书的第4部分重点是在项目管理和过程管理问题，以及这些原则各个方面的细节。

4.2.2 指导实践的原则

软件工程实践有一个最重要的目标——按时交付包含了满足所有利益相关者要求的功能和特性的高质量、可运行软件。为了实现这一目标，必须采用一系列的核心原则来指导技术工作。这些原则的优点是不用考虑你所使用的分析方法和设计方法、你所使用的构建技术（例如，程序设计语言、自动工具），或者你所选用的验证和确认方法。以下列举的一组核心原则是软件工程实践的基础。

KEY POINT

当问题被分解为多个独立的部分，问题会比较容易求解，每个部分都是明确的、独立求解和可验证的。

原则1：分治策略（分割和攻克）。更具技术性的表达方式是：分析和设计中应经常强调关切问题分解（SoC, Separation of Concerns）。一个大的问题分解为一组小的元素（或关切问题）之后就比较容易求解。从概念上讲，每个关切问题都传达了可以开发、在某些情况下可被确认的特定功能，这与其他关切问题是无关的。

原则2：理解抽象的使用。在这一核心原则中，抽象就是对一个系统中一些复杂元素的简单化，用以传达词组的含义。我使用抽象报表，是假设你可以理解什么是报表，报表表示的是目录的普通结构，可以将经典的功能展示其中。在软件工程实践中，可以使用许多不同层次的抽象，每个通告或暗示都意味着必需交流信息。在分析和设计中，软件团队通常从高度抽象的模型开始（例如，报表），然后逐渐将这些模型提炼成较低层次的抽象（例如，专栏或SUM功能）。

Joel Spolsky[Spo02]提出“在某种程度上，即使是正规的抽象也都是有漏洞的。”抽象的意图是减弱交流细节的需求。但有时，由于细节的“渗漏”，有问题的影响会不期而至。由于对细节不了解，所以对问题产生的原因并不容易分析。

原则3：力求一致性。是否创建一个需求模型、开发一个软件设计、开发源代码、或者创建测试用例，一致性原则建议熟悉的上下文使软件易于使用。例如，为WebApp设计一个用户界面，一致的菜单选择、一致的彩色设计的使用，以及可识别的图标的一致性使用都有助于使界面在人体工程学方面更为合理。

KEY POINT

未来的软件工程师使用范例（第12章）来获取知识和经验。

原则4：关注信息传送。软件所涉及的信息传送是多方面的——从数据库到最终用户、从现有的应用系统到WebApp、从最终用户到图形用户界面（GUI）、从操作系统到应用问题、从一个软件构件到另一个构件。每一种情况，信息都会流经界面，因而，就有可能出现错误、或者遗漏、或者歧义的情况。这一原则的含义是必须特别注意界面的分析、设计、构建以及测试。

原则5：构建能展示有效模块化的软件。对重要事务的分割（原则1）建立了软件的哲学。模块化提供了认知这一哲学的机制。任何一个复杂的系统都可以被分割许多模块（构件），但是好的软件工程实践不仅如此，它还要求模块必须是有效的。也就是说，每个模块都应该专门地集中表示系统的一个良好约束的方面——其功能具有内聚性和（或）局限于所表示的内容范围。另外，模块应当以相对简单的方式关联起来——每个模块应同其他模块、数据源以及环境方面是低耦合的。

原则6：寻找模式。Brad Appleton[App00]指出：软件界内模式的目标是建立一个典集，帮助软件开发者解决整个软件开发过程中反复出现的问题。模式还有助于创建一种共同语言，交流有关这些问题及其解决办法的见解和经验。对这些解决办法以及它们之间关系的正式地编纂成典可以使成功地捕获知识体系，这一知识体系中明确了对满足用户需求的良好体系结构的认识。

原则7：在可能的时候，用大量不同的观点描述问题及其解决方法。当对一个问题及其解决方法从大量不同的观点进行描述时，就很有可能获得更深刻的认识，发现错误和遗漏。例如，需求模型可以用面向数据的观点、面向功能的观点、或者行为的观点（第6章和第7章）来陈述，每个观点都提供了一个对问题及其需求的不同看法。

原则8：记住：有人将要对软件进行维护。从长期看，缺陷暴露出来时，软件需要修正，环境发生变化时，软件需要适应；利益相关者需要更多功能时，软件需要增强。如果可靠的软件工程实践能够贯穿于整个软件过程，就会便于这些维护活动的实施。

虽然这些原则不能包含构建高质量软件所需要的全部内容，但是它们为本书讨论的每个软件工程方法奠定了基础。

4.3 指导每个框架活动的原则



“理想的软件工程师是复合型的……他不是科学家，不是数学家，不是社会学家或者作家；但是他或许会使用到任一或全部这些学科规范来解决软件工程问题。”——N.W.Dougherty

以下几节中，我关注的是对作为软件过程一部分的每一个通用框架活动的成功产生重要影响的原则。在很多情况下，所讨论的每个框架活动的原则都是对4.2节中提出的原则的提炼。它们只是处于抽象的低层次的核心原则。

4.3.1 沟通原则

在分析、建模或规格说明之前，客户的需求必须通过沟通活动来收集，客户有一些问题适合于使用计算机求解，软件人员就要对客户请求作出响应。沟通从开发者对客户提出的需求做出回应之时就开始了，但是从沟通到理解这条路并不平坦。

高效的沟通（与其他技术人员的沟通、与客户和其他利益相关者的沟通、与项目经理的沟通）是一个软件工程师所面临的最具挑战性的工作。在这里，我们将讨论与客户沟通的原则和概念。不过，许多原则同样适用于软件项目内部的沟通。

原则1：倾听。一定要仔细倾听讲话者的每一句话，而不是急于叙述你对这些话的看法。如果有什么事情不清楚时可以要求他澄清，但是要避免经常打断别人的讲述。当别人正在陈述



在沟通之前确定你理解了其他人的观点，知道他或她需要什么，然后倾听。



“简洁的问题和简洁的回答是解决问题最好的办法。”——Mark Twain

的时候不要在言语或动作上表现出异议（比如：转动眼睛或者摇头）。

原则2：有准备的沟通。在与其他人碰面之前花点时间去理解问题。如果必要的话，做一些调查来理解业务领域的术语。如果你负责主持一个会议，那么在开会之前准备一个议事日程。

原则3：沟通活动需要有人推动。每个沟通会议都应该有一个主持人（推动者），其作用是：（1）保持会议向着有效的方向进行；（2）能调解会议中所发生的冲突；（3）能确保遵循我们所说的沟通原则。

原则4：最好当面沟通。但是，如果能把一些相关信息写出来，通常可以工作得更好，例如可以在集中讨论中使用草图或文档草稿。

原则5：记笔记并且记录所有决定。任何解决方法都可能存在缺陷。参与交流的人应该记录下所有要点和决定。

原则6：保持通力协作。当项目组成员的想法需要汇集在一起用以阐述一个产品或者某个系统的功能或特征的时候，就产生了协作与协调的问题。每次小型的协作都可能建立起项目成员间的相互信任，并且为项目组创建一致的目标。

原则7：把讨论集中在限定的范围内。在任何交流中，参与的人越多，话题转移到其他地方的可能性就越大。最简便的方法就是保持谈话模块化，只有当某个话题完全解决之后再开始别的话题（不过还应该注意原则9）。

原则8：如果某些东西很难表述清楚，采用图形表示。语言沟通的效果很有限，当语言无法表述某项工作的时候，草图或者绘图通常可以让表述变得更为清晰。

原则9：（a）一旦认可某件事情，转换话题；（b）如果不认可某件事情，转换话题；（c）如果某项特性或者功能不清晰，当时无法澄清，转换话题。交流如同所有其他软件工程活动一样需要时间，与其永无止境地迭代，不如让参与者认识到还有很多话题需要讨论（参见原则2），“转换话题”有时是达到敏捷交流的最好方式。

如果无法与客户就项目相关问题达成一致，将会发生什么？

原则10：协商不是一场竞赛或者一场游戏，协商双赢时才发挥了协商的最大价值。很多时候软件工程师和利益相关者必须商讨一些问题，如功能和特征、优先级和交付日期等。若要团队合作得很好，那么各方要有一个共同的目标，并且，谈判还需要各方的协调。

INFO

客户与最终用户之间的差别

软件工程师与许多利益相关者交流，但是客户与最终用户对将采用的技术影响最大。有的时候客户与最终用户是相同的人，但是对于许多项目来说，客户与最终用户是不同的人，为不同商业组织的不同管理者工作。

客户是这样的人（或组织）：（1）最初要求构建软件的人；（2）为软件定义全部业务目标的人；（3）提供基本的产品需求的人；（4）为项目协调资金的人。在产品业务或者系统业务中，客户通常是销售部门；在IT环境下，客户或许是一个业务单位或部门。

最终用户是这样的人（或组织）：（1）为了达到某种商业目的而将真正使用所编写软件的人；（2）为达到其商业目的将定义软件可操作细节的人。

SAFEHOME

沟通出现问题

[场景] 软件工程开发团队工作场所。

[人物] Jamie Lazar、Vinod Raman和Ed Robbins，软件团队成员。

[对话]

Ed: 对这个SafeHome的项目你们听到什么了？

Vinod: 第一次会议安排在下周。

Jamie: 我已经做了一些调查，但是进行得不那么顺利。

Ed: 你的意思是？

Jamie: 是这样，我给Lisa Perez 打了一个电话，她是销售部经理。

Vinod: 然后呢……

Jamie: 我想让她告诉我关于SafeHome的特征和功能……之类的事情。可是，她开始问我一些关于安全系统、监视系统等方面的问题，这些我并不精通。

Vinod: 这对你有何启示？

(Jamie耸了耸肩)

Vinod: 那个销售部门需要我们扮演顾问的角色，我们最好在第一次会议之前对这个产品领域做一些了解。Doug说过他希望我们与客户“协作”，这样才能更好地了解如何开发。

Ed: 也许你应该去她的办公室，电话不适合做这样的工作。

Jamie: 你们说的都对。我们应该努力做好这方面的工作，做好早期的交流。

Vinod: 我看到Doug在看一本“需求工程”的书，我敢打赌这本书上肯定罗列了一些好的交流原则，我要从他那里借来看看。

Jamie: 好主意，然后你就可以教我们啦。

Vinod (微笑): 哈，没问题。

4.3.2 策划原则

沟通活动可以协助软件开发团队定义其全局目标(主题,当然这会随着时间的推移而变化)。可是,理解这些目标与为达到这些目标而制定计划并不是一回事。策划活动包括一系列管理和技术实践,可以为软件开发团队定义一个便于他们向着战略目标和战术目标前进的路线图。

不管我们做多少努力,都不可能准确预测软件项目会如何进展。也不存在简单的方法来确定可能遇到的不可预见的技术问题,在项目后期还有什么重要信息没有掌握,以及会出现什么误解,或者会有什么商务问题发生变化。然而,软件团队还必须制定计划。

“在为战役做准备中,我经常发现,计划是无用的,但做计划是必不可少的。”——
General Dwight D.Eisenhower

有很多不同的制定计划的哲学^①。一些人是“最低要求者”,他们认为变化常常会消除详细计划的必要性;另一些人是“传统主义者”,他们认为计划提供了有效的路线图,并且计划得越详细,团队损失的可能性就越小;也有一些人是“敏捷主义者”,他们认为快速制定计划是必需的,而路线图将会在真正的软件开发工作开始时浮现出来。

要做什么?在许多项目中,“过度计划”是一种时间的浪费并且是在做无用功(因为事物有太多的变化),但是“最起码的计划”是制止混乱的良方。就像在生活中的很多事情一样,适度执行计划足以为团队提供有用的指导——

① 关于软件项目制定计划和管理的具体讨论在本书第4部分给出。

WebRef

一个关于计划和项目管理信息的优秀资料库可以在以下网址找到：
www.4pm.com/repository.html。

“成功更多的是一种协调能力而不是天赋。”
——An Wang

不多也不少。无论多么严格地制定计划，都应该遵循以下原则。

原则1：理解项目范围。如果你不知道要去哪里，就不可能使用路线图。范围可以为软件开发团队提供一个目的地。

原则2：吸收利益相关者参与策划。利益相关者能够限定一些优先次序，确定项目的约束。为了适应这种情况，软件工程师必须经常商谈交付的顺序、时间表以及其他与项目相关的问题。

原则3：要认识到计划的制定应按照迭代方式进行。项目计划不可能一成不变。在工作开始的时候，有很多事情有可能改变，那么计划就必须调整以适应这些变化。另外，迭代式增量过程模型指出了要根据用户反馈的信息（在每个软件增量交付之后）重新制定计划。

原则4：基于已知的估计。估计的目的是基于项目组对将要完成工作的当前理解，提供一种关于工作量、成本和任务工期的指标。如果信息是含糊的或者不可靠的，估计也将是不可靠的。

原则5：计划时考虑风险。如果团队已经明确了哪些风险最容易发生且影响最大，那么应急计划就是必需的了。另外，项目计划（包括进度计划）应该可以调整以适应那些可能发生的风险。

原则6：保持脚踏实地。人们不能每天每时每刻都工作。噪音总能侵入人们的交流之中。生活的现实常常会有疏忽与含糊。变化总是在发生。甚至最好的软件工程师都会犯错误，这些现实的东西都应该在项目制定计划的时候考虑。

KEY POINT

粒度表明项目计划中表述和控制的元素的详细程度。

原则7：调整计划粒度。粒度主要指项目计划细节的精细程度。一个“细粒度”的计划可以提供重要的工作任务细节，这些细节是在相对短的时间段内计划完成的（这样就常常会有跟踪和控制的问题）。一个“粗粒度”的计划提供了更宽泛的长时间工作任务。通常，粒度随项目的进行而从细到粗。在几个星期或几个月的时间里可以详细地策划项目，而在几个月内都不会发生的活动则不需要细化（太多的东西将会发生变化）。

原则8：制定计划确保质量。计划应该确定软件开发团队如何去确保开发的质量。如果要执行正式技术评审^①的话，应该将其列入进度；如果在构建过程中用到了结对编程（第3章），那么在计划中要明确描述。

原则9：描述如何适应变化。即使最好的策划也有可能被无法控制的变化破坏。软件开发团队应该确定在软件开发过程中一些变化需要怎样去适应，例如，客户会随时提出变更吗？如果提出了一个变更，团队是不是要立即去实现它？变更会带来怎样的影响和开销？

原则10：经常跟踪并根据需要调整计划。项目一次会落后进度一天的时间。因此，需要每天都追踪计划的进展，找出计划与实际执行不一致的问题所在，当任务进行出现延误时，计划也要随之做出调整。

最高效的方法是软件开发项目组所有成员都参与到策划活动中来，只有这样，项目组成员才能很好地认可所指定的计划。

4.3.3 建模原则

我们可以通过创建模型来更好地理解需要构建的实体。当实体是物理实物（例如：一栋建筑、一架飞机、一台机器）时，我们可以构建在形式和形状上都和实物相同只是比实物缩小了

^① 技术评审在第15章讨论。

的模型。可是，当实体是一个软件的时候，我们的模型就是另外一种形式了。它必须能够表现出软件所转换的信息、使转换发生的架构和功能、用户要求的特征以及转换发生时系统的行为。模型必须能在不同的抽象层次下完成那些目标——首先从客户的角度描述软件，然后在更侧重于技术方面表述软件。

KEY POINT

需求分析模型表达了客户的需求。设计模型为软件的构造提供了详细的描述。

ADVICE

任一模型的目标都是传递信息。为了实现这一目标，使用一致的格式。假设你不能够解释此模型，那么它始终都只是个模型而已。

在软件工程中，要创建两类模型：需求模型和设计模型。需求模型（也称为分析模型）通过在以下三个不同域描述软件来表达客户的需求：信息域、功能域和行为域。设计模型表述了可以帮助开发者高效开发软件的特征：架构、用户界面以及构件细节。

Scott Ambler和Ron Jeffries[Amb02b]在他们关于敏捷建模的书中定义了一系列建模原则^①，提供给使用敏捷模型（第3章）的人，但也适用于执行建模活动和任务的软件工程师。

原则1：软件团队的主要目标是构建软件而不是创建模型。敏捷的意义是尽可能最快地将软件提供给用户。可以达到这个目标的模型是值得软件团队去构建的，但是，我们需要避免那些降低了开发过程的速度以及不能提供新的见解的模型。

原则2：轻装前进——不要创建任何你不需要的模型。变化发生时，创建的模型必须是最新的。更重要的是，每创建一个新模型所花费的时间，还不如花费在构建软件上（编码或测试）。因此，只创建那些可以使软件的构建更加简便和快速的模型。

原则3：尽量创建能描述问题和软件的最简单模型。不要建造过于复杂的软件[Amb02b]。保持模型简单，产生的软件必然也会简单。最终的结果是，软件易于集成、易于测试以及易于维护（对于变更）。另外，简单的模型易于开发团队成员理解和评判，从而使得反馈正在进行的形式可以对最终结果进行优化。

原则4：用能适应模型改变的方式构建模型。假设模型发生了变化，但做这种假设并不草率。例如，由于需求发生变化，就需要迅速改变需求模型。为什么？因为不管怎样它们都会发生变化。所带来的问题是如果没有相当完整的需求模型，你所创建的设计（设计模型）会常常丢失重要功能和特性。

原则5：明确描述创建每一个模型的目的。每次创建模型，都问一下自己为什么这么做。如果不能为模型的存在提供可靠正当的理由，就不要再在这个模型上花费时间。

原则6：调整所开发模型来适应待开发系统。或许需要使模型的表达方式或规则适用于应用问题；例如，一个视频游戏应用或许需要的建模技术与实时的、嵌入式控制汽车引擎的软件所需的建模技术完全不同。

原则7：尽量构建有用的模型而不是完美的模型。当构建需求模型和设计模型时，软件工程师要达到减少返工的目的。也就是说，努力使模型绝对完美和内部一致并不值得。我建议过模型应当草率或低质量吗？答案是“没有”。但是，对当前建模工作的管理要始终考虑到软件的下一步骤的实施。无休止地使模型“完美”并不能满足敏捷的要求。

原则8：对于模型的构造方法不要过于死板。如果模型能够成功地传递信息，那么表述形式是次要的。虽然软件团队的每个人在建模期间都应使用一致的表达方式，但模型最重要的特性是交流信息，以便软件工程执行下一个任务。如果一个模型可以成功地做到这一点，不正确的表达方式可以忽略。

① 本节给出的原则因本书目标已经做了删节和改写。

原则9：如果直觉告诉你模型不很妥当，尽管看上去很正确，那么你要仔细注意了。如果你是个有经验的软件工程师，就应相信你的直觉。软件工作中有许多教训——其中有些是潜意识的。如果有些事情告诉你设计的模型注定会失败（尽管你不能明确地证明），你有理由再花一些时间来检查模型或开发另一个模型。

原则10：尽可能快地获得反馈。每个模型都应经过软件团队的评审。评审的目的是为了提供反馈，用于纠正模型中的错误、改变误解，并增加不经意遗漏的功能和特性。

需求建模原则。在过去的30年里，已经开发出了大量的需求建模方法。研究人员已经弄清了需求分析中的问题及其出现原因，也开发出了各式各样的建模表达方法以及相关启发性解决方法。每一种分析方法都有其独立的观点。不过，所有的分析方法都具有共同的操作原则：

原则1：必须描述并理解问题的信息域。信息域包括了流入系统的数据（从最终用户、其他系统或者外部设备）、流出系统的数据（通过用户接口、网络接口、报告、图形以及其他方式）以及那些收集和组织的永久性数据对象的数据存储（例如：永久存储的数据）。

KEY POINT

分析模型集中于软件的三个属性：要处理的信息、要发布的功能和要展现的行为。

原则2：必须确定软件所要实现的功能。软件功能直接为最终用户服务并且为用户可见的特征提供内部支持。一些功能对流入系统的数据进行转换。在其他情况下，有些功能在某种程度上影响着对软件内部过程或外部系统元素的控制。功能可以以不同的抽象层次描述，这些抽象层次从对目标的笼统陈述到对那些必不可少的过程细节的描述。

原则3：必须描述软件的行为（作为外部事件的结果）。软件的行为受到与外部环境交互的驱动。最终用户提供的输入，由外部系统提供的控制数据，或者通过网络收集的监控数据都会引起软件的特定行为。

原则4：描述信息、功能和行为的模型必须以一种能揭示分层（或者分级）细节的方式分解开来。需求建模是解决软件工程问题的第一步。它能使开发者更好地理解问题并且为确定解决方案（设计）准备条件。复杂的问题很难完全解决，基于这样的原因，我们使用“分而治之”的战略。把大的复杂问题划分成很多易于理解的子问题，这就是“分解”的概念，这也是分析建模的关键策略。

原则5：分析任务应该从本质信息转向实现细节。需求建模从最终用户角度描述问题开始。在没有考虑解决方案的前提下描述问题的“本质”。例如，一个视频游戏需要玩家在他所扮演角色进入一个危险的迷宫时控制其角色行动的方向，这就是问题的本质。实现细节（通常作为设计模型的一部分来描述）指出问题的本质将如何实现，对于视频游戏来说，可能需要用到语音输入。或者采用另一种办法，键入键盘命令，或者朝某个特定方向移动操纵杆（或者鼠标），或者在空中挥舞动作感应设备。

通过应用这些原则，软件工程师可以系统地解决问题。但这些原则如何应用到实践中呢？对这个问题的回答在第5章到第7章可以找到。

设计建模原则。软件设计模型类似于建筑师的房屋设计方案。首先表达所有需要建造的东西（例如，房屋的三维透视图），然后逐渐进行细化，为构建各个细节（例如，管线分布）提供指导。类似的，软件设计模型为系统提供了各式各样的不同视图。

现在已经有许多方法可以导出各种软件设计的要素。一部分方法是数据驱动的，它通过数据结构来得到程序构架和最终的处理构件；另外一部分方法是模式驱动的，也就是使用问题域（需求模型）信息来开发架构风格和处理模式；还有一些方法是面向对象的，使用问题域对象来驱动创建数据结构以及操作这些数据结构的方法。不过，无论使用什么方法，都使用同一套设计原则。

“工程师在设计时的首要问题是发现真正的问题”。——无名氏

“首先要理解，设计时考虑周全而且合理的：一经证明是无误的，就要坚定地追求它；不要因为有人排斥已经确定的意图，使你动摇取得最终结果的决心。”
—— William Shakespeare

原则1：设计可追溯到需求模型。需求模型描述了问题的信息域、用户可见的功能、系统的行为以及一套需求类，需求类把业务对象和为这些对象服务的方法结合在一起。设计模型将这些信息转化为系统架构、一套实现主要功能的子系统以及一套实现需求类的构件。设计模型的各个元素都应该能追溯到需求模型。

WebRef

带有设计美学讨论、关于设计过程的深入评论可以在以下地址找到：
cs.wmc.edu/~aabyan/Design/.

“差异并不小，就像Salieri和Mozart之间的差异。一项又一项的研究表明最优秀的设计师构建的结构更快、更小、更简单、更清晰、而且不甚费力。”——
Frederick P.Brooks

原则2：要始终关注待建系统的架构。软件架构（参看第9章）是待建系统的骨架，它决定着系统接口、数据结构、程序控制流和行为、测试的方法以及在建系统的可维护性等。基于上述原因，设计应该从考虑架构开始，在架构确定以后才开始考虑构件级设计。

原则3：数据设计与功能设计同等重要。数据设计是架构设计的基本要素。在设计中数据对象的实现方法绝不能忽略，一个结构良好的数据设计可以简化程序流程，让软件构件设计与实现变得更简单，使得整个处理过程更为高效。

原则4：必须精心设计接口（包括内部接口和外部接口）。系统中构件之间数据流的流动方式大大影响着系统的处理效率、误差传播以及设计简单化等方面。一个好的接口设计可以让构件集成变得更为容易并能辅助测试人员确认构件的功能。

原则5：用户界面设计必须符合最终用户要求。在任何情况下，界面的设计都强调使用的方便性。用户接口是软件中可见的部分，无论系统的内部功能多么复杂，无论其数据结构多么容易理解，无论系统架构设计有多好，一个不好的界面设计都会令人感到整个软件很糟糕。

原则6：构件级设计应是功能独立的。功能上独立是软件构件“单一思想”的度量方法。构件提供的功能应该是内聚的——也就是说，它应该关注于一个且仅仅一个功能或子功能^①。

原则7：构件之间以及构件与外部环境之间松散耦合。耦合可以通过很多方式来实现，如构件接口、消息传递及全局数据。随着耦合程度的提高，错误传播的几率也会随之提高，整个软件的可维护性也会降低。因此，应该合理地保持较低的构件耦合度。

原则8：设计表述（模型）应该做到尽可能易于理解。设计的目的是向编码人员、测试人员以及未来的维护人员传递信息，如果设计过于复杂而难以理解，就无法成为一种高效的沟通媒介。


原则9：设计应该迭代式进行。每一次迭代，设计者都应该力求简洁。就像大多数创造性活动一样，设计是迭代完成的，第一次迭代的工作是细化设计并纠正错误，之后的迭代就应该让设计变得尽量简单。

恰当地应用以上设计原则，软件工程师就能创造出兼顾内部高质量和外部高质量的设计[Mye78]。外部质量因素是软件在最终用户使用时所遇到的实际属性（例如：速度、可靠性、正确性、可用性），内部质量因素对与软件工程师来说非常重要，他们要从技术的角度做出高质量的设计。要想有一个高质量的内部设计，设计师们必须理解设计的基本概念（见第8章）。

4.3.4 构造原则

构造活动包括一系列编码和测试任务，从而为向客户和最终用户交付可运行软件做好准备。在现代软件工程中，编码可能是：（1）直接生成编程语言源代码（例如Java），（2）使用待


^① 内聚的附加讨论在第8章。

 “在生活中，我有软件窥探癖，常常偷看到别人很糟糕的代码。偶尔我也能发现一些真正有价值的东西，一些结构很好的代码，这些代码书写形式固定，与具体机器无关，每个构件都是那么简单、易于组织且易于修改。”——David Parnas

 **ADVICE**
避免开发一个解决错误问题的漂亮程序。特别要注意第一个准备原则。

WebRef

各种关于编码标准的链接参看：
www.literateprogramming.com/fpstyle.html。

 **软件测试**
的目标是什么？

开发构件的类似设计的中间表示来自动生成源代码；或者（3）使用第四代编程语言（例如Visual C++）自动生成可执行代码。

最初的测试是构件级的，通常称为“单元测试”。其他级别的测试包括：（1）集成测试（在构建系统的时候进行）；（2）确认测试——测试系统（或者软件增量部分）是否完全按照需求开发；（3）验收测试——由客户检验系统所有要求的功能和特性。在编码和测试过程中有一套原则，下面就讲述这些原则和概念。

编码原则 这些原则和概念与编程风格、编程语言和编程方法紧密结合。下面陈述一些基本的原则：

准备原则。在写下每行代码之前，要确保：

- 理解所要解决的问题。
- 理解基本的设计原则和概念。
- 选择一种能够满足构建软件以及运行环境要求的编程语言。
- 选择一种能提供工具以简化工作的编程环境。
- 构件级编码完成后进行单元测试。

编程原则。在开始编码时，要确保：

- 遵循结构化编程方法来约束算法[Boh00]。
- 考虑使用结对编程。
- 选择能满足设计要求的数据结构。
- 理解软件架构并开发出与其相符的接口。
- 尽可能保持条件逻辑简单。
- 开发的嵌套循环应使其易于测试。
- 选择有意义的变量名并符合相关编码标准。
- 编写注释，使代码具有自说明性。
- 增强代码的可读性（例如：缩进和空行）。

确认原则。在完成第一阶段的编码之后，要确保：

- 适当进行代码走查。
- 进行单元测试并改正所发现的错误。
- 重构代码。

讲述程序设计（编码）以及编码原则和概念的书比软件过程其他论题的书要多很多。如[Ker78]的编程风格入门，[McC04]的软件构造实践，[Ben99]的编程精华，[Knu99]的编程艺术，[Hun99]的实用编程等。关于这些原则和概念的广泛讨论不是这本书的涉及范围。如果你有更多的兴趣，可参看一个或更多上述参考书籍。

测试原则 在一本经典软件测试书中，Glen Myers[Mye79]描述了一系列测试规则，这些规则很好地阐明了测试的目标：

- 测试是一个以查找程序错误为目的的程序执行过程。
- 一个好的测试用例能最大限度地找到尚未发现的错误。
- 一个成功的测试能找到那些尚未发现的错误。

这些目标意味着软件开发者在观念上的一些戏剧性的变化。他们持有与常人相反的观点——常人的观点认为那些找不到一个错误的测试是成功的测试。我们的目标是要设计一些能用最短的时间、最少的工作量来系统地揭示不同类型错误的测试。



在软件设计中，我们通常由着眼于软件架构的“宏观”层面开始，到着眼于构件模块的“微观”层面结束。在测试中，则正好相反。

如果测试成功（按照前面论述的目标），就会发现软件中的错误。测试的第二个好处就是，它能表明软件功能的执行似乎是按照规格说明来进行的，行为需求和性能需求似乎也可以得到满足。此外，测试时收集的数据为软件的可靠性提供了一个很好的说明，并且从整体看来也为软件质量提供了一些说明。但是测试并不能说明某些错误和缺陷不存在；它只能显示出存在的错误和缺陷。在测试时保持这样一个观念是非常重要的，其实这并不是悲观。

Davis[Dav95b]提出了一套测试原则^①，本书对这些原则做了一些改动：

原则1：所有的测试都应该可以追溯到用户需求^②。软件测试的目标就是要揭示错误。而最严重的错误（从用户的角度来看）是那种导致程序无法满足需求的错误。

原则2：测试计划应该远在测试之前就开始着手。测试计划（第17章）在分析模型一完成就应该开始。测试用例的详细定义可以在设计模型确定以后开始。因此，所有的测试在编码前都应该计划和设计好了。

原则3：将Pareto原则应用于软件测试。简单地说，Pareto原则认为在软件测试过程中80%的错误都可以在大概20%的程序构件中找到根源。接下来的问题当然就是要分离那些可疑的构件，然后对其进行彻底的测试。

原则4：测试应该从“微观”开始，逐步转向“宏观”。最初计划并执行的测试通常着眼于单个程序模块，随着测试的进行，着眼点要慢慢转向在集成的构件簇中找错误，最后在整个系统中寻找错误。

原则5：穷举测试是不可能的。即便是一个中等大小的程序，其路径排列组合的数目都非常庞大。因此，在测试中对每个路径组合进行测试是不可能的。然而，充分覆盖程序逻辑并确保构件级设计中的所有条件都通过测试是有可能的。

4.3.5 部署原则

正如我们在本书第一部分中提到的，部署活动包括三个动作：交付、支持和反馈。由于现代软件过程模型实质上是演化式或是增量式的，因此，部署活动并不是只发生一次，而是在软件完全开发完成之前要进行许多次。每个交付周期都会向客户和最终用户提供一个可运行的并具有可用功能和特征的软件增量。每个支持周期都会为部署周期中所提到的所有功能和特征提供一些文档和人员帮助。每个反馈周期都会为软件开发团队提供一些重要的引导，以帮助修改软件功能、特征以及下一个增量所用到的方法。



在提交软件增量前，要确保客户知道所期待的是什么。否则，可以保证，客户期待的一定会超出了你提交的。

软件增量交付对于任何一个软件项目来说都是一个重要的里程碑。以下将讲述一些软件开发团队在准备交付一个软件增量时所应该遵从的重要原则：

原则1：客户对于软件的期望必须得到管理。客户通常并不希望看到软件开发团队对软件的提交更多的作出承诺后又很快令其失望。这将导致客户反馈变得没有积极意义并且还会挫伤软件开发团队的士气。Naomi Karten[Kar94]在她的关于管理客户期望的书中提到：“管理客户期望首先应该认真考虑你该与客户交流什么与怎样交流。”她建议软件工程师必须认真地处理与客户有冲突的信息。（例如：对不可能在交付时完成的工作作出了承诺；

① 这里给出的仅仅是Davis测试原则的一小部分。更多的信息参见[Dav95b]。

② 这个原则是针对功能测试，也就是，测试重点在需求。结构测试（测试重点在体系结构的和逻辑的细节）可能没有直接涉及详细的需求。

在某次软件增量交付时交付了多于当初承诺要交付的工作，这将使得下次增量所要做的工作随之变少。)

原则2：完整的交付包应该经过安装和测试。光盘或其他包含可执行软件的介质（包括Web下载包）以及一些支持数据文件、文档和一些相关的信息都必须组装起来，并经过实际用户完整的 β 测试。所有的安装脚本和其他一些可操作的功能都应该在所有可能的计算配置（例如：硬件、操作系统、外围设备、网络）环境中实施充分的检验。

原则3：技术支持必须在软件交付之前就确定下来。最终用户希望在水发生能时得到及时的响应和准确的信息。如果技术支持跟不上或者根本就没有技术支持，那么客户会立即表示不满。支持应该是有计划的，准备好支持的材料并且建立适当的记录保持机制，这样软件开发团队就能按照支持请求种类进行分类评估。

原则4：必须为最终用户提供适当的说明材料。软件开发团队交付的不仅仅是软件本身，也应该提供培训材料（如果需要的话）和故障解决方案，还应该发布关于“本次增量与以前版本有何不同”的描述^①。

原则5：有缺陷的软件应该先改正再交付。迫于时间的压力，某些软件组织会交付一些低质量的增量，还在增量中向客户提出警告：“这些缺陷将在下次发布时解决。”这样做是错误的。在软件商务活动中有这样一条谚语：“客户在几天后就会忘掉你所交付的高质量软件，但是他们永远忘不掉那些低质量的产品所出现的问题。软件会时刻提醒着问题的存在。”

已交付的软件会为最终用户提供一些好处，同时它也会为软件开发团队提供一些有用的反馈。当一个增量投入使用后，应该鼓励最终用户对软件的功能、特性以及易用性、可靠性和其他特性做出评价。

4.4 小结

软件工程实践包括概念、原则、方法和在整个软件过程中所使用的工具。虽然每个软件工程项目是不同的，但却有着通用的普遍原则和一些与项目或产品无关的适用于每个过程框架活动的实践任务。

核心原则集有助于有意义的软件过程的应用以及有效的软件工程方法的执行。在过程级，核心原则建立了一个哲学基础，指导软件开发团队引导软件过程。在实践级，核心原则建立了一套价值准则，可以在分析问题、设计解决方案、实施方案以及测试解决方案以及最终向用户部署软件时提供指导。

在开发者与客户进行沟通时，客户沟通原则主要着眼于两点：减少争吵和扩大双方的交流广度，双方必须互相协作以更好地交流。

策划原则着眼于为了使开发整个系统或产品沿着最佳路线前进提供指导。计划可以只是为某个软件增量而设计，或者也可以为整个项目而制定。无论如何，计划都必须涉及要做什么，谁来完成以及什么时候完成。

建模包括分析和设计，描绘了逐渐细化的软件。建模的目的是加深对所完成工作的理解并为软件开发人员提供技术指导。建模原则就作为建立对软件进行表述的方法和注释的基础。

构造包括了编码和测试循环，在这个循环中为每个构件生成源码并对其进行测试。编码原则定义了一些通用动作，在编码开始之前这些动作应该已经完成了。尽管有许多的测试原则，但是只有一个是最主要的：测试是一个为了发现错误而执行程序的过程。

部署发生在向客户展示每个软件增量的时候，它包括了交付、支持和反馈。交付的关键原

^① 在交流活动中，软件团队应该确定用户希望的帮助材料是什么类型。

则是管理客户期望并且能为客户提供合适的软件支持信息。支持需要预先准备。反馈允许客户提出一些具有商业价值的变更意见，为开发者的下一个软件工程迭代周期提供输入。

习题与思考题

- 4.1 由于关注质量需要资源和时间，那么既要敏捷又要维持对质量的关注可能吗？
- 4.2 在指导过程的8个核心原则中（在4.2.1节中讨论），你认为哪一个是最重要的？
- 4.3 用你自己的语言描述“关切问题分解”这个概念。
- 4.4 一个重要的沟通原则说到“有准备的沟通”。在早期的工作中你都需要做哪些准备？在早期的准备中都应该产生哪些工作产品？
- 4.5 在指导实践的8个核心原则中（4.2.2节），你认为哪一个是最为重要的？
- 4.6 敏捷沟通与传统的软件工程沟通有什么不同？又有哪些相似之处？
- 4.7 “转换话题”为什么是必要的？
- 4.8 在沟通活动中需要做一些“协商”方面的调查研究，并且为“协商”准备一些指导方针。
- 4.9 说明为什么项目策划需按迭代方式进行？
- 4.10 在软件工程中为什么模型是很重要的？它们总是必需的吗？在你描述其必要性时会用一些修饰的词语吗？
- 4.11 在设计建模的过程中有哪3个软件“特征”需要考虑？
- 4.12 试着为4.3.4节中的编码添加一条附加的原则。
- 4.13 说明为什么穷举测试不能用以证明软件不含有错误？
- 4.14 你是否同意下面的说法：“既然我们要向客户交付多个增量，那么为什么还要关注早期增量的质量——我们可以在今后的迭代中逐步改善这些问题。”说说你的看法。
- 4.15 为什么对于软件开发团队来说反馈是至关重要的？

推荐读物与阅读信息

客户沟通在软件工程中是至关重要的活动，然而很少有团队愿意花时间阅读这方面的资料。Withall的书（《Software Requirements, Patterns》，Microsoft Press, 2007）中提出了大量解决沟通问题的有用模式。Sutliff的书（《User-Centred Requirement Engineering》，Springer, 2002）中重点聚焦于与沟通相关的挑战。Weigers的书（《Software Requirements》，2nd ed. Microsoft Press, 2003），Pardee的书（《To Satisfy and Delight Your Customer》，Dorset House, 1996）以及Karten[Kar94]都提供了许多关于客户交互的有效方法和观点。虽然并没有聚焦于软件，但Hooks和Farry的书（《Customer Centered Products》，American Management Association, 2000）中对与客户沟通提供了有用的通用指导方针。Young（《Effective Requirements Practice》，Addison-Wesley, 2001）强调客户和开发人员组织联合小组进行协作需求分析。Somerville和Kotonya（《Requirements Engineering: Processes and Techniques》，Wiley, 1998）讨论需求诱导的概念和技术以及其他需求工程原则。

沟通和策划的原则与概念在许多项目管理书籍中都有所涉及。一些比较有用的项目管理书籍包括：Bechtold（《Essentials of Software Project Management》，2nd ed. Management Concept, 2007），Wysocki（《Effective Project Management: Traditional, Adaptive, Extreme》，4th ed. Wiley, 2006），Leach（《Lean Project Management: Eight Principles for Success》，BookSurge Publishing, 2006），Hughes（《Software Project Management》，McGraw-Hill, 2005），以及Stellman和Green（《Applied Software Project Management》，O'Reilly Media, Inc., 2005）。

软件工程原则在Davis[Dav95]的书中有非常好的整理和评述。此外，每一本软件工程书中实际上都

包括了对分析、设计和测试概念及原则的有益讨论。其中的佼佼者有（除了本书以外！）：

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

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

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》, McGraw-Hill, 7th ed., 2006.

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

这些书也对建模和创建原则作了详细的讨论。

建模原则可以参考很多着眼于需求工程和（或）软件设计的书。Lieberman的书（《The Art of Software Modeling》, Auerbach, 2007）, Rosenberg和Stephen的书（《Use Case Driven Object Modeling with UML: Theory and Practice》, Apress, 2007）, Roques的书（《UML in Practice》, Wiley, 2004）, Penker 和Eriksson的书（《Business Modeling with UML: Business Patterns at Work》, Wiley, 2001）都讨论了建模原则和方法。

Norman的著作（《The Design of Everyday Things》, Currency/Doubleday, 1990）是每一个设计从业者的必读教材。Winograd及其同事（《Bringing Design to Software》, Addison-Wesley, 1996）汇集了很多讨论软件设计实践的优秀文章。Constantine和Lockwood（《Software for Use》, Addison-Wesley, 1999）提出了与“以用户为中心的设计”相关的概念。Tognazzini（《Tog on Software Design》, Addison-Wesley, 1995）提出一些有价值的哲学讨论，包括关于设计的本质、决策对于软件质量的影响以及团队的能力对向客户提供巨大价值软件的影响。Stahl及其同事（《Model-Driven Software Development: Technology, Engineering》, Wiley, 2006）讨论了模型驱动开发的原则。

很多书籍都关注软件构建活动的一个或多个问题。Kernighan和Plauger[Ker78]写了一本关于编码风格的经典书籍，McConnell[Mcc93]提出了软件构建实践的实用指导原则，Bentley[Ben99]提出大量关于编码的宝贵建议，Knuth[Knu99]写过一套关于编码艺术的三卷的系列丛书，Hunt[Hun99]提出了实用的编程原则。

Myers及其同事（《The Art of Software Testing》, 2nd ed., Wiley, 2004），主要修订了他的经典著作版本，并且讨论了许多重要的测试原则。Perry（《Effective Methods for Software Testing》, 3rd ed., Wiley, 2006）、Whittaker（《How to Break Software》, Addison-Wesley, 2002）、Kaner和他的同事（《Lessons Learned in Software Testing》, Wiley, 2001）以及Marick（《The Craft of Software Testing》, Prentice-Hall, 1997）的著作都提出了重要的测试概念、原则和很多实用指导。

在因特网上有大量关于软件工程实践的信息资源，与软件工程相关的最新WWW参考资料可以在SEPA网站<http://www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm>上找到。