Broadview®
www.broadview.com.cn

IBM

从菜鸟到测试架构师

一个测试工程师的成长日记

『从菜鸟到测试架构师』编委会 编著
孙磊 张明明 审校

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

内 容 简 介

本书以新人小艾为主角，从小艾的视角出发展开讨论。小艾是一位新入职的菜鸟测试工作者，他面对着测试这一未知的领域，在导师的悉心栽培和指导下走进了测试工作。在这个过程中，他经历了测试的各个方面和阶段，积攒下丰富的理论和实践经验，经过各种学习和历练，终于成长为一名测试架构师。小艾的成长经历贯穿了测试的各个领域，理论与实践并重，将测试以一个完整的体系展现给读者。

本书在内容编排上力求理论联系实际，每一个章节都通过小艾的学习工作过程展开讨论，通过实践进行理论提炼。在各个章节的结尾，以小艾学习笔记的方式总结论述该章节，帮助读者更好地理解和掌握测试理论及方法。

本书适合从事软件测试及软件质量管理的工程人员、企业 IT 主管，以及高校软件测试、软件质量及其他计算机相关专业的教师和学生阅读。

希望本书能够帮助读者形成测试理论及体系认识，帮助测试工作者更好地展开测试工作。

图书在版编目（CIP）数据

从菜鸟到测试架构师：一个测试工程师的成长日记 / 《从菜鸟到测试架构师》编委会编. —北京：电子工业出版社，2013.4
ISBN 978-7-121-19395-8

Ⅰ. ①从… Ⅱ. ①从… Ⅲ. ①软件－测试 Ⅳ.①TP311.5

中国版本图书馆 CIP 数据核字（2012）第 318317 号

策划编辑：刘 皎
责任编辑：高洪霞
印 刷：
装 订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×980 1/16 印张：24.75 字数：487 千字
印 次：2013 年 4 月第 1 次印刷
印 数：3000 册 定价：58.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

第 1 章

上班第一天，新人培训

1.1 测试专家的第一步

小艾是某名牌大学计算机科学专业硕士毕业生，这天是他离开校园走上工作岗位的第一天。他将成为大型外资 IT 公司 IBM 的软件测试工程师（Software Test Engineer），开始一段新的旅程。

1.1.1 我是菜鸟

在离开校园以前，小艾对将要从事的工作几乎一无所知。记得面试时被问及对测试的想法时，他的理解是，测试就是给产品挑错吧，目标应当是保证产品以高质量交付给用户。面试经理告诉他，其实测试是软件开发过程中必不可少的重要流程。在追求质量和效率的软件工程里，如何有效地对复杂的软件半成品进行测试，其实有许多问题值得工程师们去思考和探索。软件测试工程师的工作将很有趣、充满挑战。于是，对新事物充满好奇心的小艾欣然接受了软件测试工程师这个职位邀请，充满期待地走进这个他完全不了解的神秘领域。

产品开发组的同事，包括组长和老员工，对小艾这只菜鸟照顾周到，一会儿工夫他就把入职的流程办妥，工作的机器也准备就绪。坐在新的座位，小艾开始憧憬自己的新工作。可是测试却是一张陌生的面孔，让他有点无所适从。于是，小艾找到公司给他安排的“导师”凯文，希望凯文能帮他排解困惑。凯文是测试组组长，一位具有丰富工作经验的老员工。未来，就从这一刻开始向小艾展露出微笑。

“凯文，我对将要从事的工作一无所知。你能告诉我测试工作都包含些什么内容吗？我们应该如何做测试？什么时候可以真正开始工作？”

凯文对小艾的问题一点儿也不陌生，这些问题不正是几年前他入职时的困惑吗？“小艾，别着急，请慢慢听我说。我也像你一样，是从菜鸟一步一步成长起来的……”

经过与凯文的谈话，小艾心中的一团迷雾逐渐消除了。

原来，在大型软件开发团队中，测试被分成很多种类和步骤，每种测试有针对性地模拟使用测试对象的场景，并试图找出测试对象的潜在问题和缺陷（Bug）。在确定原因后，制定严谨完善的解决方案并根据方案修复缺陷。测试其实是发现并解决问题的过程，而其目标则是让软件产品以尽可能高的质量交付给客户，使软件产品中存在的问题尽可能少，这样，软件的用户可以得到最完美的使用体验。

除了小型项目，进行完全（各种输入和前提条件的组合）的测试是不可行的。可行的方法是运用风险分析和不同系统功能的测试优先级，来确定测试的关注点，从而替代穷尽测试。软件开发本身是追求产出和投入比的工程性过程。因此，考虑测试的内容和方式时，都应当以高产出投入比为最终目标，最大化地利用现有资源排除潜在的问题。

小艾听说过风险控制，在软件测试过程中，风险控制是通过专业有效的方法实现的。测试团队由许多个测试分队组成，每个分队的测试任务和方法都具有高度的针对性。

小艾回想，在学校的时候，他曾经参加过软件工程课程的项目实训。在项目中，测试很简单，其目的仅仅是验证开发的功能点是否正确并与设计一致。测试是在所有功能开发完毕后才开始的。当时项目规模很小，从计划的时候开始，大家就没有仔细地考虑过怎样做测试。由于项目组人数很少，在功能开发阶段大家也无暇顾及测试，而是到了功能开发已经完成后，大家才匆忙地花些时间测试。当然，这种测试非常简陋，没有计划细节，方向也不清晰，测试过程中的所有流程都手工操作一遍。发现问题则随时修改代码，如果修改后流程能走通，就认为测试已经通过了。

通过凯文对测试的类型和当今流行的开发模式的介绍，小艾发现测试远不是从前软件

工程项目实训测试那般随便和简单。软件测试是一个严谨、全面且有条理的过程。这个过程中包含了多种测试类型，每种测试类型都有针对性地验证软件，发现相应的问题。测试就像河流中一张精心编织的网，软件的功能和流程就像河流中的鱼，要通过这张网的鱼必须足够优秀才能最终存活。正是这种“优胜劣汰”的思想，保证软件只有通过了测试这张网才得以与用户见面。

凯文娓娓道来，小艾对 IBM 的测试方法有了初步的了解：原来测试的种类可以如此多种多样。

单元测试是和开发最接近的一种测试。开发人员编写单元测试用例并执行，验证单元模块是否得出预期的结果。在敏捷开发模式中，有一种流行的开发模式叫做测试驱动开发（Test Driven Development，TDD）。测试驱动开发的核心就是把单元测试用例先做好，功能开发以通过相应的单元测试用例为目标。单元测试是粒度最小的软件测试，小粒度能保证复杂系统中的每个“螺丝钉”都质量合格。通过了单元测试的代码才可以继承到系统中，进行进一步测试。

功能测试是通过黑盒子模式发现代码集成后存在的功能问题的测试。顾名思义，功能测试关注的重点是系统的功能。通过执行自动或手动的测试用例，可以验证相应的功能点是否正确。只要测试用例设计完整，功能测试的网能把最终用户可能遇见的功能问题都“提前”发现并解决。可以说，功能测试保证了提供给用户的是产品而不是一堆垃圾。单元测试和功能测试的区别主要是粒度的不同。单元测试关注的是一个最小的代码片段，如一个类或接口，而功能测试关注的是一个完整的业务功能。

功能测试通过后，性能测试就随之开始了。性能测试是重点验证软件的非功能性需求的测试。企业级软件通常用于应对复杂苛刻的用户场景。在软件设计和安装的过程中，有许多细节能提供软件的性能，包括吞吐率、稳定性、可靠性等。性能测试通过自动化的方法模拟真实用户并发访问的场景，以验证系统的性能指标或发现其性能瓶颈。性能缺陷常常不是显而易见的，有时候得通过复杂的场景模拟方可重现问题。由于性能问题的复杂性，要定位问题的原因也是一个艺术的过程。通过了性能测试的软件系统从根本上保证了用户体验和长远利益。

正式版本发布前，测试组还要组织成品测试（GMV），测试软件的安装、部署、发布等情况，确保软件能最终顺利地安装到用户的环境中。通过集成测试后，软件的质量有了进一步的保障。

软件正式版本发布以后，根据用户的反馈，产品组需要发布多个小版本。发布以前，

当然也要有针对性地测试小版本的功能、性能，以及和正式版本的兼容性。小版本的开发和测试周期比正式版本短得多，因此对测试团队的项目管理要求更高。

“在 IBM，为了保证软件质量而进行的测试是全面而苛刻的。”凯文说，“等你完成了新员工培训并开始接触项目时，你将有更多机会从方法到实践了解我们的软件测试。”

1.1.2 苦练基本功

小艾所在开发团队所负责的产品是电子商务平台。电子商务平台是一个功能强大的企业级应用，它支持多种计算机平台。要成为一名合格的测试工程师，首要的就是熟练掌握基本功。所谓基本功，指的是作为任何开发团队的一员，都应该掌握的基本知识和技能。

小艾发现上大学时学习的专业课还是非常有用的，诸如数据结构与程序设计、计算机组成原理、操作系统原理等。但这些课程大多只注重理论，缺少在真实环境中实践的经验。公司的软件开发通常都有比较成熟的基础设施，对于新人来说，了解开发平台和方式也是锻炼基本功的一部分。

操作系统是平台的基础。IBM 的产品通常都支持多种流行的操作系统，如 Windows，UNIX，Linux 等；为了满足更大规模的应用，产品还能提供对大型机的支持。小艾最熟悉的操作系统是 Windows，在学校和日常生活中基本都用这个系统，而对 UNIX 和 Linux 却是一知半解。在开发组的数据库里，小艾学习了大量关于 UNIX/Linux 基础的材料。在学习的过程中，他有机会操作各种平台的机器，这样在实践中学习效果是最好的。几周下来，小艾已经掌握了 UNIX/Linux 的系统管理知识和编程基础。他发现，在简洁的黑色文本界面背后，隐藏着超乎想象的强大计算能力。

学习了操作系统基本知识后，小艾满怀兴奋地找凯文：“我是不是可以开始测试产品了？”凯文摇头说：“离测试还远着呢！不过你可以接着实践测试环境搭建了。”所谓测试环境搭建，是指在操作系统基础上，安装测试需要的应用程序，并部署测试的功能代码，准备测试数据，建立起一个可供测试的环境。电子商务系统是基于中间件的网络应用程序，因此，必须从安装服务器程序开始。一个完整的企业级网络应用程序，通常需要集成多个服务器，包括网络服务器、应用服务器、数据库服务器等。坐在机器前，小艾发现搭建测试环境是一个艰巨的过程。

测试环境的搭建从安装网络服务器开始，接着是数据库服务器和应用服务器的安装。网络应用程序作为企业级应用，部署在应用服务器上。面对一系列的设置步骤，小艾感到

头晕目眩。一连几天，小艾都在直接和服务器打交道。在 UNIX/Linux 上搭建测试环境，用户的权限管理是关键的部分。因为涉及许多手动操作的部分，一时疏忽引起的用户权限错误会导致搭建失败。小艾自问还是很有耐心的，这次也被测试环境安装折腾得“体无完肤”。经过凯文的多次“指点迷津”，在安装配置好服务器以后，电子商务应用程序也安装好了。

小技巧：正确的流程和步骤一定要及时记录。有了流程和步骤的指引，可以避免大量不必要的重复劳动。

这也仅仅相当于一个新建的购物商城做好了“硬装修”，商城还是空空的，还不足以接受业务流程。根据测试用例的要求，需要安装并激活业务模块。网上商城的经营模式、页面的样式、能够提供的功能等特性，都是通过激活业务模块并配置数据等步骤决定的。

激活了业务模块以后，工程师的测试才可以开始。

领教了测试环境安装的烦琐步骤，小艾想，要是测试工程师不得不耗费大量资源兼顾测试环境的安装和配置，那么将难以保障软件测试的质量。从资源利用优化的角度而言，这似乎也不是很好的方式。

带着疑惑，小艾找到了凯文，“环境安装耗时费劲，测试人员必须每次都从头到尾安装测试环境吗？”

凯文说：“看来你对环境安装的难度有了很深的体会啊。你的顾虑项目组很早以前已经考虑到了。我们发现，更细的分工是提高效率的源泉。你应当看看我们用什么样的方式实现了测试平台的高效搭建。”

凯文说，其实许多新同事也有着和小艾一样的疑惑，解决环境安装问题的方式是执行构建测试（Build Verification Testing，BVT）。

的确，构建测试是个烦琐、重复性的过程。为了有效搭建环境，避免人为原因的错误，采取的策略是最大程度上地使构建过程自动化。因为环境的原型和步骤基本上是类同的，这种条件很适合自动化。于是工程师们使用构建了参数化的脚本、响应文件等构建测试的元素，有了这些元素，构建过程不再每一步都需要人工干预，出现错误的可能性有效降低。当然，由于平台本身的复杂性，自动化元素的构建由构建组专门维护。构建组把这个过程称为构建测试。通过测试验证环境安装的正确性。

软件的构建（Build）本身是依赖于 Java 的，因此没有平台依赖性。但是，Java 虚拟机是安装在不同的操作系统中的，于是测试环境的安装就有了平台依赖性。构建完整的测

试应该包括对多种平台的支持。不同操作系统平台结构通常很不一样，需要提供有针对性的自动化构建程序。构建完成后，构建组还必须完成对所有支持的平台的构建测试。

有了自动化的构建程序和构建测试的步骤，可以保证测试环境正确和顺利地安装。但是，每次安装还是得花时间的。熟练的工程师使用构建程序在某个平台构建一个测试环境得花大半天时间。小艾从兴奋转为沮丧：每次安装半天时间的成本并不小啊，大家测试环境的资源耗费问题还没有解决。

幸运的是，开发实验室利用虚拟技术构建了基于不同平台的测试镜像。有了虚拟技术，时间和步骤也是“可复制”的。由于测试环境必须支持多平台，使用自动化方式搭建第一个测试平台的时间是不可节省的；但是，第二个、第三个测试环境的搭建时间确实可以节省。奥秘就在于虚拟技术。成功搭建一套测试环境后，就可以把这个环境保存成镜像（Image）。以后任何时间需要再使用这套环境，不必重新安装，仅需要把镜像恢复，并替换必要的机器信息即可。虚拟技术被多个平台支持，包括 AIX、Windows、UNIX/Linux。用于恢复镜像的硬件环境既可以是实际存在的，也可以是虚拟的。

虽然没有仔细了解过虚拟技术，但小艾在学校的时候使用过 Ghost 克隆软件。凯文说：“虚拟技术的原理和 Ghost 有相似的地方，随着使用的深入，你会对虚拟技术有更多的认识。”

对测试环境安装有了初步的了解，作为菜鸟，小艾接着需要知道的是中间件技术。要知道，功能强大的电子商务平台是建立在 IBM 的 WebSphere 中间件基础上的。凯文开始给小艾介绍一些基于中间件的应用服务的基础内容。

中间件（Middleware）是提供系统软件和应用软件之间连接的软件，以便于各种部件之间的沟通，特别是应用软件对于系统软件的集中的逻辑。中间件技术在现代信息技术应用框架，如 Web 服务（Web Service）、面向服务的体系结构（Service Oriented Architecture，SOA）等应用中应用得比较广泛。中间件不提供具体的功能，但它却是系统中各个部件有机连接的桥梁。中间件可以提供对外围服务器，包括数据库服务器、应用服务器、Web 服务器等的支持和管理。中间件技术建立在对应用软件部分常用功能的抽象上，将常用且重要的过程调用、分布式组件、消息队列、事务、安全、连接器、商业流程、网络并发、HTTP 服务器、Web 服务等功能集于一身或者分别在不同品牌的不同产品中分别完成。

接下来的日子里，小艾开始研究 WebSphere 中间件提供的功能。经过一段时间的学习，他掌握了通过应用服务器对应用程序进行管理和监控的方法。这部分知识对于测试中发现和解决问题起着关键的作用。

经过基本能力的训练，小艾基本上已经达到了进入项目组的要求。然而，对于项目如何运作，如何确保项目顺利达到预期结果，小艾却还是一知半解。接着，小艾在凯文的指导下，认识了敏捷项目管理的基本知识。

对于敏捷开发（Agile Development）的定义，工业界其实还没有标准的定义，而相关的描述倒是五花八门，各种定义也出现在出版物或网络博客中。缺乏标准定义，其实是因为敏捷开发的实现方式非常多样化。我们可以容易地找到关于敏捷的方式、方法、实践及技术等的描述。在 IBM 的软件开发和测试中，团队使用了多种流行的敏捷开发方式进行项目管理。使用敏捷项目管理的目的并不复杂，是为了提高开发效率，激发团队的积极性并尽可能降低项目失败的风险。

提到敏捷开发，会把某种开发方式作为“非敏捷”方式来对比，而这种开发方式通常会是传统的瀑布开发模型。在瀑布开发模型中，整个系统的开发被划分成需求分析、设计、实现、测试、集成和维护等阶段。这种划分本质上是把不同性质的项目内容分隔到不同的阶段，而某个阶段则专注地进行某种任务。专注在许多情况下带来了高质量，单一流程的划分却很容易带来资源浪费和失败风险的增加。如果在一个阶段，项目组只完成一组相同性质的任务，那么，团队中其他无关的人员在这段时间里就无事可做了。项目的成果必须到最后阶段才完成，中间任何步骤出现差错都有可能导致项目全盘失败，这样的项目风险是很高的。

敏捷开发从根本上避免了瀑布模型的弱点，它有两个核心点——迭代开发（Iterative Development）和增量开发（Incremental Development）。

迭代开发是一种“重复时序安排”的开发方式。迭代开发把一个完整的瀑布模型开发流程分成多个迭代，每个迭代可以看做独立的开发过程，其中包含了项目的主要步骤，如设计、开发和测试等。把完整过程分成多个持续时间较短的迭代，其好处是生产的周期变短了，每个完整的周期都会产出相应的产品，这种方式有利于在完整项目开发的过程中跟踪和控制开发进度及产品质量。

增量开发用的则是一种“分段完成”的策略。在增量开发模式中，系统中不同的部分被安排在多个阶段完成，各个部分完成后再集成到系统中。

在敏捷开发模式中，迭代开发和增量开发的策略通常会被同时使用，并统称为迭代开发，迭代开发框架如图 1-1 所示。

增量地实现系统的思想是迭代开发的基础。项目成员通过不断学习和总结，使开发效

率不断提高，同时避免在后期的迭代中重犯某些错误。因此增量开发对于团队的进步也很有好处。

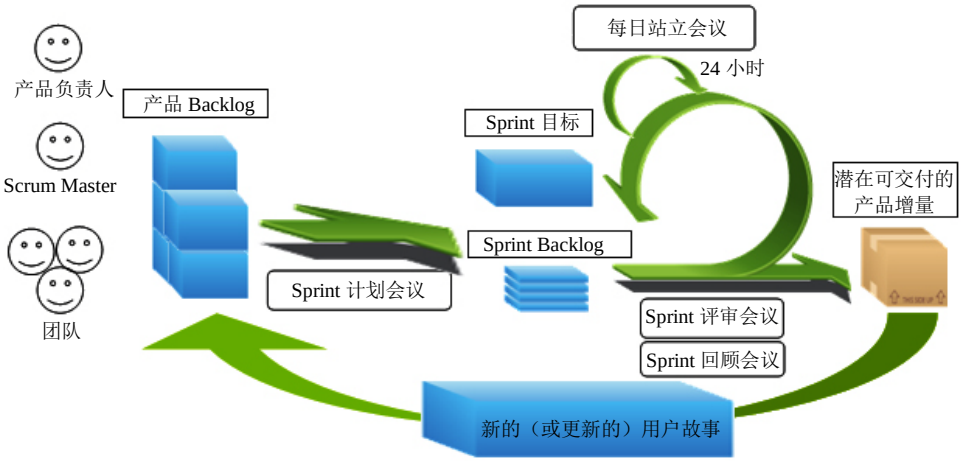

图 1-1　迭代开发框架示意图

因为时间周期缩短，和传统瀑布模式相比，迭代开发的进度会紧凑很多。项目组必须定期开会确保项目如期进行。项目的周期称为冲刺（Sprint），每个 Sprint 通常持续 2～6 周时间。在 Sprint 开始之前，项目组会进行 Sprint 计划会议，安排当前开发周期的任务；而在 Sprint 结束时，项目组举行 Sprint 评审会议，对这个周期的交付件进行评审核实；评审会议结束后，项目组还需要进行简明扼要的 Sprint 回顾会议，回顾这个开发周期中做得好的或需要提高的方面，以便在下一周期提高开发效率。

在每个 Sprint 中，为了确保开发进度，项目组还需要举行周期的会议（通常是每天），确定各个小组成员更新已经完成的任务、即将开始的任务及使进度受阻的问题，并通过讨论得出解决问题的方案。这种周期的会议叫做每日站立会议（Daily Scrum Meeting），由 Scrum Master 主持。

小艾最讨厌开会了，他认为开会浪费时间却常常没什么有效的结果。但凯文告诉他，每日站立会议有个重要原则，为了避免浪费时间，每个成员的汇报时间必须严格限制，会议的持续时间只能在 15 到 20 分钟。更细节的问题都放在会后讨论。这样的安排优势是明显的，既能让所有成员了解项目进度，又不耽误所有人的时间。

听完凯文的介绍，小艾对敏捷开发条件下的项目运作有了最基本的了解。对于敏捷开发范畴而言，这仅仅是冰山的一角。凯文还列举了许多开发的方法，包括测试驱动开发（Test

Driven Development）、极限编程（Extreme Programming）、开发统一过程（Open Unified Process）、结对开发（Pair Development）等。不同的项目根据实际情况会使用不同的具体开发方式。然而，有两样东西是所有这些方法的基础——计划和流程。

计划定义的是做什么（What）和什么时候做（When）。在项目或迭代的初期，项目负责人要根据需求定义项目的计划，计划的内容包括要执行的任务、任务依赖条件、负责人选、执行任务的时间等。对于测试而言，测试计划的制订非常重要。测试计划详细描述了测试的环境、场景、执行要点、依赖等内容。项目执行完全根据计划实施，因此，好的计划是项目成功的基础。

流程是项目成功的保障，它定义了怎么做（How）。无论是开发还是测试，每个步骤都有标准的流程。在软件测试中，测试环境的准备有标准流程，没有按标准流程安装的环境就可能有潜在的错误；执行测试的步骤有标准控制流程，没有按流程执行的测试结果是无效的。完全按照标准流程完成的测试，如果存在失败的测试用例，我们就可以很直接地从产品本身找原因，而不用怀疑是错误的执行导致了失败。流程控制每个步骤的正确完成，有了流程控制，软件的质量才得以保证。

作为菜鸟的小艾对测试工程师的基本功掌握得差不多了，他明白，这些基本功需要在实践锻炼中不断加深理解，才能得心应手，融会贯通。

1.1.3　培养专业技能

这天中午，小艾经过凯文的座位，发现他正在专心致志地阅读一本技术书籍。小艾好奇地问凯文，究竟是什么书那么有趣。面对小艾的一脸好奇，凯文细致地解释自己学习的内容：“我在学习脚本语言编程技术。作为测试工程师，开发技术对我们而言也同样重要。为了更好地进行测试，得学点脚本编程知识。”

“测试工程师也必须掌握开发技术吗？还有哪些技能是测试工程师应该掌握的呢？”小艾有些疑惑。

“的确，我们的工作应该专注在测试上，但这并不代表我们不需要掌握开发技术。恰恰相反，开发技术是测试工程师应该掌握的基本专业技能之一。先给你解释什么是专业技能，再介绍我们应该掌握哪些专业技能吧。”

在软件开发团队中，作为软件测试工程师，为了完成测试任务，有一些技能是必须掌握的，我们把这部分技能定位为专业技能。在开发团队里的“专业”，与高等院校中的

“专业”不同。开发团队的专业，强调的是长时期从事的行为作业规范，规范保证了产品的质量。

一名专业的测试工程师，应该把开发技能作为其技能体系的基础。测试人员虽然不需要编写功能代码，但是测试人员在测试过程中需要完成测试代码的编写。掌握开发技能，有利于理解功能实现的方法和逻辑。我们知道，测试就像一把手术刀，务求一击即中要害，不让存在问题的地方漏网。测试工程师掌握了开发方法，基于对开发的了解，更容易设计出有效的测试场景和用例。例如，使用 Java EE 开发的应用程序，程序的实现逻辑很有可能影响垃圾回收的效率，那么设计测试用例时，应当重点测试功能模块对垃圾回收的影响。又如，使用 JavaScript 实现的前端页面，在不同的浏览器中的显示状况可能有明显差异，有些脚本是针对特定的浏览器开发的。了解 JavaScript 的这种特性及开发的逻辑后，设计测试用例时就应当在不同浏览器中针对可能有问题的脚本进行测试。测试工程师应该掌握数据结构和算法设计、设计模式和体系结构，了解不同的开发语言和平台的差异。

开发技能不仅包括设计和实现功能的技术，还包括发布和部署代码、配置环境的技术。软件产品的测试通常具有严格的版本限制，小版本的差异也完全有可能得出不一样的测试结果。测试人员了解代码的发布和部署，能够评估新代码的影响范围，并根据评估适当地调整测试的内容。当然，掌握代码发布和部署以后，至少不会糊里糊涂地就开始测试，而能够在确认代码的发布和版本都没有问题之后再行动。

开发团队使用了版本控制工具来管理程序的版本，了解新代码如何进入到软件版本中，对测试人员而言也很有用。在对原始代码进行修改前，程序员或测试人员需要创建一个“缺陷”，“缺陷”意味着系统有需要更新的地方。缺陷创建后，系统会生成唯一的缺陷号码，用于跟踪代码的更新状态。无论是因为加入新功能还是测试失败而创建的缺陷，都通过统一的平台进行管理。有了这种版本控制方法，对代码的任何改动都记录在案，任何引起新问题的线索都得以保留，系统也可以在需要时回滚到任何较早的状态。了解“缺陷”模式的运作，有助于测试人员执行测试或回归，验证缺陷是否修复。

鉴于测试工程师的专职工作是完成软件测试，因此测试专业技能是测试工程师技能体系的核心。

小知识：软件测试，从宏观角度而言，是指针对被测试的产品或服务进行的一系列关于软件质量的调查，软件测试结果对软件的拥有者（Product Owner）负责。软件测试还从一种独立的视角为业务运作提供客观评估，这种评估包括软件的质量达标程度及因为某种相应的实现方式而存在的风险等。

测试得到的是一种相对客观的结果，通过事实和数据对软件的质量进行定义，为软件的拥有者提供决策的参考。测试通过执行程序或应用的方式，达到发现存在的错误或缺陷的目标。至于测试，使用的方法则是多种多样的。理论上，任何方法，只要发现的错误或缺陷是确实存在的，都是可行的测试方法。但是在项目实践中，我们不可能把所有可能的方式都用上，而只能采取最有效和可控的方法。有效，是指这种方法有效模拟真实的应用，并有效地暴露潜在的问题；可控，指的是使用方法有明确的步骤，通过相应的步骤可以使暴露的问题重现。

真正的测试中，有效可控的测试方法通常有两类，分别是白盒测试（White-box Testing）和黑盒测试（Black-box Testing）。

白盒测试指测试人员可以直接访问内部数据结果、算法及其代码实现的测试，常见的方法包括编程应用接口测试、代码覆盖率测试、缺陷注入方法等。通过调用应用程序的公有或私有接口，验证返回内容的正确性的方式，是很常用的白盒测试方法。通常一个测试用例可以用于验证一个被测的接口。如果需要验证一个代码分支，还可以把分支需要使用的多个接口调用放在一个用例中。

代码覆盖率测试是检验代码是否满足指定覆盖率的测试。例如，可以设计一个测试，通过改变输入条件，使程序中所有代码行都被执行至少一次，并检验输出是否符合预期。覆盖率测试在一般条件下，最大限度地覆盖代码，评估代码的整体质量。缺陷注入关注代码在错误和临界条件的表现。错误和临界条件在所有输入条件中只占小部分，但这部分输入却非常关键，而且存在问题的可能性较大。测试涵盖了错误和临界条件，就能保证代码的健壮性。健壮的程序不仅能处理期望的输入，对于“不速之客”，同样能够从容应对。

白盒测试用最直接的方式，从根源上发现程序中的缺陷。然而，底层代码的逻辑往往错综复杂，分支繁多，白盒测试的方法需要测试人员对实现的细节比较了解，方可设计出有效的测试用例。而且，因为时间等条件的限制，要覆盖所有分支的所有条件，简直是个“不可能的任务”。于是便有了黑盒测试，通过触发业务相关的功能点，检验集成条件下系统的正确性。虽然不能期待黑盒测试方法能覆盖更多的代码分支，但这些方法针对的都是和实际系统应用相关的分支，因此黑盒测试对于评估系统是否达到需求是至关重要的。理论上，只要黑盒测试的用例设计得足够细致，测试能发现所有应用中可能存在的问题。当然，因为进行黑盒测试的时候系统是集成的，所以发现问题后，需要“打开黑盒子”，用额外的工作定位问题的确切原因。相对而言，白盒测试发现问题后，定位原因的过程会简单得多。

综合应用白盒测试和黑盒测试，可以达到有效测试系统的目的。

定义了测试方法，测试工程师应该明确的下一个内容是测试的执行。大型电子商务系统的业务逻辑非常复杂，集成测试往往需要涉及多个功能模块。如何执行测试用例问题，实际上是如何提高工作效率的问题。

对于界面操作、简单的功能验证用例，通常可以使用直接手工操作的测试方式；但是对于大多数逻辑复杂或者有特殊要求的测试用例来说，自动化测试是主要的测试执行方法。

手工操作的优势是方便灵活，只需有明确的测试用例作为指导便能执行，不需要花额外的时间准备完备的自动化测试材料。我们甚至可以通过手工操作的方式执行随机测试（Adhoc Testing）。探索式测试不使用可识别的测试用例，而采用相对“随机”的方式验证某些功能点的正确性。但是手工操作的重复开销是测试策略的设计人员必须重视的。由于测试步骤必须通过手工的方式执行，重复执行测试用例的时间与资源开销和第一次执行基本上没有任何区别。对于一线测试工程师而言，不停地重复手工测试用例是个无法摆脱的梦魇，至少热衷于接受新挑战的小艾这么认定。

自动化测试则能弥补手工测试在重复开销方面的不足。自动化测试，顾名思义，就是测试过程并不需要人工干预的测试方式。其优点是一旦测试的材料（包括自动化工具、自动化测试环境和测试脚本）准备好，测试可以在无须人工干预或在有限度的人工干预的条件下重复执行。对于流程复杂的测试场景，自动化测试在节省重复执行的资源的同时还能很好地控制测试质量和效率。当然，自动化测试对测试团队的组织和技术要求更高。要进行自动化测试，首先，测试团队必须有一套完备的测试工具集；其次，测试人员需要掌握测试工具的使用方法，包括如何编写自动化测试代码，如何执行并收集结果等；最后，对测试资源的维护也有更高的要求。自动化测试脚本和对应的测试环境应当严格归档保存，以备日后查询和重复使用。

作为一个成熟高效的测试团队，手工测试和自动化测试都是不可或缺的测试执行方法。两者优势互补，可以有效保障软件产品的质量。

在产品开发过程中，特别在使用敏捷开发模式的项目中，新功能的完成大多是分阶段的。功能点在不同的迭代中陆续发布，同时，在新的发布中还会包含对老版本的问题修复。作为测试，除了需要测试新发布的内容，还必须验证已经存在的功能是否正确，存在的问题是否已经修复。这是测试执行中很重要的一环——回归测试。就理论而言，只要有新的软件版本发布，对执行所有测试用例的回归测试是必需的，因为只有这样，才能从事实上保证所有功能在最新版本中的正确性。随着开发完成的功能越来越多，回归测试中要执

行的测试用例也随之增加。自动化测试在重复执行方面的优势正好能满足回归测试的这种要求。

作为测试工程师，掌握了测试执行方法，可以认为已经掌握基本的专业技能了。然而测试工程师专业技能的涵盖范围其实非常广，因为高质量的测试要求工程师掌握更多的技术，包括架构设计、软件开发技术等。更好地掌握这些专业技术，目的是更好地服务于测试，测试的目的则是发现和排除软件中存在的问题。

“发现、解决问题其实是一种艺术。”凯文语重心长地指导小艾，“等你熟练掌握基本的测试专业技能的时候，我们还会从更深入的层次探讨测试的核心价值和技术。”

1.2　开发团队做的远不仅是开发

加入团队已经有一段时间，小艾逐渐发现，在工作中，似乎每个人都在做不一样的事情，而每个人在团队中的位置都显得不可或缺。一个疑问从小艾的心中冒出来：为什么团队中大家似乎都在做不同的事情？开发团队有多少种角色？测试团队又有多少种角色？测试专家的核心价值究竟在哪里？

凯文的解答从讲述团队分工开始。

细致的分工能提高效率的道理已经在许多需要专门技能的领域得到充分证明。在软件开发领域，分工的作用同样突出。分工的结果是，团队的成员根据分工的结果担当不同的角色，在其位，谋其职。

在一个软件开发团队中，狭义上从事开发任务的成员其实只占其中一部分；开发以外的其他任务，如项目管理、设计、测试等在软件开发过程中同样非常重要。作为测试团队中的一员，小艾有必要了解更多关于测试团队中角色分工的内容。测试在软件开发过程中的重要性，在许多人心目中并不那么突出，但实际上，软件的好坏，在很大程度上是由测试决定的。

1.2.1　术业有专攻

在知识密集型的社会，人是最关键的资源。软件开发的产出和从业人员的技能密不可分。然而，随着软件系统的复杂度越来越高，驾驭软件开发所需的人力资源远远超出单一

个体的可承受限度。从这个角度而言，分工是解决个人控制力有限的唯一方法。从经验和效率的角度看，一个团队中的个体在技术和经验上各有千秋，分工能更有效地发挥人的长处。

在一个成熟的开发团队中，细致严密的分工使团队能以更高的效率运作。分工有助于提高效能的奥秘在于，绝大多数情况下，专注的人在效率上要高于注意力分散的人。细致的分工有利于凝聚人的注意力，提高熟练程度的同时减少切换带来的开销。在软件开发团队中，特别是使用了敏捷开发以后，团队的角色分工是非常明确的，每个成员根据各自的角色，专注于开发过程中的相应任务。

电子商务平台是一个涉及多种业务场景的复杂软件系统。软件开发任务的顺利完成，必须依赖于一个角色完善的团队的紧密合作。那么团队中究竟都需要哪些角色分工呢？

小艾所在的开发团队使用Scrum模式敏捷开发，是已经被证实可以提高开发效率的开发方式。按照敏捷开发的团队角色划分方式，Scrum团队的核心角色主要包括产品负责人（Product Owner）、Scrum Master及团队成员（Team）。在Scrum团队的外围还包括客户（Stakeholder）、经理（Manager）等角色[1]。

团队成员主要负责产品的具体开发。每个团队成员有具体的分工，如架构师、开发人员、测试人员、文档设计人员等。团队成员组成执行团队，这是个自组织、自定向和跨功能的执行团队。执行团队通过直接的行动推进项目的进度，以达到计划的目标。执行团队一般由 5～9 个各方面的专家组成，团队的组织结构文档贯穿整个开发周期。团队成员都是开发周期里各个领域的专家，他们使用专业的技能完成开发任务。对每种角色，通常都有一套技能集（Skill Set），通过技能集，可以定义一个角色应该具备的能力，反过来也能够判定一个员工是否具备担当某个角色的专业素质。

架构师是对软件开发过程的各个领域都具备一定专业技能的人员，主要任务是把软件开发的需求转化为可以实现的抽象设计和具体设计，并完成相应的设计文档。同时，架构师还需要把业务化的需求转化为技术化的功能性需求及非功能性需求。架构师需要参与软件开发各个阶段，也作为审核人员对详细设计和开发计划进行审查。架构师的技能特点是，具有更高视角，对技术的发展方向能够有全局的把握，对业务也有深刻的认识。可以说，架构师的知识体系兼顾了深度和广度。

开发人员的职责是根据抽象设计和高层次的具体设计进行更细化的具体设计，并按照

1　参考 12.1.2 节关于 Scrum 团队角色的详细描述。

设计完成编码实现及单元测试任务。在测试阶段，开发人员还有完成问题分析和解决缺陷的任务。由于软件的代码实现都是由开发人员完成的，因此开发人员的开发技能与软件是否以高质量完成有重要的关系。开发人员具有把宏观任务抽象化和把抽象概念具体化的能力，能够以微观的视角完成功能细节的开发。作为开发人员，卓越的理解能力和编码能力是必需的。

测试人员的任务是根据软件设计文档编写测试计划，并按照测试计划对软件进行测试。要完成的测试种类是根据需求定义的，在复杂软件系统的开发团队中，通常包括多种类型的测试人员，分别对各自的领域进行有针对性的测试。测试人员从另一个角度促进软件的开发过程，其工作的重点是发现问题和解决问题，因此，这对测试人员的洞察能力和分析能力提出非常高的要求。测试人员除了掌握测试方法学以外，还需要具有良好的抽象思维能力和逻辑分析能力。

文档设计人员的任务是根据需求文档和设计文档，设计编写交付给用户的说明文档和使用手册。文档对于软件产品的重要作用是不言而喻的，完备的文档是成熟软件产品必须具备的交付成果。一套好的文档在很大程度上决定了软件产品能否顺利被最终用户接受。文档对于开发团队也有重要的意义。清晰细致的技术文档对于产品维护的帮助也是不言而喻的。为了完成文档的设计，对文档设计人员的技能要求丝毫不能马虎。文档设计人员需要具有突出的表达能力和叙述能力，善于把抽象的问题具体化，另外，还需要有一定的艺术才能。

在团队的外围，相关的角色还有客户和经理。

客户是软件产品的直接利益相关者，他们从业务的角度提出对软件产品的需求。从本质上而言，客户是开发软件的根本动力，为软件开发支付相应的费用。在开发过程中，他们需要对软件的进度、是否满足需求进行相应的把关，并参与阶段性的回顾。客户的特点是对业务有深入的理解，能够清晰地理解业务流程。

经理在团队中的任务是控制开发进度、解决团队的资源问题、对团队的运行进行技术性的指导等。根据这三种不同的任务，可以有三个人分别担当不同的经理角色，分别是项目经理（Project Manager）、人事经理（People Manager）及指导经理（Coaching Manager）。通常情况下，也可能是一个成员同时兼顾多个经理角色。经理不直接参与项目，但在项目的外围提供关键的支持，为软件开发营造良好的环境，因而需要有更高的视觉和领导力以完成相应的任务。

作为测试团队的一员，小艾最关心的是测试团队中都有哪些角色分工。可以认为，测试团队的成员都是敏捷开发项目组的测试人员，都具备测试人员的一般技能。

由于软件测试本身就可以作为一个项目来看待，因此测试团队中需要相应的项目组角色。测试负责人（Test Lead）是测试的主要统筹者，需要担当测试项目经理的角色，其任务包括定义测试计划、统筹人员调配、监督测试项目进度等。测试负责人定期向项目团队发布有关测试项目的进度和更新，对测试项目进度负责。作为测试的统筹者，为了顺利完成测试任务，测试负责人需要利用相关的规则结合经验安排测试的执行顺序，降低测试进度受阻或测试检测出严重问题但难以解决的风险。测试负责人的技能要求是综合的，既需要掌握测试的专业技能，又要具备良好的组织能力和协调能力。

测试架构师的职责是定义测试策略，从宏观上定义测试的方向和方法。测试架构师对测试目标的技术特性和业务需求有准确的把握，能为测试团队提供方法论方面的全面建议。在测试计划完成后，测试架构师需要审核计划是否全面覆盖应该包含的验证点，根据经验给出相关的执行建议。作为测试团队的“智囊”，测试架构师应该具有较高的技能水平，包括深入和全面的测试经验，对软件开发和测试的模型有全面的认识，对商业模式及客户的业务需求也有比较深刻的理解。

相对于具有宏观视角的测试架构师，测试工程师以“微观”的视觉专注于具体的测试任务。根据特定的测试场景，测试工程师重点关注其测试的目标业务部分，根据特定业务场景制订该部分的测试计划。由于不同的测试类型之间存在依赖关系，测试工程师得进行团队的直接沟通，使测试计划可以满足这种依赖。测试计划制订完成以后，测试工程师就根据计划执行测试；同时，在测试过程中发现缺陷时，测试工程师还有义务和开发人员一起分析问题的原因并提出解决方案。测试工程师的技能集主要包括设计和执行测试用例的专业技能，良好的业务理解能力和问题分析能力。

测试经理（Testing Manager）从资源调配的角度给不同的测试项目分配资源。通常来说，测试和开发的执行步调是不一致的，因此同一个测试人员有可能同时承担多个子项目的测试任务。在一个敏捷软件开发团队中，对测试人员的多任务处理能力有较高的要求。

在软件开发团队和测试团队中，有些角色对承担者的资历有明确要求，如架构师角色要求对业务和技术有一定的经验。然而，角色的不同仅仅体现分工的不一样，并没有级别的区分。因此，角色并没有贵贱之分。每个角色的技能集都非常明确，相同角色的承担者在技能的层次上也可能存在很大的差异。随着技术和经验的积累，每种角色都可以成长出资历深厚的专家。专家和菜鸟，在某种程度上，仅仅是一种“闻道有先后”的差异。

了解到团队中原来有这么多不同的角色，而不是仅有“程序员”，小艾觉得非常兴奋。在一个完备的团队中，体验不同角色的奥妙，该是一件多么有趣的事情！

1.2.2　好软件由测试决定

进入电子商务平台开发一段时间以来，小艾不时听身边的前辈提及 IBM 的软件质量不错。似乎每个人都认为自己有清晰的标准衡量软件的好坏，但对于什么样的软件才是好软件，小艾心中并没有明确的界定标准。用“好”来形容软件，显然是一个相当笼统的描述。

软件质量[1]包括两个相关但截然不同的概念——功能性质量（Functional Quality）和结构性质量（Structural Quality）。功能性质量反映的是软件是否按照设计实现并满足相应功能性需求（Functional Requirements）；结构性质量反映的是软件是否满足相关的非功能性需求（Non-Functional Requirements，NFR）。

要评价软件的功能性质量和结构性质量，有一系列衡量指标。有了衡量指标以后，另一个重要的问题就是如何获得这些指标的量化数值。软件测试是验证这些指标的有效方法。通过测试可以在一定程度上模拟真实的使用场景，并得到质量指标的具体水平。如果测试发现某些指标无法达到要求，则需要对系统进行改进，以求通过测试。测试的通过指标是根据质量的需求来定义的，系统通过了测试，可以从量化的角度说明它符合需求。

正确性（Correctness）反映了实现的功能达到设计规范并满足用户需求的程度。这是功能性质量的基本指标。正确性可以通过功能测试来验证。

可靠性（Reliability）衡量在规定的时间和条件下，系统维持其性能水准的程度。这是结构性需求的重要指标。对于企业级的应用系统，对可靠性通常都有很高的要求。可靠性指标可以通过系统可靠性测试获取。

易用性（Usability）反映用户掌握软件操作及理解软件事务所需付出的时间及努力程度。具体的指标诸如界面是否友好，是否有在线帮助，是否提供容易理解的异常信息等。易用性指标通常由功能测试获得。

可移植性（Portability）衡量系统从一个平台转移到另一个平台的容易程度，包括把程序从一种软/硬件环境转移到另一种软/硬件环境的容易程度等。大型软件的安装和部署可能也是一个复杂的过程，高可移植性的系统应该是容易安装和更新的。此外，企业级系统对多国语言的支持程度也是可移植性的一个衡量指标。可移植性在多平台的功能、系统

1　http://en.wikipedia.org/wiki/Software_quality

测试、安装测试、多国语言测试中得到验证。

可迁移性（Migratability）衡量系统版本升级的容易程度。大型系统的迁移通常是一件非常复杂的事情，可迁移性需要通过迁移测试来验证。

效率（Efficiency）衡量系统执行某功能所需的计算机资源和时间有效程度，包括功能和性能是否经过优化，是否检验内存泄漏或溢出问题等。效率是系统测试的一个重要检测点。

可维护性、可扩展性（Maintainability、Scalability）反映当环境改变或出现错误时，执行修改或修复的难易程度。系统的设计是否很好地考虑日后扩展的需求，架构是否灵活等因素决定可维护性和可扩展性。系统测试可以获得系统的可扩展性指标。

健壮性（Robustness）衡量系统在接受异常或错误输入后能否返回正确的提示信息且不影响正确运作的指标。详细的功能测试是检验健壮性的主要方法。

安全性（Security）衡量系统对攻击性或不当的访问的抵御能力，检测的方向包括在受到没有授权的访问时系统对自身及数据的保护程度，系统的安全机制是否正确地实现，系统在受到攻击时是否能保持正常的业务运作等。系统测试有专门的测试涵盖安全性的审核。

有了诸多衡量质量的指标，软件的好坏就可以量化了，可见有效的测试是软件质量的重要保证。测试除了提供量化指标以外，还可以作为动力来驱动开发的进度，这就是极限编程倡导的测试驱动开发（Test-Driven Development，TDD）。

测试驱动开发的要点是先写测试程序，然后再编码实现使其通过测试。测试可以有效推动需求的实现，但是测试场景的覆盖度不足以涵盖所有的分支，因此，开发前的完整设计及第一轮开发过后的详细功能测试能够避免测试场景的覆盖问题。这样，测试场景相当于提供给开发人员的指导性主线，加快主要功能点的开发速度。

测试驱动开发的方法为开发提供一种新的方式，测试处于主导的位置。但测试的更重要作用还是在于提供衡量软件质量的量化指标。因此，我们认为，软件的好坏是由测试决定的。

1.2.3 测试也有大学问

既然软件测试对于软件质量有非常重要的影响，那么，如何有效地进行软件测试，则是非常值得关注的问题。

测试的目标是发现软件系统中存在的缺陷，这其中有一个关键的原则——尽可能早地发现问题。

在软件开发的前期甚至是设计阶段，某些缺陷可能已经存在，然而在这个时期要发现问题并不是件容易的事情。原因是多方面的。首先，在系统的很多功能模块尚未开发完善时，由于相互依赖关系而引起的缺陷一般难以被发现，因为缺陷只有在系统进行了集成以后才会真正暴露出来。例如，有些模式在简单的系统架构下可以带来不错的开发效率和运行效率，但是随着新模块的加入，这种架构的集成复杂度会急剧提高，系统原有的“精简”优势在这种条件下不复存在，新模块的运行效率会受到明显影响。这是个很明显的缺陷，但是这种设计的缺陷在新模块加入以前不可能很容易地暴露。其次，有些缺陷在开发前期看起来也许算不上是个缺陷，测试人员很容易忽略或仅仅把它当做一个“事件”。这种问题的严重性随着开发的深入才会逐渐显现。一个典型的例子是数据库查询的效率问题。使用全表扫描（Table Scan）可以获取完整的数据集合，全表扫描的效率缺陷在开发初期常常不容易被察觉，随着越来越多的数据和查询的加入，全表扫描的问题才会变得越加明显。另外，在开发周期的前期，项目人员通常会把注意力放在开发新功能上，在测试方面投入的资源相对较少，因此发现问题的可能性也降低了。

从图 1-2 中可以看到，一个相同的问题如果在不同的开发阶段解决，所需的开销是不一样的。越到开发后期，解决问题所需的开销越大。

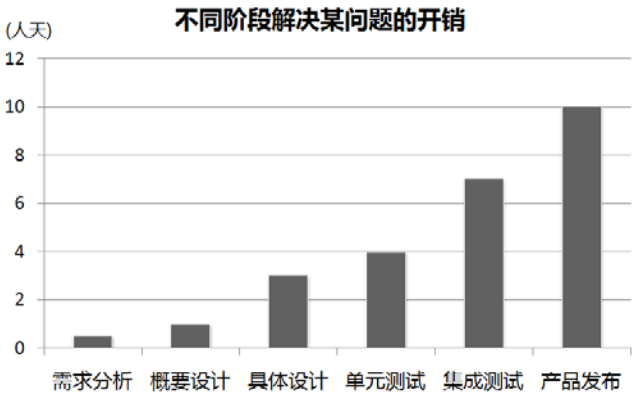

图 1-2　相同的问题在不同开发阶段解决的开销

有人对“尽可能早地发现问题”这个观点持怀疑态度。既然问题在开发的前期隐藏得非常好，而在后期更容易暴露，那为什么不干脆等到后期才专注于缺陷的发现和解决？这样不是更能降低测试的成本吗？

就测试本身而言，这种想法有道理，减小测试的开销同时让更多的问题暴露，似乎是个很理想的策略。然而，从软件项目的整体上考虑，这就是个非常糟糕的选择了。同一个缺陷，在不同的时间段发现，对其进行修复所需的努力很可能极不相同。一个典型的例子是使用了不恰当的消息模式而导致系统与外围通信效率低下的问题。如果问题在需求分析阶段即被发现，那么只要考虑更换一种消息模式。因为这个过程中任何实质性的劳动都不存在，所以并不需要额外的劳动开销。如果问题是在设计阶段出现的，那么更换消息模式还是必需的，同时，还需要考虑更换模式对外围接口的设计是否有影响。当然，这种考虑主要是评估性质的。假如是在功能实现的前期发现问题，那么除了评估对外围的影响以外，重写一部分代码是必不可少的。随着开发的进一步推进，可能有很多模块对消息模块存在依赖，如果针对依赖的开发已经完成以后才发现问题，不可避免地，相应的依赖模块的设计和实现就得重做，所需的代价变得更加高昂。

软件开发是一个迭代和累积的过程，越是底层的缺陷，发现的时间越晚，修复缺陷的代价越高。软件开发项目一般是以工程的方式运作的，作为工程会有明确的目标，因此，风险的控制非常重要。如果已知缺陷存在而无法修复，软件产品是无法发布的。如果在后期发现严重问题，修复难度很大或者需要大面积的改动，项目的风险会陡然增加。项目组面临的选择只有延期完成或宣布项目失败。很多失败的项目都是因为多次的延期而宣告失败的。可见，从项目整体的角度而言，“尽可能早地发现问题”才能降低风险，而问题越迟发现，项目的风险越高，对整体进度的影响越大。

作为测试专家，应该考虑的问题是如何更早地发现缺陷，以及有效地解决缺陷。

正如之前曾经提及的，开发的前期，缺陷可能已经存在但不容易暴露。那么，有没有办法让问题“提前”暴露呢？

发现问题的可行方法有两类，分别是分析方法[1]和测试方法。分析主要使用逻辑分析推理的方法发现缺陷和评估问题的严重性，并根据所处的阶段得到解决的方法。例如，有一个报表系统，起初是使用直接SQL查询的方式实现报表功能的。对实现的方法进行分析，可以发现，要生成条件复杂的报表，需要完成执行效率较低的查询语句，执行查询的时候是需要给相应的表格加锁的，因此有可能影响对相同的表有业务操作的模块。单从报表模块的设计和功能实现而言，使用直接的SQL查询已经可以满足功能上的需求，但综合考虑

1 分析方法也可归类为静态测试，也算是测试的一种。分析中发现的问题也应该当做是“缺陷”——“缺陷”并不仅限于代码出现的问题，也不仅限于由测试执行人员发现的问题。

运行效率和跨模块的相互影响，通过分析可以得到结论——报表系统使用直接SQL查询的方法，在系统完成实现后会带来性能和系统级别的缺陷。

测试和分析人员可以针对这个问题直接创建一个缺陷，虽然这并不是通过测试发现的。针对这个问题，项目组有机会在设计阶段修复这个潜在的缺陷。这里使用“潜在的”作为定语，是因为这个缺陷没有经过测试证实，而是通过分析的方法推导认定的。但由于理据充分，本质上这个缺陷和测试发现的缺陷是一样的。为了提高查询效率，考虑使用物化表存储报表的内容，再通过筛选物化表的记录生成报表。因为有了物化表，可以把生产数据和报表数据最大限度地隔离，避免了数据锁定而引起的冲突；物化表从性质上也保证了查询的效率。重新设计和实现以后，可以认为对这个缺陷有了一个解决方法。最后要做的是通过严谨的测试证实问题已经解决或者潜在的问题得到避免。测试的方式是对设计分析时认为有问题的场景进行模拟，如果在这种场景下没有出现此前认为会出现的问题，那么这个缺陷解决方案就被认为是可以接受的。

分析方法不需要等待缺陷目标的开发完成并使用测试进行验证，然而，这种方法对分析人员技能要求较高。分析人员在需求分析和设计方面的经验必须比较丰富，才能准确定位问题所在。实施难度相对较低的是测试方法。测试方法设计出有针对性的场景，并在测试环境上模拟该场景。如果测试的输出和预期的输出存在差异，则能证实问题的存在。然而，要使用测试的方法发现问题，对测试环境是有要求的。要运行测试场景验证用例是否成功，前提是测试的场景能够在测试环境中正常运作。黑盒测试方式的测试用例一般是端对端（End-To-End）的，也就是测试用例是个完整的业务场景，而不仅仅是一个单元。在开发的早期，要走通一个端对端的用例大多情况下只是个奢望。可用的方式是使用“假对象”（Mock Object）或模拟器把端对端场景中没有完成实现的部分补充完整。

例如，在一个面向服务的体系结构（SOA）开发模式下的系统，如果某些流程中服务并没有开发完成，但目标测试用例必须用到这个没完成的服务，为了让用例完整地走通，可以设计一个简单的假对象。假对象不实现任何逻辑，对于任何输入，仅返回符合格式要求的特点数据。有了假对象返回的内容，业务流程就能以这种临时的方式完成。因为测试的重点在于已经完成的部分，因此假对象没有任何业务逻辑，也不会影响测试的有效性。如果测试验证失败，则证明测试目标存在缺陷。这时候可以对缺陷进行跟踪和解决。为了在早期发现问题，使用假对象或“假业务数据”来完成测试，都是常用的“主动测试”策略。

无论是使用分析方法还是测试方法发现的问题，都通过创建缺陷来跟踪。高效的项目组一般都有完善的缺陷跟踪机制和系统，使应有的资源流向相应缺陷并尽早解决问题。缺

陷跟踪系统定义了缺陷的生命周期和相关信息[1]，如图 1-3 所示。

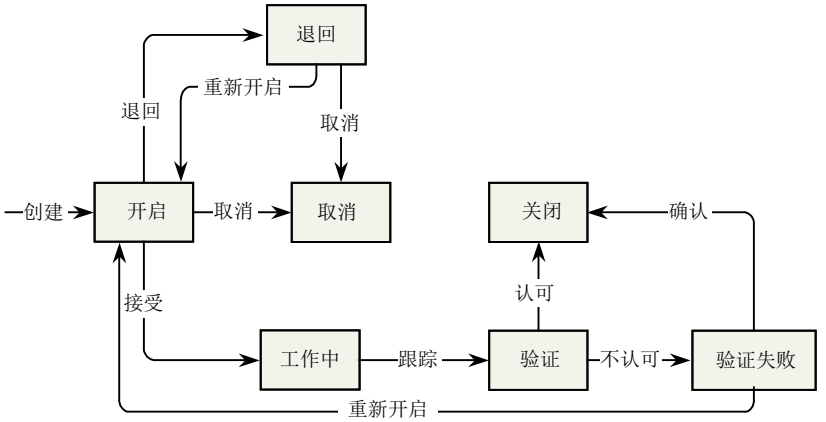

图 1-3　典型的缺陷生命周期流程图

缺陷（Bug），一般是指系统存在的问题或者需要加强的细节。从广义的角度而言，系统中任何需要修改或强化的任务都可以归类为缺陷。发现缺陷的人员在系统中创建一个新的缺陷，对于这个缺陷而言，创建的人是缺陷的创始人（Originator）。创始人会明确说明缺陷的内容，包括测试的时间、环境、测步和问题的描述、建议等。缺陷创建后，它处于开启（Open）状态，在任何时候它都会有一个直接的负责人（Owner），负责人是必须对缺陷采取行动的那个人。负责人的义务是推动缺陷的解决。初始化的时候，系统会根据一定的规则指定缺陷的负责人，创始人或者被指定的负责人可以重新指定（Assign）更适合解决该缺陷的人员为新的负责人。

针对一个开启状态的缺陷，首要的任务是验证其有效性，因为在不少情况下，缺陷对应的问题是由于不符合规定的操作导致的。遇到这种缺陷时，负责人仅需要把缺陷退回（Return）给创始人。如果缺陷被返回，创始人可以再次确认缺陷是否合法，假如缺陷确实合法，则可以重新开启（Reopen）缺陷，使缺陷回到开启状态；如果证实缺陷仅是不符合规定的操作引起的问题，则可以把它取消（Cancel）。取消状态是缺陷生命周期的其中一种终结状态。如果负责人经过验证后证实了缺陷的有效性，那么下一步的工作就是谋求解决方法。开始考虑解决方案前，需要接受（Accept）这个缺陷，使缺陷的状态从开启转成工作中（Working）。

1　此处使用的缺陷管理模型覆盖了典型的缺陷生命周期状态，根据项目差异不排除其他衍生状态。

在工作中状态下，缺陷的负责人可以与测试人员（缺陷创始人）及相关人员一同讨论问题的原因和解决办法，并根据方案对文档或系统进行修改。修改完成以后，负责人需要把缺陷的状态改为验证（Verify）状态，并创建一个验证记录（Verification Record）供缺陷创始人验证。这时缺陷的创始人同时也是负责人。如果验证通过，问题已经解决，则缺陷创始人可以认可（Accept）这个解决方案，这时缺陷的状态会变成认可（Accept）。系统可以关闭（Close）处于认可状态的缺陷。

从开启到关闭状态的流程，是缺陷生命周期的主要流程。在任何时候，基于新线索的发现，缺陷创始人都可以取消缺陷。有一些问题可能在不同的测试用例中以不同的方式暴露出来，或者分别被不同的测试人员发现，这种问题所被报出的缺陷最终都会被归结为同一个缺陷。缺陷负责人仅需要跟踪第一个被创建的缺陷（主缺陷），而其他缺陷会被标上重复（Duplicate）的记号。然后，验证缺陷的步骤无论是对主缺陷还是重复缺陷的创建人都是必需的。

这种基于状态的缺陷生命周期管理模型有利于跟踪缺陷的状况并推动缺陷的及早解决。缺陷的属性中会包括重要性、严重性、发现时间等信息，根据这些信息，项目组可以把更多资源分配给更重要的、严重的和紧急的缺陷。

测试专家在大多数情况下会作为问题的发现者出现，因而也顺理成章地成为缺陷处理的推动者。如何更早地、有效地发现问题，是测试专家的一项非常有技术含量的工作。而测试专家的另一项有技术含量的工作，就是发现问题后的问题分析（Problem Determination，PD）。

开发人员要做的，远不仅限于开发；而测试专家要做的，也远不仅限于测试。系统出现缺陷好比一个人生病了，而问题分析的过程则好比医生对病情的诊断，问题分析的主要任务是找到问题的原因。只有发现了问题的原因，才有对症下药解决问题的可能。

问题分析常用的系统方法有两种——自顶向下（Top-Down）和自底向上（Bottom-Up）方法。在分析错综复杂的问题，如系统级别的结构问题或性能问题时，这两种方法能够有效地定位问题。

自顶向下方法，其着眼处在于整体。使用自顶向下方法，首先应该认同的一个观点是，系统整体的问题可能是系统某个部分的原因引起的，而这个局部的问题放大后会在系统的宏观级别上表现出来。通过观察和分析存在缺陷的系统整体情况，把系统分成若干部分，并逐个部分判断可能存在的问题。判断确定“嫌疑”所在后，使用调整——测试的方法证实判断的正确性。如果判断正确，宏观的问题确实存在于系统中，那么可以对这个细节进行修改，解决问题；如果判断不正确，则需要重现判断。从宏观到微观的过程也可能不是

一步到位的，在复杂的系统中，这个过程可能会分为多个级别的细化来完成。有时候，整体的问题可能是由多个划分同时存在的问题引起的。通过逐个验证测试系统划分的“地毯式排查”方式，可以扫清所有“患处”。

一个系统的局部分解方式通常是稳定的，使用自顶向下方法的理念在于通过这种稳定的分解，使用穷举方式最终可以确切地找到发生问题的局部。其好处是对于经验不多的人员，使用自顶向下法也能最终找到问题所在。当然，缺乏经验也可能带来苦头——对某些局部的验证也许是不需要的。

相对而言，自底向上方法对分析者的能力要求较高。使用自底向上方法的前提是，承认缺陷的全部或部分是由于系统局部细节的问题引起的。分析时，根据系统表面看到的蛛丝马迹，直接判断出现问题的根源，并验证这个判断是否正确。实践上，可以从最底一层起，对系统在逻辑上或功能上进行划分，然后设计特定的测试场景，有针对性地验证这些划分是否正常。因为对系统进行过划分，所以一旦验证性的测试重现了问题，则能够比较清晰地定位究竟是哪个部分存在问题。

这个从下往上的判断过程当然是很有讲究的，遭遇问题的经验、对问题的“嗅觉”，都有助于提高判断的准确性。复杂的系统中可能有问题的细节成千上万，作为分析人员，首先应该对系统的这些细节及其在整体中的作用比较熟悉，其次要拥有直达问题根源的“直觉”。如果缺少这种一击即中要害的技能，自底向上的问题分析方法可能并不奏效。

值得注意的是，无论是自顶向下还是自底向上的问题分析方法，其本质其实都是准确地重现和定位问题。只要问题得以有效重现和定位，离找到解决的办法就不遥远了。

自顶向下和自底向上方法可通用于一般各种测试类型，针对不同测试类型，这两种方法的具体使用方式可能存在不同之处。

发现、定位和解决问题的方法，是测试人员的核心技能。由于测试的门类众多，针对系统的测试也有多种角度，这些方法在不同的测试中会有不同的具体表现。

经过与凯文的谈话，小艾觉得自己对测试本身及其重要性、测试的技巧和窍门等方面的理解都加深不少。许多要点其实小艾在日常工作中已经有所接触，只是缺少系统的总结和提炼，而其他一些能力，则需要通过更多积累才能达到。凯文提醒道：“作为测试专家，核心的能力其实还是思考的能力。五花八门的测试方法和技术，得通过自己的实践、总结和思考，转化成系统的测试方法论。当一套属于你自己的测试方法论已经形成的时候，意味着你已经从专家成长为高手了。测试，在这种角度看，就是一个完备的哲学体系。”

1.3　从专家到高手

小艾成为软件测试工程师加入项目团队已有一段时间，参加的几个开发项目给他带来了一些实践经验。时常倾听同事的经验介绍，让他有机会对自己所处的水平做出一个合理的判断。随着软件测试基本理论及实践经验的积累，小艾感觉自己跟刚加入时已经有明显的区别。这种区别体现在几个方面，包括对组织结构的认识、对工作内容和工作方法的理解、对测试相关的专业技能的掌握等。软件测试工程师是一个技术背景非常强的职位，因此，技术是这个职位的立足点。尽管没有详细实践过不同的测试，但小艾已经对测试的来龙去脉有了比较系统的了解，对于测试的关键点有了清楚的认识。应该说，小艾逐渐从测试菜鸟成长为一名测试专家。

对于在技术上进一步修炼的方向，小艾依然有自己的疑惑。究竟达到什么样的程度才算是脱离“菜鸟”头衔而成为一名专家呢？专家是否就意味着技术上已经达到了顶峰？优秀的测试工程师应该有哪些标准？

带着这些疑问，小艾又找到他的导师凯文。面对有着类似经历的新人，凯文一直都是知无不言，言无不尽的。这种分享的氛围，使软件开发实验室成为一个非常适合学习和交流的地方。

凯文的解释从区分新手、专家和高手三个级别谈起。

刚开始接触一个新的领域时，对这个领域一无所知或者知之甚少，对于这个领域，我们就是新手。正如之前已经提及的，术业有专攻，同一个人，在一个领域是新手，并不妨碍他在另一个领域是专家。作为测试工程师新手，要成长为专家，需要对测试的相关专业技能进行系统的学习和实践。在开发项目组里，测试团队本身就包括多个角色，对于每种角色，从技能水平上也有新手和专家的区分。新手到专家的学习曲线主要包括学习测试和开发方法、测试计划和测试设计、测试文档的编写、发现和解决问题的一般方法等。在从新手到专家的发展过程中，“准专家”可以在专家的指导下完成特定的测试任务，能发现和解决一些比较常见的问题。随着经验的积累，测试新手可以成长为测试专家。

软件测试的关键考量点是软件的质量，因此，对于软件工程师而言，经验积累是新手成长为专家的过程中不可缺少的环节。当测试新手对测试的相关技能都熟练掌握、经验积累也达到一定程度之后，新手就基本上成长为测试专家。测试专家对相关的测试技能和工具都已经熟练地掌握，能够以标准化的方式策划并完成测试任务。专家的“专业”，主要

体现在他对软件质量的控制把握方面的专业。

而高手则是专家更进一步的发展方向。专家的特点表现在对流程的严格把握，通过一种控制力保证软件的质量；而高手则比这走得更远。有一些非常复杂的系统或复杂的应用场景，已经超出了正常流程的控制范围。在这种条件下依然要保证系统能够正常满足需求，于是对测试人员提出了更高的要求。高手的价值就在这种超越一般情形的条件下体现出来了。总体而言，高手对于发现问题、解决问题，有更多超出一般步骤的灵感和直觉，这种灵感和直觉是建立在对系统的深入理解及高手本身强大的洞察力基础上的。

如果说新手到专家的成长过程是一个学习“硬性技能”的过程，那么专家成长为高手则更在于“软性技能”的修为。软性技能的提高，归根结底是思考能力和分析能力的提高。新手可以随着经验积累和技能学习成长为专家，然而，经验的继续积累只是专家成长为高手的其中一部分。专家到高手的修炼，重点在于思考的方法和技巧。

1.3.1 像外行一样思考，像专家一样实践

如果抛开测试的具体技术和实现细节，只是关注测试的目的，那么测试的本质其实就是发现问题和解决问题的过程。在讲述问题分析的方法时，我们介绍过自顶向下和自底向上的方法。作为对一般性问题的分析方法，这两种方法都有助于问题的分析。但对于一些非常规性的问题，这种系统的方法却不一定奏效，可能效率并不高。这时，需要有非常规的方法来应对。

如果让完全没有经验的人员进行测试并发现问题（我们称这个人为外行），遇到问题时，这个人可能有两种应对情形，第一种情形是束手无策，不知发现问题和解决问题都该从何入手；第二种情形是，这个外行不受任何成规的约束，提出一些天马行空的想法。因为没有专业背景的限制，这些想法可能真的不着边际，甚至扰乱了问题本身的解决，然而，这些不着边际的想法有时却能带来令人意想不到的效果，或是从全新的角度发现了问题的本质，或是找到了不同的思路和方法。相对而言，一个普通的专家因为受到许多既有方式的限制，就不太可能得出这种天马行空的点子了。外行以一种随性的方式思考，这种方式往往会带来意外的效果，因为随性的思考方式不会被规则限制。

作为测试人员，在发现和解决问题时，外行的思考方式可以成为有效的切入点。好的切入点是一个不错的开始，然而，真正的实践还是必须以专家的严谨和慎重来验证想法是否正确。像专家一样实践——我们倡导的还是一种小心求证的态度。软件测试是一个非常严谨并且以事实说话的过程，任何假设都必须通过测试实践的检验。

对测试已经有深入了解的人，想做到像外行一样思考，并非易事。测试专家往往下意识地对一个方法的技术可行性做判断，这种下意识能够高效地排除许多不可行的方式方法。但在某些时候，这种下意识却阻碍了创造性灵感的萌生。很多有意义的想法会因为可行性的判断而被扼杀在摇篮之中。像外行一样思考——追求的是一种新的方法或者角度，仅仅考虑某个问题是否可能存在或者某种方法是否能解决问题，不考虑方法是否有理论依据，也避免过多地考虑可行性。

其实，可行性是基于以往的经验做判断，但是谁也不能认定，当前不可行的方法就永远不可行。认为科学已经进步到了终点的想法早已被证明是荒谬的。遇到棘手的问题时，测试高手能够跳出既有的条条框框，像一个外行一样重新审视系统的整体。当然，我们并不认为测试高手是个外行，因为这种“不受约束”的审视其实是建立在对系统的全面深入的理解基础上的，并不是一种纯粹的盲目。在这种大胆假设的前提下，即使是高手，也必须小心地求证假设是否正确。求证过程离不开反复的实验和验证。

面对复杂系统的问题，这种“大胆假设，小心求证”的方法往往能产生神奇的效用。以下是一个真实的例子，在对一个多节点的集群电子商务进行系统测试时，发现在高并发访问的条件下，从应用服务器到数据库的连接数骤然增加，并很快到达连接数的上限。数据库连接数一旦到达上限并且没有及时释放时，新的请求会因为无法获取连接而被阻塞，在表面上看，系统的性能会表现得非常糟糕。

面对这样的问题，一般的问题分析方法可能难以在短时间内找到原因。这时候需要在现有的条件下大胆假设可能有问题的地方。“现有的条件”指的是一些表面的系统运作数据，如应用服务器日志、数据库锁信息、数据连接池信息等。根据这些信息，发现有种特定的操作一旦出现，系统的数据库连接请求会急剧增加。通常情况下，因为存在系统级别的缓存，重复的访问一般不会给系统带来重新计算的负担。然而，问题的表现是，反复的访问似乎对系统产生了明显的性能影响。

这种情况下可以大胆假设系统的缓存设置可能有问题，虽然按照正常流程安装和配置的系统不可能存在缓存的问题。基于这个假设，接着要做的是检查所有和缓存相关的配置内容。检查发现，价格模块的对象缓存并没有设置，而这个设置正常情况下应该是激活的。如果没有价格的对象缓存，那么相同的价格对象都不会被缓存在内存中，而是每次获取的时候都重新计算生成。

在电子商务系统中，价格信息是使用非常频繁的一类信息，因为缺少对象缓存，实际应用就有可能出现不断地查找数据库计算价格的情形，这会导致数据库连接被大量占用。

更改设置后重新测试，验证发现使用了价格的对象缓存，数据库的连接数不再出现被异常地大量占用的情况，问题得到解决。发现了对象缓存的设置错误，进一步追寻原因，发现原来是系统安装的过程中配置脚本运行出现了异常，从而导致缓存的创建步骤并没有被执行。如此一来，整个问题的来龙去脉就非常清晰了。解决问题以后，这种看似很复杂的问题，其原因也许很简单。解决这个性能问题的关键在于假设问题的原因是缓存的设置有问题；而验证恰好证实了假设的正确。如果仅仅使用一般的问题分析方法来寻找问题的原因，这种“意外”的问题往往是非常棘手的。

“像外行一样思考，像专家一样实践”的方法是一位著名的计算机学者谈及学术研究时提出的一种方法论。软件测试虽然和学术研究有着明显的差异，但是测试过程中需要发现和解决问题的时候，这种方法论很有借鉴意义。在软件测试中，对待问题同样需要开阔的视野和严谨的求证态度。我们认为，测试专家能够在测试中发现绝大部分的问题并能够使用合理的分析方法找到解决绝大部分缺陷的方案，而高手则能够更进一步，最棘手的问题也能够有效解决。

能否以这种收放自如的思维方式应对测试中遇到问题，是高手和专家的一个重要分野。在大部分情况下，这种分野是不明显的，因为最困难的问题只会占所有问题的很小一部分，而这种问题在测试中不会很容易地暴露。然而这种问题被发现了之后，高手和专家在造诣上的差别就会显现出来。

1.3.2 工欲善其事必先利其器

对于测试工程师而言，虽然发现和解决问题才是体现其价值的事情，然而测试工程师不得不花大部分时间执行测试。

从图 1-4 中可以发现，对于一个普通的测试工程师来说，执行测试消耗了很大一部分时间，而常规项目[1]的任务把可用时间的 90%都占用了，剩余可以用于提高生产效率的资源变得非常紧缺。而提高生产效率从长远来说又能降低常规项目任务占用时间的比例。

在一个水平较高的开发团队中，设计和代码实现的水平通常是比较高的。在这种团队中，测试的注意力会更多地放在验证和问题解决方面。验证是通过执行测试的方式完成的，真正运行一个场景，查看系统的反馈是否和预期吻合。对于结构复杂的系统和对软件质量

1 测试自动化任务不在常规任务的范围内。

要求很高的软件，需要执行多种类型的测试验证各种场景，而每种测试都可能包括大量的测试用例。例如，在电子商务系统一个新版本的开发过程中，功能测试的用例可能多达成千上万，涵盖各种正常或异常的分支场景。对于如此大量的测试用例，执行的工作量之大可想而知。

图 1-4　测试工程师的任务及任务所占资源的比例

如果测试工程师的绝大部分时间都被执行所占据，那么可以用来分析解决问题的时间就相对很有限了。开发水平提高不能减小测试的工作量，那么，测试工程师通过什么方法更有效地完成测试任务呢？答案是提高测试效率。对于同一个测试人员，效率的提高有两种外在的表现，第一种方式是使用相同时间完成更多的测试用例执行；第二种方式是对于同一个或同一组测试用例，耗费的时间减少了。

相对于测试新手对测试执行的生疏，测试专家以熟练的执行更快地完成测试任务。提高技能的熟练程度，能够提高效率。通常来说，执行一个测试，需要完成一系列操作步骤，首先需要安装测试环境、准备测试数据，如果是使用自动化操作的方式执行的测试，则需要准备测试脚本或代码；接着需要开启监控测试环境的工具，然后才能开始执行测试用例；执行完毕后，需要收集必要的数据和结果，确认测试是否通过。这一系列步骤的执行效率可以随着熟练的程度得到提高。

如果要通过提高熟练程度来提升测试的执行效率，提升的空间是有限的。对于测试高手而言，进一步提高效率，考虑的方向应该是减少对测试的人工干预，让测试自动完成。在工业化的测试条件下，自动化水平的高低，在很大程度上衡量了一个测试团队的水平。测试自动化指的是通过编写程序完成执行测试用例的所有或部分步骤。自动化的优势是减少人工的干预，把团队中最宝贵的资源——人释放出来；同时，由于自动化是使用程序的方式实现的，因此可以保证每次执行自动化测试程序的条件是一致的，避免了人为因素引起的不一致，影响某些缺陷的可重现性。测试自动化把许多烦琐的步骤交给程序来完成，测试的执行对人的依赖也得到减弱，从这个角度来说，自动化可以提高测试的质量。

自动化测试工具是测试工程师提高效率的利器。对于不同的测试方法，已经有一批针对性很强的自动化工具可供使用。针对基于 Java 的单元测试，JUnit 是最常用的测试框架。通过对 JUnit 进行扩展，还可以实现单元测试调度和自动结果收集等功能。功能测试的测

试目标是端对端（End-to-End）的用例场景，在测试基于浏览器的网络应用程序时，常用的自动测试工具有 IBM Rational Functional Tester（RFT）、Selenium、JMeter 等。对于胖客户端的应用程序功能测试，IBM Rational Functional Tester 也可以提供不错的支持。系统测试的过程需要模拟更复杂的用例场景，如不同行为的虚拟用户并发地访问用户界面，这种测试用例通常来说只能使用自动化测试工具来执行。

有一类性能测试工具使用结构化代码来模拟并发的用户行为。自动化测试工具构造测试代码的一种常用方法是录制操作步骤。当人工对界面进行操作时，录制程序可以完整地记录整个操作过程，录制完成后，测试程序员再对录制的内容进行标准化修改，修改后的测试代码就能够满足标准的测试场景要求。具备这种功能的系统测试工具很多，如 IBM Rational Performance Tester、Borland Silk Performer、Load Runner 等。

对于要记录测试页面响应时间的性能测试用例，可以使用 Firebug、IBM Page Detailer 等工具。对于安装测试，系统安装的过程可以使用安装测试脚本来自动完成，测试脚本同时可以验证系统安装的每个步骤结果是否正确。类似地，构建测试也有自动化构建测试工具完成构建的流程。作为一个成熟的团队，为了适应产品的特点，在使用一款自动化测试工具前，往往还得对工具进行定制，使工具更符合特定的测试需求。在测试创新的章节，我们还会讲述关于自动化工具定制开发的内容。

然而，自动化也会给团队带来额外的负担。如果要追求全局的自动化，那么一个完善的自动化测试框架必不可少，但是搭建自动化框架本身就是一个规模不小的工程。软件开发和测试方式随着技术的革新不断变化，这种变化有可能导致原有的自动化框架不再适用。另外，即使有完善的自动化测试框架，测试人员依然得完成基于自动化框架开发的测试用例，测试完成后，也还要对这些自动化执行资源进行维护。如果自动化框架实现的是部分自动化，那么执行过程中有些步骤还是需要人的参与，并不能完全脱离人工干预。可以说，没有经过深思熟虑而仓促上马的自动化执行方案，也许不但不能提高执行效率，反而会增加测试团队的负担。

脱离人工干预的程序控制在执行效率上的优越性很明显，因此在允许的条件下，一个测试团队应该有逐步实现测试自动化的目标和路线图。在初期，可以仅仅使用有针对性的自动化测试工具辅助测试，而随着自动化测试经验的积累，可以基于这些工具开发集成化的自动化测试框架。在实现提高效率目的的同时，也降低了“过度自动化”的风险。

小艾所在的测试团队，正是沿着这种方式逐步完成了手工测试到自动化测试的转变。当然，由于许多测试有着明确的需求，手工测试和人工干预不可能被自动化完全取代；而

不同的测试种类，使用的自动化策略也可能完全不一样。

提高效率的方式多种多样，提高熟练程度和测试自动化是比较常见的两种方式。除了完成测试任务，提高测试的效率和质量，同样是测试工程师的一个重要任务。测试高手区别于一般测试人员的关键在于测试高手更善于运用创新，在实践中不断提高。

1.3.3　从拿来主义到创新

小艾所在的电子商务系统，从技术和功能上而言，几年来的变化非常明显。早期的版本采用标准的 Java EE 技术，业务逻辑是通过命令模式实现的。随后，基于服务的架构开始流行，服务的灵活性的确有利于提高系统的适应能力，于是，系统的实现开始从基于命令转变为基于服务。系统的前端的早期实现主要是基于 JSP 的标准界面，而随着 Web 2.0 的兴起，许多新的前端元素逐渐被加入到前端界面中，如 Ajax，Remote Widget 等技术的使用，使系统的前端可扩展性和可操作性得到很大的提升。早期的电子商务系统实现的功能比较单一，而随着社交化应用的流行，越来越多的社交元素被集成到电子商务系统中，系统的复杂性进一步提高。除了新技术的引入，电子商务系统的核心——中间件的版本同时也在不断更新，在这个过程中，计算机的硬件配置的发展也相当迅速。一个系统似乎从来就没有最终版本。

电子商务领域仅仅是个缩影，整个信息技术领域都是快速发展的，变化之快可用日新月异来形容。一名技术专家，如果不能紧跟技术发展的步伐，那么最终的命运很可能是被技术所抛弃。新的技术常常是伴随着新的业务模式的流行而产生的，除了紧跟技术，有前瞻性的技术专家还应该洞察业务模式的发展。

新技术的出现也对测试提出了新的要求。因为实现的框架变了，测试模型必须做出相应的调整，在新的框架下完成测试需求。前台技术革新了，如果继续使用原来的技术，测试的结果可能不再准确，因此有必要引入新的前台测试技术。随着中间件、系统硬件的升级，测试的基线和指标也必须重新构建。业务在变化，技术在更新，测试技术同样需要创新。

创新不是空想，它是以对现有技术和业务的清晰理解为基础的。对于测试工程师而言，一开始往往需要学习和借鉴现有的经验，如测试的流程、使用的测试技术、分析问题的方法等。随着学习和实践的深入，在特定产品中，特定的需求会逐渐显现。因为是特定的需求，技术上很可能没有现成的解决方案，这种需求就会被作为创新的目标。有了明确的目标，就可以开始寻找新的解决办法，寻找的过程就是一个创新的过程。

创新的源泉一般都是现实的问题。项目开发过程中有过一些典型的例子。在测试项目中，系统测试的开始时间点比较晚，这不利于及早发现系统问题。如果系统问题到了后期才被发现，那么修复的成本又高得多。那么，有没有办法使系统测试在开发过程的早期开始呢？基于现有的技术和条件，这不可能实现。在开发的中前期，系统的功能界面并不可用，只有后台的部分服务可供调用。开发服务所要提供的单元测试用例和代码是具备的。

那么，能否对单元测试的资源进行重用，把它们扩展成可以支持系统测试的资源？从可行性来看，这个方法具备了部分条件，单元测试用例从业务上可以拼装成完整的业务流程，只是没有相应的前端界面。需要进一步完善的方面包括系统测试需要更大的测试数据集，如何配合单元测试用例实现灵活的测试数据集？单元测试的用例只能满足单用户测试的需求，而系统测试则需要模拟并发场景，那么，能否把单元测试进行扩展以支持并发？基于这些考虑，测试项目组设计了一个能够基于普通单元测试用例进行并发系统测试的集成测试框架。框架具体解决了数据集的问题和用户并发的问题。有了这个新的测试框架，测试团队具备了在项目的中前期开始性能测试的能力。

可以说，创新需要建立在对本领域最新技术的深入学习及对这些技术无法满足的业务需求的充分理解基础之上。

创新可以是对实践方法的创新，也可以是对指导实践的理论的创新。对于测试工程师而言，实践创新和理论创新同样非常有价值。

1.3.4 测试的广度和深度

在设计测试方案时，可以涵盖的分支非常繁多。功能模块的操作场景可以抽象成一个树状的结构（测试场景树），对模块进行测试，相当于对这棵树进行遍历。遍历有深度和广度之分。广度遍历体现在对更多功能的涵盖，当一个功能点发生变化时，具有高广度的测试用例需要验证所有可能受影响的功能。深度遍历体现在对功能细节的深入，除了涵盖正常分支以外，还要涵盖异常分支和错误分支，把一个功能点的每种可能情况都包含在测试用例中。

面对众多选择，测试工程师必须权衡资源的投入和测试的涵盖度的矛盾。权衡的一般方法是，先确定测试的可用资源和测试方法，资源和方法确定以后，能够完成的任务数量就可以估算了。对同一个功能点的测试，每增加一个深度或广度级别，对测试资源的消耗也相应增加。这种增加的对应关系不一定是线性的，提高一个深度或广度级别，增加的相应工作量可能是一倍或接近一倍，因为需要有一个独立的子用例来关注一个分支。因此，

权衡的时候得考虑种子覆盖深度和广度的提升在逻辑上是否必要，是否有业务需求。如果回答为否或一般，那么这种高投入的测试最好还是排除在测试计划之外。

与测试的覆盖相对应的重要指标是软件失败的风险。把有限的资源投放到风险最大的关键点，设计相应的测试深度和广度，测试工程师需要在矛盾中追求平衡。

割舍了测试的深度或广度，质量难免受到一定的影响，究竟如何判断是否应该割舍，是一种取舍的艺术。对于测试高手来说，有助于决策的，除了清晰的目标、丰富的经验以外，判断时的直觉也不可或缺。

1.3.5　无招胜有招

作为专业技术人员，测试工程师应该具有优秀的理性思考能力和逻辑分析能力。从前面的讲述中，不难发现理性思维对于一个优秀测试工程师的重要性。专业知识的学习、对系统和设计文档的理解、测试计划的编排和设计这一系列测试工程师涉及的主要工作任务，都需要理性思维方式来支撑。至于软件测试中最有价值的部分——发现及解决系统缺陷、测试方法的创新，理性思维的支撑依然不可或缺。然而，对于解决问题这部分，除了缜密的逻辑思考之外，对问题缘由的准确直觉可以发挥意想不到的功效。而方法创新，则对工程师的创新灵感有更高的要求。

直觉和灵感，都不属于理性思维的范畴，但对于测试高手而言，这两者至关重要。有一类原因比较复杂或很不明显的缺陷，使用通常的逻辑分析方法很难找到原因。这类问题，对于测试高手而言，直觉可以给予方向性的帮助。从问题表面来看，原因并不明显，但以往的测试经验可能会暗示问题出在某个不显眼的地方。这种寻找问题原因的方式似乎没有严格的科学依据，但是在工作中，直觉却很少看走眼。从思维的角度来看，直觉其实是右脑非逻辑性思维得到的一种输出。这也能够解释，为什么经验丰富的人，直觉通常更为可信。右脑思维的输入是人以往的经验及问题本身的表象，经过非逻辑性处理，得到的输出就是我们认为的“直觉”。人们通常觉得直觉不完全可信，是因为我们对右脑思维的方式并不完全理解，但是直觉的可信度在很多领域已经表现出来，在测试领域也不例外。我们的右脑是一个专门完成处理抽象信息处理的处理器，知识因为我们对其处理过程的不理解，因此这种方式往往没有得到重视。创新的灵感和直觉的产生方式是很类似的，也是右脑思维的产物。

无论测试新手、测试专家还是高手，其实都能完成右脑思维的过程。但是为什么测试高手能得到更多有效的直觉和创新的灵感？这正取决于输入信息的有效性。我们提到，输

入的信息包括知识、经验及问题本身。知识通过学习获取，经验是实践积累得到的，而对问题本身的理解，是一种从表面现象进行抽象提取的过程。掌握的知识越多、经验越丰富、对问题的理解更深入，那么提供给非逻辑处理的输入就越有效、越准确，得到高质量输出的机会就越大。抽象思维的能力是正常人都具备的一种能力，然而，如何进行有效的抽象思维，如何激活右脑，则是一个思维方法的问题。

有一种有趣的右脑思考的方法是这样的。测试中遇到一个棘手的问题要解决，这个问题通过逻辑分析的方式找不到解决方案。这种情况下，可以尝试的方式是，把这个问题的所有表面现象和相关的技术信息画成一张发散的意识图（Mind Map）。以问题为中心，所有的相关信息向外发散。意识图使所有已知的内容都会体现在其中。明确所有内容以后，接着需要做的就是到户外走两圈，或者干脆睡一会儿，又或者直接把这个事情先给“忘”了。过些许时候，可能是一个小时，也可能是几天，问题的答案有可能在不经意的时候从脑海里蹦出来，同时带来的是一种恍然大悟的感觉。

其实这一个小时或者几天的时间，正是右脑进行“后台处理”的时间。构造意识图的过程，其实是过滤非逻辑思维的输入信息的过程，有了干净的输入，接下来就什么都不需要做了。这种方式很容易让人联想到武侠小说经常提到的一种境界——无招胜有招。非逻辑思维的结果常常以直觉或灵感的方式出现，这都是非常有价值的内容。我们说，要得到的结果越有价值，风险也会越高，因此，使用这种方法进行右脑思考之前，也要考虑到，右脑的“处理进程”可能处于不响应的状态，或者返回的结果并不理想。遇到这种情况，也许得回过头想想，当初提供的输入是不是足够有效。

思考是一种修炼，修炼可能是个艰苦的过程，而思考则可以是轻松愉快的。人最厉害的招数不在于看得见摸得着的技术，而在于看不见摸不着的想象力。思考，是一种境界。

测试既是技术，也是艺术。技术在不断变革，而艺术则更偏向于美学和哲学的范畴。技术和艺术的共同点是，在这上面的修为都不存在终点。即使成为了测试高手，在测试领域，依然还有许多值得追求的事物。随着对测试的理解认识越来越深，测试工程师可以对测试的哲学提出独创性的见解，这种见解也许是技术上的，也许是方法上的，或者是管理上的。能够提出这种见解，测试工程师已经到达了测试大师的水平了。新的技术、新的方法论和新的测试哲学的不断出现，推动着整个软件测试领域乃至信息科学领域的不断进步。

听过凯文对于测试工程师的发展的一番论述，小艾发现，在测试的专业领域上，可以做的事情还有很多。优秀的测试工程师，除了专业技能，最重要的还在于懂得如何思考。与其说解决问题的能力是不同水平的测试工程师的分野，不如说是思考的能力把不同的层次区分开来。

1.4　职业生涯的考虑——技术还是管理

职业生涯规划是每个专业人员都应该慎重考虑的问题。测试工程师是一个专注程度很高的技术背景职位，职业上应该往哪个方向发展，是测试工程师到了一定时候应该考虑的问题。在 IBM，因为业务涉及很广的范围，因此，提供给员工们选择的空间非常广阔。就技术职位而言，发展的方向主要包括技术方向和管理方向。另外，转向商业业务部门也是可能的选择。那么，作为测试工程师，在技术上或管理上都有哪些可以选择的发展方向？一个菜鸟工程师，应该如何选择方向？

1.4.1　测试工程师的技术发展路线

作为职业发展起步，测试工程师可以先把自己定位为某种专门测试的专家，如功能测试专家、系统测试专家等。无论从哪种专门的测试开始，测试新手都有机会了解测试的一般方法论和通用的原则，这些知识对于不同的测试领域都是必需的。此外，针对不同的测试，测试新手需要学习特定的测试方法。例如，对于功能测试工程师而言，黑盒测试理论、测试用例构建和分析等技术是必不可少的；而对于系统测试工程师，了解不同种类系统的特性、性能指标，学习系统测试构建方式等技术是必修课；而对于构建测试工程师而言，学习构建技术的原理、系统部署技术、增量和全量发布技术等内容对于进行构建测试是非常重要的。有了一般性的技术和针对特定测试种类的专业技术作为基础，测试新手需要通过项目经验的累积，逐渐达到测试专家的水平。关于测试工程师的技术发展路线见图 1-5。

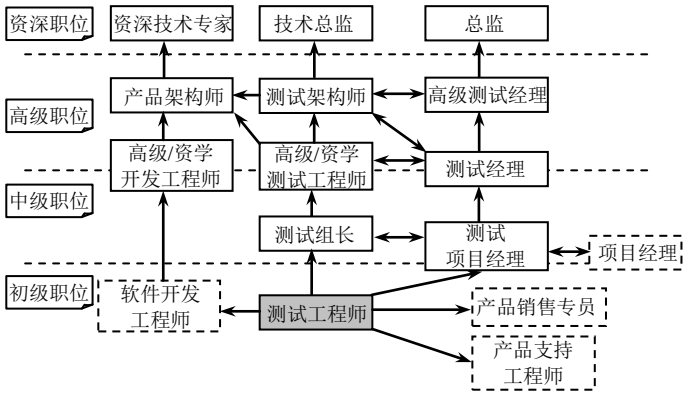

图 1-5　测试工程师技术发展路线图

由于不同的测试种类之间是有关联的，对于不同的业务步骤也很具有针对性。作为某种测试的专家，通常只熟悉和这类测试相关的系统特性或功能。如果希望对系统有更全面的认识，这时可以考虑转为另一种测试类型的测试工程师。在转换测试类型时，并不需要重新学习测试理论，而仅需要学习新的测试类型的测试技术和被测试的系统模块。测试工程师成为多种测试类型的专家以后，对整个系统的测试方法和测试流程都会有全面且深入的理解。

到了这个阶段，测试工程师要面临的是另一次选择。有了对多种测试类型的积累，工程师对被测的业务也有相对深入的理解，这时，可以转向基于基础产品的项目开发的主要测试负责人的角色，或者成为一名技术支持专家，专门解决和客户有关的技术问题。如果希望在测试方面做得更专注，那么，也可以选择成为产品的测试架构师，从不同的高度更深远地影响产品测试的方法论和策略。如果工程师起初并没有转向别的测试类型，而是专注在同一种测试类型上，那么，就有更多的时间集中地把一种测试类型研究透彻，目标是成为这种测试类型的专家或大师。每种测试类型都需要方法和实践上的创新，而测试高手和大师无疑更能推动这种创新。

我们说过，在敏捷开发团队中，测试工程师作为开发项目的成员存在。在积累了一些项目经验并具备相关的技能以后，测试工程师可以开始尝试敏捷开发团队的其他角色。一种典型的选择是成为开发工程师，把专注的点从测试变成开发。拥有产品质量控制经验的开发工程师，从技术上更能把握对设计和实现的质量考虑。当然，开发和测试就工作性质而言差别较大，对于刚从测试工程师转型的开发工程师，学习开发模型及相关技术会是一个不小的挑战。

本身经验已经比较丰富的测试工程师，可以转成产品架构师，直接参与设计。直接成为架构师以前，测试工程师应该已经了解一些测试以外的内容，如业务需求和市场背景、产品整体架构、适当的实现细节等。因此，从测试工程师转型为产品架构师，也是个很有挑战性的变化。如果希望技术做得更深入，产品架构师也许不是最好的选择；信息架构师角色，如应用架构师、基础设施架构师等，为走深入技术路线的测试工程师提供广阔的选择空间。

在 IBM，纯技术背景的专家只要做得足够好，都能得到认可。在这种氛围条件下，测试工程师可以根据自己的兴趣选择职业发展方向。有人说，兴趣是一个不时变化的玩意儿，不错，因此，在这里不乏从测试工程师转型成其他角色后又重新回来的案例。有一个足够大的舞台，周围有许多各方面技术的专家，也有许多成功或失败的案例作为借鉴，测试技术专家在确定自己的职业发展路线时拥有非常广阔的选择空间。

小艾作为测试工程师加入团队，随着经验的积累和技能的提升，越来越多的选择会摆在面前。当达到一定程度的时候，如何选择职业生涯发展路线的问题会再一次出现在小艾的面前。

1.4.2　与人打交道——管理测试团队

在一个大型的软件开发项目里，完成开发任务的主体是整个团队，而非单独的个人。为了使团队有效运作，管理角色不可或缺。在测试团队中，管理的任务通常由测试组长和测试经理负责。区别于纯技术的职位更多地关注技术的细节，管理人员必须花相当多的时间与人打交道。作为测试管理人员，为了让团队以高效率运作，需要关注人力资源和合理分配，使测试工程师的技术优势得以发挥。在定义测试任务时，测试组长从宏观的角度考虑现有资源是否可以满足完成测试任务的需求；如果资源不足，是否可以通过沟通解决。测试经理还需要关注辖下组员的技术和职业发展诉求。测试管理者好比测试团队的大脑，“大脑”要完成对团队整体行动的决策和总体指挥。从分工角度来看，管理人员和技术专家的区别是非常明显的。

相对于一般的项目管理，测试管理需要更专注于风险控制和质量控制。项目出现变更时，测试进度就有可能受影响，进而有可能影响产品的最终质量，或者说，项目的风险有可能提高。管理人员应采取必要措施，控制项目的风险。从技术角度看，测试管理人员需掌握项目管理的相关技能。

随着测试工程师的经验累积，以及对相关测试技术和项目管理技术的掌握，到一定程度时，测试工程师就可以选择转向管理角色，成为测试组的技术组长或测试项目经理。技术组长在技术上至少能达到一般组员的水平，而工作上则体现出更强的技术领导力和影响力。因为技术组长关注的面更广，同时有更多和人相关的事情需要处理，因此对技术细节的关注度会不可避免地降低。测试项目经理则关注测试项目的进度和项目管理的细节。

测试组长的职位本身还是技术性质的，而测试组长的进一步发展方向又是什么呢？测试组长面临的选择相当丰富，在管理的道路上进一步发展，测试组长可以尝试测试经理或开发经理的职位，把更多的注意力投放在管理方面；从业务方向看，由于已经积累了比较深厚的测试专业知识和一定的业务知识，测试组长可以转而尝试业务方面的职位，如技术销售、技术服务等；如果希望继续从事专注于技术类的工作，测试组长可以基于某个测试种类深入钻研，成为这个测试类型的高级角色。

1.5 学习笔记——测试入门之小艾观

万事开头难，小艾作为菜鸟测试工程师加入到测试项目团队，以一名新人的视角学习了不少关于测试入门的知识。有了基本的知识及对测试专家广阔的发展前景的把握，小艾将有机会在测试的海洋开始一段新的旅途。学习之余，小艾也得到不少感悟。

- 要成为专业技术人员，花大量时间和精力学习专业技术是不可避免的。学习基本功是一个苦练的过程，但同样需要讲究学习方法。知识学习其实是体系性的，软件测试有一套完整的知识体系，把握知识体系的脉络，会让学习更加得心应手。
- 除了技术，对于测试新手而言，了解测试的业务和基本方法同样重要。测试是一个工程化的过程，测试在软件工程过程中必不可少，而且和项目的成败关系密切。测试新人既应该知道测试人员肩上的重担，同时也会以作为一名测试专家而自豪。
- 同是测试工程师，修为和级别可以有很大的差异。有不少“特性”会从测试专家和测试高手的身上显现出来。如何才能成为测试专家和测试高手？与其说这种修为是专业技能的提升，不如说是思考能力和思维深度的提升。“学而不思则罔”，作为测试工程师，思考（Think）是让学习成果升华的过程，同时也是解决问题所依赖的“必杀利器”。这种思维的能力不光测试专家应当具备，对所有专业技术人员来说都非常重要。
- “人无远虑，必有近忧”。虽说加入测试团队的时日很短，但作为测试新人，对职业生涯的考虑也是很重要的。测试工程师的职业发展有多种方向可供选择，根据个人的兴趣和特点，以测试工程师作为起步职位，可以尝试的职位也非常多。职业规划的最佳方向可能是：找到自己感兴趣的，同时又是能做得好的。小艾发现，工作也是生活的一部分，只有对感兴趣的事情，才乐意花更多的时间和精力去钻研。随着学习和工作的深入，测试其实是件既有趣又很有挑战的事情。
- 在软件开发团队，最宝贵的资源不是知识，也不是技术，而是人。团队中大多数同事都是乐于分享的人，共享和沟通的氛围对新人的入门很有好处，对于团队的长远发展和提高确实大有裨益。通过向身边同事的提问，小艾了解了许多书本和

文档中无从获取的信息，开阔眼界的同时，技术也明显提高了。小艾发现，这里从来不缺能回答问题的人，只要爱动脑筋，多提问题，总有意外的惊喜和收获。

第 9 章

黎明之前最后的冲刺：成品测试

项目结尾时是小艾最能感受到成就感时，总结一段时间以来的项目经验，总会有很多收获。小艾非常享受这种学习与收获的过程。但是，明明测试和开发已经完成了，距离计划发布日还有一段时间，这段时间要做什么呢？

9.1 产品包装成金蛋，手握光碟抓虫子

这两天小艾明显感觉到项目团队的气氛紧张。项目经理凯文和各团队主要负责人每天都神色凝重地在会议室激烈地讨论着什么。这又勾起了小艾的好奇心。

9.1.1 成品测试全体总动员

他问身旁的凯文：“现在我们的各方面测试都基本完成了，不是应该松口气庆祝了吗？

怎么领导们看着比以前还紧张，是出什么问题了吗？”

凯文拍了一下小艾的肩膀答道：“知道什么是黎明前的黑暗吗？这就马上到最后的冲刺——GMV 测试阶段啦。”

小艾一头雾水地问：“什么是 GMV？”

凯文看小艾很迷惑，就放下手头工作，给小艾认真讲解起来。

凯文的话：

GMV 是 Golden Master Verification Test 的简称，也就是我们通常说的成品测试或介质测试。它的测试目的是保证客户拿到的成品没有质量问题。从软件发布的角度来说，就是保障客户顺利安装并使用产品生产部门提供的光盘（CD 或 DVD）或网上下载的应用程序。

在 GMV 阶段构建团队会向测试团队提供 DVD ISO 文件和 DVD 光盘作为驱动，测试团队则使用这些介质来进行测试。GMV 的另外一个测试目的是保证产品在前期缺陷修复过程中不会因为代码改动而产生新问题。

因为 GMV 测试的重要性及其在进度上的紧迫性，GMV 测试阶段需要各个测试团队在已有的测试案例里有针对性地选择重要案例进行重新测试，以保证之前的代码改动不会为最终成品带来新的质量问题。切记不要在此阶段运行新的测试案例，以保证 GMV 能在合理时间内完成（一般 2～4 周）并成功交付给客户或投放市场。

小艾听了凯文的简单介绍，对 GMV 有了初步认识，但是对 GMV 具体计划和测试策略要点仍旧不甚了解。到底 GMV 需要怎么进行？而我们应该在 GMV 中做些什么呢？

小艾星期四刚到公司就接到项目经理发的通知，要求下午 2 点项目全体人员参加重要会议。

会议准时开始，凯文站在会议室前方扫视了下面一张张熟悉的面孔说：“这个时间召开此次会议的目的，我想在座的老员工都能猜出来吧？”

“不错，经过大约一年各团队的努力，我们已经到了最后的冲刺阶段。前两天和各团队负责人一起详细讨论了目前项目的进展情况。我很高兴地宣布明天将构建第一个成品候选介质！”

随后凯文站在放映机前，详细汇报了各测试团队到目前为止的测试结果。总体来讲，除了性能测试还有一些少量收尾工作，其他测试类别像功能测试、安装测试、产品迁移测

试、多国语言测试等都已顺利完成，测试用例都已 100%完成，通过率都达到甚至超过质量计划里所保证的百分比。失败的测试案例目前都有了解决方案，并在现有测试驱动上通过了补丁测试，按照计划，第二天下午 4 点前就能完成所有源代码改动。

会议最后，凯文凝重地说：“接下来两周的成品测试将是关键的一场战役，就像足球场上的临门一脚，关系着我们整个产品发布的成败。希望大家鼓足干劲坚持到最后的胜利！”

9.1.2　协同作战——成品测试特性

小艾听完凯文的讲话，对即将到来的成品测试充满了期待。但他还不是很清楚自己具体应该做些什么。会议结束后凯文找到小艾给他分配测试案例。小艾看了一眼测试案例，不解地问：“以往我们功能测试都是在已装好产品的机器上直接进行测试，这次为什么让我们自己安装产品呢？安装测试不是应该安装团队负责吗？”

凯文回答说：“你这个问题问得很好。由于成品测试的特性，成品测试有其独特的测试策略。它要求各团队协同作战，才能在较短时间完成所有测试任务。”于是凯文给小艾详细讲述了成品测试的特点。

项目经理凯文的话：

成品测试的一个主要目的是测试最终介质和包装有无缺陷，以保障客户拿到最终产品时能顺利安装并使用我们提供的光盘（CD 或 DVD）或网上下载的应用程序。对于不同操作系统平台或数据库，调用的安装程序和启动的包装有可能不同。这就要求在成品测试阶段，尽可能涵盖所有系统平台和数据库，以保障客户在不同系统上的正常应用。

对于数目繁多的操作系统平台（Window，AIX，Solaris，xLinux，pLinux，zLinux，iLinux 等）和数据库类型（DB2，Oracle，Cloudscape 等），安装团队是不可能在短时间内涵盖所有的，这就要求各团队协同合作。一般来讲，功能测试团队、构建团队、产品迁移测试团队和多国语言测试团队会用 DVD 或 DVD ISO 文件在较常见的操作系统平台上简单安装产品，并进一步进行功能或多国语言测试。安装团队会在此阶段着重测试一些复杂的安装路径和操作系统平台。

小艾听完凯文的解释，明白了成品测试阶段各团队协同作战的必要性。脱口而出道：“是不是人人都要手握光碟来抓虫子啊？”

凯文笑道：“比喻很形象，但我们除了传统物流方式销售给客户 CD 或 DVD 光碟，还允许客户通过网络下载我们的应用程序。一般会把 DVD ISO 文件经过压缩放到网络上供客户直接下载。所以除了测试真正的物理光碟，我们还要测试放到网上的可供客户下载的 DVD ISO 文件，即常说的电子版光碟。”

小艾点头说：“哦，我明白了。凯文，成品测试阶段测试范围是如何规定的？还有，它的测试策略又都有哪些呢？”

凯文说：“我很高兴看到你对成品测试的求知欲，我现在还要去构建团队安排一下工作。这样吧，我把各测试团队成品测试的计划发给你，那里面详细描述了各测试团队的测试范围和策略。你仔细看一下，有什么问题可以问我。”

9.1.3 取舍之间——测试范围和策略

小艾系统地阅读了凯文发给他的成品测试计划，并且根据各团队计划对成品测试阶段的测试范围和策略做了总体归纳总结。

成品测试范围及策略归纳总结：

- 成品测试的测试案例必须是以前测试阶段测试过的案例，不应该有新测试案例或对新系统平台设置的测试。
- 所被挑选的回归测试案例要尽量能够涵盖程序的主要功能，以确保程序的主框架没有由于前期代码改动而产生缺陷。
- 对于前期测试中发现较多问题并改动代码较多的功能部分，应多挑选一些回归测试案例进行回归测试。
- 性能测试一般选择最被广泛使用或者大型客户常用的平台。测试用例选择最简单的分支，但是尽量扩大分支覆盖的范围，一般选用可靠性测试（关于可靠性测试的定义见 6.2.4 节），测试时间一般在 24～72 小时。
- 所有测试都应基于 DVD 或 DVD ISO 文件安装的应用程序，严禁再用构建测试环境来安装应用程序并进行测试。
- 安装应尽量涵盖应用程序支持的所有系统平台（Window，AIX，Solaris，xLinux，pLinux，zLinux，iLinux 等），数据库类型（DB2，Oracle，Cloudscape 等）和安装模式（快捷安装，定制安装等）。
- 由于时间限制，成品测试案例大约占前期测试阶段所有测试案例的 5%～10%。

除了各测试团队详细的测试计划，在凯文所提供的材料中，还有一份在测试中要求各测试团队统一检查的清单列表。清单列表里主要需要检查下列内容。

a．DVD 布局

应按照产品构建团队提供的 DVD 布局文件里所列信息核对所测 DVD 光碟或电子光碟，布局一般包括：

- 不同系统平台的安装设置文件：setup.exe，setup_aix 等。
- 不同支持语言的说明文件（readme）。
- 自动调用文件（AutoRun）：autorun.exe，autorun.inf 等。
- 产品版本文件等一些和产品有关的文件。

除了核对布局里应存在的文件名外，还应该核对文件大小以保证文件没有损坏和缺失。整体 DVD 内容大小也是需要核对的一个数据。

b．产品安装

根据安装说明书和安装测试模式要求进行产品安装。

c．产品许可同意书

在安装过程中，检查产品许可同意书是否正确显示，并阅读上面所列内容是否正确。最重要的是要检查产品许可同意书里产品名称和版本号是否正确。

d．产品主要页面

安装成功后，根据测试的不同产品特性，浏览一下主要网页是否能够正确显示。

e．产品数据核心文件

产品中有些是产品信息核心文件，文件中的数据会被多个子程序引用。例如 Product.xml 中产品名称及版本号等。应检查这些文件是否安装在正确文件夹里，并打开文件，根据产品情况检查文件里面所列的信息是否正确。

f．产品构建号码

最终提供给客户的产品中一定要有产品构建号码并存储在一个产品文件中，以方便以后查询。这个产品构建号码一般以构建日期来命名。安装成功后，需要打开文件检查产品构建号码是否正确，以确定所测试的驱动版本是正确的。

g．数据库核心数据

一般产品都会有些核心数据存储在数据库里。可以根据情况列一些数据库查询语句在清单列表里，并要求各团队进行查询，以确保核心数据在安装过程中被正确引入到数据库中。

h．产品文档文件

对于每个产品来讲都会有文档文件，其中包括产品使用说明，产品问题帮助等。安装成功后，应检查一下这些文件是否被安装在正确文件夹里以确保能够被程序正确调用。

i．产品卸载

所有测试都结束后，应按照产品卸载说明书卸载产品。

9.1.4 争分夺秒——成品测试周期

GMV 测试开始当天小艾很早便来到公司，看到各团队负责人都已在办公室里。凯文递给他一张 DVD 光碟，说："这是构建团队周末烧刻的产品光碟，里面是上星期五的驱动。你今天的任务就是用 DVD 安装产品并完成上星期四分配给你的相应功能测试案例。所有测试必须在第二天下午 4 点之前完成，有什么问题吗？"

小艾问道："如果没有发现任何新缺陷，明天上午就应能完成所有案例测试。如果真是这样，我是否就再也不用在新驱动上测试了？"

凯文说："这需要视情况而定。就像我以前讲的，成品测试是各团队协同作战。一个团队的完成并不代表整体的完成。成品测试一般需要 1～2 周时间，对于每个驱动，大约需要 2 天完成。根据以往成品测试经验，大约需要构建 3～5 个驱动才能最终完成。对不同的驱动，由于周期短，时间有限，每个测试团队根据情况有自己的测试策略。并不是每个驱动都要重新测试所有案例。"

小艾稍后收到凯文的邮件，上面是各测试团队对不同成品测试候选驱动的测试安排。

功能测试：

- 在第一个成品测试候选驱动上 2 天内完成所有被选定的成品功能测试案例。
- 在接下来的成品测试候选驱动上，只运行功能测试可接受性案例和由于修正程序缺陷而可能受到影响的测试案例。
- 在最终项目组宣布的成品候选驱动上，重新运行所有被选定的成品功能测试案例。

安装测试：

- 在所有成品测试候选驱动上 2 天内完成所有被选定的成品安装测试案例（集群环境安装测试案例除外）。
- 在第一个成品测试候选驱动上，3 天内完成集群环境安装测试案例。在以后的成品测试候选驱动上如果没有和集群环境相关的代码改动，就无须重新运行集群环境安装测试案例。

构建测试：

- 在第一个成品测试候选驱动上 2 天内完成所有被选定的成品构建测试案例。
- 在接下来的成品测试候选驱动上，如果无 Java 代码改动，则无须重新运行 java API 扫描。但需重新运行源代码扫描案例、版权扫描案例及 SAR 扫描案例。

性能测试：

- 在第一个成品测试候选驱动上 3 天内完成所有被选定的成品性能测试案例。
- 在接下来的成品测试候选驱动上，如果无性能相关的代码改动，则无须重新运行性能测试案例。
- 在倒数第二个成品测试候选驱动上，重新运行所有被选定的成品性能测试案例。

迁移测试：

- 在第一个成品测试候选驱动上 3 天内完成所有被选定的成品迁移测试案例。
- 在接下来的成品测试候选驱动上，如果无迁移相关的代码改动，则无须重新运行迁移测试案例。

多国语言测试：

- 在第一个成品测试候选驱动上 2 天内完成所有被选定的成品多国语言测试案例。
- 在接下来的成品测试候选驱动上，只运行多国语言测试接受案例和由于修正程序缺陷而可能受到影响的测试案例。
- 在最终项目组宣布的成品候选驱动上，重新运行所有被选定的成品多国语言测试案例。

定制测试：

- 在第一个成品测试候选驱动上 2 天内完成所有被选定的成品定制测试案例。
- 在接下来的成品测试候选驱动上，只运行由于修正程序缺陷而可能受到影响的测试案例。
- 在最终项目组宣布的成品候选驱动上，重新运行所有被选定的定制测试案例。

小艾看后挠挠头说：“这么复杂啊，每个测试团队对不同的成品测试候选驱动的安排都不太一样，完成时间也不尽相同，怪不得你要作为总调度来统筹安排。”

凯文笑了笑说：“各个团队对不同成品测试候选驱动的安排是基于测试类别的不同特性，其实解读起来不外乎以下几点。”

凯文对不同成品测试候选驱动测试策略的解读：

- **各团队必须完全测试第一个驱动，以后的驱动需要审时度势，看编码改动情况来决定测试范围。**
- **由于每个驱动都需要重新构建打包，因此每个驱动都需要进行必要的安装测试和构建测试，以保障没有重要文件的缺失。**
- **项目组决定的最终成品候选驱动，将会是客户最终拿到的产品。各测试团队尽可能在此驱动上重新完成重要测试案例，以防功亏一篑。**
- **迁移测试和个别安装测试（例如集群安装）案例测试周期长，极少受由于其他类别测试缺陷而修改代码的影响。因此在整个成品测试周期一般只需要测试一遍。**
- **由于大部分测试都会在两天内完成，为了有效缩短整个成品测试周期，第二个成品测试候选驱动会在两天后开始构建，而不必等待迁移测试和个别安装测试（如集群安装）案例测试结束。**

小艾听了凯文的解读，觉得很有收获，深刻认识到成品测试周期的紧迫性，感觉各团队都是在争分夺秒来完成最后冲刺。于是他匆匆和凯文告辞，拿着 DVD 赶快到实验室开始执行分配给他的成品测试案例。

9.2　黎明前的黑暗——漏网之虫

坐在实验室的机器前，小艾用 DVD 光碟按照安装指示文档安装好产品，就开始运行所分配的功能测试案例。因为所分配给的测试案例都是小艾在测试第一和第二阶段运行过的案例，小艾感觉执行起来轻车熟路。一上午就完成了快一半，小艾暗暗松口气，为自己的进度感到高兴。

9.2.1　老案例生新虫子

中午吃过饭之后，小艾又急忙回到实验室运行测试案例。接近下午 3 点时，小艾在运行一个案例时发现了一个较严重的问题。小艾记得自己两周前在功能测试第二阶段运行过这个案例，当时没有发现任何问题。小艾怕自己记错，赶忙到数据库里查了一下这个测试案例运行的历史，发现最后一次成功运行的确是在两周前。

小艾于是怀疑最近两周这个功能有新的代码改动，为了进一步确定，他就又查了一下和这个功能有关的最近两周的缺陷修改历史，发现一周前开发团队的姚圳曾由于修复和这个功能有关的性能缺陷而修改过相关代码。由此，小艾基本肯定他新发现的功能问题是由于最近的代码改动引起的。于是他在代码管理系统里开了一个缺陷记录，标明是回归问题，并把自己的一些初步调查结果也注明在记录里，以方便开发人员参考。

考虑到问题比较严重，时间又很紧迫，小艾立即给姚圳打了个电话，把情况说了一下，并告诉姚圳可以参考他所开的缺陷记录来得到案例步骤及错误信息等详细情况。

姚圳听后说：“好的，谢谢你及时通知我，我会立即看一下。这个问题听起来很严重，而且对客户影响很大，需要马上解决。我会争取今天找到问题原因和解决办法。不知你明天能否早一点到公司，我需要你帮忙试一下代码补丁。由于现在是成品测试阶段，项目组要求所有正式源代码改动都需要事先在测试环境里通过案例测试和相关回归案例测试，测试完成后才能得到项目经理授权集成到下一个成品测试候选驱动里。”

第二天早晨小艾早早来到办公室，就看到姚圳发来的电子邮件。从电子邮件时间上看，姚圳一定为这个问题忙到了半夜。小艾不敢耽误，马上开始安装补丁并进行相应案例测试。为了保险起见，小艾又把和这个功能有关的一些主要案例运行了一遍，以防止又产生新的回归问题。

接近中午时，小艾终于顺利完成了补丁测试，他于是兴冲冲地跑去通知了姚圳。

姚圳松了一口气，笑着对小艾说：“太好了，我们在昨天下午 4 点的成品测试缺陷评判例会上已讨论过你发现的这个问题及它对客户的影响，项目组让我们两天内尽快解决这个缺陷。这下好了，我们提前一天完成了任务。我会参加今天下午 4 点的成品测试缺陷评判例会并汇报一下最新情况。如果不出意外，这个问题的源代码改动应集成在明天的成品测试候选驱动里。”

小艾很不好意思地说：“呵呵，我可知道你昨天为了解决这个问题熬到快半夜 1 点，你才是劳苦功高呢。姚圳，你是公司的老员工，参加了好多项目开发工作。你能告诉我成品测试缺陷评判例会主要有哪些内容吗？”

姚圳拍了拍小艾的肩膀说：“你可问对人了！ 我参加了好多次这样的例会，的确有一些了解。这个例会主要用来及时分析所发现的缺陷并根据缺陷影响给出解决方案。一般项目经理都会要求各开发团队代表、各测试团队代表、客户支持代表甚至产品补丁版本项目经理一起参加。在会上项目经理通过听取各方意见来决定缺陷的最终解决方案。”

小艾很疑惑地问道：“难道我们不应该通过及时修订代码，在把产品交付客户之前灭掉所有发现的虫子吗？”

姚圳回答道：“你说的是理想情况。实际来讲，成品测试阶段时间有限，所有测试都已基本接近尾声。如果这时改动大量代码，又没有足够时间进行必要回归测试，就很容易造成回归缺陷不被发现。这样反而会给客户造成更大损失。所以成品测试阶段有和测试前期阶段不同的灭虫策略。我们一般需要在此期间对发现的虫子进行综合分析，并根据对客户的影响和其紧迫性提出相应解决方案。”

小艾很期盼地问道：“根据这么多年的经验，你能告诉我一般都对虫子做哪些分析，相应地都有哪些解决方案吗？”

9.2.2 艰难抉择——漏网之虫综合分析及灭虫策略

在中午吃饭时间里，姚圳给小艾详细解说了成品测试阶段缺陷综合分析及相应灭虫策略。

缺陷综合分析要点：

- 缺陷是如何发现的。如果不是回归问题，为什么在测试前期没有发现，是否存在其他潜在的测试漏洞。

通过对虫子漏网原因的分析，能够更清晰地明白是否存在严重测试漏洞。修复此缺陷后是否需要增加回归测试范围以防再出现同样功能的回归问题。

- 有几种方法可以解决缺陷，每种方法的优缺点及客户接受程度。

例如某些缺陷是可通过多种方法解决的，但每种方法对代码架构的影响、对测试成本的影响和对客户的影响都不尽相同。需要平衡时间、成本及客户接受程度等要素来决定此时最佳解决方案。

- 缺陷修改对当前代码架构的影响，会影响到哪些测试团队。

例如有些代码改动可能需要功能测试团队，性能测试团队和安装测试团队重新运行一些测试案例。

- 受影响的测试团队需要多少时间和人力来完成由于代码修改而必须运行的测试案例包括回归测试案例。

通过测试成本计算，能够清晰地知道是否能有足够时间和人力在产品交付时间之前完成缺陷修复。

- 如果此缺陷不在当前版本里修改，会对客户和客户技术支持团队造成什么影响。

客户影响是非常重要的一个要素。有些缺陷被叫做“禁止交付”缺陷，顾名思义就是如果产品存在这些缺陷，则产品不能交付给客户。因此所有“禁止交付”缺陷一定要在产品交付前有解决方案。

另外，客户技术支持团队需要衡量一下，如果某缺陷不在当前版本修复，在当前产品版本推向市场后如果客户遇到相同问题，客户技术支持团队会需要额外付出多少人力资源来和客户沟通并解决客户问题。

- 客户对此产品缺陷的最大容忍时间大约多久。

通过分析客户对修复此缺陷的紧迫性来决定是在当前产品版本修复还是将缺陷修复推延到以后的产品新版本包括补丁版本、小版本或产品下一升级版本。

根据缺陷分析结果，一般会有以下解决方案：

- 在当前产品版本里修改缺陷，并按时交付客户。

例如：某些缺陷对客户影响大且缺陷修改对代码构架影响较小，测试团队能按期完成测试任务。

在成品测试阶段，所有缺陷修改必须经过项目经理同意并通过案例测试和回归测试后才能集成到下一个成品测试候选驱动上。

- 在当前产品版本里不修订缺陷，尽快在产品补丁版本或小版本里修改缺陷，并尽快交付客户。

例如一些产品功能客户在产品上线初期不会用到，和此功能有关的一些缺陷可以在随后发布的补丁版本里修复。

- 在当前产品版本里不修复缺陷，在下一个产品升级版本里修改缺陷。

例如有些缺陷对客户业务影响不是很大，只是在应用上需要一些改进而使客户应用起来更方便，对于此类缺陷，如果是在测试前期当然是尽可能修复。但在测试后期，尤其是成品测试阶段，就尽量不要因为此类缺陷改代码而引起不必要的回归问题。

- 在当前产品版本里修改缺陷，但需要推迟交付客户。

这种情况很少见。原因可能是前期测试存在重大漏洞，而存在的缺陷是客户不能接受的，但产品开发团队又无法在原定交付时间完成缺陷修复和相关测试。当然这种情况是每个产品项目都要想方设法避免的。

注意：如果缺陷修复需要推延到以后的产品新版本包括补丁版本、小版本或产品下一升级版本，除了需要得到开发团队、测试团队和项目经理的同意外，还需要得到客户技术支持团队的同意和支持，以便客户遇到相同问题时，可以和客户沟通。如果对此问题有临时解决方案，也需要第一时间通知客户支持团队以便在客户需要时提供给客户。

小艾在姚圳的详尽介绍中受益匪浅，他开始考虑或许这些缺陷综合分析方法不只在成品测试阶段需要，也可借鉴到整个测试阶段，尤其是测试后期。

9.3 金蛋闪亮登场

下午的测试进行得非常顺利，小艾在3点时就完成了所有案例测试和清单列表的检测，并把结果报告发给凯文。报告中也详细说明了缺陷产生的原因和缺陷修复进度，以及对姚圳所提供代码补丁的测试结果。

9.3.1　成品测试胜利退出

5 点多，凯文找到小艾说：“你报告的那个缺陷项目组已决定会在今夜的驱动里修复。明天早晨 10 点左右你应该就能拿到新的驱动。需要你在新驱动上重新运行和这个缺陷有关的一些案例并确保没有产生新的回归问题。”

小艾问道：“这会是最后一个成品测试候选驱动吗？”

凯文摇了摇头说：“很遗憾，不是最后一个。安装测试团队在卸载产品案例中发现了一个比较严重的问题，开发团队正在研究解决方案，目前来看代码改动无论如何也进不了今天晚上的驱动。最好情况是明天让安装测试团队测试一下代码补丁，确保没问题了再进下一个成品测试候选驱动。另外迁移测试团队的测试案例明天才能全部测试完成，现在不知是否会有新的缺陷产生。”

第二天，小艾按照所定计划完成了对缺陷修复相关案例的测试，再没有发现新的问题。小艾稍稍松了口气。

由于其他测试团队在新驱动里又发现了些问题，需要多个系统环境进行问题调查，小艾在接下来的几天里，被找去帮忙准备测试环境。凯文穿梭在不同团队之间协调人员和机器资源以加速整体测试进度，忙得不可开交。

所有人都忙而有序地工作着，等待着最后的胜利。

终于，凯文发出电子邮件给所有人员，计划将第四个成品测试候选驱动拟定为最终成品候选驱动，要求各测试团队按照计划对最后成品测试驱动进行最后一轮测试。

又经过两天紧张的忙碌，所有测试最终胜利完成，所有测试团队包括性能测试团队都相继发出电子邮件正式宣布测试完成，成品测试胜利退出。

凯文在接到成品测试胜利退出的喜讯之后，随即宣布确定第四个成品测试候选驱动即为最终成品驱动，并告知大家质量检查报告也刚已通过最终审批。按计划明天将最终 ISO 文件交付给生产商制造物理光碟和供网上下载的电子光碟。

整个办公室充盈着喜悦的气氛，经过近一年的努力，终于结出了胜利的果实，捧出了沉甸甸的金蛋，小艾由衷地为自己和团队感到骄傲和自豪。高兴之余，小艾对凯文信中提到的质量检测报告有些迷惑，于是他就去找凯文请教。

凯文一见到小艾，就拍着他的肩膀笑着说：“你在这个项目中成长了很多，表现不错。下一个项目你就是老员工，不再是菜鸟啦！”

小艾很不好意思地说：“要学的东西太多了，我也就刚入门。接触的事情越多，觉得要学习的东西就越多。这不又来向你请教了。”

凯文说：“你之所以能够成长很快，就是因为你有很好的求知欲。我很高兴能帮到你，有什么问题尽管说。”

小艾于是问道：“在你的邮件里提到的质量检测报告是怎么回事？我在测试阶段并没有接触到，你能给我简单介绍一下报告内容吗？”

凯文于是给小艾做了详尽的介绍，并把一些资料发给小艾供他参考和学习。

9.3.2 质量检测报告之大观

小艾认真阅读了相关材料，对质量检测报告有了很深的了解。

质量检测报告是项目中需要完成并得到审批的一个重要文件。在项目初期，项目经理需要和各团队负责人共同制定产品质量检测计划，其中大致包括：

1．项目特征

其中包括产品交付日期、项目大致介绍、项目大约所需人力物力数据、项目管理所需的几个关键日期、已知或潜在的开源产品介绍、开发流程特征等。

2．产品用户体验

需要给所开发产品的可用性、可靠性、安全性、集成性、维护性等方面制定一些具体指标，以保证用户在使用该产品时有良好的用户体验。

3．性能指标

需要制定一些性能方面的指标，客户可根据这些数据来配置硬件设施以期有效地应用产品，避免超负载、运行过慢等有害现象。

4．产品试用版本计划

需要指明产品是否有产品试用版本计划，如果有此计划，需要列明试用版本交付日期（一般要早于产品正式交付日期）和潜在试用客户名单。最后还需列明期望客户在哪些方

面（例如可用性、可靠性、功能性、维护性、安装性等方面）给予反馈信息以促进产品的改进和提高。当然在产品正式交付前，还应有个客户试用版满意度调查，以了解客户对产品的功能和质量的总体看法，从客户反馈方面看产品是否达到交付标准。

5．质量预见性指标

这项指标是产品质量保障的重要监测指标。通过这些指标的制定，可以对开发和测试环节的流程，方法及范围起到有效的监督作用。主要包括：

- **评审指标**：这项指标会通过判定相关人员是否评审了产品构架方案、设计方案、测试架构、测试方案、测试案例及程序源代码来进行质量把关。它会对每项评审列出详细评审方法。评审指标非常重要，通过专家一起评审，能够有效杜绝严重设计错误和测试漏洞。
- **测试指标**：这项指标会通过判定测试是否包括功能测试、全球化测试、性能测试、系统测试及回归测试来进行质量把关。

指标还包括缺陷发现计划进度和实际进度的比较，误差越小越好，最好小于某指定百分比（例如 10%），说明测试计划和实际相吻合，产品质量危险系数小。否则需要进行误差分析，看是否存有潜在的问题。

如图 9-1 所示，实际和计划进度误差在上下 10%之内。可以说测试基本是按计划进行的，没有太多出入。后期缺陷发现率显著下降最后趋于零，说明产品已趋于稳定，可以交付使用。

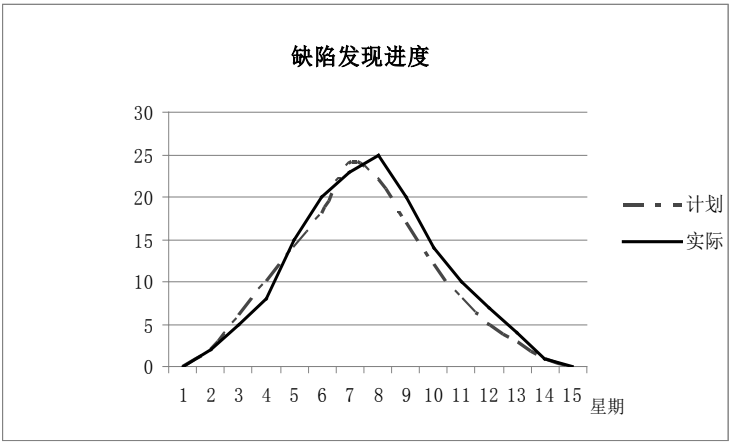

图 9-1　缺陷发现计划进度和实际进度的比较

测试指标还包括各测试类别测试案例尝试执行和测试案例成功完成的百分比。在产品质量检测报告里需要明确指出在测试计划里是否所有测试案例都能 100%尝试执行，也需

要明确预期在交付产品时各测试类别能最终成功完成多少测试案例。

一般情况下，如果交付产品时缺陷修复在 95%以上，所有测试案例成功率在 95%以上，可以认为产品质量是有保障的，存在问题的风险系数比较低。如果交付产品时各测试类别达不到 90%的测试案例成功率，这个产品的质量就会有较大隐患，以后出问题的风险比较高。[1]

- **项目退出指标：**在这项指标里会要求在交付产品里，所允诺的功能都完成了开发和测试。如果是升级版本，还会要求没有回归问题产生，即老版本中的功能在新版本中依然正常工作。
- **延缓缺陷指标：**我们在前面提到过（请参考 9.2.2 节），出于时间和安全性考虑，某些缺陷修复需要推延到以后的产品新版本包括补丁版本、小版本或产品下一升级版本。在延缓缺陷指标里一般会要求交付产品时，这些延缓修复缺陷不能超过总体发现缺陷的一定百分比（例如 10%），从而对产品所知缺陷率有总体控制，保证产品质量。在这项指标里一般也会硬性规定不能延缓修复任何严重缺陷。
- **源代码分析指标：**这项指标会评测源代码是否利用一些工具进行了静态代码分析、架构分析、代码测试覆盖度分析，用于从侧面判定产品质量是否有保障。

质量检测计划报告通过审批后，所列项目特征和指标就不能随便改了。如需改动，需要重新审批。

在项目测试完成后，需要把最终结果填写到报告中通过最终审批，拿到质量检测合格证书，才能交付产品。最终审批主要是看以下几个方面：

- 项目最终是否和计划的项目特征相符，是否按照计划进行开发流程。
- 产品用户体验的指标诸如可用性、可靠性、安全性、集成性、可维护性等方面是否达到要求。
- 所定性能指标是否能达到计划的标准。
  - 产品试用版本的客户满意度调查结果是否达到交付标准。
  - 是否按照计划评审了产品构架方案、设计方案、测试架构、测试方案、测试案例及程序源代码。

1　http://www.stevemcconnell.com/articles/art04.htm　Software Quality at Top Speed, Software Development, August 1996

- 缺陷发现计划进度和实际进度误差是否小于指定百分比（例如 10%）。
- 各测试类别测试案例是否 100%尝试执行，测试案例成功率是否能达到指定百分比（例如 95%）。
- 所允诺的功能是否都完成了开发和测试。如果是升级版本，是否没有回归问题产生。
- 延缓修复缺陷是否小于总体发现缺陷的指定百分比（例如 10%），而且无严重缺陷延缓修复。

9.3.3 趁热打铁总结经验教训

产品顺利交付了，各个团队都在组织讨论和总结经验教训。项目经理也发出产品调查表让大家填写，希望大家在评定产品质量如何的同时，进一步总结在开发和测试过程中好的经验和需要改进的方面，鼓励大家提出如何改进的建议以便在将来的项目中进行改善，从而使产品质量得到持续的提高。

产品调查表包括下列几个问题。

请选择你在项目中的角色：

(1) 产品架构师

(2) 产品开发人员

(3) 文档开发人员

(4) 测试人员

(5) 其他

请评估项目产品质量如何：(分为 1 级到 5 级，其中 1 级为最差，5 级为最好)

(　)5　(　)4　(　)3　(　)2　(　)1

请列举项目中你认为产品质量最好的部分：

请列举项目中你认为需要提高的部分：

请列举项目中你认为应该在将来的项目中继续执行的最重要的经验（包括影响力，相关团队，以及如何在将来的项目中执行）：

请列举项目中你认为应该在将来的项目中改进的最重要的部分（包括影响力，相关团队，潜在的问题或者原因，改进的建议）：

请列举你是否有其他的建议：

小艾根据自己测试的部分填写了自己对产品不同功能的意见和建议。并认真回顾了一下在整个项目中的所有测试历程，给团队和项目组提出了如下反馈：

应该在将来的项目中继续执行的最重要的经验：

（1）各团队之间总体有很好的沟通，有问题能够很快找到相关人员解决。

（2）测试计划和实际基本相符，开发及测试都能按计划时间完成。

（3）各测试类型团队之间没有壁垒，能够很快互相调动人力物力互相支持，以保障测试整体顺利按期完成。比如，功能测试人员帮助安装测试等。

应该在将来的项目中改进的最重要的部分：

产品功能上存在设计变动时，开发团队需要和测试团队提前沟通。测试团队需要根据新的设计来变动或添加测试案例。否则，测试团队会根据老的设计而开出无效缺陷，不但浪费了测试人员和开发人员的时间，还容易产生测试漏洞。

在产品测试阶段，任何人有需要安装产品时，也一定要严格按照安装文档来安装，如果发现文档里步骤不正确，要报告问题并要求相关团队修改文档。因为安装测试需要涵盖的范围太广，安装测试团队不可能涵盖所有情况。其他人可在力所能及的情况下，帮助安装团队发现问题。

项目组在收集到所有反馈后，会召开会议和各团队代表一起逐条讨论。对于有效反馈需要相关团队制定改进计划以便在今后的项目里实施。这些反馈里，既包括开发和测试流

程的改进建议，也有对产品设计和所用开发及测试工具的改进建议。总之，给大家一个平台可以让所有人畅所欲言，总结经验教训，并以此作为参考来更好地完善今后的项目计划和实施。

9.3.4　贯穿始终的缺陷分析

小艾完成调查表后，就去找凯文看是否有下一步任务。看到凯文正在画花花绿绿的表格，他就很好奇地问：“这是什么？”

凯文抬起头说：“我正在做项目最终缺陷分析，你来得正是时候，我正想给你详细介绍一下缺陷分析，好让你做个帮手。”

在凯文图文并茂地认真讲解下，小艾对缺陷分析有了深刻的了解。

在整个项目测试过程中，会不间断地做一些缺陷分析，来帮助监控在开发和测试中是否存在问题和漏洞，并根据分析结果来调整测试范围和策略。在最终测试完成后，会系统做一次缺陷分析，并以此总结经验教训以便在今后的项目中做出改进。

不同测试类型会有不同的缺陷分析侧重点。比如安装测试和迁移测试，会专注于分析产品安装在不同操作系统及数据库上出现的问题，所以会专门分析不同操作系统，不同数据库上出现的缺陷，看是否这个问题只发生在特定系统或数据库上。

一般来讲，所有测试类型都会对以下方面进行分析。

1．有效缺陷和无效缺陷的比率

前面的章节中详细介绍了缺陷管理模型中的缺陷生命周期状态及相关信息。 简而言之，缺陷从被发现到最终解决存在下列典型状态：**关闭状态，开启状态，工作状态，验证状态，退回状态，取消状态。**

其中，在关闭状态和取消状态下的缺陷称为完结缺陷，除此之外状态的缺陷称为活跃缺陷。项目结束时，理论上所有缺陷都应处在关闭状态或取消状态。当然由于时间原因，还存在极少量开启状态下的缺陷被延缓到补丁版本或新版本里修复。我们把关闭状态缺陷和延缓缺陷称为有效缺陷。而把取消状态下的缺陷称为无效缺陷。在测试过程中，应尽量提高有效缺陷比率，避免开出无效缺陷。

如图 9-2 所示，测试过程共发现 103 个缺陷，其中有效缺陷 85 个，占总缺陷的 83%，无效缺陷 18 个，占总缺陷的 17%。一般来讲，有效缺陷在 80%以上就算是比较良好的。

造成无效缺陷的原因大致有以下几点：

- 重复缺陷：是指在系统里相同原因的缺陷已经被其他人报告。在此缺陷被作为重

复缺陷返回时，先不要立即取消。必须等到核查修复后，才在系统里取消。这是因为有些缺陷被误认为是重复缺陷，实际上是不同原因造成的问题。我们只有在核查修复代码后，才能确认这是否是无效缺陷。

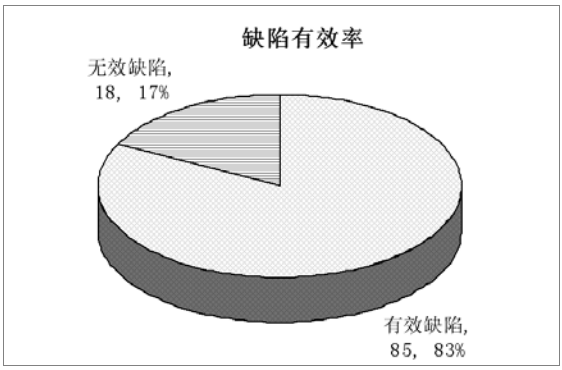

图 9-2　缺陷有效率分析

- 用法错误：是指测试人员在测试过程中有一些操作错误或对功能的错误理解而造成测试案例失败，由此开出的缺陷是无效缺陷。
- 符合设计：是指测试人员在测试过程中主观上认为测试结果是有问题的，并开出缺陷。但实际上产品设计就是应该如此表现。如果测试人员也同意如此设计是有道理的，就应取消所开缺陷，并视之为无效缺陷。
- 产品局限性：是指所开缺陷由于产品本身设计框架等局限性而无法修复。这样的缺陷通过改动代码是无法修复的，一般来讲最好转变为修改文档，通过文档来告诉用户这些局限性。如果既不能修改代码又不能修改文档，那么只好取消缺陷，并视之为无效缺陷。
- 未来功能：是指所开缺陷实际上是要求产品增加新功能。如果新功能不能在当前版本里实现，就应记录在未来版本的新功能候选名单上，并取消此缺陷且视之为无效缺陷。
- 不可重现：是指所开缺陷无法在相同场景中重现。有时可能由于网络，机器等问题而使测试案例无法正常运行，但重新测试很多遍也无法重现当时的问题。这种情况会建议取消缺陷，一旦同样问题再次发生，就需重开缺陷。

一般应该对无效缺陷进行分析，找出导致无效缺陷比例高的原因并提出相应的改进计划。

如图 9-3 所示，在 18 个无效缺陷里，重复缺陷占比例最高，其次是用法错误。所以就要进一步了解造成重复缺陷比例高的原因并予以改进。例如，可通过一些方法来提高测试团队之间的沟通，来有效降低重复缺陷。但有些重复缺陷是很难避免的，尤其是那些显示不同错误信息，但造成问题的代码是相同的缺陷。这些缺陷在很多测试专家眼里可以认为是有效缺陷。

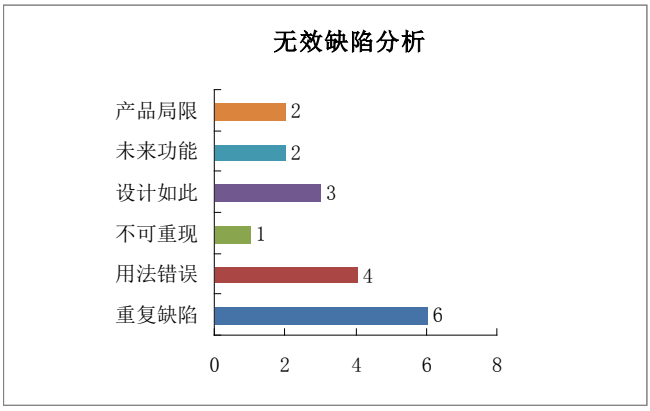

图 9-3　无效缺陷分析

用法错误的无效缺陷，在测试过程中应该尽量避免。这就要求测试人员要对测试案例有正确的理解而避免不必要的用法错误。另一方面，有些用法错误的缺陷可以转换成提高实用性的有效缺陷。

例如，小艾在安装测试时安装失败，小艾认为是安装代码的问题而报告了这个缺陷。开发人员在经过调查后发现是小艾不小心输入了错误的数据库密码而导致安装失败，就认为是用法错误而返回缺陷。小艾咨询了安装测试负责人，安装负责人认为虽然是测试人员用法错误，但从使用性上来讲，客户会遇到相同问题，产品应该能够在客户输入错误密码时提示客户，并禁止进一步安装。最终，开发人员接受了安装负责人的建议并修改了代码。而小艾所开的缺陷也成为提高使用性的有效缺陷。

2．按严重性分析缺陷

缺陷按问题的严重程度分为以下几个等级。

一级严重：是指缺陷造成整个应用或功能不工作，从而导致产品不能交付。

二级严重：是指缺陷造成功能输出结果错误。

三级严重：是指缺陷造成功能没有按预期方式实现。

四级严重：是指功能上需要加强。

在测试过程中，一级和四级严重缺陷应占比例较少，二级和三级严重缺陷比例应该占大多数。这是因为大部分一级严重缺陷在开发过程的单元测试中应该被发现和解决，而不应该流入到测试流程。

如图 9-4 所示，一级严重缺陷占 39%，二级严重缺陷占 33%，三级严重缺陷占 21%，四级严重缺陷占 7%。在此图中，可以发现一级严重缺陷的比例偏高。需要分析一下为什么会出现这种状况，是否有些缺陷应该在开发过程的单元测试中发现。应该把分析结果反馈给相关开发人员，并共同完善单元测试的测试范围，尽量让一级严重缺陷在早期发现并修复。

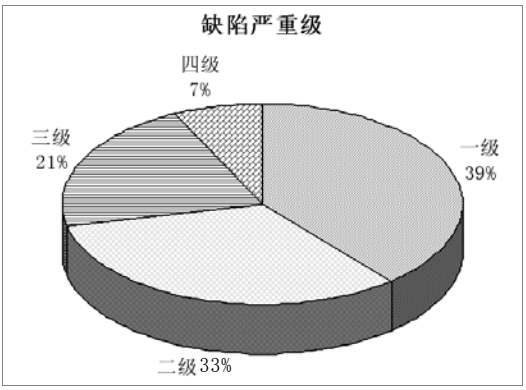

图 9-4　缺陷严重级分析

3．按功能分析缺陷

我们一般会看不同功能上发现多少缺陷。有些功能逻辑比较复杂，涉及的方面较多。这种情况所发现缺陷比例就会稍大。有些功能较简单，和其他功能没有太大交集，所发现缺陷就会相应较少。

如图 9-5 所示，功能 2 所发现缺陷占总数的 41%，而功能 4 仅为 7%。这说明功能 2 的复杂度相对大些，可能有必要再重新看看测试范围并增加一些测试案例。

4．按修复时间分析缺陷

一般来讲，一级严重缺陷需要开发人员一天之内解决，二级严重缺陷需要两天之内解决，三级严重缺陷要在一周内解决，四级严重缺陷应在两周内解决。

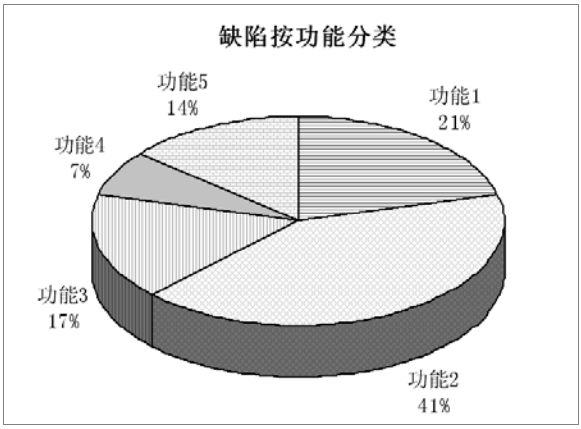

图 9-5　缺陷按功能分类分析

所以大部分缺陷从开始报告到最终核对并确认修复的周期应在一周内。不同测试类型缺陷修复时间是会稍微有一些差别的。例如性能测试的缺陷和迁移测试的缺陷由于测试环境较为复杂，并且较难找出造成缺陷原因，所以这两个测试类型的缺陷完成周期会相应长一些。

图 9-6 和图 9-7 分别是安装测试缺陷修复时间和性能测试缺陷修复时间数据图，从图中可看出，性能测试缺陷修复在 20 天以上的明显增多。这是由性能问题的复杂性决定的。依照经验，解决一个复杂的性能问题，有时候需要一个月甚至两个月的时间。所以对缺陷修复时间的分析要根据不同测试类别去进行。不同测试类别的缺陷修复数据可以用来作为今后各测试团队做项目测试计划的参考数据，例如性能测试要计划出足够的时间来核查缺陷修复。[1]

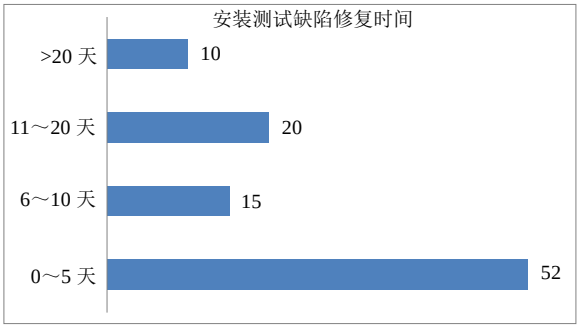

图 9-6　安装测试缺陷修复时间分析

1　出自《构建高性能 WebSphere 企业级应用》，2008 年，孙磊，孙静等编著。

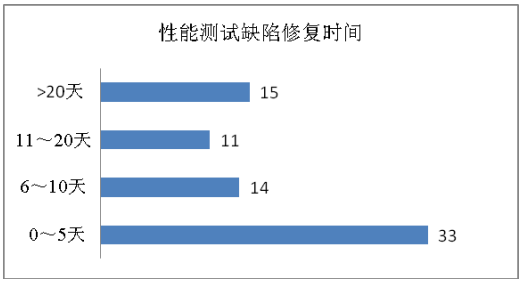

图 9-7　性能测试缺陷修复时间分析

5．按所属类别分析缺陷

理想情况下，各测试团队在测试时只发现自己所测类型的缺陷。但现实情况是所有测试类型虽然测试有先后，但也存在很大程度的时间重迭。

例如，由于性能测试的环境和数据要求很复杂，性能测试的测试成本远远大于功能测试的测试成本。所以性能测试一般会在其所需要的主要功能测试基本完成的情况下开始。但主要功能测试基本完成，并不代表此功能不存在缺陷。另外，主要功能在测试成功后，也可能由于后来的某些相关代码改动而存在回归缺陷。这就造成性能测试中不可避免地发现功能方面的问题。

通过对性能测试团队所发现缺陷的分析，可以统计出功能测试缺陷所占比例，看是否功能测试缺陷比例过高，是否严重影响了性能测试的进度，从而分析在今后的测试过程中，哪些测试流程需要提高而尽量减少性能测试中的功能缺陷。例如增强和功能团队的沟通，尽量了解性能测试中，每个测试案例里所需主要功能的进展情况等。

另外，性能测试需要安装和客户相近的测试环境，所以也会不可避免地发现安装缺陷。它的测试环境的复杂度也会帮助安装团队扩大安装测试范围。

如图 9-8 所示，在性能测试中，功能缺陷占大约 28%。这个比例稍微偏高，会对性能测试的进度有所影响。我们在分析过程中可以着重看看哪些功能缺陷是可以通过提高流程而避免的，并制定好相应计划，在今后测试中有所改进。

6．操作系统分析缺陷

如果是 Java EE 的应用程序，应该不会有太多操作系统相关的测试缺陷。但由于安装程序或多或少会和操作系统的设置文件有关，所以在测试中不可避免地会发现操作系统特定问题，例如某些缺陷只在 Solaris 上存在，在其他操作系统上没有问题。

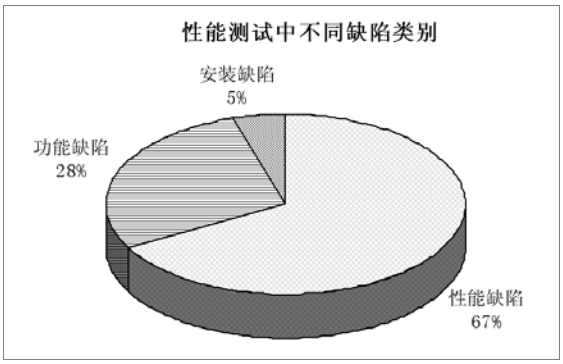

图 9-8　性能测试不同缺陷类别分析

通过缺陷分析，可以清晰地了解操作系统特定缺陷所占比例。如果比例不大，在测试中可以把大部分测试案例都放在机器资源相对充足、便宜且操作系统相对简单的环境里测试，以减少测试人员和机器资源的消耗。

如图 9-9 所示，安装测试中和操作系统无关的缺陷占 88%，所以大部分测试案例可在一种操作系统上进行测试。但因为其他操作系统也存在特定缺陷，所以在所有其他操作系统上，也要安排一些测试案例。

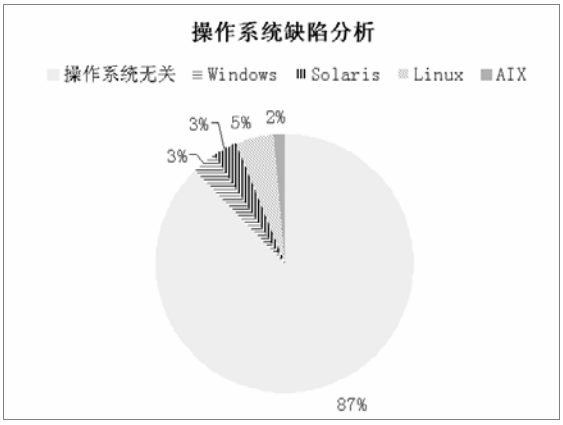

图 9-9　按操作系统分类缺陷分析

7．按数据库分析缺陷

现在绝大部分软件产品会需要数据库来存储并提取数据。不同的数据库处理数据的方式会有所不同。如果一个产品同时支持多种数据库，就可能存在同样的测试案例，在一种数据库上工作正常、但在另外数据库上出现问题的情况。

在测试过程中要尽量覆盖所有支持的数据库。如果有些功能对数据库不敏感，就只需在某一数据库上集中测试，在其他数据库上做些小测试即可。如果功能对数据库极度敏感，就要考虑不同数据库都要有足够的测试范围。

按数据库分析缺陷，可以相应看到哪些功能对数据库敏感且特定缺陷多，就需要考虑是否在此功能上增加不同数据库的测试范围。

8．按可用性分析缺陷

在测试过程中，不但要测试产品功能是否正确，还要看产品是否好用，即通过用户的使用角度来评估产品。由于它反映了用户的真实使用经验，所以可以视为一种不可或缺的可用性检验过程，例如界面是否友好、是否提供在线帮助，用户在使用错误的情况下是否显示容易理解的异常信息等。测试团队应鼓励测试人员尽可能多地发现可用性缺陷来提高产品的质量，从而使用户有非常好的产品使用经历。[1]

例如在安装产品过程中，如果操作系统不符合产品安装要求，就应该报出错误信息，并终止安装。如果测试过程中没有报错误信息，只是在安装中途失败，这就是一个可用性缺陷。在测试过程中，一定要从客户的角度来看问题。对于任何在使用时觉得不方便，不通用且容易造成用户困惑的功能，都要想想是否是可用性缺陷，是否应该进行修订而提高其易用性。

例如在网上商店注册时，会要求用户输入很多个人信息，例如姓名、年龄、职业、住址、家庭状况等。有些是必填项，有些是自愿填的。假设在输入信息时忘记填写必填的一项，在单击“完成”按钮时，如果系统只是提示你有未填项目，并要求你重填所有信息，这对于用户来讲是非常不能忍受的。这种情况就应该开出可用性缺陷。

可用性好的产品会这样处理：系统提示你有一项没填，并把未填项标志出来方便你继续填写。其他已填好信息依旧保存在页面上。这样既方便又快捷地帮客户解决了问题，从而使用户得到非常好的使用体验。

测试过程中可用性缺陷的多少能够在一定程度上衡量测试人员的专业性和产品质量的高低。一般产品可用性测试没有专门事先写好的测试案例，更多的是依靠测试人员的测试经验和对产品的熟悉程度。如果整个测试过程中，只有很少甚至没有可用性缺陷，就要好好分析一下原因，看是否是因为测试时间紧张，测试人员只是着重测试了产品的正确性，而忽略了产品使用性的测试。

1　Jakob Nielsen，《Usability Engineering》，1994 年

9．按照其他指标分析缺陷

缺陷分析的指标多种多样，每个团队的侧重点存在一些差别。我们在前面列举了几种分析缺陷的主要方面，除此之外，还有许多指标可以用来做缺陷分析。

例如把缺陷的创始人作为指标来分析，可以看到哪些测试人员所发现的缺陷数量最多，有效性最高。一般来讲，同样的测试类型，有经验的测试人员平均比无经验的测试人员发现缺陷多且有效性高。缺陷的拥有者也可以作为指标来分析，从中可以看到哪些开发人员在整个项目中所修复的缺陷最多。

总之，缺陷分析是项目测试时和完成后都需要重视的一项任务。它既能帮助测试人员随时调整测试范围和侧重点以防患于未然，又能很好地帮助总结经验教训，以便于今后项目的提高和发展。

9.4　学习笔记——成品测试之小艾观

小艾通过成品测试及成品测试之后的经验总结学到了很多东西：

成品测试的一个主要目的是测试最终介质和包装有无缺陷，以保证客户拿到最终产品时能顺利安装并使用我们提供的光盘（CD 或 DVD）或网上下载的应用程序。成品测试另外一个测试目的是保证产品在前期缺陷修复过程中没有因为代码改动而产生新问题。

成品测试阶段有其独特的测试策略和灭虫方案：

- 在已存在的测试案例里挑 5%～10%重要案例来重新测试以保障以前的代码改动没带来新的质量问题。所被挑选的回归测试案例尽量能够涵盖程序的主要功能，以确保程序的主框架没有由于前期代码改动而产生缺陷。
- 尽量不要在此阶段运行新的测试案例，用于保障成品测试能在合理时间内完成（一般为 2～4 周）并成功交付给客户或投放市场。
- 由于不同操作系统平台或数据库调用的安装程序和启动的包装有可能不同，所以在测试中各测试团队要协同作战，尽可能涵盖所有系统平台和数据库，以保障客户在不同系统上的正常应用。

- 所有测试都应基于 DVD 或 DVD ISO 文件安装的应用程序，严禁再用测试驱动来安装应用程序并进行测试。
- 由于成品测试是测试的最后一个环节，且周期很短，因此就要求代码改动要特别慎重，以防引起回归问题。成品测试阶段项目组一般要每天审核所发现缺陷并做缺陷综合分析，根据分析结果制定相应灭虫策略，有些缺陷可延缓到以后修复。对于要修复的缺陷，必须要有足够的回归测试，以保证代码改动没带来新的质量问题。

成品测试之后，需要把最终结果填写到质量检测报告中并通过最终审批，产品才能最终交付客户。质量检测报告是项目中需要完成并得到审批的一个重要文件，其中包括下面几个方面：

- 项目特征；
- 产品用户体验；
- 性能指标；
- 产品试用版本计划；
- 质量预见性指标，包括评审指标、测试指标、项目退出指标、延缓缺陷指标、源代码分析指标。

在产品交付后，需要对项目做经验总结，同时各测试团队要对所有缺陷做最终缺陷分析，以此总结经验教训，以便在今后的项目中做出改进。不同测试类型会有不同的缺陷分析侧重点。一般包括缺陷有效性分析、缺陷严重性分析、不同功能上发现的缺陷分析、按修复时间分析、按缺陷类别分析、按操作系统或数据库分析、按可用性分析等。

缺陷分析应该贯穿整个项目测试过程，而不仅仅是在测试完成后。通过不断进行缺陷分析，来监控在开发和测试中是否存在问题和漏洞，并根据分析结果来调整测试范围和策略，防患于未然。

参考资料：

[1] Software Quality at Top Speed, Software Development, August 1996. http://www.

stevemcconnell.com/articles/art04.htm

[2] 孙磊，孙静等.《构建高性能 WebSphere 企业级应用》. 2008.

[3] Jakob Nielsen.Usability Engineering.1994.