Software Testing Basics

软件测试基础

周元哲 主编

周元哲 胡滨 潘晓英 刘海 编著

西安电子科技大学出版社
http://www.xduph.com

软件测试基础

周元哲　主编

周元哲　胡　滨　潘晓英　刘　海　编著

西安电子科技大学出版社

内 容 简 介

本书较为全面、系统地介绍了当前业界测试领域的理论和实践知识，反映了当前最新的软件测试理论、标准、技术和工具，展望了软件测试的发展趋势。

全书共分三大部分，分别是测试理论、测试实践和测试考试指导。第一部分内容主要包括软件测试概论、软件测试基本知识、软件测试过程、黑盒测试、白盒测试、自动测试技术、性能测试、面向对象测试、嵌入式测试和软件测试管理。第二部分内容主要包括软件测试工具、测试管理工具、性能测试工具、缺陷跟踪管理工具、单元测试工具和功能测试工具等。第三部分内容主要包括计算机认证考试和测试行业，介绍了四级软件测试工程师考试和企业招聘测试工程师考试的一些情况。

本书可作为高等院校相关专业软件测试课程的教材或教学参考书，也可供从事计算机应用开发的各类技术人员参考，或用作全国计算机软件测评师考试、软件技术资格与水平考试的培训资料。

图书在版编目(CIP)数据

软件测试基础 / 周元哲主编. —西安：西安电子科技大学出版社，2011.6

ISBN 978-7-5606-2491-4

Ⅰ. ① 软… Ⅱ. ① 周… Ⅲ. ① 软件—测试 Ⅳ. ① TP311.5

中国版本图书馆 CIP 数据核字(2010)第 202970 号

策　　划　云立实

责任编辑　杨丕勇　云立实

出版发行　西安电子科技大学出版社(西安市太白南路 2 号)

电　　话　(029)88242885　88201467　　邮　　编　710071

网　　址　www.xduph.com　　电子邮箱　xdupfxb001@163.com

经　　销　新华书店

印刷单位　陕西天意印务有限责任公司

版　　次　2011 年 6 月第 1 版　2011 年 6 月第 1 次印刷

开　　本　787 毫米×1092 毫米　1/16　印　张　16

字　　数　377 千字

印　　数　1～3000 册

定　　价　30.00 元

ISBN 978-7-5606-2491-4/TP • 1241

XDUP 2783001-1

前　言

随着软件规模和复杂性的大幅度提升，软件质量可靠性的问题变得日益突出。软件测试是保证软件质量的关键技术之一，同时也是软件开发过程中的一个重要环节，其理论知识和技术工具都在不断革新。

本书较为全面地介绍了当前业界测试领域的专业知识，追溯了软件测试的发展史，反映了当前最新的软件测试理论、标准、技术和工具，展望了软件测试的发展趋势。全书共分三大部分，分别是测试理论、测试实践和测试考试指导。第一部分内容主要包括软件测试概论、软件测试基本知识、软件测试过程、黑盒测试、白盒测试、自动测试技术、性能测试、面向对象测试、嵌入式测试和软件测试管理。第二部分内容主要包括软件测试工具、测试管理工具、性能测试工具、缺陷跟踪管理工具、单元测试工具和功能测试工具等。第三部分内容主要包括计算机认证考试和测试行业，介绍了四级软件测试工程师考试和企业招聘测试工程师考试的一些情况。

软件测试从软件工程中演化而来，并且还在不断地发展。在学习本书之前，需要一些先行知识作为本书的支撑，如掌握一门高级语言(Visual Basic 或 Java 语言)、数据库、数据结构以及软件工程的基本理论知识等。

本书介绍了软件测试的基本理论和当前流行的一些软件测试工具的应用，内容精练，文字简洁，结构合理，综合性强，定位明确，面向初、中级读者，由“入门”起步，侧重“提高”，特别适合作为高等院校相关专业软件测试的教材或教学参考书，也可供从事计算机应用开发的各类技术人员参考，或用作全国计算机软件测评师考试、软件技术资格与水平考试的培训资料。

本书由周元哲、胡滨、潘晓英、刘海编写，其中第 13、16 章由胡滨编写，第 9、12 章由潘晓英编写，第 14 章由刘海编写，其余章节由周元哲编写。周元哲任主编，负责拟订大纲与统稿工作。

西安邮电学院计算机科学与技术学院王忠民院长、王曙燕副院长和陈莉君教授对本书的编写给予了大力的支持并提出了指导性意见。

在本书的写作过程中，西安电子科技大学出版社的云立实、人民邮电出版社的贾楠对写作大纲、写作风格等提出了很多宝贵的意见。本书在写作过程中参阅了大量中外文的专著、教材、论文、报告及网上的资料，由于篇幅所限，未能一一列出。在此，向各位作者表示敬意和衷心的感谢。

由于作者水平有限，本书难免有不足之处，诚恳期待读者的批评指正，以使本书日臻完善。我的电子信箱是 zhouyuanzhe@163.com。

编　者

2010 年 10 月

目　录

第 1 章　软件测试概论

本章主要介绍软件的发展历史，当前流行的软件过程模型、软件缺陷、软件质量以及质量保证体系等理论知识，引出软件测试对于软件开发和软件质量的重要性，为学习本书后续内容作必要准备。

1.1 软　　件

软件是一系列按照特定顺序组织的计算机数据和指令的集合。一般来讲，软件被划分为编程语言、系统软件、应用软件和介于系统软件与应用软件之间的中间件。其中系统软件为使用计算机提供最基本的功能，但是并不针对某一特定应用领域；而应用软件则恰好相反，不同的应用软件根据用户和所服务的领域提供不同的功能。

一般认为，软件包括如下内容：

(1) 运行时，能够提供所要求功能和性能的指令或计算机程序集合。

(2) 程序能够妥善地处理信息的数据结构。

(3) 描述程序功能需求以及程序如何操作和使用的文档。

1.1.1 软件发展史

软件的发展经历了如下几个阶段：

第一阶段从 20 世纪 50 年代初期至 60 年代中期，这一阶段又称为程序设计阶段。此时硬件已经通用化，而软件的生产却是个体化。软件产品为专用软件，规模较小，功能单一，开发者即为使用者，软件只有程序，无文档。软件设计在人们的头脑中完成，形成了“软件等于程序”的错误观念。

第二阶段从 20 世纪 60 年代中期至 70 年代末期，称为程序系统阶段。此时多道程序设计技术、多用户系统、人机交互技术、实时系统和第一代数据库管理系统的出现，催生了专门从事软件开发的“软件作坊”，软件广泛应用，但软件技术和管理水平相对落后，导致“软件危机”出现。软件危机主要表现在以下几个方面：

(1) 软件项目无法按期完成，超出经费预算，软件质量难以控制；

(2) 开发人员和开发过程之间管理不规范，约定不严密，文档书写不完整，使得软件维护费用高，某些系统甚至无法进行修改；

(3) 缺乏严密、有效的质量检测手段，交付给用户的软件质量差，在运行中出现许多问题，甚至产生严重的后果；

(4) 系统更新换代难度大。

第三阶段从20世纪70年代末期至80年代中期称为软件工程阶段。微处理器的出现及分布式系统的广泛应用，使得计算机真正成为大众化的东西。以软件的产品化、系列化、工程化和标准化为特征的软件产业发展起来，软件开发有了可以遵循的软件工程化的设计准则、方法和标准。1968年，北大西洋公约组织的计算机科学家在联邦德国召开国际会议，会议讨论了软件危机问题，正式提出并使用“软件工程”概念。这标志着软件工程的诞生。

软件工程学涉及与生产软件相关的所有活动，相关的学科包括计算机科学、管理学、经济学、心理学等。软件工程研究的主要内容是：如何应用科学的理论和工程上的技术来指导软件的开发，从而达到以较少的投资获得高质量软件的最终目标。

第四阶段从20世纪80年代中期至今，客户端/服务器(C/S)体系结构，特别是Web技术和网络分布式对象技术的飞速发展，导致软件系统体系结构向更加灵活的多层分布式结构演变，CORBA、EJB、COM/DCOM等三大分布式的对象模型技术相继出现。

2006年提出的面向服务架构(Service-Oriented Architecture，简称SOA)作为下一代软件架构，是一种“抽象、松散耦合和粗粒度”的软件架构，根据需求通过网络对松散耦合的粗粒度应用组件进行分布式部署、组合和使用，主要用于解决传统对象模型中无法解决的异构和耦合问题。

至此，软件发展经历了从Mainframe结构、Client/Server结构、B/S多层分布式结构到SOA的演变过程，整个软件系统变得越来越分散、越来越开放、越来越强调互操作性。

1.1.2 软件生命周期

同任何事物一样，一个软件产品或软件系统也要经历孕育、诞生、成长、成熟、衰亡等阶段，一般称为软件生命周期。通过将整个软件生命周期划分为若干阶段，可使得每个阶段有明确的任务，使规模大且结构和管理复杂的软件开发变得容易控制和管理。通常，软件生命周期包括可行性分析与开发项目计划、需求分析、设计(概要设计和详细设计)、编码、测试、维护升级直至废弃等阶段，这种按时间划分过程的思想方法是软件工程中的一种思想原则，即按部就班、逐步推进，每个阶段都要有定义、工作、审查并形成文档，以提高软件的质量。

软件生命周期具有如下六个阶段：

(1) 问题的定义及规划。此阶段由软件开发方与需求方共同讨论，确定软件的开发目标及其可行性。

(2) 需求分析。在确定软件开发可行的情况下，对软件需要实现的各个功能进行详细分析。需求分析阶段作为一个很重要的阶段，在整个软件开发过程中是不断变化和深入的，因此必须制定需求变更计划来应付这种变化，以保护整个项目的顺利进行。

(3) 软件设计。此阶段主要根据需求分析的结果，对整个软件系统进行设计，如系统框架设计、数据库设计等。软件设计一般分为总体设计和详细设计。

(4) 程序编码。此阶段是将软件设计的结果转换成计算机可运行的程序代码。程序编码必须符合标准的编写规范，以保证程序的可读性、易维护性，提高程序的运行效率。

(5) 软件测试。在软件设计完成后要经过严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。整个测试过程分为单元测试、组装测试以及系统测试等

阶段。测试的方法主要有白盒测试和黑盒测试两种。在测试过程中需要建立详细的测试计划并严格按照测试计划进行测试，以减少测试的随意性。

(6) 运行维护。软件维护是软件生命周期中持续时间最长的阶段。在软件开发完成并投入使用后，由于多方面的原因，软件有可能不适应用户的要求，要延续软件的使用寿命，此时就必须对软件进行维护。

1.1.3 软件缺陷

1. 软件缺陷案例

让我们回顾一些“臭名昭著”的软件缺陷案例，它们都是由于软件测试不充分而导致的严重问题。

1963年，由于用FORTRAN程序设计语言编写的飞行控制软件中的循环语句“DO 5 I=1，3”误写为“DO 5 I=1.3”，结果导致美国首次金星探测飞行失败，造成价值约1000多万美元的损失。

1979年，新西兰航空公司的一架客机因计算机控制的自动飞行系统发生故障而撞在阿尔卑斯山上，机上257名乘客全部遇难。

1983年，美国科罗拉多河水泛滥，但由于计算机对天气形势预测有误，水库未能及时泄洪，以致造成严重的经济损失和人员伤亡。

1990年1月15日，通信中转系统软件发生故障，导致主干远程网大规模崩溃，使数以千计的电信运营公司损失惨重。

1992年10月26日，伦敦救护中心的计算机辅助发送系统刚启动就崩溃了，导致这个全世界最大的(每天要接运五千多名病人)救护机构全部瘫痪。

1994年，美国迪士尼公司的“狮子王”软件在少数系统中能正常工作，但在大众使用的常见系统中无法正常运行。后来证实，这是由于迪士尼公司没有对市场上投入使用的各种PC机型进行正确的测试。同年，英特尔奔腾浮点除法发生软件缺陷，英特尔为处理软件缺陷支付了4亿多美元。

1996年6月4日，欧洲航空航天局耗资80亿美元发射的阿里亚娜501火箭，在发射升空37秒后爆炸。原因是主发动机打火顺序开始37秒后，制导信息由于惯性制导系统的软件出现规格和设计错误而完全丢失。

临近2000年时，计算机业界一片恐慌，导致这种现象发生的就是著名的“千年虫”问题。其原因是在20世纪70年代，由于计算机硬件资源很珍贵，程序员为节约内存资源和硬盘空间，在存储日期数据时，只保留年份的后2位，如“1980”被存储为“80”。当2000年到来时，问题出现了，计算机无法分清“00”是指“2000年”还是“1000年”。例如银行存款的软件在计算利息时，用日期“00”减去当时存款的日期，结果存款年数就变为负数，导致顾客反要付给银行巨额的利息。为了解决“千年虫”问题，花费了大量的人力、物力和财力。

2008年，我国举办了首次奥运会。2007年10月30日上午9时，北京奥运会门票面向境内公众销售第二阶段正式启动，系统访问流量猛增，官方票务网站流量瞬时达到每小时800万次，超过了系统设计每小时100万次的承受量，奥运门票系统访问量超过原计划的8

倍，造成网络拥堵，售票速度慢或暂时不能登录系统的情况，直接造成公众无法及时提交购票申请，北京奥运票务中心就此向广大境内公众购票人发布了致歉信。

2. 软件缺陷概念

软件缺陷是指软件系统或系统部件中那些导致系统或部件不能实现其功能的问题或差错。通常认为，符合下面 4 个规则之一的就是软件缺陷。

(1) 软件未达到软件规格说明书中规定的功能；

(2) 软件出现了产品说明书中指明不会出现的错误；

(3) 软件功能超出了产品说明书中指明的范围；

(4) 软件测试人员认为软件难于理解，不易使用，运行速度慢，或者最终用户认为软件使用效果不好。

【例 1-1】 软件缺陷举例。

计算器说明书一般声称该计算器能准确无误地进行加、减、乘、除运算。如果测试人员选定了数值，按下“+”号后，再按下一数值，没有任何结果出现，或者得到了错误的答案，根据第一条规则，这是一个缺陷。

计算器产品说明书指明计算器不会出现崩溃、死锁或者停止反应等情况，如果测试人员按键后，计算器却停止接收等，根据第二条规则，这是一个缺陷。若在测试过程中发现，由于电池没电等原因导致计算不正确，但产品说明书上没有指出此情况下应该如何进行处理，这也是一个缺陷。

若在进行测试时，发现除了产品说明书规定的加、减、乘、除运算功能之外，还能够进行求平方根的运算，而这一功能并没有在说明书中给出，根据第三条规则，这是一个缺陷。

如果测试人员发现计算器某些功能不好使用，如按键太小、显示屏在亮光下无法看清等，根据第四条规则，这是一个缺陷。

3. 软件缺陷分类

按照 CMM5 中定义的规范，软件缺陷分为多个等级，分别为致命、严重、一般和提示。

(1) 致命。致命性漏洞主要为：内存泄漏、用户数据丢失或被破坏、系统崩溃/死机/冻结、模块无法启动或异常退出、严重的数值计算错误、功能设计与需求严重不符、其它导致无法测试的错误和造成系统不稳定等情况。

(2) 严重。严重性漏洞主要为：功能未实现、功能严重错误、系统刷新错误、语音或数据通信错误、轻微的数值计算错误、系统所提供的功能或服务受明显的影响但不会影响到系统稳定性。

(3) 一般。一般性漏洞主要为：操作界面错误(包括数据窗口内列名定义、含义是否一致)、边界条件下错误、提示信息错误(包括未给出信息、信息提示错误等)、长时间操作无进度提示、系统未优化(性能问题)、光标跳转设置不好、鼠标(光标)定位错误等。

(4) 提示。提示性漏洞主要为：界面格式等不规范、辅助说明描述不清楚、操作时未给出用户提示、可输入区域和只读区域没有明显的区分标志、个别不影响产品理解的错别字、文字排列不整齐等一些小问题及建议。

1.1.4　三种纠错技术

纠错先要查错。查错的工作量通常占整个纠错的 90%以上。下面介绍几种常用的查明程序错误时可能采用的工具和手段，这些方法能明显提高查错的效率。

1) 插入打印语句

在程序中插入暂时性的打印语句，是一种十分常见的查错技术。这类打印语句的作用主要是显示程序的中间结果或有关变量的内容。插入打印语句适用于任何高级语言编写的程序。

2) 设置断点

查错的基本技术之一，就是在程序的可疑区设置断点。每当程序执行到设置的断点时，就会暂停执行，以便纠错者观察变量内容和分析程序的运行状况。

3) 掩蔽部分程序

对可疑程序进行检查时，常常要让程序反复执行。如果整个程序较长，可疑区仅占其中的一小部分，则每次运行整个程序必将浪费许多时间和精力。在这种情况下，往往将不需要检查的程序掩蔽起来，只让可疑的部分程序反复运行。

掩蔽无关程序可使用下述方法：

(1) 在要掩蔽的语句行加上注释符。

(2) 把要掩蔽的程序段置入“常假”的选择结构中，使其无法执行。

1.2　软 件 过 程

1.2.1　RUP

RUP(Rational Unified Process)译为 Rational 统一过程，是 Rational 公司(现归属 IBM 公司)推出的一种软件过程产品。RUP 以统一建模语言(UML)描述软件开发过程，具体如下所示。

1. RUP 各个阶段

RUP 中的软件生命周期在时间上被分解为 4 个顺序的阶段，分别是初始阶段、细化阶段、构建阶段和交付阶段。每个阶段结束于一个主要的里程碑；每个阶段本质上是两个里程碑之间的时间跨度。在每个阶段的结尾执行一次评估以确定这个阶段的目标是否已经满足，如果评估结果满意，则允许项目进入下一个阶段。其中每个阶段又可以进一步分解迭代。一个迭代是一个完整的开发循环，产生一个可执行的产品版本，作为最终产品的一个子集，产品增量式地发展，从一个迭代过程到另一个迭代过程，直到成为最终的系统。

如图 1.1 所示，RUP 可用二维坐标来描述。横轴通过时间组织，是过程展开的生命周期特征，体现开发过程的动态结构，用来描述它的术语主要包括周期、阶段、迭代和里程碑；纵轴以内容来组织，体现开发过程的静态结构，用来描述它的术语主要包括活动、产物、工作者和工作流。

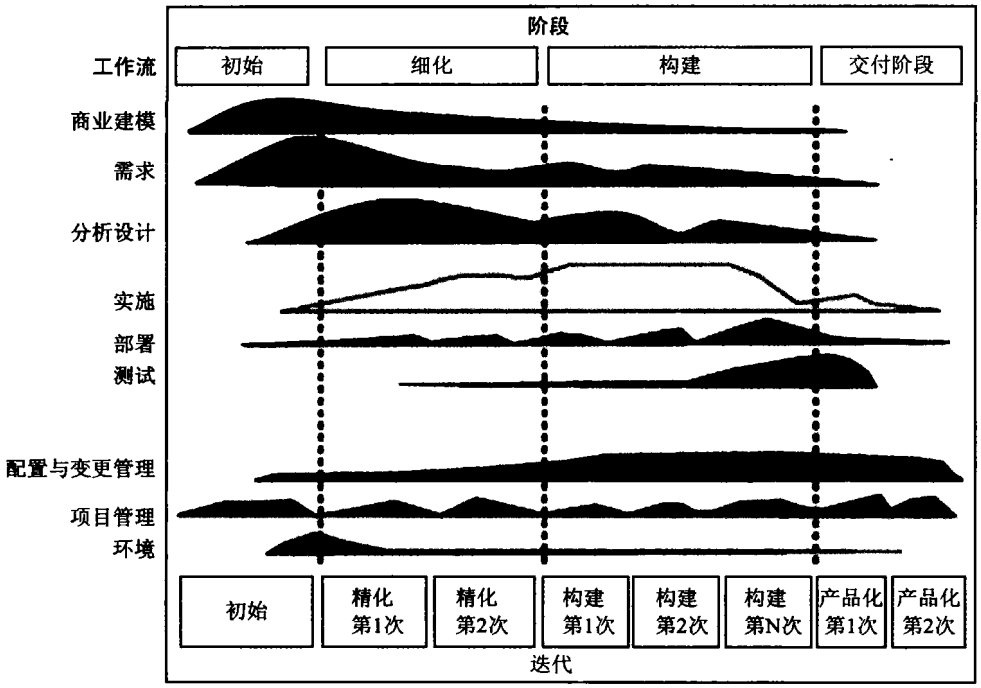

图 1.1 RUP 的过程图

下面依次介绍 RUP 软件生命周期的四个阶段。

1) *初始阶段*

初始阶段的目标是为系统建立商业案例并确定项目的边界。为了达到该目标必须识别所有与系统交互的外部实体，并在较高层次上定义交互的特性。这包括识别出所有用例并描述几个重要的用例。本阶段具有非常重要的意义，在这个阶段中所关注的是整个项目进行中的业务和需求方面的主要风险。对于建立在已有系统基础上的开发项目来讲，初始阶段可能很短。

在初始阶段应该取得如下成果：

(1) 蓝图文档，即关于项目的核心需求、关键特性、主约束的总体蓝图。

(2) 初始的用例模型(约占总体的 10%～20%)。

(3) 初始的项目术语表。

(4) 初始的商业案例，包括商业环境、验收标准等。

(5) 初始的风险评估。

(6) 项目计划。

(7) 一个或多个原型。

2) *细化阶段*

细化阶段的目标是分析问题领域，建立坚实的体系结构基础，制定项目计划，消除项目中风险最高的因素。为了达到该目标，必须在理解整个系统的基础上，对体系结构作出决策，包括其范围、主要功能和诸如性能等非功能需求，同时为项目建立支持环境，包括创建开发案例，创建模板、工作准则并准备工具。

细化阶段的成果包括：

(1) 用例模型。所有的用例和参与者都已被识别出，并完成大部分的用例描述。

(2) 补充非功能性要求以及与特定用例没有关联的需求。

(3) 软件体系结构的描述。

(4) 可执行的软件原型。

(5) 修订过的风险清单和商业案例。

(6) 整个项目的开发计划，该开发计划应体现迭代过程和每次迭代的评价标准。

(7) 更新的开发案例。

(8) 初步的用户手册。

3) 构建阶段

在构建阶段，组件和应用程序的其余功能被开发并集成为产品，所有的功能都被彻底地测试。从某种意义上说，构建阶段是一个制造过程，其重点为管理资源及控制运作，从而降低成本、加速进度和优化质量。

构建阶段的成果是可以交付给最终用户的产品，它至少包括：

(1) 集成于适当操作系统平台上的软件产品。

(2) 用户手册。

(3) 当前版本的描述。

4) 交付阶段

交付阶段的重点是确保软件对最终用户是可用的。交付阶段可以跨越几次迭代，包括为发布做准备的产品测试，基于用户反馈的少量的调整。在这一阶段，用户反馈应主要集中在产品调整、设置、安装和可用性问题。

2. RUP 核心工作流

RUP 共有 9 个核心工作流，分为 6 个核心过程工作流和 3 个核心支持工作流。6 个核心过程工作流可能使人想起传统瀑布模型中的几个阶段，但应注意迭代过程中的阶段是完全不同的，这些工作流在整个生命周期中被多次执行。

1) 商业建模

商业建模工作流描述了如何为新的目标组织开发一个构想，并基于这个构想在商业用例模型和商业对象模型中定义组织的过程、角色和责任。

2) 需求

需求工作流的目标是描述系统应该做什么，并使开发人员和用户就这一描述达成共识。为了达到该目标，要对需要的功能和约束进行提取、组织、文档化，最重要的是理解系统所解决问题的定义和范围。

3) 分析设计

分析设计工作流是将需求转化成系统的设计，为系统开发一个健壮的结构，使其与实现环境相匹配。设计活动以体系结构设计为中心，其结果是一个设计模型和一个可选的分析模型。设计模型是源代码的抽象，由设计类和一些描述组成。设计类被组织成具有良好接口的设计包和设计子系统，而描述则体现了类的对象如何协同工作实现用例的功能。

4) 实施

实施工作流的目的包括以层次化的子系统形式定义代码的组织结构，以组件的形式(源

文件、二进制文件、可执行文件)实现类和对象，将开发出的组件作为单元进行测试，集成单元使其成为可执行的系统。

5) 测试

测试可通过可靠性、功能性和系统性的三维模型来进行。测试工作流要验证对象间的交互作用，验证软件中所有组件的正确集成，检验所有的需求已被正确的实现，识别并确认缺陷在软件部署之前被提出并处理。RUP 提出的迭代方法是在整个项目中进行测试的，从而尽可能早地发现缺陷，从根本上降低了修改缺陷的成本。

6) 部署

部署工作流的目的是成功地生成版本并将软件分发给最终用户。部署工作流描述了那些与确保软件产品对最终用户具有可用性相关的活动，它包括：软件打包、生成软件本身以外的产品、安装软件、为用户提供帮助。在有些情况下，还可能包括计划和生成 beta 测试版、移植现有的软件和数据以及正式验收。

7) 配置和变更管理

配置和变更管理工作流描绘了如何在多个成员组成的项目中控制大量的软件产物。配置和变更管理工作流提供了准则来管理演化系统中的多个变体，跟踪软件创建过程中的版本。工作流描述了如何管理并行开发、分布式开发，如何自动化创建工程，同时也阐述了对产品修改的原因、时间、人员进行记录，依靠程序员主动、开放、高效的面对面交流来达成对需求、目标、设计实现的理解。

8) 项目管理

软件项目管理平衡各种可能产生冲突的目标，管理风险，克服各种约束并成功交付使用户满意的产品。其目标包括：为项目的管理提供框架，为计划、人员配备、执行和监控项目提供实用的准则，为管理风险提供框架等。

9) 环境

环境工作流的目的是向软件开发组织提供软件开发环境，包括过程和工具。环境工作流集中于配置项目过程中所需要的活动，同样也支持开发项目规范的活动，提供了逐步的指导手册并介绍了如何在组织中实现过程。

3. 用例驱动为核心

开发软件系统的目的是要为该软件系统的用户服务。因此，要创建一个成功的软件系统，必须明白此软件的用户需要什么。“用户”这个术语所指并不仅仅局限于人，还包括其它软件系统。一个用例就是系统向用户提供一个有价值的结果的某项功能。用例是软件的功能性需求。所有用例结合起来就构成了“用例模型”，该模型描述系统的全部功能。用例模型取代了系统的传统的功能规范说明。功能规范说明描述为 “需要该系统做什么？”，而用例驱动则是“需要该系统为每个用户做什么？”因此，用例模型是从用户的利益角度出发进行考虑，设计人员创建一系列用例模型，开发人员审查每个后续模型，以确保它们符合用例模型。测试人员将测试软件系统的实现，以确保实现模型中的组件正确实现了用例。这样，用例不仅启动了开发过程，而且与开发过程结合在一起。

1.2.2 敏捷过程

传统计划驱动的开发方法往往不能获得良好的效果，并且由于过分强调过程控制，因

此在开发过程中产生大量的文档，给开发人员带来很多额外的工作量，因此被称为重量级方法。这种方法使程序员花费大量的开发时间在开发文档的撰写和维护上，而真正花在代码上的时间较少，并且由于依赖过程控制，而不是程序员自我管理，导致开发过程管理复杂且低效，极大地阻碍了软件开发效率。

1. 敏捷开发简介

2001年是全球软件行业具有历史意义的一年，就在这一年，“敏捷联盟”成立了，并发表了对传统软件开发过程具有颠覆意义的《敏捷宣言》：“通过开发软件和帮助别人开发软件，我们找到了一些更好的开发软件的方式。通过这一工作，我们得出了这些价值：

- 个体和交互 胜过 过程和工具；
- 可以工作的软件 胜过 面面俱到的文档；
- 客户合作 胜过 合同谈判；
- 响应变化 胜过 遵循计划。

也就是说，尽管右边的项也有价值，但我们认为左边的项更有价值。”

从那以后，在敏捷过程价值观的指导下诞生了很多具体的开发过程，包括XP(极限编程)、FDD(功能驱动开发)、SCRUM开发过程等等，其中XP是影响最为广泛的一种开发过程。

2. 敏捷开发的特征

敏捷方法具有如下两个主要特征：

(1) 开发采用适应性方法，经过多次小型迭代开发过程逐步逼近实际需求，从而为客户提供实际需要的软件。这种开发方法的核心是小型发布，不断集成和严格回归测试。每一次小型发布都经过严格测试后集成到最终产品中，保证每一次小型发布都是经过测试的高质量的代码。在每一次小型发布后和客户沟通，得到客户反馈，不断修改，增加新的客户需要的功能，从而生产出符合客户需要的产品。敏捷开发过程以代码为核心，而不是以文档为核心。传统的以文档为主要输出的设计过程(需求分析、高层架构设计、概要设计、详细设计等)被大大压缩，以最快的速度进入代码的生产过程。设计中以简单为原则，不进行多余的设计活动，小组通过密切而有效率的交流达到对设计的统一理解和认识。文档作为交流工具的作用被弱化，文档作为管理监督的功能被取消。一切以代码为核心，代码经编写、测试、发布、重构，然后进入第二次迭代。

(2) 以人为本。传统计划驱动方法企图以文档、过程为核心，从而抹杀人的重要性。在敏捷方法里，程序员在软件开发中不再是单纯被管理的对象，而是开发的主体。所有的主要设计策略的制定、开发方法的选择、需求的确定都由程序员决定，以往开发过程中的对开发过程的严格控制和检测，以及对软件的各种测量等等都大大简化。

3. 敏捷开发的价值与局限

敏捷方法抛弃了繁琐的文档管理，依靠程序员主动、开放、频繁的面对面的高效交流来达成对需求、目标、设计实现的理解。敏捷方法抛弃了机械、严格的过程控制，依赖程序员和开发团队的高标准自我要求：严格的自律，团队合作精神，个人高度自觉的主动性，责任感。

敏捷方法的高效和高质量实际上是以程序员的高素质和开发团队的高度合作的开发文化为基础的。因此敏捷方法的实施前提是必须找到愿意并有能力实施敏捷方法的团队。XP

的创始人 Beck 也曾建议过有些情况是不适合采用 XP 方法的。比如，不能接受 XP 文化，团队规模过大，重构开销过大等。

1.3 软件质量

1.3.1 概述

软件质量具有多种定义。ANSI/IEEE Std 729—1983 定义软件质量为“与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体”。CMM 对质量的定义是：① 一个系统、组件或过程符合特定需求的程度；② 一个系统、组件或过程符合客户或用户的要求或期望的程度。M.J.Fisher 定义软件质量为“所有描述计算机软件优秀程度的特性的组合”。

因此，软件质量是一个复杂的多层面概念。

(1) 从用户角度出发，质量是对需求的满足，软件需求是度量软件质量的基础。

(2) 从软件产品角度出发，质量是软件的内在特征。

(3) 从软件开发过程出发，质量是对过程规范的符合。

软件质量框架是由“质量特性—质量子特性—度量因子”构成的 3 层结构模型，如图 1.2 所示。

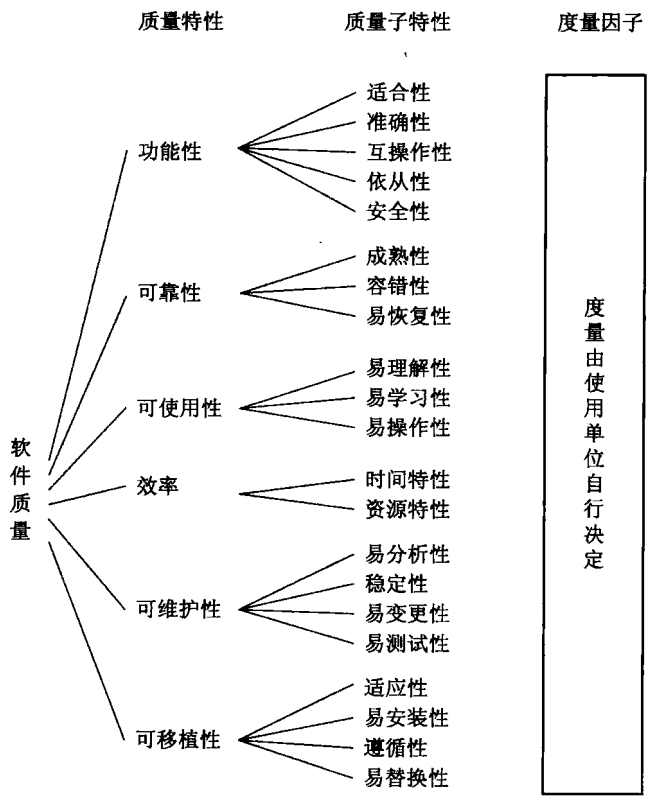

图 1.2 软件质量框架