 从菜鸟到测试架构师

『 从 菜 鸟 到 测 试 架 构 师 』 编 委 会 编 著

電子工紫出版社 ·

PUBLISHING HOUSE OF ELECTROMICS INDUSTRY

<http://www.phel.com.cn>

孙磊张明明审校

***Broadview°***

[***www.broadvlew.com.cn***](https://www.broadvlew.com.cn)

热

P

**从菜鸟到测试架构师**



**一个测试工程师的成长日记**



——

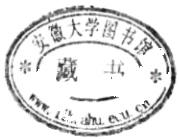
《从菜鸟到测试架构师》编委会编著

孙磊张明明审校

编委会主任：邓佰臣

编委会副主任：张艳梅王俊华邓郁敏

编委(以写作章节排序):邓郁敏潘家铭李强傅群姜丽丽邓佰臣 孙静王蕊郑传杰王俊华张艳梅喻翔冯庆华赵松林

) ·

**電子工業出版社** **·**

**Publishing House of Electronics Industry**

**北京** **·BEIJING**

**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。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

**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》编委会编著.—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 一测试 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：三河市双峰印刷装订有限公司

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。

序

我们所生活的时代是一个信息技术高速发展的时代，信息化产业已成为社会发展的巨 大推动力。这其中，各类软件产品在人们的生活与工作中扮演着越来越重要的角色。随着 人们越来越多地依赖于这些软件产品，软件质量不可避免地成为关注的焦点。

任何功能性或者非功能性的缺陷，不但会影响软件的用户体验，甚至有可能会直接影 响到人们的生命财产安全。因此，软件质量监管一直是软件研发过程中的一个重要环节， 也是提升企业产品竞争力的关键因素。作为全球500强企业，IBM 始终以成就客户为核心 价值，通过自身的不懈努力为客户提供高质量的软件产品和服务。众所周知，IBM 的软件 以其高质量、高稳定性著称，是客户最信任的合作伙伴。IBM 的测试方法和流程也一直为 业界所津津乐道。我很高兴看到这些成就，更深知这其中包含了我们研发工作者的大量心 血。我希望看到我们的同事能将这些知识和经验分享给更多的软件研发工作者，帮助这 些同行建立起更完善的软件质量检测和控制体系，以及更精准高效的软件测试策略和方 法，这本书的目的正在于此。本书的作者团队成员都是在软件质量与软件测试领域具有丰 富经验的专家，他们以IBM 的电子商务平台- IBM WebSphere Commerce为产品背景， 把他们丰富的软件测试实战经验融汇贯通于本书的各个章节之中。

本书从一位新入职的测试菜鸟——小艾的视角出发，伴随着小艾的成长，为读者全方 位地详细介绍了软件测试的策略、流程和方法，涉及构建测试、安装测试、单元测试、黑 盒测试、白盒测试、性能测试、迁移测试、成品测试、补丁测试等各个阶段，更包含自动 化测试、敏捷测试等诸多方面。整本书由浅入深地为读者展现了一个测试工程师从菜鸟到 测试架构师的成长经历，其中既有丰富的理论知识，也有详尽的实践经验，以及系统的要 点总结，引导读者一步步深入了解这个看似枯燥实则丰富多彩的软件测试的世界。

我很高兴将这本书介绍给各位读者。希望在阅读完小艾的成长经历之后，读者能够更

加深刻地理解软件质量检测和软件测试，并能够将本书的理论与实例应用到日常的软件研 发工作中，进而更加有效地提升软件产品的质量。也希望本书能成为测试人员职业发展道 路上的参考指南。

王阳博士

IBM 全球副总裁兼中国开发中心总经理

**前** **言**

**写作背景**

IBM 是一家全球著名的信息技术和业务解决方案公司。历经百年的发展，IBM 始终致 力于为客户提供高质量的服务及产品。IBM 的软件遍布我们工作和生活的各个角落，不论 是公共关系数据库还是在线交易处理，乃至面向企业的新的社会网络功能等应用，在信息 管理、业务整合、企业协作、系统管理等各个方面为人类服务。进入软件行业十几年来， IBM 始终认为质量是软件的灵魂与核心，其软件也以高质量、高稳定性著称。

20世纪至21世纪，信息产业影响并改变着整个世界，软件的应用也渗透并改变着人 类的生活方式。我们的娱乐、生活、工作都与软件的应用密切相连。同时软件质量也成为 行业内备受关注的话题。1968年 NATO 会议中提出的“软件危机”中指出：人们依赖于 脆弱的软件，目前的软件不可靠、缺乏安全性、性能下降、出错、难以升级，73%的软件 项目被延迟、超资、取消或失败。随着近年来软件行业的发展，人类对于软件的需求愈发 庞大，软件的功能需求愈发复杂，软件质量问题也愈发成为软件行业的核心。

如何确保软件质量、减少质量问题给企业及用户带来的隐患，是软件测试工作的核心。 软件测试工作者不仅需要确保软件的需求一致性，更加需要确保软件从安装到功能使用的 正确性、系统的健壮性、运行的效率等，以及功能的可理解性、系统的可维修性、灵活性、 可测试性等。可以说，软件质量是一个庞大的体系，软件测试是一个复杂的过程。

IBM 始终坚持认为作为信息技术和业务解决方案提供者，应尽可能地向客户及企业提 供令其满意的、强劲而完善的软件产品，并始终致力于软件质量测试监督体系的完善。为 此 ，IBM 投入大量人力与技术支持软件测试工作。

本书的作者主要来自 IBM 中国开发中心，他们具有雄厚的技术根基和多年的软件测 试工作经验，涵盖了测试的各个领域，包括安装测试、功能测试、性能测试、自动化测试 及构建测试等，确保软件从安装到使用乃至后期维护的稳定性和健壮性。他们的工作中测 试范围广泛，对于测试的理解具有一定的深度。然而在多年的测试工作中，他们发现，随

着软件行业的发展，关于测试的理论及实践都愈发完善，但是关于测试方法和体系却缺乏 完整性的讨论。因此，他们希望能够将自己多年来的经验、知识与广大软件工作者分享讨 论，帮助软件测试者更好地进行测试工作，提高软件质量，让人类能够更好地利用信息技 术为自己的工作和生活服务。

这本书从准备初期至完稿经历了漫长的过程，作者们期待以一种循序渐进而系统的方 式探讨软件测试方法和体系。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。

**本书结构**

本书章节编排如下。

第1章：概括介绍了测试基础理论知识。本章从基础理论出发，通过讨论测试基础理 论知识、测试人员的职业发展，从广度和深度上对测试工作进行了概括，给予读者对于测 试体系的基本而全面的认识。

第2章：介绍了测试过程中的安装测试的方法及理论。本章从软件安装开始，深入讨 论了安装测试，从测试之初体验到安装测试高手，从手动安装测试到自动化安装测试，都 分别进行了详尽的讨论。

第3章：介绍了测试过程中的构建测试的方法及理论。本章以代码管理为切入点，简 述了在大型软件开发中的代码管理及系统构建测试，并简要讨论了测试中的静态测试。

第4章：介绍了测试过程中白盒测试的方法及理论。本章从虫子的摇篮——代码入手， 探讨了单元测试的重要性，并着重介绍了单元测试的理论和方法，然后，针对测试驱动开

发理念进行了简要的介绍。

第5章：介绍了测试过程中黑盒测试的方法及理论。本章首先阐述了黑盒测试的基本 理论，继而介绍了功能模块分解分析，从而由浅入深，深入讨论黑盒测试的方法和理论。 本章还对回归测试、客户化环境测试、随机测试、可访问性测试、全球化测试等测试的林 林总总进行了简要的介绍。

第6章：介绍了测试过程中的性能测试的方法及理论。本章以性能测试的重要性为切 入点，首先从测试方法上讨论了性能测试，然后论述了系统监视理论及方法，以帮助读者 从系统到应用服务器乃至数据库等各个角度分析性能问题，并深入讨论了性能问题分析定 位策略。

第7章：介绍了测试过程中迁移测试的方法及理论。本章首先介绍了迁移测试的基本 概念，继而详细讨论了迁移测试的基本方法，包括：垂直迁移测试方法、水平迁移测试方 法，并对迁移测试的计划和实践进行了简要的阐述。

第8章：介绍了测试过程中的客户化测试的方法及理论。本章首先介绍了客户化测试 的基本理论。并通过代码开发、代码部署、联合测试三个步骤阐述了客户化测试的方法和 理论。然后详细论述了客户化测试过程中对于产品推进的方法和过程。

第9章：介绍了测试过程中的成品测试的方法及理论。本章站在成品测试人员的角度， 对成品测试进行了讨论和分析，在内容上包括成品测试的特性、测试范围、测试周期，并 对于成品测试后的测试分析进行了详细的论述。

第10章：介绍了测试过程中的补丁包测试的方法及理论。本章首先介绍了补丁包测 试的基本概念，然后深入讨论了补丁包测试的基本流程，并简要介绍了补丁包测试过程中 的其他流程及重要影响因素。

第 1 1 章：介绍了测试过程中的自动化测试的方法及理论。本章以自动化测试的基本 概念和自动化测试的利与弊为切入点展开论述，继而讨论了自动化测试框架、自动化测试 策略，并结合实际对自动化测试的应用进行了讨论。

第12章：介绍了测试过程中的敏捷测试的方法及理论。本章作者站在一个敏捷测试 初学者的角度进行论述，从初识敏捷测试开始，以测试现状为出发点，通过讨论传统测试 到敏捷测试的转型，深入讨论了敏捷测试的方法及理论。

第13章：本章全面系统地讨论了软件测试方法及理论。从测试规划、测试发展、测 试过程的设计和改进等各个角度，以一定的高度总结和概括了测试体系，完善和总结了作 为测试架构师的高级测试人员的测试理念。

**致谢**

本书是集体创作的结晶。首先要感谢每一位作者为本书所付出的心血。本书的创作占 用了各位作者大量的休息、娱乐时间、本应和家人共享的假日。所以，特此感谢各位作者 为本书的付出，感谢各位作者家人的理解和支持。

感谢 IBM 中国开发中心的资深软件工程师孙磊、IBM 中国开发中心的资深软件工程 师张明明女士等审阅本书的内容并提供了宝贵的修改意见。他们的建议和意见为本书的完 成作出了巨大的贡献。感谢 I B M 著书出版协会的刘慎锋、岳治宇，感谢他们对本书的出 版过程给予支持和推进。此外，特别感谢 IBM 的同事栾忠世、王月在本书第5章中给予 的帮助。

本书的成书过程中得到了各位领导的关怀、鼓励和支持。其中包括 IBM 全球副总裁 兼中国开发中心总经理王阳博士、IBM 全球行业解决方案开发总监沈丽琴女士、IBM 智慧 电子商务部开发总监崔峰女士、高级经理王克非先生等，在此一并表示感谢。

本书在写作过程中参考了大量的文献，在此对这些文献的原作者一并表示衷心的感谢。

最终，是电子工业出版社博文视点公司和 IBM 中国开发中心的联合出版活动促成了 本书的出版。在此要感谢出版社的有关领导、工作人员及编辑，没有他们的支持和参与， 这本书不可能如此顺利地出版。

**免责声明**

本书中提到任何国际商业机器公司(以下简称 “IBM”) 的产品、程序或服务，均不 表示或暗示只有 IBM 公司可以提供。

本书以 IBM 的部分产品为例帮助读者更好地理解测试方法和理论，其中所提到的一 些产品始终在进行新功能的开发及完善，所以针对本书所提到的IBM 的产品功能、界面， 并不保证用户将来购买到的版本与本书所描述的完全一致。

为了内容介绍的方便，本书提到了一些产品名称、人名、网站名称等，这些仅仅是作 者为了内容完整性的需要而提及，不代表 IBM 对这些产品、人员、网站有任何正面或者 负面的评价。

本书不带有任何明示或暗示的保证。本书中任何带有主观性的陈述都只代表本书作者 的个人观点，不代表 IBM 公司的官方立场。本书的内容有可能不太准确或者包含错误， 作者对此深表歉意。

**目** **录**

[**第** **1** **章** **上** **班** **第** **一** **天** **，** **新** **人** **培** **训** 1](#bookmark1)

[1.1 测试专家的第一步 1](#bookmark2)

[1.1.1 我是菜鸟 1](#bookmark3)

[1.1.2 苦练基本功 4](#bookmark4)

[1.1.3 培养专业技能 9](#bookmark5)

[1.2 开发团队做的远不仅是开发 13](#bookmark6)

[1.2.1 术业有专攻 13](#bookmark7)

[1.2.2 好软件由测试决定 17](#bookmark8)

[1.2.3 测试也有大学问 18](#bookmark9)

[1.3 从专家到高手 25](#bookmark10)

[1.3.1 像外行一样思考，像专家一样实践 26](#bookmark11)

[1.3.2 工欲善其事必先利其器 28](#bookmark12)

[1.3.3 从拿来主义到创新 31](#bookmark13)

[1.3.4 测试的广度和深度 32](#bookmark14)

[1.3.5 无招胜有招 3](#bookmark15)

[1.4 职业生涯的考虑——技术还是管理 35](#bookmark16)

[1.4.1 测试工程师的技术发展路线 35](#bookmark17)

[1.4.2 与人打交道——管理测试团队 37](#bookmark18)

[1.5 学习笔记——测试入门之小艾观 38](#bookmark19)

[**第** **2** **章** **万** **事** **开** **头** **难** **：** **软** **件** **从** **安** **装** **开** **始** 39](#bookmark20)

[2.1 软件，是装出来的 40](#bookmark21)

[2.1.1 安装测试概念解读 40](#bookmark22)

[2.1.2 测试之初体验——一份手册，N 台机器，一堆软件 41](#bookmark23)

[2.2 全面撒网，重点排查 43](#bookmark24)

[2.2.1 选择测试配置 43](#bookmark25)

[2.2.2 找出测试场景 46](#bookmark26)

[2.2.3 组合出测试用例 4·8](#bookmark27)

[2.3 安装测试质量之大观 48](#bookmark28)

[2.3.1 产品安装 49](#bookmark29)

[2.3.2 产品卸载 52](#bookmark30)

[2.4 客户的圣经——用户手册验证 53](#bookmark31)

[2.4.1 一般验证方法 54](#bookmark32)

[2.4.2 文档审阅流程 54](#bookmark33)

[2.5 引进先进设备——安装自动化测试 55](#bookmark34)

[2.5.1 效率的提高从自动化开始 55](#bookmark35)

[2.5.2 自动化测试的实现 56](#bookmark36)

[2.6 学习笔记——安装测试之小艾观 60](#bookmark37)

[参考资料： 60](#bookmark38)

[**第** **3** **章** **发现新方法：快速分发新的测试版本** 61](#bookmark39)

[3.1 构建的过程 62](#bookmark40)

[3.1.1 构建原材料的管理——代码的管理 63](#bookmark41)

[3.1.2 构建的环境 65](#bookmark42)

[3.1.3 整体构建和部分构建 67](#bookmark43)

[3.1.4 自动化的构建 69](#bookmark44)

[3.1.5 模拟翻译构建 70](#bookmark45)

[3.1.6 构建的范围和频率 71](#bookmark46)

[3.2 构 建 测 试 (Build Verification Test - BVT) 73](#bookmark47)

[3.2.1 构建测试的配置(BVT Configuration) 75](#bookmark48)

[3.2.2 构建测试的用例(BVT Scenarios) 75](#bookmark49)

[3.2.3 自动化的构建测试 76](#bookmark50)

[3.2.4 构建测试的环境再利用 76](#bookmark51)

[3.3 静态测试 7](#bookmark52)

[3.3.1 静态测试的作用及环境 78](#bookmark53)

[3.3.2 自动化的静态测试 78](#bookmark54)

[3.3.3 静态测试的频率 79](#bookmark55)

[3.4 不间断的构建与测试(Continuous Integration) 79](#bookmark56)

[3.5 学习笔记 — — 小艾的构建测试观 80](#bookmark57)

[参考资料： 81](#bookmark58)

[**第** **4** **章** **把缺陷扼杀在摇篮里：开发人员的测试** 82](#bookmark59)

[4.1 怪物来自缺陷，开发人员也需要做测试 82](#bookmark60)

[4.1.1 深入摇篮内部 83](#bookmark61)

[4.1.2 吃一堑长一智，走出单元测试的误区 84](#bookmark62)

[4.1.3 潘多拉盒子出来怪物——开发人员眼中的测试 89](#bookmark63)

[4.2 单元测试测点啥，怎么测 91](#bookmark64)

[4.2.1 摇篮有多大——单元测试的范围 91](#bookmark65)

[4.2.2 有规范、有步骤地捉虫子——单元测试的流程 92](#bookmark66)

[4.2.3 来一套杀虫装备：单元测试的工具 94](#bookmark67)

[4.2.4 单元测试够不够，我来把把关 9](#bookmark68)

[4.3 测试还能驱动开发(TDD) 101](#bookmark69)

[4.3.1 什么是测试驱动开发，测试驱动的工作流程 102](#bookmark70)

[4.3.2 测试驱动开发好处多 105](#bookmark71)

[4.4 学习笔记——单元测试之小艾观 107](#bookmark72)

[参考资料： 108](#bookmark73)

[**第** **5** **章** **黑** **色** **盒** **子** **：** **打** **着** **手** **电** **抓** **虫** **子** 110](#bookmark74)

[5.1 黑色的盒子里面有什么 110](#bookmark75)

[5.1.1 小艾的功能测试第一课——准备手电 11](#bookmark76)

[5.1.2 导师对功能测试的深入浅出的解读 17](#bookmark77)

[5.1.3 参与实际测试执行，编写自动化测试脚本 130](#bookmark78)

[5.1.4 找到了第一条虫子 132](#bookmark79)

[5.1.5 手记——执行高手 133](#bookmark80)

[5.2 如何把黑盒子分块 134](#bookmark81)

[5.2.1 如何精准找寻某一种虫子——分而治之 136](#bookmark82)

[5.2.2 客户的反馈——虫子依然存在吗 136](#bookmark83)

[5.3 对黑盒子的全方位照明 137](#bookmark84)

[5.3.1 找到了小盒子间的虫子—合而治之 137](#bookmark85)

[5.3.2 手记——策略高手 139](#bookmark86)

[5.4 功能测试之百种变身 141](#bookmark88)

[5.4.1 回归测试 141](#bookmark89)

[5.4.2 客户化环境下的测试 142](#bookmark90)

[5.4.3 随机测试(Adhoc Test) 143](#bookmark91)

[5.4.4 可访问性的测试(Accessibility Test) 144](#bookmark92)

[5.4.5 全球化测试(Globalization Test) 144](#bookmark93)

[5.4.6 其他之林林总总 145](#bookmark94)

[5.5 学习笔记——功能测试之小艾观 146](#bookmark95)

[参考资料： 147](#bookmark96)

[**第** **6** **章** **大促带来的灾难：究竟能支撑多少访问量** 148](#bookmark97)

[6.1 洪水般的访问量——初识性能测试 148](#bookmark98)

[6.1.1 一旦宕机，流失的就是白花花的营业额 148](#bookmark99)

[6.1.2 假如我是甲方——关于性能客户都关注些什么 149](#bookmark100)

[6.2 让访问来得更猛烈些吧——模拟客户的访问行为 151](#bookmark101)

[6.2.1 更真实更高效的模拟——自动化的性能测试 151](#bookmark102)

[6.2.2 春节大促——压力测试 153](#bookmark103)

[6.2.3 日常的访问量——正常的响应时间 155](#bookmark104)

[6.2.4 保证长时间的稳定运营——可靠性测试 156](#bookmark105)

[6.2.5 客户的成长不比产品慢：想象不到的数据量——可扩展性测试 157](#bookmark106)

[6.2.6 摸爬滚打过来还得挨一刀——产品经历性能测试 158](#bookmark107)

[6.3 像 F B I那样缜密——全面监视系统的一举一动 167](#bookmark108)

[6.3.1 操作系统的监视 169](#bookmark109)

[6.3.2 应用服务器的监视 170](#bookmark110)

[6.3.3 数据库的监视 171](#bookmark111)

[6.4 大海捞针也要讲究策略——性能问题的分析定位 179](#bookmark112)

[6.4.1 磨刀不误砍柴功—了解性能问题定位的一般策略 180](#bookmark113)

[6.4.2 越忙越乱——高并发访问量下的出错率 182](#bookmark114)

[6.4.3 越跑越慢——吞吐量下降问题 185](#bookmark115)

[6.4.4 穿了小两号的鞋——通过参数调优能解决的问题 187](#bookmark116)

[6.5 学习笔记——性能测试之小艾观 189](#bookmark117)

[参考资料： 189](#bookmark118)

[**第** **7** **章** **更** **新** **换** **代** **，** **重** **装** **上** **阵** 191](#bookmark119)

[7.1 永葆青春活力——迁移 191](#bookmark120)

[7.1.1 企业级应用架构 192](#bookmark121)

[7.1.2 企业级应用迁移内容及主要步骤 194](#bookmark122)

[7.1.3 企业级应用迁移中客户关注的问题 196](#bookmark123)

[7.2 迁移测试 196](#bookmark124)

[7.2.1 垂直迁移测试方法(VMTM) 197](#bookmark125)

[7.2.2 水平迁移测试方法(HMTM) 203](#bookmark126)

[7.2.3 其他迁移测试内容 207](#bookmark127)

[7.3 迁移测试的计划和实践 208](#bookmark128)

[7.3.1 如何选择迁移测试平台组合和测试用例 209](#bookmark129)

[7.3.2 迁移测试的一个具体场景 212](#bookmark130)

[7.4 学习笔记——迁移测试之小艾观 214](#bookmark131)

[参考资料： 214](#bookmark132)

[**第** **8** **章** **假如我是客户：面目全非脚与还我漂漂拳** 215](#bookmark133)

[8.1 其实，我是一个演员——如何扮演客户 216](#bookmark134)

[8.1.1 我知道，我不知道：只知道客户能够知道的 216](#bookmark135)

[8.1.2 我做，我不做：做客户会做的 218](#bookmark136)

[8.1.3 我想，我继续想：客户会怎么想 221](#bookmark137)

[8.2 面目全非脚——狠下心来改产品 222](#bookmark138)

[8.2.1 第一脚，学会开发 222](#bookmark139)

[8.2.2 第二脚，部署代码 231](#bookmark140)

[8.2.3 第三脚，联合测试 237](#bookmark141)

[8.3 还我漂漂拳——让产品对用户更友好 238](#bookmark142)

[8.3.1 协助完善产品设计 238](#bookmark143)

[8.3.2 客户化联合测试工作的痛苦和解决方案 239](#bookmark144)

[8.3.3 撰写详细易读的客户化教程 240](#bookmark145)

[8.4 学习笔记——客户化测试之小艾观 241](#bookmark146)

[参考资料： 242](#bookmark147)

[**第** **9** **章** **黎明之前最后的冲刺：成品测试** 243](#bookmark148)

[9.1 产品包装成金蛋，手握光碟抓虫子 243](#bookmark149)

[9.1.1 成品测试全体总动员 243](#bookmark150)

[9.1.2 协同作战——成品测试特性 245](#bookmark151)

[9.1.3 取舍之间——测试范围和策略 246](#bookmark152)

[9.1.4 争分夺秒——成品测试周期 248](#bookmark153)

[9.2 黎明前的黑暗——漏网之虫 250](#bookmark154)

[9.2.1 老案例生新虫子 251](#bookmark155)

[9.2.2 艰难抉择——漏网之虫综合分析及灭虫策略 252](#bookmark156)

[9.3 金蛋闪亮登场 254](#bookmark157)

[9.3.1 成品测试胜利退出 254](#bookmark158)

[9.3.2 质量检测报告之大观 256](#bookmark159)

[9.3.3 趁热打铁总结经验教训 259](#bookmark160)

[9.3.4 贯穿始终的缺陷分析 261](#bookmark161)

[9.4 学习笔记——成品测试之小艾观 269](#bookmark162)

[参考资料： 270](#bookmark163)

[**第** **1** **0** **章** **缝** **缝** **补** **补** **又** **三** **年** 271](#bookmark164)

[10.1 补丁包测试简介 271](#bookmark165)

[10.1.1 目标驱动的补丁包测试 272](#bookmark166)

[10.1.2 补丁包测试在产品生命周期中的意义 273](#bookmark167)

[10.1.3 补丁包需涵盖的内容及来源 274](#bookmark168)

[10.2 产品补丁包测试的基本流程 275](#bookmark169)

[10.2.1 确定补丁包里需要包含的内容 277](#bookmark170)

[10.2.2 制定好计划是关键 280](#bookmark171)

[10.2.3 执行开始前要做好准备工作 285](#bookmark172)

[10.2.4 测试开始及状态跟踪 286](#bookmark173)

[10.2.5 补丁包成功发布 288](#bookmark174)

[10.3 其他的流程和考虑因素 288](#bookmark175)

[10.3.1 缺陷分析要做到位 289](#bookmark176)

[10.3.2 补丁包测试的自动化考虑 291](#bookmark177)

[10.4 学习笔记——补丁包测试之小艾观 293](#bookmark178)

[**第** **1** **1** **章** **一** **遍** **又** **一** **遍** **：** **自** **动** **化** **的** **好** **处** **多** 294](#bookmark179)

[11.1 软件测试的工业化——自动化 294](#bookmark180)

[11.2 自动化的利与弊 295](#bookmark181)

[11.3 武功入门口诀——自动化脚本的关键 297](#bookmark182)

[11.3.1 有效发现缺陷 298](#bookmark183)

[11.3.2 详细的错误日志 299](#bookmark184)

[11.3.3 良好的可读性 300](#bookmark185)

[11.3.4 运行独立性与可重复性 302](#bookmark186)

[11.4 修炼进阶——开发适合自己的自动化框架 303](#bookmark187)

[11.4.1 需求分析——我们需要什么 304](#bookmark188)

[11.4.2 环境准备自动化——将自动化进行到底 305](#bookmark189)

[11.4.3 实现需要考虑的问题 306](#bookmark190)

[11.4.4 多台机器同时跑——自动化效率更上一层楼 307](#bookmark191)

[11.5 收发自如——灵活的自动化测试策略 307](#bookmark192)

[11.5.1 越早越好?——自动化脚本开发的时间 307](#bookmark193)

[11.5.2 让软件健康成长——回归测试中的自动化 309](#bookmark194)

[11.5.3 永远的录制/回放?——自动化方式的选取 310](#bookmark195)

[11.5.4 论持久战——测试脚本的配置管理 311](#bookmark196)

[11.5.5 拒绝半成品——测试脚本的验收 312](#bookmark197)

[11.6 实战洗礼——养兵千日用兵一时 313](#bookmark198)

[11.6.1 高，实在是高——自动化测试的效率 314](#bookmark199)

[11.6.2 无处不在的地雷——自动化测试的稳定性 314](#bookmark200)

[11.6.3 人员匮乏——现实和理想的差距 315](#bookmark201)

[11.6.4 如何定义自动化比例 316](#bookmark202)

[11.6.5 自动化VS 手动测试 316](#bookmark203)

[11.7 学习笔记——自动化测试之小艾观 317](#bookmark204)

[**第** **1** **2** **章** **辗** **转** **腾** **挪** **：** **身** **手** **敏** **捷** **的** **测** **试** 319](#bookmark205)

[12.1 我和天使有个约会——初识敏捷 319](#bookmark206)

[12.1.1 团结协作，质量为要，快速交付，响应变化，持续改进——](#bookmark207)

[敏捷的基本原则 320](#bookmark207)

[12.1.2 一切以用户价值为中心——敏捷开发基本流程 326](#bookmark208)

[12.1.3 敏捷=不做计划吗 31](#bookmark210)

[12.2 敏捷转型 — 一 步 一 个脚印 32](#bookmark211)

[12.2.1 分析现状，尝试转型 33](#bookmark212)

[12.2.2 万里长征第一步，从迭代周期开始 34](#bookmark213)

[12.2.3 开发测试同步，按Sprint 交付增量 35](#bookmark214)

[12.2.4 组建统一团队，通过用户故事驱动 37](#bookmark215)

[12.2.5 适合敏捷开发的实践和工具 341](#bookmark216)

[12.3 互通有无——与其他实施敏捷的团队的探讨 343](#bookmark217)

[12.4 学习笔记——敏捷之小艾观 346](#bookmark218)

[**第** **1** **3** **章** **涅** **槃** **!** **华** **山** **论** **剑** **：** **测** **试** **架** **构** **师** **的** **诞** **生** 348](#bookmark219)

[13.1 走技术路线还是管理路线，或者当架构师 348](#bookmark220)

[13.2 测试架构师是干啥的 351](#bookmark221)

[13.3 测试架构师之N 项 修 炼 354](#bookmark222)

[13.3.1 技术技能修炼 354](#bookmark223)

[13.3.2 软技能修炼 358](#bookmark224)

[13.4 测试“九阴真经”— — 总体测试计划 360](#bookmark225)

[13.5 将缺陷扼杀在萌芽阶段 362](#bookmark226)

[13.6 火眼金睛——到底哪儿被测试遗漏了 364](#bookmark227)

[13.6.1 什么是代码覆盖率测试 364](#bookmark228)

[13.6.2 代码覆盖率测试工具 365](#bookmark229)

[13.6.3 代码覆盖率测试实战 368](#bookmark230)

[13.6.4 代码覆盖率测试与自动化测试的集成 369](#bookmark231)

[13.7 测试过程改进 370](#bookmark232)

[13.7.1 什么是价值流程图 370](#bookmark233)

[13.7.2 时间被消耗在了哪些地方 371](#bookmark234)

[13.8 测试驱动的软件设计变革 372](#bookmark235)

[13.8.1 测试人员没有主动权吗 372](#bookmark236)

[13.8.2 性能引发的软件设计变革 374](#bookmark237)

[13.9 软件系统的RAS 保 证 376](#bookmark238)

[13.10 学习笔记——测试架构师之小艾观 378](#bookmark239)

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 。

从菜鸟到测试架构师——一个测试工程师的成长日记

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

第1章 上班第一天，新人培训

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

第1章 上班第一天，新人培训

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。

第1章 上班第一天，新人培训

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。

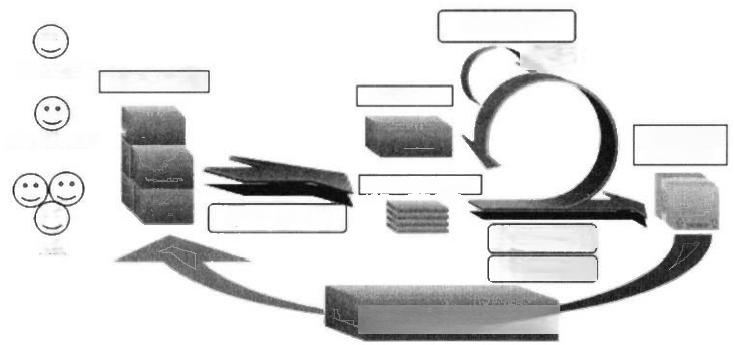
增量开发用的则是一种“分段完成”的策略。在增量开发模式中，系统中不同的部分 被安排在多个阶段完成，各个部分完成后再集成到系统中。

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

从菜鸟到测试架构师 — — 一 个测试工程师的成长日记

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 。



每日站立会议

90

24小时

产品Backlog

Sprint目标

潜在可交付的 产品增量

Sprint Backlog二

Sprint 计划会议

Sprint 评审会议

Sprint 回顾会议

*新的(或更新的)用户故事*

产品负责人

Scrum Master

团队

0

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

第1章 上班第一天，新人培训

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

从菜鸟到测试架构师———个测试工程师的成长日记

“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。

●

10 ·

第1章 上班第一天，新人培训

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。

从菜鸟到测试架构师———个测试工程师的成长日记

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

第1章 上班第一天，新人培训

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。

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。

第1章 上班第一天，新人培训

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!

第1章 上班第一天，新人培训

**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。

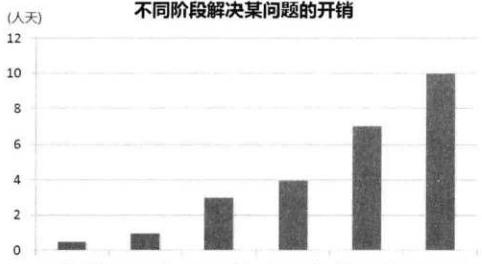
第1章 上班第一天，新人培训

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。



**需求分析概要设计具体设计单元测试集成测试产品发布**

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?

从菜鸟到测试架构师—一个测试工程师的成长日记

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

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。

第1章 上班第一天，新人培训

虑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。

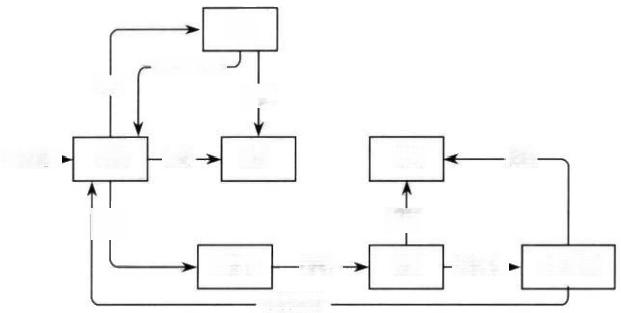
例如，在一个面向服务的体系结构 (SOA) 开发模式下的系统，如果某些流程中服务 并没有开发完成，但目标测试用例必须用到这个没完成的服务，为了让用例完整地走通，

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

从菜鸟到测试架构师——一个测试工程师的成长日记

陷跟踪系统定义了缺陷的生命周期和相关信息',如图1-3所示。



退回

重新开启一

退回

取消

一创建→ 开启 取消 取消

接受

工作中 跟踪一

重新开启

不认可→ 验证失败

认可

验证

关闭

确认 ·

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。

第1章 上班第一天，新人培训

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。”

第1章 上班第一天，新人培训

**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。

第1章 上班第一天，新人培训

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。

从菜鸟到测试架构师———个测试工程师的成长日记

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。

第1章 上班第一天，新人培训

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。



问题分析/验证 问题解决

15% 10%

学 习 自 动 化

?% ?%

执行测试

40%

需求分析

15%

剩 余 10%

计划

10%

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

从菜鸟到测试架构师———个测试工程师的成长日记

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

第1章 上班第一天，新人培训

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，

第1章 上班第一天，新人培训

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。

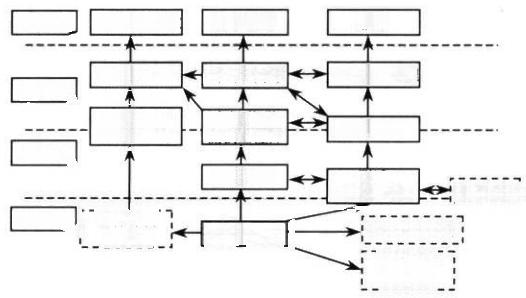
第 1 章 上 班 第 一 天 ， 新 人 培 训

**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 。



总监

高级测试经理

高级职位

测试经理

中级职位

测试

项目经理 项目经理

产品销售专员

测试工程师

产品支持

工程师

资深技术专家

产品架构师

高级/资学 开发工程师

软件开发 工程师

技术总监

测试架构师

高级/资学 测试工程师

测试组长

初级职位；

资深职位

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。

第1章 上班第一天，新人培训

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 2 章

**万事开头难：软件从安装开始**

在凯文的指导下，小艾开始对于测试有了初步的了解，就这样忐忑地开始了自己的测 试生涯。

“可是，测试要从哪里开始呢?”他坐在办公室对凯文问出了这样的问题。

“测试要先熟悉自己的产品。这样吧，我先安排你到安装测试组工作，熟悉一下产品。”

“安装测试组?安装也需要测试吗?这不是很简单的事情吗?”小艾想起自己在读书 的时候，经常给自己的计算机安装系统和各种软件，是一件很轻松的事情。于是不由得有 些失望。

“以前你所使用的软件都是经过安装测试后发布的，所以你可以简单顺利地完成安装 过程，但是事实上安装测试并不像你想象得那么简单，试一下你就知道了。而且，你平时 用的都是个人桌面计算机上的简单应用，而对于大型商用软件来说，它所需要支持的各种 集成环境、集群配置等，都是很复杂的。”凯文笑着说：“你去安装测试组报到，安装测试 组的组长安岩会安排并帮助你熟悉工作，遇到任何问题都可以请教她。”

从菜鸟到测试架构师———个测试工程师的成长日记

**2.1** **软件，是装出来的**

“安装测试组都需要做什么呢?”这是小艾问安岩的第一个问题。安岩给小艾的解释 是这样的：

安装可以很简单，像一些简单的桌面应用程序，只是简单地复制一些文件，对于这种 应用，不需要专门的安装测试组，安装测试能够和其他测试合并在一起。

安装也可以很复杂，比如说像企业级Java EE应用软件的安装， 一般来说，它要支持 多个操作系统平台、多种数据库、多个版本的中间件、多种网络服务器 (Web Server)、 多种拓扑结构，等等，这就要求测试人员具有较好的操作系统、数据库及网络服务器等知 识。一般需要一个专门的安装测试组来进行相关测试。我们正在测试的就是一个企业级 Java EE 应用软件。

小艾接着问：“企业级 Java EE应用软件?能举几个例子吗?”

安岩说：“没问题，Java EE应用软件是符合Java EE技术规范开发的相关应用， 一般 需要部署到Java EE 应用服务器上才能对外提供服务。 一般来说，企业级Java EE应用， 都需要使用数据库软件。典型的拓扑结构是三层架构：前端是网络服务器，中间是应用服 务器，后端是数据库服务器。

“典型的企业级Java EE 应用软件，像IBM WebSphere Commerce应用套件，提供一整 套完整的电子商务解决方案。”

**2.1.1** **安装测试概念解读**

小艾仍然略带疑惑地接着问“那么什么是安装测试呢?”

安岩说道：“软件产品多种多样，很难就所有产品的特性下一个安装测试的定义。就

一般的企业级Java EE应用软件来说，安装测试应该做到以下几点：

确保待测产品能够在所有支持的操作系统、数据库、应用服务器中间件、网络服



1 <http://baike.baidu.com/view/651442.htm> 安装测试-百度百科2010-05-17

第2章 万事开头难：软件从安装开始

务器、拓扑结构等的各种组合情况下，被正确地安装和卸载。

确保安装文档的正确性和易读性。

通俗来说，就是确保安装相关的代码和相关的安装配置文档的正确性。”

安岩继续说：“了解了安装测试的概念，我介绍一下我们是如何来规划安装测试的， 就是安装测试计划。可以这么说，一个好的测试计划将成就一个软件产品。一个坏的测试 计划将毁灭一个软件产品。

“每一个测试人员都需要认真仔细地阅读安装测试计划，并且按照这个文档的规定来 进行具体的测试，这是对每一个测试人员最基本的要求。测试计划的主体部分详细描述了 安装测试的测试配置和测试场景，这部分内容也最多，我们以后还会详细讲到。”

小艾听了安岩对安装计划的讲解之后受益匪浅，又跟安岩要了安装计划相关的材料， 暗下决心，一定先要把安装测试计划搞懂、搞透。

**2.1.2** **测试之初体验** **—** **—** **一** **份手册，** **N台机器，** **一** **堆软件**

“纸上得来终觉浅，绝知此事要躬行”。第二天，安岩安排小艾开始进行基本的安装测 试，通过实践来熟悉安装测试。

安岩接着详细介绍了安装测试的基本流程，小艾认真地做着笔记。

**1.** **学习测试计划与测试用例**

正确的开始点非常重要，学习测试计划与测试用例是每个安装测试人员开始的第一 步，这个原则对其他测试类型同样适用。

在安装测试计划中，包含所有的测试用例，一般要求每个测试人员对所有测试用例有 一个基本的了解，对自己要测试的那部分，要有全面和细致的了解。

比如，对于一个三节点的安装测试用例，需要搭建一个典型的三节点的环境(网络服 务器节点、应用服务器节点、数据库节点)。测试用例中规定了详细的测试步骤和检查点， 这些都需要认真阅读和特别加以注意。

**2.** **搭建测试机器**

每个产品对其运行的软件和硬件都有具体要求，测试用例会明确规定使用什么样的硬

从菜鸟到测试架构师———个测试工程师的成长日记

件配置和操作系统版本，测试人员需要自己来搭建测试机器，并且必须保证与测试计划的 描述严格一致。

此外，安装测试人员在此阶段还有一个重要的工作，就是检查安装文档中关于软硬件 配置描述的正确性。千万不能闹出“产品实际支持的是Windows 2008操作系统，文档中 却说支持的是Windows XP” 这样的错误。这可是非常严重的错误，这个错误可能会导致 为客户提供的方案信息不正确，直接影响客户的预算和实施，甚至可能延误上线时间。所 以一定要认真对待。

一旦得到一套干净的测试环境之后，最好做一个备份，以供下次重复测试时直接恢复， 从而节省部分测试时间。

**3.** **准备待测试软件产品**

这一步需要把待测软件下载到测试机器上。 一般来说，构建测试组 (Build Team) 会 准备待测软件包，并且按照一定的频率(比如每天或每周)发布到某个文件服务器上，你 需要确认你要使用哪天的构建，并下载到测试机器上。如果待测软件需要依赖某些基础软 件(如应用服务器软件、数据库软件等)来运行，你也需要在这个阶段把它们一并准备好。

**4.** **按照安装手册中的步骤来执行**

在实际执行中，这个步骤是最容易被忽视的一步。资深的测试人员可能对安装测试非 常熟悉了，就会常常不看安装手册而按照自己的经验来安装，这就导致安装手册有问题而 未被发现纠正。导致的后果就是用户按照安装手册来安装时，根本安装不下去，客户的失 望和愤怒可想而知。[0]

所以不能怕麻烦，一定要按照安装手册一步一步来安装，安装手册如果有问题，要联 系文档人员及时纠正。如果安装程序有问题，就需要开缺陷报告。

**5.** **撰写测试报告，详细记录测试结果**

最后，我们要写测试报告，详细记录测试的流程和发现的问题及处理结果。

测试组长会仔细阅读每个测试人员的测试报告，掌握当前测试状态和进度情况。

小艾总结说：“看来安装测试真的不简单呀，有那么多步骤，要准备测试机器，还要 安装一堆相关的软件，最后还要检查文档。”

第2章 万事开头难：软件从安装开始

安岩说道：“总结得不错，一份手册，N 台机器，一堆软件，这是对安装测试执行很 形象的总结。”

**2.2** **全面撒网，重点排查**

一个月过去了，小艾顺利地完成了安岩交给的第一份任务，对安装测试流程已经基本 掌握清楚了。

然而，小艾一直有一个问题：“一个好的可执行的测试计划是确保测试质量的关键， 那么测试计划是怎么写出来的呢?”

一天小艾困惑地问安岩：“在测试计划中，为什么要选择这些测试配置和测试场景呢? 像我们这么复杂的产品又要支持这个，又要支持那个，要测试的内容好多呀，在时间和人 力有限的条件下怎么保证既测得全面又测得完呢?”

安岩笑着解释：“好问题，这可是一门学问，是有规律可循的，听我慢慢道来。”

**2.2.1** **选择测试配置**

前面曾经介绍过安装测试的概念，其中重要的一部分是要确保待测产品在所有支持的 操作系统、数据库、应用服务器中间件、网络服务器、拓扑结构等的各种组合情况下，能 够被正确地安装和卸载。

撰写测试计划时，首先要清楚地列出产品所有能够支持的测试配置。

测试配置指待测软件所能支持的硬件环境、软件环境和配置方法的组合。

对于不同种类的待测软件，可以从各个角度来找出测试配置，对于Java EE 应用软件 来说，可以尝试从下面的角度来考虑。

**操作系统：**Java EE 应用是跨平台的应用， 一般都能支持多种操作系统，例如 Winodws,AIX,Linux,Solaris 等。

**应用服务器：** Java EE 应用需要部署到应用服务器才能发挥作用，例如 IBM Wesphere Application Server, 简称 WAS。

**数据库：**Java EE 应用一般都需要使用数据库来存取信息，例如DB2 和 Oracle 两

从菜鸟到测试架构师—一个测试工程师的成长日记

种数据库。

网络服务器 (Web Server): 像我们前面说过的， Java EE应用一般都需要使用网 络服务器来提高性能和支持可扩展性。例如 Sunone,Internet Information Services (下面简称IIS),IBM [HTTP SERVER](HTTPSERVER) (下面简称IHS) 等 。

拓扑：Java EE 应用部署时一般都支持不同的拓扑结构来满足不同的对性能和可 扩展性的支持。例如单节点、三节点、集群三种拓扑。[0J

版本：为了适应不同客户群的需求， Java EE应用软件一般被包装成不同的版本。 例如企业版、专业版、精简版三种。

安装类型： 一般的Java EE软件都会支持多种安装类型以满足不同的安装需求， 例如快速安装、定制安装两种。

对于上面所列的每一个角度的具体取值，每个产品不尽相同，可以从设计文档中获得 具体的信息。

对于上面所列的每一个角度，都可以看做一个变量。所有变量的乘法组合有几十种甚 至好几百种，每一个组合就是一个测试配置。下面是一个例子：

AIX/WAS/DB2/IHS/ 三节点/企业版/定制安装

我们能用一个乘法公式来计算测试配置的总数：



其 中 ，V表示某个变量的种类， n 表示变量的个数， S 表示可能组合的总数。

在时间和人力有限的条件下，不可能每种组合都测试，就需要根据一些限制条件来进 一步缩小测试范围，比如以我们总结的最佳实践。

1. 每种支持的操作系统版本至少需要测试一次

每个支持的操作系统版本都至少需要被测试一次，为什么有这个要求呢?因为在安装 文档中，会叙述产品支持的操作系统版本，不测试哪一个，都会有比较大的风险。这里要细 化到操作系统版本，例如对于Windows 操作系统，某产品支持Windows 2003和 Windows 2008, 那么这两个版本都需要测试到。

第2章 万事开头难：软件从安装开始

**2.** **每种支持的网络服务器版本至少测试一次**

同理，安装文档中，会叙述产品支持的网络服务器版本，所以需要确保在测试配置组 合中，每种网络服务器版本被测试至少一次。

**3.** **每种支持的数据库版本至少测试一次**

同理，安装文档中，会叙述产品支持的数据库，所以需要确保在测试配置组合中，每 种数据库版本至少被测试一次。

**4.** **设计文档中需要重点测试的配置必须测试**

开发人员的一些代码可能在某些测试配置下会有问题，开发人员如果有这种担心，就 会在设计文档中特别指出，测试人员就要特别注意，这个配置很可能会有问题的。

**5.** **客户典型配置必须测试**

在总结大多数客户使用软件产品的习惯后，我们发现一些典型的测试配置，大多数用 户都这么用。例如，选择 Windows的客户一般选IIS 作为网络服务器 (Web Server)。

**6.** **以往版本产品由客户报告的问题分析和由测试人员报告的缺陷分析**

如果测试的是一个新产品，可能这部分数据没用。如果测试的是一个以前产品的升级 版本，就能分析这部分数据。

客户报告的产品问题：是指产品发布后，客户发现的问题。这部分数据非常有价值。 为什么客户发现了这样的问题，而测试人员却没有发现呢?从中能分析出测试计划中可能 存在的漏洞。

以往版本的缺陷分析：这里的缺陷是由测试人员在产品发布前发现的，从中能分析出 哪些部分特别容易出现问题，在新版本中这些部分可能仍然会出问题。

这样，根据上面的原则选择后，就能得到合适的测试配置覆盖(Test Config Coverage)。 为了直观起见，我们也能画一个矩阵图更形象地表达出测试配置覆盖，如表2-1所示 (画“×”的组合表示被选中。限于排版的要求，只能列出 Windows 平台下的所有组合， 在实际应用中，可以补齐AIX,Linux 平台下的相关组合)。

·

● 45

从菜鸟到测试架构师———个测试工程师的成长日记

**表2-1测试覆盖表**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Windows** | | | | | |
| **DB2** | | | **Oracle** | | |
| **Sunone** | **IHS** | **IIS** | **SunONE** | **IHS** | IIS |
| 企业版 | 单节点 | × |  |  |  |  |  |
| 三节点 |  | × |  |  |  |  |
| 集群 |  |  | × |  |  | × |
| 专业版 | 单节点 |  |  |  | × |  |  |
| 三节点 | × |  |  |  | × |  |
| 集群 |  |  |  |  |  |  |
| 精简版 | 单节点 |  |  |  | × |  |  |
| 三节点 |  | × |  |  |  |  |
| 集群 | × |  |  |  | × |  |

从表2-1 中能够看出至少有一个测试配置被选中分别去测试每种数据库： DB2 和 Oracle; 至少有一个测试配置被选中分别去测试每种网络服务器： SunONE 、IHS 和 IIS; 至少有一个测试配置被选中分别去测试每种拓扑结构：单节点、三节点和集群；至少有一 个测试配置被选中分别去测试每种发行版本：企业版、专业版、精简版；至少有一个测试 配置被选中分别去测试典型的客户配置，像 Windows/IIS, 等等。这样选择下来，一共选 中了11个测试配置，而所有的测试配置有54个之多，可见在根据最佳实践选择后，测试 工作量大大降低了，并且测试的质量仍然有保障，因为所有的测试点都有相应的测试覆盖。

**2.2.2** **找出测试场景**

有了测试配置后，接下来就需要找出测试场景 (Test Scenario)。

测试场景就是一系列紧密关联的操作步骤的组合。比如产品安装流程、产品卸载流 程等。

可根据以下信息来源来设计测试场景。

**1.** **需求说明书**

需求说明书是设计测试场景的主要依据，从中能够找出最终用户对产品安装和卸载方

第2章 万事开头难：软件从安装开始

面的各种需求。每一个测试人员都需要认真阅读，特别是测试计划的作者，需要对它非常 清 楚 。

**2.** **用户手册**

就是安装文档，最终用户将按照用户手册来安装产品，其中会陈述一些典型场景和注 意事项，我们可以据此来设计测试场景。

从一个软件产品的生命周期来看，一个产品会经历安装、升级、卸载的过程，所以能 从中找出一些基本及典型的场景。

**1.** **在选定的测试配置上安装产品**

这是一个最基本的测试场景。从下面的描述中，能发现基本的步骤。

确保在看安装手册的情况下，产品能够被正确、轻松地安装。如果安装过程中遇到问 题(例如磁盘满、输入信息错误等),安装程序能够清晰明确地指出问题的原因所在，并 且在用户纠正问题后安装能够继续进行，产品安装后，应用能够正常启动。

**2.** **卸载产品**

确保基本产品能够被正确地卸载。这也是一个最基本的测试场景。

**3.** **在卸载后重新安装产品**

确保产品卸载后能够被正确重新安装。有些客户可能会用到此场景。

**4.** **文件权限检查和敏感数据检查**

确保产品安装后，文件权限是正确的，没有敏感数据暴露。这个场景是确保软件产品 安全性的典型的场景。

**5.** **残障人员也能顺利安装产品**

比如确保为盲人提供语音提示，为近视眼患者提供放大功能，等等。这是确保产品的 可访问性的典型的场景。

上面列了一些最基本、最通用的测试场景。当然，针对特定的产品，可以从需求说明 书和用户手册里找出更多特定的测试场景，这里就不一一赘述了。

**2.2.3** **组合出测试用例**

现在，我们就能写出具体的测试用例了：

**测试用例=测试配置+测试场景**

**测试用例**就是在某个测试配置上来执行某个测试场景。

为了方便看出在哪个测试配置上执行了哪些测试场景，可以画出如表2-2所示(画“×” 表示被选中)的矩阵表。

表2-2测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 测试场景1 | 测试场景2 | 测试场景3 | … | **测试场景N** |
| 测试配置1 | × | × | × |  | × |
| 测试配置2 | × |  |  |  | × |
| 测试配置3 |  | × | × |  |  |
| …… |  |  |  |  |  |
| 测 试 配 置 N | × |  |  |  | × |

我们分配任务一般是以测试用例为单位来分配的，跟踪项目状态也是以测试用例为单 位 的 。

小艾听了以上分析，豁然开朗，疑惑全解开了，感慨地说：“测试配置、测试场景、 测试用例是测试计划的精髓呀，看来下次我也能写测试计划了!”

**2.3** **安装测试质量之大观**

在一次讨论中，小艾有了新问题：“对于每一个具体的安装测试用例，怎样才能保证 测试质量是达到标准的呢?”

安岩说到“很简单，需要制定一套通过标准，当测试用例满足这套标准后，我们就可 以认为测试达到了质量要求。”

第2章 万事开头难：软件从安装开始

小艾接着问：“都有哪些通过标准?能和我说说吗?”

安岩笑着说：“我们有过一个系统的总结，请听我慢慢道来。”

**2.3.1** **产品安装**

对于一般的Java EE应用软件，以下是我们总结的产品安装测试相关的通过标准。

**1.** **安装程序能够自动检查安装前提条件是否满足**

对于软件安装来说，都或多或少地有一些前提条件，如果前提条件不满足，安装就不 能继续或者失败。下面列出最通用的一些前提条件：

磁盘空间是否符合安装需求；

网络状况是否符合安装需求；

CPU 和内存是否符合安装需求；

操作用户是否有足够的权限；

操作系统是否符合安装需求；

操作系统必须的补丁包是否安装等。

一般需要针对每一项设计出一些异常情况，确保安装程序能够报告出相应的错误信息。

比如最小的磁盘空间需求是2GB, 如果指定一个仅有1GB 自由空间的安装目录，应 该报告磁盘空间不足。

**2.** **软件安装向导的用户界面(User** **Interface)测** **试**

这也是安装测试重要的一方面，大多数程序都会提供图形化安装方式，所以我们需要 重点测试这部分， 一般我们需要验证：

用户界面出现的描述性文字是清晰正确的。

用户界面出现的输入框，选择框功能正常，如果输入异常信息，需要提示输入 错误。

用户界面出现的按钮，像“下一步”、“确定”、“取消”等，功能正常。

安装过程中如果有进度条显示，需要确保进度信息显示正确。

**3.** **软件安装各个选项的组合确保符合概要设计说明**

软件安装可能存在多个选项组合，例如，典型安装，最小安装，完全安装等。要确保 每个选项的安装过程和安装后的结果是符合设计文档中的期望行为。

**4.** **软件安装过程是否可以取消，单击“取消”按钮后，写入的文件是否如设计文档** **中期望的说明来处理**

如果安装正式开始后不能取消安装，“取消”按钮应该变灰。

如果安装正式开始后能够取消安装，用户单击“取消”按钮后，安装能够正常停 止，写入的文件应被自动删除01,如果某些不能删除，则需要给出提示说明。

**5.** **软件安装过程中意外情况的处理是否符合需求(如死机，重启，断电等)**

具体看设计文档中如何处理意外情况。 一般来说软件安装过程中如果有意外发生(如 死机，重启，断电等),重启后，安装程序一般有如下两种处理方式：

安装程序侦测到程序意外终止，能够恢复安装进度，继续安装。

安装程序侦测到程序意外终止，不能够恢复安装进度，回滚所有安装操作，例如 删除已经写入的文件，删除已经写入的注册信息。然后用户能够重新安装。

**6.** **安装过程是否是可以回溯的(即是否可以回上一步重新选择)**

这个比较好理解，是属于安装易用性的一种测试。一般的安装程序都具有这种功能， 所以在测试时，我们要确保功能是正确的。

**7.** **软件安装过程中是否支持快捷键，快捷键的设置是否符合用户要求**

这属于安装可用性方面的测试。在考虑用户没有鼠标设备的情况下，通过键盘也能完 成整个安装。

第2章 万事开头难：软件从安装开始

**8.** **软件静默安装测试**

一些软件除了支持图形界面安装方式外，还支持静默 (Silent) 安装，一般来说就是 把安装过程中必要的用户输入信息写到一个文本响应文件中，然后使用命令行来安装。静 默安装一般用于自动化安装程序中。

对于静默安装测试，一般需要注意：

正确的响应文件，静默安装能够成功完成，并且在日志文件中记录安装成功相关 的信息。

错误的响应文件，静默安装失败，并且在日志文件中表明安装失败的原因。

**9.** **软件安装后安装日志中没有错误信息**

软件安装后，需要检查安装日志文件，确保没有错误信息或警告信息。

**10.** **软件安装后应用是否能够正常运行**

这也是一个重要的通过标准，即使软件安装过程没有问题，但是安装后应用不能正常 运行，也表明安装测试用例是失败的。

为了保证这一点，一般我们在软件安装后，需要执行一些基本的功能测试用例。

**11.** **安装后的文件夹及文件是否写到了指定的目录里，文件的大小和权限是否正确**

我们用这一点来检查安装的完整性和安全性是否符合预期。一般安装后会使用一个扫 描程序扫描安装后的文件夹及文件，然后和通过标准进行比对，来确保这一点。

**12.** **安装后一些重要文件的内容是否正确**

一些重要文件像版本信息文件、注册文件等，需要确保它们的内容是符合设计文档要 求的。

**13.** **安装后数据库中的信息是否正确**

如果测试的软件需要使用数据库，需要检查这一点。一般需要检查：

数据库是否被正确创建；

数据库模式(像表、索引、触发器等)是否被正确创建；

数据库中的数据是否正确；

检查数据库工作工作量一般比较大，需要使用工具来帮助检查。

上面列的各项属于最佳实践总结，针对不同的产品本身的特点，要灵活运用，有些产 品可能需要增加一些测试，有些产品可能需要减少一些测试。总之，要根据具体情况去应用。

**2.3.2** **产品卸载**

对于一般的Java EE应用软件，以下是我们总结的与卸载测试相关的通过标准。

**1.** **测试软件自带的卸载程序**

一般来说，软件都会有自带卸载程序，需要确保它功能正常。

2. 测试使用操作系统自带的添加/删除工具(以Windows 为例)来卸载程序的情况 在 Windows 操作系统中比较普遍，需要确保它功能正常。

3. 测试卸载程序在程序运行/终止等状态时的卸载情况

一般来说卸载之前需要使应用程序处于停止状态，卸载程序能够检查程序状态，并且 给出正确提示。

**4.** **测试卸载软件过程中，能否取消卸载进程**

如果支持取消操作，单击取消后，观察软件能否继续正常使用。

如果不支持取消，用户应该不能单击“取消”按钮。

5. 测试卸载后文件是否全部删除，包括安装文件夹、注册表、系统环境变量 如果某些文件在卸载过程中没有删除，应该明确提示用户。

第2章 万事开头难：软件从安装开始

**6.** **卸载过程中出现的意外情况的测试(如死机、断电、重启)**

具体要看设计文档中如何处理这种异常情况， 一般来说：

再次卸载时，能够侦测前次卸载失败，恢复卸载进度，继续完成卸载。

再次卸载时，能够侦测前次卸载失败，但不能恢复卸载进度，需要用户按照相关 文档手工卸载。

**7. 软件自带卸载程序的UI 测** **试**

一般来说，卸载程序的UI 要比安装程序简单，检查方法类似于安装UI 测试。

这里要强调一点，卸载测试和安装测试同样重要，都和用户体验息息相关，测试人员 有时会轻视卸载测试，这一点要引起注意。

**2.4客户的圣经——用户手册验证**

安岩在检查小艾测试报告时，发现小艾犯了一般测试人员的通病——不重视用户手册 验证，小艾负责的一个测试用例的相关文档有不少明显的错误。

安岩找到小艾后，语重心长地对小艾说：

“用户手册验证是安装测试重要的一部分，但往往被测试人员忽视。这一点需要引起 特别注意。”

“用户安装手册对最终客户非常重要，如果安装手册的某些信息是错误的，将导致安 装失败。如果某些信息是含糊不清的，将引起用户理解歧义。这些问题将导致安装工作暂 停和项目进度延期， 一方面将给客户带来不必要的经济损失，另一方面，用户对这个软件 的质量产生怀疑，这个软件的口碑也会越来越差。”

“所以不论是项目经理或是相关的测试人员真的要从此刻开始注意用户手册验证的重 要性。”

小艾知道是自己的疏忽，频频点头道：“从今天开始，我一定认真对待文档验证工作， 能给我讲讲一般的验证方法吗?”

安岩缓和了一下语气：“知错能改，善莫大焉!请听我下文分解。”

**2.4.1** **一般验证方法**

用户手册验证工作要贯穿整个安装测试始终。我们不仅要保证安装文档是清晰和正确 的，还要保证文档是易读的。下面介绍几种常用并且行之有效的方法。

测试人员在执行具体的测试用例时，必须严格按照安装文档的步骤来操作。测试 人员有责任就不正确的部分提交修改建议并且验证。为了保证这一点，通常创建 专门的文档相关的测试用例，来跟踪其状态。

测试组长定期组织所有测试人员集体审阅所有文档内容，并且就不清楚的内容展 开讨论和深入追查，也是一个行之有效的办法。

**2.4.2** **文档审阅流程**

文档验证是一个繁杂和反复的过程，对于篇幅比较长、逻辑比较复杂的文档尤其如此。

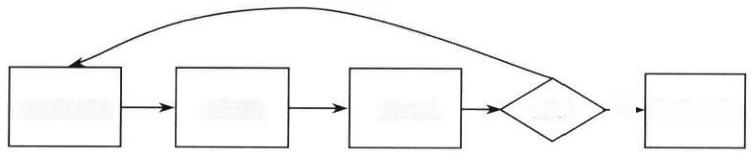
为了提高文档审阅的效率，一般需要借助文档管理系统来统一管理，其一般的功能应 包 括 ：

测试人员可以同时审阅同一文档，并且提交修改建议。

信息开发人员根据修改建议更正文档，并且给提交者发送验证通知。

提交者接收到验证通知后，能够关闭修改建议以表明验证通过。如果验证未通过， 能够拒绝相关的更改，继续提交修改建议。

小艾听后颇有感触，安岩真是太认真负责了。小艾随后也用起功来，认真总结了一般 的文档审阅流程，如图2-1所示。



否，再次提交修改建议

提交修改建议 等待更改 验证更改 成功? 是 关闭修改建议

图2-1 文档审阅流程

第2章 万事开头难：软件从安装开始

**2.5引进先进设备——安装自动化测试**

项目已经进行到中期了，小艾基本熟悉了安装测试内容和测试流程。平时没事经常和 功能测试和性能测试的同事交流，发现他们的测试都是自动化的，小艾好羡慕呀，想知道 是否安装测试也能自动化呢?

**2.5.1** **效率的提高从自动化开始**

在一次周会上，小艾问出了上面的问题，安岩讲道：“自动化测试是测试的发展方向 和趋势，能够大幅提高测试的效率。从这个版本开始，安装测试组已经开始尝试自动化测 试 了 。

“我们做过统计， 一个测试用例，手工执行要三个小时完成，而自动化后只需要一个 半个小时。当然开发自动化脚本也要花一定时间和成本，维护自动化脚本也需要花成本， 但总体来说，自动化后效率还是比手工执行效率有大幅的提高。测试用例需要重复执行的 次数越多，自动化后能节省的成本越高，投资回报率越高。”

小艾疑惑地问：“手工执行要三个小时完成，而自动化后只需要一个半个小时，这是 怎么算出来的，怎么会差那么多?”

“我来算一笔账就清楚了。看一个典型的安装场景，如表2-3所示。”

**表2-3安装场景**

|  |  |  |  |
| --- | --- | --- | --- |
| **步** **骤** | **自动化(分钟)** | **手工(分钟)** | **原** **因** |
| 1.下载待测构建 | 30 | 50 | 由于要从多个FTP站点下载多个文件，手工下载不是连续 的，并且要手工输入命令，额外的20分钟是手工操作和等待 |
| 2.解压缩(unzip) | 10 | 20 | 手工unzip多个文件不能连续完成，并且要手工输入命令， 所以多花10分钟 |
| 3.安装产品 | 30 | 40 | 10分钟是手工输入 |
| 5.验证安装 | 20 | 80 | 安装验证有多个验证点，手工验证非常费时，而自动化去 验证既快又准确 |
| 总计 | 90 | 190 |  |

小艾恍然大悟，“原来如此，我明白了，计算机最适合高速连续地执行确定的任务， 而用人脑来执行这些重复且枯燥的步骤就要慢得多。”

“说得对，人脑更适合做创造性的工作，自动化测试就是把人脑从枯燥的工作解放出 来，而去做更适合人脑的事情，比如去分析问题、写自动化脚本等。”

安岩接着说：“对于安装测试，我们的测试用例一般要重复执行多次，所以自动化对 安装测试意义很大。现在项目进行到中期了，我们已经完成了大部分测试用例的自动化开 发，接下来就是执行了，现在谁有兴趣做这部分测试?”

小艾对自动化很感兴趣，首先举手报名“算我一个!”,接下来又有几个人报了名。小 艾接着说：“但是自动化测试怎么做?”

安岩说，接下来我将给大家做一个培训，来解惑答疑，请大家准时参加。

**2.5.2** **自动化测试的实现**

为了提高测试效率和增加测试覆盖，我们自己为安装测试开发了一个自动化安装/卸 载/验证结果的工具。现在给大家来讲一下这个工具设计的思想。

其实，仔细想一想，对于安装测试，我们每天手工执行的测试用例，包含众多手工步 骤，比如下载测试构建、解压缩、输入必要信息、验证结果等。把这些手工部分使用某种 工具变成自动化部分，就把人给解放出来了。

对于一个典型的Java EE 应用的安装场景，让我们来看看一个手工的测试用例执行 步 骤 。

**1.** **安装数据库**

验证：检查相关日志文件，确保安装成功。

**2.** **安装应用服务器**

验证：检查相关日志文件，确保安装成功。

**3.** **安装网络服务器**

验证：检查相关日志文件，确保安装成功。

第2章 万事开头难：软件从安装开始

**4.** **安** **装Java** **EE应** **用**

验证：检查相关日志文件，确保安装成功。

**5.** **安装后基本功能验证**

要把手工步骤自动化， 一般要选择合适的自动化工具来实现自动化测试。自动化测试 的工具很多，下面主要介绍两个和安装测试相关的自动化工具：

IBM Rational Functional Tester¹

IBM Rational Functional Tester(RFT)是一款先进的、自动化的功能和回归测试工具， 它适合做GUI 界面相关的自动化测试。

Apache ANT²

ANT 是一个基于Java 的自动化脚本引擎，脚本格式为 XML 。 除了做Java 编译相关 任务外， ANT 还可以通过插件实现很多应用的调用。 ANT 适合命令行交互相关的自动化 测试。

我们的产品是支持命令行方式来安装的，并且自动化测试的目的是做回归测试，所以 我们选择 ANT 工具来实现自动化。下面主要介绍一下相关的原理，对于具体的实现，大 家可以参考ANT 相关的书籍来具体实现。

(1)首先要把测试用例中的每一步细化，细化到足够明确，能够用ANT 语言来表达。 比如第一步“安装数据库，验证：检查相关日志文件，确保安装成功”可以细化到：

a) 从某个文件服务器上下载数据库安装包；

b) 解压安装包到/tmp 目 录 ；

c) 提供数据库安装所需的所有必要信息，例如目标位置，用户名，密码等；



1 <http://baike.baidu.com/view/1435992.htm>,RFT-百度百科，2011-05-04

2 <http://baike.baidu.com/view/1479196.htm>,Apache ant-百度百科，2011-12-29

d) 启动数据库静默安装；

e) 验证安装日志，确保其中包含相关的安装成功信息，不包含相关的失败信息；

f) 生成测试报告。

(2)把每一个小步骤分别做成一个个ANT Target。

(3)做另一个ANT Target 去按顺序调用它们，这样就实现了第一步的自动化。

(4)同理，可以实现其余步骤的自动化。

(5)把所有步骤按顺序串起来，就实现了整个测试用例的自动化。

(6)同理，测试计划里的其他测试用例也都可以自动化。

执行者只需要做下面的步骤：

(1)根据每个人分配到的任务来更新一个控制文件，在这个文件里，每一行表明那个 测试用例的执行状态 (true,false)。

例如测试计划里有5个测试用例，要运行其中的1,3,5,那么编写控制文件如下：

TC\_ 1=true

TC\_2=false

TC\_3=true

TC\_4=false

TC\_5=true

然后就可以通过 ANT 来调用执行了。

(2)检查测试结果报告，对于没有通过的部分就可以进行问题分析和缺陷报告了。

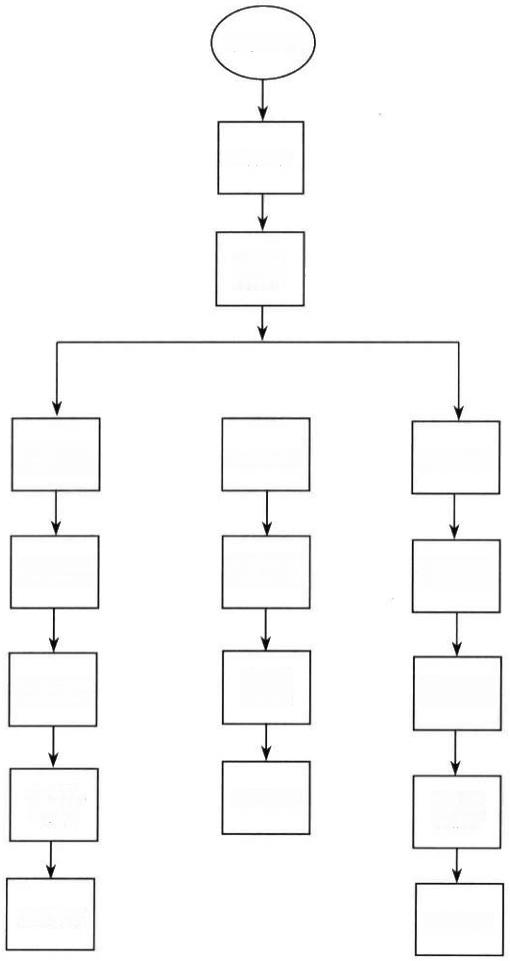
图2-2描述了整个自动化执行的流程。

其中，Case M和 Case N是没有依赖关系的，它们可以以任意的顺序执行。

Casel.m 和 Casel.n 是有依赖关系的，Casel.n 依赖于Casel.m,Casel.n 必须在Case1.m 之后运行。

小艾说：“我现在明白了，对自动脚本的开发看起来挺复杂，需要有相关的自动化工 具的开发经验。但是对于执行人员来说真是简单呀，工作量大大减轻了呀!”

第2章 万事开头难：软件从安装开始



干净的OS

环境备份

读取控 制文件

执行TC2.1

执行TC2.2

生成测 试报告

恢复环境

执行TC1.1

执行TC1.2

执行TC1.3

生成测 试报告

恢复环境

执行TCn.1

执行TCn.2

执行TCn.m

生成测 试报告

恢复环境

图2-2 自动化执行流程

**2.6** **学习笔记——安装测试之小艾观**

经过4个月的实践和学习，安装测试项目已经接近尾声了，小艾从一个新手成长为一 个合格的安装测试人员，对安装测试深有感触。

**1.** **安装测试其实不简单**

以前可能习惯了安装一些简单的桌面应用程序，潜意识里觉得安装是一件很简单的事 情。这次真正接触大型商业软件的安装测试，才真正感觉到安装测试不简单。安装是客户 使用这个软件的第一步，如果安装不上或者不好安装，就会给客户留下不好的第一印象， 从而影响口碑和销量，因此必须认真重视安装测试。

**2.** **一个好的测试计划是保证质量的关键**

产品质量主要是由测试计划来保证的。如何写测试计划真是一门学问，既要保证能够 发现产品的缺陷，又要保证不超过预定的预算并且要按时交付。

**3.** **文档的重要性不低于产品本身**

对于一个复杂大型的商业软件，大多数客户一般只有很少的了解，从而文档成为客户 了解产品的最重要途径之一，文档的好坏将直接决定客户的满意程度和市场的普及程度。 文档的验证工作需要成为安装测试的重要的部分。

**4.** **自动化测试是测试的趋势**

小艾深刻体会到了自动化测试对效率的提高和对人脑的解放，其将是测试的趋势和方向。

**参考资料：**

[1]安装测试-百度百科<http://baike.baidu.com/view/651442.htm>

[2]RFT- 百度百科 <http://baike.baidu.com/view/1435992.htm>

[3]Apache ant-百度百科 <http://baike.baidu.com/view/1479196.htm>

第 **3** 章

发现新方法：快速分发新的 测试版本



新的一周又到来了，这一天小艾刚到公司就被叫进安岩的办公室。

安岩问小艾：“听说过构建组吗?”

小艾点点头说：“知道一点，我之前测试安装的软件包应该就是构建组做的吧?”

安岩说：“你说得很对，我们所有测试和开发项目组用的软件包都是他们做的，这是 软件开发中非常重要的一个环节。我们的项目现在刚好处在非常繁忙的一个阶段，构建组 的压力很大，需要支援，他们的头儿朱莉对我说希望能够往那边派一个有安装测试工作背 景的同事，我觉得你刚好合适。过去做一段时间，对整个项目是一个很好的支持，同时这 也是一个很好的学习机会，能够学到所有Java EE 软件构建的基本知识，以及一些诸如脚 本语言的工作流自动化开发方法，这对于你了解软件开发的整个生命周期是非常有帮助 的。通过构建的工作，你能够了解整个软件的架构与构建，这些知识可是成为测试架构师 必须掌握的知识啊。”

小艾对这份工作很是神往，可是构建组具体做些什么、怎么做，他还不是很清楚。

**3.1** **构建的过程**

下午小艾带着满心的疑惑来到了构建组，他问朱莉：“构建组是做什么的呢?”

朱莉反问小艾：“我们的软件是怎样做出来的?”

小艾回答：“软件开发组的人员开发的吧。”

朱莉笑了：“你只回答对了一半。Java EE的应用软件开发过程是一个很长的生命周期， 从最初的需求分析，到最后的成品交付给市场，中间有很多环节，开发人员写代码只是众 多环节中的一个。产品构建也是中间的一个非常重要的环节，任何软件开发都不可或缺， 关于它具体做些什么,以及怎么做的，你可以先学习一下构建相关的文档。”

在朱莉的指导下，小艾开始了构建项目的学习。通过一段时间的学习，小艾了解到了 构建的一些基本流程。

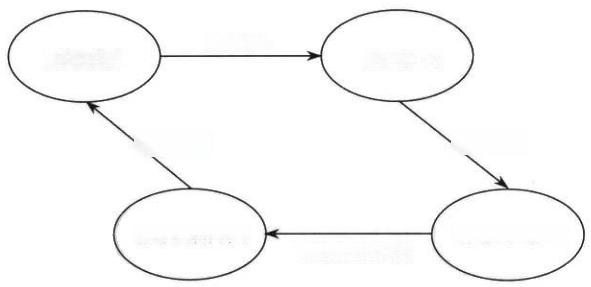
开发人员在他们的个人计算机上编写源代码文件。

他们将编写好的文件存放在一个统一集中的地方。构建组会将所有的源代码编译 成可以在计算机上运行的二进制文件 (Binary Code), 并且用安装工具把各种需要 安装到用户服务器上的文件包装成可以安装到不同的平台 (Platform) 的软件包。 构建的过程就好比是一个组装生产线。源代码文件就好像是各种大小配件，它们 被存储在一个大的仓库里。这些源代码文件中，有些在构建过程中还需要再加工。 经过生产线的各道工序，它们最后会组合成一个产品。

软件的构建生产线不同于大多数工业生产线的地方，它是经常随着产品的需要而改变 的。而且，它不间断地反复生产同一件产品的不同版本，只有最后一个通过了所有测试的 版本会成为提交给用户的最终产品。在那之前的所有产品都是用来支持开发和测试的。图 3-1就是上述构建流程的一个很好的诠释。

图3-1中，源代码仓库是指存放源代码的地方，构建生产线是指构建一个可安装的软 件的过程。如图3-1所示，软件生产过程的原材料是存放在“仓库”里的源代码。构建过 程将这些源代码转换为可安装的软件，并进行构建测试。构建测试后的产品交给各个测试 组进行更为全面的测试。一些源代码被修改以更正测试时发现的问题。修改后的源代码被 存进源代码仓库，并用来构建下一个版本的软件。

第 3 章 发现新方法：快速分发新的测试版本



投入生产

构建生产线

储存入仓 构建测试

修改过的源代码

全面深入的测试

测试版本的产品

源代码仓库

图3- 1 构建过程示意图(软件生产线)

当建立一个新的构建软件“生产线”时，软件开发部门，特别是构建测试团队需要 考虑 ：

储存源代码的“仓库”;

可以反复生产的“流水线”;

快速简单的测试以保证产品可以更全面深入地测试和利用系统备份技术来分享 测试环境。

在朱莉的帮助下，小艾对构建的过程有了 一 个整体的认识，并且对构建的细节，诸如 源代码的管理、构建的环境、构建的方法等，也有了自己的理解。

**3.1.1** **构建原材料的管理——代码的管理**

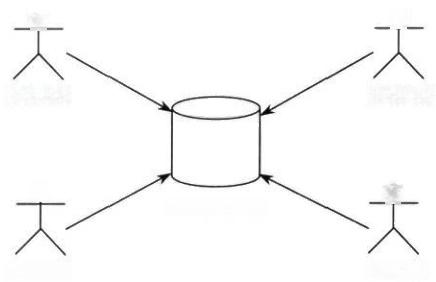
源代码是构建过程的基础，它们也是软件开发公司的宝贵财产，就像所有的工业生产 原材料 一 样，任何软件开发公司首先要想到的就是如何将源代码放在安全可靠的地方。

一 般来讲，源代码会被存放在数据库里，并运用版本控制系统管理源代码。版本控制 系统的软件有很多，例如 IBM 的 Rational ClearCase 和 Rational Team Concert(RTC)[01; 开放源代码的版本控制系统 (Concurrent Versions System,CVS) 和 (Apache SubVersion,

SVN) 等 。

版本控制系统可以用来帮助我们记录文件更改的过程及细节。对于同 一 个文件，每次 被修改后再次存到库里时，并不会替换以前版本的文件，而是将修改过的部分记录下来， 这样修改前后的同 一 个文件会以不同的版本形式存在数据库中。

版本控制系统一般都基于客户端/服务器 (Client/Server) 的结构。如图3-2所示，版 本控制系统可以同时为多个开发人员提供服务。



o

开发人员

o

开发人员

版本控制系统

开发人员

开发人员

o

o

图3-2 版本控制系统运用示意图

版本控制系统还可以进行安全方面的设置，如要考虑到哪些人可以提取或更改哪些文 件等权限控制。有些软件可以将设置精确到每个文件上，这样可以确保重要的文件有更好 的保护。一种通常的做法是为了方便访问权限的管理将存放文件进行分类，可以按照类别 将文件分放到不同的文件夹或组别。一般开发人员只具有那些需要他编辑与修改的源代码 文件的访问权。

版本控制系统一般都具有的功能包括：创建新文件，提取文件，存入新版本文件，协 调或控制多人对同一个文件的同时修改，以及记录文件的修改历史并供查询。通过版本控 制系统对文件保存、提取的管理需要考虑以下一些方面。

**1.** **存入文件的截止时间**

在进行大型软件开发时，由于参与的人较多，通常需要和所有的参与开发的人员约定 一个时间来做全部文件的快照(Snapshot) 。 这样，可以帮助减少各个部门、组织之间不必 要的联络。开发人员只要在约定的截止时间前存入新版本的源代码，这些新版本的代码将 进入下一个版本的测试产品，供测试人员查对验证。而构建者也不用去和每一个开发部门 沟通，他只需预定构建的程序，让构建在约定的时间自动开始。

**2. 源代码文件的阶段** **(Level), 即产品的版本**

有了版本控制系统，软件开发公司部门等构建开发产品的版本很方便。有些工具支持

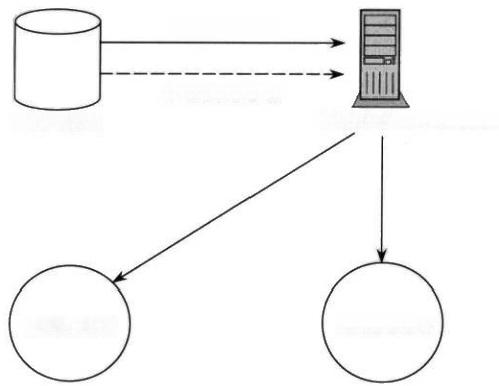
第3章 发现新方法：快速分发新的测试版本

源代码文件的快照(Snapshot), 通常称这个快照为阶段(Level)。作为构建的第一步，应 该针对所有源代码文件做一个阶段。每一个源代码文件在形成阶段时的最新版本会被加入 到快照里。我们将阶段里所有文件提取后，就可以开始构建的过程了，然后会做成所开发 产品的一个版本。

将阶段引入到构建过程，可以让软件开发公司部门等以后随时可以重新构建同一版本 的产品，甚至在有的文件版本改变后也能重构该产品的以前版本。

**3.** **文件的提取方式**

每一次产品的构建都需要从版本控制系统中取出最新的版本或在某一阶段的文件。版 本控制系统记录了文件的改变历史，这有助于软件构建时只取出某段时间或某个阶段里更 改过的文件，用于构建该软件产品的版本。当软件的规模较大时，这个功能特别有用，因 为一般限于资源的限制，整体取出文件有可能需要几小时，而取出更改过的文件有可能只 需几分钟。图3-3所示是构建过程的一个简单描述，可以看到文件提取可以是取出全部， 也可以是只取更改过的文件。



取出所有的文件

取出更改过的文件

存放取出的源文件的服务器

开始静态测试

版本控制系统

开始构建过程

图3-3 文件提取与构建测试示意图

**3.1.2** **构建的环境**

当某个阶段的源代码都已经准备好之后，接下来就可以构建这个阶段的测试产品了。

从菜鸟到测试架构师———个测试工程师的成长日记

构建产品的前提条件是要建立构建的环境。在建立构建环境时，需要考虑的因素主要包括：

选择构建使用的服务器；

选择构建环境平台；

构建所需要的软件或工具。

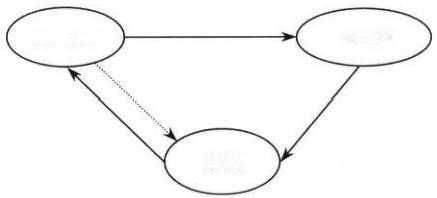
构建环境的搭建在软件开发中的重要性可以通过对图3-4的解析得到证明。图3-4描 述了软件开发的一般流程：

开发人员编写软件代码，然后将源代码交给构建组进行构建；

构建组将源代码文件做成可以安装的软件产品，再交给测试组进行测试；

测试组将测试时发现的问题反馈给开发组；

开发组修改代码，再将修改后的代码交给构建组来进行新版本的构建。



开发人员组 构建组

测试组

图3-4 软件开发人员示意图

如此反复直到测试组成功完成所有的测试，进而构建组构建最终交付市场的软件成 品。从软件开发的流程不难看出，构建过程经常会成为软件开发的瓶颈部分。这是因为在 进行构建的时候，有可能开发人员和测试人员都要等待新的测试版本产品来进行下一步的 工作，这在实际的软件开发过程中非常普遍。所以优化和稳定构建过程是在建立环境时应 重点考虑的因素。大的Java EE产品或者项目的开发中，构建一般都是建立在专属的独立 使用的服务器上的，并且选用最佳配置的服务器来优化构建的速度及稳定性。

图3-4中从开发人员组到测试组的虚线代表开发人员有时将修复的补丁直接交给测试 人员来检验，而不是构建产品新的测试版本。当然，这只是临时方法，不能算测试通过。 如果补丁安装合理正确，大多数情况下，可以帮助减少测试组对构建组的依赖，从而减少 构建过程的瓶颈问题。也可以帮助我们在提交源代码前确认其质量，而减少构建测试的失 败机会。

第3章 发现新方法：快速分发新的测试版本

有了专属的独立的服务器，构建中还要考虑构建环境平台的搭建。一般而言，为了提 高效率、保证安全，可以搭建多个具有相同或不同构建环境平台的服务器。条件允许的情 况下，一般提倡尽量利用多个平行构建过程。多个平行构建的条件包括在同一服务器上有 多个处理器，或有多个服务器可供构建过程用。

根据所构建的产品的需要，一般要安装一些必需的软件或工具以完成构建的过程，例 如编译器及一些自动化工具。

朱莉告诉小艾，在创建构建环境时，还得考虑其他一些因素。

构建环境应与开发环境分离。一般来说，在开发环境中，开发人员也有构建的步 骤，他们可以从他们工作的源代码产生二进制的可运行代码，然后进行单元测试。 但是开发人员所做的构建是零碎的、无规律的(没有计划可遵循，每个开发人员 都可以做自己的构建),这样的构建是没办法满足一个大型的Java EE产品或者应 用开发的整体规划的。

构建环境应与开发环境保持一致性。这是指构建的步骤应尽量与开发环境的构建 步骤采用一样或同等的程序。如此，开发人员用来进行单元测试的代码与构建过 程所产生的代码一样，会减少由于不同的构建方式而带来的软件开发过程中的很 多非软件代码本身的问题。

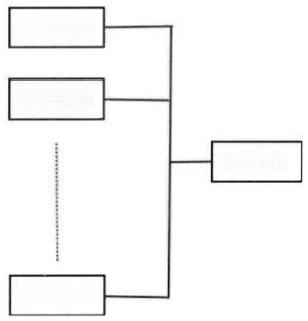
记录好建立及修改构建环境的步骤及细节。需要时，根据记录，我们应该可以重 新建立起一模一样的构建环境，从而可以构建所需构建的产品。通常，在产品进 入维护阶段(发布补丁包阶段)或作为灾难恢复 (Disaster Recovery) 时，会有 重新建立构建环境的要求。

**3.1.3** **整体构建和部分构建**

在设计构建的过程时，一般还要考虑到整体构建及部分构建的需要。总的来说，整体 构建是由许多部分构建所构成的，也就是说，整体构建可以被分解为许多小的构建步骤， 如图3-5所示。

部分构建甚至组成部分构建的更小更细的构建步骤，都应该可以反复多次单独运行。 理想的构建过程是：应可以根据需要，在任何中断或者出现错误的地方能够重新开始构建 的过程。比如，构建在某个步骤失败了，解决问题后，应该可以从失败的步骤开始重新构 建，而不需要从头来过。为了实现这一点，一般应尽量不改变存放源文件的文件夹。如果

需要改变，将改变的文件存在临时文件夹中。如图3-6所示，虚线部分在重复的构建过程 中会被重建，而不影响其他已经完成或未完成的构建步骤。这样一来，构建过程的步骤总 是可以从存放源文件的文件夹的文件开始。



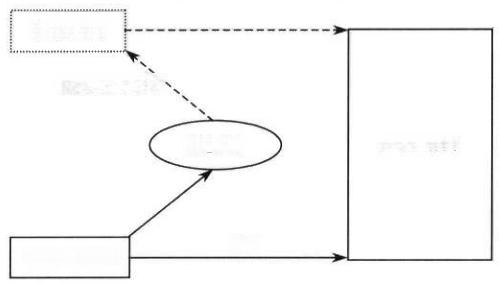
部分构建1

部分构建2

整体构建

部分构建N

图3-5 整体构建和部分构建示意图



临时文件夹

修改或新创文件

构建步骤 产品文件夹

构建

源文件文件夹

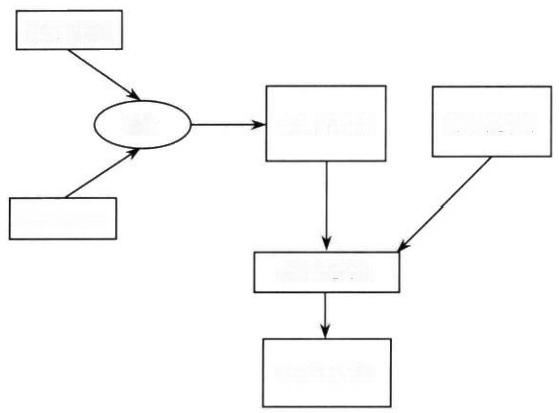
图3-6 构建用例示意图

正式的构建过程也有可能是由部分的构建组成的，特别是当整体构建需要的时间很 长而阶段之间的改变不多时。例如，产品可能会有几百个工程 (Project), 但是在每个阶 段里改变的只有少许工程。这种情况下，可以选择需要运行的部分构建来大大缩短构建的 时 间 。

图3-7显示了从源代码文件变化历史和整体构建结构来产生部分构建结构，从而可以 只做部分构建而完成与进行整体构建一样的功能，也就是完成产品的完整构建。这里无论 是哪种方法，得到的产品都应该是一模一样的。每个源代码文件与构建步骤时间的关系是

第3章 发现新方法：快速分发新的测试版本

设计部分构建逻辑的关键， 一般版本控制系统都支持查询阶段间源代码文件的变化，这保 证了部分构建的可行性。



源文件阶段1

修改文件列表

源文件阶段2

部分构建逻辑

部分构建结构

整体构建结构

比较

图3-7 自动化部分构建过程示意图

总的来说，多个部分构建组合可以构成最后的整体构建。它的好处体现在以下几个方面：

如果有关的源代码自上一次构建没有改变，构建可以被跳过。在每次的构建过程 中，判断每一个部分构建是否需要。如果某一个部分构建的源文件在构建的阶段 里没有改变，就可以略过这个部分构建，从而使整体构建的时间缩短。

部分构建之间如果没有前后顺序的关系，可以让它们同时运行来缩短构建的时间。 部分构建所产生的二进制代码可以直接应用到测试环境来快速检验新的产品功 能，如果测试通过，部分构建代码会进入下一个版本的测试产品。

**3.1.4** **自动化的构建**

为了避免让构建过程成为软件开发的瓶颈，缩短构建的时间和减少构建过程中的问题 是非常重要的。实施自动化的构建是其中一个非常重要的方法。自动化的构建可以保证软 件开发过程中能够定制比较灵活的构建时间表。特别是全自动的构建过程，它可以使得在 非工作时间不用加班也一样可以进行构建，并且全自动的构建也可以确保每一次构建过程

从菜鸟到测试架构师——一个测试工程师的成长日记

的一致性，没有因为人为的错误而引入产品的缺陷。

在实现构建自动化而设计编写构建程序时，与一般的软件开发，特别是自动化程序的 开发一样，需要考虑一些一般性原则：

在源代码文件改变时，比如有新文件加入或有文件被删除，不需要构建程序的改 变。即使有不可避免的改变，也应该使改变的过程尽量容易、简单。

避免把输出或输入的相关参数(例如路径、服务器名、输出的形式或格式)直接 写死在构建的代码中。这样在构建的环境改变时也不需要改变构建的程序。

经常需要改变的一些变量，应采用属性文件(\*.properties) 来统一管理。这样一 来，如果需要改变，只需要修改属性文件中的参数值。

使用样板 (Template) 文件和扩展样式表转换语言 (XSLT), 在构建运行时根据

构建需要生成构建程序文件。这样可以减少在更改构建程序文件方面的投入。

在构建过程自动化程序开发中， ANT¹是一种广泛运用而且非常方便使用的语言。它 是 Java 与命令行的结合，不依赖于系统平台的种类，可以用来实现自动化的构建过程。 当然，还有很多工具可以帮助我们实现产品构建的自动化，比如： Build Forge², Cruisecontrol³,OpenMake Meister⁴,Visual Build⁵等。

**3.1.5** **模拟翻译构建**

一个企业级应用产品往往支持多国语言。因而构建过程也得考虑多国语言的源代码 构建。

源代码文件的翻译一般要等到产品的开发到达一定的阶段之后进行。通常还会将英文



1 <http://ant.apache.org>

2 <http://www-01.ibm.com/software/awdtools/buildforge/>

3 <http://cruisecontrol.sourceforge.net/>

4 <http://www.openmakesoftware.com/>

5 <http://www.kinook.com/VisBuildPro/>

第3章 发现新方法：快速分发新的测试版本

版的文件锁定，以便跟踪自翻译工作开始后英文版文件的改变。这样绝大部分的翻译工作 基本上是基于最后的英文源文件版本进行的，这样可以节省软件开发过程中翻译工作的 开 支 。

这种优化翻译开支的安排却引入了翻译文件不能即时被测试的风险。为了使在翻译文 件存在之前，能尽早测试软件的国际化功能，一般可以进行模拟翻译，并将模拟翻译文件 装到测试系统代替翻译文件进行测试。

为了区别需要翻译的英文文件，可以在版本控制系统中为这些文件做上记号。通过记 号可以将这些需要翻译的文件单独取出来。例如，创建一个文件参数“可翻译”,对于需 要翻译的文件，这个参数设为“是”,而对于其他文件，则设为“否”。在模拟构建过程中， 取出文件时，规定只取出文件参数“可翻译”为“是”的文件。

模拟翻译是通过一种工具产生与翻译后的文件尽可能一致的文件。例如，文件名、翻 译后字符串的长度、文件的编码 (Encoding) 都是模拟翻译工具应该实现的。

模拟翻译构建过程是指运用工具来模拟翻译过程，从英文版本的文件自动产生各种语 言的模拟翻译文件，然后运行部分与翻译文件有关的构建过程，产生一个简化的容易安 装的文件包。它可以被安装到测试环境上来代替翻译文件。图示3-8描述了模拟翻译构建 过 程 。

在模拟翻译构建的实践中，需要考虑以下因素：

选用接近于翻译过程的模拟翻译工具。比如，模拟翻译与翻译过程会产生同样的 字符串长度。这样，它可以帮助我们提前找到由于字符串长度引起的错误。

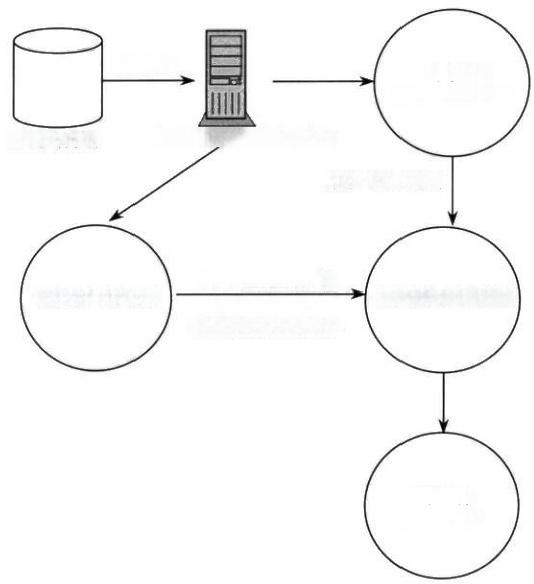
模拟构建的步骤与产品的构建过程要保持一致。要保证模拟构建所产生的文件包 只包括翻译文件，不会引入其他与产品构建不同的其他文件。

由于模拟翻译构建是内部的使用，可以进行简化处理，如把模拟翻译构建产生的 文件包简化为 ZIP 文件包，解压后，直接复制到测试环境或用命令行文件来将文 件包里的文件复制到测试环境，从而节省构建开支，并能够快速响应。

**3.1.6** **构建的范围和频率**

在实际构建过程中，应该根据实际的需求来定义构建范围和频率。对于不同的构建范 围，可以有不同的构建频率。

从菜鸟到测试架构师———个测试工程师的成长日记



取出文件

版本控制系统 存放取出的源文件的服务器

测试产品准备就绪

对于未翻译文件，

增加模拟翻译的文件

构建测试及其他测试

对于已翻译文件，

替换为模拟翻译的文件

呈报支持 全球化问题

构建过程

(包括已翻新文件)

模拟翻译构建过程

图3-8 模拟翻译构建过程示意图

对于大型的企业级应用软件，设计不同的构建范围会带来一些好处。比如，前面提到 的部分构建与整体构建，就是不同范围的构建的例子。可以对不同构建范围的构建过程设 定不同的构建频率。

比如，更频繁的部分构建以加快新的功能测试，较不频繁的整体构建来保证诸如 安装等测试需要。

对于很少或几乎不会改变的源代码的构建，甚至可以将其构建的二进制代码或其 他的构建结果存放在版本控制系统，在每一次的常规构建过程中略去。

软件构建的频率通常是根据软件测试的需要而定的。在敏捷开发模型(Agile) 的开发 环境下，提供频繁的测试产品非常关键。 一般来讲，最大可能地保证构建的频率是软件敏 捷开发模型中的一个很必要的保证。

学习了解了这么多的构建知识，小艾现在可以说已万事俱备，可以对构建测试进行实

第3章 发现新方法：快速分发新的测试版本

践和学习了，这方面朱莉给小艾的指导让他尽快掌握了构建测试的基本知识。

**3.2构建测试(Build Verification Test-BVT)**

朱莉告诉小艾，构建测试也称为构建可接受性测试(Build Acceptance Test),一 般 是 在每一个测试产品生成后，构建测试团队执行一组最基本的测试用例，来确定做成的测试 产品的质量是否达到可以交到各个测试组来进行更全面、更深入的各项测试的要求。

构建测试测试用例基本都是功能测试用例，相对比较简短，应着重于产品的最基本、 最重要的功能。测试用例的选择一般应遵循以下几个原则。

只测试最重要的、最基本的功能，通过了这些功能测试足以让其他的各种测试展 开；

只测试已经测试过而且相对稳定的用例，不要将未稳定的测试用例加入到构建测 试中，新增测试用例只有在功能相对稳定以后再加入到构建测试中。

只有构建测试的顺利通过了，其他测试团队才可以使用新构建的测试产品进行测试。 任何一个构建测试用例的失败都会让测试产品的分发推迟甚至停止，从而影响到其他测试 团队的测试工作。当构建测试失败时，应及时通知相关的开发人员，并让他们尽快提供修 正的代码，从而可以进行重新构建及测试。

构建测试非常重要，它可以让开发人员马上知道新版本的源代码是否可以被成功地构 建成软件产品，同时它还可以帮其他测试团队避免把时间浪费在不稳定的或根本不工作的 测试产品上。理想的构建测试是能够自动化实现的，整个测试时间应该在非常短的时间 内完成，即使是对于一个非常大的Java EE 应用，构建测试的测试时间也不应该超过一个 小 时 。

构建测试通常包括以下几个步骤：

安装测试产品及其需要的其他软件；

进行产品所需要的系统配置；

测试几个最基本的产品功能。

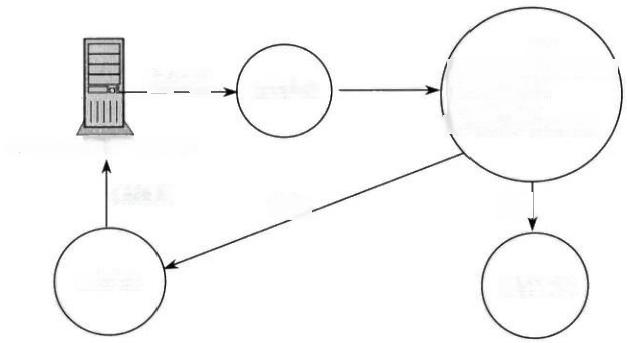
构建测试还包括对构建过程本身的检验，主要内容包括：

确认构建是否包括了源代码文件新的改变；

检验构建的日志是否有报错；

最后产品文件的大小是否有异常等。

图3-9是构建测试的示意图，显示了构建和构建测试在整个产品测试过程中的重要性。



BTV

/1.检验构建过程日志记录

2.安装测试产品

3.产品的配置

4.产品的基本功能测试

存放取出的源文件的服务器

更新文件

成功

其他测试组

开发人员

构建过程

测试产品

失败

图3-9 BVT示意图

在搭建构建测试环境时，需要考虑下面这些常规步骤：

(1)采用一些能实现系统配置自动化的工具，作为构建测试的一部分，自动安装所有 构建测试需要的软件；

(2)使用一些系统备份和恢复的工具，至少备份下面的几种环境：

备份安装好构建测试所需软件的系统。每次开始构建测试之前，将这样的系统备 份直接恢复到测试服务器以节省一部分时间。对于安装及配置较为复杂的测试环 境，或者在构建测试频率比较高的情况下，这种方法是非常有帮助的。

备份构建环境本身。 一般构建环境需要保留足够长的时间。当发现构建测试用例 失败时，可以恢复备份的构建环境，重新构建有问题的编码，这样可以避免版本 的完整构建，减少时间的损失。

构建测试是其他测试类型的前提条件，反复的构建测试失败会严重影响后续的各种测 试的进行。在一个大的Java EE的项目开发中，这会导致测试成本的极大增加。因而构建 测试团队应针对每次构建测试失败来进行原因分析，并根据分析结果，改进现有的开发过

第3章 发现新方法：快速分发新的测试版本

程，以避免类似的问题反复出现。

经过一段时间的学习和实践，小艾对构建测试有了一定的了解和认识，他将构建测试 的内容做了一个小结。

**3.2.1** **构建测试的配置(BVT** **Configuration)**

构建测试的目的是检验测试产品构建过程是否成功完成，构建出的测试产品是否有足 够好的质量可以交给其他各个测试组进行更深、更广的测试。

为了缩短各个组等待新版本的测试产品的时间，在考虑构建测试的环境配置时，应尽 可能选在最为方便快捷的系统上，以保证测试产品能较快地完成安装和部署。例如，如果 产品支持单节点系统和集群系统，我们应考虑在单节点而不是集群系统上进行构建测试。 因为集群系统的安装配置步骤更为复杂，需要花费更长的时间，而且，集群系统往往有更 多潜在的问题，会导致构建测试的失败，从而推迟合格的测试产品发布。一些只局限于集 群系统的问题不应该成为阻止测试产品交付的原因。所以，一般来说，只要测试产品可以 在单节点系统配置环境上正常工作，它就可以发布给其他的测试组进行测试。

另外，如果产品支持多种不同的配置组合，构建测试应尽可能安排在最多客户使用的 系统和配置上来进行。

**3.2.2** **构建测试的用例(BVT** **Scenarios)**

在构建测试的过程中，测试用例的选择是一个关键步骤。

首先，在制定构建测试用例时，需要和其他的测试组有良好的沟通。构建测试团队需 要了解对于所测试的产品，什么是最基本、最核心的功能，这些测试用例可以很少，但足 以让其他的各种测试开始进行。

其次，应考虑根据需要改变构建测试的测试用例，这主要是因为在不同的产品开发时 期，最重要、最基本的功能有可能是不同的。在需要加入新的测试用例时，一定要保证它 是已经测试过而且相对稳定的测试用例。将未经测试或未稳定的测试用例加入到构建测试 中，很有可能导致构建测试的失败，从而使产品测试成本大幅度增加。

最后，构建测试的测试用例的运行时间必须控制在合理的范围内。比如，如果构建的 频率是每天一次，那么构建与运行构建测试用例的时间总和不应该超过一天。

从菜鸟到测试架构师—一个测试工程师的成长日记

**3.2.3** **自动化的构建测试**

像其他的测试类型一样，自动化的构建测试是非常必要的，它可以保证测试过程的准 确性，避免构建测试过程中的人为错误。手工进行的构建测试过程，因为疏忽或粗心，容 易造成一些错误。比如，一些必要的步骤被遗漏了，或者有些程序用了错误的用户来运行。 鉴于构建测试在整个测试过程中占据的重要位置，这些错误有可能导致构建失败，从而成 为整个产品测试的瓶颈。

自动化的构建测试还可以提高构建测试的效率。它可以被安排在自动的构建过程后， 当测试产品构建成功后，立即开始构建测试。通常，可以将自动的构建及构建测试过程安 排在非工作时间，比如，在夜间进行构建及构建测试。理想的情况下，如果构建测试成功， 其他测试小组早上上班就能拿到最新版本的测试产品，从而避免使构建及构建测试过程成 为产品开发过程中的瓶颈。

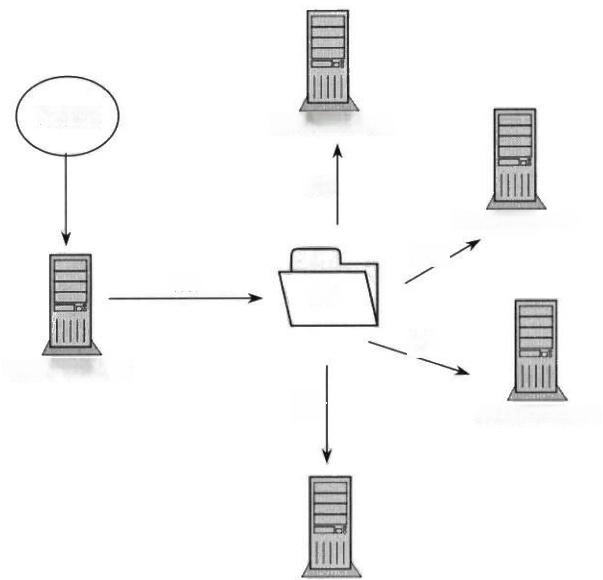
自动化的构建测试也可以保证构建测试过程的一致性和稳定性，可以减少“意外”的 发生。构建测试包括测试环境的搭建，自动化的构建测试可以保证每一次的构建测试都是 在同样的环境设置上进行的，不会因为环境设置的差异而导致测试不同或失败。构建测试 用例应该在新的编码被加到版本前就由有关人员，特别是开发人员进行重点测试，从而提 高这些测试用例在新的测试产品上成功通过的概率。这可以通过与开发人员共享自动化构 建测试的用例来实现这个目标。

**3.2.4** **构建测试的环境再利用**

为了方便其他的测试组，可以把通过构建测试的构建环境的备份文件提供给其他各测 试组使用，如图3-10所示。

构建测试完成后的系统环境，是一个可以再利用的测试环境，它有已安装及配置好的 最新版本的测试产品。其他测试组，比方说功能测试组，他们的侧重点是产品安装以后， 检验是否所有的功能都正确工作，构建测试完成的环境完全满足这样的需求。这些测试组 都希望获得直接可以进行测试的系统环境，而不需要额外的时间和人力花费在测试环境的 准备上。但是构建测试的服务器只有一个，它不能满足所有测试组的需要。解决的办法是 以构建测试服务器为模板，通过系统备份及恢复的流程为各个测试小组创建出他们所需要 的测试环境。图3-10描述了构建测试系统环境的备份及恢复的流程。

第3章 发现新方法：快速分发新的测试版本

构建测试

备份

构建测试的服务器

其他测试的服务器

恢复

其他测试的服务器

恢复

影像

恢复

恢 复 其他测试的服务器

其他测试的服务器

图3-10 构建测试系统环境的备份及恢复示意图

各种常用的服务器都有用于系统备份和恢复的工具。例如： Windows 有 Norton Ghost, VMware 。AIX 有 Mksysb,NIM Server,Soalris 有 Flarcreate,Jumpstart Server。运用这些 工具可以将备份系统恢复的过程自动化，方便测试环境的搭建，从而提高工作效率。

构建测试主要从功能的角度对构建测试产品进行验证。构建测试成功执行是其他测试 开始的前提条件，高效的构建测试可以提高整个测试团队的测试效率。

**3** **.** **3** **静态测试**

与构建测试针对安装了软件产品的系统，主要进行功能测试不同，静态测试是针对源 文件直接做测试分析，发现问题。

**3.3.1** **静态测试的作用及环境**

静态测试适用于在源文件中就能发现问题的情形。比如，我们在编写 Java 程序时， 在源文件中详细地注解所编程序的应用界面，然后，可以用 JavaDOC 工具来自动生成程 序的使用说明书。这些存在于源代码中的注解，通常会有语法错误、错字等，这些错误都 可以通过静态测试的方法去发现。常见的静态测试用例有： Java/Java EE最佳实践或用户 化的规则检验；语法及拼写检验；网页亲和力检验等。

由于构建的环境上有所有最新的源文件，因而在构建系统上进行完整的静态测试通常 是最为容易的。

针对完整的静态测试，根据其不同的需要和特点，既可以将静态测试加入到构建过程 中，也可以将两者分开进行。为了不增加构建过程的时间，原则上，只要是可以分开进行 的静态测试，一般都不考虑加入到构建过程中。为了简化处理，可以将源文件提取出来存 放在一个专用的服务器上，为多个构建过程或静态测试过程分享。如果有些静态测试对于 构建过程有依赖性，比如说，一定要有相对应二进制代码才可以完成的，在不增加很长的 构建时间的情况下，可以考虑加入到构建过程中。

除了完整的静态测试，在一般情况下，开发人员应该在开发环境上安装各种相关的静 态测试工具，针对自己开发的代码部分进行部分的静态测试，以保证源代码文件在存放到 版本控制系统之前，已进行了详细的测试，尽量杜绝和静态测试相关的缺陷和问题。

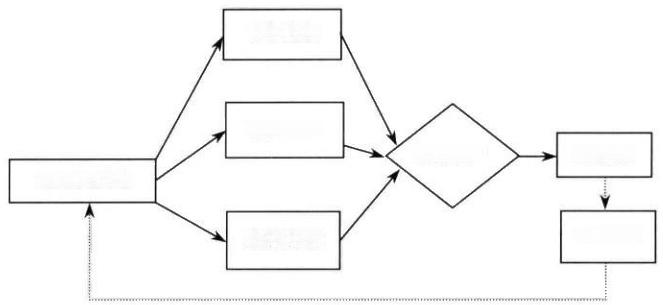
**3.3.2** **自动化的静态测试**

和构建过程和构建测试一样，自动的静态测试可以处理大量的源代码文件，可以节省 大量的人力和时间的投入。

如图3-11 所示，由于静态测试通常会发现多个问题，呈报问题的过程会有一定的难 度。所以理想的静态测试过程也应该运用自动化的工具来发现并报告静态测试中的问题。 一般建议将图3-11中实线所表示的过程都包含在自动静态测试中进行。

第 3 章 发现新方法：快速分发新的测试版本

静态测试1



静态测试2

发现问题? 呈现问题

源文件文件夹

修复问题

静态测试N

图3-11 全自动静态测试示意图

**3.3.3** **静态测试的频率**

静态测试的频率与构建的频率相比，可以稍微低一些。所以，从这个角度也说明了分 开进行静态测试是更好的选择。可以根据不同测试用例的需要来指定相应的测试频率。

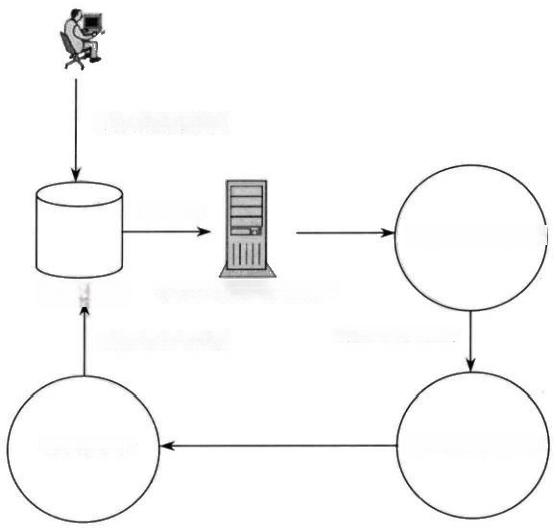
如果静态测试是构建过程的一部分，在建立构建过程时，应将静态测试的任务设置为 可以选择是否需要执行。这样一来，不用每一个构建过程都进行此静态测试用例的测试， 从而减少潜在的构建问题及构建时间。这种安排会减少那些不容易分开进行而且用时较长 的测试用例对构建时间带来的影响。

**3.4不间断的构建与测试(Continuous Integration)**

不间断的构建与构建测试是指不间断、循环往复地进行构建和构建测试，这是一种理 想的构建与测试模式。在这种模式下，构建过程和构建测试必须是完全自动化的，并且保 证缺陷和错误也是能自修复的。在这样的模式下，只要合理制定构建的时间表，构建将不 会成为后面各项测试的障碍或者瓶颈。

图3-12显示了一个完整的构建及构建测试的循环过程。要实现这个循环过程的不间 断，杜绝构建及构建测试失败是关键。

从菜鸟到测试架构师——一个测试工程师的成长日记

开发人员提交源代码

取出文件

开始全自动构建过程

存放取出的源文件的服务器

版本控制系统

开始新版本产品构建

测试产品准备就绪

发布测试产品 开始全自动构建测试

图3-12 不间断的构建与构建测试示意图

**3.5学习笔记——小艾的构建测试观**

通过一个阶段的紧张学习和实践工作，小艾对构建测试有了一个全面且深入的了解。 小艾将构建测试要点总结如下：

构建团队要保证快速地完成产品的版本构建及构建测试，以快速发布新的测试版 本 ；

构建测试团队要建立自动的可以重复的构建过程来实现构建测试环境自动构建； 源代码文件应存放在版本控制系统以保证完全构建及部分构建的可行性；

源代码文件应被提取出来存放到专门的服务器并定期更新，从而保证静态测试的 可行性；

构建测试成功后，测试产品被分发到其他测试组进行更深、更广的测试。

第3章 发现新方法：快速分发新的测试版本

成功的构建测试需要所有开发人员和测试人员的共同努力，确保最新提交的源文件不 会引起任何构建和构建测试失败。自动化的构建及构建测试过程可以分享给相关人员，从 而缩短构建及构建测试的时间，减少构建及构建测试过程中“意外”的发生。这样，新的 版本测试产品可以高质量、迅速地分发出去，从而帮助提高产品开发的效率。

**参考资料：**

[1]<http://ant.apache.org>

[2]<http://www-01.ibm.com/software/awdtools/buildforge/>

[3]<http://cruisecontrol.sourceforge.net/>

[4]<http://www.openmakesoftware.com/>

[5]<http://www.kinook.com/VisBuildPro/>

**第** **4** **章**

把缺陷扼杀在摇篮里： 开发人员的测试



这几天小艾被一个缺陷折磨得生不如死，这个看似简单的缺陷导致小艾手里的大部分 测试案例执行失败。最让小艾无法理解的是，开发人员怎么会把这种具有最基本功能缺陷 的代码录入版本控制器?

“为什么不能把缺陷扼杀在摇篮呢?”

小艾认为，一个优秀的测试不仅应当熟悉产品功能、测试案例，还应当从代码设计的 角度出发去思考问题。但是，对于自己可以做些什么却很茫然。

**4.1怪物来自缺陷，开发人员也需要做测试**

新的版本开发周期刚刚开始，小艾所在的测试组除了做测试计划、讨论测试方案以外， 事情并不多，能开始测试的只有几个基础功能。但是开发组的工作压力明显非常大，整个

第4章 把缺陷扼杀在摇篮里：开发人员的测试

小组都奋战在代码开发和单元测试之中。

**4.1.1** **深入摇篮内部**

“最近忙得焦头烂额，我认为现在测试组还没有完全介入，是否可以安排测试组成员 协助开发进行工作?”开发组组长姚圳提出了自己的思路。

Scrum Master也正有此意，敏捷开发讲究的是团队合作，经常会有开发人员、测试人 员互相支持的情况。小艾正好有机会深入摇篮内部。

小艾来到开发组之后，组长姚圳就给他分了一个用户故事，并和他讨论了用户故事的 需求和开发架构，小艾已经跃跃欲试了。

姚圳：“很好。小艾啊，你也知道，把你从测试组借过来帮忙，就是因为我们的项目 进度很紧，所以你这部分的代码开发及单元测试要在这周五之前完成，这样才能保证功能 测试组下周一开始测试。时间上有问题吗?”

小艾：“我刚才在心里估计了一下，这个用户故事的开发时间大概是3天，周末完成 肯定没问题。”

小艾上一次大段写代码还是在做毕业设计的时候，写代码的时候他总是很 high, 调 试成功之后，总是很有成就感，每次想到这些，他的手就痒痒的。用户故事的需求很清 楚，姚圳虽然是个年轻的组长，但是很干练，又是典型的技术牛人，有他指点，小艾很 快就上手了。

开发时间比他预期的要长一些，到了星期五的下午，代码的开发工作才算完成。小艾 随便跑了几个测试，确定代码的主要分支是没有问题的，赶紧转身对姚圳说：“头儿，我 这部分的代码开发及单元测试完成了，可以进入版本控制器了吧?现在都已经4点半了， 快要赶不上今天的构建了。”

“如果单元测试通过了，就进入版本控制吧。”

“好嘞。”小艾很快将代码放入了版本控制，就等着做进构建，看功能测试的结果了。

第二天小艾就收到邮件，自己刚进入版本控制的代码竟然出现了功能错误，小艾不 由得有些沮丧。即使沮丧，遇到错误也要修正，看缺陷相关描述，按照重现步骤，在自 己的开发环境果然发现同样的问题，原来是自己当时疏忽了对边界值的处理，是个小缺 陷。但这次小艾比较谨慎，问题解决后，他把补丁发给功能测试组验证，补丁测试通过

从菜鸟到测试架构师———个测试工程师的成长日记

才提交代码。

小艾觉得有缺陷是正常的，解决得还算顺利，于是开始处理其他几个转到自己名下 的缺陷。结果花了一下午时间找问题原因，发现这几个缺陷和功能测试组发现的某个缺 陷是源自同一个问题，原来这个功能也被别的用户故事用到了，几个功能测试人员分别 发现了问题，都报告了缺陷。他不得不将这些缺陷重复至之前收到的那个缺陷上。并给 每个人发了补丁，使他们在拿到新的构建之前，不被这个问题影响到进度，能够继续其 他功能的测试。

整个下午功能测试组连续在这个功能上报告了几个缺陷。小艾越来越觉得奇怪，本来 自己在编码的时候考虑得很周到了，在调试模式下也跟踪过代码，为什么还是会有这么多 小缺陷呢?

同时，还有件事情一直让小艾感到奇怪，姚圳完成代码开发的时间经常比小艾晚。作 为一个经验丰富的老开发人员，不应该。但是姚圳那边报出来的缺陷比自己的少，倒是说 得过去。这里边一定有原因。

**4.1.2** **吃一堑长一智，走出单元测试的误区**

姚圳发现了小艾的困惑，他也正想找小艾谈谈。

“小艾啊，有没有发现最近你写的代码缺陷比较多啊?我大概看了一下你的那些代码 缺陷，很多都应该是在单元测试阶段就发现的。上次我问你有没有做单元测试，你不是说 做了吗?”

“是啊，我确实做了啊。”

“嗯，能不能跟我说说，你是怎么做单元测试的呢?”

单元测试是开发人员最容易误解的一个概念。通常，如果我们问一个开发人员，是否 进行了单元测试，他们通常都会说是的，已经做了单元测试。但是如果问他的单元测试的 步骤，答案很可能是：

(1)开发完代码之后，我实际运行了程序，跑了几个功能的测试，没有问题。

(2)我用断点调试进行了代码跟踪。

第4章 把缺陷扼杀在摇篮里：开发人员的测试

虽然这些任务都有价值，但是他们都没有关注到单元测试的最核心价值。有效的单元 测试需要编写简单的、自动化的测试代码，并且几乎是和开发代码同时完成的。

听姚圳讲完这些，小艾不禁更奇怪了，“啊，要写测试代码，还要和开发代码同时完 成，有这必要吗?不是有功能测试组去做测试吗?”

“开发人员做单元测试不仅必要，而且重要。今天我先给你讲讲我们为什么要做单元 测试。至于单元测试和功能测试，二者是有差别的，不能互相代替，这一点我后面会详细 给出解释。”

“每个开发人员都有责任对自己开发的代码进行单元测试。我每次宣布自己开发完成 的时间比你晚，就是因为我所做的，包括了产品代码和单元测试代码，这也是我那边的缺 陷比你的少的原因。”

单元测试的特点和作用如下。

**1.** **保证代码的质量**

当开发人员开发新代码，或者是更改已有代码时，比如添加新方法，修改缺陷，或者 是代码重构等，都要对这些新代码或更改的代码进行单元测试，这样可以保证已有代码不 会受到影响，有效地提高代码的质量。

**2.** **更容易发现缺陷**

单元测试是在系统的最低级别进行的测试，与别的方法或者模块隔离，所以可以很容 易地发现并定位问题。我们都能理解，功能层面的缺陷要比系统集成层面的缺陷更容易发 现并解决。同样的道理，单元测试的缺陷要比其他层面的缺陷更容易发现并解决。

**3.** **可重复执行**

有效的单元测试应该是自动化的，这样可以很容易重复的执行。比如在进行代码重构 时，自动化的单元测试就可以保证代码改动引入的缺陷可以马上被发现，不用等到功能测 试甚至系统集成测试阶段。

**4.** **代码更容易维护**

单元测试还可以起到文档的作用，有效记录了相应代码的功能和用法。这样，哪怕是

从菜鸟到测试架构师——一个测试工程师的成长日记

过了N 年，哪怕是换了别的开发人员，再看到这段代码的时候，也很容易从单元测试用例 理解其功能和用法。如果你想知道如何去调用一个特定API, 会有一个测试告诉你如何做。 如果你想知道如何去创建一个特定对象，会有一个测试告诉你如何做。关于现有系统你想 知道任何事，都会有一个测试演示给你看。测试就像是小型的设计文档，小型的代码样例， 描述了系统的工作和使用方式。大师 Ron Jeffries 在 Essential XP中也强调了单元测试的文 档作用：

Unit tests are also documentation.The unit tests show how to create the objects,how to exercise the objects,and what the objects will do.This documentation,like the acceptance tests belonging to the customer,has the advantage that it is executable.The tests don't say what we think the code does:they show what the code actually does.

**5.** **解决缺陷的成本更低**

这一点是进行单元测试最主要的原因之一。开发人员在单元测试中发现缺陷，可以马 上就解决，这是从缺陷生成到解决的最短的周期。对比一下功能测试中发现并解决缺陷的 周期：

(1)功能测试人员在正式的构建中发现缺陷。

(2)在缺陷跟踪记录系统中创建记录，描述缺陷。

(3)相关的开发人员接到缺陷生成的通知，在开发环境中重现缺陷。

(4)开发人员调试并解决问题，将新代码集成到源代码控制系统中。

(5)功能测试人员拿到新构建，验证缺陷是否得到解决。

而且，如果测试人员对缺陷的描述不够清楚，或者开发人员不能一次性解决问题等， 都会使这个周期变得更长。

“就像你的那几只虫子，因为没有在单元测试中及时发现并解决，最后都经历了这么 长的过程才被消灭。”



1 Essential XP,Ron Jeffries,<http://xprogramming.com/articles/expdocumentationinxp/,Documenting> Code

第4章 把缺陷扼杀在摇篮里：开发人员的测试

“嗯，是啊，不亲身体验一下，还真不理解呢。同一个缺陷，如果被不同的功能测试 人员发现，我要分析每一个缺陷的问题根源，之后每个功能测试人员又都要进行验证，这 些原本都是可以通过单元测试避免的。”

一般来讲，在功能测试阶段解决缺陷的成本，是在单元测试阶段解决缺陷的成本的10 倍；在系统集成测试阶段解决缺陷的成本，是单元测试阶段解决缺陷的成本的40倍。

所以，真正的问题不应该是为什么开发人员要做单元测试，而是为什么不做单元测 试 。

小艾终于明白了为什么开发也需要做测试，原来单元测试有这么多好处。但是马上又 有问题了，“项目时间那么紧，没时间写单元测试啊，不是更加完不成开发任务了吗?”

“嗯，这个问题问得很好，这也是很多开发人员的困惑，或者说是他们不能做好单元 测试的原因之一。”

实际上，开发人员写单元测试的时间几乎和他写产品代码的时间相当。所以，当做项 目计划的时候，把单元测试考虑进去是合理的。尽管单元测试增加了相当大的开发工作量， 看上去开发时间延长了，但实际上对于一个长期不断改进和维护的项目而言，我们不能忽 视级联效应，要从项目整体来看。

小艾突然想到自己当时估计用户故事开发时间是3天的时候，并没有考虑做规范的、 足够的单元测试的时间。

如果单元测试做得不够，功能测试组就不可避免地遇到一些单元测试层面的缺陷，用 来发现并验证这些缺陷的成本增加，没有时间和精力去做更复杂的测试，整个功能测试进 度受到严重影响。

事情还没完，大量的功能方面的缺陷没有在功能测试阶段被发现，被遗留到了系统集 成测试阶段。性能测试小组被这些功能缺陷困扰，没法做足够的性能方面的测试。

更糟糕的是，是虫子，就总会出来捣乱。那些在产品发布前没有被及时发现的功能和 性能方面的缺陷，到了客户的环境一个个地都跳出来了，不仅解决这些客户报告的问题的 成本高昂，更恶劣的后果是影响了客户对产品的满意度。没有足够的单元测试带来的影响 如图4-1所示。

小艾听得背后一阵发凉，直起鸡皮疙瘩。

系统集成测试阶段：

被大量的功能缺陷困扰

· 性能测试组无法投入更多时 间做系统层面的性能测试

客户环境：

发现大量的功能和性能缺陷

●解决客户报告问题的成本 更高昂

●客户满意度受影响，进而影 响收益

功能测试阶段：

发现大量的单元测试层面的缺陷

●功能测试组无法投入更多时间做更复杂 的测试

●功能测试的进度受到严重影响

图4-1 没有足够的单元测试

相反，如果开发人员通过单元测试，保证最基本的缺陷能够在单元测试阶段发现并 及时解决，那么用来解决被流转到后期的测试阶段的缺陷的时间实际上缩短了。所以， 单元测试反而是削减项目整体成本的一个有效手段。如图4-2所示为足够的单元测试带 来的益处。

客户环境：

更多的功能缺陷和性能缺陷

●解决客户报告问题的费用减少

系统集成测试阶段： ●客户满意度更高，进而促进收益 极少数的功能缺陷

●性能测试组可以专注于

系统层面的测试

功能测试阶段：

极少数单元测试层面的缺陷；

更多复杂的功能缺陷

●功能测试表示产品更加稳定，代码 质量更高了

●功能测试的进度得到改善

**图4-2** **有足够的单元测试**

当然也有例外的情况。对于某些特殊的模块，如果写自动化单元测试所投入的精力过 大，和其所带来的风险已经不成比例的时候，就可以不写自动化单元测试用例，用手动的 单元测试来替代，并在单元测试报告中给出说明。

“开发人员眼中的测试肯定和测试组做的测试不一样吧?”小艾又有了新的问题。

第4章 把缺陷扼杀在摇篮里：开发人员的测试

**4.1.3** **潘多拉盒子出来怪物** **—** **—** **开发人员眼中的测试**

姚圳为了让小艾有更加深入的理解，了解开发人员眼中的测试，向小艾详细介绍了开 发人员眼中的测试——单元测试。

**定义：什么是单元测试**

单元测试是由开发人员在开发产品代码的同时进行的一种独立测试，验证其开发的每 个代码单元。

从定义中不难看出单元测试的几个要素如下。

**1.** **谁来测试?开发人员**

对，你没有看错，是开发人员进行单元测试，不是测试人员。如果你知道了单元测试 的目的，就不难理解为什么是开发人员进行单元测试了。

单元测试的目的是要确保程序的逻辑和开发人员对它的预期是一致的。再没有任何其 他人比开发人员自己更了解他对这段程序的预期是什么了。所以要编写有效的单元测试， 开发人员本人是最合适的人选。

“但是，既然开发人员都已经很明确自己的预期了，为什么非要以测试用例的形式验 证这段程序符合预期?直接跑一遍，或者是用断点调试等方式，不是更省时省力?”

“要记住，我们是开发大型的企业应用，不是你上学时候做课堂作业，哈哈。”

我们在开发阶段，会把需求拆分成很多个用户故事，由不同的开发人员负责，但是到 了维护的时候，这批开发人员很可能都被分到新的任务中去了。如果发现原有用户故事中 的缺陷，修改缺陷的人不一定熟悉所有的这些用户故事，更不可能清楚每一段程序本来的 预期!那么在修改某个缺陷的时候，就无法知道是否破坏了某些代码的本来预期。这时， 单元测试就派上用场了，无论世事如何变迁，它永远记得自己的初衷啊!

**2.** **什么时候进行单元测试?开发产品代码的同时**

是的，是和开发产品代码同时进行，不是之后进行。

通常，开发人员采用的方式是开始一段时间内都用来编码，之后利用一段时间进行单 元测试，发现并修改缺陷。这样做的结果将导致有效测试的难度更大。甚至有可能一些人 干脆就不做单元测试了。有效的单元测试要求在编写产品代码的同时进行单元测试，不允

从菜鸟到测试架构师———个测试工程师的成长日记

许写出来的代码放置很长一段时间之后再测试。

实际上在引入敏捷开发理论之后，更高的要求是测试驱动开发，也就是说单元测试代 码要在产品代码之前编写。测试驱动开发是一个比较大的话题，将在后面的4.3节详述。

**3.** **测试对象?代码单元**

呃，啥是单元?

从实用角度讲，单元通常是指类的成员方法。也可以是任何具有明确功能、规格定义、 明确接口定义，并且其规模一般比较小的程序代码模块的组合体。

**4.** **何为独立测试**

“独立”是指将代码从原始程序中隔离出来，尽可能地与程序其他部分或者外界依赖 隔离，针对各个单元单独进行测试。

**小结：理解单元测试要把握的几个关键点：**

◎ 单元测试记录预期的行为

◎ 每个单元测试针对一个单独的行为进行测试

◎ 尽可能地与程序其他部分或者外界依赖隔离

◎ 一旦失败，可以清楚地定位失败的原因

◎ 可以重复运行，且每次运行都有确定的行为，不受上一次运行的影响

◎ 可以快速地执行(10秒左右),简单、实用、高效

◎ 有效的单元测试是自动化的

**5.** **单元测试与功能测试**

虽然单元测试和功能测试的目的都是发现代码中的缺陷，二者却是截然不同的测试类 型，如表4-1所示。单元测试并不是开发人员编写的简单的功能测试用例。表4-1列举了 功能测试和单元测试的区别。

第4章 把缺陷扼杀在摇篮里：开发人员的测试

**表4-1单元测试和功能测试的区别**

|  |  |
| --- | --- |
| **功能测试** | **单元测试** |
| 在开发人员开发阶段完成之后开始 | 整个开发阶段随时进行，与产品代码开发同时 |
| 由功能测试组写测试用例并执行 | 由开发人员写测试用例并执行 |
| 验证产品的功能和行为是否和客户预期一致 | 验证代码段/模块的行为是否和开发人员的预期一致 |
| 针对项目的一部分或者全部进行测试，可以涉及不同 的类与方法，并可依赖外部资源，如文件输入/输出、数 据库操作、网络连接等 | 针对一个单独的方法，尽可能与其他方法或外部资源隔离 |
| 需要在产品环境执行测试，通常需要搭建测试环境 | 在开发环境执行测试，必须是快速的，只能有很少的、最好是 没有任何环境搭建步骤 |
| 黑盒测试，测试包括主要的流程和出错的分支 | 白盒测试，测试包括主要的流程和出错的分支，以及边界条件 |
| 以测试的场景的数量来衡量测试覆盖率 | 以代码量来衡量测试覆盖率 |

**4.2单元测试测点啥，怎么测**

总算有些明白单元测试是怎么回事了。为了让自己更好地把握单元测试的摇篮，小艾 翻起了文档。

**4.2.1** **摇篮有多大——单元测试的范围**

“单元测试要覆盖到边界值和正常输入，也要测试每个方法的出错条件和无效输入。 另外，开发人员还要在单元测试层面保证代码满足国际化、 API 兼容性及可访问性测试 (Accessibility Testing) 的要求。”

单元测试都需要保证：

覆盖到所有新开发代码、修改代码及受影响的代码。

覆盖代码所有分支，包括正常路径和错误路径。

覆盖所有有效的输入/输出情况。

覆盖无效的或预期以外的输入/输出情况。

覆盖所有的日志文件和返回代码。

如果有出错恢复步骤，确保其正确。

验证代码的逻辑正确。

为满足国际化要求，确保在包含虚拟翻译的构建环境中通过测试。

对敏感代码要测试其性能。如果有必要，需要描述代码路径及对数据库的访问。 单元测试需要在开发环境中完成。

修改所有可翻译的代码片段，保证没有错误。

对于 Java 代码和 JSP 源文件，需要经过代码审查，保证其满足代码开发标准和编程 规 范 。

对于JSPs 和 HTML, 需要满足XHTML 标准。开发人员还需要用HTML 验证器来验 证这些文件是否满足标准。

构建代码的时候，也会得到一份报告，任何违反标准的代码都会被反映出来，并有相 应的缺陷来跟踪。

同样的，构建过程中，HTML 验证器报告的错误和警告也会有相应的缺陷来跟踪。

“看来这摇篮还不小呢。一定得系统地、有步骤地做，不然覆盖率一定受影响。”

**4.2.2** **有规范、有步骤地捉虫子——单元测试的流程**

姚圳也发给小艾一些关于单元测试的文档，小艾从单元测试的流程开始仔细研读。

**1.** **单元测试开始的标志**

开发人员接到一份新的设计文档或者一个缺陷后，就需要开始考虑单元测试了。

**2.** **单元测试结束的标志**

全部的新增代码和更改代码都已经完成，并且集成到代码存储库。

代码审核已经完成。

所有的单元测试已经执行并通过，单元测试用例已经集成到测试用例存储库。

第4章 把缺陷扼杀在摇篮里：开发人员的测试

所有的注释已经在代码中编写并通过代码审查。

5 所有已知的问题都已经得到解决，或者是已经提出了解决遗留问题的下一步计 划。

**3.** **程序**

(1)创建/修改代码和相关文档

代码——开发人员接到新的设计文档或者缺陷后，开始研究如何开发新代码或者修改 现有代码，来实现设计或解决缺陷。新代码，或者对现有代码的修改，都应该遵循已有的 编程规范和数据库规范。

Javadoc——开发人员创建或者修改Javadoc 来反映新的代码实现，或者是对现有代码 的修改。新创建的或者是修改的Javadoc 都应该遵循已有的Javadoc 编写规范。

编译——开发人员都有自己的开发环境，所有包括新开发代码或者是修改代码的源文 件必须在遵循编码标准的情况下能够编译通过，不能有任何问题。

(2)对新开发代码或者是修改代码进行单元测试

开发人员和新功能的所有者或者是缺陷所有者一起商量确定单元测试方法。如果开发 人员同时也是新功能的所有者或者缺陷所有者，建议找其他同事结对，一起商讨单元测试 的方法。

不管采用哪种方法，都要保证单元测试的范围。

从每个功能的单元测试代码中选取一定比例，实现全自动化。这些单元测试代码应该 是这个功能的全部单元测试代码中有代表性的子集。自动化单元测试代码使用JUnit 框架， 或者是基于 JUnit 框架衍生的其他框架，如 JUnitEE,JSUnit 等。自动化单元测试代码应 该遵循单元测试规范和最佳实践。这些自动化单元测试代码应该单独进行代码审查，或者 是作为产品代码的一部分一起进行代码审查。

(3)代码审核

所有的代码和 Javadoc 都必须经过代码审核。通常由模块负责人或者经验丰富的人 帮助审核，找出新开发的代码里存在的潜在问题，并检查代码是否遵循编程规范和最佳 实践。

从菜鸟到测试架构师——一个测试工程师的成长日记

(4)版本控制

为了避免新代码或者修改的代码不会破坏构建 (Break Build), 或者破坏已有功能， 或者是对其他部分造成影响，在将代码集成到构建之前，还需要做一定的集成测试，即将 代码部署到最新的构建上，执行最基本的流程。可参考3.2节构建测试的相关内容。

“嗯，嗯。”小艾看到这里，不停点头表示赞同，因为在这个版本开发的最初阶段，每 天都有大量的新代码集成构建，由于缺乏集成测试，经常导致构建失败，功能测试组不能 拿到当天的构建，没法开始测试，小艾当时就曾深受其苦。要是每个开发人员都能够在提 交代码之前，做最基本的集成测试，保证不破坏构建，对整个测试工作的展开会有极大的 帮助。

将产品代码和单元测试代码都集成到合适的存储库。

保留自动化单元测试结果和覆盖率报告，比如建立一个专门的数据库，用来管理单元 测试准则和最佳实践，并存储自动化单元测试报告。另外，在每个版本产品开发之初的综 合测试计划中，会设定自动化单元测试的覆盖率标准，最终的自动化测试报告显示的覆盖 率要满足这个标准。

**4.2.3** **来一套杀虫装备：单元测试的工具**

作为一个测试工具， JUnit 是一个好的开始。 JUnit 是用来测试Java 代码的开源框架。 JUnit 的理念很简单，用它来开发测试用例也很简单。开发人员利用Java 代码来测试Java 代码。在测试 Java 代码的时候， JUnit 提供了一套方法，可以对比实际的返回值和预期返 回值。

为了更好地理解如何创建一个 JUnit 测试用例，下面来看一个实例。这是一个非常简 单的例子，根据每一天不同的时间返回相应的问候信息。早上的问候信息和下午、晚上的 问候信息不同。下面就是待测的Java 类 。

**package** com.ibm.commerce.junit.example;

/\*\*

\*Simple greeting class to return the appropriate greeting to display

\*to the caller.

\*/

**public class** Greeting(

第 4 章 把缺陷扼杀在摇篮里：开发人员的测试

/\*\*

\*Morning greeting.

\*/

**public static final** String GREETING\_MORNING="Good Morning Sunshine!"; /\*\*

\*Afternoon greeting.

\*/

**public static final** String GREETING\_AFTERNOON ="Just a few more hours

before quitting time.";

/\*\*

\*Evening greeting.

\*/

**public static final** String GREETING\_EVENING ="And I'm outta here!";

//static for the hour where morning has expired

**private static final int** MAX\_HOUR\_MORNING =12;

//static for the hour where afternoon has expired

**private static final int** MAX\_HOUR\_AFTERNOON =17;

//holds the time for the greeting

**private** java.sq1.Timestamp iGreetingTime = **null;**

/\*\*

\*Creates the greeting object.

\*/

**public** Greeting(){

**this** (System.currentTimeMillis());

Creates a greeting object based on the specified time.

\*This is a protected method to test the greeting based

\*on different times of the day.

\* **@param** greetingTime The time to base the greeting on. \*/

**protected** Greeting **(long** greetingTime)(

**super**();

iGreetingTime = **new** java.sq1.Timestamp(greetingTime);

}

Returns the right greeting for the time of day.

从菜鸟到测试架构师——个测试工程师的成长日记

\* **@return** The appropriate greeting.

\*/

**public** String getGreeting(){

**if** (iGreetingTime.getHours()<MAX\_HOUR\_MORNING){

**return** GREETING\_MORNING;

} **else if(**iGreetingTime.getHours()<MAX\_HOUR\_AFTERNOON){

**return** GREETING\_AFTERNOON;

} **else** {

**return** GREETING\_EVENING;

**1. 创建一个JUnit 测试用例要遵循以下几点**

(1) 一 个JUnit 测试类需要继承junit.framework.TestCase **package** com.ibm.commerce,junit.example;

**import** java.sql.Time;

**import** junit.framework.TestCase;

/\*\*

\*Test driver to test the Greeting object based on different times of the day.

\*/

**public class** GreetingTest **extends** TestCase {

}

(2)setUp() 和 tearDown() 为每个测试方法准备或销毁测试装备

/\*\*

\*Setup procedures before executing the test case.This

\*sample does not require any.

\*/

**protected void** setUp() **throws** Exception

**super.** setUp();

}

/\*\*

\*Teardown procedures after executing the test case.This

\*sample does not require any.

第4章 把缺陷扼杀在摇篮里：开发人员的测试

\*/

**protected void** tearDown() **throws** Exception(

**super.** tearDown(); }

( 3 ) 创 建public 方 法 ， 方 法 名 以“test” 开 头 ， 比 如“testMorningGreeting()”, 用 assert() 判断实际返回值和期望值

JUnit 框架会执行所有的测试方法，无论前一个测试方法是否执行成功。如果一个测 试方法执行后没有抛出异常或错误，就认为该测试成功。JUnit框架引入断言(Assertion) 工具类，它可以比较不同的对象，如果比较结果失败，则会抛出异常。TestCase 抽象类扩 展了断言工具类。

每个测试方法都通过执行一系列简单的操作，使用 Assert() 方法来比较期望结果和实 际 执 行 结 果 。

// ------------ Test Cases

/\*\*

Test case to verify the morning greeting. The time tested is \*9:00 am.

\*/

**public void** testMorningGreeting() **throws** Exception

Time time= **new** Time(9,0,0);

Greeting greeting = **new** Greeting(time.getTime());

assertEquals("the morning greeting expected",Greeting.GREETING\_ MORNING,greeting.getGreeting());

/\*\*

\*Test case to verify the afternoon greeting.The time tested is \*12:00 pm.

\*/

**public void** testAfternoonGreeting() **throws** Exception (

Time time= **new** Time(12,0,0);

Greeting greeting= **new** Greeting(time.getTime());

assertEquals("the afternoon greeting expected",Greeting.GREETING\_ AFTERNOON,greeting.getGreeting());

/\*\*

\*Test case to verify the evening greeting.The time tested is

从菜鸟到测试架构师 — — — 个测试工程师的成长日记

\*5:00 pm.

\*/

**public void** testEveningGreeting() **throws** Exception(

Time time = **new** Time(17,0,0);

Greeting greeting= **new** Greeting(time.getTime());

assertEquals("the evening greeting expected",Greeting.GREETING\_ EVENING,greeting.getGreeting());

}

(4)super(testMethod) 为每个测试方法生成实例。首先执行 setUp(), 然后执行 testMethod(), 最后执行tearDown()

/\*\*

\*Creates the test driver to test the Greeting class.

\* **@param** testMethod The method of the test case to execute. \*/

**public** GreetingTest(String testMethod){

**super** (testMethod);

**2. 使用JUnit测试套件组织想要执行的测试用例**

例如，假设你已经有AddJobCmdImplTest 、RemoveJobCmdlmplTest 等 TestCase的子 类别，如果想要一次运行这两个测试，方式很简单。

JUnit 测试套件用于组织一组测试用例，比如将一个 component 的所有测试用例组织 在一起。一般每个单元测试包里都有一个叫做AllTests的类，代表一个测试套件。例如：

/\*\*

\*The test suite for testing the commands written to configure the scheduler. \*/

**public class** Al1Tests {

/\*\*

\*Returns the test suite representing all the tests for administrating the scheduler.

\*@return The scheduler administration test suite.

\*/

**public static** Test suite(){

TestSuite suite = new TestSuite("Test for com.ibm.commerce.

scheduler.commands ");

//$JUnit-BEGINS

suite.addTest(new

suite.addTest(new //$JUnit-END$

第4章 把缺陷扼杀在摇篮里：开发人员的测试

TestSuite(AddJobCmdImplTest.class));

TestSuite(RemoveJobCmdImplTest.class));

return suite;

注意测试用例执行的顺序。在测试套件类，可以指定测试用例执行的顺序。如果某个 测试用例与另一个测试用例有依赖关系，确保它们的顺序是正确的。尽管如此，仍然建议 每个测试用例是完全独立的，不依赖于其他的测试用例。

**3. 其他各种** **Unit**

推荐的自动化单元测试工具还包括 JUnitEE,DbUnit,JsUnit,Rational PureCoverage 和JTest 等。实际项目中，可以根据所开发代码的类型来选择相应的自动化单元测试工具。 可以很容易地找到详细介绍这些工具的书籍和文档，这里不再赘述。

JUnitEE 是 JUnit 测试框架的一个扩展。使用JUnit 来测试Java EE应用程序仅仅是理 想化的，因为应用服务器不能运行这些测试。而且，在测试用例中构建必需的客户端属性 是一种枯燥的过程。幸运的是，JUnitEE 能够方便地用来测试你的 Java EE 应用程序。 JUnitEE 是 一个JUnit TestRunner,JUnit TestRunner是使用HTML/XML 作为用户界面表示 的 servlet。这意味着JUnitEE 提供了servlet 作为你的测试用例的入口点，并以HTML/XML 的形式得到所需输出。

DbUnit 是一个基于JUnit 扩展的数据库测试框架。它提供了大量的类，对与数据库相 关的操作进行了抽象和封装。它通过使用用户自定义的数据集及相关操作，使数据库处于 一种可知的状态，从而使测试自动化、可重复和相对独立。

JsUnit 是一个客户端JavaScript 单元测试框架。它其实是JUnit 给 JavaScript 的一个口 岸。熟悉JUnit 的人应该很容易学会如何使用JsUnit。

Rational PureCoverage 用来检查单元测试的代码覆盖率。

JTest 是一款用来做静态分析的工具。

**4.2.4** **单元测试够不够，我来把把关**

为了保证单元测试通过并且有足够的覆盖率，小艾所在的组要求开发人员提供两份单

从菜鸟到测试架构师———个测试工程师的成长日记

元测试报告，保存在指定的地方。

(1) 一 份报告是JUnit 运行结果报告，如图4-3所示。开发人员需要在宣布代码开发 及单元测试结束的时候提供这份报告，表明所有的单元测试都已经通过。如图4-3所示为 一份 JUnit 运行结果报告。

|  |  |
| --- | --- |
| Problems Tasks Properies Servers Console,JUnit× earch | |
| mзneа ahвг128О422 зесопаз | |
| Runs:573/573 BErors:0 D Fallures:0  □°Failures Hierarchy 三Falure Trace | |
| 日Test for com ibm.commerce marketing facade chent  com.ibm.commerce marketing facode cient ExperimentActivityTest 田啪com.ibm.commerce marketing facade.chient ESitesTest  \*均com.ibm.commerce marketing facade clientActivityStatesTest 田国com.ibm.commerce marketing facade.cient RuntimeFlowTest  ⑧咆com.ibm.commerce marketing facade cient FiteningAndOrderingTest 闭怕com.ibm.commerce marketing facade.cient TimeBasedActivityTest 由com ibm.commerce marketing facade cient ActivityServiceTest |  |

图4-3 JUnit 运行结果报告

(2)另一份报告是单元测试覆盖率报告。这份报告需要显示测试覆盖率达到了30%。 如果没有达到这一目标，需要给出原因。这一数字因不同项目不同的特点而不相同。需要 根据经验得出一个投入产出比最优的单元测试覆盖率指标。

同时，为了保证单元测试更加有效，也避免功能测试和单元测试出现重复的测试用例， 一个最佳实践是让功能测试审核单元测试报告。

在审核单元测试报告的时候，需要注意以下几点：

手动的单元测试报告需要有详细的步骤，包括使用到的数据集。

自动化的单元测试报告不仅要有运行结果，还要给出描述，说明这一组单元测试 所测的是哪一部分代码。可以使用Javadoc。

关于单元测试覆盖率，要保证：

所有的新增流程都已经被覆盖到；不论是正常流程还是异常流程，都要求完全覆 盖到每一分支。

覆盖有效和无效输入/输出。比如边界输入值；比如数据类型，在期望整型的地方 输入浮点型等。

覆盖日志信息。

第4章 把缺陷扼杀在摇篮里：开发人员的测试

覆盖可接受性测试 (Acceptance Test) 涉及的全部的流程。

对于Services, 覆盖每一个API。

覆盖全球化和国际化的要求。

无法通过UI 验证的功能点，也要在单元测试中覆盖。

审核完单元测试报告之后，功能测试人员需要把所有在单元测试报告中发现的遗漏点 都以审核意见的形式给出，并和开发人员一起分析：

哪些应该在单元测试阶段覆盖的测试点被遗漏了。

这些被遗漏的测试点应该由谁来执行，在哪个阶段执行，也就是应该被添加到单 元测试中，还是可以在功能测试中被覆盖。理论上，属于单元测试范畴的测试用 例，毫无疑问应该加到单元测试集合里。但是结合项目的时间安排和工作量的分 担来考虑，可以有例外的情况。即在功能测试中覆盖被遗漏的测试点。

所有审核意见都解决了之后，就可以允许单元测试退出。

**4** **.3测试还能驱动开发(TDD)**

学会了单元测试之后，小艾的开发更有效率了，缺陷密度下降了。但是有时候，他发 现代码的耦合度比较高，不容易隔离，编写单元测试还是有难度，有时候只能放弃为这样 的代码编写单元测试，测试覆盖率一直上不去，郁闷。

他去单元测试数据库里看了姚圳的单元测试报告，着实吓着了，姚圳的单元测试覆盖 率高出他一大截，而且几乎是100%覆盖!“姚圳一定是留了一手，有什么高招还没教给我。” 小艾决定找时间再去跟姚圳套套近乎。

开会的时候，产品负责人带回来了beta 版的用户反馈。

“姚圳，你这部分的用户故事，客户给了一些反馈，他们的业务上有一些调整，所以 现在已经实现的部分，需要做一些改动。另外还要增加一些新的功能。”

小艾听到这里差点叫出来，他甚至不敢想象在后期阶段做这么大的变动会有什么后 果。他曾听其他测试同事讲过，最悲剧的就是在项目后期出现大的设计上的变动，已有功 能一定会受到影响，回归测试缺陷层出不穷，开发测试人员加班加点地解决这些缺陷，项

目的交付时间也可能受到影响，噩梦啊。

“别担心我这部分了，TDD 能应付得了这么紧急的变化啊。”姚圳看着反倒是很轻松的 样子。

直觉告诉小艾，这个TDD 估计就是姚圳留的那一手。

原来 TDD 是测试驱动开发，姚圳一直在实践测试驱动开发，准备积累一些经验之后 在组里推广。见小艾这么感兴趣，姚圳决定先给小艾洗脑。

**4.3.1** **什么是测试驱动开发，测试驱动的工作流程**

测试驱动开发每次针对一个很小的功能点，通常是小到一个单独的方法。流程是：

(1)在实现新功能之前，先考虑代码的使用需求(包括功能、过程、接口等),为其 编写测试代码。

(2)让新写的测试代码和已有的测试代码一起运行。大师 Kent Beck教导我们，尽快 地让测试程序可运行是压倒一切的中心任务。

但是在这一步，毫无疑问，新测试代码运行会失败，因为我们根本还没有实现任何 功能。

(3)为新功能编写最少的实现代码。切记，切记，是最少的实现代码。

(4)再次让新测试代码和已有的测试代码一起运行。根据运行结果修改实现代码，直 到全部测试代码通过。

(5)在此过程中，积极地对代码进行重构，使底层设计和实现更加优化，使接口更加 简单。

(6)重复前面(1)至(5)步，直到完成全部功能的开发。

不可运行/可运行/重构——这就是测试驱动开发的口号。²

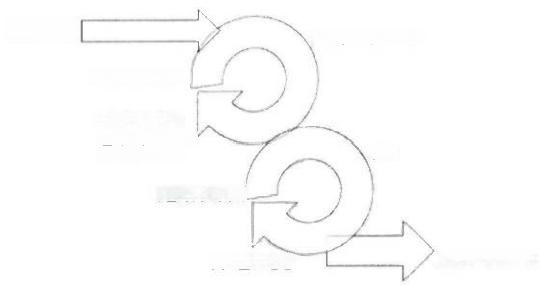


1 Test-Driven Development By Example,Kent Beck著, 孙平平，张小龙，赵辉等译，崔凯校，P25, 2004年

2 Test-Driven Development By Example,Kent Beck著, 孙平平，张小龙，赵辉等译，崔凯校，P4.

第4章 把缺陷扼杀在摇篮里：开发人员的测试

测试驱动流程如图4-4所示。



新功能需求[ 写最少的实现代码

**写单元测试代码**

新测试代码和

已有的测试代码

一起运行通过 重构

与已有代码集成

测试重新通过 完成新功能需求

图4-4 测试驱动开发流程

**小结：什么是测试驱动开发**

测试驱动开发是一种编写软件的模式，是一种敏捷开发实践。其指导思想就是让开发 人员在编写功能代码之前，先根据需求编写测试代码。先思考如何对将要实现的功能进行 验证，并完成单元测试脚本的编写，然后编写足够，仅仅是足够的功能代码满足这些测试 用例，直至通过测试。按照这个方法，递增地在迭代中增加新功能的单元测试和功能代码 编写，直到完成全部功能的开发。

“哦，我们以前的做法通常是先编写代码，在代码中融入有关代码如何工作的想法， 然后编写单元测试来测试这些代码。而测试驱动要求我们反过来做，强调先进行测试，只 有在编写了测试并运行失败之后，才可以编写代码。两种做法最大的区别就在于测试和开 发的顺序，所以是内容相同，形式不同而已吧?”小艾开始比较测试驱动开发和以往的先 开发后测试的区别。

“此言差矣。两种做法不仅形式不同，内容和效果也不完全相同。测试驱动开发和先 开发后测试两种做法，虽然都对所开发的代码进行了测试，验证了代码的正确性，但是 如果我们比较代码的可测试性、内聚性、可复用性，接口的简洁程度，代码中缺陷的密 度及调试的难易程度等方面，就会发现，测试驱动开发已经不再是单纯的测试行为，而 是上升到了一种设计行为。或者说，测试驱动开发的目的不是为了验证代码实现，而是 为了描述一段代码的用途和用法的设计规格说明，而且这种描述是无二义的，是可执行 验证的。”

**小结：测试驱动开发与先开发后进行单元测试的异同**

**相同点：**

(1)都对底层功能进行了测试验证。

(2)都得到了单元测试资产。

(3)使项目容忍变化，可通过单元测试来保证引入的变化不会带来负面影响。 **不同点：**

(1)测试的覆盖率不同。

测试驱动开发要求开发人员首先考虑全部可能的测试，所以几乎可以达到100%的测 试覆盖率。

先开发后进行单元测试的情况下，开发人员完成测试覆盖的动机不足，覆盖率相对 较低。

(2)代码可测性不同。

很明显的，测试驱动开发中实现的代码都是为了满足失败的测试而写的，所以天生具 有可测性。

而先开发后进行单元测试则不能保证这一点，这也是使得测试覆盖率有限的原因之

*一。*

(3)对需求的明晰程度不同(笔者认为这是最主要的不同)。

需求包括用户功能需求，也包括对代码的使用需求。

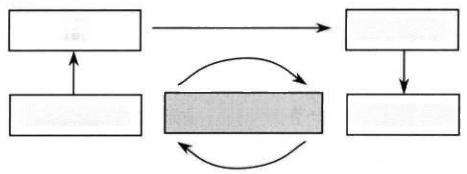
测试驱动开发中，编写测试代码的过程就是从代码级别对需求逐渐明晰的过程，包括 功能分解、使用过程、接口的设计等。

如果先开发后进行单元测试，虽然编写程序之前也会进行设计，但是不论是绘制模型， 还是编写设计文档，亦或是某些大牛在大脑中构思的设计，都不可能精确到代码级别，而 且也没人能承受得了精确到代码级别的文档设计的成本。实际编码实现的时候，就会发现 实际情况和之前的这些设计不太一样，到了后期还要对某个类或者函数的接口进行修改或 者扩展的“杯具”也就会经常发生。

第4章 把缺陷扼杀在摇篮里：开发人员的测试

**4.3.2** **测试驱动开发好处多**

测试驱动所倡导的可测试的代码、单元测试覆盖、重构和更优化的设计之间，是互为 因果、良性循环的行为，如图4-5所示。

更优化的设计

重构

单元测试覆盖率 测试驱动开发

可测试的代码

图4-5 测试驱动开发的好处

测试优先使得代码天生具有可测性，也因此保证了几乎100%的单元测试覆盖；代码 中的缺陷密度低，有利于更早地发现缺陷，方便调试。

全面的单元测试覆盖使得开发人员在项目的自始至终都有足够的信心对代码持续地 进行重构。不会出现因为担心修改会影响到已有功能而畏手畏脚，该修改的地方不敢修改 的情况。

重构是“一种严格的技术，可以重新构造现有代码体，修改代码的内部结构，但是不 会影响代码的外部行为”'。重构可以消除重复设计，优化设计结构，使接口更简单，产 生高内聚、低耦合的代码。代码复杂度降低，更便于维护。如果你理解了测试驱动开发并 不是单纯的测试行为，而是设计行为，那么应该很容易想到，测试驱动开发的最大好处， 不是最终得到的单元测试资产，也不是使单元测试全部通过，而是通过不断对代码进行重 构，而最终从设计层面对代码做出的改进。

接口和设计简洁，高内聚、低耦合的代码无疑是可测性更高的代码。

测试驱动开发是敏捷开发的基础。敏捷开发中，项目的节奏更快。在项目开始的时候， 信息很少，可能只有很少的需求是确定的，所以项目中出现各种变化也就再正常不过了。 预先做好的详尽设计，也就很可能会因为信息不足或者随时可能出现的变化，而变得不再 适 用 。



1 Refactoring:Improving the Design of Existing Code Martin Fowler et al.,Addison-Wesley,1999年

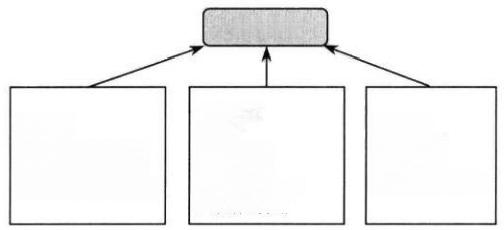
从菜鸟到测试架构师——一个测试工程师的成长日记

但是在敏捷开发项目中，这种变化是自然的，也是受欢迎的，因为我们在整个过程中 都处于不断的学习和探索的状态中。随着项目的进行，我们的知识也在不断增长，因此我们 会增量地做决定，通过实验和探索来解决问题。敏捷开发不会去消除变化，而是通过提高适 应性、灵活度及对变化的反应能力，去尽可能地消除变化对效率和质量带来的负面影响。

任何变化都可以容易加入，包括更优的设计考量，包括那些来自客户的紧急的需求变 化，应对这些变化，只需添加测试，编写使测试通过的代码，重构，就可以搞定了。决定 设计，尤其是底层设计的时间也因此推到了最晚，并且可以随时改进设计。

敏捷开发还主张消除浪费。客户不需要的即是浪费。测试驱动开发使得开发人员对需 求更明确，每一个测试对应一个需求，每一行代码满足一个测试要求。做到了这一点，就 很难有浪费，不允许编写和测试无关的代码，就更加不可能有客户不需要的功能。

这些测试驱动开发和敏捷开发的关系总结如图4-6所示。



更加敏捷

代码质量 ●耦合

●接口简洁

●设计简洁

●代码复杂度低

●缺陷率降低

支持变化

●支持项目后期变化

●推迟做决定的时间

· 支持紧急的需求变化

客户关注

●跟踪需求

· 减少浪费

· 更好的理解需求

**图4-6** **测试驱动开发和敏捷开发**

“嗯，我明白了，测试驱动有这么多好处啊。但是为什么没见到它被广泛使用呢?”

“测试驱动开发实际上是一种观念的改变，和其他有变革意义的观念一样，在起初的 时候会遇到一些阻碍，需要给时间慢慢接受。”

测试驱动开发确实需要有一段前期的学习时间成本的投入。我们的经验是，通过一个 项目周期，就能从测试驱动开发中收到直接的益处，相比较而言，成本、时间及资源上的 投入都是值得的。

和很多有价值的创新一样，测试驱动开发要求我们从观念上改变对软件开发的认识和 做法。测试驱动开发是一种创新，它始终把客户的利益放在首位，它能够帮助我们更有效 地合作。

还有一些阻碍来自项目实际。的确有些东西天生就难以构建单元测试，比如 Web 应

第4章 把缺陷扼杀在摇篮里：开发人员的测试

用的用户界面，比如C 语言编写的代码，因为支持这方面的单元测试的工具确实很少。这 种情况下，可以从相对容易开展测试驱动开发的部分着手，把测试驱动开发和其他测试方 法相结合，比如可能需要一些必要的手工测试。同时这些不方便开展测试驱动开发的地方， 也正是我们进行创新的动力和源泉。

所以，尽管现在我们周围的开发组里，真正做到测试驱动开发的并不多，但是只要测 试驱动开发的好处是显而易见的，就应该毫不犹豫地开始采用，变化也许就是从我们开始 的，我们成功的例子可以鼓励其他开发组采用测试驱动开发，随着我们一起分享测试驱动 开发的知识和经验，星星之火就可以燎原了。

**4.4学习笔记——单元测试之小艾观**

小艾回想起在开发组的这些日子，从自己第一次做的不正规的单元测试，到后面的测 试驱动开发，真是既长了见识，又长了本事。重新梳理一下单元测试的主要内容吧。

**1.** **什么是单元测试**

单元测试是由开发人员在开发产品代码的同时进行的一种独立测试，验证其开发的每 个代码单元。

**2.** **单元测试的特点和作用**

保证代码的质量；

更容易发现缺陷；

可重复执行；

代码更容易维护；

解决缺陷的成本更低。

**3.** **单元测试测什么,用什么测**

单元测试要覆盖到边界值和正常输入，也要测试每个方法的出错条件和无效输入。另 外，开发人员还要在单元测试层面保证代码满足国际化、API 兼容性及可达性测试的要求。

实际项目中，可以根据所开发代码的类型来选择相应的自动化单元测试工具，包括 Junit,JunitEE,DbUnit,JsUnit,Rational PureCoverage 和 JTest 等。

为了保证单元测试通过，并且有足够的覆盖率，最佳实践要求开发人员提供两份单元 测试报告。一份报告是JUnit 运行结果报告。另一份报告是Rational PureCoverage得到的 单元测试覆盖率报告。同时，为了保证单元测试更加有效，也避免功能测试和单元测试出 现重复的测试用例，一个最佳实践是让功能测试人员审阅单元测试报告。

**4.** **什么是测试驱动开发**

测试驱动开发是一种编写软件的模式，是一种敏捷开发实践。其指导思想就是让开发 人员在编写功能代码之前，先根据需求编写测试代码。先思考如何对将要实现的功能进行 验证，并完成单元测试脚本的编写，然后编写足够，仅仅足够的功能代码满足这些测试用 例，直至通过测试。按照这个方法，递增地在迭代中增加新功能的单元测试和功能代码编 写，直到完成全部功能的开发。

**5.** **测试驱动开发的优点**

代码天生具有可测性，保证了几乎100%的单元测试覆盖；代码中的缺陷密度低，有 利于更早地发现缺陷，方便调试。

全面的单元测试覆盖使得开发人员在项目中自始至终都有足够的信心对代码持续地 进行重构。

重构可以消除重复设计，优化设计结构，使接口更简单，产生高内聚、低耦合的代码。 代码复杂度降低，更便于维护。测试驱动开发的最大好处，不是最终得到的单元测试资产， 也不是使单元测试全部通过，而是通过不断地对代码进行重构，而最终从设计层面对代码 做出的改进。

接口和设计简洁，高内聚、低耦合的代码无疑是可测性更高的代码。

**参考资料：**

**[1]Essential XP,Ron Jeffries**

<http://xprogramming.com/articles/expdocumentationinxp/>

第4章 把缺陷扼杀在摇篮里：开发人员的测试

<http://www.junit.org>

<http://www.junitee.org/>

<http://www.dbunit.org/>

[2]Test-Driven Development By Example,Kent Beck著, 孙平平、张小龙、赵辉等译， 崔凯校

[3]Refactoring:Improving the Design of Existing Code,Martin Fowler et al., Addison-Wesley,1999 年

[4]林星. 《软件质量之路(3):测试驱动开发》 .2004年3月01日

[5]<http://www.ibm.com/developerworks/cn/linux/software_engineering/l-frmwk/index3.html?> ca=dwcn-newsletter-linux

**第** **5** **章**

**黑色盒子：打着手电抓虫子**



“黑夜给了我黑色的眼睛，我却用它寻找光明”。经过一年多的锻炼，小艾被轮转到了 功能测试部门，开始了新的测试旅程。这天，小艾一直在思索着诗人顾城这首只有两行的 诗的含义。贺晨是一个被大家称为 IT 文化人的高级工程师，已经有超过5年的测试工作 经验。这首诗就是他告诉小艾要去理解的，他还说，黑夜——眼睛——光明，就是他理解 的黑盒测试的精髓所在。小艾最初很糊涂，不明所以，直到他抓到了第一条虫子之后……

**5.1黑色的盒子里面有什么**

相较于之前驾轻就熟的单元测试，一上来做功能测试，小艾还真有些找不到门道。以 前边读代码、边调试、边找问题，多爽啊!现在可倒好，只能对着二进制的可执行程序， 小艾有点不知如何下手。难怪这叫黑盒测试，真是如同黑夜一样，什么也看不到。黑色的 盒子，你的里边到底是怎么回事?我怎样才能找到虫子(缺陷)呢?

带着这个疑惑，小艾开始了测试旅途新的探索，在贺晨的帮助下，很快，小艾开始了 他的功能测试的第一课。

第5章 黑色盒子：打着手电抓虫子

**5.1.1** **小艾的功能测试第一课——准备手电**

贺晨告诉小艾，功能测试很简单，但同时也很复杂。简单到下面的一句话就可以描述 什么是功能测试。复杂的是要做到这句话所说的可不是件简单的事情。

功能测试，简单的理解就是黑盒测试，就是检测黑盒子，抓到里面的虫子(缺陷)。 所以你要有一个好用的手电，作为自己的眼睛，来划破黑夜，找到虫子，寻到光明!

其实，对于小艾来说，功能测试既陌生也熟悉，熟悉是因为在以前的大学课程里学过 功能测试的概念，陌生是因为他并没有在一个项目中做过哪怕一次功能测试。好在贺晨给 小艾制订了学习计划，如表5-1所示，参照这样的学习计划，小艾开始了他的功能测试的 第一课，开始寻找属于自己的手电。

**表5-1功能测试新人学习计划**

|  |  |
| --- | --- |
| **任** **务** | **目** **的** |
| 对于产品的学习 | 了解产品的基本功能、架构、模块及主要商业流程，学习所要测试部分的设计 和需求文档，对产品的局部进行深入了解 |
| 熟悉各种测试文档 | 学习测试计划、测试场景、测试用例等相关文档规范及内容 |
| 了解功能测试的流程 | 了解功能测试是如何进行的，它的进入、退出的标准是什么等 |
| 对产品整个安装包的各层软件的了解 | 理解产品所依赖的各层软件(如OS,DB,Web Server,ete)及一些辅助工具  软件的了解，也就是该产品的安装包的整体结构，包括从底至上，各级软件的 版本要求、硬件需求和依赖关系等 |
| 学习自动化测试工具 | 对进行自动化的方法和意义有初步认识，能够进行简单的自动化实现 |

**1.** **对于产品的学习**

按照这个学习计划，小艾开始了对产品知识的学习。贺晨告诉小艾，要站在客户的角 度去学习产品、看待问题：

Act as Customer! 功能测试人员，是代表用户来进行测试、把关，而不是帮助开发 人员去完成对功能的检查，从某种意义上讲，测试人员要站在开发人员的对立面，去和 开发人员进行“针锋相对”的讨论，甚至“争吵”。你现在就要把自己当成一个客户去 学习我们的产品。

这就是贺晨常和小艾唠叨的。为了能真正做到这一点，对产品的学习就成了这个部门

从菜鸟到测试架构师—一个测试工程师的成长日记

对于新人培训的第一步，这也是功能测试最基础、最关键的一步。

为了打下扎实的基础，贺晨在第一周只要求小艾通读产品的用户文档，全面了解该产 品是用来做什么的，具体有哪些功能模块，每个模块的具体功能有哪些等。虽然来功能测试 项目组之前，小艾已经对这个产品有所了解，但他还是坚持着读完产品的用户文档。一周下 来，小艾感觉受益良多，虽然还是没有全面了解这个产品，但是基本上了解了用户为什么使 用这个产品，以及一些具体的商业流程的实现，并且小艾逐渐转换了思路，能够以一个终端 用户的角度去看待一些问题。用贺晨的话来讲，小艾已经具备了功能测试人员的灵魂。接下 来的时间，小艾又尝试着了解产品的实现架构和部署方式，对这个产品有了进一步的了解。 小艾已经初步了解了这个产品的外貌，为下面进一步了解局部的功能奠定了很好的基础。

学习的第一步让小艾明白要想在黑暗中抓到虫子，必须先学会转换角度，明白如何从 客户角度去学习和认识产品。但是要想精准地抓到虫子，对产品具体功能特征的学习是小 艾下一步的主要任务，为此，小艾接受的第二个任务是去学习要参与测试部分的需求说明 书、设计文档等相关的文档。

通过学习这些文档，可以了解到为什么用户需要这些功能，这些功能是怎么设计实 现的，具体到每个功能点是个什么样子，都要有一个明确而详细的理解。

在这个任务之前，小艾简单地认为在使用这个产品时没有产生错误就是正确的。在贺 晨的指导之下，认真学习需求说明书、设计文档等相关文档之后，小艾理解了，对于功能 测试而言：

首先功能测试人员不是简单地按照开发人员的设计文档去撰写测试相关文档，测试 人员对于设计文档的准确性同样负有责任。测试人员必须认真学习需求说明书，然后审 核设计文档，确保设计文档中包含了所有的需求说明，并且设计是能够满足需求的，也 就是每个功能都要严格地按照需求说明书实现。如果功能实现之后与设计文档不一致， 即使产品运行的时候没有错误产生，这个功能的实现仍然是错误的。同时，要站在用户 的角度去理解功能设计是否合理，即使简单到一个按钮的位置是否合理也要顾及到。

在学习的过程中，小艾通过对需求说明书的学习，了解了具体功能特征与用户需求之 间的对应关系；通过对设计文档的学习，特别是对每一个用例 (Use Case) 的仔细研读， 小艾了解了这个功能特征的功能点及实现方法，具体到实现过程中是在页面体现，还是具 体标准功能接口的实现(如 SOA), 还是数据库具体数据表格数据的变化，等等。

第 5 章 黑色盒子：打着手电抓虫子

**2.** **熟悉各种测试文档**

完成了对于产品的这些文档的学习后，由浅入深，由全貌到局部，小艾对这个产品在 功能上有了一定程度的了解。此时的小艾已经跃跃欲试，准备大干一番了。贺晨这时也给 小艾真正的机会开始学习功能测试。贺晨让小艾学习前面学过的具体功能特征所对应的测 试文档：一系列相关的测试计划、测试场景和测试用例。

贺晨让小艾在阅读测试文档之前，再看看设计文档，想想如果是他自己测试这个功能， 会怎么去测试?然后再对比测试文档的内容看看自己有什么不足之处。通过对比学习，小 艾发现自己测试角度和思维方面与有经验的测试人员是有差距的。首先，对产品功能学习 不够深刻，很多小功能点都考虑不到；其次，对功能的异常情况考虑不够，缺乏产品运行 时错误处理的测试意识；最后，测试场景、测试用例的设计不够细化，不同的测试用例中 包含了太多的测试冗余点。看来功能测试学习远没有想象的那么简单，做精做细不是一件 容易的事，后面的道路还需要付出更多的努力才行。

**3.** **了解功能测试的流程**

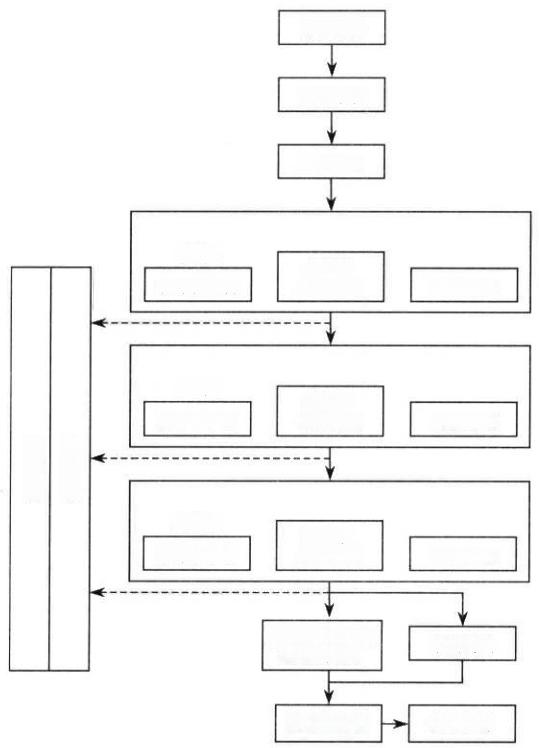
贺晨也指出了以后学习及经验积累的重要性，同时贺晨也告诉小艾，当前要学习的事 情还很多。学习了解现在的测试流程，特别是功能测试流程，就是一件很重要的事情。

如果说前面的学习是从横向上了解这个产品，那么了解这些流程就是从纵向上了解 这个产品的生命周期及具体功能测试是如何进行的。

测试流程是整个软件开发生命周期的一部分。作为测试的一部分，功能测试在不同的 开发模式下，不同的公司、不同的产品，其流程也不尽相同。如瀑布开发模式 (Waterfall Model) 和敏捷开发模式 (Agile Model) 下，功能测试流程的定义就可能会有比较大的区 别。本书会结合瀑布开发模式(Waterfall Model) 和敏捷开发模式 (Agile Model), 来介绍 功能测试的流程是什么样的。图5-1是一种比较典型的敏捷开发模式流程图，图5-2是一 种比较典型的瀑布开发模式流程图。

流程是做好事情的保障，无论是什么开发模式，都要了解流程，进而做好工作的规划。 以敏捷开发模式为例：小艾认为敏捷开发模式(Agile Model) 就是不断地重复迭代的一种 开发方式，与以往传统的瀑布开发模式相比，功能测试的开始时间提前了。在敏捷开发模 式 (Agile Model) 的每一次迭代结束时，都需要通过相应的测试，迭代提交的产品是可工 作的软件，而不是等到所有的迭代过程都结束了才开始进行全面的功能测试。

从菜鸟到测试架构师--一个测试工程师的成长日记



需求分析

发布标准

架构设计

迭代1

编码和 单元测试

迭代2

编码和 单元测试

迭代3

编码和 单元测试

迭代1,2,3的功 能测试通过

发布和实施

安装测试

性能测试

设计和审查

设计和审查

设计和审查

技术支持

功能测试

功能测试

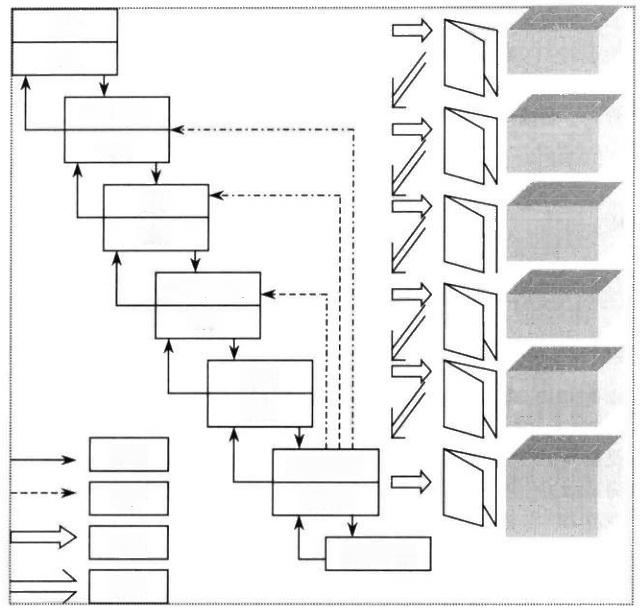
功能测试

系统测试

图5-1 敏捷开发模式流程图

贺晨帮助小艾进一步理解了这个产品的开发方式。在这个产品的开发模式下，产品解 决方案是由不同的用户故事组成的，不同的用户故事的优先级不同。处于某一迭代过程中 的高优先级的故事的需求是明确的、详细的，低优先级的故事在前面的迭代过程中，需求 往往可能是概略的、不确定的。即使是处于高优先级的故事，它的设计也可以是由简单到 完善的一个跨若干个迭代的过程。所以最初的测试场景和测试用例的设计可能只是针对产 品某一功能特征的一个设计雏形来进行的，需要在每次迭代的过程中不断地完善，直到最 终产生一个可交付的测试场景和测试用例，以及最终的测试计划书。

第5章 黑色盒子：打着手电抓虫子



计划

确认

需求分析

确认

设计

确认

实现(编码)

确认

测试

确认

开发

运行/维护

维护

维护计划

输出

输入

测试计划 测试报告

软件问题 变化报告

软件需求 规格说明

初步设计 详细设计

废弃/退役

软件规划

代码清单

再确认

图5-2 瀑布开发模式流程图

**4.** **对产品整个安装包的各层软件的了解**

“工欲善其事，必先利其器”,贺晨告诉小艾，要想做好功能测试，对所测产品整个安 装包的各层软件的了解是功能测试人员及其他测试人员必不可缺的基本技能。

在贺晨给小艾布置了对产品的整个安装包的各层软件的学习之后，小艾才真正认识到 软件的复杂性。一个大型的应用，它所涉及的各层软件种类繁多，通常是一个复杂的矩阵 结构，这和不同的用户对软件及整个IT 基础架构的不同需求和设计有关。表5-2是一个 简单的操作系统和数据库的二维矩阵表。

对这些不同层次的软件的学习不是一件很容易的事情，既要学习不同的操作系统、数 据库、Web Server 等的一些基本功能，又要学习异常的分析与解决办法。虽然困难重重，

从菜鸟到测试架构师———个测试工程师的成长日记

但小艾凭借自己的努力，还是在最短的时间内，掌握了最基本的一些操作。贺晨对小艾说， 万里之行，始于足下，虽然不用精通每种软件，但是至少要掌握常用的一些基本软件的用 法。除了产品的安装包涉及的软件，一些常用的工具软件也要了解，比如，版本控制软件、 状态跟踪软件、一些编辑工具等。

**表5-2操作系统和数据库对应表**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Windows | AIX | … |
| DB2 |  |  |  |
| Oracle | \* |  |  |
| … |  |  |  |

**5.** **学习自动化测试工具**

掌握了上面的这些基本知识和技能，就可以进行一些简单的手工测试了。但是，贺晨 对小艾说，要想做好功能测试，一个不可分割的部分就是自动化测试。对于功能测试而言， 自动化测试是提高工作效率、保证测试质量及减少累积的回归测试工作量的重要保证。所 以，自动化测试是功能测试人员的另一基本技能，随着对功能测试越来越重视，自动化测 试已经成为业界的一个重要考量指标。

自动化测试是功能测试中一项重要且具有挑战性的任务。自动化测试在整个功能测 试中占据的比例越高，说明功能测试做得越好。对于新人而言，必须要在自动化的开发 和测试方面多下工夫，这是成为一个测试专家的很重要的条件。自动化测试是师傅领你 进这个门，但最后能不能做好就要看你个人的修行了。

贺晨告诉小艾，对自动化测试的学习，要从以下几个方面入手： 一是要理解功能测试 用例自动化所依附的自动化开发框架，二是要学会自动化功能测试用例的自动化工具，三 是要依据一定的规范开发功能测试用例的自动化脚本。关于自动化方面的深入探讨，请参 见本书第11章“一遍又一遍：自动化的好处多”。

经过一系列学习之后，小艾发现功能测试并不如想象般容易，不是任何人都能做好、 做精的，它是需要真正用“心”去做的事情。贺晨语：

细心，耐心是一个优秀的功能测试人员必须具备的素质。技术创新和管理创新是功 能测试面临的重要挑战和机遇。

经过上述的学习，小艾似乎已经有了自己的手电，但是小艾还不能很好地驾驭这个手电。

第5章 黑色盒子：打着手电抓虫子

主要是由于对小艾这样初次接触功能测试的人来说，欠缺的不仅仅是贺晨给制订的学习计划 中的知识，另外一个重要的方面就是客户意识的欠缺，也就是说，还做不到“Act as Customer”。

在贺晨给安排的测试任务中，小艾更多的是机械地按部就班，只关心最后的结果，对 中间的过程基本上不大注意。举一个例子，贺晨问他测试某一功能时的中间过程，或者说 是中间状态是什么样的，小艾答不上来，他只记得功能完全按照测试案例完成了，最后的 结果是正确的。有了类似的经验，小艾在之后的功能测试中慢慢地学会了仔细地观察每一 个测试用例的中间过程/中间状态，果然发现有些测试用例中间的有些状态是错误的或者 状态虽然是正确的，但产生了一些异常，这些都会对整个应用带来潜在的或重或轻的问题。 因此，在功能测试中，最终结果固然很重要，中间的过程也不容忽视，否则很可能会在最 终的客户环境中出现一些意想不到的问题。

在有了产品知识、软件技能和客户意识之后，小艾终于可以打亮自己的手电，全副武 装上阵，去寻找产品中的虫子了 ……

**5.1.2** **导师对功能测试的深入浅出的解读**

经过了一段快速而紧张的学习之后，小艾已经对功能测试概况有了一定的了解。用贺 晨的话来说：

小艾同学的水平已经到了似是而非的境界，需要一些理论上的突破，从而达到内功 小成的地步。但即使理论上有突破，离华山论剑的层次也还差得远呢。天还很黑，依然 需要了解更多的概念、流程、策略及实战让手电变得更亮!

为此，贺晨不厌其烦地对功能测试进行了详细的解释。

“老三篇”之黑盒测试与白盒测试：虽然小艾已经知道黑白盒测试的一些区别，但教 条的贺晨还是又讲了一遍。维基百科是这样定义的：

黑盒测试，软件测试的主要方法之一，也可以称为功能测试、数据驱动测试或基于规格 说明的测试。测试者不了解程序的内部情况，只知道程序的输入、输出和系统的功能，这是 从用户的角度针对软件界面、功能及外部结构进行测试，而不考虑程序内部逻辑结构。



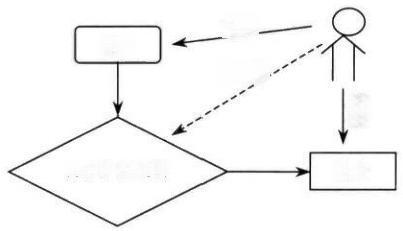
1 <http://zh.wikipedia.org/wiki/%E7%BB%B4%E5%9F%BA%E7%99%BE%E7%A7%91> 维基百科

2012-03-29

从菜鸟到测试架构师—一个测试工程师的成长日记

白盒测试，软件测试的主要方法之一，也称结构测试、逻辑驱动测试或基于程序本 身的测试。测试者了解待测试程序的内部结构、算法等信息，这是从程序设计者的角度 对程序进行的测试。

图5-3显示了这两种最基本的软件测试方法之间的差异。在黑盒测试中，对测试人员 的基本要求是他要知道软件的外在行为，但是不必清楚软件内部是如何运作的，也就是不 用弄清楚黑盒子里面的东西。对应的白盒测试的区别就是测试人员知道白盒子的内部逻 辑，知道该测什么,怎么测，想要什么结果， 一切尽在掌握。

清楚

输入

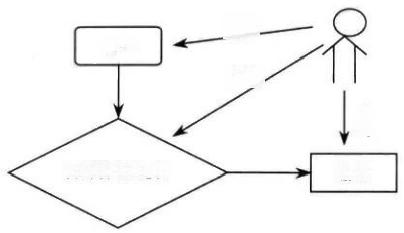
不清楚

清楚

内部算法/逻辑

输出

黑盒测试

清楚

输入

清 楚

清楚

内部算法/逻辑

输出

白盒测试

图5-3 黑盒测试和白盒测试的区别

所以，贺晨对小艾说，我们的这个功能测试部门实际上就是做黑盒测试的。关于功能 测 试 ，IBM 把 它 定 义 为 ：

(1)基于软件功能模块的技术设计 - 需求 - 规格说明文档，来验证软件功能模块满足 用户对其功能的要求。

第5章 黑色盒子：打着手电抓虫子

(2)功能测试的目标是测试软件产品的单个功能正确地实现规定的需求和设计文 档。 功能测试同时也要发现各个功能模块的缺陷(也就是缺陷),包括新增代码的错 误和修改老代码产生的错误。

一个好的功能测试人员要成长为测试专家，甚至测试架构师，除了要从执行层面找到 自己的手电，能够按照测试计划书和测试用例，去找到藏在黑盒子里面的虫子，他还要能 够制定或者理解：

(1)功能测试的策略，这是整个功能测试如何进行的最基本准则。包括诸如功能测试 的范围、从哪些角度来测试、回归测试的比例、如何进行自动化开发等；

(2)功能测试在整个软件开发流程中的生命周期，以及功能测试自己的流程，特别是 在当前敏捷开发模式下，如何做好功能测试与开发及其他测试类型之间的协调；

(3)功能测试的测试计划书、功能测试的场景和对应场景的具体的功能测试用例。

**1.** **策略**

测试策略通常是描述测试工程的总体方法和目标，用于规定测试的范围、种类、每个 测试活动该如何制订计划，以及具体测试活动该如何进行等。测试策略是从测试目标、计 划、实施、分析全方位给测试人员以方向性指导，用于保证项目的进度和质量符合市场和 客户对产品和服务的要求。

功能测试策略作为整个测试策略的一个重要组成部分，其目的是给功能测试人员提供 一套策略作为方向指导。一般功能测试策略应该包括但不限于以下方面。

1)功能测试的范围，比如具体功能测试场景/测试用例该覆盖哪些范围、哪些功能点。

具体而言，功能测试策略描述并规定功能测试应该在产品测试中覆盖哪些测试范围， 包括：如何根据需求规格说明书和设计文档中挑选出来的用例 (User Case) 创建功能测试 场景(Test Scenario),如何根据功能测试场景结合测试数据创建功能测试用例(Test Case)。 根据这些定义的规定，测试人员就可以去创建功能测试用例，对产品的功能进行行之有效 的测试。贺晨特别强调了功能测试场景和功能测试用例的重要性，这一点小艾会在之后的 学习过程中进一步领悟。

2)功能测试与其他测试种类的边界及前后承接的条件。

在软件功能测试中，对于功能测试和其他测试类型的边界关系，测试中的承接关系一

从菜鸟到测试架构师———个测试工程师的成长日记

定要描述清楚，这样既可以避免测试漏洞，又可以避免资源的浪费。以黑盒测试和白盒测 试为例，也就是功能测试和单元测试，它们是紧密相连的两个测试类型，黑白分明，但也 存在着彼此之间的边界不清问题，如同黑夜和白昼一样，并不是那么容易区分。在实际的 软件工程项目中，常会有开发人员和测试人员对彼此职责不清楚而造成的测试漏洞。

3)功能测试中不同的测试用例的定义与选取。

在功能测试中定义功能测试用例时要有不同的策略考虑，哪些是作为可以进行某种测 试开始的可接受性测试的测试用例 (Acceptance Test Cases), 哪些是作为当前版本及以后 版本的回归测试用例 (Regression Test Cases),在功能测试中一般都要描述清楚是如何选 取的，以及选取的比例如何。以回归测试为例，回归测试用例数量一般占总的测试用例数 量的5%～100%。但在实际项目的具体实施中，可以根据不同情况，进行进一步的策略上 的划分。比如可以将所有回归用例分为两部分。其中一部分是更基本的回归测试用例，它 的测试范围更窄，主要用来验证在系统经过一个小的改动后其某一部分功能没有问题，它 是回归测试的一个子集，具体比例可视情况而定，如只占总的用例数量的5%～15%,第 二部分可以称为完全回归用例，这个类型的数量大约占总的用例数量的30%左右。

注：以上的这些数据是基于IBM 的最佳实践。

4)针对于某个产品，或者某个产品的某个版本中的功能模块，或者具体功能解决方 案的功能测试策略的描述。

对于一般的Java EE 应用，产品的不同功能模块所涉及的技术、实现的原理，是有所 不同的，那么在制定一个比较大的产品的功能测试策略时，要对不同的功能模块，制定具 体的功能测试策略。比如对于一个Java EE应用，它可能有移动应用相关的应用模块，也 可能有呼叫中心相关的应用模块，对于这两个不同的功能模块，它的测试策略是有很大差 别的。类似的细节要在功能测试策略中多考虑，比如针对产品界面的测试，针对 API 的测 试，或者针对基于服务的框架 (SOA) 的服务的测试等。

5)对于功能测试中涉及的诸如访问控制、异常条件、全球化等的策略。

功能测试策略中要对上述的这些都有所考虑。像边界条件(比如输入框允许的最大、 最小值)、无效的输入、错误场景测试(用来验证产品处理异常输入等情况)、浏览器支持 (IE,Firefox 等)、全球化、可访问测试等。

6)功能测试中的自动化测试相关的策略。

它提供了测试用例自动化开发的相关信息，包括自动化测试用例的选取策略、自动化

第5章 黑色盒子：打着手电抓虫子

开发工具的选取、自动化测试的方案、自动化的目标等。自动化测试对功能测试来说是相 当重要的一个部分，它能够帮助测试人员在整个开发过程中不断地重复执行测试用例，通 过自动化测试，可以在很大程度上减少测试人员花在回归测试上的工作量，这样测试人员 可以将更多的精力放在新模块、新功能的测试上。关于自动化测试，请参见本书的第11 章“一遍又一遍：自动化的好处多”。

7)对于功能测试在不同的环境下进行的策略描述，比如移植环境下的功能测试，客 户化环境下的功能测试等。

通常的功能测试都是在一个“干净”的可运行的环境下进行的，但实际上对于客户而 言，它们的生产、测试环境都复杂得多。为了最大限度地在软件交付给客户之前发现问题， 使得客户得到一个保障质量的软件，功能测试要在不同的环境下去实施、验证。这里也涉 及为提高测试效率而在不同环境下按比例分拆功能测试用例的比例。

8)对于所测试的软件的支撑环境的选择和测试用例的比例分配。

几乎所有的企业级Java EE 应用软件都支持不止一种操作系统、数据库、Web Server、 浏览器等，如果把每种组合都测试到，需要巨大的花销。因此，在功能测试策略中，要对 软件的上述支撑环境进行合理而有效的选择，并考虑到测试成本和测试效率及风险，在不 同的软件平台上分拆功能测试用例。

9)对于软件产品的不同版本的测试。

对于Java EE产品，通常都有不同的版本，如企业版、专业版、个人版等。功能测试 策略中一定要有针对不同的版本的测试策略，在定义功能测试用例时， 一定要分清是针对 哪个版本的功能测试。

听了贺晨的“夸夸其谈”之后，小艾有拨云见日的感觉，功能测试要做好可真是不容 易啊!小艾想，什么时候我才能纵论各种测试策略，像武林高手一样涅磐，去华山论剑呢?

目前，还是踏踏实实地按流程做事情，学好执行和如何去设计测试计划、测试场景、 测试用例吧。

**2.** **流程**

贺晨告诉小艾，流程可是 IBM 人安身立命之所在。一个熟悉各种流程的人，才能够 按照正确的方式做事情。特别是在IBM 这样的一个以提供企业级应用为己任的大公司里， 按照流程做事情才能够保证一个大的团队动作的整齐划一，最终提供给客户高质量的软硬

件产品。在 IBM, 几乎所有的项目都需要 IBM 的位于全世界不同国家，不同城市的不同 的团队协调工作，共同完成的。

功能测试的流程主要用来规范功能测试是如何进行的，它的进入、退出的标准是什么, 等等。贺晨告诉小艾，功能测试的流程一般会涉及以下几个方面。

1)功能测试流程的输入元素

一般而言，对于功能测试，输入的元素主要包含需求说明书、设计文档、代码、产品 的构建版本 (build), 以及产品的文档(如在线文档)等，有了这些输入元素，功能测试 才能够开始最初的计划阶段。

2)功能测试流程的输出元素

功能测试的输出元素主要包括：功能测试计划书、功能测试场景、功能测试用例、功 能测试总结、功能测试执行记录、功能测试缺陷的记录和分析报告、功能测试交付的产品等。

3)功能测试计划书的准备

功能测试计划书主要用来描述和规定以下一些内容元素：

2 不同功能测试阶段的测试目标；

达成这些测试目标的整体策略；

鉴别功能测试的依赖关系和风险；

明确各种资源的可用性，比如人力资源、机器资源等；

建立计划、执行和报告的过程和约定；

建立功能测试的重要里程碑(如具体日期、时间表等),以及功能测试完成和每 个功能测试阶段成功的进入和退出标准。

功能测试计划书分不同的层次，比如针对整个产品的、针对具体模块的、针对具体的 功能特征的，它们在不同层面来定义以上的全部或者部分内容。

4)功能测试用例的设计

功能测试用例是具体功能测试最细化的信息。一个测试用例是一系列相关的测试条件 的逻辑化的拼装。功能测试用例应当考虑有效的输入/输出，以及无效的、不期望的输入/ 输出。功能测试用例需要按照一定的规范来存储，比如功能测试用例的编号、测试的目的、

第5章 黑色盒子：打着手电抓虫子

测试建立的需求、运行测试用例的说明、测试数据及期望结果等。

测试用例的信息应当包含进测试计划书中，具体功能特征的功能测试计划书应当包含 测试用例的具体信息。

5)功能测试计划书的审阅和批准

在完成功能测试计划书的准备和功能测试用例的设计之后，需要不同角色的人(包括 开发人员、架构师、测试组长等)进行审阅，并最终得到批准。得到批准的功能测试计划 书是功能测试进行的唯一指导，必须严格遵循。

一般而言，审阅的流程包括：

计划

保证时间表和参与者，并且需要审阅的资料可用。

 准备

保证参与者有足够的时间进行审阅。

审阅

审阅可以通过正式的会议或者通过在线的方式来进行，审阅的目的是发现问题和疏忽 之 处 。

修改

解决审阅过程中发现的问题。

追踪

保证所有的问题都被圆满地解决。

批准

在最后的阶段，当所有的审阅意见都处理完后，要拿到具体负责人处批准。

6)功能测试的执行

功能测试的执行是验证产品的所用功能特征都符合产品设计说明书的要求。执行的目 标是测试到每一个功能特征和不同功能特征之间的交互，包括测试的输入、输出和功能流 程本身的表现。

从菜鸟到测试架构师——一个测试工程师的成长日记

功能执行的环境必须和产品的建立及用户所使用的环境一致，在这样的环境下执行并 验证功能测试用例。对于发现的任何缺陷，都要用相应的缺陷管理工具来记录并跟踪，直 到缺陷被解决。原则上，任何问题都要在产品发布之前被解决掉，除非产品发布管理团队 批准某一缺陷可以不在当前版本解决。

在功能测试开始之前， 一般要进行功能测试开始的可接受性测试用例(Acceptance Test Cases) 的验证，如果这个验证通过，证明最基本的功能已经实现，可以开始功能测试。

无论是什么产品的功能测试， 一般都分为几个不同的阶段，用来包含不同的测试涵盖， 如对于新功能的测试、回归测试、对于整个产品的完备的回归测试等。

在功能测试的执行过程中，要在相应的工具里面做好执行的记录。

7)功能测试状态的报告

功能测试部门(组)要把功能测试流程的状态从计划初始阶段就要定期地、及时地和 版本发布管理部门/版本发布经理沟通。状态报告必须是基于客观的事实，要对重要的里 程碑、产品的质量等做好评估。

对于发现的任何缺陷，都要做好文档记录、跟踪和报告，直到这个缺陷被解决，进而 关掉。

在了解了功能测试流程的主要元素之后，小艾还是不太清楚功能测试要在什么时候开 始。贺晨给小艾举了两个例子， 一个是和瀑布开发模型 (Waterfall Model) 相关的， 一个 是和敏捷开发模型 (Agile Model) 相关的。

在传统的瀑布开发模型 (Waterfall Model) 中(见图5-2),功能测试是在所有开发都 交付的情况下才开始的， 一般分以下几个阶段：测试新功能为主的阶段、回归测试为主的 阶段、成品测试阶段。对于新功能测试、回归测试，本章都会有大篇幅涉及。对于成品测 试，请参见本书的第9章。

相对于传统的瀑布开发模型(Waterfall Model),目前比较流行的敏捷开发模型(Agile Model) 中功能测试开始的时间点是不太一样的，具体的流程也不尽相同。敏捷开发模型 中，开发和测试的任务是在每一个迭代周期中完成的，关于这部分更多的内容，可以参见 本书的第12章。无论是哪一种开发模式，在不同的阶段，功能测试人员都需要经历以下 的过程。

在需求讨论阶段，测试人员以自己的理解，站在客户的角度上来阐述自己的观点，

第5章 黑色盒子：打着手电抓虫子

和项目组的不同角色的人员，如产品经理、开发工程师、文档开发工程师等，进行充分 的交流和讨论，在学习理解产品功能的同时，把自己作为“客户”的感受，反馈给开发 设计人员。

在设计开发阶段，功能测试人员扮演着开发人员和“客户”的双重角色。 一方面根据 需求规格说明书和设计文档制定测试计划书、功能测试场景和功能测试用例，并根据这些 测试文档进行测试，发现问题(缺陷);另一方面站在客户的角度与设计开发人员进行有 效的沟通，及时提供软件质量反馈，包括代码质量、接口一致性、用户体验等。

在成品交付阶段，功能测试人员要对产品进行充分的回归测试，以确定没有问题遗留 或者回归问题的出现。同时还要保证文档相关测试的进行，保证客户使用产品的时候有准 确而清晰的文档进行参照和指导。

在每个阶段，无论是大的需求、开发、交付阶段，还是不同的小的迭代阶段，每个阶 段都要有明确的进入和退出标准，这要在功能测试流程中定义清楚，在功能测试中要严格 按照定义的流程去做事情，这样才能做到整个团队都步履一致，整齐划一。对于大型的Java EE 的产品开发而言，功能测试团队必须严格遵循制定的流程，以功能测试策略为指导， 才能够保障功能测试的任务能够高质量地完成，交付给市场和客户一个高质量的软件产 品。

除了贯穿软件生命周期的这些以时间为轴的功能测试流程之外，为了保证功能测试的 正常进行，通常还会有一些辅助的流程帮助功能测试人员更好地工作。比如：测试计划书 审核的流程、跨不同功能测试组的交互验证的功能测试流程、随机测试的流程、自动化脚 本审核的流程、功能测试人员对单元测试的审核和签注流程等。这些流程一方面保证了功 能测试的质量，另一方面保证了团队协作的效率。测试流程不是越多越好，要根据项目实 际情况取舍。这需要测试组长或者架构师有比较丰富的经验和先见之明。

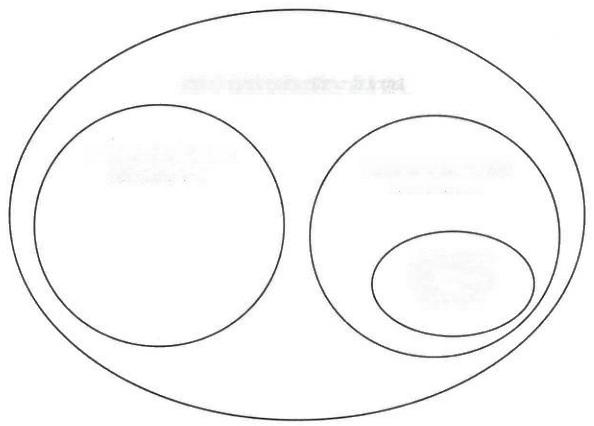
小艾听了贺晨对功能测试流程的解读之后，多少明白了功能测试的工作是怎么展开 的。但是具体对于一个功能测试的新人来说，理解了具体的流程之后，该怎么做呢?具体 做些什么呢?小艾对此还没有直观的印象。为此，贺晨决定系统地给小艾讲解他该做什么, 以及怎么做。

**3.** **文档**

贺晨对小艾说，功能测试人员除了之前学到的微观的基本技能和宏观的策略和流程之 外，最重要的就是要理解和掌握各种各样的文档。能够很好地理解别人写的文档，是对功

能测试人员最基本的要求，只有这样，才能够把之前掌握的各项技能熟练运用于实际的操 作过程，进而达到执行高手的境界。能够写出好的文档，是资深功能测试人员，特别是测 试组长及测试架构师的基本素质，最基本的文档包括如何写功能测试计划书、功能测试场 景、功能测试用例等。

功能测试计划书涵盖了整个产品的功能测试范围及各种相关条件。对于一个大型的 Java EE产品开发而言，功能测试计划书又分为不同层面，比如具体功能特征的功能测试 计划书、具体功能模块的功能测试计划书、整个产品版本的功能测试计划书。它们的关系 如图5-4所示。

整个产品版本的功能测试计划书

具体功能模块的功能 测试计划书2

具体功能模块的功能 测试计划书1

具体功能特

征的功能测

试计划书

图5-4 功能测试计划书的关系

在敏捷开发模式下，具体功能特征的功能测试计划书是一个按照迭代周期迭代开发的 测试计划书。在这个功能测试计划书中，包含对该功能特征的具体的测试范围，由测试场 景和测试用例来涵盖各个功能点和一些异常和边界条件。而不同的功能测试计划书都会包 含一些和所涵盖的功能测试范围相关联的条件，比如该功能测试的前提条件、依赖关系、 相关文档和工具的连接等。

测试场景是功能测试中一个非常重要的概念，只有设计好的功能测试场景，才能够真 正代表客户，站在客户的立场上进行测试。通常，在测试人员与开发人员关于产品的交付 功能达成一致后，测试人员需要跟开发人员一起工作，挑选出所有需要的用例(Use Case),

第5章 黑色盒子：打着手电抓虫子

组织测试场景来描述用户的行为。一般同时进行的还有，功能测试人员与文档开发人员一 起工作以确保所有的功能都在设计文档中被正确定义，从而保障测试场景及后续测试用例 的准确性和可执行性。每一个功能测试场景是由一个或者多个用例组成的。

Scenario functional(S+)={UCs₁,UCs₂,…,UCsn}

下面是一个电子商务网站相关产品的功能场景的例子。

示例1: 注册一个用户并在商店里浏览商品、购买并退出登录。

UC₅1 = 注册用户

UCs₂= 登录商店

UCs₃= 浏览商品

UC₅4 = 添加到购物车，完成购买

UC₅s = 从商店退出

这里的功能测试场景SF={UCs1,UCs2,UCs3,UCs4,UCs5}。

因为功能测试场景是功能测试用例的逻辑，要完成功能测试用例，功能测试人员需要 创建数据和结果验证。不同的功能测试用例可以是同一个功能测试场景，但是是不同的数

据和结果验证。因此，功能测试用例可以定义为Test Case=(Sf,D,RV),其 中 S= 功能测试场景

功能测试场景是一个由设计文档中的一组用例按照一定的逻辑组织起来的场景。

D= 数据

数据是输入信息或者场景中需要的材料。

RV= 结果验证

结果验证是一系列步骤，用来验证在功能测试场景中列出的每一步具有期望的结果。 因此，在功能测试场景中列出的每一步，都需要有一对一的数据和结果验证的对应。

好的测试设计需要对测试的产品和目标用户的商业需求有深入的理解，同时要对产品 的功能非常熟悉。好的测试场景和测试用例的设计对于测试的质量和测试执行的效率是非 常有好处的。一个好的测试用例将清楚地描述什么会被测试，将产生什么样的测试结果， 以及为什么产生这样的结果。

从菜鸟到测试架构师——个测试工程师的成长日记

贺晨告诉小艾，完整的测试用例至少要包含以下几个方面。

输入：输入可能是来自键盘、鼠标、图形用户界面、数据库或者电子表格读取的 信 息 ；

期望的结果：测试用例是可预见结果的，需要在测试用例中准确提供期望结果， 否则不是完整的测试用例；

系统运行的前提条件或者说明：测试用例正确执行需要哪些系统条件，这些系统 条件可能是一些需要运行的应用程序或者需要提供一些必需的文件，也可能是哪 些用例需要在某些用例之前或之后执行等。

设计功能测试场景和功能测试用例的任务与设计软件本身一样有难度，大多数功能测 试人员感觉到他们没有足够的知识、经验和时间来完成这些设计任务。 一方面是行业知识 缺乏，不了解客户的使用习惯；另一方面对于产品的功能缺乏足够的了解，一个功能测试 人员对于自己所测的功能模块之外的内容了解甚少；还有在敏捷开发模式下，功能测试人 员往往是被“项目驱动”的，总是在赶工，缺乏充裕的时间去理解消化。

在导师的指导下，小艾尝试设计了第一个功能测试用例，如表5-3所示。

表5-3商店登录功能测试用例

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **测试类型** | | 功能测试 | | **模块名称** | 电子商务商店用户登录 | |
| **用例描述** | | 该用例用来测试在登录界面用户能否正常登录；若出现错误的信息，系统将给出正确提示信息 | | | | |
| **前置条件** | | 存在正确的用户名和密码：登录页面正常装载；(用户名/密码为TestUser/Test1User) | | | | |
| **编号** | **测试项** | | **操作步骤** | 数据 | | 期望结果 |
| 1 | 界面检查 | | 1)界面装载  2)检视界面的标题  3)检查默认焦点  4)检查TAB键控制 | 1 ) 商 店 界 面 的 U R L :  [http://chostname>/index.html;](http://chostname>/index.html;2)  [2](http://chostname>/index.html;2))<N/A>;  3)N/A;  4)N/A; | | 1)界面正常装载  2)界面标题显示为“Homepage”  3)默认焦点为用户输入框  4)能够通过TAB键控制 |
| 2 | 用户登录 | | 1)输入正确的用户名  2)输入正确的密码  3)按回车键 | 1)用户名：TestUser  2)密码：Test1User  3)N/A: | | 1)N/A  2)N/A  3)正常登录，转入对应的系统页面 |

第 5 章 黑色盒子：打着手电抓虫子

**续表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **编号** | **测试项** | **操作步骤** | **数据** | **期望结果** |
| 2 | 用户登录 | 1)输入未区分大小写 的用户名  2)输入未区分大小写 的密码  3)按回车键 | 1)用户名：testuser  2)密码：testluser  3)N/A | 1)N/A  2)N/A  3)区分大小写，显示出错信息“用户名或 密码错误”并清空输入框，不能正常登录 |
| 1)输入正确的用户名  2)输入错误的密码  3)按回车键 | 1)用户名：TestUser  2)密码：Test2User  3)N/A | 1)N/A  2)N/A  3)显示出错信息“用户名或密码错误”并 清空输入框，不能正常登录 |
| 1)用户名：TestUser  2)密码：¥§(在用户名 中输入特殊符号)  3)N/A | 1)N/A  2)N/A  3)提示用户名或密码中存在特殊符号， 并清空输入框，不能正常登录 |
| 1)输入错误的用户名  2)输入正确的密码  3)按回车键 | 1)用户名：TestUser2  2)密码：Test1User  3)N/A | 1)N/A  2)N/A  3)显示出错信息“用户名或密码错误”并 清空输入框，不能正常登录 |
| 1)用户名：¥§(在用户 名中输入特殊符号)  2)密码：TestIUser  3)按回车键 | 1)N/A  2)N/A  3)提示用户名或密码中存在特殊符号， 并清空输入框，不能正常登录 |
| 1)输入错误的用户名  2)输入错误的密码  3)按回车键 | 1)用户名：TestUser2 2)密码：Test2User  3)N/A; | 1)N/A  2)N/A  3)用户名或密码错误 |
| 1 ) 直 接 单 击 登 录 (submit) | 1)N/A | 1)出现“请输入用户名、密码”的提示框 |
| 3 | 用户重置 或取消 | 1)输入用户名  2)输入密码  3)单击重置或取消 | 1)用户名TestUser  2)密码Test1User  3)N/A | 1)N/A  2)N/A  3)清空用户名/密码输入框 |

从菜鸟到测试架构师——一个测试工程师的成长日记

看着自己写的第一个功能测试用例，以及自己对于功能测试策略、流程的初步理解， 小艾很有成就感，贺晨也非常满意。

手电已经找到，就看你如何去运用它找到产品中的虫子了。功能测试之道，源于心， 成于行。

关于下一步的工作安排，贺晨要求小艾开始进行实际的项目锻炼，进行实际测试用例 的执行，以及开始编写自动化脚本。

**5.1.3** **参与实际测试执行，编写自动化测试脚本**

在敏捷开发模型中，随着开发人员不断地交付代码，每一个迭代功能测试人员都要对 所交付的功能进行测试用例的设计、执行和自动化脚本的开发，用于后面迭代开发阶段、 系统测试阶段及以后版本的回归测试。

小艾是在迭代开发的某个阶段进入到实际项目中的，贺晨要求小艾从熟悉搭建产品测 试环境入手。小艾所在的部门有一套完整的测试环境构建流程，在贺晨帮助下，小艾完成 了用于不同测试目的的测试环境的搭建工作。贺晨告诉小艾，为了适应不同客户的不同需 求，一个大型的Java EE应用总是对应着各种各样的安装配置，也就是软件安装包的不同。 对于不同安装配置，都要分配相应比例的测试用例，进行有效测试覆盖，这是测试工作中 必须要考虑的测试覆盖面之一。有了前面的理论基础，小艾很快就熟悉了这项工作。

搭建好测试环境后，小艾开始了对功能测试用例的执行工作。产品开发初期的迭代过 程，主要是以手动执行的方式执行功能测试用例，同时对这些功能测试用例进行自动化的 开发，这样在后面的迭代过程及其他过程中，会以自动化的方式进行回归测试。对于整个 功能测试用例的执行过程，会借助于各种工具进行对执行过程的跟踪、记录。如果发现了 缺陷，要对缺陷进行比较详细的分析，粗判缺陷的产生原因，提供尽可能多的错误信息， 以帮助开发人员定位和解决问题。

在迭代开发阶段，在功能测试用例的开发和手工执行过程的同时，功能测试人员最好 同时开发对应功能测试用例的自动化测试脚本。开发自动化脚本的好处除了前面提到的之 外，还能够帮助功能测试人员更好地理解测试用例的逻辑，突出对功能测试用例中的功能 点的理解。自动化开发除了要熟悉相关编程技巧，还要有相应的自动化开发的框架作为支 撑，并有完备的自动化测试策略和流程保证自动化代码的重用性和代码质量。对于自动化 的更多论述，请参见本书的第11章。

第 5 章 黑色盒子：打着手电抓虫子

下面附一个简单的商店登录自动化测试用例的自动化脚本，通过JUnit 来定义、运行 测试用例，功能上主要使用了开源测试工具Selenium 来实现自动化网店页面操作和验证。

关 于 JUnit 和 Selenium 的详细介绍，请参考4 . 2 节 及 <http://www.junit.org/>¹ 和 <http://seleniumhq.org/>²。

**public void** testUserLogOnStore() **throws** Exception(

//根据网店地址打开网店页面

Store.openStore(url);

//等待页面加载完毕

HomePage.waitForPageLoad();

//单击“登录”按钮

Header.clickSignIn();

//验证登录是否成功

boolean status =Store.verifyLogon();

Assert("Failed to log on store front",true,status);

其中主要函数的实现为：

//根据网店地址打开网店页面

public void openStore(String url)

{

//通过Selenium API打开网址

selenium.open(url);

//设置页面加载的过期时间(如果在此时间内没有加载成功，则视为加载失败)

selenium.setTimeout(String.valueof(Config.getTestTimeout()\* ILLISECONDS\_IN\_SECONDS));

}

//单击“登录”按钮登入网店

public void clickSignIn()

{

//验证当前页面是否是页眉 (Header)

Header.verifyIsHeader();

if(!Store.isUserLoggedOn())



1 [http://www.junit.org/Junit 2012-03-21](http://www.junit.org/Junit2012-03-21)

2 [http://seleniumhq.org/Selenium 2012-03-30](http://seleniumhq.org/Selenium2012-03-30)

从菜鸟到测试架构师——一个测试工程师的成长日记

{

test.waitForElement(StoreConstants.HEADER\_SIGN\_IN\_LINK); test.click(StoreConstants.HEADER\_SIGN\_IN\_LINK);

test.waitForPageToLoad();

else

{

Header.clickLogoutButton();

//验证是否登录成功并跳转到登录后的页面 public boolean verifyLogon()

{

//通过检查HTML中的账号元素来判断登录状态

test.waitForElement(StoreConstants.STORE\_MY\_ACCOUNT\_SUMMARY\_TEXT); if(test.isElementPresent(StoreConstants.STORE\_MY\_ACCOUNT\_SUMMARY\_TEX

T))

{

logger.info("Logon to store successful"); return true;

else

logger.info("Logon to store failed");

return false;

**5.1.4** **找到了第一条虫子**

小艾在功能测试之旅中，很快就找到了第一条虫子。小艾按照自己设计的商店登录功 能测试用例描述的测试步骤一步一步地严格对照测试。

首先在浏览器里输入登录商店的网址，出现提示输入登录商店的用户名和密码的 页面 。

小艾按照测试用例要求，输入了测试数据列里提供的用户名和密码。这里小艾输入的 用户名是 TestUser, 密码是Test2User, 然后单击“确定”按钮。

第5章 黑色盒子：打着手电抓虫子

测试用例里提示执行到这一步出现的结果应该是“用户名或密码错误”,并清空输入 框，不能正常登录。但是页面却没有消息提示，只是清空了输入框，不能正常登录。没有 提示信息，与用例的期望结果不一致，虽然看起来不影响功能，但是对于客户而言，提示 信息不够准确、友好，并且和设计文档不相符，各种证据表明这是一个不折不扣的缺陷。 小艾很兴奋，找到第一条虫子的喜悦，让小艾很有成就感，小艾对未来的功能测试工作充 满了期待。

通过对整个商店登录功能测试用例的测试，小艾觉得自己已经发现了所有的问题，把 这部分工作汇总并报告给贺晨，等着开发人员解决所有问题，再重新测这个测试用例。但 是没想到，贺晨指出了一个小艾没有注意到的小问题，登录界面中的“确认”按钮的颜色 偏灰，用户体验的效果很不好，不符合用户的使用习惯。唉，还是没有做到100%的细致， 小艾稍微有点沮丧，看来要做到一个非常专业的功能测试人员，还要积累更多的经验。

**5.1.5** **手记** **—** **—** **执行高手**

小艾通过贺晨的帮助及在实际项目中的锻炼，从执行层面上已经能够驾轻就熟，完成 功能测试的执行任务已经不是什么问题。虽然离一个真正的执行高手还有段距离，但小艾 通过这段时间的实践已经知道如何能成为一个执行高手。小艾记到：一个好的功能测试人 员需要掌握以下技能并加以持续的锤炼，才能成为真正的执行高手。

(1)文档的理解

功能测试最重要的是理解业务需求，熟悉产品，熟悉具体的功能模块、功能特征的设计 文档，熟悉功能测试场景和用例。从上而下、从整体到细节了解自己的工作，做到心中有数。

(2)产品框架和数据结构的了解

要做好功能测试，还需要对整个产品的框架和系统的数据库结构比较清楚，从整体和 细节上了解产品的技术构建。

(3)产品环境(各层支撑软件)的理解和技能的培养

任何一个产品都不是孤立存在的。要做好功能测试，就需要了解支撑产品的各层软件 的功能和操作。

(4)测试工具的理解和技能掌握

只有熟练地掌握测试工具的使用，才能在功能测试中事半功倍。比如对自动化测试工

从菜鸟到测试架构师———个测试工程师的成长日记

具的使用和掌握。

(5)问题分析能力

对功能测试中出现的问题的分析能力，是一个功能测试人员的一个重要衡量指标。好 的功能测试人员总能够提供给开发人员准确的分析和帮助。

贺晨领着小艾走过了对功能测试知识从初步理解到具体执行的过程，完成了从菜鸟到 一个合格的功能测试执行人员的成长。小艾渴望着能够从更高、更全面的角度去了解功能 测试。贺晨适时地引导小艾进入了对基于模块的功能划分和功能测试，以及跨模块/解决 方案功能测试 (Cross Component/Solution Testing) 和集成功能测试的学习中。

**5.2** **如何把黑盒子分块**

贺晨对小艾讲，一个大的Java EE的产品或应用，通常代码量都非常巨大，其业务逻 辑、体系架构都非常复杂。如图5-5'所示为 IBM WebSphere Commerce 的产品框架，可以 能够看到这样的产品从各方面讲都是一个非常复杂的Java EE 应用。

对于这样的产品或应用，如果采用简单的方式去从整体上测试其功能，是没办法做到 的。业界常用的办法是把产品或者应用进行模块化，或者基于解决方案(solution) 的分解， 也就是要把我们看到的大的黑盒子分块。小艾能够理解对黑盒子进行分块的好处，但是不 知道分块的原则。贺晨告诉小艾，没有固定的原则，但是一般从操作层面上，功能测试中 对黑盒子分块主要要考虑的因素有以下几点。

(1)产品或者应用的自然的模块划分。

(2)产品或者应用中功能的相似性：把相似的功能分到一个小盒子中。

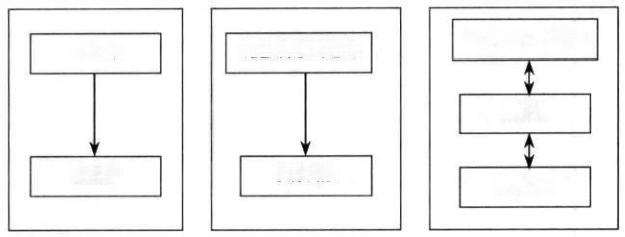
(3)功能测试团队的资源(主要是人员)状况：实际的项目中要考虑资源状况，对黑 盒子进行粒度合适的功能模块分解。

贺晨告诉小艾，这里面合适粒度非常重要，并不是模块分的越多，模块分的越小，就 越好。粒度过大会带来在同一个功能测试计划书里面涵盖的细节太多，功能测试用例的量 级也相应的变大，易读性，可控性都会变差。粒度过小会造成整体上功能测试计划书的量

1《下一代电子商务》.编著:《下一代电子商务》编委会.电子工业出版社，2009.8

第 5 章 黑色盒子：打着手电抓虫子

级太多，人员分配上也比较凌乱，项目的整体可控性会变差，后续的回归测试的复杂度也 会提升。需要在实际项目中慢慢积累经验，进行适度的力度分割。



企业到顺客(B2C) 业务模型 企业到企业(B2B) 直接业务模型企业到企业(B2B 间接业务模型

制造/发行/零售商

制造/发行/零售商

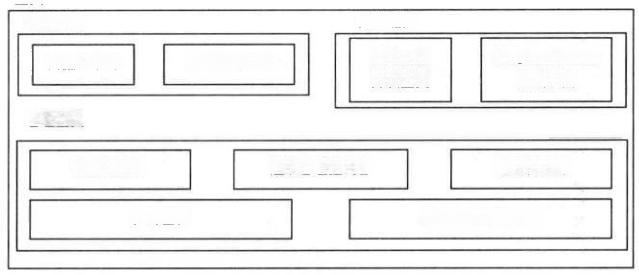
渠道商

买方商家

买方商家

消费者

零售商



工具

商业工具 开发工具

商业加速器 IBM 销售中心 Rational

开发工具

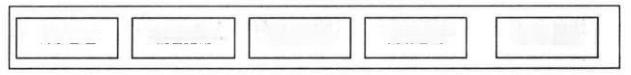
管理工具

管理控制台 组织管理控制台 配置管理器

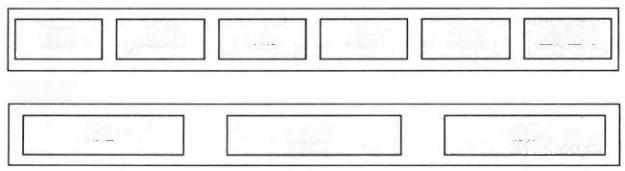
实用工具 工作空间管理工具

WeLephere Commerce 开发工具

功能特性



目录管理 销售指南 营销管理 交易管理 订单管理



商业上下文引擎

授权 全球化

商业能力

综合

商店 工作空间

市场 预览

协作

分析

图5-5 IBM WebSphere Commerce 的产品框架

**5.2.1** **如何精准找寻某一种虫子——分而治之**

小艾对贺晨讲的黑盒子分块知识有了初步的认识，但是如何在小盒子里展开功能测试 的工作呢?为解决小艾的疑惑，贺晨给小艾介绍了如何在小盒子内精准地寻找虫子。

从原理上讲，按照一定的标准把需要测试的产品分成不同的模块后，功能测试团队就 可以对小的模块按照功能测试流程进行完整的测试。把大黑盒子分块后，对每个小盒子分 而治之，由于小盒子模块更小、更灵活，降低了功能测试的复杂度，更容易在小盒子里面 抓到虫子，这将大大降低定位虫子的时间和难度。

把黑盒子分块后，对每个小盒子从功能的角度进行仔细划分，如按照用例的不同把所 有的功能点都列出来，如表5-4所示，其中UC 代表用例(Use Case),FP代表功能点(Function Point)。

表5-4功能点矩阵列表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | FP¹ | FP2 | FP3 | FP4 | FP5 | …… | FPn |
| UC₃I | X |  | X |  | X |  | X |
| UC₃₂ |  | X |  | X | X | X |  |
| …… | X |  | X |  |  | X |  |
| UC₃N | X | X | X | X |  |  | X |

通过这样的详细划分，就可以开发足够而准确的功能测试场景和用例，来对小盒子进 行完备、准确的功能测试覆盖，执行高效的功能测试。在敏捷开发模型中，每个小盒子(功 能模块)的所有功能点并不是同时开发交付的，在每一个迭代过程，要确保该迭代过程交 付的功能点都被功能测试覆盖到，同时确保随着迭代过程的深入，所有的功能点都能够被 有效地通过回归测试所覆盖，保证小盒子交付的质量。

把黑盒子的所有小盒子，也就是所有的功能模块，都进行这样的功能测试覆盖和执行， 似乎这样交付的产品应该是完美的，不会有任何虫子逃掉，跑到最终用户那里去。但事实 是这样么?贺晨给小艾出了这样的问题。

**5.2.2** **客户的反馈——虫子依然存在吗**

贺晨的问题很快就有了答案。在小艾参与的这个基于敏捷开发模型的项目中，中间有

第5章 黑色盒子：打着手电抓虫子

客户参与的对产品的体验与测试，这个过程体现了敏捷开发的优点，在产品开发的每个交 付阶段，都能够有来自客户的及时的需求和反馈，确保产品开发是基于市场和客户的最新 的和实际的需求。

通过客户的试用，还是发现了一些缺陷。通过对这些缺陷的分析发现，其中比较典型 的是小艾他们功能测试团队只考虑了模块测试，但忽略了对业务的端到端的功能测试场景 的设计，这种功能测试往往是跨几个不同的功能测试模块，或者不同的解决方案之间的集 成。贺晨对小艾说，一个完整的功能测试一定是在进行了单元测试之后，对产品进行基于 模块化，或者基于解决方案的测试，对产品进行分而治之，然后进行跨多个不同模块，或 者解决方案的跨模块/解决方案功能测试 (Cross Component/Solution Testing), 或者称为功 能集成测试，以对产品进行“合而治之”的测试，从而达到对小盒子内部及盒子之间的完 全的功能测试覆盖。

**5.3** **对黑盒子的全方位照明**

所谓跨模块/解决方案功能测试 (Cross Component/Solution Testing), 就是为了确保所 有的功能特征，或者用户故事在一个产品或者应用的开发过程中能够被彻底测试，而对产 品的各个模块或者解决方案，按照用户的使用方式，根据模块/解决方案间的逻辑关系， 而设计的跨越多个模块/解决方案的端到端的功能测试场景和功能测试用例，并利用这些 功能测试用例对系统进行全面的功能测试。通过跨模块/解决方案功能测试，能够很好地 解决对于整个黑盒子的自顶向下的功能检查，建立对黑盒子的全方位照明。从对模块的功 能测试的分而治之，进而完成基于集成测试的合而治之。

**5.3.1** **找到了小盒子间的虫子——合而治之**

贺晨对小艾讲，在开发跨模块/解决方案的功能测试的场景时， 一个重要的策略是对 商业逻辑的梳理，明确可能的实际商业场景，通过对实际商业场景的理解，开发出跨不同 模块/解决方案的端到端的功能测试场景；另外一种就是基于模块之间的消息的传递，进 行技术层面的验证，确保技术上的正确性；还有就是基于不同产品之间的集成，做产品集 成的跨产品的集成测试。

跨模块/解决方案功能测试的测试场景应该被定义为：通过一个模块或者解决方案的

不同的输出以及这些输出是怎么作为其他模块或者解决方案的输入(比如一个电子商务产 品来自于数据录入解决方案的产品目录的数据，可以作为产品目录展示界面解决方案的输 入),而定义的能够体现这种整合或者合并的功能测试场景。通过这样的测试场景，能够 确保基于实际的商业流程，以及数据流程的测试完备性。

为了便于理解，贺晨打算给小艾写一个简单的测试用例，来加深小艾对这种跨模块/解 决方案功能测试的理解。

依旧以电子商务网店为例，有这样两个功能模块。模块1主要实现网店产品管理， 负责网站后台的产品录入、产品描述、产品价格显示等。模块2实现购买者在网店通过产 品名称、属性等进行产品的搜索，将符合搜索条件的产品返回给购买者。两个功能模块的 测试人员对两个模块各自都做了非常完备的功能测试，确保各自模块的功能点都被测到 了。模块1的测试人员对产品的各个操作进行了验证，数据库里正确反映了产品的操作。 模块2的测试人员对模拟的后台数据进行测试，在网店页面的搜索结果正确。

表5-5是跨这两个模块的一个功能测试用例。

**表5-5商店产品搜索功能测试**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测 试 类 型 | | **功** **能** **测** **试** | | 名 称 | **电子商务商店产品搜索功能测试** | |
| 用例描述 | | 该用例用来测试在用户登录后能否搜索到后台已经删除的产品 | | | | |
| 前置条件 | | 产品管理员(Admin/Admin)正确删除产品，用户(用户名/密码为TestUser/Test1User)正常登录， 有搜索权限 | | | | |
| 编号 | 测试项 | | 操作步骤 | | 数据 | 期望结果 |
| 1 | 以管理员身份登录后 台产品管理系统 | | 输入正确的用户名 和正确的密码 | | 用户名：Admin  密码：Admin | 登录产品管理系统 |
| 2 | 从库存中删除产品 | | 删除产品 | | 产品A | 产品A从数据库中被删除 |
| 3 | 用户登录Submit | | 输入正确的用户名 和正确的密码 | | 用户名：TestUser  密码：Test1User  直接车键(Enter)进行登 录 | 正常登录，转入对应的系 统页面 |
| 4 | 用户搜索产品A | | 在搜索框中输入A, 单击“搜索”按钮 | | 输 入 产 品 A | 提示用户：“没有符合条 件的产品” |

贺晨对小艾说：“在实际集成测试中，这个测试用例发现过系统的缺陷。当管理员删 除了产品后，依旧能够在页面搜索到本已被删除的产品，测试人员在执行测试的第四步，

第5章 黑色盒子：打着手电抓虫子

会发现实际结果与期望不一致。”

为什么会出现这种状况呢?贺晨告诉小艾，出现这种情况的可能性是比较多的， 例如产品在数据库中被删除后，同步机制没有做好，或者有些网店采用的搜索机制比 较复杂，在搜索产品的时候不是直接去数据库读取，而是使用缓冲技术，如 cache 技术 等，这些都会导致上述问题的出现。小艾听了贺晨分析，才理解了问题的根源及“合 而治之”的必要性，这些小虫子真够狡猾的!贺晨告诉小艾，除了这种的跨模块/解决 方案功能测试之外，在大型Java EE的应用中，还经常有下面两种“对黑盒子的全方位 照明”的测试：

(1)有的产品的是基于模块化开发，或者本身就是由若干个小的产品组成的。每个模 块/小产品开发完成之后，进入统一的集成开发阶段，这时候的测试也是一种常见的“合 而治之”的例子。

(2)不同产品间的集成。在有些应用开发中，存在着很多产品的集成，取长补短，完 成真正的用户商业流程。比如一个电子商务的产品，会与仓储管理系统的产品集成，进而 完成统一的电子商务流程。在这样的应用开发场景中，需要做充分的产品集成测试，以保 证功能、性能等的可靠性。

无论哪种情况，都要充分理解实际的商业需求和流程，进而定义完备而准确的功能测 试场景和测试用例，做到对黑盒子完全照明，让小虫子真的无路可逃。

**5.3.2** **手记** **—** **—** **策略高手**

通过这段时间对功能测试项目的完整参与，小艾不仅积累了丰富的执行经验和各种功 能测试技能，同时在贺晨的帮助和自己的努力下，对功能测试的测试策略有了比较深刻的 体会。小艾总结出了以下的一些功能测试策略要点：

**1.** **功能测试范围**

大型的Java EE 应用涉及的开发技术、展现层次都不尽相同，比如有基于Web的应用 和展示，基于Java 的应用和展示，基于服务的SOA 框架，命令行相关的功能等。在定义 功能测试范围的时候，要兼顾功能测试的各个不同层面，比如要兼顾界面测试(UI testing) 与命令行相关(内部命令、方法之间的数据和消息传输的测试，也就是API 或者Command level 的测试)的测试用例的比例关系。

**2.** **与其他测试类型的关系**

一个大型产品的测试都是很复杂的，会涉及不同的测试类型。在考虑功能测试的时候， 一定要考虑到和其他测试类型的交叉与依赖关系，比如功能测试中一个模块或者解决方案 的被选为可用性回归测试 (Sanity regression testing) 的功能测试用例，其是否能够顺利通 过测试，往往是性能测试开始的标志。还有，从测试环境的选择考虑，除了要在纯粹的运 行时环境去测试产品功能外，还要考虑在客户化的环境及移植后的环境去测试产品功能， 以保证在这些环境里原有的或者新开发的功能能够正常运行。

**3.** **测试覆盖率和测试效率**

从功能测试的目的来说，一方面要保证测试的完整性，达到最大的测试覆盖率，覆盖 到所有的功能点；另一方面，从实际项目开发的角度看，做到100%的没有任何缺陷的测 试是不可能的，除了技术层面的原因，测试资源是最大的限制，因此要充分考虑测试的效 率。因而从测试策略上就要兼顾测试覆盖率和测试效率，比如通过合理定义回归测试的比 例选择回归测试的种类，来达到测试覆盖率和测试效率的平衡。另一个例子是为了兼顾不 同的操作系统和数据库 (OS/DB), 在开发功能测试用例的时候，会把操作系统和数据库 敏感的用例标为 OS/DB 相关的，这样在有类似需求的时候，只运行这些OS/DB 相关的功 能测试用例就可以了，既保障了测试覆盖率，又提高了测试效率。

**4.** **功能测试用例自动化开发的策略考虑**

自动化对于功能测试能否重复执行是非常关键的。在功能测试策略的层面一定要定义 自动化开发的范围，方法，标准，这是功能测试用例重复执行的基线。

**5.** **功能缺陷的问题分析**

策略层面要明确问题分析时的一些具体要求，比如需要提供一系列标准的信息，以帮 助开发人员更好地理解问题、发现问题，找到解决办法。

**6.** **工具和模板**

定义合适的功能测试工具和各种测试模板(比如测试计划书模板、缺陷模板等)是功 能测试能够高效进行的策略考虑之一。

第5章 黑色盒子：打着手电抓虫子

**5.4** **功能测试之百种变身**

上述的功能测试更多的是小艾参与的、关于在大型应用某一个版本中的新的功能或者 新的解决方案从开发流程中各个不同阶段，对于如何发现隐藏的虫子，保证软件质量，而 进行的针对这些新的功能或者新的解决方案的功能检查。

但对于一个大型应用的功能测试，从策略的角度，要考察的维度很多，限于篇幅，不 一一详述，下面是小艾学到的一些方面，简单列举出来供大家参考。

**5.4.1** **回归测试**

**1.** **什么是回归测试**

在软件开发的整个生命周期中，每个阶段都有可能对软件进行修改或更新，而当软件 发生变化后，都有可能对软件的现有功能造成影响，因此，必须重新测试软件的现有功能。 这一重新测试的过程，就是回归测试。

**2.** **为什么要做回归测试**

无论是瀑布开发模式(Waterfall Model) 还是敏捷开发模式(Agile Model), 回归测试 在整个软件生命周期中都扮演着非常重要的角色。在敏捷开发模式 (Agile Model) 下，软 件新版本进行周期性发布，为确保软件功能的正确性和一致性，回归测试进行得更加频繁， 有效地降低项目的风险，提高软件的质量。

**3.** **回归测试的执行策略**

回归测试过程中，并不需要将所有的测试用例都用来做回归测试，而是根据策略选择 部分或全部测试用例，如表5-6所示。

在回归测试的执行过程中，要兼顾效率和覆盖率两个方面，表5-5中列出了常用的回 归测试策略，在迭代开发过程中，可以根据实际情况进行选择。

**表5-6回归测试策略**

|  |  |  |
| --- | --- | --- |
| **回归测试策略** | **比** **例** | **选** **择** **原** **则** |
| 完全回归测试(Full Regression Testing) | 100% | 为了最有效地降低遗漏回归错误的风险，选择全部测试用例 |
| 回归测试(Regression Testing) | 通常比例较小，一般 为20%～40% | 选择主要功能相关、易受修改影响的、少量非关键功能相关 的测试用例 |
| 可用性回归测试(Sanity Regression Testing) | 比例很小，通常为 5%～15% | 只选择主要功能相关、易受修改影响相关的测试用例。其目 的是确认当前发布的版本是否已经具备了进行进一步测试的  条件。一般来说，可用性回归测试是回归测试的一个子集 |

注：以上的这些数据是基于IBM的最佳实践。

**5.4.2** **客户化环境下的测试**

**1.** **什么是客户定制环境下的测试**

在软件的功能测试中，除了根据测试计划在开发的软件本身进行测试外，由于现在的 软件通常都预留接口，提供丰富的定制功能，所以，还需要在客户定制环境下进行测 试，以保证客户定制环境下软件预留接口的可定制性、易用性及定制后软件的功能没 有问题。

**2.** **为什么要做客户定制测试**

在现实中，由于软件供应商开发的软件一般要么包括一般性的功能，要么是一些特殊 的功能，用户通常都需要根据自身的需求进行定制。而软件的定制功能是否易定制、可定制 都将影响客户体验。而定制后的软件是否会影响到已有的功能，也需要通过测试来保证。

客户定制环境下的功能测试的执行

在定制环境下进行功能测试，主要是对软件的重要功能进行测试，以保证定制后的软 件功能没有受到定制的影响。另外，对软件的定制部分，也要做一定的测试。在定制环境 下进行测试，可以结合回归测试一起进行。可以按照以下的步骤执行。

(1)根据定制接口，开发定制的代码，将定制的功能部署到软件上。

(2)在定制后的环境中执行回归测试用例，保证已有功能不会因为定制出现问题。

第5章 黑色盒子：打着手电抓虫子

(3)对定制的部分进行测试，保证定制的功能。

(4)定制功能与系统OOB (开箱即用)功能的集成测试。

**5.4.3** **随机测试(Adhoc** **Test)**

**1.** **什么是随机测试**

随机测试是指为了某一个未计划的目的进行测试，这种测试一般不会重复进行。实践 中，随机测试一般没有书面测试用例、测试目的，主要依靠测试人员的经验对软件的功能 进行测试。

**2.** **为什么要做随机性测试**

在实际应用中，一个成功的软件通常拥有很多用户，而测试计划是很难覆盖所有实际 应用中的场景的，而随机性测试正好可以部分地弥补测试计划的不足，还可以衡量测试计 划的质量。

**3.** **随机测试的执行**

随机测试在软件测试过程中扮演着重要的角色，尤其当软件临近发布、测试接近尾声 的时候，可以适量安排做一些随机性的测试以发现测试计划未覆盖到的那部分可能存在的 问题。表5-7中列出了随机测试执行中的一些方式和它们的优缺点。

**表5-7随机测试**

|  |  |  |
| --- | --- | --- |
| **随机测试功能点选取原则** | **优** **点** | **缺** **点** |
| 选取缺陷密集的功能模块 | 缺陷集中的模块可能会有潜在的问题， 所以可以有效保证软件缺陷不被遗漏 | 由于是缺陷集中的模块，会有很多冗余的 测试，造成资源浪费 |
| 选取缺陷重现率低的功能模块 | 能发现隐藏较深的缺陷 | 只能发现特定的问题，造成软件问题遗漏。 |
| 随机选取功能模块 | 能有针对性地选取覆盖率低的模块 | 需要测试人员经验丰富 |

通常随机性测试需要由测试经验丰富、熟悉软件的测试人员执行，而且不需要很长的 时间来执行。由于这种测试的随机性、探索性，通常都是不可重复的操作，因此是不能自 动化的。

**5.4.4** **可访问性的测试(Accessibility** **Test)**

**1.** **什么是可访问性测试**

软件特别是大型软件，在开发过程中，为了能够为残障人士的使用扫除障碍，使软件 实现更大的商业价值和意义，都需要支持一定的可访问性标准。比如W3C 的 Web 内容 可访问性规范 (WCAG) 。 在功能测试阶段就需要保证产品的可访问性符合标准。可访问 性测试就是给那些身体出现不同程度功能障碍问题的人们使用软件的可能性的测试。

**2. 为什么要做Accessibility 测试**

(1)做好软件的Accessibility 支持，会帮助一些特殊用户如盲人等更好地使用软件。

(2)有人喜欢使用键盘，如果软件的Accessibility 设计有问题，有的软件元素用键盘 就不能被访问到。

(3)有些国家颁布了法律，要求软件或网页必须符合相关的Accessibility 标准。

**3.** **怎么做可访问性测试**

在 WCAG 文档中，给过了评价软件可访问性的4个基本标准：

可感知，如图像可以通过文本表达。

可操作，如用户不使用鼠标，只用键盘或其他设备便可以与软件交互。

可理解，如正文不应该比它需要的更加复杂，且网站应以可预测的方式来运行。 健壮性，软件的可访问性不会在不同系统、不同平台上有差异。

根据可访问性测试的一般标准，可以选取回归测试用例作为基础测试用例，使用辅助技 术，比如Screen Readers, 不使用鼠标，只是用键盘来完成测试用例，保证软件的可访问性。

**5.4.5** **全球化测试(Globalization** **Test)**

**1.** **什么是全球化测试**

全球化测试的目的是对软件全球化功能的实现进行测试，确保软件适应全球化的市

第5章 黑色盒子；打着手电抓虫子

场。使软件满足对语言、文化、多语种支持的需求，并保证软件的用户界面风格和信息的 可转换性。确保软件可以在世界各地，在不同的语言、文化、习惯的环境下正常工作。

**2.** **为什么要进行全球化测试**

随着软件市场的全球化，对软件本身提出了能够支持全球化的要求，为了确保软件能 在世界各地被不同语言、不同文化背景的用户正常运行，在软件的测试阶段就需要进行全 球化测试，为软件获得更大的市场份额提供有力的保证。

**3.** **全球化测试要做什么**

一般全球化测试都由功能测试人员来执行，将全球化测试与功能测试整合，有以下 优 点 ：

功能测试人员更熟悉软件功能。

功能测试人员在测试过程中积累的测试用例，可以用来进行全球化测试，可以节 约时间成本。

全球化测试可以在软件的一部分功能发布后，与功能测试同步执行，且可以多个 功能整合进行全球化测试，这也符合敏捷的精神。

全球化测试包括以下几个方面的内容，作为一般性标准，可以根据所测软件的具体情 况，做有针对性的细化和调准。

(1)处理不同的字符集的能力。保证软件在不同的区域设置环境下，正确转换编码， 数据无丢失。

(2)保证软件在不同区域环境下，处理多语言文本没有错误，时间格式、货币格式显 示正确，根据不同语言排序正确。

(3)保证软件功能在不同区域环境下的正确性、 一致性。

**5.4.6** **其他之林林总总**

**1.** **大对象测试**

大对象测试是测试软件处理大对象时在数据边界是否能正常工作。

**2.** **文档测试**

文档测试是指在软件开发过程中对用户文档、安装手册等进行测试，验证文档是否与 实际软件存在不同，保证文档的正确性。

**5.5** **学习笔记——功能测试之小艾观**



时间过得很快，小艾在功能测试项目组已经完成了一个完整的项目测试工作。回头 看来，可以用收获颇丰来总结这段工作。梳理了一下，小艾对功能测试做了一个小小的 总 结 。

回顾功能测试的几个基本概念如下

功能测试的目标：按照客户实际使用产品的方式去功能测试该产品。

功能测试场景：按照客户需求把一系列用例 (UC) 组合到一起构建的场景。基 于实际，端到端。

功能测试用例：功能测试场景结合多种数据集/测试条件和验证结果构成功能测试 用 例 。

验证结果：功能测试用例要按照客户想要见到的结果去验证。

理解了功能测试基本概念的前提下，一个好的功能测试人员，既要在广度上熟悉各种 功能测试相关的技能、工具、流程，又要在深度上理解产品、功能测试策略。只有做到这 样的“宽度/深度”双修，才能够做到如图5-6所示这样一个典型的功能测试人员应该遵循 的流程图里每一步的工作要求，交付完美的功能测试工作。

笔者认为中国的软件产业目前还处于比较混乱的状态，开发成本越来越高，质量却有 越来越差的趋向。一个重要的原因就是对测试特别是功能测试的忽视，而且测试人员知识 与经验的不足。针对这样的状况，每个从事功能测试工作的测试人员，都应该苦练内功， 找到自己的手电，用来寻找光明。

第5章 黑色盒子：打着手电抓虫子

|  |  |  |
| --- | --- | --- |
| 一个典型的功能测试流程  分析需求  创建测试策略  制定测试计划  设计测试场景/用例  开发测试用例  执行测试用例  报告状态 | 安装  执行  清理 | 确定系统时间表  准备测试环境  安装/配置测试用例  执行测试用例  复制日志和结果文件  分析结果  更新状态记录 |
| 图5-6 | 典型的功能测试流程图 | |

**参考资料：**

[1]维基百科

<http://zh.wikipedia.org/wiki/%E7%BB%B4%E5%9F%BA%E7%99%BE%E7%A7%91> [2]<http://www.junit.org/>

[3]<http://seleniumhq.org/>

[4]下一代电子商务：理论、应用及基于IBM WebSphere Commerce平台的实践/《下 一代电子商务》编委会编著.北京：电子工业出版社，2009.8.图1-16

第 **6** 章

大促带来的灾难：

究竟能支撑多少访问量



性能测试是产品测试流程的必经之路，产品的性能好坏不但关系到产品的用户体验， 对于像电子商务这样的应用来说，性能好坏直接关系到客户的网站是否有好的用户忠诚 度，从而也就影响到订单转化率等跟客户盈利直接相关的指标。

**6.1洪水般的访问量——初识性能测试**

小艾已经逐渐成为有经验的测试人员，而更多的测试类型正在等着他。不久，小艾就 得到了机会，在性能测试团队体验了性能测试的全过程。

**6.1.1** **一旦宕机，流失的就是白花花的营业额**



测试团队每周都会有例行会议，这天小艾作为功能测试组的代表参加了这个会议。会

第6章 大促带来的灾难：究竟能支撑多少访问量

议的一个议题是客户之一的ABC 公司元旦会有一个大型促销活动。

“又快过元旦啦，客户那边活动很多啊。”凯文提到。

“是啊，ABC 过几天会有一个大促活动，不知道能不能挺住啊。”性能测试组的组长史 静看上去不太乐观。

“不至于吧。上次B 客户搞活动你也担心，最后还不是什么事情都没发生?”

“不一样啊。上次是限时抢购，访问量好歹还分散在几个小时之内。这次不但有限时 抢，还有秒杀，很极端，所有访问几乎在同一时间冲过来，对硬件和软件都是考验啊…… 一旦挂了，客户损失的就是白花花的营业额。”

小艾很好奇。我们的软件不是都测过了吗?在100%压力下跑也不会有问题的。

“我们能控制的不过是软件本身的部分，到客户那里被定制过之后，用户本身的代码 就不能完全保证了。之前在实验环境上测试秒杀就出了问题，我们当时去帮忙解决了， 但实际上线会不会有新问题还不确定。我这几天还是得准备准备，可能随时都得去客户 那里。”

“秒杀”,这个词小艾听过啊，现在人们恨不得见面就问“今天你秒了没有”。小艾也 曾经大半夜守在网上等着秒自己心仪的电子产品(当然大部分时间以失败告终)。他也碰 到过秒杀一开始网页就无法刷新的情况，骂网站的同时从没有想过这些秒杀背后的应用服 务到底出了什么问题。

小艾的好奇心顿时如野草一般疯长了起来。隔壁那个神秘的性能测试团队每天都在干 什么?他们怎么去确保应用服务能够撑得住洪水般的用户同时访问网站?正在琢磨着，他 蓦然发现双脚不知道什么时候把自己送到了凯文面前……

“又想换了?”还没等小艾开口，凯文已经把他的想法说了出来。

“是……”小艾不太好意思地说，“我现在想了解性能测试怎么做的。”

“好想法，我支持你。我希望你能从更多角度了解我们的软件是如何设计和测试的。” 就这样，小艾来到了性能测试团队。

**6.1.2** **假如我是甲方** **—** **—** **关于性能客户都关注些什么**

史静几乎每天都在对着一堆花花绿绿的图左看右看，小艾也不知道他在看些什么。从

从菜鸟到测试架构师——一个测试工程师的成长日记

这些图图表表中就能看出性能问题吗?于是便向史静请教这些图怎么看。史静劝小艾先不 要急着去看这些细节。要了解性能测试怎么测，首先得知道性能测试测什么,也就是我们 的客户在系统的性能方面都关注些什么。

首先需要明确两个概念：客户和用户。客户指的是购买我们产品的企业，他们购买 了产品之后，经过定制上线，展示给最终的用户来使用。因此客户非常关心用户的使用 感受，因为这直接决定他们的网站或者终端是不是受欢迎，能不能留住用户。而产品性 能好坏，会在很大程度上影响用户的使用感受。没有人愿意在一个反应缓慢、总是出错、 动不动就失去响应的网站上停留。

举个例子说，对于一个应用服务的性能来说，客户一般关注以下几个方面：

页面/客户端的响应时间

响应时间直接影响最终用户的使用体验，从而在很大程度上影响用户忠诚度。 一些 行业权威评测机构的评测数据可以作为响应时间通过标准的参考，某些客户也会对产品 的响应时间有自己特殊的需求。

服务器的吞吐量

常用的是系统每小时能处理的业务量。例如，最高每小时能接受的网站浏览次数， 或者一个电子商务网站每小时能下多少个订单。

最大并发用户数

这里的“最大”其实并不是真正的最大，而是在响应时间合理的情况下，能承受的 并发用户数。也就是系统最多能承受多少用户同时访问且用户感觉不到页面响应变慢。

是否能稳定的长期运行

任何客户都不希望自己的系统动不动就宕机，这会对最终用户造成非常不好的印象 从而给客户的业务带来大的损失。应用服务系统是否能长期稳定运行除了跟服务器的硬 件有关系之外，与软件本身也有很大的关系。

最大数据规模

能保证正常访问的最大容忍数据规模。

服务器后台操作是否会影响前台性能 C 什么情况下会导致系统崩溃

第6章 大促带来的灾难：究竟能支撑多少访问量

这些性能指标相互之间并不是独立存在的，它们之间相互影响。某些指标的提升可 能会引起其他指标的下降，在测试的时候需要做综合的考量。

了解了性能方面客户都关心些什么,我们就大概明白了应该用什么方法去做性能测 试。其实说得直白一些，最普通的性能测试，就是给系统一些压力，看系统跑得快不快、 好不好。说得文一些，就是让系统工作在一定的负载状态下，把系统工作的性能指标与期 望的性能指标相比较。¹这里说的负载是广义的负载，包括并发用户、用户连续两次操作 之间的间隔时间(称做思考时间),也包括系统中包含的业务数据规模。

性能测试策略和方法的出发点，就是要模拟客户对系统的访问行为，包括并发、压力、 长时间的访问等。接下来就讨论一下在性能测试里如何模拟客户的访问行为。

**6.2** **让访问来得更猛烈些吧——模拟客户的访问行为**

想要让用户得到最好的性能体验，最简单有效的方法就是模拟客户使用我们的应用时 实际遇到的访问行为，包括并发用户访问量、访问操作间隔时间、服务器所接受的访问压 力(比如服务器的CPU 使用率)、管理员在服务器上做的维护性操作等。而对于并发访问 等行为，在测试环境里最常采用的方法就是用自动化测试程序来进行模拟。

**6.2.1** **更真实更高效的模拟——自动化的性能测试**

如果说功能测试还有选择手动测试或者自动测试的权利，性能测试则不可避免地要用 到自动化测试。所谓自动化测试，以电子商务应用的测试为例，就是通过程序来模拟实际 用户对于网站的访问行为。一般通过这些测试程序来实现多用户混合场景的并发访问。

自动化测试的程序可以自己开发，也可以使用现有的自动化测试工具，比如 Rational Performance Tester(RPT),LoadRunner,SilkPerformer 和 JTester 等。

一般来讲，对于类似于WebSphere 应用这种庞大且功能繁多的大型软件的测试来讲， 不是简单地录一些脚本加一下压力就可以测的。通常会设计一个比较完备的测试框架。这 个框架需要满足以下的要求。



1 引自《构建高性能WebSphere企业级应用》一书。

**1.** **测试程序的模块化和可重用性**

实际测试中，不同的测试分支可能会具有同样的步骤。比如注册用户在注册完毕后会 和匿名用户走到同样的浏览页面->添加物品到购物车->下单->付款的步骤。但是没有必要 针对注册用户和匿名用户各写一套从头至尾的测试脚本。如果把注册、浏览产品、添加到 购物车、下单、付款分别做成独立的脚本模块，在使用的时候添加不同的模块即可组成不 同的测试分支。但是因为不同的分支可能在每个模块的使用上具有特定的条件，不同的输 入会导致该模块进行不同的操作，从而产生不同的输出，所以在定义每个模块的时候把输 入/输出定义清楚是关键。

**2.** **功能分支的可选择性**

为了模拟实际客户的访问，往往将多个测试分支混合在一起进行并发访问。因此，测 试脚本必须是可以选择混合哪些分支的。

**3.** **可定义的输入数据**

脚本必须可以定制输入的数据，比如登录用户的用户名密码、访问哪些产品目录、输 入的送货地址等。

**4.** **高可靠性**

意味着对于同一套脚本，同样的输入，同样的测试环境，多次运行出的结果应该相同 或者非常接近。否则测出的结果没有可信性。

**5.** **可扩展性**

可扩展性表现在很多方面。比如一个新版本的产品，在旧版本的脚本上面添加新的功 能分支就可以使用。再比如，同样的脚本在小规模的测试上可以使用，换到大规模数据的 测试环境上也可以使用。对于高负载压力测试或者普通负载的响应时间测试，甚至长时间 的可靠性测试，一套脚本就可以解决，测试人员只需修改一下并发用户的数量、测试时间 等就可以实现。

有了自动化的测试框架和脚本，下一步就可以讨论怎么使用这些脚本来做测试了。先 看看有哪些性能测试类型。

第6章 大促带来的灾难：究竟能支撑多少访问量

**6.2.2** **春节大促——压力测试**

像前文提到的那种节日的大促销，例如限时抢购，是一种短时间内把系统负载压到极 限的例子。在测试的时候，一般通过压力测试来模拟。

压力测试 (Stress Test) 主要考察让系统运行在比较大的负载条件下的性能表现。 一般会采用多个并发用户进行一段时间高压力的访问(用户的思考时间不一定限制为0, 但通常要求有多个测试用户)。压力测试可以发现系统工作在多任务并发情况下可能出 现的性能问题。由于压力测试下系统工作在非常高的负载状况下(至少不是应用系统日 常负载条件下),所以衡量的性能运行指标一般为吞吐率和错误率，而不包括响应时间。

压力测试的目的：

发现并解决在并发、长时间运行或者大量数据情况下才出现的系统缺陷。

理论上在通过功能测试之后，单用户的操作已经不会出现问题。但是有些缺陷只有 在多用户并发访问、系统长时间运行或者数据量达到一定规模情况下才会暴露，比如数 据库死锁、代码级死锁、内存泄漏等。

在指定并发用户数下是否能达到吞吐率目标。

吞吐率通常是指应用系统在单位时间内实际处理的交易数量(交易吞吐率)或页面 点击数量(页面点击吞吐率)。与系统负载指标类似，吞吐率也随时间而变化，存在最 大值和平均值。

一些重要的组件的吞吐率往往被作为一个版本的关键质量标准，决定软件版本是否 能顺利发布。

新版本的吞吐率往往用来与旧版本相比较，以确保新功能的加入不会引起整体的性 能下降。

压力测试按照目的不同又可细分为两类。一类压力测试是找出应用系统的饱和点，即 系统的性能瓶颈。通常希望该瓶颈是系统的 CPU 处理能力。这类压力测试的主要通过标 准是系统在CPU 运算能力饱和状态下(如 CPU占有率大于95%)的吞吐率和错误率。另 一类压力测试则是找出应用系统的崩溃点。这类测试通常不关心具体的吞吐率和错误率指 标，而是考察吞吐率、错误率指标与负载的关系曲线。

在没有特殊要求的情况下，压力测试一般会针对系统中最常用或最容易出问题的功能 分支进行施压。一个测试用例可以把不同的功能分支混合在一起，参考实际情况下各个分

支被调用或访问的比例来执行。有些测试用例也会依据一些典型的客户行为实例。譬如限 时抢购，其实是一种类似于极限压力测试的客户访问行为。更进一步说，是一种针对小范 围商品的极限压力访问行为，因此它比一般的压力测试更为极端，用户的浏览和下单都集 中在少数的几个参加促销的产品页面上。因此在设计压力测试的测试用例的时候，可以设 计一些针对小范围产品目录的大并发浏览和访问(当然具体的目录访问范围、客户的整体 数据规模、最大并发数等可以作为设计用例的参考输入。

压力测试的通过与否可以参考以下的标准：

在承受指定负载(例如应用服务器CPU 使用率超过90%)的前提下，系统的 吞吐率是否达标

并发用户数是否达标

出错率是否达标

在并发量很大的情况下，有的事务可能因为等候资源超时而造成回滚，少量的访问 失败在所难免。但是如果大并发下出错率很高，就说明系统处理并发的能力不足，有可 能是并发处理逻辑有问题。

小艾问：“那么吞吐率和并发数的好坏受什么影响呢?”

史静回答道：“一般认为，吞吐率越高表示系统的性能越好。但值得注意的是，各种 交易的复杂程度不同，每个交易页面的点击数也可能相差很大。脱离具体交易的内容，片 面通过两个系统的吞吐率来衡量两个系统的性能好坏是没有意义的。”

而对于同一个交易系统来讲，我们希望吞吐率能够与并发数成正比。当然这是理想情 况，现实中由于硬件等因素的影响，CPU 的使用率不可能无限上升，因此会在某个点(也 就是CPU 使用率接近100%时)达到饱和。饱和之后，再增加并发数，吞吐率也不会增加。 相反，由于系统要在大量的线程之间进行资源的切换以满足更多的并发访问，反而会使每 个访问的响应时间增加，出现吞吐率下降的现象。如果饱和之后继续增加并发量，系统往 往会在某个点崩溃。

考虑以上情况，最大吞吐率和最大并发用户数可能并不是在同一点出现的，在设计压 力测试用例时，就可以考虑把吞吐率和并发用户数作为性能指标分别测试。而对于最大并 发数引起的崩溃点，也不一定要定义成整个系统真正意义上的崩溃，可以定义成响应时间 长度超过某个阈值(比如定义某个用户可以容忍的最大响应时间)或者大量操作超时出现

第6章 大促带来的灾难：究竟能支撑多少访问量

的时间点。

吞吐率测试往往可以用于产品新旧版本之间的比较试验。比如，在新版本增加某个新 功能之后，你想验证新功能的引入会不会对性能产生负面的影响，就可以通过在相同的硬 件条件下做吞吐率的比较试验来确认。

**6.2.3** **日常的访问量——正常的响应时间**

响应时间测试(Response Time Test)是另一个常见的衡量系统快慢的运行指标。 一 般来说，响应时间越短越好。

与将系统负载压到极限状态的压力测试相比，响应时间测试测的是在正常负载状态 下，被测系统的服务器端和客户端响应时间(不考虑外部网络传输影响)。也就是说， 响应时间的测试结果更接近通常状态下的系统真实性能表现。

响应时间测试的目的：

在正常负载(例如应用服务器 CPU 使用率在50%左右)前提下，保证服务器或者 客户端的操作响应时间不超过规定的标准。

小艾问：“网络上传输数据的时间也包含在响应时间内吧?”

史静又解释道：“一般外部网络的影响比较不稳定，而且是测试无法衡量的。因此一 般响应时间测试都是在内网环境中进行的。也就是说，我们所测得的响应时间里面包含了 内网的网络传输时间。”

首先需要知道用户感受到的网页响应时间由哪些部分组成。用户的页面请求从发出到 完全得到响应有一个过程：在网络上传输数据、在服务器端处理，服务器将响应数据传回 到客户端，在浏览器中显示。这几部分是相互独立的。

在网络上数据传输的时间取决于页面数据量的大小、网络速度和网络上的拥挤程度。

在浏览器中显示的时间取决于客户端程序(JavaScript,Dojo,Flash 等)的复杂程度， 以及运行浏览器的客户计算机的处理速度。

服务器端处理的时间与服务器系统的性能相关，这段时间称为页面处理时间 (Page processing time), 又可细分为队列等待和实际处理两部分。实际处理时间与吞吐率关系较 小，主要取决于处理程序的执行效率(算法好坏)和硬件处理速度。而队列等待时间除了

与硬件处理速度相关，还跟各种共享资源的参数设置有关，比如网络服务器/应用服务器 的连接池参数设置。

响应时间中的最不确定因素是网络上传输数据所需要的时间和浏览器显示时间。前者 受网络速度和拥挤程度的影响，而后者受到客户计算机的处理速度和浏览器运行速度影 响。尤其是后者，随着富客户端技术 (Web 2.0,Flash等)的迅速发展，越来越多的计算 被转移到客户端浏览器内执行，客户PC 的运算能力大大影响到客户看到的响应时间。这 也就意味着实验室里测到的响应时间好不意味着真实用户看到的响应时间好。但这并不意 味着我们就没有必要关注这两部分的时间，只考虑提高服务器端的处理时间。尽管无法控 制网络传输速度和客户端的计算能力，还是应该从设计上尽量避免在这些方面的影响。比 如，不要在网络上传输过大的内容。在客户端执行程序的复杂度也要尽量小。

在设计响应时间这方面的测试用例的时候，会针对服务器端和客户端分别设计用例， 并且分别定义通过标准。

**6.2.4** **保证长时间的稳定运营——可靠性测试**

一般来说，可靠性测试(Reliability Test)对待测系统施加的负载要小于压力测试的 负载。一般可以选取系统日常处理的平均负载。所以测试用户在交易过程中一般都采用 与终端用户实际操作接近的思考时间。可靠性测试评估的性能指标通常是响应时间、错 误率和系统资源占用率。为了更好地发现问题， 一般可靠性测试的测试周期都比较长， 比如压力测试执行3小时，对应可靠性测试可能需要执行72小时甚至更长时间。

可靠性测试的目的：

(1)评估长时间的性能指标是否满足要求，往往考察平均值。

(2)考察性能指标的变化趋势，以发现系统可能存在的内存泄漏、性能下降等与执 行时间有关的性能问题。

(3)考察某些维护性操作(比如一些后台操作，像数据备份/加载/清理，多节点同 步，某些定期作业)是否会对前台用户访问的性能造成大的影响。

可以说可靠性测试是一个版本性能的终极测试。可靠性测试的设计往往需要考虑把要 测试版本最关键功能和最基本功能都涉及到。所以往往要在所有功能都单独测试过确认没 有性能问题了，才会把它们都混合起来放在一起长时间运行。

第6章 大促带来的灾难：究竟能支撑多少访问量

很多时候3小时的压力测试看不出来的问题，长时间运行可能会暴露。或者1小时的 响应时间结果满足要求，但长时间运行会发现响应时间越来越长最终导致超出通过标准。 还有一些后台资源的使用问题也有可能通过长时间的运行暴露出来，比如内存泄漏。

而对于后台维护性操作(比如数据备份、更新商品信息、清理过期订单等)也要注意。 一般客户都会把维护性操作放到夜间或其他访问量不高的时段，为的就是避免维护性操作 影响前台用户体验。但是也有一些必须要做的紧急维护等操作，可能会直接与用户的操作 遭遇。因此在维护性操作执行的时候要注意前台的响应时间和错误率有没有明显的上升。

**6.2.5** **客户的成长不比产品慢：想象不到的数据量——可扩展性测试**

随着客户的成长，业务扩展得越来越大，用户积累得越来越多，对系统处理能力的要 求也就越来越高，往往会发现用户要求的数据量很快就会超出我们预想的数据量。这时， 对系统进行扩展就成了当务之急。

一般来说，如果想对系统进行扩展(增加系统的负载承载能力),增加硬件(比如CPU, 硬盘等)是一种可行的办法。但是是不是增加了硬件就一定能增加性能呢?这就依赖于系 统可扩展性的好坏了。对于可扩展性差的系统，即使增加了硬件，应用服务可能也不能达 到预想的处理能力。对系统进行可扩展性测试能在一定程度上验证系统在这方面的能力。

可扩展性测试(Scalability Test)是验证被测试系统随着计算能力的扩展是否能够承

受更大的负载。负载的增大包括数据规模、业务种类、数据复杂度等。

可扩展性测试的目的如下：

可扩展性测试往往针对的是系统在某一个负载方向上的可扩展性。所以可扩展性测 试通常包括一组测试，该组测试的每一个测试用例中使用的负载不同，通过这一组测试 评估系统在不同负载下的性能情况，从而分析系统在此负载变化方向上的可扩展性。

扩展性测试一般在不同的负载下衡量系统响应时间或者吞吐率的变化趋势。

举个例子，在电子商务的测试中常用的一种扩展测试是针对业务数据的规模，比如订 单列表的大小。按订单列表从小到大设计一组数据，在每种情况下测试订单列表显示的时 间，看看响应时间随着订单数目增长而增长的情况。

不管是吞吐率、并发用户数、响应时间还是可靠性，任何性能指标的好坏，除了与软 件本身的并发性能好坏有关，还在很大程度上取决于硬件系统的处理能力。在高速或者多

核/多处理器的服务器上运行的并发性能肯定比低速或者单核处理器更好。因此在谈及一 个系统的性能测试结果的时候，不可能脱离硬件系统的先决条件描述。在制定性能测试计 划的时候，也需要定义好每个测试用例要使用的硬件环境。

**6.2.6** **摸爬滚打过来还得挨一刀——产品经历性能测试**

小艾现在对于性能测试都要测些什么有了基本的认识，下一步就是搞清楚怎么测了。 性能测试的测试流程大体与功能测试等其他测试阶段很类似，都从制定测试计划开始。

**1.** **制定性能测试计划**

性能测试计划需要涵盖以下内容：

性能测试可以开始的必要条件，也就是开发人员、其他类型测试人员为了保证 性能测试顺利进行而必须做的工作。

举个例子：对于某个功能模块的性能测试，必须在开发人员完成整体的设计、编码 和单元测试，以及功能测试人员完成相关功能测试后才能开始；或者所有性能测试所涉 及的硬件或者软件平台必须经安装测试验证过没有安装问题等。对于使用敏捷开发 (Agile) 模式的产品，性能测试可以在更早的阶段进入，但是仍旧需要对进入的条件进 行量化的定义。否则可能使性能测试人员碰到大量的安装或者功能问题从而影响性能测 试结果的可靠性，阻碍性能测试的进程。

性能测试退出条件。退出条件定义什么时候性能测试结束，退出条件通常与产 品的质量控制标准有关。理想状态下，性能测试的退出条件应该是：所有的测 试用例都已经执行过，所有的性能问题都100%得到了解决。但是实际项目中， 往往很难达到这样的状态。所以定义明确的退出条件对于性能测试能够顺利结 束非常重要。

性能测试目标与范围。性能测试的目标往往直接来自于用户需求规格说明书中 的非功能性需求描述。如果这种需求在说明书中定义得不明确，测试人员有义 务与设计人员沟通以明确性能需求。

测试用例。测试需求/目标和范围只是对于测什么的总体定义，具体测试的内容 需要在测试计划中通过测试用例来定义。测试用例应该包括测试场景的描述、

第6章 大促带来的灾难：究竟能支撑多少访问量

测试输入/输出描述。输入包括测试数据、测试时长、并发虚拟用户数量和变化 曲线、测试软件/硬件/平台类型等。输出包括测试结果通过标准、测试结果及日 志包的位置等。

小艾说：“那么编写性能测试计划的第一步是不是就是获取用户需求，并且搞清楚这 些性能需求从哪里来呢?”

性能需求可能有几个来源。

(1)性能需求根本上应该来自实际用户的需求

测试人员往往是通过开发人员的解决方案及设计文档中对性能的描述来获得需求。因 此测试人员应该参与解决方案规格说明书的审阅，并就其中涉及的性能指标进行澄清。对 于有明确性能指标定义的描述都应该在测试计划中制定相应测试用例，量化的性能指标应 该成为制定测试用例通过标准的依据。此外，可以通过产品的销售人员、技术支持人员得 到一些用户的实际需求信息。比如对于测试用例中的测试硬件配置、测试数据规模、各种 测试路径分支在测试用例中占用的比例等信息都应该来自于真实客户的信息。

(2)以前版本的用户缺陷报告

软件产品很难在发布之后没有任何性能问题。所以性能测试人员应该关注用户报告的 性能缺陷。由于用户报告的性能缺陷反映了之前的产品性能测试中的漏洞，因此这也成为 下一个版本改进性能测试的依据。

(3)行业的质量标准

比如，电子商务应用对页面响应时间的通过标准可以来自于一些行业标准。此外，电 子商务应用的版本性能对比测试也来自于一些企业标准，每个版本的性能测试都会选择一 些回归测试用例用于不同版本性能对比测试，以确保版本之间的性能区别符合企业制定的 标 准 。

(4)由于硬件/软件版本升级带来的性能提升测试需求

比如，产品所基于的应用服务器产品或者数据库产品有了新的版本，产品在采用了新 的软件版本之后可能需要做相应的回归测试，以保证在新版本上不会出现性能下降甚至应 该出现相应的性能提升。

当然，通过这些渠道获得的只是理论上应该涵盖的性能测试的范围。对于任何一个实 际的测试项目而言，资源和时间都是有限的。所以在测试范围上必须有所取舍。具体的取

舍依赖于各种需求的重要程度、项目的周期、产品管理者对质量标准的定义等。

了解了测试需求，下一步是怎么把测试需求转化为测试用例。测试需求只是对于测什 么的总体定义，具体测试的内容则需要在测试计划中通过测试用例来定义。 一个完整的测 试用例描述如下。

(1)测试场景

描述待测系统处理的业务内容，或者说测试用户所执行的业务操作。在性能测试中设 计一个测试场景与功能测试中设计测试场景类似，包括：

系统交互的角色(比如注册用户、非注册用户、销售代表、销售经理、网站管理 员 等 ) 。

角色所执行的业务流程(比如登录→选择进入产品目录、浏览→选中商品，并添 加到购物车→浏览购物车的内容，生成订单→生成订单的详细信息→添加新的送 货地址→更新订单→添加支付信息→提交订单)。实际性能测试使用的测试场景 往往包含多个分支处理和随机控制。比如注册用户登录之后，有一定的概率进入 产品目录浏览，也有一定的概率检查注册信息。

与功能测试不同，性能测试一般都不会涵盖系统所支持的所有可能的功能场景，所以 需要做出取舍。设计测试用例的原则之一就是用尽可能小的测试场景覆盖尽可能多的测试 目 标 。

(2)测试负载和通过标准

这里主要是指并发测试用户数、思考时间、测试持续的时间等。通过标准则包括吞吐 率通过标准、页面响应时间通过标准、资源占用率标准等。不同的性能测试类型可能采用 不同的负载定义和通过标准。

(3)测试数据规模

测试数据在广义上属于测试负载的一部分。实际上，即使测试系统相同，测试的场景 和并发用户数也相同，测试数据规模不同，系统的性能表现或者说性能测试的结果也会完 全不同。测试数据的设计应该从实际用户的需求出发。但是一个产品的测试不可能将所有 用户的数据需求分别进行测试。为了简化测试数据的生成和维护，可以按照典型的客户的 数据规模为测试数据进行分类，然后将近似需求的测试数据合并，形成测试数据集 (Test DataSet) 。每个测试数据集代表一组经典的用户实际数据规模，以电子商务应用为例，数

第6章 大促带来的灾难：究竟能支撑多少访问量

据的描述可以包括商业模式、用户数目、产品目录大小和结构、合同数量、促销打折数据、 订单数量等。不同测试用例可以共享测试数据集。

(4)待测系统软/硬件配置和拓扑结构

在不同软/硬件平台和拓扑结构上测出的性能测试结果会有或大或小的区别。测试计 划中应明确描述操作系统版本在内的所有测试相关软件版本信息、硬件平台和详细配置信 息及系统的拓扑结构。以压力测试为例，不可能期望在拥有1个CPU 的普通PC 上得到和 在拥有4个CPU 的 p 系列服务器上相同的吞吐率测试结果。对于比较大的数据量而言， 还要考虑比较详细的硬件配置，包括网络速度、磁盘阵列配置等。应该尽可能地接近用户 实际使用的软硬件配置。

测试用例是进行测试的基本单位。测试用例的设计与实现与软件开发一样，都是一个 创造性的过程。测试用例设计不合理有可能导致测试结果的无效性，进而引起测试返工甚 至有可能发现不了系统性能缺陷。

**2.** **执行测试**

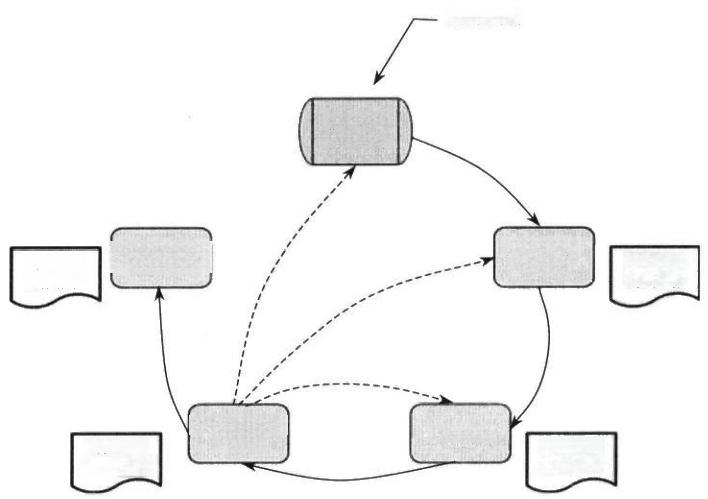
执行测试用例从测试用例的实现开始，包括测试脚本和测试数据集的开发、测试执行、 测试过程中的系统监视、收集测试结果、分析测试结果等。

性能测试的特点决定了性能测试的执行与其他类型测试相比要复杂许多。性能测试执 行过程中至少要监视待测系统各方面的运行状况，所以性能监视技术是性能测试人员必备 的技能。同时，由于系统性能可能会被很多因素影响，对于同一个测试用例，不同的测试 人员因为操作的差别可能会得到有差异的测试结果，因此，有一个明确的测试执行清单有 助于测试执行过程的规范化。测试执行清单将测试执行过程分解为若干小的步骤，每一步 包括怎么做及为什么这样做。为了确保测试人员能够完全按照测试清单执行，还设计若干 检查点(Check Point)。测试执行清单的制定可以便于新加入测试项目组的成员快速掌握 执行测试的方法，降低因为人为因素导致的测试结果差异。明确而清晰的测试执行清单还 便于设计一定的自动工具进行自动化的测试执行工作。随着被测系统的版本增加或者软/ 硬件的升级，测试执行清单应该随之进行更新并被所有测试人员知晓。因此，测试执行清 单应该像测试脚本和数据一样被实时和定期维护。

与功能测试不同，性能测试结果如果不满足通过标准不一定是应用系统本身的缺陷， 因此执行测试用例得到的测试结果需要进行分析。分析的结果很可能是系统配置参数的问 题，即需要修改系统调优参数后重新执行测试；也有可能是测试用例有问题(脚本或数据

有问题),那就需要修改测试脚本或测试数据后重新执行；甚至有可能是测试计划本身有 问题(通过标准有问题乃至测试方法有问题),那就需要从测试计划开始返工。最终也可 能确定是被测应用程序本身的问题，那就可以开缺陷报告，要求开发人员修改应用程序。

因此，可以将性能测试的流程分为5个阶段，图6-1中标出了每个阶段所产生的文档或 输出。性能测试的过程很可能是个递进的过程，很多环节的问题都有可能导致测试的返工。



从这里开始

编写性能 测试计划

设计与实现 测试用例

测试环境 测试结果

性能问题确定 与调优

调优参数 新测试结果

测试报告 缺陷报告

测试脚本 测试数据

编写测试报告

执行测试

图6-1 性能测试流程图

接下来详细看一下一个测试用例的执行过程的每一个详细步骤。

**步骤1.熟悉测试计划与测试用例**

有一些测试人员习惯于拿到测试用例就开始执行测试，但是缺乏对测试用例的正确理 解就开始执行测试往往会有偏差导致得不到正确的测试结果。由于性能测试的复杂性，反 复阅读测试计划中对测试方法和测试用例直至彻底理解是开始执行测试的前提。对性能测 试用例的各方面要素：测试场景、测试负载直到测试软/硬件环境的理解都要确保没有偏差。

第6章 大促带来的灾难：究竟能支撑多少访问量

性能测试的组长可以要求所有测试项目组成员参与测试计划的审阅，召开测试计划审 阅会议，由所有测试人员分块介绍测试计划的内容等，以确保所有测试人员都阅读并理解 了测试用例。

**步骤2.搭建或检验测试客户端**

性能测试的运行通常要依赖于一定的自动化测试工具或框架。测试工具通常运行在称 为“测试客户端”的计算机上。为了保证测试工具能够良好地运行，必须在搭建客户端环 境或使用其他测试人员提供的客户端时检查以下方面：

测试客户端的操作系统版本是否满足测试工具软件的要求，安装的测试工具软件 版本是否满足测试要求。

测试客户端的硬件配置 (CPU 主频、内存大小)是否满足测试工具软件的要求。 各种硬件是否工作正常，包括磁盘剩余空间、网络连接速度等。

是否有异常的系统软件进程影响性能测试的执行，比如防病毒软件、防火墙软件等。 是否有别的用户同时使用该测试客户端，如果有，应该防止他们在测试过程中使 用该测试客户端，干扰测试结果的稳定性。

测试工具软件的各项配置参数是否正确。

**步骤3.搭建或检验测试服务器环境**

待测系统的软/硬件配置及正常工作状态是得到有效测试结果的前提。所以不论对于 自己搭建的测试服务器环境，还是从别的测试人员那里转交过来的测试环境，测试人员都 要仔细检查待测系统的以下信息：

软件版本和硬件配置，保证与测试计划的描述严格一致。对此，测试计划检查表 应该提供详细的检测步骤或自动化检测工具。

硬件是否工作正常(包括磁盘剩余空间、网络连接速度等),是否有异常的系统 软件进程消耗系统资源。这一点与测试客户端检查类似。

是否有别的用户同时使用待测系统。与测试客户端类似，其他用户在测试进行期 间访问待测系统的各项服务(比如FTP 服务)都可能造成测试结果出现异常。

应用程序安装过程是否正确。尤其是对于一些比较复杂的系统，安装过程中很可

能出现一些错误。这些错误或许并不会影响一般功能的使用，但是有可能影响性 能测试的结果。所以测试人员需要关注应用程序本身及包括应用服务器在内的各 种支持软件的安装日志，确保没有任何安装错误存在。

应用程序功能运行是否正常。一般通过手动执行一遍需要运行的测试分支，查看 界面功能是否正确，以及后台服务器日志是否在操作期间出现记录异常信息。手 工操作时的少量错误日志，看起来似乎与性能无关。但是在大量并发用户访问时 可能会引起大量的磁盘读/写，从而在很大程度上影响性能测试的结果。

测试服务器环境的检查。测试服务器上要留有足够的磁盘空间，用于运行时产生 文件和表的存储。这些文件包括日志文件、临时文件、数据库产生的临时表和维 护性文件，以及磁盘上的缓存文件等。

与待测系统集成的其他系统运行是否正常。比如第三方的目录访问服务器、消息 服务器等。

**步骤4.设置调优参数**

按照测试执行清单中的描述设置系统调优参数。也可以使用自动化的调优脚本对系统 进行调优，以避免不同的操作人员的操作区别引起的差异。

测试清单中的调优应该做到“适度”。所谓适度，就是不要因为片面追求调优带来的 好的性能而掩盖了应用程序代码本身的性能问题。况且某些通过参数调优提升性能的做法 也只是对具体的测试环境有效而并不一定适用于客户的实际生产环境。因此，在性能测试 过程中，参数调优应该只作为一种必要的辅助手段。可以考虑使用客户普遍使用或者适合 推荐给客户的调优方法。

**步骤5.编写或校验测试脚本和数据**

测试脚本需要尽可能完全模拟客户端访问服务器的行为。因此一般会先通过录制的方 式得到模拟的自动化脚本。然后根据测试需求定义输入数据、输出校验、分支比例控制、 并发用户数及思考时间等。

自动化测试工具和测试脚本自身对于健壮性和算法效率的要求也要高。如果测试工具 和脚本自身就有性能问题，就不能保证性能测试结果的可信性。

在每次进行性能测试执行之前，即使是把已经存在的脚本拿来重用，也要对自动化测

第6章 大促带来的灾难：究竟能支撑多少访问量

试脚本进行校验。因为每次待测系统的变化都可能引起脚本相应的变化。校验的过程一般 是通过单用户执行一遍所有的功能分支，检验是否每一步的输入/输出都正确。

校验脚本的过程可能会对系统的状态有一些影响，以电子商务应用为例，可能会在数 据库中添加了一些订单，或者添加了一些促销信息等。因此在正式执行测试之前，最好恢 复脚本校验之前的数据库备份，并重新启动系统。

测试脚本的执行需要配合测试所需要的数据。测试数据一般在测试用例中进行详细的 定义，通过数据生成工具或其他方法提前生成完毕，并在脚本录制前导入到数据库中。功 能验证或脚本的调试过程中，数据也可能会做相应的调整。

**步骤6.预热执行及逐步增加并发用户负载**

在检查完测试客户端和待测服务器环境，以及校验完测试脚本之后，在开始正式执行 性能测试之前，还需要一个预热执行 (Warm up) 过程。系统刚启动时，有些JSP 程序还 没有编译，应用服务器的包括动态高速缓存在内的各种缓存还没有填充，各种等待队列可 能还没有建立，数据库服务器的缓存机制还没有发挥作用。如果骤然让服务器应对大并发 用户量的测试(尤其是压力测试),系统很可能会出现大量的错误甚至崩溃。即使不崩溃， 测出的结果也不能体现正常运行中的性能。预热执行还可以弥补自动测试数据生成工具的 不足。比如在购物测试场景中，需要在测试数据中包含一定数量的订单，而且要求这些订 单中有一定比例是缺货订单(Back Order)。缺货订单的数据关联比较复杂。用数据生成工 具产生比较复杂，所以改为用测试脚本运行一段时间(选择特定的执行路径),生成满足 要求的缺货订单。这就成为购物场景测试执行清单的一个重要步骤。

另外在正式执行测试用例时，对于大并发用户数下的压力测试，一般不采用将大量并 发用户同时开始的方法(除非针对特殊用例),而是采用从少到多地逐步增加的方式(Ramp up) 。 比如，在1分钟之内将并发用户从0增加到5000。

**步骤7.配置监视工具**

预热执行之后，可以开始正式执行。但是为了在测试结束后收集到完整而正确的测试 结果，还需要一些配置工作。比如在测试正式开始前，清空所有的日志文件，开启需要的 监视工具等。

对于需要安装的监视工具，应该提前安装好(可作为准备测试服务器环境的一部分)。 对于已经安装好的监视工具，测试执行开始前还要再次检查，确保其能正常工作。各种类

型的性能测试对监视信息的要求不同，而且有些性能监视工具还会消耗一定的系统资源。 所以在测试执行清单中应该对各种类型的测试需要对监视工具做何种配置有明确的规定。 通过自动工具完成配置监视工具的工作能在一定程度上避免操作差异。我们会在后面详细 讨论各种监视工具的使用。

**步骤8** **.执行测试**

“万事俱备之后，终于可以开始正式执行测试了。”史静笑着说。

“哇，真不易。终于开始了。 一个 case 动辄跑好几个小时，测试用例跑起来之后，是 不是测试人员就可以泡一杯咖啡去聊天啦?”小艾笑道。

“想得美啊，”史静说，“如果你喝完咖啡聊完天回来发现用例跑失败了，你怎么办?” “那我能干什么?”

“测试人员千万不要认为单击了测试工具的执行按钮之后就可以关上计算机屏幕去休 息了。在测试运行过程中，尤其是需要长时间运行的可靠性测试过程中，测试人员要定期 通过监视工具检查服务器和测试客户端的运行状态。对于产品测试而言，待测系统往往 是含有性能问题的未发布系统，尽早发现已经出现的性能问题更为重要。 一些比较严重 的性能问题出现之后，如果自动化测试脚本还在继续运行，可能会导致一些后继的问题， 这会对最后性能问题的定位分析造成很多干扰，想找到最初的问题往往需要花费更多的 时间。

另外，执行测试过程中需要注意监视工具的开销。在压力测试进行过程中，在服务器 端打开一个日志文件查看，或者将该日志文件通过FTP 传输到另外一台机器上查看，都有 可能带来额外的系统开销，影响测试结果的准确性。因此需要慎重选择监视的方式。

**步骤9.收集、分析测试结果并整理测试报告**

测试执行结束就要及时按照测试执行清单规定的步骤收集测试结果，并分析判断测 试用例是否通过。为保证没有遗漏，也可以通过自动化的工具或者脚本来收集结果。不过 使用自动化脚本情况需要注意，测试人员往往因为结果自动化收集而不去人工检查测试结 果的正确性和各类日志中是否存在问题。因此需要提醒测试人员注意，即使使用自动化工 具收集结果，也需要仔细检查各类结果和日志。

另外一点需要注意的是，不论测试是否通过，都应该按照测试执行清单的要求收集各

第6章 大促带来的灾难：究竟能支撑多少访问量

种结果信息。测试人员的一种常见错误，是看到测试工具的报告不满足通过标准就立刻清 除日志恢复环境，再执行一次测试，并祈祷再执行一次能够顺利通过。其实对于产品测试 而言，测试的目的是发现问题而不是仅仅为了让测试用例顺利通过。收集测试失败情况下 的日志反而对于分析解决问题至关重要。

收集分析完测试结果之后，测试人员需要把测试结果整理成正式的文档，在测试结果 管理系统中提交。如果测试失败，需要按照缺陷管理流程和规范提交缺陷报告，将测试脚 本和数据上传到脚本/数据版本管理服务器。如果测试在未来还有可能重新执行(比如需 要在缺陷解决之后进行验证),则需要备份一些较难搭建的测试环境。

**6.3** **像FBI那样缜密——全面监视系统的一举一动**

小艾终于开始正式做性能测试了。虽然能够按照性能测试规范和测试执行清单来执行 测试用例，但小艾对于形形色色的监视工具和报告依然不是十分了解。他不明白每个监视 工具的具体作用，为什么要开这些监视，以及如何从这些监视报告中分析和发现性能问题。 于是他又找到史静，希望能够详细了解测试执行清单中每种性能监视工具的使用方法，以 及如何分析这些报告。

性能监视 (Performance Monitoring), 就是通过各种监视工具了解当前应用系统运 行的状况，发现已经存在的性能问题或者发生性能问题的潜在危险。当通过性能监视发 现已经出现性能问题时，则可以通过系统的各种日志发现引起问题的根源。

对测试系统和生产系统的性能监视也要区别对待。测试系统的性能监视一般由性能 测试人员或开发人员进行。测试系统的系统运行时间短压力大，而出现性能问题的可能 性比较高。测试人员可以在测试过程中开启各种监视工具而不必太在意其对应用系统运 行的影响，而生产系统的性能监视一般由系统维护人员进行。生产系统系统长时间运行 (常常是7×24小时运行),出现性能问题的可能性比较小，但是一旦出现问题，产生的 影响很大。所以生产系统的性能监视重点是在不影响生产系统运行的情况下，对出现的 性能问题或可能出现问题的不良趋势尽早报警并采取相应措施。



1 引自《构建高性能WebSphere企业级应用》。

尽管存在这样的差别，然而在两种系统上进行性能监视的目的都是为了尽早地发现性 能问题。在性能测试进行过程中也需要及时检查各种监视信息，而不要等到测试运行完毕 再检查各种监视工具的结果。

但是这并不意味着每次执行测试用例时要把各种监视工具都打开。性能监视不可滥 用，因为性能监视工具本身作为运行的软件也会占用系统资源，甚至可能与应用系统竞争 资源。正确地选择适合的监视工具并设置合适的监视级别，才能既保证不影响应用系统的 运转，又能为分析性能问题提供监视数据。此外，需要对使用的监视工具产生的日志文件 或数据文件的大小有一个正确的认识，以便留出足够的磁盘空间。

性能监视一般采取从前到后的监视顺序。首先从终端客户能够感受到的性能运行指 标开始：如错误率、响应时间和吞吐率。这些指标一般来自自动化测试工具的用例脚本 执行的结果。如果发现这些性能指标出现异常，则可以按照Web 服务器、应用服务器到 数据库服务器的顺序检查这些服务器工作是否正常。当然，正常和异常都是相对的， 一 般性能测试人员只有拥有长期的监视经验才能总结出什么是正常的状态，偏离正常状态 多少算异常。

监视系统的性能运行指标，在测试环境中一般通过测试工具进行。许多性能测试工具 都提供实时的运行指标显示，测试人员可以通过检查这些实时的指标，做出是否存在性能 问题的判断，而不需要等到预定的测试执行时间结束。史静说，“这就是为什么我说最好 不要在用例运行的时候离开的原因所在，时刻监视着系统有助于你在第一时间发现问题出 现在什么地方。找到问题出现的时间点往往很重要，因为系统日志很多时候是先从引起问 题的根源处开始记录出错的。如果你错过了这个时间点，就要在事后花时间一点一点去追 溯引起问题的根源在哪里。”

多数监视工具都提供监视级别以控制监视工具的开销，或者说消耗系统资源的多少。 监视级别代表了监视目标的多少，显然，监视目标越多，监视工具的工作负载就越重，消 耗的系统资源也越多。绝大多数监视工具，例如 WebSphere 应用服务器的监视工具(Tivoli Performance Viewer)、数据库服务器的监视工具、ITCAM 等都提供多种监视级别或开关。 一般来说，监视级别越低，监视工具提供的信息越少。监视级别越高，提供的信息越多。 但是较高的监视级别意味着很高的资源开销，所以一般不推荐在系统高负载运行的同时开 启最高的监视级别，通常只在需要确定特定的问题时才对特定的系统部分开启最高的监视 级别。举个例子，一般我们跑正常的用例，就把应用日志级别开到打印错误或者警告的级 别。如果出现错误，再只针对错误分支打开所有信息甚至trace 级别，以详细追溯错误发 生的原因。与监视级别相关的一个设置是监视工具的监视频率，或者叫采样率 (Sampling

第6章 大促带来的灾难：究竟能支撑多少访问量

Rate) 。 监视工具一般采用给系统照快照的方式获取系统的工作状态。采样率就决定了监 视工具隔多长时间照一次快照。显然，采样率越高，监视工具的开销越高。 一般情况下并 不需要设置很高的采样率，半分钟或1分钟一次就够了。只有在已经发现系统存在性能问 题时才需要提高采样率。

一般来说，基于应用服务器的系统一般需要做以下几个级别的监视：操作系统级别、 数据库级别、应用服务器级别。

**6.3.1** **操作系统的监视**

操作系统监视工具监视的是各种系统资源的使用状况。系统资源的使用状况间接反 映了各服务器程序的运行状况。以电子商务应用的性能监视为例，主要监视以下系统资源：

CPU 占用情况；

内存占用情况；

I/O 使用情况。

CPU 即中央处理器。软件程序的运行都需要占用CPU 资源。CPU 运行有4种状态： System/User/Wait/Idle 。 以 User 表示的适当的CPU 占用率是服务器是否正常工作的重要 标志。System 占用率非常高则说明系统进程的工作比较繁忙。如果 Wait 太高，则说明 系统经常处于等待(以等待I/O 为主)状态，通常需要进一步查看是什么原因导致了系 统等待。

内存使用情况主要指系统内存的使用和对 Java 虚拟机内存的使用。内存使用的监 视，尤其是对 Java 虚拟机内存使用情况的监视在性能问题的发现中扮演着重要的角色。 因为很多Java EE 应用的性能问题是由Java 虚拟机内存使用不当引起的，比如内存泄漏、 内存碎片过多等。

I/O 即系统的输入/输出，包括应用程序对磁盘的读/写、对内存的读/写及网络传输 等。网络I/O 反映应用服务器接收请求和提供响应的数据量，磁盘I/O 则反映服务器读 取磁盘数据、写日志等行为的多少。当系统性能出现瓶颈时，查看系统的I/O 情况也是 确定问题的重要手段。

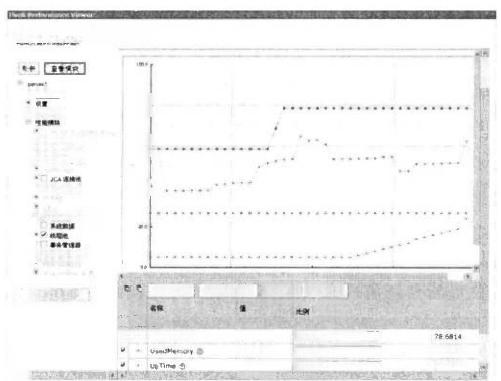
常用的操作系统监视工具有Nmon,top,topas,typeperf,zVM 等，需要根据不同的 操作系统和不同的监视需求选择使用。

**6.3.2** **应用服务器的监视**

如果通过操作系统资源监视发现应用服务器运行可能存在问题，就可以启动针对应用 服务器的监视工具。应用服务器一般本身都集成了监视工具，比如 WebSphere 应用服务器 就集成了Tivoli Performance Viewer和动态缓存监视器等。与第三方监视工具相比，这些 工具提供的功能并不弱。

WebSphere 应用服务器为性能监视提供了性能监控基础结构 (Performance Monitor Infrastructure,PMI)[0]。该结构启动之后，其他第三方厂商的性能监视工具就能够通过 SOAP 协议获取 WebSphere 应用服务器的实时性能参数。 PMI 收集到的数据可以通过 WebSphere 应用服务器内置的Tivoli Performance Viewer(Tivoli性能查看器或简称TPV) 来进行分析和统计。

如图6-2所示就是将JVM 运行时(包括JVM 内存使用状况)和线程池两个监视模块 打开后的监视结果。显示区会自动随时间刷新以动态显示曲线的变化情况。



Tea Borfomonce bewer >srven

肌里

鹰警极

*En*

\*ec n

8遵行时

+sm 话警道器

w 应阳程年

清空理冲区

H

M 城行

755814.0

497105.0

26970

1.DE-4

1.0E-4

1.0E-4

49.7105

269336

复济造痛平南清面

显示围洼

复盐为琴

·keepSize5

此道

琶

闭

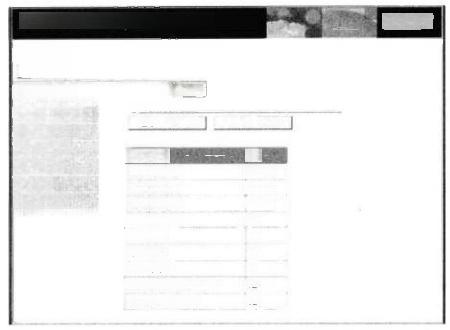
图 6 - 2 JVM 运 行 时 监 视 结 果

另一个比较重要的应用服务器级别的监视是动态高速缓存监视。以 WebSphere 应用服 务器为例，动态高速缓存对提高 WebSphere 应用系统性能起重要作用。在测试环境中监视 动 态 高 速 缓 存 ， 一 般 推 荐 使 用WebSphere 应 用 服 务 器 内 置 的 高 速 缓 存 监 视 器(CacheMonitor)。

除了对动态高速缓存的各种详细信息外，高速缓存监视器还提供对动态高速缓存的一些手

第 6 章 大促带来的灾难：究竟能支撑多少访问量

工操作，比如清空缓存、使特定的缓存条目失效等。高速缓存监视器是一个独立可安装的 应用程序，但 WebSphere 应用服务器默认并不安装它，所以需要用的时候要先安装，如图 6- 3和图6-4所示。



SM.

离速额存实例 期新实例

确定 高这性存就计值息

清除高速额存

慈计始息

20000

9735

9759

27

0

0

3

是

是

高速族存大小 已便用的条目 高速鳜存命中 高速媛存不命中 LRU 逐出

显式脉去 缺省优先圾

启用 Servlet高速暖存 启用起用磁盘即献

baseCache

高连康存热计值

边度热过值息 蒸油内E 相关性丘识 概放载

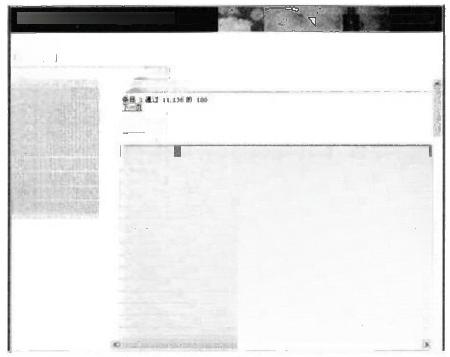
高流速存然吸

srye 速 缓 监 器

Wusphorf arcocamto

重置统计信息

图6-3 高速缓存监视器显示动态缓存使用情况



*ite* *z*

画道额存实例 复E

满 空

2音画清有六W

清站设盘

ZmbauMsatacdscom fDInacineusnta ICALDC15nsLn Dazanlas/CateuC

intDB05ist2s:cRntamoc.stS ECACEen5tnnLcass.Daaon/CLoGv

(ubagn/stan/Lnmunaeu staECADanbtnatLGn:uardu:CaRon6

Bsebesmcsstoes/com.DmLommeu sthECActonsenetcn.pacnoe/Cregon

dtbassssatansLOLuamoeiustats ECActao5eLcns DazadoniCrac6 20nsuocI connec s ECAIcn5cstcs pathnio=Lategg6

atoaagnCitma om anctctstnssECALenStont ca:DaaetoCrnQa6 tebac ano LancaustECAtonenst capaneoeGnei

asowatectcCPmDtoomnuttECActon5etcn.DSrfan.PosucDs

ac natassoLIonurer st4SECAtoncnstcnt atantasoduan nacstec n canecna ECAuntoisLCnDanunrodID

2000cststDmagectstsECAstan5eeLCsspabetasngutD

bacnc mue n cannace ataLCAtanLcat cna.baataEuguan

goaLECAConoieLcaspazintoePoducD wLEADnScosLdus pcnlanadua

oaseCache

五族开究计

天又绩热世媒筑黑

微 性 出

盘数

瓶 图

性关性死会 独

teoDDmcistoEAcemnonnete msaDmLateconn cee

图6-4 显示磁盘卸载内容

**6.3.3** **数据库的监视**

如果通过系统资源监视发现数据库服务器的资源使用出现异常，或通过应用服务器性 能监视发现和数据库的连接存在异常，就可以开启数据库服务器的性能监视器监视数据库

● 171·

管理系统的工作状态。通过数据库服务器的性能监视可以获取各种数据库内部资源(缓存、 数据库连接等)的使用情况，以及数据库服务的各种操作(比如排序)的执行情况，死锁 出现的状况或者有异常的SQL 语句。

这里我们以DB2 为例看看有哪些监视器可以用。DB2 (这里介绍的主要是DB2 UDB 的工具)内置了很多性能监视的工具，比如DB2 快照监视器(Snapshot Monitor)、行为监 视器(Activity Monitor)、事件监视器(Event monitor)、健康监视器(Health Monitor) 等 。 另外还有一个图形化的健康中心。这些监视工具分别有不同的用途。其中事件监视器根据 监视事件的不同又可分为死锁监视器、语句监视器等。

以电子商务应用的 DB2 性能监视为例，较常用到的是快照监视器、死锁监视器和语 句监视器。

**1.DB2** **快照监视器**

DB2 提供的最常用的快照监视器，可以在某一时刻对 DB2 的状态做一个快照。这个 快照能够提供在这一时刻数据库管理系统中某个或者某些对象的动作状态信息。DB2 工作 的大多数关键信息都可以通过快照监视器获得。既可使用快照监视器获取 DB2 某一时刻 的工作状态，也可以间隔一段时间比较快照监视器获取的快照，发现某些工作状态的变化 趋势。例如，当性能测试中发现错误率比较高，并且对应在 WebSphere 应用服务器的日志 中出现了 SQL 异常时，就可以打开快照监视器的死锁开关，执行一次快照。从快照监视 器的输出记录中可以快速查看关于锁使用情况的各种信息。

前面提到过监视级别，DB2 快照监视器的监视级别是通过对各种监视对象的开关控制 的。监视特定对象之前，需要把要监视对象的开关打开。用户可以针对 DB2 数据库管理 系统进行监视，也可以针对数据库实例进行监视。监视对象包括缓冲池 (Buffer Pool)、锁 (Lock) 、 排序 (Sorting) 、SQL 语句 (SQL Statement)、表动作 (Table Activity)、时间戳 (Timestamp) 和工作单元 (Unit of Work) 等。这些监视对象的开关可以通过如下DB2 命 令查看和打开。

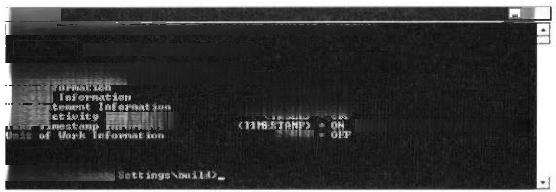
一个典型的过程如下。

首先查看数据库监视对象开关状态：

db2 GET MONITOR SWITCHES

返回结果如图6-5所示。

第 6 章 大促带来的灾难：究竟能支撑多少访问量



DB2QP 回区

D:Docunents and Settingsbulld>dh2 GET MONITOR SVITCHES

Monitor Recording Switches

|  |  |
| --- | --- |
| Suitch list for dh partition nunber  Buf fer Pool activity Infornation BUFFERPOOL OFF  artn SocK 熙  (STATABNE:PP  Infornat ion | 87/85/209714:14:58.142541 |
| <UOW |  |

D:Docunents and

图6-5 DB2 查看数据库监视对象开关状态示例

查看数据库管理系统监视对象开关状态的命令：

db2 GET DBM MONITOR SWITCHES

打开指定的数据库监视对象开关的命令：

db2 Update MONITOR SWITCHES USING switch-name ON/OFF

可以一次打开一个或多个开关。例如，针对数据库打开对缓冲池和锁的监视，可以通 过如下的命令：

db2 Update MONITOR SWITCHES USING LOCK ON BUFFERPOOL ON

执行该命令之后，系统返回：

DB20000I The UPDATE MONITOR SWITCHES command completed successfully.

再执行

db2 GET MONITOR SWITCHES

会发现缓冲池监视开关和锁的监视开关都已经打开了，如6-6所示。

DBZCP

□回区

D:\Docunents and Settings\buila>db2 GET MONITOR SVITCHES

Monitor Recording Suitches

3源200?

87/85/2897

unitofork Pnfontion

图6-6 同时打开缓冲池监视开关和锁监视开关后的输出示例 一般来讲，不管打开什么监视对象，推荐同时打开时间戳监视开关。 打开锁和缓冲池的开关之后，执行命令：

db2 GET SNAPSHOT

会发现快照监视器的输出信息中已经包含锁和缓冲池的信息，而其他未打开开关的对 象则只是显示 “Not Collected”, 如图6-7所示。

|  |  |  |
| --- | --- | --- |
| Total Privace Sort heap a11ocated | -0 |  |
| Total Shared Sort heap allocated | 0 | Sorting开关为关闭状态，所以日志里 显示为“Not Collected”。其他打开 |
| Shared sort heap high water mark | o | 的开关有统计信息显示。 |
| Total sorts | 2559 |  |
| Tota1 sort time (mn) | Not col1e | cted |
| Sort o verelous | 0 |  |
| Active sorts | 0 |  |
| Bufrer pool date 1oqical reads | 227 |  |
| Buftez poo1 data phys1cal reads | 0 |  |
| Buffer pool tenporary data 1ogical zeads | 0 |  |
| Bufter pool tenporory data physscal reads | -0 |  |
| Asynch ronous pool data page reads | 0 |  |
| Bufter poo1 data vrites | 0 |  |
| Asynchr onous pool data page vrites | ○ |  |
| Bufter pool index loqical reada | 142 |  |
| Burter pool index physical read | ·5 |  |
| Bufter pool tenporary index 1ogical reads 0 |  |  |

图6-7 没有打开开关的监视器在结果中显示Not Collected

通过快照监视器监视到的各种统计信息都非常重要。通过这些信息可以发现 DB2 的 工作状态异常，并可以判断是否需要调优 DB2 的某些配置参数。比如查看锁的相关统计 信息可以判断是否需要增加LOCKLIST 和 LOCKTIMEOUT 等参数；根据缓冲池的统计信 息可以判断是否需要增加 BUFFPAGE 的大小，根据排序的统计信息可以判断是否需要增 加 SORTHEAP 的大小等。

**2.DB2 死锁监视器**

数据库上发生死锁是并发访问时很常见的问题，尤其是压力测试用例这种比较极端的 用例。因此监视死锁也是对数据库监视很重要的一部分。这里会用到 DB2 的事件监视器 (Event monitor)。它获取的是当发生某个事件时的系统信息。这些事件包括一个事务的结 束、一个 SQL 语句的完成，以及检测到一个死锁等。

前面提到的 DB2 快照监视器只能提供一段时间内关于锁使用的基本信息，比如死锁

第6章 大促带来的灾难：究竟能支撑多少访问量

的数量、LOCKLIST 的使用情况等。如果想得到死锁比较详细的信息，必须使用死锁监视 器。死锁监视器是一种特殊的事件监视器，单纯的死锁监视器只能提供关于锁的信息，而 不能提供关于使用锁的SQL 语句的信息，所以在使用死锁监视器的时候通常联合SQL 语 句监视器共同使用。

要使用死锁监视器，首先要创建一个时间监视器来监视语句 (statements) 事件和死锁 (deadlocks) 事件。在需要监视的时间间隔之初执行以下命令：((dbname,username,password

表示数据库名、用户名和密码)

db2 "connect to dbname user username using password"

db2"create event monitor demoMon for statements,deadlocks with details write to 'path\monitor\demoMon'"

db2 "set event monitor demoMon state=1"

demoMon 是在这个例子中要创建的时间监视器的名字，path 是需要保存分析数据的 路径。with details是比较重要的命令行参数，如果没有指定这个参数，产生的事件日志就 会缺少很多信息，强烈推荐使用这个参数。该命令还可以有其他参数，比如MAXFILES、 MAXFILESIZE 等，用于控制产出的事件文件大小。如果确信有足够的磁盘空间，可以省 略这些参数。

在执行完上述的步骤后，不要关闭 DB2 命令窗口，因为在监视时间间隔结束后还要 在同一个命令窗口执行下一组命令：

db2 "set event monitor demoMon state=0"

db2evmon -path path\monitor\demoMon >deadlocks

如果没有在同一个命令窗口执行这两组命令，有可能收集不到死锁事件信息。在执行 这两组命令之间，如果数据库操作产生了一些死锁，DB2 就会在 path/monitor/demoMon 下生成一组扩展名为“.evt” 的文件，这些就是事件文件。

最后一条 db2evmon 命令将这些事件文件转换为容易读懂的关于死锁和语句信息的文 本信息，作者在命令中将其输出到一个名为 deadlocks 的文本文件中，接下来需要做的就 是对这个文件进行分析。

如图6-8和图6-9所示是一个死锁监视器收集到的结果。可以看到里面监视到了两个 应用程序引起的死锁。在同一个死锁分析日志文件中的这两段显示，对于同一个死锁ID, 应用A 所请求的共享锁在等待应用B 所持有的排他锁，而同时应用B 所请求的另一个排 他锁在等待应用 A 锁持有的共享锁，于是死锁发生。死锁的结果是最终应用 A 发生事务 回滚以释放其中的一个锁。

从菜鸟到测试架构师——个测试工程师的成长日记

w

З»ФеваШоскеа соппесртот

磷买 ana

Des41oek 1D: 1 村 布

Part eipant m0,

Partieipant sB.holsng the lock 1

арь!Xaз оэняевео,сбъс,отэтовто4а6

App1 seą nunberi 0382

Appl 1a ot eonnectian holting the 1eek:09858520.c998.07A72810414.

Seq no.ot eonnection hoidng t 10ck:0417

Loek wait start sim!11/28/200518157131.774475

Lock Nane ожобогозтраооосзтзоооопроэзз

Loek Atteubusea:060000000

elease Fiaga оячовоооов

Lock Coant 11 tN

НоАТСошпх 0

Current Mode нпвр

fessex dete10 1t 11/18/200519157140.204047

Tabieer 1eck 1840E :2sERs

no

Bcbenn ot loek wited s DN2NSTL

Tabiape of lox es.a+ iCE1

TYpe or loek:Ro nxn

sppl1 reqe on tock:N ·Share(nt fext Key hare 中代.04等

Node lonx eeeured on

Ьосх оЬуесг паъет ЗЭТ

Applicnon Hand1e:079

pes1ocked Ststenest;

TYpe :Bynsmie

Operat1oni Fetch

3ec510 :19 , 范

Creatpe 1NULLIP

Psekng157558200

Счепот ллд,сираигооето

Cucnor wa blocking:FALSE

Taxt зесттьвтатпмвводлон,отсовииниикт,н тлонвтатюпрате,

**图** **6** **-** **8** **DB2 死** **锁** **之** **A 等** **待** **B 的** **锁**

3A- 地a

7ВемЦосКва СовшесИАевняя

T4iook 10: 1 4少eA 体

Fartieipant so. 1

Fasticipant ss.hotdng the loek:2 hau

Appi 14:G9850820,CP9T.07A728104142/

АррА еа шшанє з очТТ

Appl ta ot connection hoiaing the tocx:6985参 2 0 .C9D.073708104436

S84.n.ot gonectios hoiaing t loek;0362

Loek nit start tine:11/20/200518157:31.774400

Loek Maon 10x0002014roo0002590000000052

Lock Aribste :QD0000000

Feiease Fisg 040000000

妒的味点

Cucr 住nt tode none

ted1se\*sdeteetion tet11/6/z00518:57+0,20448 描 一

Tab1e ot loek sted on an

Schenn ot Lock v1ted c кева тнета A

Tblergoce ot loek \*ited o+ ACK1

Type ot leek:Re

Mosde ot ioek; 描 ·sare (ann Mest Eey Sharej 满 e, 对 ?

Mude spplicst4on regueet on 1oek!x EMe10+ive

Node 1sek eesured on:0

Lock object noet 857

Applicat1on Man4te:856

De4inck 读 S5atemnt1

Typ\* :Dynie

Орпгоє ъопг ЕкосшЄ а

Secr1an 117 8 7 四 转

Creator NULLID

Fack\*gt:5Yas200

Согвосто.. спрешоосат

curset war bloekingi FALSK

Text пФъате пенвевАдтэтатортсоштея чмкьшвклаевшЕ

3)Deadieek fvent … 他的是489 peadloek 1D1 1 1.73

Naber ot applicat iont deadockedt 4

Bea4lock dete1Dn tLae 11/28/200510:57:40,202928

Po11ed back Appi partieipant no12

o11ed back pp114;G950820.C90c,072708104436

o11ed back Appl ae nueier::017e

**图** **6** **-** **9** **DB2 死** **锁** **之** **B 等** **待** **A 的** **锁**

第6章 大促带来的灾难：究竟能支撑多少访问量

**3.DB2** **语句监视器**

除了和死锁监视器联合使用外，语句监视器也很常用。有时候并没有出现死锁问题， 而仅仅是怀疑某些操作使用的 SQL 语句不合理，就可以单独使用语句监视器监视特定操 作使用的 SQL 语句。

使用的方法就是将 create event monitor 的参数从 for statements,deadlocks 变为 for statements, 如下所示。

在需要监视的操作发生前创建事件监视器并开始事件监视：

db2 create event monitor name for statements write to file 'path'

db2 set event monitor stmon state=1

在执行要监视的操作后停止事件监视并将输出信息导出到一个文本文件：

db2 set event monitor stmon state=0

db2evmon -path path>outputfile

如果需要多次使用语句监视器，需要在两次监视之间重置监视器的状态：

db2 reset monitor for database databasename

在获取到SQL 语句的详细信息后，可以通过以下几个方面进行考察：

SQL 执行时间 (Exec Time);

用户状态CPU 的时间 (User CPU);

系统状态CPU 的时间 (System CPU);

读取的行数 (Rows read);

写的行数 (Rows written);

排序的使用情况 (Sorts);

□ 缓冲池的使用情况 (Bufferpool)。



比如 SQL 语句的执行时间相对过长说明有可能该语句编写不合理，读取的行数很大 而且接近于数据表的总行数说明可能使用了表扫描等。

图6-10为语句监视器的输出实例。

从菜鸟到测试架构师——一个测试工程师的成长日记

409)statesent tvent..

App1 Bandte:1373

Appi ta:P7957356.D9C4.044425183000

Appi Seu nunbe!0057

Wecord iz the resuIE or s ttush:TALSE

SELECT r1,ATTRISUT\_ 1D,T1,LANGUAGF\_ 1D,T1,ENUM,T1.NAKE,

T1.DESCRIPT1CN,T1.DISPLATLABEL,T1.GLC3SARY\_ID,T1.13STOPZSPECIrIc.

T1.LASTUPDATE,T1.CCNTENTSOITPCK,T1.USAGE,T1.YIELD1,T1,FIELD2,

T1.OPTCOUNTER FRCn ATTBIBUTE T1 IHEFE T1.DESGPIPTICN=)

Start Tine:62/23/200712:50:04.372305

Nuaker ot Agents ereated;1

Uaer CPU:0.000000 second

syaten CPU;0.000c00 seconds

F 楼eh Count!1

sorta;0

九 起 讲 ( 前mmdi

Sort overtiows:0

Fows read:3202

Rows written:0

Internal rowa deleted:0

Internal rowe updated:0

Internal rowa inaerted:0

Bufferpool data logien!rnada:103

Bufrerpool data phyaical reads;0

Bufferpoo1 teporary daa log1cal reada:0

Bufrerpoo1 tesporary dacs physieml read:0

Bufrerpoot index 1ogscal read:0

Burferpoo1 index physical ceads:0

Bufterpook seaporary index 1ogical resdai 0

Bufferpool teaporary index phyical readz:0

32LCA;

图6-10 语句监视器输出实例

从这个的例子中可以发现这条 SQL 在执行期间进行了大量的记录的读取(Rows read: 3202),非常有可能进行了表扫描，可以通过查看表的大小加以确认。进一步需要对数据 库的表结构进行分析，找出可能的原因，例如需要在字段Description上增加索引。

**4.Oracle 监视**

史静开始继续介绍Oracle 的监视。“我比较常用的Oracle 性能监视工具是Oracle 内 置 的工具AWR(Automatic Workload Repository) 。Oracle的 AWR 工具提供的信息有些类似 于DB2 快照监视器提供的信息。”

Oracle AWR 的作用主要有两点：第一，在Oracle 数据库实例出现问题时用于发现和 解决问题；第二，它可以提供 SQL 语句处理方面的统计结果，通过这些统计结果可以得 知有哪些 SQL 语句使用的频率较高，处理所用的时间较长或者执行效率较低，从而可以 对这些 SQL 语句进行重点优化。

AWR 会默认每小时抓一次快照，但是也可以通过手动的方式抓一次快照。抓一次快 照的命令如下：

SQL >exec DBMS\_WORKLOAD\_REPOSITORY.CREATE\_SNAPSHOT();

第6章 大促带来的灾难：究竟能支撑多少访问量

生成AWR 报告的命令如下：

@SORACLE\_HOME/rdbms/admin/awrrpt.sql

命令会提示选择生成 HTML 或者文本方式的报告(如果条件允许，比较建议生成 HTML 格式。因为 HTML 格式会比较清晰方便阅读和分析):

Enter value for report\_type:text

为了更便于进行性能分析，可以在性能测试或者性能问题出现前后进行多次AWR 的 抓取。每次抓取时系统会指定一个 ID 并把统计数据存在数据库表中，之后用户可以指定 任何两个 ID 之间进行性能数据的输出，系统就会输出在这两个时间点之间所统计的所有 数 据 。

选择针对第几天的数据进行收集：

Specify the number of days for which you want to list snapshot Ids.

Enter value for num\_days:2

然后给出要统计哪两个点之间的信息，比如：

Enter value for begin\_snap:150

Enter value for end\_snap:160

然后系统会提示输入/输出报告的文件名。指定文件名之后按回车键，系统就会将这 两个监视点之间的统计信息输出到指定文件中。



Enter value for report\_name:

Using the report name awrrpt\_ 1\_ 150\_ 160

在输出报告中可以查看针对这个数据库实例的性能总结，包括状态、比例和百分比。 之后的每个段落提供了各个部分的性能统计的细节，包括 SQL 统计、回滚，队列、缓冲 器等待和锁信息等。因此，在进行问题定位时往往应该先查看总结页，从总结页中发现哪 些关键区域有问题，然后在后面的统计细节中进行进一步的分析。SQL statics部分按照SQL 的各种资源占用进行了排序，可以非常方便地查看最耗 CPU 或者读取行数最多的是哪些 SQL, 从而辅助问题定位，或者做有目的的优化。

**6.4大海捞针也要讲究策略——性能问题的分析定位**

小艾在了解了性能测试用例的设计、执行和过程中的监视之后，顺利地做了不少性能

测试，直到有一天，一个测试用例没有达到通过标准。小艾仔细检查了自己的测试环境、 脚本和数据，认为不是自己的测试本身出的问题。可是开发人员在了解了测试用例和失败 的测试报告之后，对于是哪里出了问题也是一头雾水。因为小艾这个测试用例实在太复杂， 整合了好几个部分的分支，谁也说不清楚到底是哪部分代码出现了性能问题。于是小艾又 找到了史静，请教性能问题分析定位的方法。

“我感觉对于比较复杂的用例，性能问题的分析定位简直像是大海捞针!”小艾显然被 这个问题折磨得头昏脑胀了。

史静说：“性能问题一旦出现，一般来说的确很难立刻知道问题出在哪里。性能测试 的用例为了尽量模拟用户的实际访问情况，往往是把多种用户类型、多个功能分支放在一 起执行。从前面的讨论你也可以看到，性能测试的用例很复杂，像可靠性测试除了正常的 用户分支之外还会有很多维护性的操作。所以一旦出现了性能问题，要找到问题的根源通 常不是一件容易的事，一些指导性的策略能减少盲目性。”

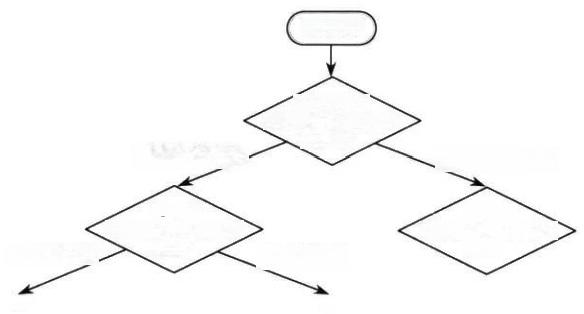
**6.4.1** **磨刀不误砍柴功——了解性能问题定位的一般策略**

类似于 WebSphere 这种网络应用服务，因为涉及 Web 服务器、应用服务器、数据库 服务器、操作系统等多个层次，性能问题往往比较复杂。因此这类性能问题的确定过程一 般有两种策略：自顶向下分析(Top-Down Analysis)和自底向上分析(Bottom-Up Analysis)。

所谓自顶向下分析，是指将应用系统划分为若干部分，采用排除法或归谬法首先确定 问题出现的位置(是哪个部分的问题),然后再在这个部分寻求问题的原因和解决方案。 在对应用系统划分部分的时候，通常采取多级划分的办法，比如首先可以把系统划分为应 用服务器和数据库服务器，应用服务器又可划分为 Web 容器和 EJB 容器等。自顶向下分 析在确定问题属于哪一部分的时候，往往采取一种逐渐细分的方法，从比较粗的定位到非 常细的功能模块划分，这就是自顶向下分析得名的由来。另外自顶向下分析重视问题产生 的条件，即出现问题之前和出现问题之后有什么不同，通过对产生问题条件的归谬确定存 在问题的部分。自顶向下分析方法如图6-11所示。

与自顶向下分析方法相反，自底向上分析方法注重问题的现象。通过考察性能问题现 象的各种细节与以往出现问题的现象进行对比，确定性能问题的类型。自底向上分析方法 往往直奔问题的日志或其他细节，比如报告的是内存泄漏问题，那么就直接进行HeapDump 分析，等等。自底向上分析方法如图6-12所示。

第 6 章 大促带来的灾难：究竟能支撑多少访问量



问题现象

是应用服务

器还是数据库服务器

的问题?

应用服务器问题 数据库服务器问题

是Web

容器还是EJB 容器的

问题?

EJB 容器问题

…

.

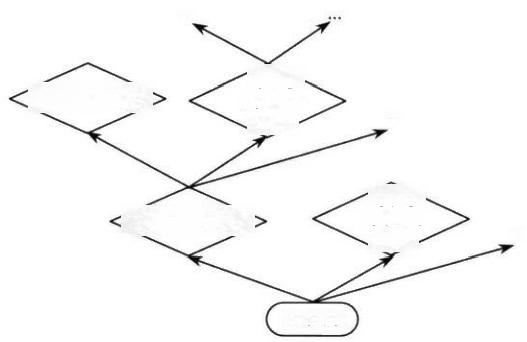
是配置

参数还是数据内容的

问题?

Web 容器问题

图6-11 自顶向下分析方法



.

是不是

SQL 语句 问题?

是不是 内存泄漏 问题?

问题现象

是不是死锁问题?

是不是索引问题?

…

.

图6-12 自底向上分析方法

采用自底向上分析方法的一般是处理性能问题的专家，他们能够根据一些蛛丝马迹迅 速找出问题的根源(Root Cause)。但是这对进行问题确定的人员的经验和判断能力要求非 常高。如果一开始就出现了判断错误，则很可能浪费了很大精力却仍然与问题的正确解决 偏离很远。

在报告性能问题的信息不太充分、问题的现象不是很典型或者进行问题确定的人员缺 乏处理相关问题的经验的情况下，则推荐采用自顶向下的分析方法。自顶向下分析方法相 对比较系统而且容易掌握。尽管找到问题根源的速度相对比较慢，但自顶向下分析的每一 步都不会是无用功。

从菜鸟到测试架构师———个测试工程师的成长日记

当然，在实际问题确定过程中，这两种分析策略并不是相互矛盾的。完全可以采取一 种中间汇聚的策略：先采取自顶向下的策略把问题出现的部分确定到一个比较小的范围内， 当确定问题属于熟悉的领域时，就可以采取自底向上分析方法快速找到问题的真正根源。

找到性能问题出现的原因，就可以确定问题的解决方案了。并不是所有的性能问题都 需要修改程序代码解决，有些可以通过调优性能参数解决，有些则必须通过修改软硬件配 置解决。在这里想多说几句，是关于设计测试计划和用例的。既然我们知道性能问题的定 位比较困难，在设计测试计划和用例的时候，就需要考虑这一点。对于一些新的功能，如 果想要验证它的性能，最好能让用例的设计相对独立一些。换句话说，就是避免多个可能 引起性能问题的新功能之间互相影响。在各自确定好各个新功能的性能都没有问题之后， 再进行多种功能混合在一起的更接近客户环境的压力或者可靠性测试，能在很大程度上避 免性能问题定位的盲目性。

听完史静对方法论内容的介绍之后，小艾觉得收获很多。但是毕竟接触的性能问题还 不多，觉得缺少一些直观的感受。史静说：“一开始觉得没有头绪很正常。接下来我们看 几种很典型且常见的性能问题，方便你累积经验时有所侧重。”

**6.4.2** **越忙越乱——高并发访问量下的出错率**

如果页面访问在高并发情况下出现错误，而且这些错误在单用户或者少量并发用户情 况下并不发生，那么可以怀疑出现了并发访问问题。引起并发问题的原因有很多，大多数 是由多个并发线程抢夺资源引起的。以一个 Java EE应用为例，这些资源可能包括 Web 服 务器的允许请求数，应用服务器上的Servlet Engine线程，数据库资源连接池里的连接， 数据库服务器上的连接池里的连接，数据库的锁资源等。当申请使用这些资源的并发用户 多于可用的资源数时，就会产生等待队列。 一般有两种情况会导致并发错误， 一种是等待 队列里的事务会出现超时或者回滚，从而导致访问出错；另一种是竞争锁资源时导致了 死锁。

对于等待超时这类问题的分析和解决方法如下。首先通过前面提到的监视工具得到各 类资源使用的监视结果并分析实际使用的资源数是否小于系统设置的最大资源数。如果达 到了最大允许资源数，则试着调高最大允许资源数。需要注意的是，有不少的参数是相关



1 本节主要内容引自《构建高性能 WebSphere企业级应用》一书。

第 6 章 大促带来的灾难：究竟能支撑多少访问量

的，所以每次调整需要对这一系列的参数都做相应的调整。比如， Web 服务器的最大客户 端 数 (MaxClient) 应该设置为使用该 Web 服务器能创建的最大子进程数 (ServerLimit)

乘以每个子进程的线程数(ThreadPerChild) 。 如果屡次调高之后仍然不能满足请求的数目， 且出错率仍不见好转，则需要考虑该请求数目是否合理，或者在更靠前的入口处调低入口 允许的请求数目。

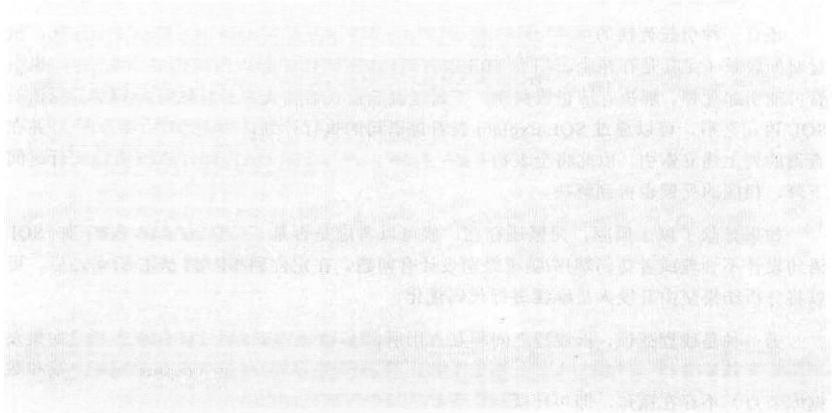
死锁问题 一般出现在多个进程互相持有对方需要的锁而得不到自己所需要的锁的情 况下。我们一般会遇到数据库死锁和进程死锁两种死锁。数据死锁一般会在应用进程的日 志中抛出异常，比如：

"[IBM][CLI Driver][DB2/6000]SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code"2". SQLSTATE=40001

然后检查DB2 的 snapshot 结果，会发现Deadlocks detected 不为零。

Deadlocks detected =25

|  |  |
| --- | --- |
| 这就可以确认发生了死锁问题。然后可以使用前面提到的Deadlock | Monitor来查看详 |
| 细的死锁信息，从而得到死锁发生在那些表上、所得类型、由哪些 SQL | 语句引起等详细 |
| 的信息。 |  |



Deadlock detection time:04-19-200404:30:04.896812

Table of lock waited on :ORDERS (被锁住的表)

Schema of lock waited on :DB2INS

Tablespace of lock waited on :USERSPACE1

Type of lock:Row

Mode of lock:X

Mode application requested on lock:NS

Deadlocked Statement:

Type :Dynamic

Operation:Fetch

Section :14

Creator :NULLID

Package :SYSSN300

Cursor :SQL\_CURSN300C14

Cursor was blocking:FALSE

Text : SELECT T1.TOTALTAX, T1.TOTALSHIPPING, T1.LOCKED,

T1.TOTALTAXSHIPPING,T1.STATUS,T1.FIELD2, … …

发现了具体的死锁信息之后，就可以采用自底向上的方式来尝试解决。首先可以尝试 是否是调优参数设置不当引起的。比如在DB2 里尝试执行 runstats 和 reorg 。 性能测试导

从菜鸟到测试架构师——一个测试工程师的成长日记

入的数据或者运行一段时间之后产生的数据有可能没有被 DB2 的数据库管理程序的优化 器统计在内，从而导致优化器不能针对当前的数据生成高效率的 SQL 访问计划 (Access Plan) 。重新执行runstats 可以针对数据库当前的状态收集统计信息，从而在执行时生成更 优的访问计划。而Reorg 可以按照最新的状态重新组织表结构从而降低访问开销。这两点 很多时候就能有效地减少死锁。

另一类调优是调整DB2 的 Locklist 和 MAXLOCKS 参数。首先查看DB2 快照监视器 的日志里有没有锁升级，当一个应用程序占用的锁列表 (LOCKLIST) 的百分比达到 MAXLOCKS 所设定的值时，数据库管理器对应用程序所挂起的锁执行从行到表的锁升级 以减少锁列表的占用数量。同样，当 LOCKLIST 资源将要耗尽的时候，数据库管理系统 会识别一个占据绝大多数行锁资源的表连接并将这些行锁定升级为一个对该表的表锁来 释放 LOCKLIST 资源。相对于一般使用的行锁，表锁对整个表加锁显然会大大降低数据 库共享对象的并发性。高并发下死锁的概率随之升高。因此可以考虑适当地把LOCKLIST 和MAXLOCKS 调高到适合该应用的数量。调整后查看是否能有效减少死锁。

从DB2 的快照输出中发现锁升级，如下所示：

Lock escalations =2 (锁升级次数)

Exclusive lock escalations =1(X锁升级次数)

还有一种引起死锁的可能原因是缺少索引。当某些 SQL 语句的执行效率比较低，执 行时间较长(尤其是存在全表扫描TBSCAN) 时，如果同时有大量用户访问这个表，也很 有可能引起死锁。解决方法也很简单，在通过快照监视器和死锁监视器定位到产生死锁的 SQL 语句之后，可以通过SQL Explain查看该语句的执行计划，找到最耗时的查询，并在 查询的列上建立索引。由此将全表扫描变成索引扫描，从而提高查询效率。SQL 执行时间 下降，相应的死锁也得到解决。

如果排除了以上原因，死锁还存在，就可以考虑是否是代码本身的问题。比如 SQL 语句设计不合理或者访问顺序/隔离级别设计有问题。在定位到相应的 SQL 语句之后，可 以将分析结果交由开发人员继续进行代码优化。

另一种是线程死锁，指线程之间相互占用所需资源而不释放导致的死锁。 一般如果发 现应用服务器端的 CPU 随着压力的升高提升不上去，但其他节点(比如测试客户端和数 据库节点)不存在瓶颈，即可怀疑是不是应用服务器节点出现了线程死锁。

如果怀疑线程死锁，可以在怀疑线程死锁时产生一个javacore 文件(线程转储文件)。 例如，在 UNIX 的操作系统上可以使用kill 命令来产生javacore 文件。

第 6 章 大促带来的灾难：究竟能支撑多少访问量

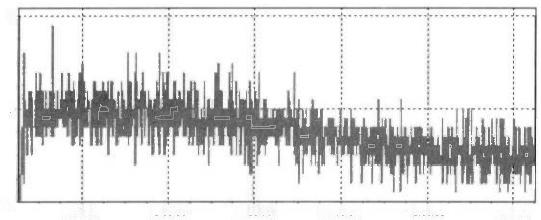
Kill-3<PID>

javacore 文件可以通过一些工具帮助分析，比如IBM 的 IBM Thread and Monitor Dump Analyzer for Java。文件中包含有应用服务的环境信息、线程信息、堆栈信息、锁信息和内 存信息。通过分析线程间资源占用情况可以分析出有哪些线程存在相互占用不释放资源的 问题。需要注意的是，由于线程间的调用关系黑盒测试人员不是非常熟悉，往往此类的分 析需要与开发人员合作完成，这里不再详细讨论。

**6.4.3** **越跑越慢——吞吐量下降问题**

性能下降 (Performance Degradation) 主要是指在负载没有变化的情况下，系统的性 能指标随时间而逐渐下降。

在性能测试中发现或重现性能下降问题主要是通过可靠性测试用例，即通过较长的执 行时间发现各种性能指标的变化趋势。当然，与发现内存使用问题一样，也可以采用提高 系统负载的方法(比如长时间压力测试)加快性能下降问题现象的出现。如图6-13 和图 6-14所示即为某3小时压力测试过程中测试工具提供的吞吐率和响应时间变化曲线，从中 可以看出明显的性能下降趋势。



2

1

0

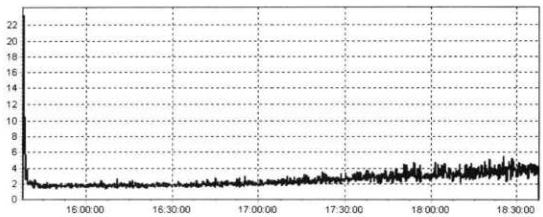
16.00:00 16.30.00 17:00:00 17.30:00 18.00.00 18.30:00

图6-13 吞吐率下降曲线示例

性能下降问题的现象虽然都差不多，但是引起性能下降问题的原因却很复杂。排除掉 服务器本身的问题和测试用例/测试脚本设计的问题，应用服务的性能下降问题由以下单 个或多个原因混合导致：

transactions/sec

应用程序资源使用问题。主要是内存使用问题，由于应用服务器的内存碎片问题 或内存泄漏问题，可能导致垃圾回收的开销随时间增大。也有可能是磁盘临时文 件积累造成磁盘访问开销增大。



response time

图6-14 响应时间上升曲线示例

应用程序设计问题。由于应用程序的设计存在可扩展性或可靠性问题，导致运行 开销随时间或业务对象的积累而增大。

数据库访问问题。数据库访问问题又可以分为许多类型：调优参数问题、表结构 或索引设计问题、垃圾数据问题等。其共同特点是导致应用程序完成特定操作访 问数据库的开销随时间而增大。

正是由于性能下降问题的原因可能很复杂，所以一开始可以通过自顶向下分析方法先 进行初步的定位。以吞吐率下降问题为例，先重启应用服务器，看吞吐率是否能恢复。如 果是，可以将问题定位到 JVM 内存使用问题上。如果不能，可以依次尝试重启服务器操 作系统/删除临时文件/重启数据库/恢复数据库备份等方式看能否恢复高吞吐率，这些方式 可以分别将问题范围缩小到服务器自身问题/临时文件积累问题/数据库问题(需要注意的 是一次只能恢复一个条件并做好系统备份工作)。之后再根据缩小之后的范围用自底向上 的方式进行详细的问题定位。

在性能下降问题中做问题定位的一个重要手段是分段和比较。简单说就是把性能高的 时间段里所需要的信息统计下来，再从性能差的时间段里取相应的监视信息与之比较，比 较结果里有显著差别的部分很可能就是引起性能下降的直接或间接原因。把一个较长的性 能测试分为几个连续的小段执行，分段收集页面响应时间/数据库快照/JVM 垃圾回收日志。 比较各时间段内同样的页面的响应时间。如果只是少数特定的页面响应时间随时间下降， 这些页面可能就是引起下降的原因。可以使用一些运行剖析分析工具来分析代码执行时间 为什么随时间延长。比较数据库快照可以看出随着执行时间数据库方面引起下降的原因， 比如随着脚本的执行，数据表增长过快引起的某些 SQL 执行时间变长，即可以通过比较 数据库快照里的某些 SQL 执行时间统计信息发现。如果所有的页面都有下降，就不是个 别页面的问题，而很可能是整个系统的问题。可以比较得到的 JVM 内存回收日志发现是 不是占用的JVM 越来越大导致了垃圾回收的工作成本变大。

第6章 大促带来的灾难：究竟能支撑多少访问量

**6.4.4** **穿了小两号的鞋——通过参数调优能解决的问题**

性能参数调优(Performance Tuning)是指通过调整配置参数改进应用系统性能的过程。 前面多次谈及很多调优就可以解决的性能问题。调优参数没有一个绝对的好或者不好的定 义，只有是否适合应用系统。不合适的参数会给应用系统的性能带来非常大的影响。更优 化的调优参数不见得会使原本已经工作在良好状态的应用系统带来成倍的性能提升。需要 注意的是，调优方法并不是一成不变的，需要通过实际调试经验的积累，针对不同的问题 加以灵活运用才能达到较好的效果。而且前面也提过，要防止“过度”调优，即因为调优 而导致掩盖掉应用本身的性能问题。性能参数调优的最低目标是避免由于不恰当的调优参 数引起的性能问题。

以 WebSphere 应用为例，应用系统参数调优一般包含操作系统调优、数据库调优、 Web服务器调优等几个方面。部分设置及调优工作可以在系统安装部署过程中或安装完成 之后进行，其他调优也可以在系统运行维护期间进行。

系统性能参数调优是一项非常复杂的工作，绝不是理解一个参数的含义，就可以对特 定的应用系统给出最优的参数值。而通常是一个调整参数一检验效果一再调整参数的反复 过程。性能参数调优的基本方法，就是通过反复的系统性能监测及分析，以配置参数调优 为手段对系统资源进行合理的配置及调整，从而最大限度地提高系统性能，避免潜在性能 问题的发生。

一般来说，完整的性能参数调优的过程可以概括为以下步骤：

(1)明确用户对系统的性能需求。

(2)全面深入了解当前系统架构及系统负载。

(3)结合系统现状及需求估计应用参数调优可以达到的最大目标。

(4)根据步骤(3)确定的参数调优目标制定初始参数调优方案(操作系统、 Web 服 务器、数据库服务器及WebSphere 应用服务器参数调优的范围及调优值等)。

(5)备份当前系统配置文件、应用参数调优方案。

(6)监测系统各项性能指标的变化情况，分析监测结果及系统日志对当前参数调优方 案效果进行评估。

(7)如果当前系统性能达到步骤(3)所制定的目标，可以尝试简化当前参数调优方

案(减少参与调优的参数或者对资源消耗严重的调优参数值加以调整，例如减小 JVM 初 始堆及最大堆参数值),重新开始执行步骤(5)。

(8)如果当前性能不能达到步骤(3)所制定的目标，通过性能监测结果分析可能的 性能问题的主要原因，确定相应作用领域的调优参数及参数值。

(9)应用步骤(8)确定的调优参数，监测系统性能变化。

(10)如果当前系统性能仍旧不能达到步骤(3)所制定的目标，重新执行步骤(8) 到(9),直到问题解决或确认应用参数调优无法满足目标要求为止。

(11)如果确认系统应用参数调优已无法满足目标要求，重新评估步骤(3)所制定的 调优目标，或者其他提高系统性能的途径(如升级系统硬件或修改系统设计或代码)。

(12)如果当前系统性能达到步骤(3)所制定的目标，重新开始执行步骤(7),直到 获得最优参数调优方案为止。

系统性能参数调优是整个Java EE应用生命周期中非常重要的一项工作。同时，系统 性能参数调优也是一个复杂而反复的过程，调优的效果通常受到多方面因素的综合制约， 例如系统可用硬件的多少、软/硬件的配置、并发及业务的类型、产品设计及代码实现等。 前文已经指出，不当的调优参数不但不能带来性能提升，反而可能对系统性能造成致命损 害。所以调整配置参数的时候一定要谨慎，尤其是在生产系统上进行性能参数调优时。

性能参数调优必须遵守以下原则：

建立合理的性能目标。超越软/硬件能力极限的目标是不可能通过参数调优实现 的。设定系统性能目标时，最好经过原型验证或系统容量规划。

越早考虑性能，性价比越高。从系统架构设计、性能测试到安装部署，参数调优 介入得越早，越容易找到理想的参数组合。

一次只调优一个参数，逐层调优。一次只调优一个参数是性能参数调优最重要的 原则。这是因为不同的参数设置对系统性能有着或积极或消极的影响，一次只调 优一个参数可以对每次调优的参数值所起到的效果进行正确评估，进而确定最优 的参数组合。反之如果通过一次性调整多个参数提高了性能，则很难判断究竟是 哪个参数影响了最终性能，因而也就无法进一步优化参数组合。

充分认识性能调优的两面性。不要为了调优而调优。如果当前系统的性能可以接 受，就不要试图通过参数调优进一步提升系统性能。

第6章 大促带来的灾难：究竟能支撑多少访问量

调优之前做好备份工作。如果不恰当的参数调优导致系统性能下降，至少还可以 恢复到以前的状态。

**6.5学习笔记——性能测试之小艾观**

通过这段时间的性能测试学习和实践，小艾对性能测试有了理性和感性的认识。每一 个用户体验良好的系统背后，都有着大量反复的性能测试，以保证在任何条件下，尤其是 高访问量大数据的情况下，终端用户仍然能够获得良好的使用体验。

首先，性能测试的设计人员必须透彻理解产品的需求和设计，尤其是设计中关于性能 目标的定义，之后设计出能够有效覆盖这些性能目标的性能测试计划。性能测试计划需要 从产品的并发能力、响应时间、可扩展性、可靠性、安全性等多方面保证产品符合性能通 过标准。而测试用例的最终出发点，还是要保证产品交付之后，客户在使用中不会出现性 能问题。在设计用例的时候始终要把这一点放在第一位。

与其他测试不同，性能测试过程中不仅要看用例本身的执行情况，还要对测试系统和 被测系统进行适当调优、实时的监视，及时发现问题，合理收集所需要的监视日志，为后 面的性能分析或问题定位做好准备。一旦出现性能问题，不能盲目地寻找原因，要遵循一 定的策略，才能避免花大量的时间和精力，走到错误的路径上。

通过与史静的长时间交流，小艾也发现，性能测试，尤其是计划的制定和性能问题的 分析定位的技巧，很大程度上依赖于对待测系统的了解程度和测试经验的积累。客户生产 环境上的问题分析解决也是积累经验的重要途径。

**参考资料：**

[1]2008. 孙磊，孙静等.《构建高性能WebSphere 企业级应用》

[2]Extended Cache monitor

<http://www.ibm.com/developerworks/websphere/downloads/cache_monitor.html>

[3]Monitoring performance with Tivoli Performance Viewer(TPV)

<http://publib.boulder.ibm.com/infocenter/wsdoc400/v6r0/index.jsp?topic=/com.ibm.websp>

从菜鸟到测试架构师———个测试工程师的成长日记

here.iseries.doc/info/ae/ae/tprf\_tpvmonitor.html

[4]Solving memory problems in WebSphere applications,Sun Jing,James Tang

<http://www.ibm.com/developerworks/websphere/library/techarticles/0706_sun/0706_sun.html>

[5]Oracle9i Database Tuning Guide 之 Oracle statspack

<http://docs.oracle.com/cd/E14981-01/wli/docs1031/dbtuning/statsApdx.html>

[6]各类DB2 数据库系统监视器简介可参考：

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/do> c/c0001138.htm

**第** **7** 章

**更新换代，重装上阵**



性能测试工作已接近尾声，这一天在项目收尾的会议上，性能测试组长史静谈到了下 一阶段的工作及人员安排。

“在接下来的几个月，我们的产品将进入下一个大版本的开发测试工作。项目小组的 成员可能会有一定的变动。”

小艾历数了自己过去几年来软件测试的经历，安装测试、单元测试、版本验证测试、 功能测试、全球化测试、性能测试……似乎比较陌生的测试领域就是迁移测试了。喜欢挑 战的小艾将下一阶段工作方向和目标锁定在迁移测试。

**7.1永葆青春活力——迁移**

在史静的力荐之下，小艾加入了迁移测试团队。对于在许多不同的测试领域都有所涉 猎的小艾来说，迁移测试还是全新的领域。小艾对于迁移的目的、概念、迁移测试做什么 以及该怎么做，都是一片茫然。麦习是迁移测试组的组长，在接下来的日子里小艾从他那 里学到了很多关于迁移测试的知识。

从菜鸟到测试架构师——个测试工程师的成长日记

小艾的第一个问题是，对于一个正常运行的企业级应用，客户为什么会有迁移的需 求?一般而言，迁移的过程势必对当前应用系统运行产生一定的影响，从而会给客户的营 业额带来一定的损失。同时客户还得投入大量的人力物力进行迁移的可行性研究、测试及 实施等工作。

麦习告诉小艾，其实你的这个疑问也就是企业级应用迁移的目的。

一般而言，迁移的目的是为了获取新版本更加完善的功能，以及最新的业务模式。众 所周知，当前技术的创新及业务模式的创新速度越来越快，为了适应这些变化，企业级应 用更新换代的步伐也在不断地加快。以IBM 电子商务解决方案WebSphere Commerce为例， 几乎每半年就会推出一个带有新功能的版本。该产品的客户为了能够使用这些最新的功 能，以增强其市场竞争力，会不断地要求更新到新的版本。类似这样的企业级应用的迁移 是比较常见的IT 需求。

当然还有其他一些原因促使客户去迁移应用，比如，随着客户业务的不断增长，对系 统的性能要求越来越高，旧版本越来越不能满足客户需求，新版本的应用系统具有更强的 处理能力；客户为了获得产品的持续的技术支持，一般而言，对于某个版本的软件产品的 技术支持，只提供一段时间的服务……以上种种原因都会促使客户去升级应用系统。总之， 企业级应用迁移的最终目的在于保证企业级应用在激烈的市场竞争中始终保持青春活力， 立于不败之地。

了解了迁移的目的，小艾接着问：“那到底什么是迁移和迁移测试呢?”

麦习对迁移及迁移测试的简单解读如下：

简单来讲，产品迁移是将旧版本的应用系统环境所有资产信息完整地迁移到新版本 的环境之上，同时要保证旧版本的功能在新的平台上正常运行，并且新版本的新增功能 也能正常工作。迁移测试是对迁移过程及其迁移结果进行验证，从而保证过程和结果的 正确性。迁移测试过程中除了会运用到自身特有的测试方法以外，还会借助于功能测试、 性能测试等来保证完成迁移的应用系统环境的正确性和可靠性。

小艾了解了迁移的基本概念后，将开始对迁移的内容做更深入的了解。

**7.1.1** **企业级应用架构**

听完了麦习关于迁移及迁移测试概念的描述，小艾对于迁移测试做些什么、怎么做， 依然一头雾水。

第7章 更新换代，重装上阵

麦习告诉小艾，要想深入了解迁移测试，首先需要理解企业级应用的架构，然后是迁 移的内容，最后才能有的放矢地制定迁移测试的计划。

麦习关于企业级应用架构的解析如下：

一般来说，企业级应用是为商业组织或大型企业提供特定解决方案的一套完整系统， 该系统包含了支持应用程序运行的各个组成部分。 一般划分如下：

底层的操作系统，如 Windows,AIX,Solaris,pLinux,zLinux 等。

后台数据库服务器，如DB2,Oracle 等。

面向用户的Web 服务器，如IBM <HTTP>,IIS,SunONE 等。

运行应用程序的中间件，如IBM WebSphere 应用服务器，WebLogic 应用服务器， Tomcat 应用服务器等。

部署在中间件之上的企业级应用。

对于一个企业级的应用程序，根据企业业务实际需求的不同，可以制定不同的解决方 案。对于企业级应用，可以从下面各种不同的角度来划分：

按不同的拓扑结构分：如单节点(应用系统的各个组成部分都在同一台机器上)、 双节点(应用系统的各个组成部分分布在两台机器上)、多节点(应用系统的各 个组成部分分布在两台以上的机器上)。

按不同的服务器集群'环境分：如垂直集群(集群成员都在一个节点上)、水平集 群(集群成员分布在不同的节点上)或混合集群(集群成员既有共存在一个节点 上的，也有分布于不同节点上的)。

按应用产品发布的版本分：如企业版 (Enterprise Edition)、专业版(Professional Edition) 、简装版(Express Edition)。

按企业应用运行平台分：如生产环境、开发环境等。

当然，对于一个企业级应用程序来说，它不仅仅是一个孤立的应用，往往集成了第三 方软件，比如LDAP,Portal,MQ, … …



1 <http://baike.baidu.com/view/292866.html?wtp=tt> 百度百科2012-02-15

图7-1所示是一个典型的企业级应用架构图，包含了上述各个层面的内容。图中所涉 及的各个方面都是迁移过程中需要考虑的因素。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 企业级应用 | | | | | | | | | | | | | | | | | 第三方软件 |
|  | 版本 | | | | | | | |  | | | 服务器环境 | | |  | |
| LDAP |
| 企业版 | | | 专业版 | | | 简装版 | | 开发环境 | | |
| Portal |
| 拓扑结构 | | | | | | | | | | | | | | | | |
| 单节点 | | | | | | 多节点 | | | | 集群环境 | | | | | |  | MQ |
| Web服务器 | | | | | 应用服务器 | | | 数据库 | | | | | | | | |
| <Http> | | IIS | Sun  ONE | | DB2 | | | Oracle | | | …… | | |  |
| 操作系统 | | | | | | | | | | | | | | | | | |
| Windows | | | | AIX | | | Solaris | | xLinux | | | | pLinux | | …… | | |

图7-1 企业级应用架构图

**7.1.2** **企业级应用迁移内容及主要步骤**

听了麦习对企业级应用架构的解析，小艾急切地想了解迁移的内容和步骤。在麦习的 指导下，小艾了解到，对于迁移而言， 一般包括下面的内容：

基础软件的升级和新版本应用产品的安装。基础软件是指图7-1中描述的操作系 统、数据库、Web服务器、应用服务器等一系列支撑企业应用运行的底层软件。 一个大型的Java EE应用，对于不同的版本，它所支持的这些基础软件的版本一 般也是不同的。基础软件的升级就是将旧版本Java EE应用所依赖的基础软件升 级到新版本Java EE应用所要求的版本。完成了基础软件的升级，然后就是安装 新版本的应用产品，但不需要进行应用的部署。

数据库的迁移。这里的数据库迁移不是指数据库管理系统本身的升级，而是将旧 版本数据库中的数据结构及数据迁移到新版本的应用系统之上，使得迁移之后的 数据结构及数据符合新版本应用系统后台数据库的规范和要求。例如，新版本数 据库和旧版本数据库相比，新版本的数据库模式增加了部分表；部分表的字段属 性做了相应的修改等，迁移过程就需要对这些变化进行相应的处理。

应用程序的迁移。即将旧版本的应用程序迁移到新的版本，迁移之后保证旧版本 功能同样适用，同时新版本的功能也能正常工作。应用程序的迁移主要包括应用 配置信息、EJB 模块、业务逻辑模块、Web 应用模块等的迁移。

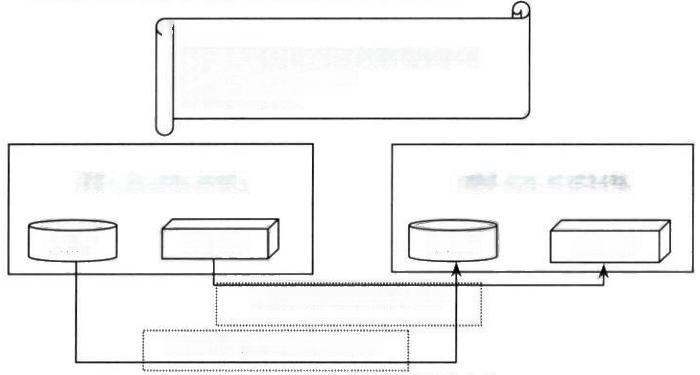
第 7 章 更新换代，重装上阵

对于应用系统迁移的步骤而言，迁移的这三个方面之间存在依赖关系，因而有先后顺 序之分。

一般而言，企业级应用迁移的第一步是基础软件的升级和新版本应用产品的安装。 第二步是进行数据库迁移。

最后完成应用迁移。

图7-2描述了企业级应用程序迁移的主要步骤。

q 迁移的步骤

1. 升级基础软件，安装新版本的应用产品

2. 迁移应用后台数据库

3. 迁移应用程序

旧版本Java EE应用系统 新版本Java EE 应用系统

数据库 应用程序 数据库 应用程序

迁移旧版本的应用到新版本

迁移旧版本的数据库到新版本

图7-2 企业级应用程序迁移步骤

在新版本产品中， 一般都会提供迁移工具来实现旧应用系统向新应用系统的迁移。这 个迁移工具可以实现企业家应用两个核心部分的迁移，即数据库的迁移和应用程序的迁 移。迁移工具的主要目标是：

简化数据库和应用程序迁移的流程。

提高迁移的效率和性能，尽可能缩短迁移的时间。

保证迁移质量，这一点对于客户来说是非常重要的。作为一个客户，也许可以容 忍迁移工具使用上的不方便、不灵活，也能暂时容忍不太高的迁移效率。但决不 能容忍客户的数据没有正确迁移，以及客户业务逻辑、商业流程的丢失等。在后 面的章节中，会提到如何利用有效的迁移测试方法来保证迁移工具的质量和迁移 的质量。

**7.1.3** **企业级应用迁移中客户关注的问题**

了解了迁移的主要内容和步骤之后，善于从客户角度思考问题的小艾又有了新的疑 问，在迁移过程中，客户最关注哪些问题呢?

麦习指出，一方面客户关注迁移步骤的简单性、易用性及迁移流程的高效性。因而在 整个迁移过程中，人工干预越少越好，我们应当尽可能实现整个迁移过程的自动化。

另一方面客户更加关注迁移前后，应用系统的宕机时间。对于企业级应用的迁移，涉 及迁移前后两个系统的切换，而这种切换一般会产生宕机时间。对于企业级应用系统，尤 其是电子商务类的应用系统， 一般来说都提供7×24小时服务，迁移过程中的暂停服务意 味着这些应用系统的客户无法访问/使用原有系统，因而会造成商业上的损失，如营销额 度的损失。因而对企业客户来说，最关注的是他们的应用系统在迁移过程中的宕机时间， 这段时间越短越好，理想状况是零宕机。但对于企业级应用系统的实际迁移来说，零宕机 一般实现起来比较困难。一般而言，美国主要银行对计算机系统的要求是一年宕机时间不 能超过5分钟，这是一个非常高的要求，在企业级应用做移植时是第一要考虑的要素。

对于应用程序的迁移，可以在旧的应用系统提供服务的前提下同时进行，而无须付出 额外的时间代价。但是对于数据库的迁移，为了保证迁移前后的数据的一致性，在开始数 据库迁移之前，应停止数据库数据的更新。如果迁移过程中，应用继续对外工作，那么数 据会发生相应变化，就无法保证迁移前后数据的一致性。因而数据库迁移之前，一般会暂 停对外服务，直到迁移结束。所以可以认为整个迁移过程中的宕机时间一般受数据库的迁 移时间影响比较大。

**7** **.2迁移测试**

经过一段时间的充电，小艾已经对企业级应用的迁移内容和步骤有了比较详细的了 解。麦习带着小艾开始了对于迁移测试的学习和实践。

对于迁移测试，麦习语：

迁移测试不同于其他的测试类型，其他测试类型都只需要设计相应的测试用例，在 当前版本的环境上执行相应的测试。而迁移测试涉及不同版本的应用系统环境，比如， 移植之前旧版本的系统环境、完成从旧版本到新版本迁移的系统环境、新版本的纯环境

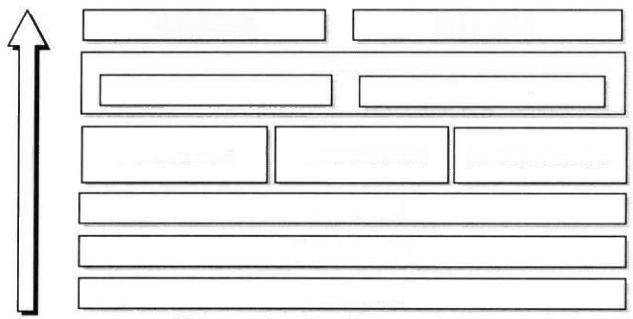
第7章 更新换代，重装上阵

等。迁移测试中，部分测试验证工作是在迁移环境上进行的，还有一部分比较验证需要 借助于不同版本的环境来完成。

根据涉及的验证环境的不同，我们将迁移测试方法分为垂直迁移测试方法(VMTM, Vertical Migration Test Methodology)和水平迁移测试方法(HMTM,Horizontal Migration Test Methodology)。

**7.2.1** **垂直迁移测试方法(VMTM)**

麦习告诉小艾，垂直迁移测试是迁移测试的基本方法，所有的验证都是在完成从旧版 本到新版本迁移的系统环境上进行的。按照迁移的先后顺序，首先是软件升级测试、数据 库迁移测试、应用迁移测试，其次是在迁移之后的环境上进行子系统迁移测试、集成系统 迁移测试、客户定制化迁移测试、功能测试、性能回归测试和迁移扩展性测试。图7-3中 涵盖了垂直迁移测试的主要内容。

性能回归测试 迁移扩展测试

功能测试

管道数据测试

子系统迁移测试

功能回归测试

集成系统迁移测试 客户定制化迁移测试

应用迁移测试

数据库迁移测试

软件升级测试

图7-3 垂直迁移测试(VMTM)

通过快速的学习，小艾了解到作为一个完整的垂直迁移测试的框架，垂直迁移测试需 要考虑从底层基础软件到上层应用的每一个功能点。同时小艾也了解到，为了更有效地进 行测试工作，并不是所有的上述工作都必须由迁移测试团队来完成，有些工作可以由不同 的测试团队合作来完成，比如功能回归测试，就可以由功能测试团队来协助完成。麦习也 告诉小艾，在后面的实际工作中，不仅要了解如何执行，还要多学习迁移测试的策略，为 以后职业的发展奠定好基础，争取成为测试领域的专家。

小艾对于垂直迁移测试有了比较深入的了解，并且对它的每一部分有了一个小的总结。

**1.** **基础改造——软件升级测试**

软件升级测试是保证支持应用运行的软件都被正确地安装或迁移。以一个具体的 JAVA EE 应用为例，旧版本的应用支持 DB2V8,IBM <HTTPV6> 及 IBM WebSphere 应用服 务器V6, 新版本的应用支持DB2 V9,IBM <HTTPV7> 及IBM WebSphere应用服务器V7, 则软件升级测试工作主要包括：

DB2 V8到 DB2 V9的升级测试，这一过程是参照DB2 信息中心的升级文档来完 成的，升级完成后需要对迁移后的DB2 V9系统进行简单的验证，比如手工创建 数据库、建表、导入/导出数据等，以保证迁移后的数据库系统正常工作。

IBM <HTTP> 和 IBM WebSphere 应用服务器的升级测试，可以参照IBM WebSphere 应用服务器信息中心²的文档来安装和迁移。对于新安装或升级的IBM [HTTP Web](HTTPWeb) 服务器，可以启动 Web 服务器的默认配置，然后访问 Web 服务器的默认端口来 验证 Web 服务器是否正常工作；对于安装后的IBM WebSphere 应用服务器，可 以创建一个默认应用，并通过启动，停止默认应用，访问服务器控制台等一些基 本操作来验证其是否正常工作。

在确认数据库，Web服务器和IBM WebSphere 应用服务器都完成了正确升级的前提 下，可以安装新版本的应用产品。为确保新版本的应用产品本身没有问题，在进行 应用系统迁移之前，需要部署一个测试用的应用系统。这里创建测试应用系统有两 个目的，一个是通过启动测试应用系统，验证应用的一些简单功能和业务流程， 来保证新版本的应用没有问题；另外一个目的在于，测试应用系统创建的数据库 和应用可以作为后面的水平迁移测试的比较对象，是水平迁移测试验证的基准。

**2.** **客户数据不能丢——数据库迁移测试**

数据库的迁移测试目的在于保证旧的数据库数据结构(schema), 数据库中加密的及没 有加密的客户数据都被正确地迁移过来。数据库迁移验证的主要内容如下：

1 <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.qb.upgrade.doc/doc/> c0023662.html,DB2 信息中心

2 <http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/> info/ae/ae/welc6topmigrating.html IBM WebSphere信息中心

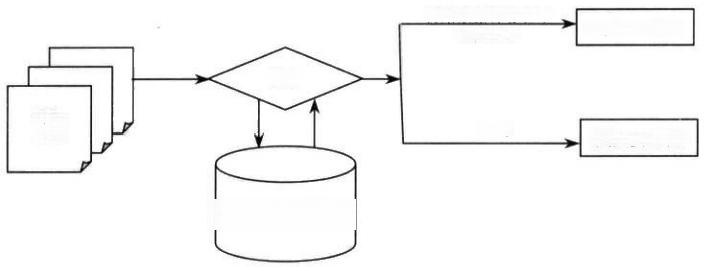
第7章 更新换代，重装上阵

数据库数据结构迁移测试，主要包括应用所依赖的后台数据库中的表、列、索引、 约束(主键，外键)、触发器以及存储过程的迁移。旧环境上定制的数据库模式 在迁移后的数据库中必须完整地得以保存，否则可能导致客户定制功能不能正常 工作。具体验证方法将在水平迁移测试方法中详细介绍。

数据库数据 (Data) 迁移。

一方面保证新版本应用的初始数据都被正确地导入数据库。初始数据是引导应用程序 开始运行的基本数据，是支撑应用正常运行的基础。它们和数据库结构同样重要，如果丢 失了这些初始数据，会导致应用系统无法正常工作。

一般而言，当企业级应用安装之后，应用的初始数据以特定的文件格式存在于产品的 安装目录下，在部署应用、创建应用数据库的过程中导入数据库。初始数据的验证可以通 过比较所有存在于初始数据文件中的数据和迁移之后的数据库表中的记录是否匹配。如图 7-4所示，如果前者都存在于数据库中，我们可以认为初始数据的迁移是正确的。



初始数据都存在于 数据库中

检查

数据迁移失败

迁移之后的数据库

数据迁移成功

初始 数据

否则

图7-4 初始数据验证

另一方面，企业的商业数据的迁移对于企业来说也是非常重要的。商业数据主要包括 存储商业业务的所有数据，比如企业的客户的个人信息。这些数据是企业的宝贵资产，任 何JAVA EE的应用在迁移的时候，都会非常关心这些数据在迁移之后的数据库中的正确性 和完整性。商业数据的验证需要用到不同数据库版本之间的数据比较，验证方法将在水平 迁移测试方法中详细介绍。

**3.** **功能点不能少——应用程序迁移测试**

应用的迁移测试是验证应用程序迁移流程和结果的正确性，主要内容包括：

确保应用的配置信息被正确地迁移。根据迁移场景的不同，迁移的结果验证也是 不同的，因而应当根据具体的迁移案例来做具体的验证。例如，如果新的应用系 统和旧的应用系统共存于一台机器上，那么在迁移之后的系统中，配置文件里服 务器名称等信息就应该与迁移之前的保持一致；如果新的应用系统和旧的应用系 统分布在不同的机器上，那么在迁移之后的系统中，配置文件里服务器名称等信 息就应该与新系统所在的服务器名称一致。

确保新版本应用的功能模块被自动地加载到迁移之后的应用系统中。比如，新增 的模块、节点、设置或属性等。如果这些信息没有被正确地加载，迁移之后的系 统上一些新功能可能不能正常工作，从而不能让客户分享新版本中的新功能。

确保应用服务器旧版本的配置信息被正确迁移到新的应用系统中。在旧的系统 中，客户可能做了很多应用服务器的性能优化或参数调优，如果这部分信息不能 被正确地迁移，在迁移之后的系统中，客户还得重新优化。

确保 Web 服务器旧版本的配置信息被正确迁移到新的应用系统中。比如，最大连 接数、超时时间的定义等。

**4.** **子系统迁移测试**

作为企业级的应用解决方案，根据不同的业务功能可以把它划分为相互联系的很多子 模块，比如一个电子商务相关的JAVA EE的应用，可以分为目录管理系统、销售系统、营 销系统、交易管理系统、订单管理系统等，参见第5章图5-5。对于这样比较复杂的Java EE 应用，一般会对各个子系统的迁移分开考虑，这就引入了子系统的迁移。子系统的迁移测 试是验证这些子模块的迁移步骤和迁移程序的正确性。

各个子系统的迁移，除了完成数据库迁移和应用程序迁移之外，可能还会有自身的一 些迁移步骤。因而我们需要设计相应的测试用例来验证子系统的迁移。

**5.** **集成系统迁移测试**

集成系统迁移测试是验证 Java EE 应用软件本身和其他应用(比如 LDAP,Portal, MQ) 作为一个集成系统在完成迁移之后，能够正常工作。

与第三方的软件集成迁移测试，可以有两种方式。一种是旧版本的应用系统和第三方 的软件都需要做迁移。集成软件迁移需要遵从集成软件自身的迁移步骤，这些步骤一般在 Java EE的应用本身的迁移手册中不会涉及，应用迁移手册只会提供双方迁移之后集成配

第7章 更新换代，重装上阵

置的相关信息。另一种是我们只需要对应用系统做迁移，第三方软件本身不需要迁移。迁移 之后的应用系统环境需要重新配置与第三方软件的集成，以保证集成环境的正常工作。

**6.** **客户定制化迁移测试**

客户定制程序迁移测试主要是测试客户在旧版本环境上根据自身业务要求的需要进 行定制的程序，比如，EJB,JSPs,Classes 等，是否被正确地迁移到新系统的版本。客户 化定制一般都是在开发环境上开发完成的，所以客户定制化迁移测试的步骤是，首先在旧 版本的开发环境上完成客户化定制，保证功能的正确性。然后将客户化定制从旧版本环境 中导出，再将其导入新版本的开发环境中，最后在新的开发环境中完成迁移步骤，并验证 定制迁移之后功能的正确性。在开发环境中完成了迁移之后，再将新环境上的客户化定制 部署到已经完成了迁移的新版本的运行环境上。

**7.** **功能测试**

功能测试主要是验证迁移之后的应用系统功能的完整性和正确性。迁移测试中的功能 测试分为两种，一种是贯穿整个迁移过程的管道数据 (Pipeline Data) 功能测试；另一种 是在完成了迁移的应用系统上的功能回归测试。

(1)管道数据功能测试

管道数据就是指那些存在很多中间状态的端到端 (end-to-end) 测试用例所使用的测 试数据。管道数据功能测试就是指迁移测试团队在整个迁移过程中执行管道数据测试用例 来确保数据被正确地迁移及验证迁移后应用系统业务流程的正确性。

管道数据测试用例必须是完整的、端到端的。每个功能模块的测试组负责选择选取管 道数据测试用例，其目标是保证覆盖其功能模块的主要功能。管道数据的功能测试是贯穿 于整个迁移过程的，管道数据测试的具体方法如下：

作为一个完整的测试用例，迁移之前，测试用例在中间的某个步骤中断，完成一 部分的步骤。在迁移之后的环境中，中断的测试可以继续进行以完成剩余的步骤。 以电子商务系统的一个订单流程为例，在电子商务应用系统迁移之前，终端用户 将其选中的商品加入了购物车；完成了系统迁移之后，终端用户在他的订单系统 中，依然可以看到这些未完成的订单，并能够顺利地完成付款从而结束这个订单。 作为一个完整的测试用例，迁移之前，整个测试用例已完成，其定制的业务流程 已生效。测试用例生成的数据在迁移之后依然生效。以电子商务系统的一个促销

流程为例，迁移之前，定义针对部分商品的促销规则，迁移之后，这些促销规则 依然生效。

(2)功能回归测试

功能回归测试是选取回归测试的用例，保证新旧功能模块在迁移之后的环境中的正确 性。一种情况是保证新版本中的新功能在迁移之后的环境中正常工作；还有一种情况是， 在迁移之后的环境中，由于数据进行了迁移，也需要保证旧功能在新版本中的质量，这两 种情况，都称为移植环境下的功能回归测试。

功能回归测试可以遵循以下两个标准：

当将旧的系统移植到新版本之后，需要保证新模块功能可以无缝链接到旧的系统 功能模块，没有软件缺陷。

当将旧的系统移植到新版本之后，需要保证旧的数据的正确性，以及功能的完 整性。

**8.** **性能回归测试**

性能回归测试是比较迁移之后的应用系统的性能测试结果和旧版本的性能测试结果。 期望结果是没有性能的下降，性能指标对于客户而言，是可以接受的。 一般情况下，性能 回归测试会选择一个最基本的压力测试的测试用例，首先在迁移之前的旧版本上运行得到 一个测试结果，然后在迁移之后的环境上运行同样的测试用例，得到一个测试结果，再比 较吞吐率(吞吐率定义可参见第6章6.2.2节)的大小。

通常情况下，性能回归测试由迁移测试团队和性能测试团队合作完成，前者提供迁移 后的迁移环境，后者负责性能测试。

**9.** **迁移扩展(Scalability)测试**

迁移测试环境一般都是采用应用系统开发默认的应用和数据库，与企业级客户的真实 应用系统环境相比，默认数据库的规模和应用的大小都非常小。所以针对这种开发环境的 测试和客户实际环境的差别，一般在迁移测试中会考虑扩展性的相关测试，也就是针对大 数据量和大型应用的测试，以保证应用和数据迁移能够在可以接受的时间内正确完成。迁 移扩展性 (scalability) 测试也是从数据库迁移和应用程序迁移两方面来考量的：

(1)一方面是大数据量的数据库迁移测试，对于真实客户环境而言，数据库大小动辄

第7章 更新换代，重装上阵

几十GB, 甚至是上百GB 的规模。在Java EE 应用的实际迁移测试过程中，一般都需要模 拟实际客户数据组成结构，生成大数据量的数据，然后运用相应的迁移工具完成数据库的 迁移。前面提到，企业级应用系统的迁移时间主要取决于数据库的迁移时间，所以大数据 量的数据库的迁移时间是迁移测试中一个很重要的考量因素。

(2)另一方面是大应用的迁移，这里大应用是指应用所包含的文件数量较多，应用包 较大。比如，默认的应用大小在200 MB 左右，而客户实际的应用可能是GB 的规模。在 迁移测试过程中，在保证不影响应用功能的前提下，手工地添加一些大规模的文件，以扩 充应用的大小，然后使用迁移工具迁移应用，测试迁移大应用的效率和正确性。

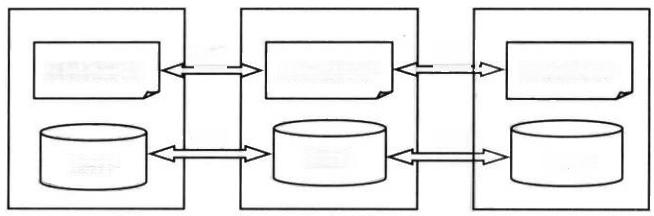
**7.2.2** **水平迁移测试方法(HMTM)**

完成了对于垂直迁移测试的实践和学习后，在麦习的指导下，小艾开始学习水平迁移 测试。麦习告诉小艾，垂直迁移测试和水平迁移测试不是相互独立的，水平迁移测试主要 运用于垂直迁移框架中的数据库迁移测试和应用迁移测试。

麦习告诉小艾，水平迁移测试与垂直迁移测试主要区别在于前者需要涉及多个应用系 统环境的比较，如图7-5所示的水平迁移测试。主要包括迁移之前旧版本的系统环境A, 完成从旧版本到新版本迁移的系统环境B, 新版本的纯环境C, 即新版本上只有初始数据 和默认应用的系统环境。而后者的测试工作只涉及迁移之后的系统环境B。

水平迁移测试多个系统主要的比较内容如表7-1所示，将完成迁移的环境B 与旧版本 环境A 比较，以保证迁移之前旧版本环境的信息被正确地迁移，不存在丢失数据或配置信 息的情况；将环境 B 和新版本的纯环境C 进行比较，以保证新版本的相关信息和数据被 正确地部署到迁移之后的环境B 上。

旧版本的系统环境 A 迁移之后的系统环境B 新版本的纯环境 C



比 较 应用配置文件

比较

数据库

图7-5 水平迁移测试

应用配置文件

应用配置文件

数据库

数据库

比较

比较

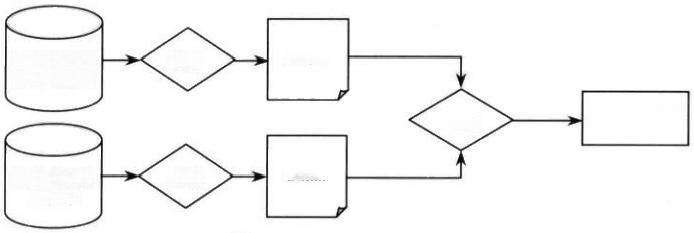
**表7-1水平迁移测试**

|  |  |  |
| --- | --- | --- |
|  | 迁移之前旧版本的系统环境A与迁移之后 的系统环境B的比较 | 迁移之后的系统环境B与新版本的纯环境C的比较 |
| **数** **据** **库** | 验证迁移之前环境A中的数据结构及商业 数据是否存在于迁移后的环境B中 | 验证新版本纯环境C中为新功能所增加的数据库结 构及初始数据是否存在于迁移之后的环境B中 |
| **应用配置信息** | 验证迁移之前环境A上应用配置信息是否 存在于迁移后的环境B上 | 验证迁移之后的环境B中的应用配置信息是否遵从 新版本的要求 |

水平迁移测试是垂直迁移测试的必要补充，垂直迁移测试主要是对整个应用的一个从 底向上的完备的考量，而水平迁移测试主要是考虑数据库和应用程序迁移前后的比较，更 多的时候是从属于垂直测试，可以看做垂直测试的一部分。对于水平迁移测试，小艾做了 如下的总结。

**1.** **新版本数据结构验证**

数据库数据结构主要包括数据库中所有的表、列、索引、约束(主键/外键)、触发器 及存储过程。验证数据结构首先需要将数据结构按照类型以一定的格式导入一个文件中以 进行比较验证，比如结构性的标记语言XML 文件。具体验证步骤如图7-6所示，将新版 本纯环境的数据库数据结构提取出来，导入XML 文件A 中。同时将迁移之后的数据库数 据结构提取出来，导入XML 文件B 中，如果A 中所有的数据结构都存在于B 中，我们认 为产品数据结构完成了正确的迁移。



A.xml

比 较 比较结果输出

B.xml

图7-6 数据库数据结构验证

新版本纯环 境的数据库

迁移之后的 数据库

提取

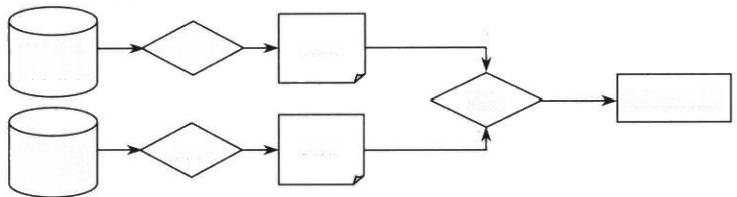
提取

**2.** **商业数据验证**

商业数据包括所有商业事务处理相关的数据，比如客户相关的信息等。商业数据验证 的方法如图7-7所示，将迁移之前的数据导入一个文件，比如 XML 文件 A, 将迁移之后

第7章 更新换代，重装上阵

的数据导入XML 文件B, 比较文件A 和文件B, 如果文件A 和文件B 内容一致，我们认 为商业数据被正确迁移了。



A.xml

比较 比较结果输出

B.xml

迁移之前 的数据库

迁移之后 的数据库

提取

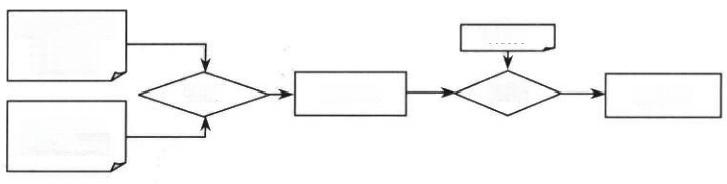
提取

图7-7 商业数据验证

**3.** **应用配置信息验证**

应用配置文件中定义了应用相关的配置信息，如数据库信息、Web 服务器信息、第三 方集成软件信息、应用功能模块的信息等。迁移后应该保证应用配置文件的正确性和一致 性，这里所用的验证方法还是比对。

比较迁移之后的配置文件与迁移之前旧版本的配置文件来验证旧版本上的配置 信息是否被正确地迁移。如图7-8所示，通过比较得到结果A1,A1 中所显示的 不同，并不意味着都是错误。对于应用功能模块的信息，一般情况下，迁移前后 应该保持一致，如果出现差异，一般是问题的可能性比较大，需要仔细检查。而 对于数据库信息、Web 服务器信息的差异，可能是正确的迁移，一般出问题的可 能性比较小。这是因为在迁移的过程中，应用可能切换了数据库机器或Web 服务 器机器。因而对于这种情况，应该定义过滤条件Rule1, 忽略这种类型的不一致， 得到进一步的比较结果B1 。对于比较结果B1, 如果所有的旧版本配置信息都存 在于迁移之后的配置信息文件里，并且不存在参数值的差异，我们认为旧版本的 配置信息被正确地迁移了。



迁移前旧版本 应用配置

信息文件

比较 结果A1

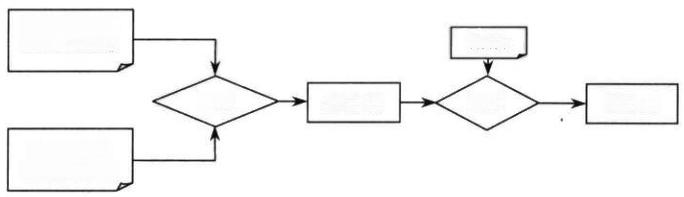
迁移后应用 配置信息文件

过滤 结果 B1

Rulel

图7-8 迁移前后应用配置文件比较

比较迁移之后的配置文件与新版本纯环境的配置文件来验证新版本的新的功能 模块是否被正确地部署。如图7-9所示，通过比较得到结果A2,A2 中所显示的 不同，并不意味着都是错误。对于应用功能模块的信息，如果只是功能模块、节 点、属性值的不一致，这种差异，往往是正确的比较结果。那是因为旧版本上的 参数值不同于新版本的默认值，旧版本上的值被正确地迁移过来了。当然如果新 版本的某些功能模块参数值在迁移之后的配置文件中丢失了，很有可能就是一个 问题。一般情况下，也是通过定义过滤条件，如图7-9所示的Rule2, 对结果A2 进一步筛选，得到进一步的比较结果B2 。对于比较结果B2, 如果所有的新版本 配置信息都存在于迁移之后的配置信息文件里，并且不存在参数值的差异，我们 认为新版本的配置信息被正确地加载了。



新版本纯环境 应用配置信息

比较 结果A2

迁移后应用 配置信息文件

图7-9 新版本应用配置文件比较

过 滤 结果 B2

Rule2

以上内容涵盖了水平迁移测试的方法和内容。水平迁移测试的各个部分相对独立，对 于数据库和应用可以分开进行验证。但它们又都是垂直迁移测试流程中不可或缺的一环。

**4.** **垂直迁移测试与水平迁移测试之比较**

通过这段时间对迁移测试方法的系统学习，小艾对垂直迁移测试和水平迁移测试有了 更加深刻的理解。他将垂直迁移测试和水平迁移测试的关系做了如下的小结：

垂直迁移测试是水平迁移测试的基础，只有完成了垂直迁移测试中的数据库迁移 或应用程序迁移才能进行水平迁移测试。水平迁移测试需要涉及多个应用系统， 垂直迁移测试都在完成从旧版本到新版本迁移的系统环境上完成。

水平迁移测试能在早期发现问题，它无须等待整个迁移流程结束后才开始。当完 成了数据库迁移成功后或应用迁移成功后，可以进行相关的验证工作。而垂直迁 移测试中的大部分测试工作只有当完成了整个迁移流程之后才能进行，比如功能 测试、性能测试。

第7章 更新换代，重装上阵

水平迁移测试更多地关注数据的准确性，如水平迁移测试能够发现数据丢失或数 据库结构的不正确这类问题。而垂直迁移测试更多地关注系统的整体功能、性能 等的迁移是否成功。

水平迁移测试相对复杂，水平迁移测试需要比较不同版本数据库之间数据库结 构、商业数据及应用的配置文件，因而需要一些额外的成本去准备不同版本的环 境。而垂直迁移测试测试只需要一套迁移环境。

**7.2.3** **其他迁移测试内容**

垂直迁移测试和水平迁移测试是迁移测试的主要方法，除此之外，迁移测试还包括其 他一些测试内容。

**1.** **兼容性测试**

兼容性测试的目的在于确保以前的低版本(比如 IBM 一般考虑前两个低版本)上部 署的经客户定制过的应用所依赖的一些因素能够正常地提供服务，而使客户的定制应用不 受影响，从而客户无须重新构建或部署他们的应用。

这里影响兼容性的因素主要有：

开发框架升级了，比如，EJB 的版本升级在迁移过程中需要重新部署。

API 的不兼容性。API 兼容性要求当我们逐渐弃用旧的类(classes) 、方法(method) 或接口 (interfaces) 时，应提供新的替代的类、方法或接口。同时，当前的客户 不做任何改动仍然可以使用旧的类、方法或接口。

行为的变化，随着版本的升级，某些方法或类的的行为发生了变化可能导致错误。 比如，某些返回值的发生了改变，而客户的程序依赖于这个返回值，这种情况就 会导致异常的发生。

**2.** **迁移文档测试**

迁移文档测试的目的在于保证迁移文档的质量，确保客户能够遵循正确的迁移文档完 成客户应用系统的迁移工作。

迁移测试团队既要保证迁移文档的正确性，又要保证手册具备较高的可读性。 一般而 言，测试团队应该多和开发人员沟通来保证细节的正确性。

迁移手册中不仅要包含迁移执行的步骤，还应该包含必要的故障解决办法，比如针对 一些常见的迁移错误步骤，给出一些建议或故障恢复的解决方案，以帮助客户从失败的情 况下恢复环境，完成迁移。

迁移文档测试是迁移测试中最重要的部分，这是客户可以遵循的唯一标准，它的每一 个错误或者不清晰的描述，都会给实际的应用迁移带来不可估量的损失。

**3.** **真实客户环境测试**

客户的真实环境资产，尤其是存在大量定制化的客户环境是最好的定制化迁移测试资 产，可以发现性能问题及由于客户定制化所引发的特有问题，同时它也是验证迁移程序和 迁移测试手册的一种最好的环境。

基于真实客户环境的迁移测试是迁移测试重要的验证方式，有了这样的环境，对于保 证迁移测试的正确性，特别是对于性能方面的验证是非常必要的。但是很多时候，并不是 所有的应用都能够拿到客户的真实环境或者真实数据，这时，只能通过模拟来实现类似的 迁移测试。但无论如何，这是迁移测试中需要特别重视的策略。

**4.** **迁移安全性(Security)** **测试**

迁移安全测试和安装测试中的安全类似，由测试人员在迁移环境上进行文件权限验证 及敏感数据检查。文件权限测试是针对非 Windows 平台，检查应用系统环境的文件系统各 个层次的文件夹及文件权限是否都遵从应用安全的要求。敏感数据验证主要是检查在迁移 过程中，诸如用户密码、账号等信息都不会以明文的方式显示在窗口或日志信息中。

另外，对于数据库中包含的一些加密数据，在迁移之后，应该保证其被正确地迁移。

**7.3** **迁移测试的计划和实践**

经过对迁移测试的全面了解，小艾已经具备了一定的理论水平。麦习告诫小艾，只有 迁移测试的基本理论是远远不够的，很多细节性的东西都需要在实际项目中去磨练、去领 悟。为了让小艾更快、更好地掌握迁移测试，麦习让小艾通过全程参与一个完整的迁移测 试项目来体会。通过一段时间的实际项目的锻炼，小艾不仅熟悉了迁移测试的理论和测试 方法，还学会了如何制定迁移测试的测试范围。

第7章 更新换代，重装上阵

**7.3.1** **如何选择迁移测试平台组合和测试用例**

迁移测试的测试流程跟其他测试类型大体类似，都包含需求分析、制定项目计划、定 义测试场景和测试用例、测试执行和测试验收等步骤。对于迁移测试，鉴于其特殊性，小 艾只是把迁移测试中比较特殊也比较重要的定义测试平台组合和定义测试用例在本书做 了总结，其他部分可以参考本书的其他章节。

**1.** **选择测试平台组合**

麦习告诉小艾，企业级应用支持的平台组合达上百种之多，在迁移测试环节不可能全 部覆盖。通常在做迁移测试计划时，只能根据项目的可用资源和进度要求，选择具有代表 性的一部分平台组合来进行测试。为了保证测试的质量和满足企业级客户的期望，通常采 用以下步骤来选择测试环境的平台组合。

(1)建立一个迁移路径矩阵表，如表7-2所示，罗列出新版本应用支持的所有的迁移 路径。然后根据从服务团队，市场团队或技术支持团队获取的信息，确定客户对各种迁移 路径的实际需求，来识别出所有支持路径中的每条路径优先级的高低程度。

如表7-2 所示，支持路径包括，旧版本简装版迁移到新版本企业版、专业版、简装版； 旧版本专业版迁移到新版本专业版、企业版；旧版本企业版迁移到新版本企业版。从获得 的客户反馈我们知道大部分的客户迁移路径是旧版本企业版到新版本企业版，旧版本专业 版到新版本企业版；其次是旧版本简装版，专业版到新版本企业版的迁移，以及旧版本专 业版到新版本专业版的迁移；需求最小的是旧版本简装版到新版本简装版的迁移。

**表7-2迁移路径矩阵表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **优** **先** **级** | | **新** **版** **本** | | |
| **企** **业** **版** | **专** **业** **版** | **简** **装** **版** |
| 旧版本 | 企业版 | 高 | 不支持 | 不支持 |
| 专业版 | 高 | 中 | 不支持 |
| 简装版 | 中 | 中 | 低 |

(2)接下来建立一个平台支持矩阵表，如表7-3所示，罗列出所有支持的软件信息， 如操作系统及数据库等信息。然后从服务团队获取信息，以确定客户各种软件配置的实际 需求，来识别出所有支持路径中的每条路径优先级的高低程度。

**表7-3平台支持矩阵表**

|  |  |  |
| --- | --- | --- |
|  | **数** **据** **库** | **优** **先** **级** |
| AIX | DB2 | 高 |
| Oracle | 低 |
| Linux | DB2 | 低 |
| Oracle | 高 |
| …… | …… | …… |

注：表7-3只是一个例子，不代表真实场景。

(3)将矩阵表7-2和矩阵表7-3合并到一张表中，并罗列出所有测试配置组合。综合

表7-2和表7-3的结果，识别出每种组合的优先级顺序。如在AIX 操作系统上旧版本到新 版本的迁移路径的优先级见表7-4。

表7-4测试组合配置矩阵表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **优** **先** **级** | | 操作系统/数据库 | | | | |
| **AIX/DB2** | **AIX/Oracle** | Linux/DB2 | Linux/Oracle | …… |
| 旧版本  > 新版本 | 企业版>企业版 | 高 | 中 | 中 | 高 | …… |
| 专业版 → 1 企业版 | 高 | 中 | 中 | 高 | …… |
| 专业版-> 专业版 | 中 | 中 | 中 | 中 | …… |
| 简装版> 企业版 | 中 | 低 | 中 | 低 | …… |
| 简装版> 专业版 | 中 | 低 | 中 | 低 | …… |
| 简装版→简装版 | 低 | 低 | 低 | 低 | …… |

在制定迁移测试计划时优先选择高优先级的配置组合。

(4)通过前面的筛选，高优先级的配置组合可能依然很多，这时就需要其他的策略来 进一步筛选。根据经验，一般优先级顺序的选择从高到低的的顺序为：应用本身低版本到 高版本的路径(如旧版本1->新版本，旧版本2->新版本， ……),操作系统，数据库，版 本(简装版，专业版，企业版)及Web 服务器。

通过以上一些策略，就能够比较好地选择出迁移测试的平台组合。

**2.** **如何选择测试用例组**

确定了迁移测试平台组合，接下来就该定义迁移测试用例了。在计划某个版本迁移测 试用例的时候，由于受到可用资源和进度要求的限制，一般不可能在所有的平台组合上测 试所有的测试用例。根据实践，小艾的项目组将所有的测试用例分成不同的组，比如基本

第7章 更新换代，重装上阵

测试用例组，扩展测试用例组和高级测试用例组，然后将这些测试用例分散到不同的测试 平台上进行测试。这样的方式保证在资源最少的情况下完成了最大限度的迁移测试。这样 的实践对于其他的JAVA EE的应用也是很有参考价值的。

基本测试用例组包括迁移测试的主要流程的测试用例，主要包括：

软件升级测试；

数据库迁移测试；

应用程序迁移测试；

子系统迁移测试；

客户定制化迁移测试；

功能验证测试。

扩展测试用例组，主要包括：集成系统测试，比如LDAP,Portal,MQ 测试等。

高级测试用例组，主要包括

迁移扩展测试；

真实客户数据应用测试；

迁移环境上的功能回归测试和性能回归测试。

**测试用例的平台组合分布的选择原则：**

基本测试用例组的测试用例在所有的配置组合上都得测试，但在每个测试阶段， 可以涵盖不同的测试用例。在测试初期，更多地关注于基础软件升级测试、数 据库迁移测试及应用程序迁移测试相关的测试用例，因为这些测试用例是整个 迁移的基础，只有这些测试都正常工作，才能进行其他的验证测试。

扩展测试用例组主要涵盖和应用相关的集成软件的迁移，根据实际需求可以选 择测试或者不测试，测试团队可以和开发团队共同决定扩展测试用例在哪些平 台组合上进行测试，不需要涵盖所有的平台组合。

高级测试用例组测试也是根据具体的测试要求单独计划，只需选择部分平台组 合进行测试。迁移测试团队应该和开发团队或技术支持团队一起根据客户的要 求对于某个具体的测试任务选择合适的测试平台进行测试。

**7.3.2** **迁移测试的一个具体场景**

在迁移测试项目的实施过程中，项目小组接到了一个企业客户的真实场景的迁移需 求，即为客户旧版本的系统环境到最新版本的系统迁移提供一个完整的解决方案。该客户 的旧版本的真实场景如下：

一套开发环境，开发环境上包含了很多客户定制过的数据和程序。

一套单节点测试环境，配置组合为：AIX 5.3 操作系统，IBM <HTTPV6>,DB2数 据库服务器V8, 部署于IBM WebSphere 应用服务器 V6 之上的旧版本的应用系 统 V6。

一套三节点的企业应用的实际运行环境，配置组合为： AIX 5.3操作系统、 IBM [HTTP V6](HTTPV6)、DB2数据库服务器V8、部署于IBM Web Sphere 应用服务器V6 之上的应用系统 V6。其中数据库服务器、IBM [HTTP Web](HTTPWeb) 服务器和 IBM WebSphere 应用服务器分布在三台不同的机器上。

新版本的应用系统V7 支持DB2 V9,IBM <HTTPV7> 及IBM WebSphere应用服务器V7。

小艾在麦习的指导下，根据前面学到的迁移理论，从客户角度出发，完成了这个比较 复杂的端到端的迁移场景的设计。迁移场景如下：

**1.** **在测试环境上进行迁移和迁移测试**

参照本书7.2.1 节完成 IBM <HTTPWeb> 服务器、DB2 数据库服务器，以及 IBM WebSphere 应用服务器升级测试；

安装新版本的应用系统并部署一个测试用的应用环境；

迁移数据库并完成数据库迁移测试：

迁移数据库；

完成垂直迁移测试的初始数据迁移验证；

完成水平迁移测试的数据库结构验证，定制化数据结构验证，商业数据验证。 迁移应用并完成应用迁移测试：

第7章 更新换代，重装上阵

迁移应用；

完成垂直迁移测试的定义的应用迁移测试点；

完成水平迁移测试的应用配置信息验证。

**2.** **在开发环境上进行迁移和迁移测试**

客户定制化迁移测试；

将开发环境上迁移之后的客户定制化程序部署到完成了迁移的测试环境上；

3. 参照7.2.1节，在迁移之后测试环境上进行功能测试和性能测试

4. 在确保上一步功能测试和性能测试都正确的前提下，开始进行生产环境的迁移测试 为了将客户应用宕机的影响降到最低，生产环境的迁移分为以下几步。

在保持旧版本生产环境运行的情况下，根据客户的资源状况选择软件升级方案。 如果客户有足够的资源，新准备一套三节点的系统环境，参照本书7.2.1节分别 完成 IBM [HTTP Web](HTTPWeb) 服务器、DB2 数据库服务器，以及IBM WebSphere应用 服务器迁移及测试。如果客户没有额外的资源，可以在现有的系统环境中，在 各自的节点上分别完成 IBM [HTTP Web](HTTPWeb) 服务器、DB2 数据库服务器，以及IBM WebSphere 应用服务器软件升级及测试。

安装新版本的应用产品。

迁移应用。

将完成迁移的客户定制化程序部署到新的生产环境中。

停止旧版本应用系统的对外服务，完成生产环境数据库的迁移。

启动新版本的应用。

在迁移之后的应用系统上进行功能测试和性能测试。

小艾的场景设计得到了麦习的认可，麦习觉得小艾已比较全面和深刻地掌握了迁移测 试，并且积累了一定的迁移测试的实战经验，已经是一名合格的迁移测试人员了。

从菜鸟到测试架构师——一个测试工程师的成长日记

**7.4** **学习笔记——迁移测试之小艾观**

通过一个阶段的迁移测试学习和实践，小艾又完成了他测试生涯的一个里程碑。小艾 将迁移测试的心得体会总结如下：

迁移测试的主要方法包括垂直迁移测试和水平迁移测试，两者相互联系，互为补 充，水平迁移测试更多的时候可以看成是垂直迁移测试的一部分。

迁移测试不是独立的，与其他测试类型存在很多交集。比如，我们需要在完成迁 移的环境上进行功能测试，性能测试等。因而迁移测试过程中需要各个测试团队 的紧密合作。

迁移测试质量的高低取决于迁移测试计划的好坏。高效的迁移测试计划在于测试 平台的选择和测试用例的定义，我们需要通过有限的测试平台组合以及有限的测 试用例发现尽可能多的问题。

迁移测试一定要考虑宕机时间， 一般大企业/银行一年里对宕机时间的要求是不高 于5分钟。

**参考资料：**

[1]百度百科：

<http://baike.baidu.com/view/292866.html?wtp=tt>

[2]DB2 信息中心：

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.qb.upgrade.> doc/doc/c0023662.html

[3]IBM WebSphere应用服务器信息中心：

<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multi> platform.doc/info/ae/ae/welc6topmigrating.html



第 **8** 章

假如我是客户：面目全非脚 与还我漂漂拳



小艾现在与开发组的同事们相处得很不错，经常去他们的座位看着他们把一个个想法 变成产品功能的一个个组成部分，看上去非常有成就感啊。但是小艾也不免遇到一些用户 的问题，本以为是一只虫子，但仔细看下来却是因为客户没有能够充分了解产品功能而引 发的一些使用误区。如果能够让客户更加深入地了解产品功能和开发过程，这样不是能使 客户更加透彻地熟悉产品吗?尽管平时小艾也会开发一些自动化测试脚本，但在复杂程度 和所用到的技术上，还是不能和产品开发人员同日而语。

小艾把自己的想法对凯文说了以后，凯文笑了：“正要找你呢。多伦多那边有一些客 户化测试的任务，因为原来的测试人员换到其他部门了，需要人手，我看你的编程技术还 可以，就推荐你了。这就能满足你刚才的愿望。”

“客户化测试?这是做什么?为什么要看我的编程能力?”

**8.1** **其实，我是一个演员——如何扮演客户**

凯文的介绍让小艾了解了什么是客户化测试：

所谓客户化测试 (Customization Verification Test), 是针对一个软件产品的客户化 定制能力所做的测试，它侧重于模拟客户使用产品的场景，通过对产品的公开文档、教 程及产品开发接口的使用，按照产品服务部门提供的客户需求，开发相应的功能定制样 例，并撰写相应文档以供客户学习和参考。

“也就是说，做客户化测试，就是由测试人员来扮演客户了?”小艾做了一个扬头甩 头发的动作：“其实，我是一个演员~”

凯文忍不住大笑起来。为了使小艾能够更好地进行客户化测试的工作，凯文向小艾介 绍了常年奋战于客户化测试第一线的测试人员：陈俊中。“小艾，以后你就和陈俊中合作 了，希望你能够在客户化测试的工作中对测试有更多的认知，得到更多的自我提升。”

**8.1.1** **我知道，我不知道：只知道客户能够知道的**

面对陌生的客户化测试，小艾有些不知所措。不过这段时间接触的形形色色的新东西 不少，小艾已经有了门道。他知道， 一切未知和迷茫都会从陈俊中那里得到答案。向陈俊 中请教，是客户化测试的第一步。

“如果你是客户，”陈俊中问小艾，“你会从哪里开始对你购买的产品实施定制工作?” “呃，这个嘛，我想我会先阅读说明文档。”

“没错!”陈俊中兴奋地开始向小艾介绍客户化测试。

通常来说， 一个软件产品提供给客户的除了编译、包装好的安装材料，就是说明文档 了。在互联网时代，软件公司一般会将说明文档以规定的格式放到互联网之上，除了方便 客户随时检索，也方便软件公司及时更新。这些说明文档包括软件版本信息、功能特性描 述、安装/升级指南、操作/管理手册、开发教程、安全信息及参考资料等。

作为软件产品的客户，他们如果想要基于该软件产品做定制开发，就必须通过说明文 档对软件有一个全面且深入的了解，尤其是在开发接口 (API) 方面。所以，为了更好地 扮演客户，客户化测试组的测试工程师必须跟客户一样，只能从说明文档获取开发所需的

第8章 假如我是客户：面目全非脚与还我漂漂拳

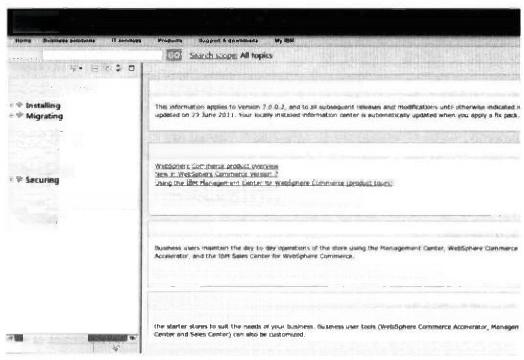
各种信息。也就是说，他们不能去阅读开发设计文档和真正的实现代码。

“这样说来，如果我要对一个新特性做客户化测试，而它的文档还不存在，怎么办呢?” 小艾很纠结。

这正是客户化测试的痛苦所在。客户化测试人员必须想办法尽早拿到相关文档的草 稿，尽管这些草稿在每一个开发周期的开始阶段会很不完整，相关功能的具体实现也常常 被改来改去甚至推翻、取消，而且开发人员对待文档总不是那么上心。但是早一天了解客 户能拿到什么,就多一天时间去做客户化测试的设计。

一般来说，客户化测试是允许查看需求分析文档的，因为既然是模拟客户，那么客 户的需求显然是客户化测试人员应该知道的东西。还有就是解决方案的规格说明书，这 个文档一般是在客户需求基础上，提出的解决方案及相关功能描述、风险、优先级排序等。 另外就是功能测试的计划文档，在这种文档中描述的功能特性肯定会写入给客户的说明 文档中。

最后是给客户的说明文档，这一般由专门的文档编写人员来负责，如图8-1所示，IBM 的最终客户所看到的产品文档一般都会以信息中心 (Information Center) 的形式呈现。在 产品开发过程中，文档编写人员也会将正在撰写或修改的文档提交到一个开发版本的信息 中心，客户化测试人员就可以使用这个信息中心来检索自己需要的内容。

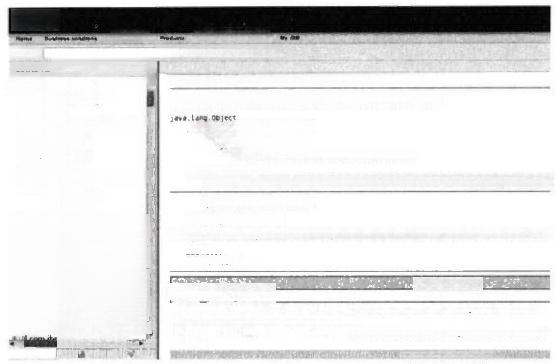


|  |  |
| --- | --- |
| TBM  Seard  Content  Product overview  PLnning  Operating  ◆dministeig  ◆utari  Samples  Developing | WebSphere Commerce Version 7 Infomatien Center  mGeting started |
| ◆Tuning  ◆Troubiecheoting and upport  ◆ Reference Ghosy | S oaim iane |
|  | 2□operating  To leere abo teman tass inyed a nunningsoe, Qosn.teanagngur  伯 custamizing and etendng weusphere Comerce  wensmeremne ons ae srter won thet youcn uw towu aeAppketon awipen cn este |
| a | t ieen atout the inan ts inymesi in estmang wbhpsre Cammare,swe Cinisneaustumiceg watie |

图8-1 IBM Information Center界面

但是信息中心说明文档的开发启动时间一般会比较晚，有时候不得不请开发人员在不 透露开发细节的情况下提前给一个初稿。尤其是API, 如图8-2所示，客户化测试人员必

须基于它来开展功能实现的开发工作。所有这些文档，都需要到开发组和文档组取得。



IBM

rch sog topics

Contents 山 章 ：

Betesncz>WetsghereCon eA>som.bencatmetcE Kceisconto>Clase

**Introduced in WebSphere Commerce Version** [**7.0.0.3Class**](7.0.0.3Class)

Ljova.tleYaj.bists**c**tCtnettcstiten

Lcow.ibw,cosercs.accesscontrot,AcessVector

**AJI Implementesd Interfaces**

**java io.Seriazable,java.l**ang.Coneable,java.iang.Iterabie,java ui.Collection,Java.unUs,jav

publie ctass Accessecter

**extends jevautilVactor**

Thes dass deflnes the cbject retumed by the gtthesources()method ofa command for resource levul accr

**See AJo;**

Seceed Fcm

\*f comibmcommerce atachment

\*com ibmcommerch.attuchment

\* 望com ibmcommrce atadhment comibmcommerce attachment

\* com ibmcommerceatachment comibmcommerce busehelper,

esan

|  |  |
| --- | --- |
| unu Jma.Lm.srin二  Cspacitytncrenent,elesentcount,eleeentData  ModCount | **eld smmary**  **Fielda inherited from ca java,utlL.Vector**  F**lelds inherited frem caes javautl.AbatractLis**t  Conatructor summary |

AYchange Demwen reeases

awetsphere Comimerce AP

Overview summary

Al dassn

How to use ths APIinformation Constant values

B5erialized form

Deprecated AP a Overview tree

com ibmcommerce acessconu 来 Classes

**AccessVector**

B Snteflesource

\*f comibmcornmerceaccessconu

3combmcommerce ACcourtcon

eom.ibmcommerce approvalbe

com ibmconimerce appvoval co

Scomibmcommerce approvlob

Searck

图8-2 Java doc

**8.1.2** **我做，我不做：做客户会做的**

陈俊中交给小艾的第一项任务是撰写一份设计文档，这是小艾从未涉及的。不过小艾 倒是阅读过开发的设计文档，写过测试设计文档。拿着软件服务部门提供的客户的真实需 求，小艾开始翻查资料，慢慢理清头绪。

其实所有的设计文档在很多方面是很类似的。小艾需要：

(1)描述开发目的、设计原则、术语概念、模块图、时序图和状态机图。

(2)将用户需求分解成设计用例；根据设计用例编写客户化定制用例。

(3)描述实现的构思：

用户界面设计；

命令行；

 工具类；

◎ Web Service; 配置文件；

第 8 章 假如我是客户：面目全非脚与还我漂漂拳

数据库结构。

(4)确定需要撰写的帮助文档，例如教程。

(5)描述此设计可能会造成的影响：

产品安装；

配置和管理；

开发环境；

产品安全，包括线程模型、用户验证、权限控制等；

全球化(语言支持);

迁移和兼容。

(6)记录设计注意事项：

此设计的假定和依赖项；

此设计的限制；

此设计已知问题；

其他可选的设计方案；

推荐的测试方法；

将来可能有的对此设计的改进；

知识产权信息；

技术上未确定的部分；

可访问性的支持。

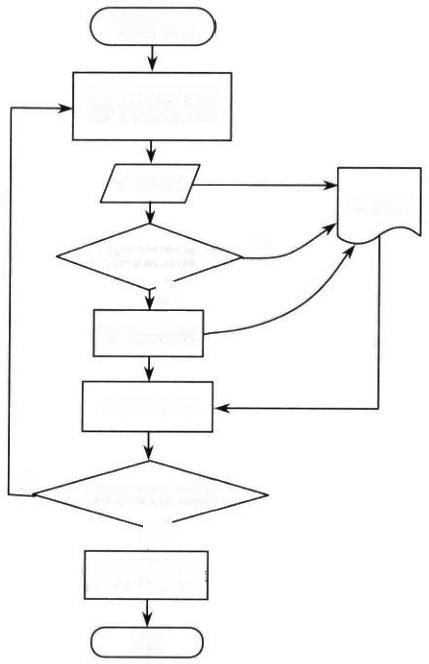
“组长，客户也要写设计文档么?”小艾写得有点头大，毕竟模板要求填写的内容真 不 少 。

“假如你是客户，你会在做开发时做哪些事情?”陈俊中反问道。

“呃，确定需求，然后研究可行的解决方案，详细设计，最后做实现。没错吧?”

所有这些都应该在文档里面记录下来，这是软件工程的基本要求。那么具体设计就必

须有设计文档。即便是做客户化测试，为了模拟真实客户，客户化测试人员也需要进行一 个比较完整的软件设计流程(如图8-3所示)。



提交设计文档

开发人员、测试组长 客户支持人员审阅

审阅意见

审阅记录

否

是否接受意见?

是

修改设计文档

正式审阅会议

否

是否满足批准条件?

*↓是*

提交项目经理批准

结束

图8-3 设计审阅流程

一般来说，客户化测试由4个阶段组成：

(1)确定与研究需求阶段。

(2)设计阶段。

(3)实现与测试阶段。

(4)文档与教程验证。

第8章 假如我是客户：面目全非脚与还我漂漂拳

所有需要测试的客户化场景都需要与软件服务部门共同商讨确定，以确保选中的是客 户最需要的。客户化测试人员将进行客户化实现、提供样例代码和基于客户化场景为文档 团队撰写教程稿件等工作。而文档团队将会把这些教程初稿提交给相关专家审阅，根据审 阅意见对教程进行编辑，并且在最后获得他们的批准。在理想情况下，这些教程还会提交 给功能测试团队或文档团队的测试人员进行测试验证，这也是前面提到的第4个阶段。最 后，相应的教程会被合并到官方文档里面供所有客户参考。

当然在一个产品周期完成之后，客户化的代码资源会被部署到某一个或几个常用的产 品环境上，供功能测试和系统测试团队在客户化的环境下进行回归测试，以保证即使客户 对产品做了定制，既有的功能和性能仍然不受影响。这种测试能够在很大程度上避免由于 扩展接口造成的一些软件缺陷。

为了适应敏捷开发模式，客户化测试需要和产品开发周期的研究/设计阶段、实现阶 段与文档阶段并行进行，这就要求客户化测试人员从最开始就介入到软件服务部门和架构 师的工作中去。

客户化测试人员还需要向开发团队提供使用反馈和提交发现的缺陷报告：

客户化问题；

易用性问题；

功能问题；

 文档问题；

代码问题。

**8.1.3** **我想，我继续想：客户会怎么想**

“实际上，所谓客户化测试，就是想在客户前面，想客户之所想，让我们的产品真正 符合客户的需求。”陈俊中做了一个小结。

客户花钱买了一款软件产品，要让这个产品符合自身企业的需求，很多时候都需要在 原来的基础上进行二次开发。或者修改界面，或者增加功能，或者改动流程，等等。

“那么身为客户的化身的你， 一定要像一个客户一样去思考。比如对于用户注册的功 能，作为一个客户，你会怎么想呢?”陈俊中问道。

从菜鸟到测试架构师——一个测试工程师的成长日记

小艾想了一下，说：“我在很多网站上都注册过用户，无外乎填写用户名、密码、邮 箱等资料吧。我们的产品不是默认都带了这些功能么?”

一般来说，软件公司开发的产品都会拥有某些基本功能。但是，真实客户经常会有特 殊需求。以用户注册功能为例，客户可能希望在搜集基本信息之外，还搜集用户的地理位 置、微博账号、收入层次和消费倾向等信息。作为客户，他一定还会从自己的行业角度出 发来思考问题。一个石油企业可能会对用户的汽车的车型、排量感兴趣，而一个服装行业 的客户则可能对用户的身材数据更有兴趣。不同行业的客户在定制用户注册功能的时候， 要添加的功能点一定是不一样的。

要更好地扮演客户，应当对客户的行业有所了解。所以客户化测试人员需要在一开始 就清楚某些产品特性是来自哪些客户的需求，作为这些客户，在对这些产品特性进行客户 化定制的时候会提出什么样的目标。

**8.2面目全非脚——狠下心来改产品**

小艾测试的产品是一个庞大的Java EE应用，要对它进行客户化定制，就需要使用专 门提供给客户的开发套件来进行开发工作。和客户一样，客户化测试人员是不能接触到产 品源代码的，客户开发套件只提供了编译好的 Java 包。和通常的软件开发流程类似，代 码开发完成以后需要进行白盒测试和黑盒测试。在进行客户化定制开发时，产品的功能和 界面常常会被修改得面目全非……毕竟在开发的过程中，出错是难免的。但是正如古人所 说：“不破不立”,只有迈出了这一步，才有实现的可能。

**8.2.1** **第一脚，学会开发**

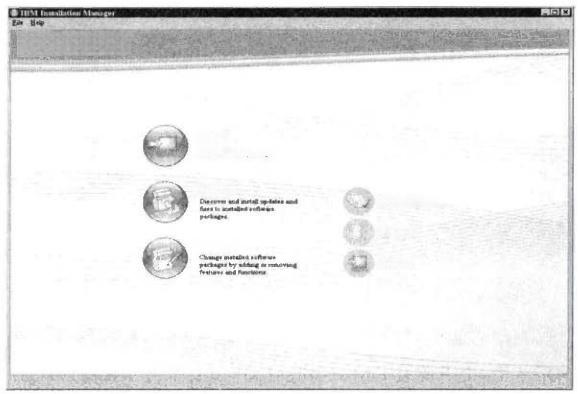
小艾虽然以前也做过一些小的开发工作，但是这还是第一次面对企业级的应用。俗话 说“磨刀不误砍柴工”,一些基础的知识和技能还是需要学习一下。

**1.** **安装开发环境**

IBM 的 Java EE 应用都是使用Rational Application Developer(简写RAD) 来进行开 发的。客户在对产品进行客户化开发也必须使用RAD 。RAD是基于著名的Eclipse 项目和 众多企业级开发插件，同时集成了WebSphere Application Server的企业级开发环境。所以

第8章 假如我是客户：面目全非脚与还我漂漂拳

小艾要做的第一件事情就是安装RAD 7.5(在本书写作时RAD 8.0已经面市，它的安装方 式与RAD 7.5基本一样),如图8-4所示。

abowve

lnstall

lntn pragt

Cptate

Mteogu Leme

ftuck

Modity

lwiastd

图8-4 RAD安装界面

遵照安装的指南，小艾又用安装管理器更新RAD 7.5 到了一个小版本7.5.5(补丁集 和功能增强),这样他的开发环境就安装好了。

(1)开发Java EE 应用

看着全新的界面(如图8-5所示),小艾有点无从下手的感觉：“好复杂!陈俊中，你 简单说说怎么用吧。”

“其实不复杂，跟用 Eclipse 做开发基本一样。RAD 的功能是非常多的，但是我们常 用的也就是创建新项目或者导入新项目、搞定项目依赖、为EJB 生成数据库后端代码，以 及源代码版本控制等。先讲讲项目依赖的事情吧，它是一个项目能否成功编译的关键。”

(2)RAD 构建路径

RAD 7.5的项目元数据(除了在项目根目录下的.project 和.classpath 文件)基本上都 被存放在项目根目录下的.settings 目录下，所以在往版本控制系统里检入一个项目时，一 定要将.project 、.classpath 文件和.settings 目录都包含进去。一个例外是不要包含.settings

目录下的.prefs 文件，因为作为一个大型软件，为了在项目间共享配置信息，使用的是全

局参数设置而非项目参数设置。

从菜鸟到测试架构师——一个测试工程师的成长日记



1e

pcge myacknge:

e aaa ot1

ystm,tprint1”1s W”

王 鞭emla

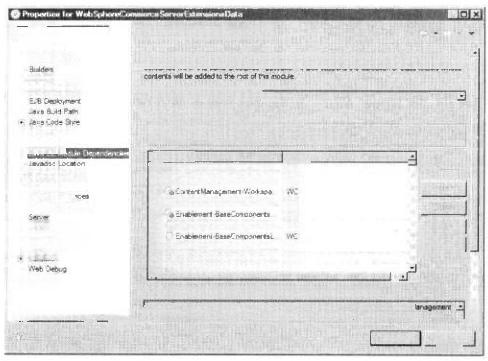
IBscs lm semlTm

0 w

图8-5 RAD 7.5 开发界面

而且在RAD 7.5中，由于使用了MANIFEST.MF 条目来参与管理构建路径(build path), 避免了RAD6 中只使用.classpath 造成的问题，因为在运行时环境里是不会有.classpath 文 件的。在构建路径中有一条 “EAR Libraries”, 它是动态的根据 MANIFEST.MF 文件来生 成的，并且它会根据模块的依赖需求自动包含额外的条目。RAD 7.5通 过“Java EE module dependencies” 这个部分来管理EAR 需要的模块(如图8-6所示),所有修改都会在 “EAR Libraries” 条目里体现出来并且将对应的依赖项记录在MANIFEST.MF 文件。

：习四四四区四四四四



Java EE Module Dependendies

Fesouce

etm re n

CodeCverag

Eeepoe zpicaanpje n

Auslabie deondant JAA

Refewncos0 EBJR wmEIBOut JRa

avaCorple

C Ue Ea JAr (hn EEder JA(owbem

SCMaiogProdutManatetD Cmaog PoductMangenerL

Poa Fe

CertetMngamert Wehsp

EnabienertBanComoonertt

aErabionert &kannCorlenE

EnaiamarBanuCartetE

Ensemet9arees grtef

XDce

Marken CoP (onotrckde certrfbbns ho dsepahetie时

MokwC oc

ok□ Canes

FunDebug Sttng

Sernce Poice

Targted furanes

Vade

IE 性 WC MC

wC

wC

w

wC 南C

s

Doea

hoekett

1ARm

Detaut Pakag

JDECCamecfions

lemka Path

图8-6 Java EE 依赖项设置界面

第8章 假如我是客户：面目全非脚与还我漂漂拳

当然如果项目不是EAR 应用的一部分，那么类路径的解决方式跟普通 Eclipse 的 Java 项目一样需要自己手工添加。

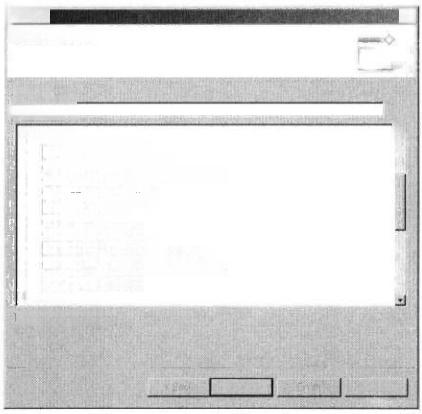
而 JDBC 的驱动不是 EAR 的部分，所以必须人工指定。因此当你导入一个对数据库 有依赖的项目到RAD 7.5 时，必须在导入后手动指定相应的JDBC 依赖，例如ojdbc14.jar。

这样 RAD 7.5就很方便地解决了构建路径对Java 库的依赖问题。

(3)生成数据库后端代码

RAD 7.5 使用物理数据模型 (Physical Data Model) 文件来管理数据库表对象的映射 关 系 。

在 RAD 7.5中创建物理数据模型很简单，在一个 EJB 项目上单击鼠标右键，在弹出菜 单里选择“新建”(New), 然后选择“其他”(Other..), 在弹出对话框里选择“数据”(Data), 再选择“物理数据模型”(Physical Data Model), 如图8-7所示。



DMew

Select a wizard

Wzads

o

Dsa

Data Desgn Project

0 Data Developmern Pnoject

Eustrg FAD6×Dasta Delrion Prooz 国 Phyeca Data Mode

saL Soapbook Psge

SQL Statemert

809cored Pocedure

扣 Ueer-Defned Function

Web Servce Lser Delned Function

靶 WebSphere MO UerDefned Function

SQLJ Apokcatons

Data Acess

厂 ShowA Wzarde.

⑦ Bes

×

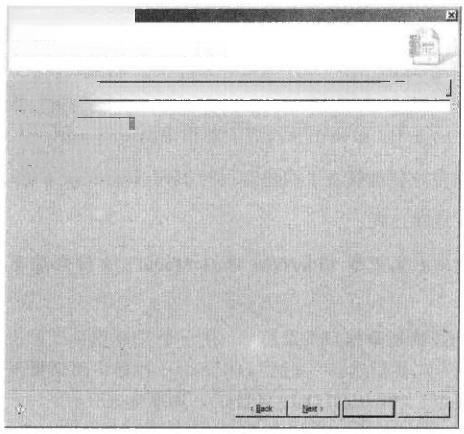
Cancel

图8-7 物理数据模型向导

这时候会有一个新的对话框弹出(图8-8),可以看到目标目录和文件名都已存在，只 需要选择正确的数据库类型即可。单击“完成”(Finish) 按钮，一个.dbm 文件就生成了。

为了映射数据表，有两种方法来创建和修改模式结构， 一种是图形界面的模型工具 (图8-9),一种是在属性视图中直接修改(图8-10)。

从菜鸟到测试架构师——一个测试工程师的成长日记



Dna fhbya Data fod

**ModelFile**

Scesty thabase.verson.and ocanon of ihe new mode fie.

ouremo ouo.NobSsyoComeesno5to n oblod6NETAStasab D62UDeVT2 Bw

Plen ee

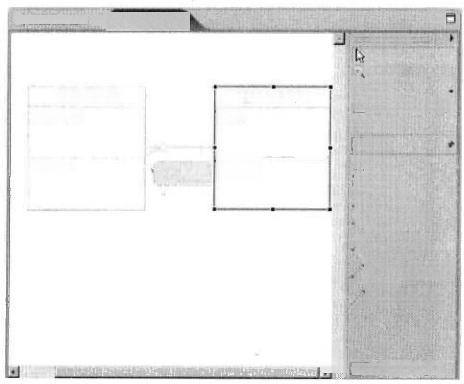
Daabae 2009 口

Yonian 132口 6Cost fio tonoste

C.Cgeate fomeeinsenng

Erah Cancn

图8-8 选择数据库类型



口

Padelte *Sled*

Zoom

te

rtolecusl Popoty *Doyom*

Data

Tabie

阳Me

ldentfang

en kdertlyng Dona

Narrthrg

Marcay

Non tertyng Qne 1o Cne

Vow Relaorahe

Geonee Sapes

databas dm 2·Dorm1×

SAMPLE idFKQ

dato

□

翻 SAMPLE2

id

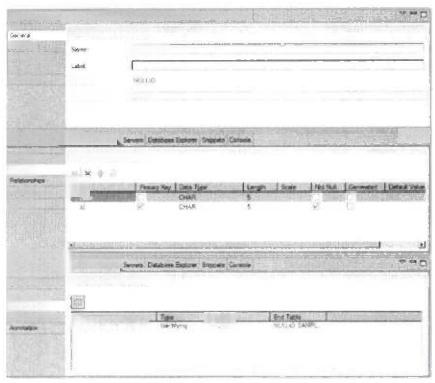
*I dt*

图8-9 数据模型映射工具

( 4 ) 实 战 中 学 习RAD

陈俊中跟小艾说：“最好的学习办法，就是动手去做。只要实际做过几个例子之后， 相信你会对 RAD 提供的功能留下很好的印象。”

第8章 假如我是客户：面目全非脚与还我漂漂拳



Netme adCFuc fe teautL nCw

市 **Tahle>SAru**

Cu

FMPIE

y

Dea

rte

hone 8

C

二 <Tae>sAuPu

\*

Deanrt

men

-

<tabke>suPLE

CLans

r

Do

x

e

e

图8-10 数据模型属性视图

“那么从哪里入手呢?”

“IBM 的开发者网络是个不错的资源，你去找找吧。”

小艾访问 IBM 开发者网络，发现上面真的是汗牛充栋，好文章比比皆是。编辑还提 供了优选的阅读清单',循序渐进地介绍了不少实际应用中能够用到的知识和技能，这让 小艾感觉受益匪浅。

小艾通过开发一个Java Persistence API(JPA)和一个调用该JPA 实体的 Session Bean、 调用Session Bean的 Servlet, 以及调用Servlet 的 Web 页面熟悉了RAD 7.5带给开发者的 强大功能。²

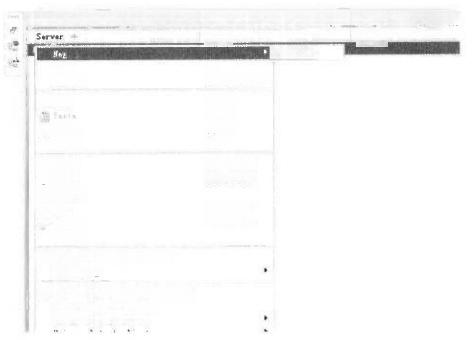
为了把自己做的项目部署起来，小艾需要一个 WebSphere Application Server(WAS) 应用服务器实例。 RAD 7.5自 带WAS 版本7,可以很方便地通过RAD 内置的工具来创建 实例(图8- 11)。



1 <http://www.ibm.com/developerworks/cn/websphere/library/techarticles/0305_issw/recommendedreading.> html(推荐阅读清单：Java EE 与 WebSphere Application Server)

2 [http://www.ibm.com/developerworks/cn/offers/lp/demos/summary/r-radjpa.html(Rational Application](http://www.ibm.com/developerworks/cn/offers/lp/demos/summary/r-radjpa.html(RationalApplication) Developer 7.5 JPA 演示)

从菜鸟到测试架构师——一个测试工程师的成长日记



Prob1ms Tsks 旧 opti

State

誓 Sarver

0pe 五

Shew In

Coyy

其 Delet Renine

如 abur 0zturt

refi1

L9:2

Publih

Clea

Add and Remove Erojets … Monitering

(creste tables and dats soures Fasj s rit

Server configuration

"Seureh 景 rcr/I?Manitor

Statu

Delete

P2

Ctrl+Alt+D

Ctrl+Alt+B

F3

AIt+Shift+Y Ctrl+C

[cnsnale 第 Stvrs

Ctrl+Alt+F

图8-11 创建 WAS 服务器实例

然后通过设置项目的参数把项目添加到RAD 内嵌的WAS 运 行 环 境 中 。

**2.** **使用版本控制系统管理源代码**

“再说说版本控制。为了维护源代码，我们客户化测试小组设置了一台内部版本控制 系统，使用的是Subversion 。”

“我更喜欢用分布式的Git 来管理本地源代码。”小艾说道。

“那也没问题，你可以在自己的工作站上使用 Git 来管理，开发完成后的代码检入到 组里的Subversion 上去。”陈俊中说，“而RAD 本身是支持Rational ClearCase 、Rational Team Concert 和 CVS 的，如果要支持Subversion 或 Git, 就需要自己去下载安装 Eclipse 插件。”

这里简单介绍一下这几个比较常用的版本控制系统。

Rational ClearCase是业界非常优秀的软件版本控制工具和软件资源配置管理工具，在 软件开发过程中，采用 ClearCase 进行版本控制能够实现真正意义上的并行开发，提高开 发效率。同时 ClearCase作为商业软件，拥有不同选项来满足从异地分布式的大中型项目 到小型本地项目的项目资产管理、版本控制、软件构建等的需求。当然由于其复杂度较高， 一般小型项目还是会选择更轻量级的版本控制系统。



1 <http://www.ibm.com/developerworks/cn/rational/08/1104_agrawal/index.html> (使用IBM Rational

Application Developer V7.5 和 WebSphere Application Server V7.0 构建Java EE 应用程序)

第8章 假如我是客户：面目全非脚与还我漂漂拳

Rational Team Concert(RTC) 是 IBM 基于Jazz 开发的一款为软件交付和团队协作开 发提供“集成工作项目”、“源代码控制和构建管理”等支持的开发环境。Jazz 是一个可伸 缩、可扩展的团队协作平台，用于集成软件生命周期中的各项任务。Rational Team Concert 基于客户机/服务器体系结构，它包含集成式工作项、源代码控制功能、报告功能及构建 管理支持功能。此产品为敏捷的团队提供了具有高度协作性的开发环境，并且此环境已针 对小型和中等规模团队进行优化。

CVS(Concurrent Versions System) 是老牌的版本控制系统了，几乎所有的Linux 发行 版都会附带 CVS, 而 Eclipse 默认就支持CVS 。但是由于其设计上的不足及C/S 的特性， 在使用上还是会有许多不方便，例如网络访问不方便、对文件夹操作不方便等。

Subversion 是为取代CVS 而设计的，它基本保留了CVS 的功能，而对其饱受诟病的 缺点进行了大刀阔斧的进化，例如文件夹的版本化、元数据的版本化操作——这使得各种 原子化提交能够实现。文件锁——当多个用户对同一个文件编辑时发出警告，基于 <HTTP> 的 WebDAV/DeltaV 协议——能够方便地与 Web 服务器集成而不会因为防火墙挡住了外部 的访问能力，等等。一些第三方软件开发者开发了各种工具来辅助操作Subversion,例如 Eclipse 的插件Subclipse 和 Subversive,Windows 系统下文件管理器的插件TortoiseSVN 等。

Git 是 Linus Torvalds为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控 制软件。Git 是一个分布式的系统，也就是说每一个客户端都包含了版本库所有的数据， 你可以使用Git 离线工作，然后把自己的更新在线更新到中心服务器上。由于Git 不是像 Subversion 那样按文件存储元信息，而是使用元数据，而且它使用了SHA-1 哈希算法来保 证内容完整性，所以它的内容完整性要优于Subversion 。并且Git 的设计使得它的分支十 分廉价而高效，这非常适合模块化开发和分布式开发。

“RAD 7.5 跟普通 Eclipse 一样，可以安装各种 Eclipse 插件。所以我们一般都安装 Subclipse 来检入和检出代码。”

**3.** **扩展产品**

在 RAD 上部署了产品开发包之后(不同的产品部署内容和方式都有所区别，这里就不详 述了，具体可参阅 WAS的在线文档),小艾可以基于现有产品代码进行客户化定制开发了。



1 Subclipse的网址：<http://subclipse.tigris.org/>

从菜鸟到测试架构师——一个测试工程师的成长日记

“为了让客户更好地定制我们的产品，我们的产品从设计上就预留了各种扩展点。”陈 俊中介绍道。“这些扩展点一般是Java 的接口、各种配置文件及JSP 文件等。客户化产品 的工作，其实就是实现Java 接口，并通过修改配置文件来用客户实现覆盖默认产品实现。”

“听上去跟普通的Java 开发没什么两样。”小艾说。

“基本上是这样。但是需要注意的是，这是对已有产品进行修改而不是从头进行开发， 所以必须保证你的改动不会对产品其他功能造成影响。比如你做的这个功能改动，要在用 户注册时提供微博账号绑定的功能，为了加入微博账号信息而直接给数据库里的用户表添 加新字段，其实是一种不太好的做法，一方面可能对功能和性能造成破坏，一方面用户如 果已经有了大量用户数据，再添加字段风险很大。对于新数据，一般都是新加扩展表。”

“对，这就是你给我的设计文档提的审阅意见，我正准备修改呢!”

小艾的任务听上去其实挺简单，就是让用户能够绑定微博账号。因为市面上微博产品 非常多，为了保证教程的通用性，他选择了国际最流行的Twitter。为了方便开发，他还特 意申请一台部署在多伦多的虚拟工作站作为开发主机。

他首先在数据库中新建了一个表，用于保存用户 ID 和用户的 Twitter 用户账号、 Access Token与对应的密钥。

然后他修改了用户注册页面的JSP,使用户可以在这两个页面提交绑定Twitter账号的 请 求 。

为了处理这个请求，他实现了用户注册的 Java 接口，这个实现同时继承了已有用户 注册实现的类，并修改配置文件让产品使用这个新的实现类。这个类通过使用一个开源的 OAuth 库'实现了 Twitter 的 OAuth 认证过程²。在用户完成OAuth 授权取得用户的Twitter 用户账号、Access Token与对应的密钥之后，这个类会将相关信息保存到新建的表中。

这样用户在产品里的账号就跟 Twitter 账号绑定起来了。

在RAD 上测试小艾的代码很简单，只需要重新发布(Publish)WAS 的实例，然后访 问用户注册的页面，看看新的 Twitter 绑定功能是否好用即可。



1 <http://twitter4j.org/>

2 <http://tools.ietf.org/html/rfe5849>

第8章 假如我是客户：面目全非脚与还我漂漂拳

当然，如果作为一个完整的功能，那么小艾做的东西还远远不够。比如作为绑定了 Twitter的账号，如何在网站上体现出Twitter 的功能(发tweet, 展示tweet), 网站如何利 用绑定 Twitter 账号的用户来实现网站推广功能等。作为真实客户，可能还需要不止支持 一种微博服务，例如国内就有新浪微博、腾讯微博……这些功能就是客户需要自己在小艾 的范例上继续实现的了。

**8.2.2** **第二脚，部署代码**

如前所述，在 RAD 上测试完客户化代码之后，还需要将代码部署到真正的产品使用 环境中进行验证。

**1. 在产品环境的** **WAS 上部署应用**

WAS提供了 Web界面的部署方式和命令行部署工具。小艾的“作品”被他分别导出 为EJB JAR包 、Web 应用的WAR 包和普通JAR 包，他决定先使用 Web界面进行部署测 试，然后为了以后的回归测试方便，再撰写一个用命令行进行部署的脚本。

Web 部署界面如图8-12所示。



Weta

L1

图oudei Aconie

p ,wax, d mai od and inee

Apaetbdted

Fglt p en

upl n d sls th teedplaten

田sente

\*p d medl

8Pc

用sacority

BE \*t 田 ranitbins

Bun n\*

团 mnons ndTen 团 Teble

[RE

田 fursic i地 Ш иоог

lac infie

pte y~ng ntelepplehion uhivfi t mess tob d dde4.

ify thu huh

Lecu fie tem

图teve

日 ehcabm

Ramet fi wim

A

**图8-12** **WAS** **的** **Web** **部署界面**

(1)首先备份已安装的应用程序或模块(这是一个好习惯):

转至管理控制台的企业应用程序页：在控制台导航树中，单击应用程序>应用程序

类 型 >WebSphere 企业应用程序。

导出应用程序到 EAR 文件或导出文件(指应用程序中的文件)。选择要导出的应用 程序，然后单击导出或导出文件。导出操作将保留绑定信息。

(2)然后更新原有的产品应用程序：

在“企业应用程序”页上选择应用程序后，单击更新。显示“正在准备应用程序更新” 页。在指定要上载和安装的EAR、WAR、SAR 或JAR 模块下：

确保要更新的应用程序指的是要更新的应用程序。在应用程序更新选项下，选择 要更新的已安装的应用程序、模块或文件。

如果选择了替换整个应用程序或替换或添加单个模块选项，单击“下一步”显示 更新应用程序文件的向导，然后完成在更新向导中的步骤。

需要注意的是，安装步骤已将新版本的绑定信息与旧版本的绑定信息合并。如果新版 本具有应用程序工件(例如 EJB Java 命名和目录接口(JNDI) 名称、EJB 引用或资源引 用)的绑定，那么这些绑定将成为合并的绑定信息的一部分。如果新绑定不存在，那么从 已安装(旧)版本获得绑定。如果旧版本中不存在绑定，而且如果默认绑定生成选项已启 用，那么默认绑定将作为已合并绑定信息的一部分。你可以选择是忽略旧版本中的绑定， 还是新版本中的绑定。

最后单击“完成”按钮。

更详细的参考信息可以从 WebSphere Application Server Library得到。

**2.** **使用ANT** **实现自动化部署**

为了在以后的回归测试中自动化地部署，小艾需要写一个自动化部署脚本。在这个脚 本中，他需要用到WAS提供的wsadmin 脚本工具。

wsadmin 脚本工具引入于 IBM WebSphere Application Server V5。它是一个使用脚本 语言的非图形化管理命令接口。wsadmin 支持Jython 和 Jacl 两种脚本语言。wsadmin 工具 通过使用下列 WebSphere 特定对象，对基本的脚本语言进行扩展：



1 <http://www.ibm.com/software/webservers/appserv/was/library/>

第8章 假如我是客户：面目全非脚与还我漂漂拳

AdminConfig—— 用于运行配置命令。

AdminApp—— 用于执行应用程序管理。

AdminControl—— 用于运行操作命令。

AdminTask—— 用于运行操作命令。

通过使用 wsadmin 工具，管理员能够替代正在运行的管理控制台来配置、部署和管 理 WebSphere Application Server的多种元素。而且，用户能够使用脚本语言的固有功能(开 发脚本)将wsadmin 命令组合使用，从而自动化用户自定义的管理过程。

wsadmin 工具是一款强大的工具，用户通过它能够实际调节 WebSphere Application Server 所提供的每个调节器，当然前提是用户知道调节哪个调节器和如何调节该调节器。 这两个知识点都比较难以掌握，在使用过程中也需要查阅相关帮助文档。¹

通过 wsadmin 工具配置 WAS 服务器与部署应用，除了需要对服务器资源和应用信 息进行配置，还需要指定 wsadmin.bat ( 或 wsadmin.sh) 所在的目录， deployment manager 所在的host, 以及连接 deployment manager 时所需的用户名和密码。

为了避免每次运行Jython 脚本都需要在命令行中指定以上的参数，我们选择 ANT 工 具作为运行工具。我们将以上这些参数配置成 ANT 脚本的 property 元素，并在运行 Jython 脚本时作为参数传入：

<target name="init">

<!--The path where wsadmin.sh(.bat)is in -->

<property name="wsadmin.path"value="C:/WAS/bin/wsadmin.bat"/>

<!--Connect to which WAS server host to update target application --> <property name="wsadmin.host"value=" <127.0.0.1"/>>

<!--The user name to run wsadmin on a secured server -->

<property name="wsadmin.user"value=""/>

<!--The password to run wsadmin on a secured server -->

<property name="wsadmin.pw"value=""/>

<!--Main script to run -->

<property name="wsadmin.script"value="main.py"/>

<!--Configuration file path -->



1 <http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/> welc6topscripting.html(对应用程序服务环境进行脚本编制，基于WAS 7.0 版本)

从菜鸟到测试架构师———个测试工程师的成长日记

<property name="wsadmin.config"value=""/> </target>

在 build.xml 中，将wsadmin.host 、wsadmin.user 和 wsadmin.pw 作为wsadmin.bat ( 或 wsadmin.sh) 的参数，将配置文件的路径作为 Jython 脚本的参数，运行 wsadmin 工具和 Jython 脚本：

<target name="wsadmin">

<property name="args"

value="-host ${wsadmin.host}

-user $wsadmin.user}

-password ${wsadmin.pw}

-lang jython

-f $wsadmin.script}

-config ${wsadmin.config}"/>

<exec executable="${wsadmin.path)"failonerror="true">

<arg line="${args}"/>

</exec>

</target>

通过 ANT 脚本，可以灵活地运行Jython 脚本，从而完成 WAS 服务器的配置和应用 部署。假定我们将脚本保存在 C:\wsadmin 目录下，配置的 xml 文件为 sample.xml, 那 么运行 wsadmin 脚步的Ant 命令将是：

ant -f C:/wsadmin/build.xml -Dwsadmin.config=C:/wsadmin/sample.xml

小艾在他编写自动化 ANT 脚本的过程中，积攒了一些好用的代码，例如启动、停止 WAS 服务(需要用到antcontrib 包)。在这个脚本里，首先导入了antcontrib, 然后对脚本 运行环境的操作系统类型进行了判断，这个判断用于确定所使用的 shell 脚本后缀是.bat (Windows) 还是.sh(Unix 和 Linux); 然后在启动 WAS 服务的宏定义中，先使用serverStatus 命令取得 WAS 服务是否已经启动的信息，再据此决定是否执行 startServer 命令；停止WAS 服务的宏则直接运行stopServer 命令，因为即使之前服务已经停止了，它也只是提示服务 已停止；最后启动、停止 WAS 服务的宏都有一个对应的ANT target 定义以方便使用命令 行调用。



<taskdef resource="net/sf/antcontrib/antlib.xml"/>

<!--Determine the command suffix -->

<condition property="cmd.suffix"value=".bat">

<os family="windows"/>

</condition>

第 8 章 假如我是客户：面目全非脚与还我漂漂拳

<condition property="cmd.suffix"value=".sh">

<os family="unix"/>

</condition>

<target name="startServer"

description="target for startServer macrodef">

<fail unless="was.home"status="1">The WAS home directory is missing </fail>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <fail | unless="was.profile.name" status="2">The | WAS profile | name | is |
| missing</fail> |  |  |  |  |
| <fail | unless="was.server.name" status="4">The | WAS server | name | is |

missing</fail>

<fail unless="was.user"status="8">The WAS global security user name is missing</fail>

<fail unless="was.password" status="16">The WAS global security

password is missing</fail>

<startServer washome="$was.home}" instance="${was.profile.name}" server="${was.server.name)"

globalsecuser="${was.user)"globalsecpass="${was.password}"/>

</target>

<!--Start the WAS Server at the location with config specified.--> <macrodef name="startServer"

description="starts the WAS Server">

<attribute name="washome"/>

<attribute name="instance"/>

<attribute name="server"/>

<attribute name="globalsecuser"/>

<attribute name="globalsecpass"/>

<sequential>

<exec dir="@{washome}/profiles/@{instance}/bin"

executable="@{washome}/profiles/@instance}/bin/serverstatus${cmd.suffix}" outputproperty="serverStatusOutput">

<arg line="@{server) -username @fglobalsecuser} -password

@{globalsecpass)"/>

</exec>

从菜鸟到测试架构师———个测试工程师的成长日记

<if>

<contains string="$serverStatusOutput}" substring="

ADMU0509I"/>

<then>

<exec dir="@{washome)/profiles/${was.profile.name}/bin"

executable="@{washome}/profiles/@instance}/ bin/startServer${cmd.suffix}">

<arg line="@{server}"/>

</exec>

</then>

</if>

</sequential>

</macrodef>

<target name="stopServer"

description="target for stopServer macrodef">

<fail unless="was.home" status="1">The WAS home directory is

missing</fail>

<fail unless="was.profile.name" status="2">The WAS profile name is

missing</fail>

<fail unless="was.server.name" status="4">The WAS server name is

missing</fail>

<fail unless="was.user"status="8">The WAS global security user name is missing</fail>

<fail unless="was.password" status="16">The WAS global security

password is missing</fail>

<stopServer washome="${was.home)" instance="${was.profile.name}"

server="${was.server.name)"

globalsecuser="${was.user}"globalsecpass="${was.password)"/>

</target>

<!--Stop the WAS Server at the location with config specified.--> <macrodef name="stopServer"

description="stops the WAS Server">

<attribute name="washome"/>

<attribute name="instance"/>

<attribute name="server"/>

<attribute name="globalsecuser"/>

第8章 假如我是客户：面目全非脚与还我漂漂拳

<attribute name="globalsecpass"/>

<sequential>

<!--Exit status of 0 or 246 on UNIX /-10 on Windows(meaning it's already stopped)are valid -->

<exec

executable="@{washome)/profiles/@{instance}/bin/stopServer${cmd.suffix}"> <arg value="@{server}"/>

|  |  |
| --- | --- |
| <arg | value="-username"/> |
| <arg | value="@{globalsecuser}"/> |
| <arg | value="-password"/> |
| <arg | value="@{globalsecpass}"/> |
| </exec> </sequential>  </macrodef> |  |

**8.2.3** **第三脚，联合测试**

这里的联合测试指的是功能测试组 (FVT) 在部署了客户化代码环境里的回归测试， 以及系统测试组 (SVT) 在部署了客户化代码环境里的性能测试。

小艾在完成了他的部署脚本之后，接到的任务就是帮助功能测试组和系统测试组部署 客户化的产品测试环境。由于在这个产品周期里几乎每一个新功能都有客户化定制测试， 小艾所做的只是其中一个功能，因此就有不止一个客户化定制项目需要被部署。小艾要做 的就是把其他同事所编写的部署文件跟自己的合并起来，统一部署。

合并部署文件的工作是个体力活，小艾需要比对各个客户化定制所添加和修改的代码 是否有冲突，例如有的时候两个项目修改了同一个配置文件或者 Java 类，那么他就需要 从源代码将这些修改合并然后重新编译、打包，而每个客户化定制项目的部署脚本也需要 合并调整。小艾花了整整一个星期才完成最终的合并部署包。

在将合并部署包部署到功能测试服务器和系统测试服务器上之后，功能测试组和系统 测试组就在各自服务器上执行一套精简的回归测试。

对功能测试组来说，他们需要验证产品的原有功能是否在客户化定制后的环境中仍然 正常。通常客户化定制在扩展或覆盖某项产品功能后，不应该破坏产品原有的其他功能。 但是如果产品在设计上有缺陷，导致产品某些功能模块的独立性不好，对其他模块产生了 依赖，那么当客户化定制了它依赖的模块后，就可能破坏这些功能。通过在客户化定制环

从菜鸟到测试架构师———个测试工程师的成长日记

境上的功能回归测试，我们能确保很多基本功能的问题被及时发现。

类似地，系统测试组要保证产品的性能、可靠性即使在被定制后仍然满足指定的标准。 这项回归测试用于发现定制代码的性能问题和产品在系统层面的扩展问题。

通过联合测试，小艾他们的代码终于通过了考验，可以作为参考发布给客户了。

**8.3还我漂漂拳——让产品对用户更友好**

在客户化定制测试中，小艾发现了好几个产品设计上和功能上的缺陷，这些缺陷或者 对客户化定制工作制造了障碍，或者是产品本身的功能问题。那么客户化定制测试如何对 产品设计提供更好的帮助呢?小艾带着这个问题和陈俊中进行了一番讨论。

**8.3.1** **协助完善产品设计**

“我发现我们的产品几乎所有地方都是遵照面向对象设计模式来设计的，这提供了 很好的定制可能性。但是，我们的实现中有些部分耦合程度太高， 一些模块之间的依赖性 太强，导致这些部分扩展起来很困难。”小艾说道。

一般软件企业的软件产品在设计和开发过程中都会尽量采用各种设计模式来提高可 复用性和可扩展性，但是由于时间的推移， 一个产品可能已经经历了很多次更新，许多老 的功能模块不符合新的需求，但是它们的一些接口或模块又被新的功能模块依赖，这样层 层叠叠的补丁打下来，不可避免地会出现系统僵硬、脆弱、复用率降低和耦合度过高的问 题。可是作为一个企业级的产品，软件企业必须为所有的客户考虑。许多购买了老版产品 的用户会要求升级到新版本，而他们的一些客户化定制必然会一起升级。为了平滑地升级， 新版的产品设计就不可能使用激进的重构方法。

“嗯，这的确是一个大问题。我们不能在每个产品周期开始的时候都重起炉灶，那么 以前的开发的遗产必然既是财富，又是阻碍。”

对于有历史的产品来说是这样，对于新的软件来说，更需要在一开始的架构设计上就 下足功夫。可是人们很难看到几年后会有什么样的功能需求，那么只能在设计产品时就坚 持开放-关闭原则 (Open-Closed Principle,OCP), 即对扩展开放，对修改关闭，而且在所 有将来的新增设计上，也要坚持这个原则。

第8章 假如我是客户：面目全非脚与还我漂漂拳

小艾高兴地说：“我知道，设计模式里讲到过。好像还有依赖倒转原则、里氏代换原 则。”

还有合成/聚合复用原则、接口隔离原则和单一职责原则等。这些都是面向对象的软 件设计里面非常重要的设计原则。IBM 的产品也一直遵循这些原则。当然，在具体实现时， 不可能每一个程序员都完全遵守它们，有时候程序员迫于短期压力会使用一些捷径，而这 些捷径会破坏设计原则，这时候就可能出现这种缺陷：软件模块之间耦合度过高，导致新 的功能无法插入。

“那么我们做的客户化定制测试能够在这方面提供什么帮助呢?只是报告缺陷么?” 小艾觉得有点沮丧。

其实自从在客户化定制测试这项工作中也实施敏捷过程以后，客户化测试人员从产品 周期的一开始就可以对产品的设计提出自己的看法了。因为客户化测试人员的视角是从客 户化出发，关注的焦点就是软件设计的可扩展性。虽然客户化测试人员不被允许阅读产品 设计文档，但是从产品规格说明书和Java Doc中可以了解到相关的设计思路，然后可以将 这种设计思路与前面提到的软件设计原则相比照，看看它有没有背离的地方。

“是的。其实在测试过程中也可以通过编程了解到产品设计是否符合设计原则，因为 我们的代码也是需要遵照这些原则来写的。”小艾举一反三。

还有，就是在联合测试中合并各个客户化定制项目的代码的工作中，应该也可以看到 各个新功能的代码是否坚持了这些原则。假如两个客户化定制项目的代码有无法解决的冲 突，要么是客户化测试人员的客户化定制代码背离了设计原则，要么是产品代码有问题。

**8.3.2** **客户化联合测试工作的痛苦和解决方案**

在开发客户化代码和合并各个功能的客户化代码部署文件的过程中(见8.2.3节),小 艾跟其他客户化测试的同事都碰到了同一个问题，那就是有时候他们需要修改的产品文件 同时也在被开发部门的开发人员修改，例如一些配置文件或者一些JSP 页面。这是因为在 敏捷开发的模式下，客户化测试工作是和新功能的开发工作同时进行的，而客户化测试人 员不能像开发人员一样使用版本控制系统来和开发部门操作同一个文件。这就导致了客户 化测试人员在开发过程中需要时时检查自己修改的源文件是否已经被开发人员修改了。

受制于人是很难受的，小艾他们经过各种调研和尝试，找到了一个还算不错的解决方 案：用一个Ant 脚本来自动将小艾他们的客户化代码插入到源文件中。这个方案基于两个

条件：

(1)源文件中存在能够定位客户化代码位置的字符串，且该字符串不会被开发人员修 改(一旦被修改，也就只能对应地修改Ant 脚本了);

(2)每次运行这个 Ant 脚本时都取到最新版本的源文件。

下面是一个 Ant 脚本的例子，它使用Ant 的 replaceregexp 任务将 soap.client.props 文 件中的com.ibm.SOAP.requestTimeout 原值注释掉，然后设置为0。

<?xml version="1.0"encoding="UTF-8"?>

<project name="Modify\_soap.client"default="Target\_Modify\_soap.client">

<target name="Target\_Modify\_soap.client">

<replaceregexp file="C:\WebSphere\AppServer\profiles\demo\

properties\soap.client.props"match="com.ibm.SOAP.requestTimeout=180"replace="

#com.ibm.SOAP.requestTimeout=180${line.separator}com.ibm.SOAP.requestTimeout =0"flags="gm"/>

</target> </project>

当然，要完成这样的事情，也可以使用Perl 、Python 或任何一种你熟悉的脚本语言。 使用这种方式，小艾他们基本从追踪源文件修改的工作中解放出来了。

**8.3.3** **撰写详细易读的客户化教程**

在整个编写代码和部署脚本的过程中，小艾都按照陈俊中的要求，对自己的每一项工 作记录了工作笔记，其中有详细的代码内容，也有安装步骤，还有缺陷报告及小艾的一些 感想。

“记录工作笔记是为了在撰写客户化定制教程时有第一手的参考材料。”陈俊中说，“好 记性不如烂笔头，很多时候我们只需要将我们的工作笔记稍作修改和修饰，就可以完成一 篇很好的教程文档。”

“好笔头不如一截图。”小艾更进一步，在自己的工作笔记中还保存了很多自己开发、 测试时的截图，这样他看到图就会想起自己当时的工作场景。而这些截图在他撰写教程文 档时，也成了最好的素材。

IBM 的客户化定制教程有标准的模板，包括：

第8章 假如我是客户：面目全非脚与还我漂漂拳

教程概述，描述此教程的背景信息和目标；

学习目标，具体描述教程学完后可完成的目标；

所需时间，对满足教程技能要求的人来说，完成此教程需要花费的时间；

难度等级，分初、中、高级；

教程受众，此教程对哪些人有用；

系统需求，完成此教程的软、硬件配置；

技能要求，完成此教程的人需要掌握的技术；

课程简述，此教程包含几个前后相关的课程；

课程详述：

课程目标描述。

课程详细步骤，这里是教程的精华所在，需要将每一个具体的细节用简洁易懂 的语言描述出来，相关的代码或配置也需要粘贴进来。

在完成了教程初稿之后，小艾将它提交到项目组让开发者和客户化测试组组长进行审 阅。审阅通过后，客户化测试组组长会找另一个对小艾项目不熟悉的测试人员按照教程一 步一步实现一遍，直到确保教程的步骤和内容没有问题。最后文档开发人员在对其进行编 辑后合并到产品的帮助文档库中。当产品最终发布时，小艾的教程也随着帮助文档库一起 发布了，这个时候，他的工作才算最终完结了!

**8.4学习笔记——客户化测试之小艾观**

完成这次客户化定制测试任务后，小艾有了如下感受：

客户化定制测试是游走在测试与开发之间的一种特殊的测试类型。它是开发，也是 测试。

作为开发，客户化定制测试需要遵循所有的软件设计原则，需要遵守软件开发流程， 完完整整， 一个也不能少。

作为测试，客户化定制测试不仅需要验证自己的功能，还需要保证能够与其他功能很

从菜鸟到测试架构师——个测试工程师的成长日记

好地集成，功能测试和系统测试能够在客户化定制的产品环境上获得通过。

要做好客户化定制测试，除了学习各种开发技能，还需要做好文档工作，每一步都富 有成就感，每一步都值得记录下来。

**参考资料：**

[1]推荐阅读清单：Java EE 与 WebSphere Application Server:<http://www.ibm.com/> developerworks/cn/websphere/library/techarticles/0305\_issw/recommendedreading.html

[2]Rational Application Developer 7.5 JPA 演示：<http://www.ibm.com/developerworks/> cn/offers/lp/demos/summary/r-radjpa.html

[3]使用 IBM Rational Application Developer V7.5 和 WebSphere Application Server V7.0 构建 Java EE 应用程序： <http://www.ibm.com/developerworks/cn/rational/08/1104> agrawal/index.html

[4]Subclipse 的网址：<http://subclipse.tigris.org/>

[5]Twitter4j 项目：<http://twitter4j.org/>

[6]OAuth 1.0 协议：<http://tools.ietf.org/html/rfc5849>

[7]WAS 资料库：<http://www.ibm.com/software/webservers/appserv/was/library/>

[8]对应用程序服务环境进行脚本编制，基于 WAS 7.0版本：<http://publib.boulder.ibm.> com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc6topscripting.html

**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

第9章 黎明之前最后的冲刺：成品测试

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。

第9章 黎明之前最后的冲刺：成品测试

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

第9章 黎明之前最后的冲刺：成品测试

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

第9章 黎明之前最后的冲刺：成品测试

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

从菜鸟到测试架构师——一个测试工程师的成长日记

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。

第9章 黎明之前最后的冲刺：成品测试

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

第9章 黎明之前最后的冲刺：成品测试

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，

从菜鸟到测试架构师——个测试工程师的成长日记

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。

第9章 黎明之前最后的冲刺：成品测试

**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。

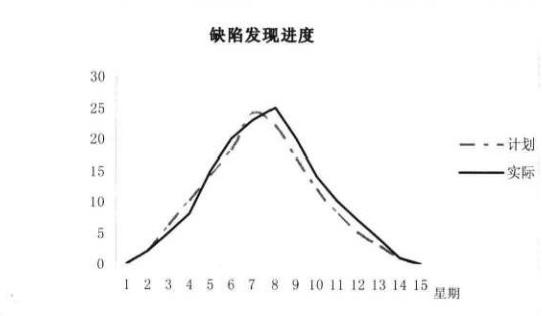


图9-1 缺陷发现计划进度和实际进度的比较

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。

**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](http://www.stevemcconnell.com/articles/art04.htmSoftwareQualityatTopSpeed,SoftwareDevelopment) [Development](http://www.stevemcconnell.com/articles/art04.htmSoftwareQualityatTopSpeed,SoftwareDevelopment),<August> 1996

第9章 黎明之前最后的冲刺：成品测试

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：

从菜鸟到测试架构师— 一个测试工程师的成长日记

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。

第9章 黎明之前最后的冲刺：成品测试

**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。

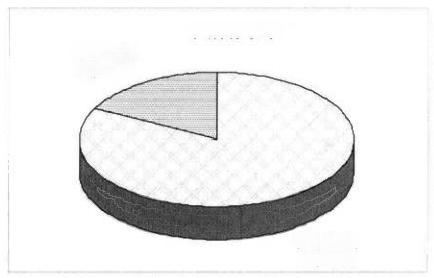
其中，在关闭状态和取消状态下的缺陷称为完结缺陷，除此之外状态的缺陷称为活跃 缺陷。项目结束时，理论上所有缺陷都应处在关闭状态或取消状态。当然由于时间原因， 还存在极少量开启状态下的缺陷被延缓到补丁版本或新版本里修复。我们把关闭状态缺陷 和延缓缺陷称为有效缺陷。而把取消状态下的缺陷称为无效缺陷。在测试过程中，应尽量 提高有效缺陷比率，避免开出无效缺陷。

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。



**缺陷有效率**

无效缺陷， 18,17%

有效缺陷， 85,83%

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。

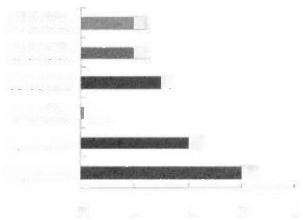
不可重现：是指所开缺陷无法在相同场景中重现。有时可能由于网络，机器等问 题而使测试案例无法正常运行，但重新测试很多遍也无法重现当时的问题。这种 情况会建议取消缺陷， 一旦同样问题再次发生，就需重开缺陷。

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。

第9章 黎明之前最后的冲刺：成品测试

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。

**无效缺陷分析**



产品局限 [2](#bookmark367)

未来功能 [12](#bookmark368)

设计如此 [3](#bookmark369)

不可重现 [1](#bookmark370)

用法错误 [4](#bookmark371)

重复缺陷 [6](#bookmark372)

0 2 4 6 8

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。

从菜鸟到测试架构师———个测试工程师的成长日记

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。

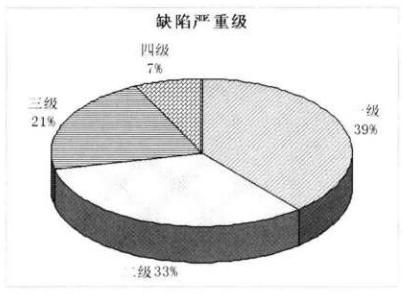


图9-4 缺陷严重级分析

**3.** **按功能分析缺陷**

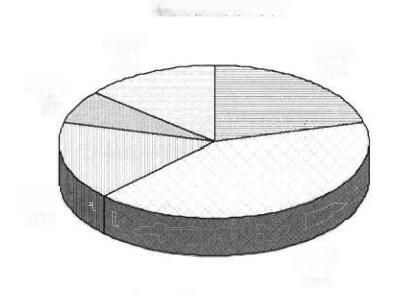
我们一般会看不同功能上发现多少缺陷。有些功能逻辑比较复杂，涉及的方面较多。 这种情况所发现缺陷比例就会稍大。有些功能较简单，和其他功能没有太大交集，所发现 缺陷就会相应较少。

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。

第9章 黎明之前最后的冲刺：成品测试



**缺陷按功能分类**

14% 功能1

21%

功能4

7%

功能3

17%

功能2

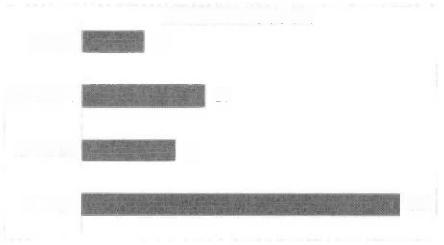
41%

功能5

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。



安装测试缺陷修复时间

>20天 [10](#bookmark373)

11～20天 [20](#bookmark374)

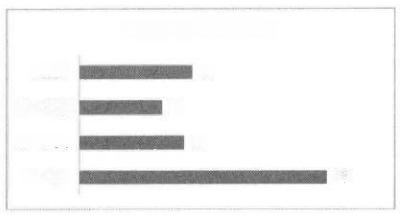
6～10天 [15](#bookmark375)

0～5天 [52](#bookmark376)

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。



性能测试缺陷修复时间

>20天 [15](#bookmark377)

11～20天 [11](#bookmark378)

6～10天 [14](#bookmark379)

0～5天 [33](#bookmark380)

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。

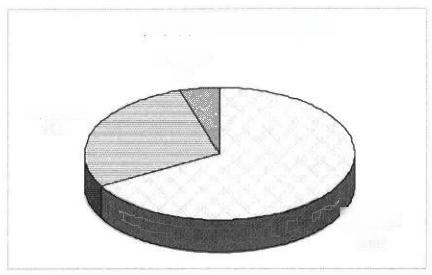
另外，性能测试需要安装和客户相近的测试环境，所以也会不可避免地发现安装缺陷。 它的测试环境的复杂度也会帮助安装团队扩大安装测试范围。

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。

第9章 黎明之前最后的冲刺：成品测试



**性能测试中不同缺陷类别**

安装缺陷

5%

功能缺陷

28%

性能缺陷

67%

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。

**操作系统缺陷分析**

操作系统无关=Windows I Solaris Linux BAIX

3%7 5% 2%

3%-

87%

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。

从菜鸟到测试架构师——一个测试工程师的成长日记

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。

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

第9章 黎明之前最后的冲刺：成品测试

**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.

第 **10** 章

**缝缝补补又三年**



产品交付使用已经有大半年的时间了。凯文对客户报告的问题及产品组内部测试发现 的问题进行分析之后，认为是时候推出产品补丁包了。由于小艾对该产品较为熟悉，凯文 向测试组要求小艾作为项目经理整体负责该补丁包0.1版本的测试；与此同时，凯文为小 艾介绍了补丁包测试项目的专家指导肖瑞。在小艾和肖瑞的带领下， 一场轰轰烈烈的扫荡 漏网虫子的运动展开了。

**肖瑞对补丁包及补丁包测试的简单解读：**

产品补丁包，即Fix Pack, 是产品发布出去之后由产品开发团队定期发布的补丁集 合。补丁包测试是针对补丁包进行的测试，该测试类型可以看做是一个简单的产品的测 试，与产品测试相关的所有内容基本都需要涵盖。

**10.1补丁包测试简介**

补丁包测试其实可以看成是一个小而全的产品开发过程，其中涉及新功能的界定、需 求优先级的分析，以及补丁包最终的发布。补丁包测试在整体上依托于主版本产品一致的

从菜鸟到测试架构师——一个测试工程师的成长日记

测试方法论、测试框架及测试工具，所以该测试类型对测试人员在测试技能的广度上有很 高的要求，这样才能满足补丁包测试方方面面的需求。

**10.1.1** **目标驱动的补丁包测试**

由于小艾对于补丁包测试是一个新手，所以他有一箩筐的问题需要向肖瑞咨询。肖瑞 由凯文引荐过来的第一天，小艾就拉住肖瑞从产品补丁包发布的目的开始学起。

一般来说，某一个产品发布之后，产品开发团队会定期发布产品补丁包，主要目的有 两 个 ：

(1)按时、定期发布的补丁包可以帮助客户获得持续提高的产品质量和用户体验。这 种提高对于一些商用软件来说是非常重要的。

(2)由于补丁包包含的内容来源于已经被客户发现的问题及未被客户发现的潜在问 题，应用补丁包将有效帮助其他的用户规避遇到类似问题的风险。

举个例子，某客户选择某电子商务产品搭建了一个在线购物的商店。在一些重要的时 间节点，如中国传统的春节或者西方传统的圣诞节和黑色星期五，人们会进行大量的网上 采购来准备节日。对于商家来说，一年当中三分之一强的营收可能来自于这短短的一个月 时间，为了保证在线购物网站(商店)能够安然地度过这一阶段而不出问题，并为商家带 来最大化的收益，客户往往会希望能应用最新的产品补丁包集合。从客户的角度，最新的 补丁包涵盖更多、更全的补丁，应用这一补丁包可以帮助客户最大化地避免在销售旺季由于 在线商店自身的原因对该季的销售额产生重大影响，因此客户在销售旺季来临之前往往会 向产品团队寻求最新的产品补丁包，并据此提前制定生产环境的升级及后期的测试计划。

听完肖瑞的解释，小艾仔细地在笔记本上记录了以下内容。

补丁包测试最大的目的在于能给产品带来持续的质量提升。质量是一个产品的生 命，所以补丁包测试的驱动之一往往就是产品开发团队和售后支持团队通过对当前已知 问题的分析和评估，共同选择在一个最合适的时机推出产品的升级补丁包。

考虑到应用补丁包进行系统升级往往需要客户的生产环境停机，而且应用了补丁包 之后需要客户预留一些时间进行测试，以确保补丁包的应用不会对客户已有的业务功能 及系统的性能带来负面的影响，所以补丁包发布的频率和时间点也是一个很重要的需要 提前计划的要素。

第10章 缝缝补补又三年

**10.1.2** **补丁包测试在产品生命周期中的意义**

通过肖瑞的介绍，小艾对补丁包测试的目标有了初步的了解。以前小艾一直认为产品 发布出去之后就没有开发团队的事情了，开发团队的专注力要完全转向下一个新产品的研 发，但是现在看来完全不是这样，他深深地意识到以前对补丁包测试的重视程度真的不够。 对已发布的产品持续进行质量和功能的完善与提高也是产品开发团队日常工作的一项内 容，这一点必须要从根本观念上有所改变。

听着小艾在自言自语，一旁的肖瑞笑了笑，“这是很多开发团队普遍存在的一个思维 误区。前面我们是从客户的角度来理解补丁包测试的重要性，即使从产品整个生命周期的角 度来讲，补丁包测试也扮演着重要的角色，这一点同样容易被忽略或者得不到很好的重视。”

补丁包测试在产品生命周期中的位置和意义可以从以下几个方面考虑：

**1.** **最重要的是它能帮助客户解决实际的问题**

在整个产品的生命周期中，补丁包版本扮演着极其关键的角色。客户购买了某个产品 后，通过自身的使用可能会发现产品的一些问题；于此同时，其他购买该产品的用户在使 用过程中也可能会发现类似的或者其他新的问题。这些已经发现的问题或者潜在的还没 有被其他客户发现的问题就如不定时炸弹一般，随时会直接影响到该客户利用该产品搭建 的生产环境带来的营收。

**2.** **其次，它帮助持续提高产品的质量**

持续产品质量的提高所能带来的好处是多方面的。对产品团队来说，概括起来有以下 两个方面：

(1)能极大地减少维护该产品所需要的开销。所有的产品团队都需要安排额外的资源 以维护已发布的产品，处理客户提交的问题并给出解决方案。

一个同样的问题可能被不同的客户在同样的业务场景碰到。假如没有定期补丁包的发 布，产品团队需要花费M\*N(M: 从接受到处理完一个客户问题所需要花费的时间；N:



1 <http://www.ibm.com/developerworks/cn/webservices/ws-agile/index.html?ca=drs-cn-1214>

● 273 · /

遇到类似问题的客户数量)的成本来维护此单一问题。这里维护成本的节约主要体现在如 果已知问题被发布的补丁包所涵盖，其他应用了该补丁包的客户在其业务运行中发现类似 问题的几率将会大大降低甚至完全避免，产品团队计划在产品维护上的开销就可以得到有 效的减少。

(2)能让产品开发团队抽出更多的精力放在新产品、新功能的研发和设计上，来满足 更多、更广的市场需求。

**3.** **最后，从长期目标看，它有助于在客户中建立良好的产品口碑，有助于培养客户** **忠诚度**

在一个充满竞争的商业环境中，客户会为了追求最佳的经济回报而改变产品购买决 定，所以增加客户黏性的一个很重要的点就是要提高客户满意度。如何实现这一目标呢? 很简单，提高质量，少出问题，让客户用得放心，用得省心!

**10.1.3** **补丁包需涵盖的内容及来源**

定期发布的补丁包里究竟应该包含哪些内容呢?小艾带着疑问来请教肖瑞，以便对补 丁包测试有更深的认识。

主版本为了保证产品在激烈的市场竞争中把握先机、挺立潮头，往往根据市场反馈来 决定需要支持的最新的业务功能拓展和新技术应用；而补丁包需要涵盖的内容，最主要的 还是客户发现的问题及内部在使用过程中可能发现的问题。已经被某一客户发现的问题存 在最大的几率被其他的客户发现，因此，将这类问题的解决方案通过补丁包的形式集中发 布，能够帮助更多的客户避免遇到类似的问题。

补丁包需要持续地对产品的易用性、性能、可靠性进行增强。这里需要强调的是文档 的增强，这往往是开发团队最容易忽视、对客户最有价值的部分。

另外，通过补丁包将对业界新发布平台的支持发布给客户。客户非常有可能期望自己 的生产环境能用到最新发布的操作平台。对于产品而言，对新平台的支持存在如下两种情况：

(1)新平台的支持不需要额外的开发工作，但是需要全部通过补丁包测试或者将部分 补丁包测试中已经计划的测试用例在新平台上执行来完成在新平台的测试覆盖。

(2)第二种情况更加直接一点，也就是新发布的平台产品不支持，这需要另外安排开

第10章 缝缝补补又三年

发人员和测试小组针对新平台可能引入的新功能和新增强有针对性地进行开发和测试。正 式发布对新平台的支持需要等补丁包的测试整体完成和发布出去之后，也就是说，客户必 须安装新的补丁包才能获得对新平台的支持。

最后，通过补丁包将产品功能增强版本发布给客户。此部分产品需求并不会经常发生， 但是类似的反馈却常常存在。客户除了抱怨产品已有功能中存在的问题，还常常要求产品 团队提供产品不支持而他们自身的业务迫切需要的功能。对于这种矛盾，产品团队通过合 适的途径了解客户的需求，并判断该功能的增强可以惠及绝大多数客户的时候，可以考虑 将特殊的产品功能升级添加到补丁包当中。

补丁包产品对新平台及功能增强版本的支持也是产品核心竞争力的一部分。说得直白 一点，客户在确定采用何种产品来满足自身需求的时候，往往同时考察多种类似的产品， 因此，产品所能支持的业界主流平台的范围和广度，以及产品所能提供的功能点与客户不 断变化的需求之间的吻合程度，直接决定了产品的竞争力。换句话说，提供竞争对手产品 所没有的平台支持和功能，即使是在补丁包当中，也是赢得市场的一个很有力的武器。

**小艾的笔记本里对补丁包可能涵盖内容的记录：**

(1)已经被客户发现的问题；

(2)对产品易用性、性能和可靠性的持续增强；

(3)对产品支持平台的拓展和延伸；

(4)客户或者市场要求的产品功能的增强。

**10.2产品补丁包测试的基本流程**

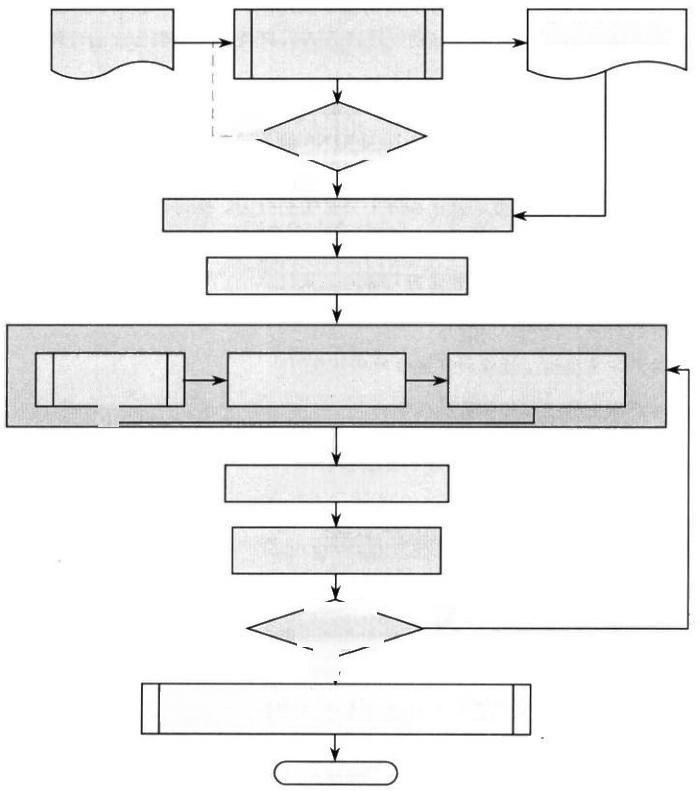
经过一周左右的磨合，基本情况都了解了，小艾很满意，至少自己在补丁包测试方面 已经有了一些知识储备。但是小艾感觉到要把产品补丁包0.1版本做出来并成功发布，仍 然有很多工作要做。

最近半个月产品售后支持部门已经给产品开发团队转过来超过10个客户发现的问题， 基本上都和半年前发布的产品相关，而且这个数字还在持续不断地增加。考虑到之前已经 解决了超过20个客户发现的问题，小艾意识到是时候将累积的问题做成一个补丁包发布 给客户了。

从菜鸟到测试架构师——一个测试工程师的成长日记

但是，真正到了要开始的时候，小艾还是觉得无从下手。问题其实很简单，三个问题： 该做什么(What)? 在什么时间段做 (When)? 该如何着手开始做 (How)? 小艾觉得 还是应该再去向肖瑞讨教一下。

肖瑞给了小艾一个补丁包测试的流程图(如图10-1所示),上面详细罗列了针对补丁 包测试需要在每个阶段完成的工作以及分解的工作步骤。具体的工作包括如下几个方面：



累计问题集合分析

否 是否需要 发布补丁包

是

制定补丁包测试计划

正式执行开始前准备

测试执行

不

补丁包的成品测试

补丁包发布审核

是否可以 发布补丁包

是

1)补丁包发布

2)项目收尾：缺陷分析和经验总结等

项目结束

测试状态记录 和汇报

需要解决问题列表

问题跟踪和解决

累计问题集合

否

图10-1 补丁包测试基本流程图

第10章 缝缝补补又三年

(1)确定补丁包的内容；

(2)制定补丁包测试计划；

(3)补丁包测试准备；

(4)补丁包测试执行；

(5)补丁包成品测试；

(6)补丁包发布；

(7)项目收尾。

小艾很清楚，客户不希望等待问题的解决，因为这意味着营业额的丢失，所以他们要 求产品开发团队能够帮助他们快速定位并解决问题；而产品部门需要维持并增加客户的信 任，没有信任就不能长久地赢得客户，对产品团队来说也是一种销售额的损失。因此，小 艾带领大家挽起袖子、热火朝天地开始操练起来。

**10.2.1** **确定补丁包里需要包含的内容**

首先需要明确补丁包里需要包含哪些内容，因为后续的计划和执行工作都必须依赖于 此。根据肖瑞向小艾介绍的补丁包测试包含的4类内容，小艾花了几天时间做了一些功课 并找到肖瑞，把自己的想法和理由简单陈述了一下。

根据产品发布出去半年时间内所收集到的问题报告和相关信息，如下内容可以作为该 补丁包的备选内容。

**1.** **已经被客户发现和暴露出来的问题**

根据小艾所提供的数据，到目前为止，售后支持部门已经接到了客户发现的产品缺陷 40个左右，这些缺陷会直接或者间接地给客户的生产环境带来破坏，导致功能工作异常甚 至宕机。毋庸置疑，这40个左右产品缺陷的解决方案必须包含在即将发布的补丁包中并 具有最高优先级。

**2.** **对产品易用性、可拓展性、可定制性和可靠性的增强**

小艾发现，除了售后支持部门收到的直接和产品功能相关的40个缺陷问题之外，售

从菜鸟到测试架构师——个测试工程师的成长日记

后服务部门还接收到了10个左右的问题报告，而这些问题并不直接和产品的特定功能模 块关联。小艾、肖瑞及产品架构师对这10个非功能的问题报告做了进一步的分类。

(1)5个问题来源于客户抱怨产品的易用性不是很理想，也就是产品比较难用。即使 是对于经过了专门培训的业务部门人员，产品内在的流程与逻辑实现和他们实际业务场景 的常规化表现也或多或少有出入；另外有客户抱怨用了该产品后感觉工作效率还不如之前 用手工的方式处理业务。

(2)3个问题来源于客户抱怨产品的可定制性不是很理想，需要进一步增强。客户在 基于产品做定制开发的时候发现了诸多局限性，而且与定制开发相关的文档太少，不能满 足实际的需要。

(3)2个问题与文档相关。客户发现发布在网络上的产品信息和相关辅助文档并不足 以帮助客户找到需要的内容，因此内容不完整是一个方面，另外一个可能的原因就是文档 的信息架构需要进一步完善。

**3.** **对产品所支持平台的拓展和延伸**

有一个客户曾经向产品的售后支持团队咨询产品是否支持最新的Google Chrome 浏览 器。这个需求确实没有包含在半年前产品发布时系统的浏览器支持列表里，但是随着 Chrome 的市场份额继续加大以及在互联网世界中使用的持续普及，产品对Chrome 浏览器 的支持必须被加到产品开发所需要考虑的列表中。

**4.** **客户或者市场要求的产品功能的增强**

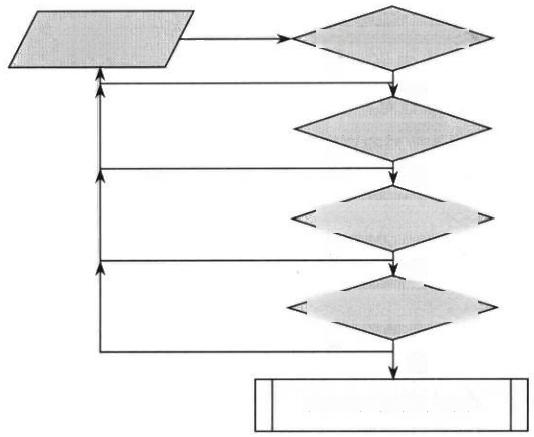
有一个客户建议将一些中国市场的业务需求通过新功能点的形式加到产品当中，这样 可以极大地降低产品定制给他们带来的困扰及开销。与此同时他们相信，这部分功能不 仅仅是中国客户所期望的，全球各地的客户都将有类似的需求来进一步增强产品本身的 功能。

“很好，”听完了小艾对四类客户发现问题的总结和分析，肖瑞很满意，“总结得很全 面，但是显然在我们计划的这个补丁包里面，做不了这么多的事情，因此我们必须要有所 取舍，有重点，有选择。”

“那么选择的标准是什么呢?我想要做的事情这么多，但是老板给我的资源却非常有 限，怎么才能把资源利用最大化呢?”小艾问道。

第10章 缝缝补补又三年

一般情况下，评估某一需求是否可以被加到补丁包里的最根本的原则说起来很简 单，那就是不能破坏客户已有的功能实现，包括产品发布时的功能实现及客户针对产品 的拓展接口做的定制实现，如图10-2所示。



不破坏客户已有的功能 否

是

对客户的商业价值

否

是

开发部门能否按时完成 否

是

测试部门能否按时完成 否

是

产品补丁包开发测试计划

产品补丁包候 选内容集合

图10-2 产品补丁包删选流程

肖瑞很有耐心地给小艾讲解了很长的时间，包括选择的标准及如何界定的一些实际的 例子，这都让小艾收获颇丰。

**小艾的笔记本里对如何对补丁包内容进行选择的相关记录：**

大前提或者说基本原则是不能破坏客户已有的功能实现，包括产品发布时的功能实 现或者是客户针对产品提供的拓展接口做的定制实现。

简单来说，如果功能点A 的实现方法和展现方式在产品发布时已经被清晰的界定， 或者随同产品发布的可定制文档已经清晰地界定可以拓展实现的方式，那么任何在补丁 包里的改动都不能对该功能点的实现或者可能的客户拓展实现做任何的覆盖，因为当客 户部署了补丁包之后，之前对该功能点的定制实现有可能被覆盖。

概括来说，评估某个需求是否可以被补丁包所包含时，需要考虑以下几点：

该需求实现对已有客户的意义和重要性，对潜在客户的意义和重要性；

产品的开发部门能否按计划实现需求的功能设计、代码开发并完成单元测试； 产品的测试部门能否按计划实现需求的功能测试、性能测试及所有可能需要的 其他测试类型，也就是说是否有能力对该新加功能需求实现完整的测试覆盖。

接下来的一周，经过和售后服务部门、开发部门和测试部门的协商和沟通，小艾获得 了更多的第一手资料来帮助他对补丁包内容进行筛选。通过和肖瑞的进一步沟通，小艾确 定了补丁包0.1版本中需要涵盖的内容：

(1)35个客户反映的和产品功能相关的缺陷问题。其中有5个问题由于产品部门对客 户承诺将在补丁包0.1版本中加以解决，因此具有最高优先级。

(2)2个客户反映的和产品可定制性相关的问题。有超过5个客户要求在这个特定的 功能点实现更加宽泛和灵活的产品功能拓展来满足特定的业务场景的需要。

(3)增加对Chrome 浏览器的支持。经过对市场研究机构发布的浏览器市场占有率的 评测报告，Chrome 浏览器的市场占有率已经从主版本发布时低于10%飙升到25%,所以 越来越有必要尽快实现对 Chrome 浏览器的支持。

(4)由于该补丁包的开发测试周期较短，经过和开发部门沟通，决定当前的补丁包版 本中不加入任何功能增强点或者新功能。

补丁包的内容确定后，小艾根据肖瑞的建议，安排了一个包括售后服务部门、开发部 门和测试部门相关人员参加的会议，终于把补丁包0.1版本的内容列表确定了。

一个成功项目的第一步完成了!

**10.2.2** **制定好计划是关键**

确定了该补丁包需要涵盖的内容后，接下来的工作就是制定一份可执行的补丁包开发 测试计划，用肖瑞告诉小艾的话说，“好的计划决定了该补丁包是否能够按期、高质量地 发布给客户”。

“问题的关键在于，什么需要计划?计划的粒度如何掌控?”一开始小艾就把一个很 大的问题抛给了肖瑞。

事实上，补丁包开发测试的计划阶段对于整个过程是至关重要的，可以说开发、测试

第10章 缝缝补补又三年

期间需要做的所有事情都需要考虑并计划在内，包括潜在的风险、对上游问题的可能的依 赖、对下游问题的影响、当前存在的问题、如何解决或者规避问题等。

为了更加清晰地向小艾解释如何做好测试计划，肖瑞为小艾列出了项目计划步骤列 表。具体的步骤清单如下：

(1)确定补丁包测试的内容和分类。补丁包测试计划往往按照内容的分类进行详细 描述和定义。

(2)明确补丁包测试的相关内容的开发和测试的工作量，工作量的估计必须依赖于 对需要解决问题的背景和相关信息的掌握，同时要依赖于测试人员和开发人员的专业知 识和技能。

(3)明确补丁包测试项目中对外部的依赖关系和负责人。

(4)确定补丁包测试需要涵盖的回归测试的范围。

(5)确定补丁包测试需要涵盖的平台和浏览器。

(6)确认补丁包测试所需要应用到的相关软件的版本覆盖范围。

(7)设计和确定补丁包测试相关软件版本、平台、浏览器和测试内容的搭配矩阵。

(8)风险估计，确定风险应对方案和负责人。

(9)存在问题分析，设定解决时间和负责人。

(10)根据(1)~(9)确定补丁包测试具体的时间和进度计划安排。

小艾看到这么详细的列表觉得目标似乎清晰了不少，整个测试计划的轮廓逐渐浮现在 脑海。但是他很清楚，很多时候做事情不是依靠轮廓便可以完成的， 一个成功的测试人员， 需要了解的是每一个细节。他开始针对列表中的每一条的细节开始思索。

**1.** **补丁包测试中回归测试的范围**

回归测试在补丁包测试中很重要，因为我们需要依赖它来保证在补丁包中解决的问题 或者引入的新的功能增强不会对系统已有的功能点造成破坏，如果做不到这一点，补丁包 发布后客户部署补丁包就得不偿失了¹。因此，如何界定回归测试的范围非常重要，最直



1 <http://baike.baidu.com/view/106720.html?fromTaglist>, 百度百科。

接的方法就是寻求专家的帮助。

为了合理地界定回归测试的覆盖范围，需要和负责特定产品模块的开发人员和模块测 试专家进行详细探讨。由于补丁包对客户具有重大意义并会产生重要影响，因此一般情况 下，对于存在风险的地方，在流程上都会遵循专家的建议，进行必要的回归测试。但是在 有的情况下(如时间和资源都非常紧张),需要从全局的角度给模块开发人员和模块测试 专家一些建设性的意见或者建议，来帮助他们做出合理的回归测试的覆盖范围界定。

**2.** **关于软件版本、平台、浏览器和测试内容的搭配矩阵**

做过开发或单元测试的开发人员都有类似的经验，有些功能点是操作系统、数据库敏 感的，有些是和特定的软件版本相关联的，有些是对浏览器敏感的，因此各个功能点与测 试敏感点的选择与搭配是补丁包测试计划中非常重要的一个环节。下面的例子说明了具体 应该怎么做。

假设需要测试的功能点 A 是对软件版本和测试平台都敏感的功能点B 的更新或者增 强。经过沟通，当前补丁包测试中需要涵盖的应用软件服务器包括 IBM 的 WebSphere Application Server和 Oracle 公司的 Web Logic 应用服务器。与此同时，补丁包测试选择了 操作平台AIX 和 Windows 2008作为主平台进行测试。因此，对于功能点B 的改动，补丁 包测试必须包含两个不同的操作系统和两种不同的应用服务器的搭配。通过不同的操作平 台和应用服务器的组合得到以下的测试矩阵：

(1)功能点A+AIX 操作平台+IBM WebSphere Application Server

(2)功能点A+AIX 操作平台+Oracle Web Logic

(3)功能点A+Windows 2008 操作平台+IBM WebSphere Application Server

(4)功能点A+Windows 2008操作平台+ Oracle Web Logic

**3.** **关于测试计划的审阅流程**

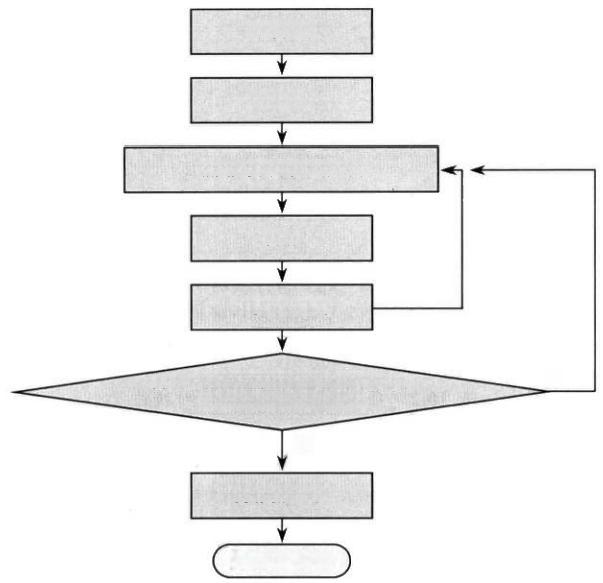
小艾对此不甚清楚，为此他特意翻找资料，终于找到了用于参考的流程图(如图10-3 所示)。

在测试计划审批的流程中，尤其需要注意以下几点：

(1)要有一个清晰的审阅人名单列表。如果不清楚，需要寻找相关的职能部门经理以

第10章 缝缝补补又三年

便确认各个方面的领域专家。



计划撰写

准备审阅

相关人员审阅/重新审阅

更新

后续问题跟进

是否所有的审阅人的建议或者意见都 得到了答复并得到认同

是

提请最后审批

计划审批通过

否

图10-3 补丁包计划审批流程

(2)预留足够的时间给审阅人，以保证计划能够得到很充分的审阅。

(3)可以考虑通过离线交流或者组织审阅会议的形式来尽快地用最有效的方式在不同 的审阅人之间就某个特定的问题达成一致。

(4)所有的更改在计划被正式提交进行审阅后，都必须有记录来描述类似于基于某人 的审阅意见修改计划的哪几个部分。

(5)在计划提交最后审批之前要保证所有的必要审阅人的意见和建议都到了很好的回 复并得到了他们的同意。

又经过一周的努力工作，计划的草稿基本完成。根据与开发部门和测试部门达成的意 见，小艾绘制出了该补丁包项目的里程碑列表，如表10-1所示。

|  |  |
| --- | --- |
| **表10-1补丁包项目的里程碑列表** |  |
| **里** **程** **碑** | **计划完成时间** |
| 完成客户反馈问题解决方案的设计、实现和单元测试 | 8月1日 |
| 完成客户反馈问题解决方案的功能验证 | 9月15日 |
| 完成客户反馈问题解决方案的回归测试 | 9月30日 |
| 完成客户反馈问题解决方案的性能测试 | 10月10日 |
| 完成补丁包的安装测试 | 10月10日 |
| 补丁包的成品测试完成 | 10月25日 |
| 补丁包正式发布 | 10月30日 |

除此之外，小艾还与项目团队成员一起分析并明确了项目当前存在的问题和潜在的风 险，并做出了适当的风险评估和对策准备，这些都清晰地写入补丁包开发测试计划中。

项目存在的问题及应对措施如表10-2所示。

**表10-2项目存在的问题及应对措施**

|  |  |  |
| --- | --- | --- |
| **问** **题** **描** **述** | **计划如何解决以及执行人** | **计划解决时间** |
| 存在5个客户反馈问题描述不清晰， 对后期开发对应的补丁和完成测试有影响 | 尽快和售后支持部门一起将问题明确化 执行人：售后支持部门负责人 | 07/25 |
| 测试机器还差5台AIX的服务器没有 到位 | 临时向项目B的团队借4台AIX服务器尽快开始环境的准备 执行人：小艾 | 07/31 |
| 补丁包安装测试人员还差一个没有 到位，计划到位时间大概在8月中旬 | 协调补丁包测试中安装测试和功能测试的执行计划，以弥 补8月前半月的资源空缺  执行人：小艾 | 07/28 |

**项目潜在的风险及应对措施如表10-3所示。**

**表10-3项目潜在的风险及应对措施**

|  |  |
| --- | --- |
| **风险描述** | **应对准备措施** |
| 新的团队成员的加入，需要时间培训和掌握相关技 能，可能会影响执行进度 | 和各部门的职能经理一起为新成员建立培训计划并指定工作导 师以便他们尽快熟悉掌握流程和相关技能 |
| 性能测试的周期较短，存在可能完不成的风险 | 需要细粒度的跟踪性能测试的执行状态，如果存在问题，可能 需要尽快向性能测试部门的经理要求更多的测试资源的支持 |

接下来的第二天，完成了计划初稿的小艾申请了进入补丁包开发测试计划的审批流

第10章 缝缝补补又三年

程，产品部门负责整体质量保证的高级经理正式批准了补丁包开发测试计划。

到此为止，从小艾接手做这件事情已经过去了3周时间。小艾深切地感觉到前期的工 作太重要了，完全和自己初期的想法不一样，而且所耗费的时间也远超过自己当初的计划。 但是正如肖瑞说的，计划是做好一件事情的关键所在，至少所做的这一切已经为后续的工 作打下了坚实的基础。万事俱备，只欠东风了。小艾和她的团队已经准备得差不多了，正 式的战斗马上就要打响了!

**10.2.3** **执行开始前要做好准备工作**

过去的几周非常忙碌，通过和不同项目团队成员的合作，小艾已经基本明确了补丁包 项目的开发测试内容并且已经制定了开发测试计划。离团队计划的正式执行开始时间还有 一周，在此之前，机器、开发人员、测试人员都还不能完全到位，换句话说，还有一周的 时间测试才能完全展开。

小艾评估了一下项目当前的状态，由于项目组存在不少的新人，而且整体看周期比较 短，项目进度的压力比较大，所以他琢磨着能否利用正式开始执行前的一周做一些事情。 肖瑞对小艾的想法非常赞许并给予了建议：“在实际的操作中，我们可以很好地利用项目 正式开始前的一段时间，做好准备工作以便为后面的正式执行做好铺垫。这仍然需要一个 计划，尽快让团队预热起来。”

通过和肖瑞的沟通及小艾自己的理解，在小艾的笔记本上又留下来这么一段记录：

补丁包开发测试的准备工作，简单来讲可以包含如下几个方面：

(1)遵循测试计划，在测试机到位后，将测试机的宿主操作系统安装上。

(2)鼓励测试人员对计划进行通盘的解读和了解。

(3)如果开发人员在解决一个特定问题时发现该问题比较严重，需要及时对问题进 行初步的定位分析，并和售后支持部门沟通问题本身的描述是否到位、是否遵循了代码 版本控制的模块。

(4)测试组长可在这期间和测试人员进行充分的沟通，以期对测试工作的具体安排、 可能的对开发或者其他功能团队的依赖关系等尽早明确，做到心中有数。

(5)需要安装的软件或者软件需要的补丁包，可以在准备期间由专人下载，并上传 到项目组的文件服务器里供整个团队使用。

从菜鸟到测试架构师——一个测试工程师的成长日记

**1** **0** **.** **2** **.** **4** **测** **试** **开** **始** **及** **状** **态** **跟** **踪**

所有的计划和前期准备都完成了，补丁包测试的计划和准备工作已经就绪，正式补丁 包测试的执行在这一天开始了。

到位的前期工作使得真正的执行变得很轻松是小艾最真切的感受。从测试执行的掌控 看，良好的测试计划很好地帮助小艾调度、管理测试的执行，与此同时，肖瑞也提醒小艾 要重视测试执行的状态汇报和问题跟踪。

在补丁包测试执行开始后小艾还有一系列的事情要做：

(1)在测试计划指定的操作系统和安装相关指定软件的环境中安装补丁包。

(2)在安装补丁包的同时需要验证补丁包安装向导，如果发现问题，需要报告缺陷 给相应的文档开发的工程师负责解决。

(3)根据测试用例的描述，对包含进补丁包的客户反馈问题或者其他的一些拓展功 能点进行功能或者性能测试。

(4)按照回归测试的测试计划和安排，根据测试计划所规定的平台和软件版本，进 行相对应的回归测试。

(5)如果客户反馈问题或者问题通过验证这两者的逻辑关系没搞清楚，那么测试人 员需要关闭该问题或者客户反馈问题对应的缺陷，然后通过文档记录测试结果并标记成 测试通过。

(6)如果是老的测试用例，需要在自动化测试脚本管理系统里找到对应的脚本执行 测 试 。

(7)如果是新加的测试用例，在完成手工的测试后由对应的测试工程师在脚本开发 工程师的帮助下完成自动化脚本的编写。

(8)如果在测试过程中发现问题，需要在缺陷管理工具中开出新的缺陷，并由对应 的测试工程师负责和开发工程师沟通解决问题。

(9)项目组成员在项目的末期需要按照补丁包安装文档的详细步骤进行补丁包的安 装和卸载测试，从而对补丁包安装文档进行验证。在测试过程中发现的一些问题或者最 佳实践需要添加到安装文档中，测试人员也应该对相应的步骤进行验证。

第10章 缝缝补补又三年

需要强调的是，无论是在验证客户反馈问题还是验证增强功能时发现的缺陷，都需要 有详尽的记录来描述问题的症状、严重程度、发现问题的测试类型、出现问题的功能模块、 测试环境的机器和访问信息、执行测试的测试工程师的信息等。这里虽然有一些工作量在 文档上面，但是如果测试人员用测试的语言描述问题，而开发人员用开发的角度去理解问 题，可能会对问题的理解产生差异。

为了避免这种理解的偏差，需要用文档模板来记录发现的缺陷，这样开发人员和测试 人员的沟通就依托于模板，所有关于缺陷的更新都在模板的基础上进行更新，就可以有效 地减少偏差的发生。对于一些大的软件公司，如果开发团队和测试团队分别在不同的国家， 用文档模板对问题进行描述，并以此为依托进行交流和沟通，由此所产生的优势会更加明显。

执行开始一段时间后，肖瑞发现有一段时间没有见到小艾过来与他探讨补丁包测试的 问题了，所以主动找到小艾想了解一下最新的情况及最新进展。

“状态还可以，发现了一些问题，但是整体状态还在一个可控的范围内。昨天刚刚向 开发组的高级经理汇报了当前补丁包测试的状态和问题分析。为了使项目的状态简洁明 了，我参考了主版本测试团队的状态汇报模板来组织汇报的内容。”小艾回答道，并将简 单罗列的汇报内容的提纲递给肖瑞。

补丁包测试状态汇报

汇报时间：××××/××/××

当前执行状态(截至到××××):

总共计划完成的执行点数/实际完成的点数/失败的点数=××/××/×× 整体状态标记：正常

存在的问题：

1)××××

2)××××

存在的风险：

1)××××

2)××××

从菜鸟到测试架构师——一个测试工程师的成长日记

**10.2.5** **补丁包成功发布**

时间到了10月31日，从一开始接手这个工作到这时已经过了4个月的时间。小艾很 兴奋，因为补丁包这天就要发布出去了。针对补丁包的发布，小艾的问题很简单，就是补 丁包发布的意思是什么?需要注意什么?

补丁包的发布，主要有两个注意点：(1)需要发布什么信息；(2)通过什么渠道让客 户了解这些信息。

补丁包内容的发布信息其实是很简单。内容必须准确、清晰，能够为客户所理解， 产品团队希望传递一个信号给客户，那就是产品在持续地提供自身的质量和可用性。简 单来讲，发布什么其实很简单，就是发布我们能带给客户价值的东西。”

小艾若有所思地自言自语，认为应该告诉客户如下几点信息。

(1)该补丁包中解决的问题，包括问题的简单描述等需要清晰的罗列。

(2)如果有一些意义重大的问题得到解决，并且能带来功能、性能或者可定制性等任 何方面的提升，需要详细让客户了解。比如，补丁包包含一个和性能相关的客户报告的问 题，该问题的解决能给产品带来15%的性能提升。

(3)产品支持的平台的拓展和延伸的信息。用户非常乐意了解到当前版本对 Chrome 浏览器的支持，所以必须在发布产品时应重点强调。

对于补丁包信息的发布渠道其实有很多种，发邮件就是其中一种。其实也可以通过一 些专业的论坛或者网站发布这类信息，甚至如果经费允许，产品团队可以邀请媒体对补丁 包产品团队进行单独的采访，或者在电视或者报纸上刊登新补丁包发布的相关消息。

10月31日的下午，补丁包被上传到了一个外部用户可以访问的网站，发布从技术层 面讲已经完成了!

**10.3其他的流程和考虑因素**

整个项目做得差不多了，项目也基本到了收尾阶段，小艾算是从头到尾玩转了一把补 丁包的测试，但是还有一些遗留的问题需要处理和解决，这些问题困扰小艾有一段时间了。 其中包括以下几个问题：

第10章 缝缝补补又三年

(1)补丁包测试是对主版本的一个补充，有没有方法可以将问题的发现和解决提前到 主版本阶段来完成?

(2)在补丁包测试中如何从自动化测试的角度更好地完善测试框架来最大化地实现回 归测试的自动化?

带着这两个问题，小艾再一次找到了肖瑞。肖瑞很高兴小艾能对工作中发现的一些问 题有所思考：“这是一种很好的、很积极的工作态度，因为流程的东西并不是墨守成规、 一直不变的，我们需要在实践过程中发现我们工作中的疏漏，并且对执行效果不理想的地 方做出改进和完善。”

**10.3.1** **缺陷分析要做到位**

补丁包中的绝大多数问题的产生都会给客户带来消极的影响，有些甚至会影响客户的 营收，这很严重，但是如果这些问题能够在产品正式发布出去之前被开发测试团队发现， 是不是能极大地减少对客户的影响，的确是一个很值得探讨的问题。具体落实到每个项目， 项目完成后，需要做缺陷分析，分析的目的很简单——查漏补缺，巩固提高。

就像主版本里所有的测试方法都适用于补丁包测试一样，主版本里所有的缺陷分析也 都完全适用于补丁包测试。由于补丁包项目的特殊性，对补丁包测试的缺陷分析还要有一 些额外的考虑。通过对产品发布后的问题的分析，可以帮助产品团队对测试策略进行持续 的提高和改进，并且可以很好地将产品进入市场之后的问题反馈到产品开发阶段，增加产 品的市场竞争力'。

通过图10-4～图10-6可以看到进行查漏补缺的基本方法和过程，从不同的角度看待补丁 包测试的内容和对它们进行剖析。这里查漏的原始数据对象就是补丁包中包含的各种补丁。

**说明：**从功能点的角度看待缺陷的分布情况(图10-4),有几点需要引起关注。

(1)出现缺陷最多的几个功能点。

(2)该功能点在当前补丁包中出现缺陷的数量和前几个补丁包的比较，是否存在异常。

(3)功能点之间的关联性，是否相关联的几个功能点出现缺陷的数量具备可比性。



1 <http://wenku.baidu.com/view/f2de3711f18583d0496459ef.html> 百度文库

从菜鸟到测试架构师———个测试工程师的成长日记

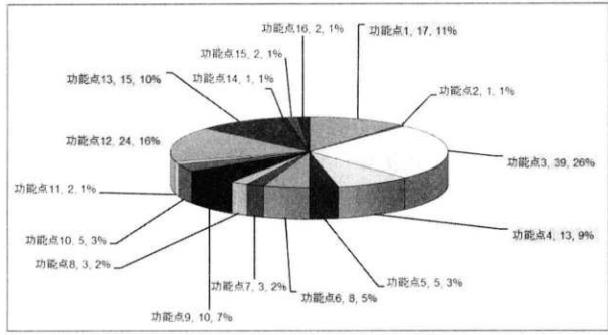


图10-4 Defect 按照功能点的分布图

**说明：**从问题发现者的角度看待缺陷的分布情况(图10-5),有几点需要引起关注。

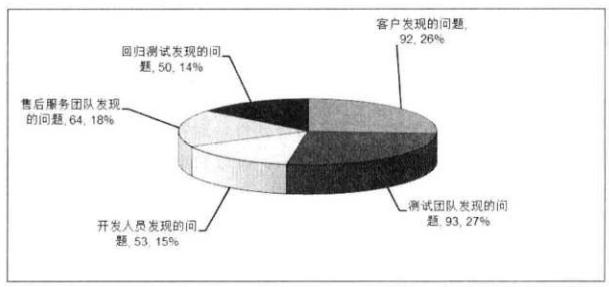


图10-5 按照缺陷发现者的分布图

(1)发现缺陷最多的发现者归类组。客户反馈问题所占比例越大，解决相应问题的投 入产出比 (ROI) 越大；相应的，开发/测试人员发现的问题越多， ROI 就会越低。

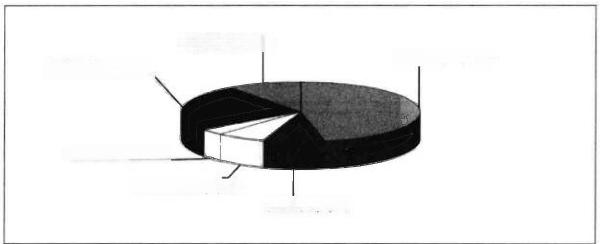
(2)如果回归测试发现的缺陷最多，这就是一个很严重的信号，表示进到该补丁包里 的客户反馈问题的解决相应的代码有很大的问题，会导致功能甚至性能的衰退。

(3)售后服务团队发现的问题如果占很大的比例，从售后服务团队即我们产品的内部 “消费者”的角度，意味着如果这些缺陷能进到当前这个补丁包里发布出去，有其他的客 户应用了该补丁包，就从很大程度上避免了外部的客户遇到类似的问题，这将极大地减轻

第10章 缝缝补补又三年

售后服务团队的工作量。

**说** **明：**从测试类型看缺陷的分布情况(图10-6),有几点需要引起关注。



迁移测试，6,.10% 安装测试15.25%

可靠性测试，2.3%

响应时间测试，4.7%

压力测试，5,8%

功能测试，28.47%



图10-6 按照测试类型的缺陷分布

(1)功能测试往往会占据一个很大的比例，考虑到功能测试的覆盖面，缺陷分析除了 需要纵向的缺陷数量的比较，更需要横向的缺陷数量比较，也就是说跨补丁包的缺陷数量 的比较。

(2)安装测试是对客户部署产品过程的模拟，因此作为最根本的测试类型，如果缺陷 数量上出现了明显的波动和异常的变化，需要额外关注。

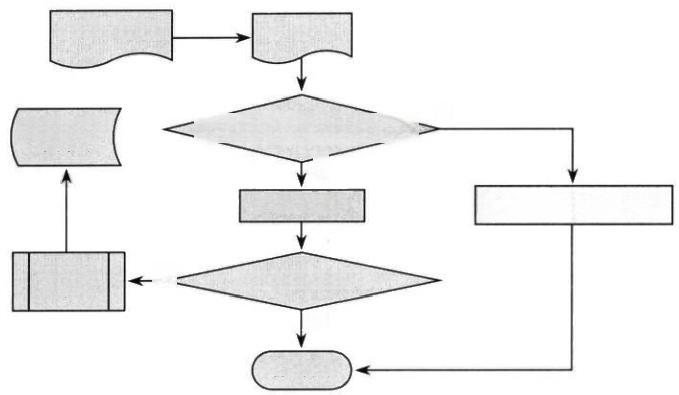
(3)可靠性、压力和响应时间的测试的覆盖面相对有限，但是在缺陷分析的时候尤其 需要注意，因为性能相关的问题往往会涉及深层次的架构设计，如果改动，代价会非常高。

**10.3.2** **补丁包测试的自动化考虑**

补丁包的测试过程中，由于涉及大量的回归测试，所以产品开发中的自动化测试资产 越丰富，自动化程度越高，就越能提升补丁包测试的效率，越早发现问题。

如图10-7所示就是补丁包测试中对于测试用例自动化处理的一个简单的流程图。可 以看到，对于任何测试用例，理论上如果能够被自动化测试覆盖，都应该被包含进自动化 测试脚本数据库，补丁包测试中的自动化开发是对已有自动化框架的一个很好的补充和增 强。但是从自动化测试的整体投入产出比的角度看，这并不是一个明智的选择，因为自动 化脚本的设计、编写及自动化框架的管理和维护都需要持续的投入，所以选择合适的自动化 测试的覆盖以帮助实现最佳的回归测试的自动化覆盖率，就成为一个需要仔细斟酌的话题。

从菜鸟到测试架构师———个测试工程师的成长日记



补丁包测试计划 测试用例

是

是香能被已有自动化测试脚本段盖

否

完成测试执行 利用已经自动化脚本完成测试

判断是否需要自动化

否

完成测试

自动化测试脚 本数据库

编写自动 化脚本

是

图10-7 补丁包测试中的自动化处理流程

对产品自动化测试的持续投入是十分有必要的，主要分为三个方面：

**1.** **已有的自动化框架的整体架构和功能提升**

这里主要是指针对自动化框架本身对新功能、新架构的支持，或者对新平台的支持。 例如：增加自动化测试框架对 Web 2.0 架构应用的支持，或者对全新的虚拟机(Virtual Machine) 平台的支持。这部分往往需要自动化框架的开发人员有一定的对产品架构的了 解及较强的编程能力。

**2.** **增加针对新功能的自动化测试资产**

新的功能需要引入新的测试用例，所以自动化测试的体系需要引入对产品新功能的支 持，这里泛指对新的测试用例的支持。

**3.** **优化和提升已有自动化测试资产的功能和质量**

已经开发出来的自动化测试资产，如同开发的产品一样，可能存在这样那样的问题， 所以随着测试资产被后续的补丁包测试广泛使用，也会发现测试脚本本身的问题。在提升 已有自动化测试资产的功能和质量方面，这需要一个持续的投入。

第10章 缝缝补补又三年

**10.4学习笔记——补丁包测试之小艾观**

经过近半年的努力工作，小艾和他的团队成功地发布了产品第一个版本的补丁包。时间 过得很快，期间的工作压力很大，但也确实让小艾和他的团队学到了很多的东西。小艾是一 个善于学习和总结的人，所以当补丁包发布后，他对在这期间所记录的笔记做了详细的梳理：

**1.** **补丁包测试的目标和意义**

产品质量(包括可用性、功能完备性、性能、可拓展性等)的持续提升；

降低产品后期维护成本；

提高客户对产品的满意度和忠诚度。

**2.** **补丁包测试的内容分类和来源**

对产品支持平台的拓展和延伸；

客户或者市场要求的产品功能的增强；

已经被产品已有客户(包括内部和外部的用户)发现的问题。

**3.** **万变不离其宗，项目的执行离不开计划**

计划先行的重要性；

计划导向的补丁包测试执行。

**4.** **问题缺陷要进行分析和跟踪**

常态化的缺陷分析的重要性；

通过缺陷分析优化流程和资源分配；

延期解决的问题需要被审批。

**5.** **补丁包要发布的内容**

公布能带给客户价值的补丁包内容。

第 **11** 章

**一遍又一遍：自动化的好处多**



从第一天走进测试组到现在，小艾已经经历了很多部门，随着经验的积累，小艾越来 越多地深入思考自己每一天的工作，越来越发现补丁包测试是一件让人很头大的事情。

**11.1软件测试的工业化——自动化**

小艾从事的产品已经有10年的历史了，版本也从最初的1.0到现在的8.0。每个版本 都不断有新功能迭加上来，随着版本的演进，产品的功能也趋向于越来越复杂。

在每个产品版本的开发阶段，软件测试组是否只需要确保当前版本所增加的功能没有 问题呢?当然不是!软件测试组不仅要保证新功能高质量的交付，也要保证原有功能不会 受到新功能的影响而产生回归测试的问题。要知道，对于客户来说，他们购买的可是整个 产品，而不仅仅是新功能!

可是现在产品的功能这么多……想到这些功能就像滚雪球一样，越来越多，那测试组 的负担岂不是越来越重了，小艾不禁发愁了。于是，小艾就这个问题咨询了测试自动化组的 资深专家林松。“这样一来，测试组不是要把已有的功能都测试一遍?这可不是小数目啊”

第11章 一遍又一遍：自动化的好处多

“岂止一遍!我们的产品可是支持多平台的，至少要在各个平台上都测试一遍。并且 为了更早地发现问题，一旦我们评估到开发人员的改动会影响某个功能点，需要尽快开展 回归测试。”

这就像滚雪球一样，随着产品的功能不断复杂，测试人员的工作量也会随着增大，如 果都依赖于手动测试，会极大增加测试人员的压力。这就是测试自动化所带来的好处了!

自动化的优势在于，它能够极大地提升测试的效率，测试人员可以迅速地在指定平台 部署测试脚本并且对相应功能进行测试。随着产品的版本不断演进，功能不断增多，自动 化所带来的好处愈发明显。否则，测试人员将被淹没在现有功能的海洋中。

自动化测试的另一个优势在于，它“弱化”了软件测试人员个体差异对测试结果的影 响。在实际的项目中，测试人员的水平经常会参差不齐，同一个功能点，一个经验丰富、 细心的测试人员往往能够比新入职的测试人员发现更多的问题。通过自动化测试，就能够 较好地解决这个问题。只要测试自动化脚本的质量达到了要求，我们就能够放心地将这套 脚本交给一个新人。

除了提高效率，有效发现产品缺陷以外，实施自动化测试还能够提高整个测试团队的 技能水平。如果一个测试团队的成员总是一味地陷入手动执行中，并不利于团队成员的成 长，也容易降低整个团队的士气。

自动化脚本是否能够发现所有的缺陷?这个问题困扰了很多自动化测试脚本的开发 者。自动化脚本的开发成本往往高于手动执行的成本，我们如何来衡量实施自动化的ROI 呢?我们在后面的小节会针对这些问题进行深入的探讨。

以上简要提到了自动化测试的优势，其重要性毋庸置疑，它能够极大地提高测试的效 率，提高一个团队的士气。然而，无论是手动还是自动化测试，都会存在相应的缺点。下 一节将阐述自动化测试给团队带来的优势和弊端。

**11.2自动化的利与弊**

正如上文所述，随着我们所从事的产品趋向于复杂，功能点的不断增加，如何能够更 加高效地开展软件测试，是摆在每个测试经理面前的难题。小艾所从事的产品已经有十年 的历史了，产品的功能复杂性也增加了好几倍。但是细心的小艾发现，测试团队的成员数 量并没有随之扩大几倍。看来，这就是自动化测试带来的最直观的好处了。

从菜鸟到测试架构师———个测试工程师的成长日记

林松给小艾展现了部分测试自动化脚本，看着这些测试程序，小艾总觉得心里不踏实， 和自动化相比，我还是更相信自己的眼睛啊，这些脚本真能发现问题吗?带着这个问题， 小艾继续向林松请教。“我知道自动化能带来很多的好处，但是自动化脚本真的能发现问 题吗?自动化脚本有没有什么弊端呢?”

自动化的缺陷就在于，它总是按照既有设定的流程往下走，不能像人一样随机应变。 一旦某个功能发生了变动，测试人员就得重新维护测试脚本。并且自动化脚本容易带来测 试覆盖率的“空洞”(Test Hole)。一旦在开发测试脚本的时候，某个产品功能点没有被测 试到，后续就很难再被发现。在和小艾描述了自动化的优点和弊端后，林松给小艾出了一 个小问题：“你认为我们应该如何平衡自动化和人工测试的关系呢?”

“嗯……我想想，应该是自动化和手工测试相结合吧，优先开发适合自动化测试的测 试用例，对于其他的测试用例，暂时采用人工测试的方法。”

的确，开发自动化脚本也是一个耗费精力的事情，如果我们花了大量的精力，只是为 了开发一个测试用例，而因为这个影响整个项目的进度，往往就得不偿失了。这或许就是 所谓的自动化与手动测试的纠结之处。

同时，一套写得糟糕、扩展性能差的测试脚本是每个测试人员的噩梦，当测试人员接 触到这样的测试脚本时，往往会花费大量的时间。林松对小艾讲述了他在刚入职时亲身经 历的一件事情。

林松在刚加入公司的时候，导师给他分配了一套测试脚本，这套脚本所测试的是产品 的一个既有功能，导师希望林松能够在验证新产品的开发过程中不破坏现有功能。这套脚 本有一定年头了，并且当时开发此脚本的工程师也已经不在林松所属的团队了。林松首先 按照测试计划里的说明搭建了测试环境，然后满怀希望地开始运行程序了(相信每个测试 自动化脚本的执行人员都有这样的体会)。运行完毕后，林松看着测试结果傻眼了，总共 有100个测试用例，居然有80%的测试用例失败了。难道真的发现产品的缺陷了?林松赶 紧向导师寻求帮助。根据林松的描述，导师首先判断，这样大面积的失败，会不会是环境 搭建的问题?于是和林松首先检查测试环境。结果，导师发现测试环境有一个地方配置错 误导致了这样大面积的失败。虽然林松是完全按照测试计划里描述的步骤进行配置的，但 是产品更新后，测试人员并没有及时地更新测试计划，从而导致了这个问题。

解决了环境配置的问题后，林松重新运行了测试脚本，这次结果好了很多，但是仍然 有10%的测试用例失败了。于是林松开始重点研究测试脚本，试图寻找错误的原因。可是 林松发现，这个过程并不轻松。首先是每个测试用例的测试目标问题，这套脚本在产品的

第11章 一遍又一遍：自动化的好处多

每个版本都运行过，由于产品某些功能改变，测试人员对测试脚本进行了修改，但是往往 忽视了将测试计划进行同步。以至于林松所拿到的测试计划和测试脚本存在很大的不一致 性。那么到底是应该以测试脚本为准，还是以测试计划为准呢?其次，林松面临的另一个 问题是，当他尝试着单个运行某一个失败的测试用例时，结果是通过的，而当他将所有测 试用例连起来运行时，就会有问题。另外，测试代码中有很多的数据是硬编码的，这就导 致部分测试用例由于产品本身数据的更新而运行失败。

在导师的帮助下，林松逐一解决了以上所提到的各种问题，并最终确保了测试脚本能 够全部通过。而测试这套脚本也花费了林松整整三天的时间。脚本本身并没有发现任何的 缺陷，林松却花费了大量的时间来优化。

从以上的例子中，小艾非常深刻地体会到，自动化开发本身就是一个双刃剑，它能够 带来的好处显而易见，但是如果没有规划好自动化开发的进度，没有控制好自动化脚本的 质量，往往会造成较多不必要的工作量。下一节会重点介绍如何开发一套高质量的自动化 测试脚本。

**11.3** **武功入门口诀——自动化脚本的关键**

了解了自动化的好处后，小艾已经跃跃欲试了，希望能够将所学到的工具用在脚本开 发中。于是他兴冲冲地找到林松，“林松，我已经了解常用的测试工具了，都说‘实践出 真知’,你赶紧给我分配一个任务吧!”

在分配任务给小艾之前，林松让小艾首先思考一个问题“什么是高质量的脚本”。小 艾仔细想了想，认为一个高质量的脚本，应该能够有效发现产品的缺陷，并且能够稳定地 运行。

目前的测试自动化领域有很多非常好用的自动化测试工具。测试工具的学习只是脚本 开发的其中一个步骤。不同的测试人员尽管都对测试自动化工具非常熟悉，但他们开发出 来的脚本质量却相差很大。这是为什么呢?虽然都是程序开发，开发测试自动化脚本和产品 开发却有着很大的不同。本节将重点讲解在开发自动化脚本时需要特别关注的几个方面。

自动化脚本开发最常用的方式是录制/回放，即测试人员通过使用测试工具将自己的 手动操作记录下来，并且通过工具生成测试代码。这就完成了一个测试用例的开发。

然而，要开发一套高质量的测试脚本，并不是简单的录制/回放，而是需要满足以下

的特点：

能够有效发现产品缺陷；

有良好的可读性和错误日志，能够方便测试人员快速定位问题所在；

能够稳定、重复、独立地运行，经过严格的审查流程；

经过充分的脚本验收流程。

**11.3.1** **有效发现缺陷**

自动化脚本的目的是为了提高测试效率，但是如果测试脚本无法有效发现产品的缺 陷，那么再高的效率也无法对产品的质量产生帮助。如果我们编写/运行测试自动化脚本 的目的是为了让脚本能够运行成功，那么就忽视了脚本存在的理由。因此，在开发测试自 动化脚本时，测试人员需要非常仔细地去核对脚本里的每个验证步骤，确保测试脚本能够 实实在在地发现产品存在的缺陷。

表11-1列举了在开发测试脚本时需要注意的几个方面。

**表11-1开发测试脚本时需要注意的几个方面**

|  |  |
| --- | --- |
| **测** **试** **步** **骤** | **验证点选取** |
| 环境搭建 | 环境搭建和数据加载后，需要有明确验证步骤，确认数据加载成功。如果数据加载失败，及时中断脚本 运行并且提示用户出错原因 |
| 脚本运行结束前，需要对所加载的数据进行清除，确保不会影响后续测试用例的运行。并且需要验证清 除结果，如果清除失败，则及时中断脚本运行并且提示用户出错原因 |
| 正常流程 | 确保结果验证和测试计划里的期望结果一一对应，不应该有遗漏的验证点。  例如，对于WEB测试，不仅仅需要验证服务端返回的是200 OK消息，还需要验证返回内容是否包含期 望的结果 |
| 每个正常测试用例执行结束后，需要检查系统日志，确保产品没有抛出异常，这往往是经常容易被忽视 的步骤 |
| 异常流程 | 确保结果验证和测试计划里的期望结果一一对应，不应该有遗漏的验证点 |
| 每个异常测试用例执行结束后，需要检查系统日志，确保产品抛出的异常符合预期 |

如果测试人员在开发测试脚本的时候能够注意以上所提到的验证点，则可以确保所完 成的脚本已经能够有效地发现产品的缺陷。这就要求测试人员不仅要熟悉测试自动化工

第11章 一遍又一遍：自动化的好处多

具，还需要非常熟悉产品的功能。

除了以上列举出来的几点要求以外，测试人员在开发测试脚本的时候，非常容易犯的 一个问题是用大量的try/catch 将自己所写的代码包装了起来，并且没有在catch 里面停止 测试用例的运行，将错误暴露出来。由于在测试过程中产生的异常被隐藏起来了，测试人 员就很容易被脚本运行成功的假象迷惑。因此，在开发测试脚本的时候，需要时刻记得脚 本的目的是暴露问题。任何在运行脚本时抛出的异常都有可能是产品问题产生的，因此需 要避免在代码中隐藏问题。

讲解到这里，林松问小艾：“小艾，你知道目前你最需要做的是什么吗?”“嗯，我应 该首先去熟悉产品的功能，这样我才能够将全面的检查点通过测试代码体现出来”。

非常正确， 一个好的自动化脚本开发人员首先必须是一个好的测试人员，只有对需要 测试的产品非常熟悉，才能够开发出真正有效的测试脚本。

这里需要强调一下，自动化开发人员首先需要熟悉的不是各种测试工具、测试框架等， 而是熟悉产品的功能。在现实的软件项目中，这一点往往被忽略了。在通常情况下， 一个 测试团队被分成了两个小团队， 一个负责测试用例编写和测试用例手动执行，另一个负责 测试自动化框架开发和脚本开发。虽然分工明确，却不利于整体效率的提高。负责测试自 动化框架和脚本开发的测试人员很少会深入地研究产品功能。

除了能够有效发现产品代码的缺陷，自动化测试脚本也应该具有非常好的可读性和可 扩展性。下面将重点介绍以下方面：

(1)如何提高测试脚本的可维护性。这就要求测试脚本有详细的错误日志和可读性。

(2)如何提高测试脚本的稳定性。这就要求测试脚本能够具备运行独立性和可重复性。

**11.3.2** **详细的错误日志**

正如上一节所提到的，测试脚本的目的是发现产品缺陷，自动化测试脚本所运行的环 境通常不是长期稳定运行的环境，而是经常会出现问题、不稳定的产品环境。当测试自动 化脚本执行失败后，需要快速分析脚本失败的原因，这是一个非常棘手的问题。在实际项 目中，测试人员往往花费大量的时间去分析脚本失败的原因和修改脚本。当一个脚本运行 失败后，可能的原因有如下几个方面：

由于产品本身的缺陷导致脚本执行失败；

从菜鸟到测试架构师———个测试工程师的成长日记

由于测试脚本本身存在的缺陷导致误报；

由于测试环境搭建产生的问题导致失败。

不幸的是，根据我们的实际经验，在一个项目中，真正由于产品缺陷导致的脚本执行 失败所占的比率并不高，测试人员往往花费大量的时间去解决脚本缺陷和测试环境导致的 失败。

为了能够节约测试人员分析问题的时间，详细的错误日志输出机制显得相当重要。那 么,我们在脚本里输出日志时应该注意哪些方面呢?

林松对此有着独到的见解，他认为在测试人员开发测试脚本的时候，需要注意以下几 个方面：

环境搭建和数据加载后，需要有明确验证步骤，如果数据加载失败，及时中断脚 本运行并且提示出错原因。

对于每一个验证点，需要在日志里输出实际的值和期望值；如果验证失败了，需 要在日志里详细描述。

尽量不要在程序里捕捉有可能出现的异常，而应该将出现的异常暴露给用户，使 得测试人员能够清楚地知道异常产生的位置。

以上几点是测试人员在开发脚本时应该注意的事项，尤其是详细的输出日志机制是一 个经常容易被忽视的环节。

**11.3.3** **良好的可读性**

“小艾，这里是一个测试A 功能的脚本，这次项目会对 A 功能进行小升级，但是我们 需要确保功能本身不受到影响，所以需要由你运行这个脚本，测试一下现有功能。”林松 给小艾分配了一个运行自动化脚本的工作。

“呵呵，这有什么难的，不就是运行测试脚本嘛。”小艾对所承担的功能有十足的把握。 可是将脚本运行一遍后，小艾发现有一半的测试用例(Test Case) 都失败了。当他试图通 过脚本代码研究失败原因时，发现面临一个很大的难题，就是不知道每个测试用例的测试 目的。于是他找到了A 功能的测试计划。经过仔细阅读后，小艾发现现存的脚本和测试计 划有很多方面已经不一致了。带着这个疑问，小艾找到了林松。

第11章 一遍又一遍：自动化的好处多

出现这个问题的原因在于，A 功能是两年前开发的功能，后续每个版本的开发陆续对 它进行了修改，而测试人员在修改脚本的时候，容易忽视(或者说没有这个意识)将相应 的测试计划进行相应的修改，这就导致随着时间的流逝，测试计划和脚本之间的鸿沟越来 越 大 。

这个问题对于一个拥有很长开发历史的产品更加明显。例如，小艾所测试的产品已经 有10年的历史了，版本也从最初的1.0到现在的8.0。每个版本都不断有新功能迭加上来， 随着版本的演进，产品的功能也趋向于越来越复杂。小艾所在的测试组积累了大量的测试 脚本，这些脚本存在的时间长，并且几乎在每个版本的生命周期内都被修改优化过。测试 人员在回归测试中很依赖这些脚本，但是他们并不知道这些脚本在测什么。更糟糕的情况 是，测试人员在每个版本中都花了大量的精力去维护它们。

造成这种问题的一个根本原因是脚本缺乏可读性，导致测试人员很难确定他们实际在 测试什么。那么,如何能够有效提高脚本的可读性呢?针对这个问题，小艾请教了林松。

要有效提高脚本的可读性，除了通用的代码编程规范外，还建议采用以下两种方式：

(1)充分利用测试代码中的注释；

(2)将测试描述和测试代码分离。

表11-2介绍了提高可读性的一些建议方法。

**表11-2提高可读性的方法**

|  |  |
| --- | --- |
| **提高可读性方法** | **详** **细** **描** **述** |
| 将测试计划的内容以 注释的形式写入到产品 代码中 | 我们经常面临的一个问题是，测试脚本和测试计划之间随着时间的迁移往往变得越来越不一 致。这是因为测试人员需要同时维护测试脚本和测试计划。  我们可以将测试计划的步骤和检验点以注释的形式加入到测试脚本中，这样会更加清晰地描述 脚本代码所实现的功能。而且当后续维护人员修改脚本时，能够很方便地将注释进行相应的修改 |
| 测试数据和代码分离 | 除此之外，还可以考虑将测试数据和代码分离，将描述测试过程和验证结果的测试用例通过配 置文件保存下来，而代码只负责执行相应的测试过程。  通过这种方式，可以很好地通过使用配置文件描述测试过程，这就使得我们的测试脚本能够具 有更好的可读性。测试人员可以方便地阅读配置文件来明白每个测试用例的测试步骤和测试目标 |
| 避免在脚本中使用硬 编码数据 | 在测试脚本中使用硬编码的数据往往容易让人疑惑，并且也不利于产品的稳定。 因此，在开发测试代码时，需要注意避免将这些数据值硬编码到代码中 |

从菜鸟到测试架构师———个测试工程师的成长日记

**11.3.4** **运行独立性与可重复性**

通过以上的阐述，小艾明白了，一个高质量的测试脚本必须具有详细的错误日志和良 好的可读性，这样测试人员才能很容易地读懂测试脚本并且减少修改脚本的时间。

不过，这并不是高质量测试脚本的全部，我们还得确保我们的脚本具有运行时的独立 性。所谓的运行独立性，指的是每套脚本之间是独立的，可以按照任何顺序独立运行，并 且在每套脚本内部，每一个测试用例也是独立的，测试人员可以从中任意选取部分测试用 例独立运行。

听到这里，小艾不仅有些迷惑了，为什么要专门强调脚本的运行独立性呢?我们可以 预先在脚本里设定每个用例的执行顺序啊，每个用例之间难道都必须独立吗?和小艾一 样，很多自动化脚本开发者均容易忽略脚本的运行独立性。在说明运行独立性的必要性之 前，我们先来比较一下手动测试和自动化测试的区别。

表11-3详细描述了手工测试和自动化测试的区别。

|  |  |
| --- | --- |
|  | **表11-3手工测试和自动化测试的区别** |
| 测试类型 | **详细区别** |
| 手工测试 | **执行手工测试的人通常都是由一个有头脑、善于思考、具有观察力的测试人员完成的。**  **只要在测试计划中显式地描述测试用例之间的依赖性，测试人员就能够很好地把握测试用例之间执行的** **先后顺序，以便一个接着一个有序地完成测试执行。**  **在执行过程中，后执行的测试用例可以利用前一个测试执行后所创建的数据记录** |
| 自动化测试 | **和手动测试不同，测试人员必须将脚本执行的顺序显式的在代码里表述出来，并且需要清晰地告诉计算** **机如何判断结果正确与否。**  **测试用例之间的依赖关系在脚本中很难清晰地表述出来。往往运行脚本的测试人员并不清楚前后测试用** **例是否具有依赖性** |

在实际项目中，我们可以根据需要，选择整套或者其中的一部分测试脚本进行回归测 试，但是，如果没有保证测试用例执行时的运行独立性，就有可能产生以下的问题：

由于所有用例之间的关系紧密，某一个用例执行失败导致了后续一系列用例的执 行失败。

增加了测试人员解决脚本问题的难度。当测试脚本运行失败时，测试人员很难快

第11章 一遍又一遍：自动化的好处多

速定位问题原因。

测试人员无法从中选取部分的测试用例单独运行。

为了确保测试脚本的运行独立性，我们需要考虑很多因素，如表11-4所示。

**表11-4运行独立性要考虑的因素**

|  |  |
| --- | --- |
| **运行独立性考虑因素** | **详** **细** **描** **述** |
| 每个测试用例都有明确的环 境设置和数据准备/清除的代码 | 在每个测试用例运行前，需要进行相应的环境搭建和数据准备。在测试用例结束后， 需要有意识地进行数据清除。  当该用例执行结束后，将系统环境的更改回滚，保证下一个测试用例可以在干净的环 境运行。虽然这种方式增加了重复代码，并且降低了程序运行的效率，但是脚本的可靠 性也大大提高了。对于提高后续的脚本维护成本非常重要 |
| 在测试代码正式发布前，多次 运行监测脚本的运行独立性 | 测试人员在运行自动化测试脚本时，经常容易发现的一个问题是，某个测试用例单独 运行时完全没有问题。可是将整套脚本完整运行时，却失败了。这通常都是测试用例之 间数据相互影响造成的。而这种问题在脚本开发阶段发现并解决的成本将比后期低。因 此，测试人员在开发完成自动化脚本后，需要有意识地将这套脚本多次无序运行，确保 脚本具有稳定的输出结果 |
| 确保脚本每次运行均输出稳 定的结果 | 所谓运行可重复性，是指我们的脚本应该相互独立，并且每个测试用例能够重复运行 并产生同样的结果。  如果脚本每次运行的结果充满了不确定性，就会大大增加脚本的维护成本。  为了确保测试脚本的运行可重复性，首先需要在脚本中增加冗余和重复的代码，例如 数据的准备和清除。在每个测试用例运行前，进行所需要的环境搭建和数据准备；当该 用例执行结束后，将系统环境的更改回滚。这样，每次重复运行测试用例时，就不会受 到上一次运行的结果影响 |

一旦脚本具有了运行独立性的特点，就能够将大量的脚本放在一个环境中顺序中运 行，尤其是当测试可以在无人看守的情况下运行时，能较大地提高测试的效率。

**11.4修炼进阶——开发适合自己的自动化框架**

上一节描述了如何开发一套高质量的测试脚本。通过阅读上一节的内容，我们知道了 一套高质量的测试脚本应该具备以下特点：有效发现缺陷、详细的错误日志、良好的可读 性、运行独立性和运行可重复性。

为了开发满足以上要求的高质量测试脚本，首先需要一个适合本项目的自动化测试框 架。在软件测试领域，可以找到大量的测试自动化工具来帮助我们开发自动化脚本。但是 这些工具通常并不是完全适合每个项目的，同样，在软件项目中，往往需要多个测试工具 的集成。为了更好地帮助测试人员开发测试脚本，我们需要开发一个具有高扩展性和可用 性的自动化框架。

一个设计良好的自动化测试框架，能够很方便地帮助测试人员开发高质量的自动化测 试用例。本节将介绍开发一套自动化框架需要注意的因素。

**11.4.1** **需求分析——我们需要什么**

在开发一个自动化框架之前，首先需要考虑该框架需要满足什么样的要求。不同领域 的需求当然会有不同。本节主要从通用功能的角度去探讨一个自动化框架的设计需要注意 哪些方面。

一套自动化脚本的运行周期中主要完成了以下工作：

(1)测试环境配置；

(2)执行待测试的用例；

(3)测试结果记录；

(4)测试环境清除；

(5)测试报告生成。

这些过程也就构成了设计一个自动化框架的最原始的功能需求。 一个设计良好的自 动化框架，应该具备如表11-5所示的特性。

|  |  |
| --- | --- |
|  | **表11-5设计自动化框架时应考虑的问题** |
| 框架特性 | **详细说明** |
| 提高测试用例脚本开发效率 | **设计框架时首先需要考虑的是，如何能够方便测试人员开发测试用例。一个设计良好的** **自动化框架，应该能够做到数据和执行过程分离。例如，我们可以使用XML文件描述测** **试逻辑，而将每个测试步骤的执行封装在框架中。**  **自动化框架负责读取配置文件的内容，明白每个测试用例的执行步骤和需要的测试数** **据。当执行完测试用例后，通过读取配置文件和期望结果进行比对。**  **用这样的方式时，测试人员只需要新增配置文件就能够新增测试用例。这样就能大大提** **亮自动化测试用例编写的效率** |

第11章 一遍又一遍：自动化的好处多

**续表**

|  |  |
| --- | --- |
| **框** **架** **特** **性** | **详** **细** **说** **明** |
| 具有完善的环境搭建的支持 | 测试人员在开发自动化脚本时，需要花费较多的精力在环境的配置和清除上。如果在框 架的层面，能够将环境配置和清除的功能进行封装，就能让测试人员更加关注于测试业务 逻辑部分代码的开发 |
| 具有完善的测试结果汇报功能 | 和环境搭建一样，将测试结果汇报封装在测试框架中能够解放测试人员，使其能够更好 地关注业务逻辑 |

**11.4.2** **环境准备自动化——将自动化进行到底**

上一节提到了在开发自动化框架需要重点实现的需求，其中，环境准备也是自动化框 架设计必须要考虑的一个重要需求。

在复杂的系统中，测试环境的准备是一个非常复杂的工程。如果我们能够搭建一个跨 平台的测试环境准备框架，就能为测试用例的开发者提供一个非常方便的工具，使得自动 化脚本开发者能够专注于测试脚本的开发。

例如，在 Web 测试自动化领域内，大量的网站操作是通过录制/回放的方法将过程进 行记录。测试自动化工具会自动生成测试脚本，开发人员只需要在其基础上进行相应的修 改。这就将测试人员从枯燥的手动点击中解放出来，而将精力更好地集中到开发测试脚本 上，从而建立了一个良性的生态链。

然而，除了录制/回放的功能以外，还需要对测试环境进行相应的准备。例如，测试 人员需要做一些常用的环境操作，从文件复制、文件内容修改到服务器的启动/关闭等。 要注意的是，这一切都是跨多个平台的!也就是说，测试人员需要考虑到多平台和数据库 组合的测试环境准备，例如Windows,Linux,AIX 等操作系统，以及Oracle 和 DB2 这两 种数据库的搭配。

我们需要将环境准备的工作在测试自动化框架的层面进行实现，具体来说，需要实现 以下几个功能：

(1)测试环境的搭建和测试数据的清除；

(2)测试用例脚本的执行调用；

(3)执行结果报告的生成。

在设计这样的自动化测试执行引擎时，首先需要考虑的是平台无关性。一旦实现了这 样的一个平台，我们的测试用例脚本就能实现“一次编写，处处执行”(write once,run everywhere) 啦。当运性能自动化脚本时，自动化框架能够自动根据目标测试环境的平台 类别完成环境搭建、测试用例执行和执行结果报告的收集。真正做到了轻点一下鼠标，让 case 去撒欢地跑吧!测试人员只等看结果了。听到这里，小艾终于明白了测试自动化的最 高境界，那就是，点一下鼠标足矣!一切皆被自动化!

**11.4.3** **实现需要考虑的问题**

在开发测试脚本时，首先需要考虑的问题是环境的搭建。在手动测试中，环境搭建往 往占用了较大的重复工作量。而环境往往是自动化脚本能够发挥重要作用的地方。因此， 测试人员往往在环境搭建自动化上花了大量的精力。

通常的 Web 自动化测试框架由一台客户端和一台测试服务器组成，我们的脚本在客 户端上运行，测试服务器则是我们的目标测试对象。 客户端完成对测试脚本的调用，以 及对远程服务器的控制。可以将客户端称为一台控制台，它主要完成以下功能：

控制环境的搭建和脚本的运行。例如，环境搭建需要完成的任务有获取远程测试 机器信息、复制文件，以及在目标系统上运行相应的进程。

完成对测试脚本的调度、测试报告的收集和展现。

为了能完成对远程机器的控制，首先需要获取远程机器的信息，例如用户名/密码、 数据库名称、数据库管理员名称和密码等。

“小艾，考你一个问题，你觉得我们可以怎样获取远程机器的信息?”

“嗯，可以在客户端侧创建一个配置文件，用于存储远程测试服务器的相关信息。”

这个方法对于客户端和服务器之间一一对应时可行，可是如果我们希望一个测试客户 端控制多个服务器，这个办法就有局限了。 而且每次更换测试环境时，都需要更新客户 端的配置信息，这也会增加维护工作量。

为了能有效获取机器信息，测试人员可以将机器信息存储在测试服务器上。例如，可 以在测试服务器的特定路径创建一个配置文件。该文件存储所有的机器信息。而在客户端， 测试人员只需要指定远程服务器的地址及系统用户名和密码。其他的机器信息均通过远程 访问服务器上的Machine.xml 文件获取。

第11章 一遍又一遍：自动化的好处多

**11.4.4** **多台机器同时跑——自动化效率更上一层楼**

有了这么高级的环境准备工具，看上去能够满足我们的所有要求了，真不错啊。小艾 已经彻底被这套自动化框架折服了。

当前的测试自动化框架，已经能够极大地提升测试自动化的效率。而另外一种提升效 率的方式是采用更多的计算机，可以避免测试人员在单机上努力地完成某些大容量的自动 化测试执行工作，那往往会花费大量的时间。

这套测试自动化框架可以理解为分布式测试自动化框架，假设现在需要运行100套测 试脚本，可以将这100套脚本分别部署到10台机器上。通过框架完成每个脚本从环境设 置到运行结果汇报的自动化步骤。

**11.5收发自如——灵活的自动化测试策略**

真正的武林高手，不仅做到了内外兼修，并且在实战中积累了丰富的经验，真正做到 收发自如。

此时的小艾，虽然已经具备了一定的自动化开发技能，但他目前最欠缺的，是在实际 项目中的测试自动化开发经验。于是林松将小艾分配到一个刚成立的测试项目组中，专门 负责其中的自动化工作。此时的他终于发现，原来，除了技术，自动化要做的还有很多 ……

**11.5.1** **越早越好?——自动化脚本开发的时间**

在上一节中简要介绍了在敏捷开发流程中测试自动化的开发流程。这对于测试人员的 要求也更高了。为了能够高效地采用敏捷流程，测试人员应该能够具有较高的开发技能， 能够快速地开发测试自动化脚本。

小艾经过一段时间的测试工具的学习，已经能够熟练使用各种自动化工具来编写测试 脚本了。

自动化可是个好东西，要是在项目过程中，第一时间把所有测试用例都自动化了，岂 不是后面就可以悠哉啦!想到这里，小艾愈发兴奋，“哈哈，我终于找到一个工作生活平 衡的方法啦。”小艾不由自主的开始了甜蜜的联想。此时，小艾对自动化的痴迷也愈来愈

深。“何必写测试计划，我的测试脚本就是我的计划”,“我要在项目初期就把自动化用例 全写完”,在小艾的心里，已经充满了对自动化所带来好处的无穷遐想。

此时，当重新审视公司测试流程时，小艾觉得这套流程已经没有存在的必要了。“何 必维护这么复杂的测试流程啊，一句话，把所有用例自动化了，每天跑得了”。小艾迫不 及待地将自己的建议告诉了林松。

听完小艾的想法，林松微笑着对小艾说，“你先别急，我问你几个问题，考考你对自 动化理念的理解。第一个问题，你觉得在项目的什么时候开发自动化脚本是比较恰当?”

小艾听后，不假思索地回答：“这还用说，当然是越快越好拉，最好是在编写测试用 例的时候直接把测试自动化脚本开发出来，然后以后的每次执行都可以用啦。”

“可是你觉得项目初期刚开发出来的功能会比较稳定吗?如果后续不断有规格变动怎 么办?另外，在功能刚开发出来的初期， 一定存在大量的问题，那个时候你的测试脚本能 运行起来吗?估计大片的程序就倒在一个主要缺陷上了。”林松说道。

小艾思索了好一会儿，明白自己的确把事情想象得太完美了，在实际的项目中，还应 该考虑到后期的项目设计变更和功能的稳定性。

的确是这样的，在实际的项目中，需要根据产品的特点设定自动化脚本开发的时间， 表11-6介绍了在制定自动化开发时间点时需要考虑的因素。

**表11-6制定自动化开发时间点时需要考虑的因素**

|  |  |
| --- | --- |
| **考** **虑** **因** **素** | **详** **细** **解** **释** |
| 不要在产品不稳定 的时候开发自动化 | 当某个功能刚开发完成时，首先需要实施一轮的手动测试。这是因为在功能刚开发完成的初期， 会存在较多的缺陷，需要测试人员通过手动测试，将大量的缺陷暴露出来。  并且，在手动测试的过程中，测试人员由于对功能有了身临其境的体验，能够发现原有测试计 划的不足和覆盖率缺失等问题。这样就能够进一步优化测试计划，使其具有更好的覆盖率 |
| 将测试脚本用于回 归测试中 | 当第一轮手动测试完成后，功能应该相对稳定，并且测试计划也得到了较好的优化。此时则可 以将目标放在如何提高后期回归测试的效率上了，也就是可以着手开发自动化脚本了 |
| 首先开发通用的task | 由于测试脚本是由一系列的通用Task逻辑组成的，在项目初期，测试人员需要提前计划本测试 项目需要使用到的Task,并且在前期重点开发Task |

因此，综合来说，测试自动化脚本应该基于一个相对稳定的测试环境下(即经过了第 一轮的手动测试),并且依照一个成熟的测试计划进行开发。如果在项目初期就开发大量 的自动化脚本，往往会导致后期大量的脚本返工量，反而降低了效率。

第11章 一遍又一遍：自动化的好处多

如果基于一个不成熟的测试计划进行开发，结果将更加危险，如果测试计划的覆盖率 有问题，则基于此计划开发的测试脚本也将无法覆盖到相应的检查点，这会导致某些问题 无法被发现，不论我们运行多少次自动化脚本。

**11.5.2** **让软件健康成长——回归测试中的自动化**

时间过得很快，小艾所在的项目已经顺利度过了 sprint 1, 并且按照计划发布了两个 功能点。在sprint 2的第一周，组长林松给小艾分配了一个测试任务：对上一个sprint 所 发布的这两个功能做回归测试。

林松首先考察了一下小艾对于回归测试的理解，小艾说，回归测试是软件测试的一个 重要项目，它用于发现在产品新版本开发过程中产生的既有功能缺陷，这通常是由新功能 的增加或者为了解决某个已知缺陷而导致的。因为如果在产品升级后，客户一直使用的功 能被破坏，这将是一个灾难性的后果，这样我们辛苦交付的新版本将没有任何价值。

“理解得很正确，回归测试的目的就是为了保证产品的功能是平滑演进的，新功能的 开发没有破坏现有的功能。然而，在实际的软件项目中，测试人员往往忽略了回归测试的 重要性。”

由于回归测试的测试目标是已有的、稳定的功能，测试人员经常会下意识地认为这个 功能是不会有问题的，并且只需要测试一次就足够了。在这种情况下，如果我们能够有一 套稳定的自动化测试脚本来代替手动测试，将极大提高测试人员的效率，使测试人员在关 注新功能的同时，不至于忽略回归测试。

对于一个复杂的产品，随着产品版本不断演进，产品所迭加的功能越来越多，而这些 新功能又成为下一个版本的既有功能。因此回归测试的工作量就像滚雪球一样，越滚越大， 这对测试人员是一个非常艰巨的考验。在实际的软件项目中，测试人员往往关注于新功能 的缺陷，他们没有过多的精力和意识去关注既有功能是否被破坏了。为了能够更加高效地 执行回归测试，自动化是其中的不二选择。

相比较新功能的开发，回归测试更加适合采用自动化测试方式。这是因为既有功能往 往比较稳定，不会有频繁的变更，可以节省测试自动化的开发成本。另一方面，回归测试 在一个软件项目中往往需要频繁执行，再执行，去检查曾经执行过的有效的测试用例没有 因为软件的变动而执行失败，在这些功能上进行自动化测试开发往往投资回报率较大。

因此，在一个软件项目周期中，尤其是在项目早期，测试人员可以优先将回归测试的 自动化工作完成。这就为后续的测试工作奠定了一个良好的基础。

**11.5.3永远的录制/回放?——自动化方式的选取**

在接受了回归测试的任务后，小艾发现需要测试的用例还没有被自动化。因此摆在他 面前的首要任务就是将这个用例进行自动化。通过分析测试步骤，小艾需要模拟真实用户 访问系统主页，并且执行一个修改客户信息的操作。基于之前所学习的 Web 测试知识， 小艾觉得完成这个任务并不难。

“这个简单，我利用测试框架下的录制功能，模拟用户的操作，然后将得到的脚本进 行修改优化，将其中硬编码的地方变成获取用户输入的方式，就完成啦。”

正当小艾兴致勃勃地开始录制时，猛然想到，目前客户信息的页面 UI 正在优化，此 时的页面并不是最终版本，很有可能在后期还会发生变化。此时如果把测试脚本自动化， 后期UI 发生变化的时候岂不是需要重新修改脚本?这不是重复劳动吗?

在实际的项目中，小艾遇到的这个问题很常见，在很多人的眼里，自动化测试就意味 着录制/回放，即GUI 测试。其实不然，自动化测试可以分成UI 测试和 API 测试两大类。 不同于单元测试，API 测试属于更高层面的测试方式，和单元测试相比，它更贴近用户的 操作(当然要稍弱于GUI 测试)。和GUI 测试相比，API 测试的实现则更加简单，并且测 试脚本的运行效率也更高。因此可以将 API 测试作为 GUI 测试的一个很好的补充，这两 类测试有很大的互补性。为了能够更好地选择这两种测试方式，下面先详细分析GUI 测试 和API 测试的优劣性。

表11-7列举了GUI 测试和 API 测试的几个不同方面。

表11-7 GUI测试和API测试的对比

|  |  |
| --- | --- |
| GUI测试 | API测试 |
| GUI测试往往采用录制/回放的方法，这种测试能够最大程 度地模拟用户的操作，也和功能测试的出发点非常契合，即 站在用户的角度去发现问题。  GUI测试和最终用户的行为联系最为紧密 | API测试则通过直接调用软件产品的外部接口，来验证返 回是否符合预期。这种方式更加容易实现。  API测试无法覆盖到界面UI的显示是否正确 |
| GUI自动化往往比预期的要困难，因为UI设计的变动会 增加自动化测试的复杂度，当UI界面处在不断变化的期间 时，开展GUI测试的效率会非常低，测试人员需要花费大量 的时间修改测试脚本，以适应图形界面的变更 | 和UI界面相比，暴露给外部用户使用的API接口往往不 会有频繁的变化，这将减少后期测试脚本维护的工作量 |
| GUI测试适用于当UI界面趋向于稳定的时候。在UI稳定 之前，通过手动测试的方式去验证UI是一个更高效的选择 |  |

第11章 一遍又一遍：自动化的好处多

根据产品的特点，可以采取不同的方式去实施API 测试，既可以直接调用产品暴露的 API, 也可以通过模拟用户的<HTTP> 请求来调用服务器端的Service。

通过以上的讲解，小艾明白了GUI 测试和 API 测试的优劣。的确，在测试项目中， 我们完全可以结合这两种测试方式，使之达到最大的效果。因此，小艾对测试用例进行了 细分，分成了API 测试和UI 测试两个子集。

对 于API 测试，因为目前涉及的接口比较稳定，不会有更改，因此小艾用很短的 时间完成了API 测试的脚本。

而对于 UI 测试用例，小艾仍然采用手动测试的方式，并且计划当页面稳定时再 进行自动化开发。

这样，通过自动化和手动的结合，将软件底层测试和 UI 测试都覆盖到，相对于纯手 动测试而言，提高了测试效率；相对于完全的GUI 自动化而言，缩短了开发周期和后期维 护的时间。

对于 API 测试自动化而言，由于其特点后期维护工作量较小，并且输入参数和输出结 果清晰，因此可以适时考虑将自动化工作提前，在编写测试计划的同时开展自动化脚本的 开发，使得测试人员可以在项目前期更有效地利用。

**11.5.4** **论持久战** **—** **—** **测试脚本的配置管理**

配置管理在软件开发领域并不是一个新概念，然而在软件测试自动化开发领域，测试 人员往往容易忽略了使用配置管理工具去管理他们所开发出来的脚本。他们认为只需要保 留一份最新的测试脚本就可以了，并不需要像产品代码那样采用严格的配置管理机制。那 么,如果缺少了配置管理，我们的自动化测试会发生什么呢?以下列举了没有配置管理所 产生的一些问题，配置管理自成体系，本节不会对配置管理的概念进行展开说明，重点是 向测试人员强调配置管理在自动化脚本开发过程中的重要性。

产品版本已经升级到V8 了，但是很多主流客户仍然使用的是版本为V6 的产品。 此时，在客户现场发现了一个遗留的缺陷，开发人员解决后，需要测试人员完成 一轮回归测试。然而，目前测试人员保留的仅仅是适用于V8 产品的自动化脚本。 由于产品在 V8 对功能点进行了较大的改动，因此将适用于 V8 的自动化脚本使 用在V6 产品上显然是不可行的。

从菜鸟到测试架构师——一个测试工程师的成长日记

某套自动化脚本历经了若干个版本，在每个版本的生命周期里，测试人员都对其 做了优化，在最近的一次脚本修改中，测试人员发现脚本的步骤和测试计划里不 一致，而测试人员无法确认这个不一致是脚本开发的时候就引入的，还是在后期 修改脚本的时候导致的。到底是应该修改测试脚本，还是测试计划呢?

从上面所描述的问题可以看出，在软件测试自动化开发中，配置管理同样是必要的， 和产品代码一样，测试脚本代码同样需要版本管理。

**11.5.5拒绝半成品——测试脚本的验收**

当完成脚本开发后，为了保证脚本的高质量交付，需要制定高效的脚本评审流程和验 收流程。

测试脚本和产品代码的区别在于，产品代码清晰描述了某个功能点，能够通过直观的 检查确定功能是否完成。而自动化脚本明确描述了测试流程、需要检查的功能点，以及期 望结果。 一套能够永远毫无差错运行的脚本不一定是高质量的脚本，因为这也有可能是由 于脚本没有发现产品的问题。因此，我们判定一套脚本的质量，不应该简单地以脚本是否 能够成功运行为标准。

我们将测试自动化脚本评审分成代码评审和功能点评审两大方面。表11-8 描述了评 审自动化脚本应该注意的事项。

**表11-8自动化评审要点**

|  |  |
| --- | --- |
| **脚本评审类别** | **详** **细** **描** **述** |
| 代码评审 | 评审人员检查代码时，应该着重检查以下方面：  (1)代码是否符合代码规范，以注释方式很详细地描述了代码的功能  (2)代码的可扩展性如何，如果有大量的重复代码，是否可以考虑对通用的功能进行封装  (3)通过阅读代码是否能够很清楚地知道每个测试用例的测试步骤、测试目标和期望结果?  (4)是否能够很好地暴露脚本运行时发现的问题。例如，测试人员是否用try/catch将各种异常进行了捕 获，并且没有在catch里面抛出该异常并停止测试用例的运行 |
| 功能点评审 | 除了评审代码质量外，评审人员还需要关注代码是否涵盖了所有的功能验证点。在测试代码中的测试步 骤和验证点是否和测试计划保持一致 |

经过了严格的审阅流程，小艾对自己所开发的测试脚本信心满满，兴冲冲地找到林松： “林松，我终于完成了我的第一套测试脚本啦!”

第11章 一遍又一遍：自动化的好处多

“哈哈，进步很快啊，恭喜你!但是你现在所完成的测试脚本，只能认为是一个半成 品，并不能说已经完全交付了。”林松微笑着说。

除了严格的评审流程外，测试自动化脚本还需要有验收的流程。如果说评审是为了保 证代码的质量和功能点的涵盖，则验收流程确保了自动化脚本的运行稳定性和可重复性。

就像我们所发布的产品需要客户验收一样，测试人员开发完成的自动化脚本也需要经 过一个验收的过程。这样做的原因是，和产品一样，测试脚本也是有用户的，这个用户往 往是项目组中的其他测试人员。他们在后期的项目中也需要经常运行此测试脚本。

我们虽然有严格的评审流程，但是这仅仅保证了代码的质量和覆盖率，并没有在真实 的环境上运行过，而往往在真实的环境上会发现很多在评审过程中无法发现的问题。因此， 我们在声称脚本已经完成前，需要由脚本的用户亲自在真实环境上运行，确保在早期发现 问题。只有验收通过了，才能声称脚本开发已经完毕了。

那么,测试脚本的验收就是为了确保脚本的用户能够顺利地运行这套脚本，并且早日 发现运行中的问题!小艾恍然大悟。

的确是这样。那么测试脚本的验收需要注意哪些方面呢?对测试脚本的验收需要主要 以下几个方面：

(1)测试脚本的验收应当由非脚本开发者，并且具备中等技能的测试人员实施；

(2)对于支持跨平台的产品，测试脚本的验收应当覆盖到不同的操作系统平台和数据库；

(3)测试脚本的验收不仅仅关注于脚本的运行是否顺利，更要关注运行时日志是否详 细，是否有助于问题的定位；

(4)在验收完成前，脚本的责任人仍然是脚本开发者，在验收中发现的任何问题，脚 本开发者都应该负责解决。

此时，自动化测试脚本已经经历了脚本评审和验收的检验，终于可以向客户交付高质 量的测试自动化脚本了。测试化脚本终于能够用于实战了!小艾很兴奋。下一节将以一个 虚拟的项目来看看自动化脚本是如何提高测试效率的。

**11.6实战洗礼——养兵千日用兵一时**

1.5 节介绍了如何根据项目的实际特点，制定灵活高效的自动化测试流程，进而保证

从菜鸟到测试架构师———个测试工程师的成长日记

测试自动化有序、高效地进行。小艾经过这段时间的训练，自动化技能成长很快，也逐步 成为项目中的一员骨干。

此时小艾所在的项目也已经进入了中后期，大量的代码已经检入 (check in) 进了产 品，此时正是问题的高发期，每天都有大量的缺陷被发现，随之而来的是一个又一个的设 计变更。本节将以小艾的亲身体验，说明自动化给项目组带来的苦与乐。从测试的角度来 看，小艾所在的项目组需要完成以下的工作：

(1)10套自动化脚本的回归测试；

(2)2套手动测试计划的回归测试；

(3)5个新增功能点的测试计划编写和功能测试；

(4)以上5个新增功能点的自动化脚本开发。

**11.6.1** **高，实在是高——自动化测试的效率**

小艾接到的一个任务是根据产品代码的改动点，选取部分自动化的脚本进行回归测试。

如果是完全手动测试，小艾需要至少三周的时间去执行回归测试。而对于自动化的脚 本，小艾只需要将脚本部署到机器上，然后等待脚本的运行结果。在等待的同时，小艾能 够继续从事新功能点的测试。为了能够提高效率，小艾将这10套脚本分别部署到三台机 器上，使得脚本能够在多台机器上同时运行!

经过不到半天的时间，这10套自动化脚本的运行结果出来了，其中有7个完全成功， 3个失败了。于是小艾用了3天的时间研究这3个脚本失败的原因，最终发现，其中两套 脚本是由于脚本本身的不稳定性造成的，而另一个则是由于发现了一个回归测试的缺陷而 抛出错误。

总体来说，小艾用了不到4天的时间，完成了手动测试需要三周的任务，并且从中发 现了一个回归缺陷。测试自动化的效率可见一斑。

**11.6.2** **无处不在的地雷——自动化测试的稳定性**

仔细分析一下小艾在做自动化回归测试的时间分布，小艾花费了不到半天的时间去部 署并运行测试脚本，但是在得到测试结果后，由于脚本的不稳定性，小艾花费了额外的3

第11章 一遍又一遍：自动化的好处多

天的时间去解决脚本的问题。由此可见，测试自动化脚本的稳定性直接决定了测试的效率。 表11-9列举了影响自动化脚本稳定性的一些主要因素。

**表11-9影响自动化脚本稳定性的主要因素**

|  |  |
| --- | --- |
| **自动化脚本稳定性因素** | **详** **细** **描** **述** |
| 脚本中出现了对某些 输入参数的硬编码情形 | 这是影响脚本稳定性的最重要的因素，因为大部分脚本都是通过录制/回放编写的。在录制的 时候，测试框架记录了用户输入的参数(如登录用户名和密码),此时需要测试人员清楚地区分 哪些是需要用户输入的动态参数，哪些是固定不变、能够硬编码的参数  如果测试人员的技能不够，就有可能将一些需要动态获取的参数误认为是能够硬编码的参数， 导致脚本在新环境中执行失败 |
| 脚本等待时间硬编码 | 在测试过程中，有时需要等待系统对某一个操作的处理，然后再运行后续的脚本。由于脚本将 来运行的环境是不确定的，在开发脚本时记录的等待时间未必就是将来测试环境中的运行时间  因此，将等待时间硬编码到程序里非常容易造成脚本的运行失败。正确的做法是，测试人员应 该通过查询系统的状态，如DB、文件等来确定该操作是否处理完毕 |
| 跨平台问题(例如操 作系统、数据库) | 在开发脚本的过程中，测试人员必须充分意识到脚本将来可能运行的平台，如果是支持跨平台 的脚本，那么需要在多个平台上测试通过  在脚本开发的过程中，测试人员往往忽略了将脚本在多个平台上进行测试的步骤。殊不知，这 将给将来的脚本维护带来不可预知的隐患 |

**11.6.3** **人员匮乏——现实和理想的差距**

“小艾，我们已经完成了功能点1的手动测试了，现在它已经具备了被自动化的条件， 你能将这个功能点自动化吗?”

“我也非常希望能马上把它自动化，可是我目前正在做功能点2的手动测试，在测试过 程中发现了很多问题，所需要的时间比预期的要多，目前实在是没有精力做这件事情了。”

以上的对话想必在每个软件项目中都会出现。的确，在一个真实的软件项目中，尽管 我们做了非常好的项目估计，留了一定的缓冲时间(buffer), 项目进行过程中往往还是会 遇到各种问题，这就导致我们预计的工作量和实际工作量存在较大的差异。

简而言之，就是目前的人员分配无法做完所有计划的事情了。在一个敏捷项目中，我 们很少能够专门成立一个自动化开发小组，测试人员通常都承担着测试和自动化脚本开发 的双重角色，每件事情都是需要占用工作量的，如何能够很好地权衡测试和自动化开发的 工作，并最大程度地利用自动化提高测试效率，是摆在每个测试经理面前的实际问题。

小艾所遇到的问题就是如何权衡手动测试和自动化脚本开发的关系。为了能够更有效 地利用自动化，以下是一些在实际项目中建议参考的原则。

对于比较稳定的测试项目(例如 API 测试),可以考虑在编写测试计划的时候同步 编写脚本，测试计划的作者同时也是测试脚本的开发者，这将极大提高自动化开发 的效率。当然，这样做的前提是，每一个测试人员都具有自动化脚本开发的能力。 在很多情况下，环境搭建和数据准备工作不会有频繁的变更，而每一个新构建到 来时，测试人员都将重复以上的工作。因此可以考虑在项目的初期先完成环境搭 建和数据准备的自动化工作，这样就能够将脚本用于每一个新的构建(Build) 中， 可以减少测试人员大量的重复工作量，同时也能够验证脚本的稳定性。

武功越高越容易走火入魔——把握测试自动化的度。

**11.6.4** **如何定义自动化比例**

经过林松的指导，小艾明白了，自动化脚本的开发工作并不是越早开始越好。自动化 脚本开发应该基于稳定的测试环境和测试计划。“好吧，那么我就在第一次手动执行后， 把所有的测试用例自动化吧。”小艾已经做好自动化脚本开发的打算了。

“一定要把所有的测试用例都自动化吗?”林松问道。

“全部都自动化了多好啊，那不就省心了吗?可以保证足够的覆盖率啊。”小艾答道。

并不是这样的，就像是在回归测试中，不需要将所有的测试用例都执行一遍一样，我 们往往是从测试用例中选取一部分来进行自动化，这个比例大概是40%。根据回归测试的 经验，绝大部分的问题都出在这40%的测试用例中。

这样做是为了在效率和质量上寻求一个平衡，将精力集中在最容易出问题的点上。小 艾恍然大悟。的确是这样的，在自动化脚本比例的选择上，比较成熟的方式是对于 API 的自动化测试选取40%～60%具有代表性的测试脚本来进行自动化；对于GUI 的自动化测 试，则一般是30%,这样可以较好地实现效率和质量上的平衡。

**11.6.5 自动化VS手动测试**

通过林松的解释，小艾明白了在何时开发自动化脚本，以及如何按照比例选取测试用 例来进行自动化开发。此时，小艾不禁想到，“好啊，那现在就可以开始开发自动化脚本

第11章 一遍又一遍：自动化的好处多

了!我曾经见到过一些测试用例，环境配置相当复杂，并且自动化的难度也很大，我就从 这些测试用例开始吧!”“当我把难啃的硬骨头都咽下去了以后，还有什么能难倒我呢?哈 哈哈 …… ”

于是，小艾兴冲冲地找到林松，“给我分配难度最大的测试用例吧!我要做自动化!”

“好啊，我可以给你一个自动化难度最大的测试用例，但是你在做之前，请先想想， 这个测试用例有必要做自动化吗?”

“啥?”小艾此时有如被人泼了一盆冷水，呆住了，“我们已经确定了何时做，以及怎 么做自动化了，怎么现在又说没必要做了呢?”

“哈哈，不要忘了，我们是在做产品，而且是软件产品。按照我的理解，软件开发的 特点就是，紧急，紧急，还是紧急!项目时间紧!项目间的间隙更紧!我们更需要考虑的 是，如何将有限的时间最大程度地利用。”

“那我们如何评估是否需要做自动化呢?”

林松表示，通常会用以下公式去评估自动化脚本开发的必要性：

脚本开发执行成本=脚本开发工作量+(平均调试脚本工作量+平均执行脚本 工作量)×每产品周期执行次数

手工执行成本=平均手工执行工作量×每产品周期执行次数

ROI = 脚本开发执行成本÷手工执行成本

如果这个ROI 比例在5以内，则说明需要用5倍的工作量去开发自动化脚本。换句话 说，如果这套脚本在将来被执行了5次，我们就把成本赚回来了，以后每运行一次，我们 就能盈利一次。而如果某功能点的手动测试需要半天时间，而我们需要花费1个多月的时 间去开发自动化脚本，那么这个比例就在60以上了，也就是说以后需要运行60次以上， 才能收回成本。

**11.7学习笔记——自动化测试之小艾观**

再来回顾一下在本章开头部分对高质量的自动化测试脚本的定义：

能够有效发现产品缺陷；

良好的可读性和错误日志，能够方便测试人员快速定位问题所在；

能够稳定、重复、独立地运行；

经过严格的Review 流程；

经过充分的脚本验收流程。

经过了自动化开发的系统培训，小艾对以上的要求也理解得更加深刻了，同时也对敏 捷环境下的自动化开发流程有了一定的了解。

测试自动化能够极大地提高测试的效率，但同时也是一把双刃剑。自动化框架和脚步 的开发是一个投入很大的项目。一个可读性不高、健壮性不强的脚本，往往会让测试人员 耗费大量的精力去维护，其结果往往得不偿失。因此，在开发测试脚本的时候，尤其要注 意确保脚本的良好可读性、运行独立性和可重复性，降低后续的维护成本。

第 **12** 章

**辗转腾挪：身手敏捷的测试**



对于软件开发而言，一年一次新产品发布日益赶不上快速变化的客户需求。开发团队 中的激进分子满肚子郁闷——市场机遇转瞬即逝，而我们的开发周期为什么那么长?测试 团队也很困惑，如何提高测试的质量和效率，确保测试案例符合客户真实的需求?突击队 式的加班使团队成员士气低落，这是大多数软件开发团队的困惑。

在软件工业界，敏捷开发已成为众多高效开发团队的制胜之道。小艾所在的团队也在 从瀑布式开发向敏捷开发转型的过程中。不过一支拥有上百人、有着十多年历史的产品开 发团队，想要华丽丽地转型，谈何容易。向敏捷转型，不是完全的自上而下或自下而上的 流程上的改进，成功的变革需要组织内部各个层次成员的思维方式的转变。在凯文的带领 下，团队中的核心成员组成了持续改进社区，积极研究并推动向敏捷的转型。

**12.1我和天使有个约会——初识敏捷**

凯文叫来小艾：“下周有个敏捷开发的培训，你去学习学习，有助于你理解我们这头 大象的敏捷转型。”“太好了，我得好好补一下理论知识。”小艾一下子兴奋起来。

从菜鸟到测试架构师———个测试工程师的成长日记

**12.1.1** **团结协作，质量为要，快速交付，响应变化，持续改进——** **敏** **捷** **的** **基** **本** **原** **则**

传统的项目开发流程是基于生产制造行业的最佳实践的总结，在软件开发的初期为软 件开发流程的建立提供了很好的借鉴，有效保证了软件产品开发过程中的项目管理、质量 管理。瀑布模型就是典型的传统的项目开发流程，从需求分析到设计、开发、测试、验收， 每个阶段都有明确的进入、退出标准和交付件。然而，随着技术和市场的快速发展，无论 是工业生产还是软件开发，都在追求更快、更优、更加适应市场变化的交付能力。

“20世纪初，从美国福特汽车公司创立第一条汽车生产流水线以来，大规模生产的方 式降低了生产成本，提高了生产率，带动并促进了生产技术的发展。第二次世界大战以后， 日本经济萧条，缺少资金，日本的汽车工业难以照搬美国的大批量生产的方式。根据日本 国情，丰田汽车提出了准时化生产(Just In Time,JIT),带领日本的汽车工业超过了美国。” 讲课的老师是来自公司软件开发质量管理中心的敏捷教练，这位白头发大叔从工业生产讲 到软件开发，从瀑布模型讲到敏捷开发，让小艾对敏捷开发有了概念上的认识。

先来看看精益生产方式，它是对准时化生产的理论总结，主要原则包括7个方面：

**1.** **消灭浪费(Eliminate** **Waste)**

什么是浪费?凡是对客户来说不需要的、没有用的都可视为浪费。这样说好像还是有 些抽象，来看几个典型的例子吧。

(1)未完成的任务。未完成的工作容易发生变化甚至被废弃。项目中会遇到这样的情 况，编码完成后，由于某种原因测试没有跟上，在发布产品的时候，这些未测试的代码是 不能提交的。有时候未完成的代码还会妨碍到其他的功能。在版本控制中，当多个人对同 一文件进行修改后，常常因为前一个修改没有提交而阻碍了后续的提交。



1 Lean Software Development,Part 1:Seven Principles([http://blog.csdn.net/dimstar/article/details/](http://blog.csdn.net/dimstar/article/details/1787840) [1783238](http://blog.csdn.net/dimstar/article/details/1787840))

Lean Software Development,Part 2:Eliminate Waste([http://blog.csdn.net/dimstar/article/details/](http://blog.csdn.net/dimstar/article/details/1787840) [1787840](http://blog.csdn.net/dimstar/article/details/1787840))

第12章 辗转腾挪：身手敏捷的测试

(2)额外的功能。客户不需要的功能就是额外的功能。很多程序员在开发的过程中都 或多或少地有完美主义倾向，认为一定要把某些功能完成，而事实上，这些功能并不是决 定产品成败的核心功能。额外的功能消耗了开发测试的资源，拖延了产品发布时间，有时 候反而阻碍了产品占领市场的先机。

(3)重新学习。在一个团队中常常会遇到这样的情况， 一个成员遇到了一个问题，费 了很大功夫终于解决了。不久后，同样的问题被团队中另一名成员遇到，他不知道这个问 题之前有人遇到过，于是又花了好大力气才解决。这对于团队来说也是一种浪费。

(4)工作交接。工作交接时即使有再完备的文档，经验和知识也是无法完全交接的。 比如完成了一半的编码，由另一位程序员接手后，需要从头熟悉需求，理清设计思路，才 能继续完成。而且在交接过程中还容易出现信息的丢失，带来很大的风险。

(5)工作切换。我们从小被教育做事情要专心致志。如果在工作的过程中被打断，即 使是短暂的中断，也需要花费一定时间才能延续上下文信息，再次进入工作状态。

(6)延迟。延迟可能会由于外界的变化而产生项目的变更。延迟还可能影响产品的发 布而导致产品丧失占领市场的机会。

(7)缺陷。在软件开发中，缺陷发现得越晚，修复缺陷的代价就越高。

(8)冗余的流程。多余的文档、无效的沟通都是浪费。

**2. 品质为先(Build Quality In)**

从一开始就注重品质，而不是依赖测试发现缺陷再去提高品质。通过代码评审发现程 序的简单性、变更容忍度、安全性等问题。通过测试驱动开发保证代码的质量。通过持续 集成，尽早发现缺陷，及时修复缺陷。避免在项目后期才将大量代码进行集成而暴露出大 量集成的缺陷，并且在这时候定位问题、解决问题也更加困难。通过快速交付尽早使用户 得到可使用的产品，并提出反馈。

**3. 推迟决定** **(Defer Commitment)**

推迟决定是指当拥有了更多的信息后再作出决定，而不是拖延决定或优柔寡断。我们 应该降低代码的依赖性，提高可重用性。依靠灵活的架构支持对变更的容忍度，使我们的 决定是可逆的。因为客户往往在一开始的时候并不能清楚地描述他对每个细节的需求，随 着一步步地互动和挖掘，需求的每一个细节才能明确。

**4. 快速交付** **(Deliver Fast)**

快速交付产品，这样产品就能够尽快被客户使用并实现客户价值，并且客户没有时间 来改变他们的想法而提出需求变更。快速交付还可以削除大量的浪费。我们需要了解哪些 功能对于我们的客户来说是最重要的，并评估这些功能的投入产出比。按照优先级逐个把 可使用的产品快速交付给客户，并持续获得客户的反馈，对待开发功能的优先级做相应调 整。需要确保所交付的产品以质量为先。

**5. 不断学习(Focus on Learning)**

软件开发是一个不断学习、改进的过程。无论是开发的流程，还是对客户需求的理解， 我们都应该持续总结、分析、学习、改进。在项目初期做计划的时候，我们无法把每一个 细节中可能的问题都预先估计到，因此需要在项目实施的过程中根据所掌握的信息做出最 佳的决策，并持续改进流程、设计及计划。

**6. 尊重员工** **(Respect People)**

以人为本，人是软件开发团队中最重要的资源。团队中的成员都充满团队自豪感，每 一个成员都能够主动地承担起团队中的责任，互相帮助，为实现团队目标共同努力。

**7. 全局优化(Optimize the Whole)**

我们在考虑优化的时候，需要从整个价值流去思考，而不是逐个部分独立地分析。比 如，通过从提出用户需求到交付产品的生命周期来分析流程的有效性、合理性，通过对交 付商业价值的分析来评估团队的执行力，通过对净推荐值 (Net Promoter Score)、保修问 题的评估来分析用户满意度。

精益生产方式的优越性不仅体现在生产制造系统，同样适用于软件产品开发、经营管 理等各个方面。在20世纪90年代，软件行业也掀起了一场寻找软件开发过程的新途径的 运动，敏捷方法成为最热门的话题。“敏捷”是软件开发的一种理念，很多方法都可以归 到敏捷旗下，遵循相同的价值观，即敏捷软件开发宣言 (Manifesto for Agile Software Development)¹:



1 <http://agilemanifesto.org/iso/zhchs/>

第12章 辗转腾挪：身手敏捷的测试

“我们一直在实践中探寻更好的软件开发方法，身体力行的同时也帮助他人。由此 我们建立了如下价值观：

个体和互动高于流程和工具

工作的软件高于详尽的文档

客户合作高于合同谈判

响应变化高于遵循计划

也就是说，尽管右项有其价值，我们更重视左项的价值。”

敏捷没有一个唯一的定义，但都遵循相同的原则：

我们最重要的目标，是通过持续不断地及早交付有价值的软件使客户满意。

欣然面对需求变化，即使在开发后期也一样。为了客户的竞争优势，敏捷过程掌控 变 化 。

经常交付可工作的软件，相隔几星期或一两个月，倾向于采取较短的周期。

业务人员和开发人员必须相互合作，项目中的每一天都不例外。

激发个体的斗志，以他们为核心搭建项目。提供所需的环境和支援，辅以信任，从 而达成目标。

不论团队内外，传递信息效果最好效率也最高的方式是面对面的交谈。

可工作的软件是进度的首要度量标准。

敏捷过程倡导可持续开发。责任人、开发人员和用户要能够共同维持其步调稳定延续。 坚持不懈地追求技术卓越和良好设计，敏捷能力由此增强。

以简洁为本，它是极力减少不必要工作量的艺术。

最好的架构、需求和设计出自自组织团队。

团队定期地反思如何能提高成效，并依此调整自身的举止表现。



1 <http://agilemanifesto.org/iso/zhchs/principles.html>

从菜鸟到测试架构师———个测试工程师的成长日记

在IBM, 对敏捷的定义是：

Uses continuous stakeholder feedback to deliver high-quality,consumable code through user stories and a series of short,stable,time-boxed iterations.

通过持续的利益相关者反馈，使用用户故事和一系列短小且是固定时长的迭代发布 高质量的、高可用性的代码。

对这个定义，我们展开了解以下几个关键点：

Uses continuous stakeholder feedback——持续的利益相关者反馈 对于软件开发项目来说，可以把利益相关者分为以下4类。

高层管理人员：决定软件产品的战略决策。比如，软件产品生产方的高层管理 人员将根据软件产品的盈利状况、市场前景作长期战略规划；软件产品购买 方的高层管理人员根据自己的企业需求决定是否需要及购买该软件产品。

内部人员：软件产品生产方的内部人员，包括开发团队、支持团队、销售团队、 市场团队、维护团队等。

合作方：软件产品实施过程中需要涉及的第三方合作伙伴。比如，对软件产品 进行定制开发的团队，负责将软件产品与其他相关产品进行集成的团队，负 责运营维护的团队。

最终用户：最终使用这个软件产品的用户。

在收集利益相关者的反馈时需要把各类人员都考虑到，他们都是决定软件产品成功与 否的关键。通过反馈，能够明确所生产的软件产品是否满足交付条件，或者存在问题需要 修改，甚至发现与需求不一致导致无法交付，从而及时发现所提交的软件产品的问题，能 够及时进行修正，以避免错误的积累而造成更大的损失。通过反馈，开发人员能够了解新 增加的需求，并及时对开发计划作调整，以确保始终专注于交付最有价值的功能。通过反 馈，开发人员能发现开发流程、开发方法中存在的问题，进行研究并改进，从而持续提高 团队的生产力。

To deliver high-quality,consumable code——发布高质量的、高可用性的代码



1 IBM\_rational敏捷开发解决方案.pdf .IBM 软件部，吴文龙。

第12章 辗转腾挪：身手敏捷的测试

在每个迭代周期结束的时候，所交付的产品都应该是经过测试的，包括功能性和非功 能性的测试，满足全面的质量标准。

Through user stories——使用用户故事

用户故事描述了用户将要利用该软件产品实现的目标，获得的商业价值。通过用户故 事作为桥梁，推动开发团队与所有利益相关者的协作。用户故事的模版如下；

*As a <role>I want to <goal>so I can <business value>*

作为<角色>我需要实现<目标>以便能够获得<商业价值>

“角色”有助于开发人员从真正用户的角度出发去理解需求。通过明确的角色 定义，能够有针对性地把相关人员(或团队)邀请到对用户故事的讨论中来。 “目标”体现了用户如何使用将要交付的产品，反映了用户的实际需求而不是 技术实现方案。

“商业价值”是衡量用户故事优先级的重要参考。

用户故事应该简洁明了，而不是需求规格说明。基于用户故事，开发团队与产品负责 人进行充分的沟通，并明确用户故事的验收标准。用户故事应该是可测量的、能够演示的。 在一个迭代周期结束时，敏捷团队演示增量交付的产品，由产品负责人确认是否达到验收 标准。

用户故事有相对的优先级，开发团队应关注于高优先级的用户故事，确定用户故事的 优先级，可以从以下几方面考虑：

交付该用户故事对丁利益相关者的价值；

不交付该用户故事对利益相关者的影响；

开发风险，如技术难点、技术不确定性。

A series of short,stable,time-boxed iterations: 一系列短小且固定时长的迭代

在每个迭代周期都应该有用户故事确认完成，完成的标准不仅仅是代码完成，还包括 测试完成、部署完成等。每个迭代周期的时间不宜过长，以确保开发团队能够专注于所计 划的用户故事，不受外界的干扰，能够快速地交付。迭代周期的时长是固定的，有助于衡 量开发团队的交付能力。

此外，成功的敏捷团队应该是通过团队自我管理，以可持续的开发节奏高效地工作。 在组建团队时，应确保团队成员具备迭代开发所需的各项技能，并具备学习能力，能够通 过学习掌握团队所缺乏的必需的技能。团队中成员应该相互信任，培养一个没有责备、埋 怨的组织环境，团队成员共同为实现团队目标而努力。通过团队自我管理，能够激发成员 的自主性和灵活性。

可持续的开发节奏是一个开发团队可长期保持的开发节奏，关键在于注重团队的高效 工作的长期可持续性。磨刀不误砍柴功，适当的休假可以让团队及时恢复精力，对新技术 的研究有助于提高团队的技术能力，对新工具的运用有助于提高团队的工作效率。一味追 求通过加班来加快进度，提高交付能力，是短视的做法。如果高强度的开发节奏超过了团 队可承受的范围，必然会影响到交付产品的质量，影响团队士气，甚至影响团队的稳定性 和创新能力。可持续的开发节奏要求团队能够在高效工作和可持续之间寻找到一个平衡 点，每个团队都可以通过实践找到一个适合的可持续开发的模式。

**12.1.2** **一切以用户价值为中心——敏捷开发基本流程**

通过白发大叔的详细讲解，小艾对敏捷开发有了基本认识。接下来，大叔又向学员们 介绍了敏捷开发的基本流程与方法。

“Scrum 是应用最广泛的敏捷开发方法。”大叔一边说一边向大家展示了Scrum 基本框 架图，“Scrum 并不是一个特定产品开发的流程，也不是一项构建产品的技术，而是一个 框架，在这个框架里可以应用各种过程和技术。作为一个迭代和增量的产品开发框架，简 单地说，Scrum 由 Scrum 团队和与之相关的关键会议、关键工件组成。”

Scrum 通过自组织、跨职能的Scrum 团队提高灵活性和生产能力，每个Scrum 团队都 有三个角色：

(1)Scrum Master: 负责确保成员都理解并遵循Scrum 过程。Scrum Master充分了解 团队，为 Scrum 过程负责，引导团队更高效地工作，帮助团队和整个组织采用Scrum, 持 续地进行改进。Scrum Master的角色是Scrum 团队的服务性领导者。Scrum Master 帮助团 队有效协作、沟通，保护团队不被外界干扰，使团队以可持续的步调完成每一个Sprint,

并使得 Scrum 的收益最大化。Scrum Master 负责组织Scrum 过程的关键会议，确保会议有

1 参考资料：《Scrum指南》、《Scrum介绍》来自Scrum中文网 ([www.scrumcn.com)。](https://www.scrumcn.com)。)

第12章 辗转腾挪：身手敏捷的测试

效进行。对于阻碍团队的问题，应该鼓励团队成员主动通过沟通协调找到解决的办法，如 果问题的解决依赖于团队之外的帮助，Scrum Master 应该协助推动问题的解决。

(2)产品负责人 (Product Owner): 负责最大化团队的工作价值。产品负责人精通行 业知识，对产品、市场和用户都有深入的了解。他需要定期与利益相关者进行沟通，充分 了解利益相关者的诉求，确保所交付的产品能带来最大的商业价值。产品负责人负责规划 产品的愿景，设定每一个 Sprint 的目标，并与团队沟通，确保团队成员能够清楚理解产 品的愿景和每一次迭代的目标。

(3)团队 (Team): 由开发人员组成的自我管理的跨职能团队，负责在每个Sprint 将 产品 Backlog 转化为潜在可交付的功能增量。团队通常由5～9个成员组成，团队成员具 备进行迭代开发所需的所有技能(比如编程、质量控制、业务分析、架构、界面设计等)。 每一个成员都尽心尽力地参与，根据团队的需要学习新技能或者巩固现有技能。团队得到 充分的授权，通过自我管理使团队以最有效的方式工作。

在 Scrum 中，一个迭代周期被称为一个 Sprint (本意为冲刺跑),它是Scrum 的核心。 每个Sprint 都采用相同的 Scrum 框架，并且都交付潜在可发布的产品增量。Sprint 的长度 由团队确定，对于同一团队，Sprint 的长度是固定不变的。一般来说，以两周、四周、六 周作为迭代周期长度的比较多。Sprint 的时间不宜过长，这样可以遏制由不确定或不可预 测而产生的风险。

Sprint 包括开发工作和以下关键会议组成：

(1)Sprint 计划会议 (Sprint Planning Meeting): 确定“做什么”和“怎么做”

在Sprint计划会议之前，产品负责人应确保产品Backlog反映了最新的需求并且有明 确的优先级。由于产品负责人充分了解产品 Backlog,Scrum 主管充分了解团队交付的能 力，为了使Sprint 计划会议更有效率，产品负责人和 Scrum 主管可以在会前进行一个准备 会议以便对 Sprint 计划达成初步的共识。

在 Sprint 计划会议中，基于团队的技术能力和交付能力的历史数据，以及在即将开始 的这个Sprint 中可能出现的节假日、休假等情况，确定团队的交付能力。基于交付能力和 优先级，从产品 Backlog 中选出用户故事，确定冲刺目标。产品负责人可以对用户故事做 进一步的说明，确保团队理解用户故事，并就用户故事的验收标准、如何演示达成一致。 为了达到Sprint 目标，实现增量交付，团队需要进一步讨论明确需要完成的任务，并对每 一项任务估算时间，确定Sprint Backlog。

(2)Sprint 评审会议 (Sprint Review Meeting)

Sprint 评审会议在Sprint 结束时举行，Scrum Master负责组织和协调会议，评审的重 点是可用的软件，无需幻灯片、讲座及其他图表等。由团队向产品负责人和利益相关者演 示完成的用户故事，由产品负责人确认验收。如果产品负责人对交付的产品有异议，Scrum Master 需要负责协调，使产品负责人能够了解在技术实现上的考虑或困难，使团队能够充 分理解产品与关键需求的差异，确保产品负责人与团队间沟通顺畅，通过讨论找到双方都 满意的折中方案。

没有完成的用户故事，将被放回到产品Backlog 中。已完成的用户故事如果有需求调 整，需要创建新的用户故事，加到产品Backlog 中。产品负责人根据反馈调整产品 Backlog 的优先级。同时，产品负责人还需要分析产品 Backlog 中剩下的用户故事，去掉不再需要 的用户故事，或对用户故事进行修改、完善。如果必要，还将对版本计划做相应的调整。

(3)Sprint 回顾会议 (Sprint Retrospective Meeting)

Sprint 回顾会议在Sprint 评审会议之后召开，主要由团队成员参加，总结在这个Sprint 中的得与失，推动团队在各方面持续成长，使团队在下一个Sprint 能够更加高效地工作。

(4)每日站立会议 (Daily Scrum Meeting)

举行每日站立会议的目的是，团队成员相互沟通，更新 Sprint Backlog, 更新障碍列 表，及时发现可能对实现 Sprint 目标有影响的问题，并及时调整Sprint Backlog。每日站 立会议举行的时间可依据团队的情况而定，但应安排在每天的同一时间。如果在第一天下 班前开了每日站立会议，第二天又把每日站立会议安排在早晨刚上班的时候，显然大家都 没有什么更新。每日站立会议由 Scrum 主管主持，团队参加，其他人也可以参加，但只 能旁听，不能干扰团队成员的交流。

每日站立会议中，每个成员需要对自己的任务做简单的总结，包括三个方面：

(1)上次会议后完成了什么工作；

(2)下一步准备做什么;

(3)有什么问题或困难影响了自己的进度。

需要注意的是，每个成员不是向Scrum Master汇报工作，而是把自己的进度、问题或 者其他相关信息跟团队分享。每日站立会议的时间不宜过长， 一般15分钟。如果有一些 话题需要深入讨论，可以在会后另安排时间组织相关成员一起讨论，避免拖延每日站立会

第12章 辗转腾挪：身手敏捷的测试

议，占用与该问题无关的成员的时间。

Scrum 的工件有助于提高关键信息的透明度，促进Scrum 过程中的各种角色之间的沟 通、协作，是确保开发团队迭代开发成功的关键，包括：产品Backlog 、Sprint Backlog和 燃尽图。

(1)产品Backlog

产品Backlog 列出待开发的产品需求，包括新功能、功能改进、需求变更及缺陷，通 常通过用户故事的形式来表述，由产品负责人负责。

产品Backlog 是根据产品和需求动态变化的，以确保产品更合理、更具竞争力。产品 Backlog 体现了产品愿景，对于Scrum 过程中的所有角色都是公开的。

产品Backlog 条目通常是以用户故事的形式展现的，按优先级排序。产品Backlog 中 优先级越高的用户故事就越紧急，需要立即进行开发。优先级高的用户故事清晰直观，拥 有比低优先级的用户故事更多的细节信息，有利于做出精确的估算和制定计划。因为产品 Backlog 是随着需求不断变化的，为减少返工，只需要把高优先级的用户故事进行分解、 细化，确保在接下来的若干个 Sprint 内将要进行开发的用户故事是细粒度的，每个用户故 事能够在一个 Sprint 内完成即可。

(2)Sprint Backlog

Sprint Backlog是团队所确认的完成 Sprint 目标需要执行的任务。Sprint Backlog由团 队确定并负责维护。团队需要对Sprint Backlog中的条目进行分解并进行估算，分解后的 任务以一人天或更少的工作量为宜。随着一项项任务的进行或被完成，需要及时修正每项 任务的剩余工作量。

(3)燃尽图

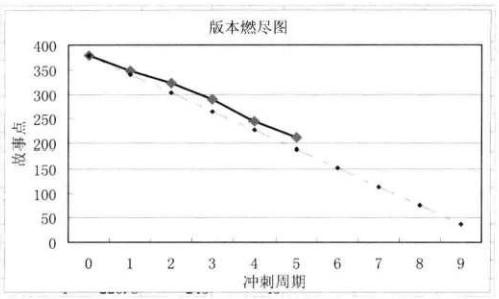
燃尽图 (Burn down chart) 是衡量团队进度的重要工具，有版本燃尽图(Release Burn down chart) 和 Sprint 燃尽图 (Sprint Burn down chart)。

版本燃尽图用于展示整个项目的进度。以这个项目的所有冲刺为横轴，以产品Backlog 中用户故事的故事点总和为纵轴。随着一个个Sprint 的完成，产品 Backlog 中的用户故事 减少。通过版本燃尽图能够迅速看出项目进展的速度和趋势，如图12-1所示。

Sprint 燃尽图用于展示一个 Sprint 内的进度，如图12-2所示。以时间为横轴，以Sprint 计划中待完成任务的理想人天总和为纵轴。随着时间的推移，任务逐一被完成，理想情况

从菜鸟到测试架构师——一个测试工程师的成长日记

下 ，Sprint 结束时，所有任务都被完成。实际项目中，可能出现当Sprint 进行一段时间后， Sprint 燃尽图没有下降，反而上升或者下降得很缓慢的情况，这有可能是在制定迭代计划 时对任务的工作量估计不足，这样团队需要考虑从Sprint 计划中去掉一些用户故事，以保 证团队能够在这个 Sprint 中专注于能够完成的用户故事。也可能出现 Sprint 燃尽图下降得 很快的情况，这样团队可以考虑从产品 Backlog 中选择用户故事添加到Sprint 计划中来。



**图12-1** **版本燃尽图**

|  |
| --- |
| **星期一** |

|  |
| --- |
| **星期二** |

|  |
| --- |
| **星期三** |

|  |
| --- |
| **星期四** |

|  |
| --- |
| **星期五** |
|  |
|  |
| 4 |
|  |
| 4 |

|  |
| --- |
| 8 |

|  |
| --- |
| 6 |

|  |
| --- |
| 4 |

|  |
| --- |
| 6 |

|  |
| --- |
| 16 |

|  |
| --- |
| 12 |

|  |
| --- |
| 8 |

|  |
| --- |
| 8 |

|  |
| --- |
| 12 |

|  |
| --- |
| 16 |

|  |
| --- |
| 10 |

|  |
| --- |
| 12 |

|  |
| --- |
| 44 |

|  |
| --- |
| 30 |

|  |
| --- |
| 28 |

|  |
| --- |
| 16 |

|  |
| --- |
| **任务** |
| 开发用户界面 |
| 开发后台程序 |
| 测试 |
| 编写帮助文档 |
| 小计 |

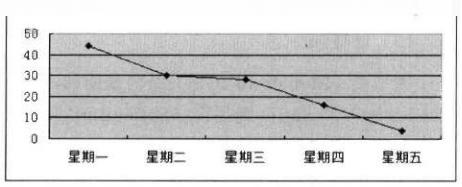


图12-2 Sprint 燃尽图

Scrum 的一个重要规则是每个 Sprint 应严格按照“完成”的定义开发潜在可交付的产 品增量。“完成”意味着分析、设计、重构、编码、文档和测试工作的完成。测试包括单

第12章 辗转腾挪：身手敏捷的测试

元测试、系统测试、用户测试、回归测试，以及非功能性测试。每个增量必须是最终产品 的一部分，每个增量都可以和前面的所有增量累加并兼容，进行过彻底的测试，能够进行 演示并交付使用的。

**12.1.3** **敏捷=不做计划吗**

“敏捷需要快速响应变化，所以就不用做计划，也没法做计划，是吗?”课程进行中， 有同学提出这样的问题。事实上，敏捷可以做计划并且应该做计划，而且，敏捷团队的计 划往往比顺序式过程的计划更为精确。

在Scrum 过程中，对于不同级别的计划有不同的估算方式。

针对 Sprint 计划，用理想人天进行估算。团队制定 Sprint 计划时，根据速率从产品 Backlog 中选择用户故事，确定完成这些用户故事需要执行哪些任务，对每项任务按理想 人天(或小时)进行估算。对具体任务的估计往往更容易，误差也会比较小。随着一个一 个Sprint 的完成，团队成员间的相互磨合，团队做出的估算也会更加准确。

针对版本计划，通过故事点 (Story Point) 对产品Backlog 中的用户故事进行估算。 故事点表示敏捷团队需要交付用户故事所需要付出的努力的相对大小，影响因素包括用户 故事的复杂程度、技术难度、不确定性，以及团队是否有交付类似的用户故事的经验，等 等。故事点是相对大小的估计，没有度量单位。比如，团队交付用户故事A 需要付出的劳 动是用户故事B 的两倍，如果我们把用户故事B 作为基准，故事点为1,那么用户故事A 的故事点就是2。故事点是由团队确定的，对不同团队的故事点进行比较是没有意义的。

在项目初期，没有足够多的信息来对细节进行一一估算的情况下，使用故事点通过相 对大小的方式能够快速地完成对用户故事的估算。

随着一个 Sprint 接着一个Sprint 的完成，每个Sprint 完成的故事点就是团队的速率。 通过历史速率可以知道不同的时间点能够交付多少功能。如图12-3所示，根据左侧的历 史数据，团队在过去的9个 Sprint 平均速率是26,速率最低的3个 Sprint 的平局速率是 22,最后一个 Sprint 的速率是28,如图12-4所示。假设在版本发布前还剩下5个Sprint, 那么,就能够估算出最少和最多能够完成的故事点数量，从而估计能够交付哪些用户故事。



1 参考资料：《Scrum 指南》、《Scrum介绍》来自Scrum 中文网([www.scrumcn.com)。](https://www.scrumcn.com)。)

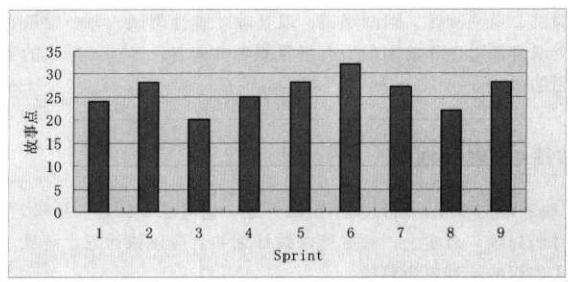


图12-3



产品 backlog

←按最低的平局速率能完成4\*22故事点

←按长期的平局速率能完成5\*26故事点

←按最近的 一个 Sprint 的速率能完成5\*28故事点

图12 - 4 Sprint速率解析

敏捷开发中，由团队对用户故事做出估计，通过团队实际交付能力进行估算往往更为 精 确 。

**12.2敏捷转型——一步一个脚印**

几天的课程下来，敏捷开发这种全新的理念深深地吸引了小艾，如果能够把敏捷方法 成功地实施，就能够紧追客户的需求，加快投入市场的速度，并且能够提高团队的士气和 战斗力。对于一个成熟团队来说，向敏捷的转型不是一朝一夕的，那么我们这个团队是怎 么一步一步向敏捷迈进的呢?

带着疑问，小艾再次找到凯文。凯文向小艾介绍了团队是如何一步步向敏捷转型的。 凯文的讲解让小艾理解了敏捷理论是如何应用到实际项目中的。

第 1 2 章 辗转腾挪：身手敏捷的测试

**12.2.1** **分析现状，尝试转型**

首先，凯文向小艾分析了转型初期团队的状况，如下所示。

从组织结构来看，直接向产品开发主管汇报的包括(参看如图12-5所示的组织结 构图):

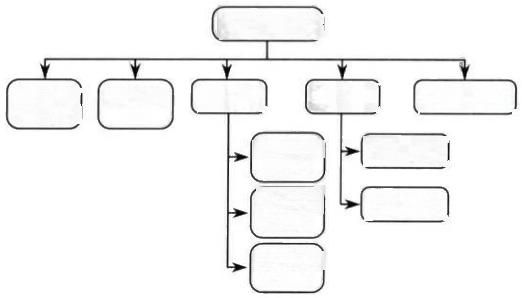
版本管理项目经理，负责制定计划、管理进度、控制成本。

架构设计部门经理，带领架构设计师团队，负责把市场需求转化为需求说明文档， 并负责系统的整体架构设计 。

开发部门经理，管理程序员团队，按功能模块包括订单模块、目录模块、营销模 块等多个开发小组(注，这里的开发部门特指程序员)。

测试部门经理，管理测试团队，按测试类型分为功能测试、系统测试。

文档部门经理，带领文档开发人员负责用户文档的编辑。



产品开发主管

版本管理 架构设计 项目经理 部门

订单模块 开发组

目 录 模 块

系统测试组

开发组

营销模块 开发组

开发部门 测试部门 文档开发部门

功能测试组

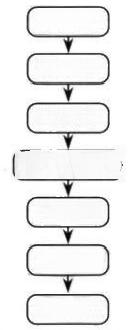
……

图12-5 组织结构

从开发流程来看(参看如图12-6所示的开发流程图),我们是典型的瀑布模型。架构 设计师把市场需求转化为需求说明文档，完成系统的整体架构设计，然后按功能模块进行 拆分，确定每个模块需要完成的项目列表。各个模块的开发组长、各个测试类型的组长以 及文档组长对每个项目估计工作量。最后由版本管理项目经理和各部门经理根据每个项目 的重要性 、 投入产出比确定需要完成哪些项目 。 接下来 ， 各个模块的开发组长完成每个项

从菜鸟到测试架构师——个测试工程师的成长日记

目的详细设计，再由程序员完成编码及单元测试。准入测试通过后，安装部署测试组进行



需求分析

架构设计

详细设计

编码+单元测试

功能测试

系统测试

验收测试

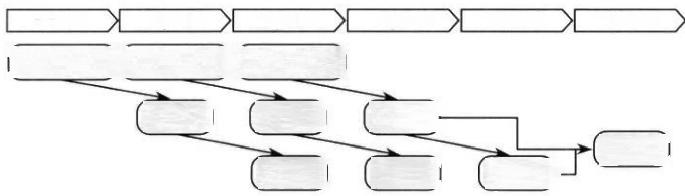
图12-6 开发流程

**12.2.2** **万里长征第一步，从迭代周期开始**

安装部署测试，功能测试组开始进行功能测试，这些测试工作都告一 段落之后，系统集成测试开始对进行集成测试、性能测试。最后，在 验收测试通过后，发布产品。

凯文说：“我们这个团队比较大，大家都习惯了原来的瀑布式开发 模式， 一下子变化太大肯定会乱的，产品开发进度还不能受影响。所 以，我们的计划就是， 一方面安排每位成员参加敏捷培训学习理论知 识，另一方面，由核心成员牵头，从Scrum 框架开始，先把团队的工 作方式变得看起来‘敏捷’了，然后再逐步尝试各种最佳实践，最后 实现真正的敏捷。”

凯文说，第一次尝试是从编码阶段开始的，按照Sprint 来组织开发工作，每个 Sprint 为4周，开发活动的安排如图12- 7所示。



Sprint1 Sprint2 Sprint3

编码+单元测试编码+单元测试编码+单元测试

功能测试 功能测试

验收测试

系统测试

功能测试

系统测试

Sprint4 Sprint5 Sprint6

系统测试

图12-7 调整后的开发流程一

每个小组按Sprint 划分工作，每个 Sprint 结束时，程序员需要提交完成的代码，并完 成单元测试。在第一个 Sprint, 程序员编码的过程中，功能测试组需要完成相应部分的功 能测试案例的准备。在第二个 Sprint, 程序员开发新的代码，功能测试组对第一个 Sprint 的代码进行测试。同时，系统集成测试组开始准备系统集成测试案例，将在第三个 Sprint 开始对第一个 Sprint 提交的代码进行集成测试。对于程序员来说，在每个 Sprint 还需要预 留 一 定的时间修复测试部门提交的缺陷。安装部署模块的开发测试也按照同样的方式组 织。通过这种调整，功能测试和系统集成测试能够更早开始，这样就能够缩短开发周期， 加快版本开发。

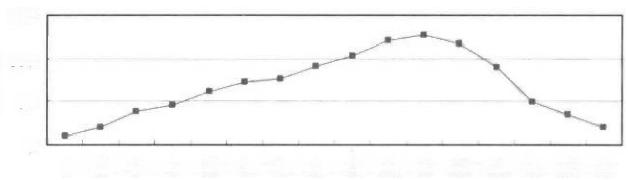
第12章 辗转腾挪：身手敏捷的测试

在实际运用后，我们发现这种模式也存在一些问题。程序员原来在编码阶段只需要专 注于代码开发，现在却因为测试的提前进入而受到干扰。对于测试人员来说，问题更加突 出。提交的代码中只有一部分能够满足一个完整的测试案例的检验，其他部分还需要等到 下一个 Sprint 的代码提交后，才能开始完整的测试。对在测试过程中发现的问题，难以作 出有效判断。有一些问题可能是由于代码只部分完成，还需要等后续 Sprint 的代码集成。 有一些问题的确是有效的缺陷，然而在下一个 Sprint 的开发过程中，这部分代码重新设计 了，这个缺陷也就不存在了。测试工作进行得很缓慢，在第n+1 个 Sprint结束时，无法完 成对第n 个 Sprint 提交的代码的功能测试，相应地，功能不稳定的情况下，系统测试也无 法顺利开始。

通过如图12-8所示，缺陷曲线图能够看出，这种模式下的缺陷曲线与瀑布模式接近。 在每一个 Sprint 结束时，产品功能不稳定，还存在不少缺陷。这些缺陷逐渐积累的结果就 是 ，团队还需要一个稳定期来修复所有的缺陷，并完成回归测试。这就增加了项目潜在的 风险。比如，在稳定期修复缺陷的过程中发现设计缺陷，如果是必须修复的问题，那么将 有可能导致项目周期的延长。又如，客户又提出了新的需求或需求变更，要求加到这个产 品版本里，这也有可能进一步影响产品的稳定，并导致开发周期的延长。

缺陷曲线

缺陷数量



150

100

50

0

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Sprint

图12-8 缺陷曲线

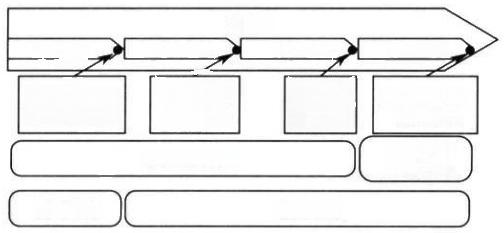
尽管问题多多，但由于测试工作提前进入，很多个设计问题在测试过程中被提前发现 并及时解决，保证了整个版本的按时发布。因此，我们还是对敏捷转型充满信心。

**12.2.3** **开发测试同步，按Sprint交付增量**

新的一个版本开发开始了，通过对前一个版本的总结，提出在架构设计完成之后，就 开始进行迭代开发，要实现真正的增量交付!

在架构设计完成之后，开发组长对待开发项目的优先级、关联性进行分析，制定计划， 确定每个项目需要在哪一个 Sprint 完成。开发、测试经理对计划进行统一的评审，确保各 个模块间有依赖性的项目、技术难度大的项目被合理安排，避免项目由于相互依赖而被拖 延，降低由于技术不确定性带来的风险。随着一个个 Sprint 的推进，开发组长需要及时调 整计划。

在一个 Sprint 中，需要完成详细设计、编码、单元测试、功能测试，进一步提前了测 试的进入，对Sprint 中的开发测试活动的时间点做了明确的规定。如图12-9所示。



Sprint

第四周

对详细设计

完成代码评审 完成测试

代码整理 修复缺陷

测试准备 功能测试

第一周

完成详细设计 测试案例设计

测试案例

进行评审 持续提交代码

详细设计 测试案例 通过评审

第三周

第三周

图12-9 调整后的开发流程二

在 一 个长度为4周的Sprint 里，详细设计、代码开发和测试工作顺序开始，从时间点 上来看，详细设计、测试案例设计的初稿在第一周完成并提交评审，同时开始代码开发。 团队邀请专家进行评审。在第三周结束前完成评审以及对评审意见的响应，确认详细设计、 测试计划评审通过，完成代码开发。最后一周，完成代码评审，完成测试，修复所有缺陷。

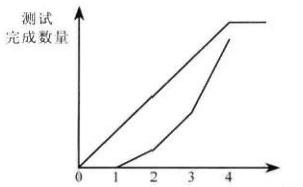
这种模式的优势在于，开发人员和测试人员都专注于当前 Sprint 的项目，有效避免了 在前期完成的详细设计，由于后期需求的变化而带来的浪费。详细设计和编码在同一个 Sprint, 能够保证从设计到编码的连贯性，减少了任务交接带来的浪费。加强测试人员与 程序员的沟通，包括详细设计、案例设计，以及根据代码开发进度相应地调整测试工作。

然而这种模式也存在如下问题。在一个 Sprint 内，依旧是瀑布模式(设计->编码->测 试),往往第一 、二周程序员会在详细设计上投入大量的精力，不能完成或只能完成少量 代码的开发，大量的代码会集中在第三周完成，甚至拖延到第四周。相应地，测试人员在 Sprint 前期的工作量不饱满，把测试工作的压力集中到了后期，可能进而影响在迭代周期 结束时测试的完成。

一 个 Sprint 的测试完成进度图常常会如图12-10所示。直线是理想情况下的进度曲线， 每周都有一定量的测试工作完成，在Sprint 结束时，测试工作完成，交付的产品是能够按

第12章 辗转腾挪：身手敏捷的测试

照预期要求运行的。曲线是实际测试工作的完成进度，由于第一周没有代码提交，所以测 试在第一周没有进度。在第二周有少量代码提交，相应部分的测试工作完成。而在第三周、 第四周，提交的代码逐渐增多，测试工作压力越来越大。测试人员往往需要加班加点才能 在Sprint 结束时完成测试工作。如果有缺陷被发现，就可能直接影响测试的顺利完成。对 于剩下的工作，如果直接在下一个 Sprint 完成，就又会遇到上一节所描述的状况。

另外，也注意到团队中的成员在 Sprint 中的各 个阶段工作强度很不均匀。Sprint 前期，开发人员 忙着编码，此时测试人员的工作不够饱满。随着代 码逐渐交付，大量的测试工作堆在了Sprint 后期。

通过总结发现，在当前的团队里，开发人员负 责详细设计、编码、单元测试，测试人员负责功能 测试。团队中的每个成员都恪守分工，缺乏对整体 的把握。原来的组织结构是按照职能部门划分(参

周

图12-10 Sprint 测试完成进度图

见图12-5所示的组织结构),主要分为开发部门和测试部门，开发部门里又按照功能模块 划分，测试部门里按照测试类型划分。每个模块的开发组长负责根据需求说明制订开发计 划，监控开发进度，确保代码按时提交；测试组长负责与开发组长协调，制定测试计划， 监控测试进度，确保测试按时完成。各个职能团队相互独立，由组长进行协调，对团队进 行指令式的管理。

另外，测试团队还是沿用瀑布模型下监控测试进度的方法，每一个迭代周期的测试完 成标准：测试案例已全部执行，通过率不低于95%;没有严重缺陷。在这种完成标准的指 导下，团队难以打破瀑布模型下的行为方式，简而言之，先开发，后测试。各个职能成员 之间以文档进行交互。以完成进度分别监控各个职能团队的执行状况。

而敏捷开发提倡“统一团队”的概念。需要敏捷团队中的每位成员都具备多种技能， 能够更加灵活地在团队中对任务分配进行合理调整，避免团队交付的瓶颈集中在某些特别 的能力上，确保团队中每一位成员能够为Sprint 目标的完成而共同努力。

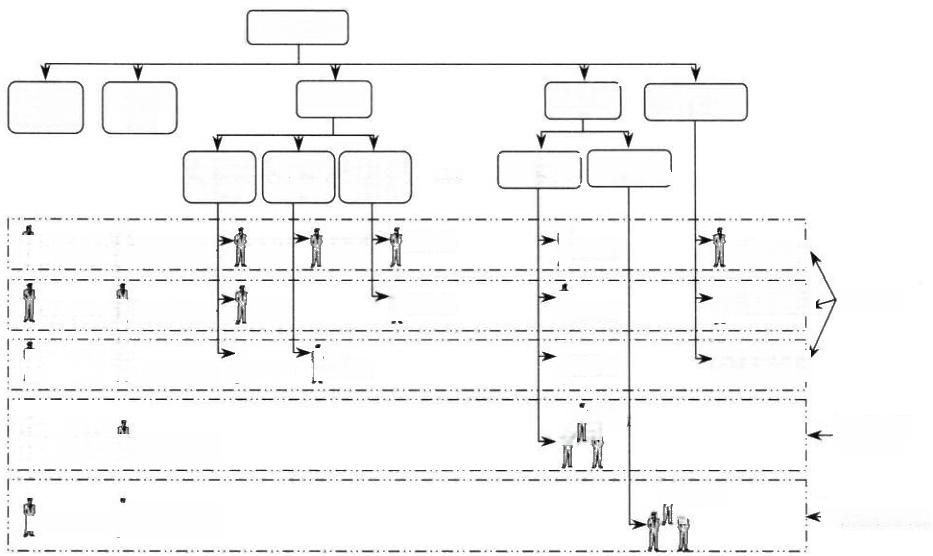
因此，需要打破严格的分工划界，实现“统一团队”。

**12.2.4** **组建统一团队，通过用户故事驱动**

在下一个版本开始之前，核心团队通过讨论，对团队组织的调整提出如图12-11所示建议。

从 菜 鸟 到 测 试 架 构 师 - - 一 个 测 试 工 程 师 的 成 长 日 记

● 338 ·



产品开发主管

架构设计 部门

功能测试组 系统测试组

开发组

产品负责人】4 Serum Maste

产品负责人:Serum Master

产品负责人 Scrum Master

测试束 白动化回归

产品负责人 Scrum Master

测试员

一系统测试团队

Scrum Master

程序员 程序员

程序员

测试员 千

可测试员

测试员 吨

程序员

程序员

秀程序员

吃

赁文档开发员

文档开发员

文档开发员：

营销模块 开发组

版本管理 项目经理

订单模块 开发组

文档开发部门

产品负责人

测试部门

特性团队

开发部门

测试团队

目录模块

程序员

吨 间

…

..

图12- 11 调整后的组织结构

第12章 辗转腾挪：身手敏捷的测试

改变后，按照敏捷的思想组建统一团队(参见12.1.2节)。针对每个特性组建一个由 跨职能的Scrum 团队，负责产品特性的开发。另外，还有一个由功能测试人员组成的自动 化回归测试团队负责回归测试、自动化测试的开发和维护，一个由系统测试人员组成的系 统测试团队负责系统集成性能测试。每个统一团队都由产品负责人负责产品 Backlog, 将 待开发的特性以用户故事的形式创建到产品订单中，并确定优先级。Scrum Master负责推 动团队按照Scrum 过程进行迭代开发，并协调与其他团队有关联的事务。团队成员以Sprint 目标为指导，以完成 Sprint目标中确定的用户故事为目标，自主协调团队内部的分工合作。

一个敏捷开发团队应该具备开发软件产品所需的所有技能。尽管这意味着需要有各个 职能的专家加入到统一团队，但并不是说每项任务都是由特定的职能人员来完成。团队中 的每位成员都可以承担测试的任务，比如自动化测试、探索性测试。在设计的时候，就需 要考虑到可测量性。

在统一团队中，不再有职能部门划分，对质量负责的不仅仅是测试人员，而是整个团 队。敏捷开发的目标是在一个时间段内生产出最大商业价值的“高质量”的软件产品，“高 质量”是统一团队的目标，而不仅仅是测试人员的责任。从代码审查、单元测试到最后确 认完成的验收测试，在敏捷开发团队中的每一位成员都或多或少地会承担测试相关的任 务。统一团队强调协作。测试人员的责任不仅仅是测试，更重要的是与程序员、产品负责 人及其他团队成员积极协作，推动所有与测试相关的工作，比如从架构设计阶段就要考虑 到产品的可测试性。

用户故事是推动敏捷开发的一个重要概念。在实施敏捷的时候，用户故事常常被误以 为只是用例 (Use Case) 的另一个叫法而已。然而，它们其实是不同的概念。

用例 (Use Case) 的定义是：在不展现一个系统或子系统内部结构的情况下，对系统 或子系统的某个连贯的功能单元的定义和描述。是以事件流的参与者和系统交互的措施来 描写用户需求，利用了一套逻辑上相对全面的事件流来定义，包括了名目、描写、重要事 件流、伸展流、失常流、前置条件和后置条件等元素。

用户故事 (User Story) 描述了用户将要利用该软件产品实现的目标，获得的商业价 值。用户故事强调的是端到端(包括前台和后台实现)的功能单位，以推动敏捷团队内部 及团队与所有利益相关者的交流。而用例更适用于分析、设计阶段。

在敏捷开发中，强调用户故事的“完成”,每一个 Sprint 结束后的增量交付都是可提 交给用户的能够稳定运行的产品，并获得利益相关者的反馈。因此我们的“完成”标准应 该针对每个用户故事，代码开发完成，测试完成(包括单元测试、功能测试、系统集成测

从菜鸟到测试架构师——一个测试工程师的成长日记

试、性能测试等),没有严重缺陷，完成代码审查，完成自动化测试脚本的开发，通过利 益相关者对用户故事的评审确认完成。

用户故事有助于敏捷团队从真正的用户的角度出发去理解需求。通过明确的角色定 义，能够有针对性地把利益相关者引入到用户故事的讨论中来。通过以用户的角度来描述 用户故事，能够体现用户如何使用将要交付产品，反映了用户的实际需求，而不是技术实 现方案。通过用户故事推动开发团队与产品负责人之间充分的沟通，并对用户故事的验收 标准达成共识。而每个用户故事的商业价值为确定用户故事的优先级提供了量化的参考， 确保团队专注于高优先级的需求。

敏捷团队对用户故事以故事点进行相对大小的估计。对于高优先级的用户故事，确定 任务并进行工作量的估算，估算敏捷团队的交付能力。在项目进行的过程中，根据实际完 成的迭代周期的交付情况校准敏捷团队的交付能力，作为后续的迭代周期的计划的参考， 必要时，可以在各个敏捷团队之间调动资源，确保版本计划的顺利完成。

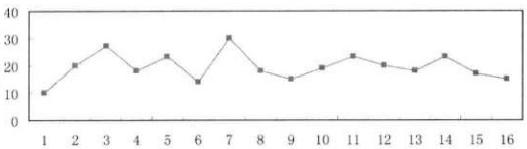
以用户故事驱动开发，使团队成员专注于当前用户故事的开发任务，可以快速完成一 个用户故事的开发，再转向下一个用户故事，降低在多个并行任务之间切换的效率损失。 开发人员完成一个用户故事的开发后及时集成代码，代码可以尽快被测试，同时降低代码 集成的风险。测试人员参与到用户故事的讨论，有助于推动澄清用户故事的验收标准，在 程序员进行详细设计和代码开发的同时，测试人员编写测试用例，开发自动测试脚本，将 测试工作提前，与开发同步完成。统一团队以快速完成用户故事为目标，每个成员根据项 目的需要快速学习并掌握新的技术，根据任务的优先级认领任务，而不局限于各自的职能。 比如，测试人员能够帮助开发人员对集成环境中的缺陷进行深入分析，开发人员能够根据 测试案例执行测试工作，或者参加自动测试脚本的开发。

快速开发完成，并对成果进行演示，能够使团队成员感受到工作成果被肯定，从而提 高团队士气。通过演示，利益相关者能够看到真实完成的产品，能够提供更加有效的反馈。 产品负责人据此对产品Backlog 进行调整，使团队能够快速响应需求的变化。及时的用户 反馈也有助于降低所开发的产品与用户需求不一致的风险，避免浪费。

通过如图12-12所示的缺陷趋势图能看到，随着时间的推移，缺陷趋势呈锯齿状，而 不是一直缓慢上升。每个 Sprint 结束时团队都有完成的用户故事提交，未能完成的用户故 事或未修复的缺陷将被作为下一 Sprint 高优先级的任务完成。

第12章 辗转腾挪：身手敏捷的测试

缺陷曲线



缺陷数量

Sprint

图12-12 缺陷曲线

**12.2.5** **适合敏捷开发的实践和工具**

随着向敏捷的转型，团队意识到有一些原有的流程和工具不再适应新的模式，甚至成 为快速开发的障碍，探索改进方法也被加入到了团队的产品Backlog 中。通过调研，发现 了一些非常有用的工具和业界常用的最佳实践。这些工具和实践并不仅仅局限于敏捷开 发，对于不使用敏捷开发的团队也能有所帮助。

**1.** **项目管理工具**

团队中原来的需求管理、配置管理、评审管理、测试计划管理、测试跟踪管理分别通 过独立的工具进行管理，各个系统相对独立。团队成员需要在各个工具之间进行切换，并 且还需要注意保证数据的一致性。项目组长、经理每周做工作报告都需要花不少功夫从各 个工具中获得数据。因此，各个工具间有效的整合能够大大提高团队的工作效率，提高项 目状态的可视性。然而，对各个独立的工具进行集成和维护势必需要大量的投入。

IBM Rational 的基于 Jazz 平台的协作应用生命周期管理 (Collaborative Application Lifecycle Management,C/ALM) 把一些很重要的工具真正整合到了一起，能够很好地支 持项目开发中各个角色成员间的协作和交流。这个平台上的典型工具有 Rational Team Concert(RTC) 、Rational Requirements Composer(RRC)及 Rational Quality Manager(RQM)。 RTC 是一个实时的协作式软件开发平台，它提供了Scrum Process 模板，内置了许多常用 工具，比如配置管理、持续集成、协作工具，以及需求管理工具。RRC 能够应用多种技术 来控制、追踪和管理需求。RQM 是一个全面的质量管理平台。

这些工具只需要通过配置就能够无缝地进行集成。业务人员在 RRC 中，通过需求的 关联查看开发计划和测试计划，了解需求被实现和被测试的进展。开发人员在 RTC 中 能

够将敏捷计划与追踪开发工作进行融合，不需要在工具与 IDE (集成开发环境)之间不断 切换，开发工作任务直接可以关联到需求，开发透明化。测试人员在RQM 中通过测试计 划和测试用例关联到需求，测试目的明确化。缺陷在 RTC 中进行管理，可以直接关联到 RRC中的需求，以及RQM 中的执行结果。所有人员能够实时、透明地访问项目数据、风 险和进度。整个生命周期开发的进展状况能够通过仪表盘的形式呈现出来，包括：系统构 建的统计、缺陷走势、燃尽图等，可以省去手工完成这些工作所花的代价。通过协作，需 求关联促进开发透明化、测试明确化，缺陷关联加速解决时间，促进业务融合，有助于提 高开发效率并提高质量。

**2.CI (持续集成，** **Continuous Integration)**

很多开发人员常常在几个功能点或特性都实现之后才检入代码，这时候需要做大量的 合并、调试工作，并且出现问题的风险也很大，定位问题、解决问题的难度也更大。

持续集成 (Continuous Integration,CI) 是持续地编译、测试、检查和部署源代码的 过程，可以让团队持续地收到反馈并进行改进，不必等到开发周期后期才寻找和修复缺陷。 团队成员应该每天提交代码。通过自动构建加快构建速度，并对构建结果进行自动化测试， 保证测试环境和生产环境的一致性。对于构建过程和自动化测试中出现的问题，应该及时 修复。通过持续集成，能够及时暴露问题，并快速定位、解决问题，降低集成风险，并且 保持软件开发过程中软件的稳定性，提高软件质量。

IBM Rational Build Forge 提供了很好的持续构建框架，可以实现构建过程自动化，并 集成用户现有的自动测试脚本和工具对构建结果进行自动化测试。

**3.TDD (测试驱动开发，Test Driven Development)**

TDD 是近年来“敏捷革命”中最热的话题之一。它是一种不同于传统开发流程的新型 的开发方法，其基本思想是，在开发功能代码之前，先编写测试代码，然后只编写使测试 通过的功能代码，从而以测试来驱动开发过程的进行。

TDD 根据需求编写测试用例，能够保证开发出的代码更符合需求，避免过度实现带来 的浪费。同时，测试用例代码又是对代码的最好的诠释，能够避免文档缺乏或更新的不及 时带来的不便。测试先行，能够有效避免和尽早发现缺陷，并快速定位问题，降低了修复 成本。测试用例代码有助于进行持续的回归测试，程序员能够安全地重购代码，优化设计。

关于TDD 的详细介绍，请参阅第4章。

第12章 辗转腾挪：身手敏捷的测试

**12.3** **互通有无——与其他实施敏捷的团队的探讨**

尽管介绍敏捷开发方法的资料很多，支持敏捷开发的工具也不少，但将理论应用到实 践的这个过程往往都不是一帆风顺的，需要经过一段艰辛的探索和尝试。在这个过程中， 多跟其他实施敏捷的团队进行经验分享常常能有意想不到的收获。下面我们看看在经验分 享中小艾记下的问题。

**问题一** **：一个** **Sprint多长合适**

这似乎是最常被提到的问题。业界常见的是以2周、4周或6周作为一个 Sprint 的长 度。Sprint 的长度没有一个固定的标准，应该根据每个团队的具体情况来确定。

一种重要的考虑因素是多长时间内团队能够不受干扰，在每一个Sprint 中，团队都专 注于完成 Sprint 目标中确定的用户故事，不会受到需求变更、优先级变化的影响。

建议尽可能选择短的 Sprint, 因为这样有助于促进团队成员改变旧的习惯，实践真正 的敏捷开发，避免把每一个 Sprint 都按照小的瀑布模型来完成。短的 Sprint 有助于更加频 繁地获得利益相关者的反馈，发现交付的版本中的问题，及时对产品订单进行调整。对短 的迭代周期作计划也更加容易、准确，便于团队成员合理安排各自的任务，避免任务交接 带来的效率损失。经常做迭代回顾，有助于团队改进。另外，对于一个新的团队来说，短 的 Sprint 便于快速了解团队效率，有助于计划的制定。如果Sprint 的时间过长，项目的复 杂性势必会增加，计划的精准性会降低，从而增加风险。

**问题二：如果用户故事不能在一个** **Sprint完成怎么办**

如果一个用户故事比较大，在制定 Sprint 计划的时候，应该考虑对用户故事进行拆分， 拆分到每个用户故事都能够在一个 Sprint完成。拆分用户故事时应注意，并不是分析出完 成这个用户故事需要完成哪些任务，然后把这些任务分到几个 Sprint 完成。拆分后的子用 户故事依然符合用户故事的定义，是能够带来商业价值的、能够交付的，依然按照用户故 事的格式描述，即“作为<角色>我想要<目标>以便能够获得<商业价值>”。对每个子用户 故事也需要明确验收条件。还需要注意，当所有子用户故事确认完成之后，有必要针对父 用户故事的验收标准进行检验，以免有所遗漏。

在一个Sprint 中，如果由于种种原因，一个用户故事没有完成，如对任务的估计不足，

从菜鸟到测试架构师— 一个测试工程师的成长日记

实际工作量比计划大，或者开发过程中遇到阻碍造成延期，在Sprint 结束的时候，没有完 成的用户故事应该放回到产品Backlog 中，重新评估优先级。通常，如果产品 Backlog 没 有大的变化，前一Sprint 未完成的用户故事依然是高优先级，会直接放到下一 Sprint 中。

在 Sprint 结束时的评审会议中，团队要向利益相关者演示完成的用户故事，如果与会 人员对用户故事提出异议，团队需要对疑问进行解答，对于利益相关者提出的修改意见， 相应地创建用户故事加进产品 Backlog 中，与其他用户故事一样，对其优先级进行评估， 根据优先级在后续的Sprint 中解决。

**问题三：如何使回顾会议更加有效**

回顾会议中要避免指责，不要把回顾会议开成检讨会。应该鼓励团队成员以积极的态 度探索新的方法改进团队的效率。其次，需要鼓励每个成员都发表意见，避免会议只是由 Scrum Master、产品负责人或者活跃分子主导，应该给每个成员机会发表自己的意见。

有效召开回顾会议的技巧很多，其中IBM Rational Self Check for Software Teams就是 一个不错的辅助团队实现自我评估以帮助提高团队敏捷性的服务。利用“团队脉冲调查问 卷 (Team Pulse Survey)”(如表12-1所示)不仅可以快速地搜集数据，还能提醒并教导团 队成员敏捷关键实践的内容。

**表12-1团队脉冲调查问卷(Team** **Pulse** **Survey)**

|  |  |  |
| --- | --- | --- |
| **实** **践** | **解** **释** | **得** **分** |
| 愿景 | 在你们的开发中，你们使用利益相关者目标来描述你们开发的目标吗 |  |
| 用户故事 | 你们是否使用用户故事来定义需求 |  |
| 单元测试 | 开发人员经常跑自动运行的单元测试吗 |  |
| 利益相关者反馈 | 你们经常从利益相关者那里拿到反馈信息吗 |  |
| 按优先级排序的产 品Backlog | 你们会从一个具有优先级排序的产品Backlog中为下一个Sprint选择工作内容吗 |  |



1 Introducing IBM Rational Self Check for Software Teams :([http://www.ibm.com/developerworks/](http://www.ibm.com/developerworks/cn/rational/edge/08/may08/kroll_krebs/index.html?S_TACT=105AGX52&S_CMP=tec.yesky) [rational/library/edge/08/may08/kroll\_krebs/index.html](http://www.ibm.com/developerworks/cn/rational/edge/08/may08/kroll_krebs/index.html?S_TACT=105AGX52&S_CMP=tec.yesky))

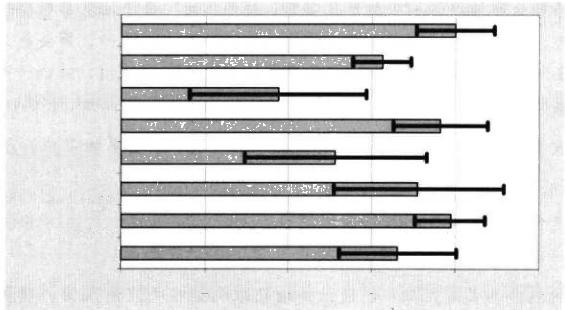
IBM Rational Self Check for Software Teams 入门简介： ([http://www.ibm.com/developerworks/](http://www.ibm.com/developerworks/cn/rational/edge/08/may08/kroll_krebs/index.html?S_TACT=105AGX52&S_CMP=tec.yesky) [cn/rational/edge/08/may08/kroll\_krebs/index.html?S\_TACT=105AGX52&S\_CMP=tec.yesky](http://www.ibm.com/developerworks/cn/rational/edge/08/may08/kroll_krebs/index.html?S_TACT=105AGX52&S_CMP=tec.yesky))

第12章 辗转腾挪：身手敏捷的测试

**续表**

|  |  |  |
| --- | --- | --- |
| **实** **践** | **解** **释** | **得** **分** |
| 估算 | 你们会根据以往Sprint的速率(完成的工作量)来重新计划当前的Sprint吗 |  |
| 可运行的软件 | 你们是否重视构建高质量的软件，在Sprint结束的时候只剩下少量不重要的缺陷 |  |
| 固定时长的迭代 | 你们是否遵守迭代项目结束的日期，根据情况调整工作内容 |  |
| 每日站立会议 | 你们是否召开15分钟的每日站立会议，讨论今天做了什么,明天打算做什么和遇 到了哪些困难 |  |
| 测试先行 | 测试是否很早做介入?测试人员是否参与Sprint计划并在早期的Sprint就开始测试 |  |
| 回顾 | 你们是否召开经验总结会来持续提高你们的流程 |  |
| 持续集成 | 你们是否经常构建?一天好几个构建分数高，相反地，一周一个构建分数最低 |  |
| 自我管理 | 团队是自我管理的吗?经理只相当于教练的作用 |  |
| 可持续发展 | 团队的进度是长期可持续的吗 |  |
| 合作开发 | 你们是否召开代码评审、结对编程等会议 |  |

在回顾会议之前，每个团队成员独立填写问卷，对团队对每一项实践的应用情况进行 打分。由Scrum Master 收集并汇总，以如图12-13和图12-14作为可视输出。图12-13显 示出团队对每一项实践应用的程度，以及团队中每个成员看法上的差异。图12-14显示出 针对软件开发实践的团队能力。



愿景 用户故事

单元测试

迭代

每日站立会议

回顾

合作开发

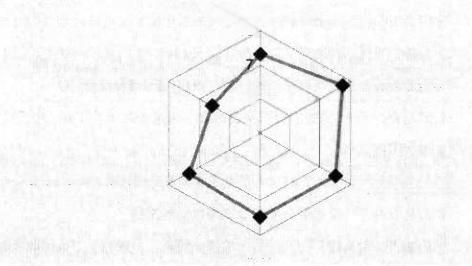
可持续发展

0 2 4 6 8 10

图12-13 团队实践应用度

通过这个工具不但能快速地搜集数据，而且能够帮助团队迅速找到自己的问题和短 板，还提醒并教导团队成员有关关键实践的内容。由于每个成员都会提交反馈，因此能够

有效保证每个成员的看法都能够反映出来。



固定时长的迭代

10

可运行的软件

3

0

利益相关者反馈

估 算

每日站立会议

按优先级排序的产品 Backlog

图12-14 团队实践能力

在回顾会议中，团队主要针对分值区间跨度大、平均值低、得分比前一次Sprint 回 顾 低的项进行讨论。通过讨论，确定出两、三项需要在下一个 Sprint 中重点改进的地方。

**12.4学习笔记—敏捷之小艾观**

实施敏捷不能生硬地照搬别人的最佳实践，每个项目、每个团队都需要根据自身的情况 持续地进行改进。敏捷更关注协作，关注质量，关注可工作的产品，需要能力全面同时又有 专攻的人才。在向敏捷转型的过程中，需要勇敢地改变一些旧习惯，尝试一些新方法。向敏 捷转型不是一蹴而就的，任何一次变革一定需要经过一段痛苦的探索、尝试，犹如凤凰涅磐。

通过与凯文探讨如何推动敏捷在项目团队中的实施，小艾更加深刻地理解了成功的敏 捷团队的特征：

快速迭代——每个 Sprint 都以固定时长完成，在每次迭代中都专注于迭代计划中 的用户故事，在 Sprint 结束时交付可稳定运行的产品。

持续的利益相关者反馈——通过持续获取利益相关者的反馈，确保产品 Backlog 能够及时调整，团队能够专注于交付最能够满足利益相关者需求的功能。

自我管理、统一团队——在被信任的环境中，团队成员更加能够发挥主观能动性， 进行创新，提高效率。

第12章 辗转腾挪：身手敏捷的测试

可持续发展——加班能够突击完成一个短期目标，但是长期的加班对一个团队的成 长和稳定性是危险的。通过一次次 Sprint, 团队能够了解适合自己的步调(速率)。

作为测试人员来说，向敏捷转型的过程更多的是观念、意识上的转型，测试流程、测 试管理上的转型。

第**13** 章

涅槃!华山论剑：测试架构师 的诞生



经过多年的测试领域的摸爬滚打，小艾已经具备了丰富的软件测试经验，对测试领域 的理论与实战都了如指掌，目前的工作已经驾轻就熟。下一步的发展规划是什么呢?

**13.1走技术路线还是管理路线，或者当架构师**

随着测试团队的逐渐壮大，测试团队做的事情也越来越深入，凯文越来越觉得项目团 队需要一个人来把握整个测试团队的技术方向，想了很久，凯文觉得小艾经历了多年的磨 炼之后，已经具备了广泛的各方面测试的能力要求，并且具有高超的技术天赋，应该是一 个好的人选。于是凯文决定找小艾聊聊他的个人发展。

凯文说：“我们测试团队越来越壮大，我想设立一个测试架构师职位，来带领整个团 队进一步发展，做深、做专、做强，我觉得你是一个合适的人选，谈谈你的想法。”

小艾觉得是个机会，不过他也和国内的大多数工程师一样，一直纠结于到底是向管理

第 1 3 章 涅槃!华山论剑：测试架构师的诞生

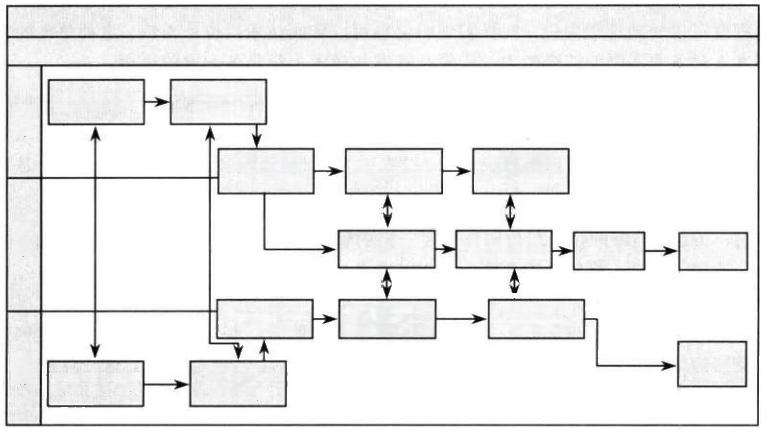
方向发展还是继续走技术路线。

小艾思索了一 下，说道：“谢谢您给我这个机会，让我想想吧。”

凯文说：“我知道你心里的想法，这样吧，我给你找个比较资深的前辈，你找他聊聊， 他叫卢涛，是我们公司目前技术职位级别最高的人之 一 ，我觉得他可以解答你的困惑，过 两天你再找我谈谈你的想法。”

卢涛可是公司里的著名牛人，他曾经做过测试工程师、测试经理、架构师等诸多职务， 是公司里的测试第 一 “大拿”。于是小艾找到了卢涛，咨询了 一 下测试工程师的职业发展 规划问题。

管理 技术



卢涛 一 听小艾的问题，会心地笑了：“你的问题是无数工程师们曾经感到困惑的问题， 其实不仅是测试工程师，开发工程师也有类似的困惑。”卢涛为小艾展示了工程师的职业 发展规划示意图，如图13 - 1所示。

测试高级 工程师

测试Tean

Leader

高级经理 总监

开发Tean Leader

高级开发 工程师

图13-1 工程师职业发展规划示意图

卢涛从宏观方面给小艾做了很多细致的解释，大致阐述了以下核心观点。

**1.** **无论是技术路线或者管理路线，并没有硬性的天花板**

至于到底从事哪个方向，主要取决于以下因素：

开发资深 架构师

测试资深 架构师

开发架构师

测试架构师

开发工程师

测试工程师

经理

长

……

.…

从菜鸟到测试架构师--一个测试工程师的成长日记

(1)个人能力

并不是每个人都适合做技术，也并不是每个人都适合做管理。所以每个人选择的发展道 路是不同的，需要考虑自己的优势和劣势，以及长远发展目标，然后选择最合适的发展路线。

(2)公司需要

当公司处于快速发展膨胀时期时，公司可能会需要更多的经理来支撑公司的正常运 转，这时很多人都可以走上管理岗位。当公司已经发展得很成熟了并且人员数量已经进入 了一个稳定时期，公司的整体发展战略就会转向做专、做深，这时一些专业技术人才就会 有更多的发展机会，因为可以向更高、更专、更精、更深的方向去研究，从而可以为公司 的发展做出更多、更大的贡献。

**2.** **技术与管理绝不是绝缘的**

从职业发展示意图中可以看到，技术与管理并不是两条永不相交的平行线。技术做到 一定程度可以转到管理岗位，同样管理职位也可以转回技术岗位。当然，这种角色的转换 需要本人具备非常强的综合能力，需要在管理与技术上都具备一定的造诣。

卢涛给小艾举了个例子：研发中心的大老板就是一个极好的例证，他在技术方面取得 了卓越的成就，在海量数据处理方面拥有业界顶尖的技术造诣，拥有本公司全球范围内最 高的技术职称，同时他目前也是我们的大老板，所以他是我们所有人的楷模，无论是管理 人员还是技术人员。

另一方面，高级的技术职位同样需要一定的管理能力，职位越高，越需要更强的沟通、 协调、计划、决策、控制等管理能力，所以技术与管理绝对不是绝缘的。我们对本公司的 顶级技术专家、管理层人员曾经做过一些调查，他们一致同意这样的观点。对于技术与管 理的角色转换的难易程度问题，主要是取决于个人的能力，本公司有太多的成功案例，有 很多资深的架构师以前就是当经理的，也有很多架构师转换角色当经理。这一点在图13-1 所示的示意图中也可以看出。

**3.** **测试工程师的发展**

从图13-1 中可以看到，测试工程师的发展路径也是非常灵活的。有的测试工程

术能力比较强，尤其是一些从事自动化测试框架开发的工程师，通常也具备较强的编码能

力，有的人就转去做开发的职位了。有的工程师做了测试的Team Lead,再往后有的人做

了架构师，有的人做了经理，还有的测试出身的工程师选择了技术销售、技术支持等职位。

第13章 涅槃!华山论剑：测试架构师的诞生

所以并不是每个测试工程师只能永远被固定在基本的测试岗位上，个人的发展主要取决于 个人的能力及公司的发展需要。

另一方面，也可以看到，本公司给予员工的认可与报酬并不是根据你的角色来定的， 而是根据你的能力级别与贡献来定的，有的员工比他的经理的工资还高。

小艾听了之后对于测试工程师的职业发展有了更清晰的了解，他分析了一下目前所在 组织的情况。目前组织里面并不缺管理人员，最缺的是具备资深技术领导能力的人。于是 小艾决定从事测试架构师这一工作。

小艾找到凯文表达了希望从事测试架构师这个职位的想法，不过他心里还是没底：“可 是我还不知道如何才能做一个合格的测试架构师。我该从哪里开始?”

**13.2** **测试架构师是干啥的**

小艾对测试架构师这一职位的角色所担当的职责略知一二，但谈不上理解得特别透 彻。卢涛耐心地给小艾介绍了什么是测试架构师。

**卢涛的话**

测试架构师是一个非常资深的专业职位，在一个组织里面的级别大致相当于开发架 构师或者管理职位序列中的经理级或者更高级别的管理职位。但是测试架构师与经理的 区别是经理承担着直接管理员工的职责，而测试架构师并不直接管理一线员工。

测试架构师的职责非常多，大致包括：

(1)领导所在测试组织的技术与战略方向；

(2)负责制定测试战略、测试架构；

(3)负责在技术层面控制整个测试组织的总体测试质量；

(4)影响整个大的产品研发团队的未来战略、发展计划与方向；

(5)负责领导确定测试领域的测试方法、策略、流程与工具等方面的选择，包括在 系统的性能、安全性、稳定性、可靠性等方面的测试方法、技术线路和质量标准；

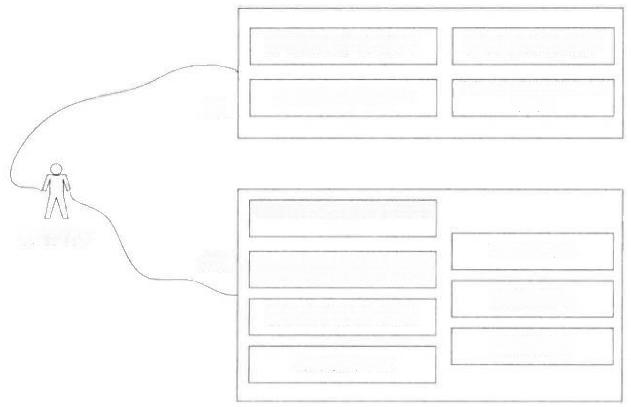
(6)领导测试领域的技术与方法革新，提出改革计划；

(7)制定测试的进入/退出标准；

**(8)领导测试领域的重大技术公关。**



小艾听了以后似懂非懂。听起来测试架构师的职责很多也很宏大，可是测试架构师到 底具体该干些什么事情呢?于是卢涛给小艾画了一个图，如图13-2所示。



参与设计讨论与文档评审

参与性能优化架构讨论

开发

编写测试战略、总休测试计划 文档

测试架构师

测试工具选型

测试

审核测试计划与测试用例设计

测试过程管理

指导测试白动化体系建设

缺陷质量审查

测试覆盖率审查

参与单元测试报告评审

参与开发质量持续改进方 案设计

图13-2 架构师的职责

然后卢涛给小艾详细地解释了这个图的含义。

总体来看，测试架构师绝不仅仅是控制测试过程的质量，也参与到开发活动当中，以 求尽早发现质量风险，并采取措施尽早防范。

**1.** **参与开发过程**

测试架构师需要在软件开发设计阶段就参与进去，这样可以尽早地了解开发需求与开 发计划。同时测试架构师要在开发设计的讨论中从测试专家的角度审查测试可行性(可测 性),并参加设计文档评审，给开发人员提出意见和建议。

**什么是可测性?**

软件的可测性是指测试对象是否能够被容易地测试。可测性不好的被测试对象可能 需要占用大量的测试时间与精力，在极端情况下甚至可能意味着该被测试对象根本无法

第13章 涅槃!华山论剑：测试架构师的诞生

被测到，从而导致质量的漏洞。

所以测试架构师必须要关注特测试的软件产品的可测性。

**如何提高可测性?**

一般来说可以通过推行测试驱动开发及在设计阶段就强制地要求开发人员考虑可 测性的设计从而加以改善。

测试架构师清楚地知道本软件产品的性能瓶颈大概在哪些方面，所以测试架构师也应 该参与性能优化架构方面的讨论，从而给性能优化提供合理化建议。

开发人员的 UT(Unit Testing, 单元测试)做得好坏，直接影响到测试工作的进展顺 利与否。作为测试架构师，需要参与开发人员的单元测试报告评审，确保单元测试覆盖率 与成功率。

开发质量持续改进是开发团队与测试团队的共同责任，测试架构师最清楚开发质量的 弱点在哪里，从而可以与开发团队共同设计改进方案。

**2.** **参与测试过程**

既然是测试架构师，显然需要参与到所有的测试活动中。测试架构师需要制定整个测 试团队的测试战略，并编写总体测试计划文档，从而让整个团队清晰地理解测试的方向与 焦点，并从技术层面对团队进行指导。

测试人员的测试计划与测试用例设计必须经测试架构师统一把关，从而使测试架构师 能够监控到测试点的覆盖率，并确保测试团队人员的工作方向是正确的。

测试自动化是软件测试领域中非常重要的组成部分，合理地使用软件自动化可以大大 提高测试效率与测试覆盖率。测试架构师需要对整个团队的测试自动化体系的建设进行指 导。有的大型研发团队可能会设立独立的自动化体系架构师职位，测试架构师需要与自动 化体系架构师一起制定合理的自动化测试战略、框架、方法、过程及工具。

测试架构师作为测试组织的技术负责人，担负着测试过程管理与优化的职责，需要根 据本测试组织的实际情况量身定制最合适的测试流程，并监督流程的贯彻执行，确保实现 测试组织的质量管理目标。

测试人员开出的缺陷报告好坏直接反映了测试组织的水平，测试架构师需要指导测试 人员精准地发现高质量的缺陷，并做好缺陷分析。

**13.3** **测试架构师之N项修炼**

小艾通过与公司内部的导师们沟通，逐渐清晰地了解到测试架构师需要修炼多项技 能，需要具备非常高的综合素质，这些素质大致可以分为两大类：技术技能与软技能。

**13.3.1** **技术技能修炼**

好的测试架构师必须具备坚实、广泛的技术技能，这样才能真正地理解产品的技术与 架构，才能发现更多的深层次的质量问题。

**1.** **对新技术的快速理解能力**

IT 行业的技术更新非常之快，新的技术层出不穷，开发架构师们经常会考虑在产品的 开发中采用一些新的技术，作为测试架构师必须要跟上技术的发展步伐，能够和开发架构 师一样把握技术的发展趋势与现状，这样才能与开发架构师、开发人员有共同语言，更深 刻地理解新技术对质量管理带来的正面与负面影响。试想当一个产品构建在SOA 架构上， 一个测试架构师如果不知道SOA 为何物，他是无法理解基于SOA 架构的测试应该怎么做的， 这样的后果可能是将会出现很多测试漏洞，从而给客户造成损失，影响产品的市场地位。

小艾很庆幸自己所在的公司是整个行业的技术领导者，这里每天都有大量的新技术讲 座、研讨会，时刻可以学习到新的东西。

**2.** **对测试相关技术的深度理解**

很显然，对于测试架构师来说，测试相关的技术是基础技能，测试架构师必须是测试 通，对测试的各种类型、流程、各种技术了然于胸，本书前面的章节描述了大量的测试方 面的基础知识，这类知识是测试架构师需要掌握的，当然不同项目做的产品不同，所以不 同项目中测试架构师所要了解的测试技术也不尽相同，会有所偏重。

**3.** **对软件开发的理解**

小艾曾听前辈们说过公司里一个测试“大拿”的经典故事，该大拿给开发人员开了个 缺陷报告，开发人员直接给他打了回来，说开发代码没问题，肯定是测试人员的误操作造

第13章 涅槃!华山论剑：测试架构师的诞生

成的，该大拿一怒之下直接把问题定位到了出问题的那段代码，告诉开发人员 XX 类中 XX 行代码有XX 问题，并且告诉开发人员应该如何修改代码，开发人员看了之后心服口 服，以后对该大拿开的缺陷报告都毕恭毕敬地对待，再也没敢轻易地打回来。

小艾了解到公司的顶级测试架构师们都具有极深的软件开发的知识，很多人从事过多 年的软件编程工作。他们对开发的过程、设计模式、开发语言、代码的细节都了如指掌， 小艾与一部分资深的测试架构师聊过，他们都觉得有一定的开发背景对他们成长为测试架 构师具有非常大的帮助，因为只有真正地懂得软件开发的技能，才能深刻地理解开发人员 程序的弱点，从而发现软件中隐藏得很深的大问题。

好的测试架构师一般都能够直接读懂开发人员的代码，甚至能够在单元测试层面或者 代码层面发现问题，从而在第一时间及早发现问题，极大地降低测试成本。即使测试架构 师不亲自读代码，在听了开发人员介绍了代码逻辑之后，也能够基本上了解代码逻辑哪里 可能会有问题，并提醒开发人员在编程时尽量避免出现这类问题。

**4.** **对产品及其商业模式的理解**

要测试的软件产品最终是要交付给客户使用的，只有真正地理解客户是如何使用这个 产品的，才能从客户的角度来测试这个产品，从而把缺陷扼杀在交付给客户之前。

例如，要测试一个 B2B 电子商务产品，就必须要了解下面的这些业务知识：

什么是电子商务?

什么是B2B 业务模式?

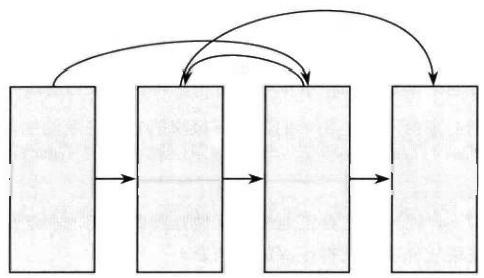
在没有使用这个软件平台之前，用户一般是如何从事B2B业务的?

用户将来怎样使用这个软件平台开展B2B业务?

..

所以测试架构师一定同时也是一个产品专家，他需要对产品的各个组件都有一定的了 解。例如，有个非常庞大的系统，共有4个业务组件，每个业务组件也有非常多的功能点， 有4个测试人员，每个人负责测试一个组件。最常见的业务流程是业务组件1→业务组件 2 →业务组件3 →业务组件4,可是这样测试完毕就真的没有问题了吗?实际上客户可能还 有个经常操作的业务流程是业务组件1→业务组件3→业务组件2→业务组件4。测试架构 师需要敏锐地察觉到这类交叉问题。很多超大型的软件会出现交叉问题，如图13-3所示。

从菜鸟到测试架构师—一个测试工程师的成长日记



3

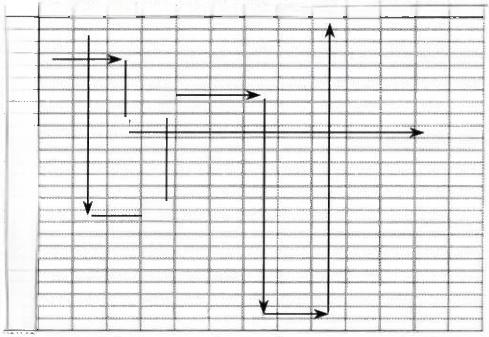
1

2

业务组件1 业务组件2- 业务组件3一 业务组件4

图13-3 大型软件的跨业务组件流程交叉

当然也有很多工具与方法可以帮助分析这类问题，例如图13-4所示的矩阵图就可以 引导我们的思维，帮助我们分析哪些跨组件流程路径没有被覆盖到，从而可以发现功能点、 业务流程覆盖率的漏洞。



组 件 1 组件2 组件3 组 件 组件5 组件6 组件7 组件6 组件9 相件10 组件11 组件12 组件13 绍件14 望件15 组件16 组 件 1 组件18 组件19 组件20 组件21 组件22 组件23 组件24 组件26 望件26

组件1 组件2 组件3 维件4 组件5 组件6 组件7 组件8 组 件 9 相件101组件11组件121组件13「

图13-4 跨业务组件流程矩阵分析图

**5.** **了解测试自动化**

测试自动化是软件测试中一个非常重要的手段，合理地构建自动化测试架构与体系能 够极大地提高软件测试的效率与质量。而且有的测试必须依赖于测试自动化才能完成，例 如性能测试、 API 测试，这些测试无法通过手工方式完成。试想一个系统上线后要支撑 10000个并发用户访问，总不能找10000万个人去同时掐表开始点击测试吧?这时测试 自动化就发挥了无可替代的作用。

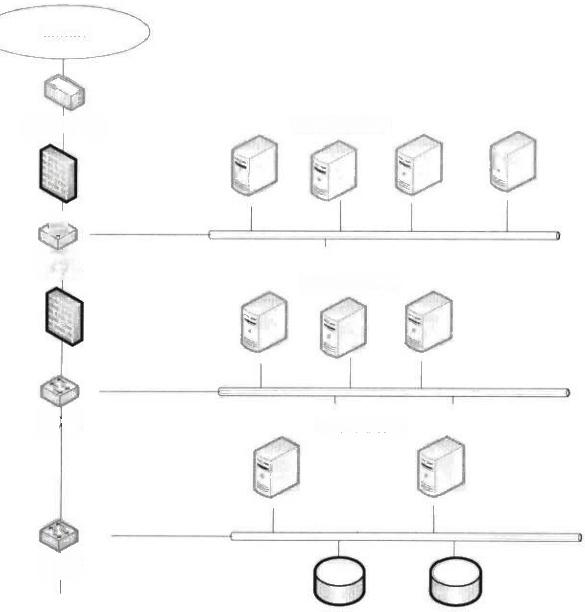
第13章 涅槃!华山论剑：测试架构师的诞生

但是如果不能科学地使用测试自动化，往往可能适得其反。作为测试架构师，必须要 对测试自动化有一定的了解，能够清晰地判断什么情况下应该使用自动化，什么情况下应 该慎重或者不应该使用测试自动化。对于可以使用的现有测试工具或软件，架构师应该清 楚各类自动化测试软件的适用范围及效果如何，以判断是否能够满足测试需求。

测试架构师需要共同参与测试自动化测试架构的设计，以确保要构建的架构与框架体 系确实是符合项目测试战略需要的，并且能具有较高的投资回报率。

**6.** **本产品之外的一些通用技术**

现代软件架构也越来越松耦合，很多软件都构建在中间件平台之上，例如对于一个电 子商务应用来说，图13-5是常见的比较简单的部署架构图。



Internet

链路负载均衡

交换机

交掉机

交操机

存储 存储

Web服务器集群

应用服务器集群

数据库服务器

图13-5 电子商务应用常见的一种简单的部署架构图

从菜鸟到测试架构师———个测试工程师的成长日记

所以测试架构师对于本产品所依赖的基础平台技术也必须要有一定的了解，包括：

操作系统知识；

中间件平台知识，例如应用服务器、消息中间件；

数据库知识；

网络知识；

硬件方面的一些基本知识。

**13.3.2** **软技能修炼**

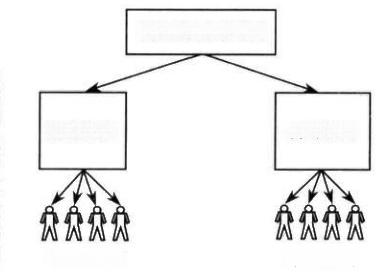
测试架构师光掌握技术就行了吗?前辈们已经无数次地告诉了小艾这个问题的答案： 架构师需要具备非常高的综合素质。光掌握技术是搞不定的，必须依赖技术技能与软技能 的完美结合才能胜任这个工作。测试架构师必须要具备的软技能如下。

**1.** **领导力**

什么是测试架构师的领导力?是对他人的影响能力，能够让他们遵循你制定的战略方 向与技术决策。大家都知道，IT 行业的人基本上都是拔尖人才，汇集了最聪明的一拨脑袋， 这些脑袋可都是高智商、高学历。领导这么一堆聪明的脑袋可不是一件简单的事情，每个 聪明的脑袋都有自己的想法，而且很多技术人员都有一股韧劲儿，凭什么就要听你的? 测试架构师是测试团队在技术方面的总负责人，而测试架构师一般并不一定是经理，如 图13-6所示的这个组织结构里面测试架构师手下并无一兵一卒。



测试架构师



测试总监/高级经理

测试经理1 测试经理2

测试工程师 测试工程师

图13-6 测试团队组织架构

第13章 涅槃!华山论剑：测试架构师的诞生

所以，要让这个不属于你直接管辖的团队实现你所制定的战略方向与目标，就需要非 凡的领导力。那么如何才能培养这种能力呢?小艾在凯文的建议下参加了公司组织的领导 力培训课程，通过该课程，小艾学习到了一些方法。

**如何提高个人领导力?**

(1)要赢得他人的尊重，你必须要具备坚实的专业能力，能够对组织有大的贡献。 作为测试架构师，你的技术能力必须要得到测试工程师们的广泛认可，这样大家自然就 会听你的。

(2)能够为他人提供方向。要领导他人，首先你本人需要非常清楚目前的团队处于 什么位置，团队的下一步发展目标在哪里，并且能够制定决策确保目标的实现。

(3)让团队清晰地理解你的决策。要知道团队里面的人背景不完全一样，工作经验 有长有短，并不是每个人都能够真正理解你为什么要让大家这么做。所以要想让你的决 策能够得到不偏不倚的执行，就必须要让大家清晰地理解你的决策。

(4)要善于倾听。测试架构师虽然资历深厚，但是绝不能有技术霸权主义思想，测 试工程师们的很多想法可能非常值得你去了解与学习。

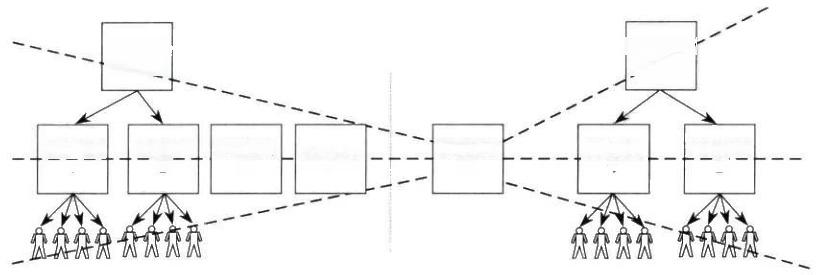
(5)不要刻意回避冲突。项目中的冲突是不可避免的，刻意地回避冲突并不能解决 问题。只有敢于正视冲突，行动起来，采取积极有效的有助于问题解决的行动，才能推 动项目的进展。

(6)激励团队。经理们会激励团队，但是作为测试架构师，由于与测试工程师们共 同奋战在一线，同样也要激励团队，这样才能共同营造一个高效的团队氛围。

**2.** **沟通能力**

测试架构师不仅要与测试人员保持很好的沟通，更要与测试管理团队的各级经理们保 持很好的沟通，保持与管理团队的良好沟通可以获取管理团队的支持，有助于测试战略、 技术与方法的深入贯彻执行，如图13-7所示。

此外，测试架构师也需要从质量保证的角度与开发团队进行沟通，例如沟通缺陷分析 报告、共同研究如何提高开发质量等。与产品经理沟通，以便更准确地理解客户需求，使 测试活动能够尽可能地贴近客户的实际业务场景，在研发阶段预见并解决可能在客户的环 境中出现的问题。

测试总监/ 高级经理

开发总监/ 高级经理

测试经理 测试经理

开发架构 师

开发经理

2

测试架构 师

开发经理

1

产品经理

1 2

开发工程师 开发工程师 测试工程师 测试工程师

**图13-7** **测试架构师的沟通能力**

**3.** **项目管理能力**

测试架构师同样需要实时追踪项目进度，对总体测试活动进行周密的安排，与管理团 队沟通测试资源的协调与安排，结合项目总体资源概况向管理团队提供资源配置合理化建 议。在项目进行过程中评估质量风险并做好质量风险控制。因此项目管理能力也是测试架 构师的必备软技能之一。

**4.** **担当导师的能力**

要让测试人员正确地理解与贯彻测试架构师制定的战略、技术与方法，测试架构师需 要给予测试人员进行必要的指导，测试架构师需要担当技术导师的职责，与管理团队一起 共同促进整个测试团队的技能储备与进一步提升。

**13.4** **测试“九阴真经”——总体测试计划**

当前版本的开发测试即将进入尾声了，新的一个版本的研发工作即将开始了。各位架 构师和经理们都开始忙活下一个版本的计划了。

开发的架构师们已经陆续提交了下一个版本各自所负责的解决方案的设计方案，并且 管理层已经在正式评审与筛选了。小艾作为测试架构师，理所当然地参与到了这个过程当 中。他开始着手编写下一个软件版本的总体测试计划了。

第13章 涅槃!华山论剑：测试架构师的诞生

**什么是总体测试计划?**

总体测试计划是一个软件版本的测试的战略性纲领文件，一般在一个软件版本计划 阶段开始编写，在正式进入该版本的开发测试工作后逐渐定稿。测试总体计划一般由测 试架构师编写，它一般包含下面这些内容：

(1)描述总体的测试范围；

(2)描述该版本测试中所发生的所有测试活动；

(3)描述测试方法、技术与工具；

(4)确定各测试小组之间的分工，避免重复性劳动或者测试点的遗漏；

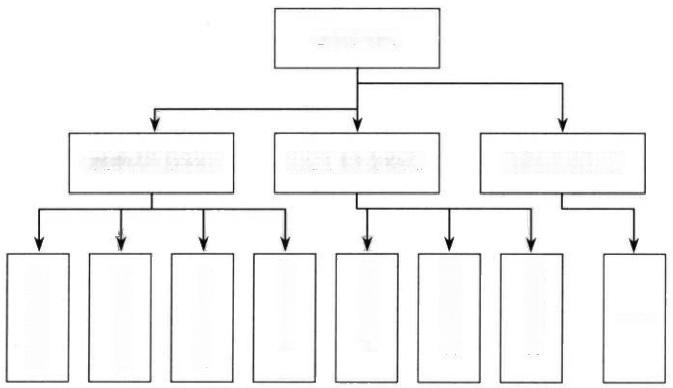
(5)确定个测试小组之间的相互依赖关系与开始的先后顺序，提高测试资源的使 用 率 ；

(6)预估测试过程中可能碰到的风险因素，并提出避免风险发生的方案与建议；

(7)确定测试时间安排与测试阶段的安排；

(8)确定缺陷报告的流程事宜。

图13-8 描述了总体测试计划与具体的测试计划、测试用例之间的关系，可以看出所 有的具体的测试计划、测试用例都要继承总体测试计划的指导思想。



总体测试计划

功能测试组测试计划 性能测试组测试计划 鬃测试组测试计划

....

模 块 3 测 试 用 例

模 块 2 测 试 用 例

模 块 1 测 试 用 例

扩 展 性 测 试 用 例

可 靠 性 测 试 用 例

集 成 测 试 用 例

压 力 测 试 用 例

图13-8 总体测试计划结构图

从菜鸟到测试架构师 — — — 个测试工程师的成长日记

总体测试计划也是需要经过评审的，以确保大家真正理解总体测试计划中制定的测试 策略与方案，并且可以民主地解决测试项目组之间的分歧。

一般来说，如图13-9所示的人员应当被邀请参加总体测试计划评审。

开发首席架构师

各测试项目组负责人

A M

各开发项目组技术负责人、架构师

本产品研发管理团队

总体测试计划 评 审

图13-9 总体测试计划评审

**13.5** **将缺陷扼杀在萌芽阶段**

从 一 个 研 发 团 队 的 总 体 研 发 成 本 的 角 度 考 虑 ， 缺 陷 发 现 得 越 早 ， 缺 陷 修 复 的 成 本 越 少 。 图13-10展示了关于缺陷在产品研发交付的不同阶段被发现给研发团队带来的成本的比较。

**成本**

**阶段**

需求分析 设计 编码 集成测试 验收测试 生产环境

图13- 10 缺陷在产品研发交付的不同阶段被发现给研发团队带来的成本

第13章 涅槃!华山论剑：测试架构师的诞生

从图13-10中可以观察到这样一个规律，晚期阶段发现的缺陷所需要花费的成本与早 期阶段发现缺陷所需要花费的成本呈指数增长态势，晚期发现缺陷需要比早期发现缺陷多 花费10～100倍的成本。

这也不难理解，例如如果一个系统有一个重大的缺陷，这个缺陷在客户复杂的生产环 境下会出现，而由于测试部门在公司内部进行测试时并没有考虑到客户的实际情况，漏掉 了这个缺陷，系统在客户现场部署上线时客户发现了这个缺陷，客户满意度立刻下降，公 司不得不派更多的人出差去客户现场解决这个问题，恰恰这个问题反映出当初在需求调研 阶段就没有深刻理解客户的需求，从而导致在设计阶段没有考虑过这个问题，于是为了解 决这个问题，需要立刻重新做设计、重新编码、重新测试，由此导致的研发成本的急剧增 加是显而易见的。

测试架构师作为研发团队中关键的技术领导者之一，必须要在项目的初期就开始与产 品经理、开发架构师们保持沟通，预估可能会出现的问题，并制定详尽的行动计划，将可 能会出现的问题尽可能地在项目的早期阶段就充分考虑到，从而将测试团队的工作方式由 一直被动发现缺陷改为关注产品研发的完整生命周期，防患于未然，尽可能地提早发现和 解决缺陷。

为了将这个精神落实到实处，小艾制定了以下行动方案。

(1)要求测试人员积极参与到项目的前期阶段，在项目的需求确立、设计阶段就参与 进去，从客户的立场出发以客户的眼光去阅读、评审项目文档，努力在前期能够发现问题 并推动问题在前期解决。

(2)与开发架构师们一起改进研发流程，进一步强化单元测试的重要性，推动单元测 试覆盖率的提高。为了使得这一措施可控，小艾将单元测试的通过率作为测试团队正式开 始测试的一个前提条件，测试人员在进入测试阶段时需要审查开发人员的单元测试报告， 确认单元测试确实已经有合理的覆盖率并且成功运行通过，这样就可以把功能测试阶段的 一部分缺陷前移到单元测试阶段解决，从而让功能测试人员将更多的精力放在查找开发环 境难以发现的问题上来，降低问题被层层遗漏的风险。

(3)在测试阶段将客户的场景考虑进来，例如客户的部署方案、性能指标、与客户现 有系统的集成方式、业务增长所带来的数据量的增长趋势等信息，将这些信息纳入到测试 团队的工作内容中去，避免将问题留给客户。

从菜鸟到测试架构师—一个测试工程师的成长日记

**13.6** **火眼金睛——到底哪儿被测试遗漏了**



为了清晰地掌握总体测试进度与质量，小艾要求各测试项目组每周一报告一次测试进 度，各测试项目组也确实把测试报告交上来了，大家的报告写得还真不错，样式也很赏心 悦目，用了各种图表、颜色来展示测试项目所取得的测试进展、碰到的问题及已经解决的 问题、亟待解决的问题等。小艾看着这些花花绿绿的报告，一方面对测试项目组的兄弟姐 妹们的进展感到欣慰，另一方面随着项目的进行，小艾也觉得心里越来越没底，大家的测 试看起来状态还不错，可是真的测到点子上了吗?写的测试用例越多真的就能证明测试得 越彻底吗?

卢涛的话还在小艾耳边回响：“衡量测试做的好坏并不在于你已经测试了多少个用例， 也不取决于你加了多少次班测试了多长时间，而在于你到底遗漏了多少个缺陷到了客户那 里。所以只有能证明遗漏的缺陷少才能证明测试做得好”。小艾想了想，不禁倒吸了一口 凉气，镇定了一下，看来不能被虚假的繁荣给蒙蔽了双眼，要有一双火眼金睛，能够察觉 到底什么地方没被测试到才是更重要的。

那么这双火眼金睛是什么呢?小艾想起要做代码覆盖率测试。

**13.6.1** **什么是代码覆盖率测试**

代码覆盖率是软件测试领域中的一个非常重要的衡量方法，它可以用来衡量开发人员 写的代码有哪些已经被测试覆盖到了，还有哪些从来就没被测试覆盖过。由于代码覆盖率 测试是基于代码层面做出的分析报告，所以属于白盒测试概念范畴。

代码覆盖率提供了一种量化的度量标准，具体来说可以做到以下方面的覆盖率检查。

**1.** **代码语句覆盖率**

简单来说，就是代码行级别的覆盖率检查。假设要测试的软件有一万行代码，那么代 码语句覆盖率报告可以告诉你这一万行代码有多少行、哪些行从来没被测试覆盖过，以及 哪些已经被测试覆盖了。

**2.** **代码条件判定覆盖率**

很多代码中有大量的条件判定，所以测试场景的步骤未必能覆盖到所有的条件判定分

第13章 涅槃!华山论剑：测试架构师的诞生

支，代码条件判定覆盖率可以告诉你哪些条件判定分支没有在测试过程中走到过。

**3.** **路径覆盖率**

用来检查是否每条逻辑路径都已经在测试过程中执行过了。

**4.** **入口出口覆盖率**

用来检查是否每个调用与结果返回都已经在测试过程中执行过了。

当然，以上这些覆盖类型通常都是交织在一起的，例如路径覆盖也隐含着包括条件判 定覆盖检查。

**13.6.2** **代码覆盖率测试工具**

市场上有很多种代码覆盖率测试工具，当然对于不同的开发语言也会有不同的工具。 对于Java 应用程序来说，常见的覆盖率测试工具有：

1.IBM Rational PureCoverage

2.Cobertura

3.Clover

4.DevPartner

5.EMMA

6.Jtest

7.Kalistick

8.LDRA Testbed

由于公司办公楼的三层就是IBM Rational PureCoverage研发团队，小艾当然首选这个 工具。小艾研究了一下这个工具。

Rational PureCoverage 是 IBM Rational PurifyPlus 套件的组件之一，IBM Rational PurifyPlus 具有三个主要组件：内存分析的 Purify 、性能分析的 Quantify 和代码覆盖分析 的 PureCoverage。



1 参考文献： <http://www.ibm.com/developerworks/cn/rational/rationaledge/content/apr07/begic_pratt/>

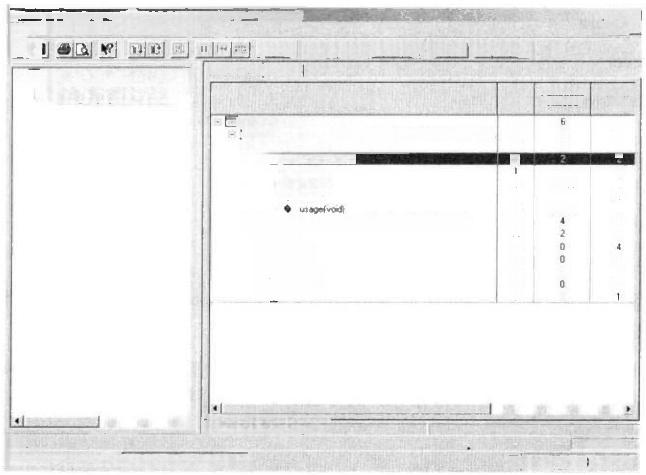
对于C/C++ 程序，Purify 可以自动化地检测并报告内存泄漏和内存访问错误，例如， 在内存释放后使用内存，重复释放相同区域的内存，或是初始化前从内存读。所有这些 都是潜伏于程序中的软件质量问题，即使通过了所有测试。这类错误可以导致严重的生 产错误。

对于 Java 和 Microsoft .NET程序(“已管理的程序”),Purify 追踪内存的使用和内 存引用，因此，你能够发现哪里是内存瓶颈，应在哪里释放内存，比对前后图形以检测程 序中未预料的内存使用(“泄漏”)。

Quantify 追踪程序性能和调用行为，因此你能够发现执行流并识别出瓶颈。 Quantify 强调了使用 River of Time(tm) 特征的花费最久的代码路径。除此之外，Quantify 为你 可视化地实现了线程执行行为。

PureCoverage 追踪代码覆盖，因此你可以发现测试与运行时分析工具不能发现的鸿 沟。由于Purify 和 Quantify 仅仅看到了实际运行的代码，PureCoverage 提供的代码覆盖矩 阵重点了解正在检测和更新的质量。

PureCoverage 可以从多个维度展现覆盖率，举例如图13-11～图13-13所示。



Browser:CPFxanple.enel

△Z 陋 中 区 口 围 回 + +

Module Wew Fle Vew|

Function

Coverage Item

Run31/01/200716.0901a

□ CDocunerts and Settngr\atepharhDesktopLCPFExample

CPURECOVERAGE CPPExamole

国 EPFExamde eso

alvod]

9 bvod)

man

nessed

国epm: fleomicroet isual studolve98%ncade

· ioalwd 日 oubeam

· 日 shreanbu 目 sotbate

4 koede

coverage Rtem:Ascendng odr

aadyrt

由 世 ”△ Rational PureCoverag-

□ CDocuments and Settingsistechanh Auto Merge31/01/200716:09:

□ Run@31/01/200716:09:01a

Ratonal PureCoverage [Coverage

FLe Edt Vew Settings Wndow He

Calls

152

152

2

2

0

1

0

150

14

58

18

28

27

5

Hit

30

30

2

h

hi

28

3

10

6

× 6×

Functionr *Mised*

NM

4:10 PM

盈“①Ⅲ

nitted

2

0

6

2

图13-11 代码模块/文件级别覆盖率

第 1 3 章 涅槃!华山论剑：测试架构师的诞生

Rational PureCoveraoe -Tunctio onList:CPPEcample.ese] 8×

rie EEdt View Settings window Hep ×

△ ∑ 临 都 图 国 制 国

口cDocments and settngsistechah Function

a b

Auto Merge31/01/200716:091

□Runo31/01/200716:09018

man

d bsK\_ioscchar,chat\_ta(cha std d baic\_otcha.cha\_ta;chan t

td baic\_onchw.chaL\_tati(chan stΦ d bacc\_ioschachar\_Iat:(chan sd ad bsic\_orccha char\_tati(chao std sd banc\_othesmcchat.cha\_iats(cha ad barc\_osheamcha,chu\_taticchao sd baic\_othesncchas.chu\_tots(chs sd basc\_oreamccha.cha\_iatcha ad baic\_oeamchat.cha\_trats<ch

sd baie\_obeamccha,cha\_Itaticcha td baic\_osheam(cha,chu\_tatschao d base\_otheamchs.char\_tratscchan ad bate\_tesnbuccha.chw\_tatacch d barc\_teanbuccha cho\_taticchy ad bae\_助esnbuk(cha.chs\_tatsccha sd bare\_theanbukcha chw\_taticcha ad bare\_iteanbukchu.chu tatschs d basc\_heanbukcha cha\_tatcchy d cho\_iatacchan eol

sd:chs\_Iratucha eqr\_type sd cha\_natacchan lkergth

t时d chs\_trat:cha)to\_intype ad ctype(WORD>id

山

Functon:36/1 Functione

Ready

生start后 ” **Rational** **PureCoverag…**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cala  1  0  0  0  6  6 2 14 2 2 1 2 2 2 o  0  22  4  1  1  22  22  3  11  5 | Lines  1 2 1 1 2 1 1 1 1  Total  3  3  12  26  2  1  5  3  2  11  2  1  1  2  1  1 | Lines  Mitned  4  10  4 1 1 1 0 3 o 0 o 2 1 0 0  0  0  0  0  0  0  0  0  0  0 | Lines  Hi  3  16  0  1.  1  0 8 0 0 2 11 2 2 8 2 11 0 2 11 1 1 1 1 | 2.Line+  Hit  10000  0.00  6667  6154  000  0.00 10000  100.00 100.00 100.00 80.00 66.67 100.00 7273 10000 10000 100.00 0.00 0.00 100.00 100.00 100.00 10000 100.00  100.00  10000 10000 100.00 | Module  CDocument CADocumert  CADocmerns CDocuoen  CDocumer  CMDocumerd  CMDocumers  CMDocunerds  CMDocumerk  CMDocumerds  CMDocuners  C.MDocumert CMDocunerks CDocuner  CDocumerki C:Documer  C Documert C:Documerls C.Documer:  C.Documers CADocumerts  C Documer; CMDocunerts  C Docunerts CMDocunerts CADocunents CDocuents  C\Documerts |

gd:<c(bak\_oreamcchar,dhw\_tatscdha:sd>:stde,char co MM

国“U》4:10PM

图13-12 函数/方法级别覆盖率

R~~aFtio~~n~~Eadlt~~ P~~V~~~~ecSvt~~~~geW~~r-LOVERAGEC  6××

盛 □ C 11 C Z 鸠 夏 国

Frctinu an Color

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Line  Coverage | | Line  Number | Source | | |  |
|  | 1 1 1 1 1 0  0  0  0  0  0  0  0  1  0  0  0 | | 11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | voie b()  cout “Method b()° endl.  )  void usege()(  cout "Dodgey C++progran(e)gmurphy 2006<endl coute"Enter either e or b as input argunents " r endl.  )  int sain(int arge.char\*argv[) (  1f(ar0110  11{strcapiargv[1],“”)- ·0}  a()  Jelse istr(3rgvL11.““10  b()  else(  usoge():  )else(  usage(): | | |  |
| ne:23038 Functiore man | | | | | | |
| Reedy  start | |  | | | | MM |  |
| 世 | | | ”Rational PureCoverag- “ 步 多 4 1 1 | |

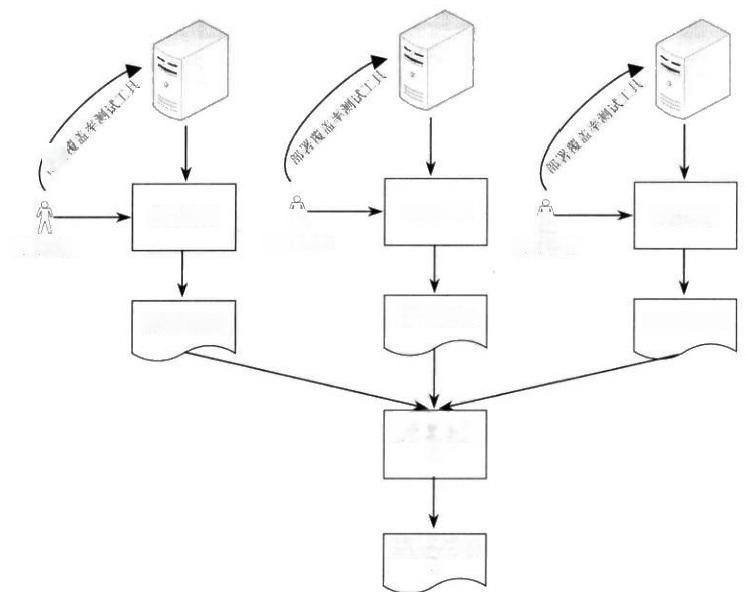
图13-13 代码行级别覆盖率

从菜鸟到测试架构师———个测试工程师的成长日记

**13.6.3** **代码覆盖率测试实战**

小艾研究了一段时间代码覆盖率理论与工具之后，决定在测试团队中正式应用这一方 法来查找测试遗漏的代码。

具体实战步骤如图13-14所示。

执行测试2

执行测试2

A

测试人员2

执行测试1

人

测试人员1

测试人员2

覆盖率报告2

覆盖率报告2

覆盖率报告1

覆盖率结果合 并

总体覆盖率报 告

图13-14 代码覆盖率实战步骤

(1)小艾所在的项目组要测试的是一个基于Java EE 架构的电子商务产品，部署在IBM WebSphere 应用服务器上。要得到代码覆盖率测试数据，第一步就是要在测试服务器端部 署代码覆盖率测试工具。

(2)测试人员基于已经部署了代码覆盖率测试工具的服务器进行正常测试，并收集覆 盖率结果。

第13章 涅槃!华山论剑：测试架构师的诞生

(3)如果所有的测试人员都是基于同一个服务器进行测试的，那么从服务器端获得的 覆盖率报告自然已经自动汇总了所有人的测试覆盖率。但是小艾所在的项目组由于测试人 员较多，所以为了避免测试人员之间的测试操作冲突，测试人员用了多台测试服务器。所 以就需要对所有测试人员的测试覆盖率结果进行合并。因为单个测试人员的覆盖率并没有 多少参考意义，测试人员甲并没有测到的地方可能被测试人员乙覆盖到了，所以测试人员 甲的覆盖率报告并不能证明测试人员甲遗漏了多少代码没被测到，这就是要对所有测试人 员的测试覆盖率结果进行合并的原因。

代码覆盖率的汇总报告出来了：

(1)总体代码方法级别覆盖率50%。

(2)按模块分析(本项目测试的是一个电子商务应用系统，模块划分参见本书第5章 图5-5)。

在所有的模块中库存管理模块方法级覆盖率最低，只有30%。

订单管理模块代码在测试过程被执行的重复次数最多。

于是小艾制订了以下行动方案：

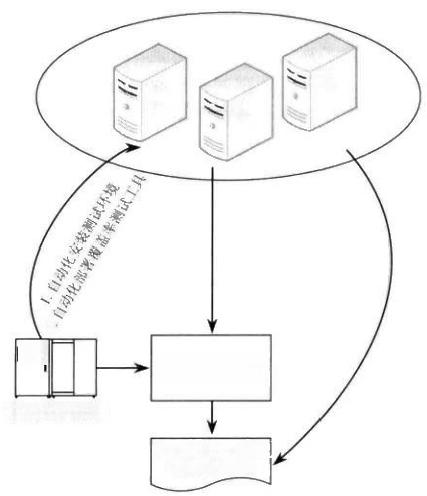
(1)安排测试项目组负责人与开发技术负责人之间的讨论，分析判断被遗漏的未测代 码的风险，并进一步优化测试用例的覆盖范围，增加新的测试用例用来覆盖未测代码的那 一部分逻辑流程。

(2)要求各测试项目组增加对库存管理模块的测试覆盖，增加更多的库存管理模块的 测试用例。

(3)进一步优化测试用例在不同模块之间的分布，减少一部分不必要的重复的订单模 块的用例，以降低不必要的测试工作量的浪费。

**13.6.4** **代码覆盖率测试与自动化测试的集成**

小艾也在实战中深刻体会到了基于手工测试做代码覆盖率测试效率实在比较低，要得 到一个覆盖率报告，要花比较长的时间才能完成。于是小艾与自动化架构师Steve 同学进 行了商讨，决定将自动化覆盖率测试与自动化执行集成在一起，制订了如图13-15所示的 自动化解决方案。这样每次自动化测试脚本执行完毕，就能够自动发布一个代码覆盖率测 试报告，极大地提高了效率。

V

0o

自动化测试引擎

3. 自动化执行测试

4. 自动化收集覆盖 率报告并合并

图13-15 代码覆盖率测试与自动化测试的集成

**13.7** **测试过程改进**

项目进行到中途，大家的抱怨也越来越多，很多人开始抱怨要验证一个缺陷，需要好 几天的周期才能搞定。小艾要求各测试项目组负责人回去调查，到底是哪个环节效率这么 低下。小艾想起了价值流程图 (Value Stream Map) 这一工具，他要求大家用这个工具分 析一下当前的工作流程并记录各阶段的时间花费。

**13.7.1** **什么是价值流程图**

价值流程图 (Value Stream Map) 是丰田精益制造生产系统框架下的一种用来描述物 流和信息流的形象化工具。¹它运用精益制造的工具和技术来帮助企业理解和精简生产流



1 参考文献：<http://en.wikipedia.org/wiki/Value_stream_mapping>

第13章 涅槃!华山论剑：测试架构师的诞生

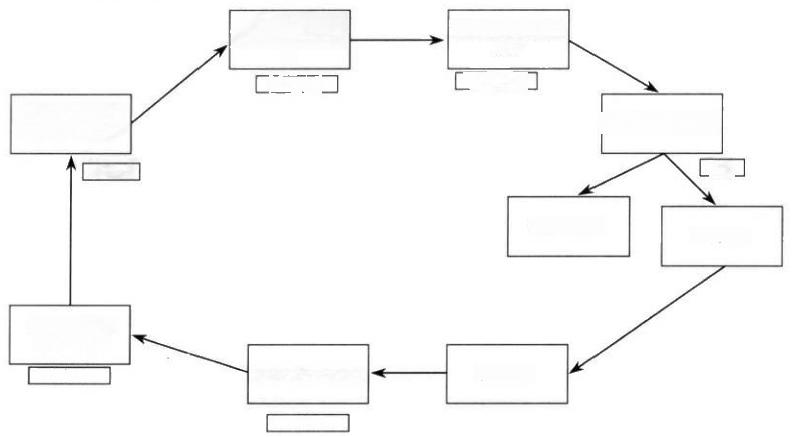
程。价值流程图的主要目的是为了发现生产过程中的浪费，“浪费”在这里是指不能够为 终端产品提供增值的任何活动。

价值流程图可以作为管理人员、工程师、生产制造人员、流程规划人员、供应商及顾 客发现浪费、寻找浪费根源的起点。另一方面，价值流程图也是一种沟通工具、战略工具、 变革管理工具。

现在价值流程图也被很多软件公司引入到软件研发管理中，可以帮助软件研发团队查 找时间是在哪儿被花费的，时间是被花费在有意义的能创造价值的活动上还是被浪费在无 意义的活动上。

**13.7.2** **时间被消耗在了哪些地方**

测试团队的缺陷流程的价值流程图分析出来了，如图13-16所示。



测试人员恢复镜像

文件

30分钟二

测试人员进行测试

4天

没发现缺陷

构建验证测试 (美国同事)

报告缺陷

1天

测试人员下载镜像

文件到自己的服务

器

2小时

上传测试环境镜像 文件到中国服务器

6小时

开发人员修补缺陷

发现缺陷

2小时二

图13-16 价值流程图-缺陷流程

小艾看了看，发现了以下问题：

(1)构建与镜像文件传输花了太长时间，总共花了6+2=8小时，浪费了很多时间。

(2)测试人员的测试周期有些长，要花4天才能测试完成。在深入剖析之后，发现测

试人员将近85%的时间所进行的测试都没有发现缺陷，这说明测试人员的测试可能没有测 到点子上，测试有些盲目，缺乏针对性。很多测试用例的执行是不必要的，因为代码变更 根本就与这些测试用例无关。

于是小艾终于基本明白大家既觉得累又没发现缺陷的原因了，价值流程图虽然简单， 但是确实非常有效。小艾决定在测试项目团队中采取以下措施进行改进：

(1)与美国的同事共同开发一种新的文件传输工具，利用多线程传输大大加快美国与 中国之间的文件传输。在新的传输工具部署之后，终于将文件传输的时间从6小时降到了 半个小时，极大地提高了效率。

(2)关于本地镜像文件下载竟然也需要花费2个小时的问题，小艾经调查后发现瓶颈 出现在网络层，公司的网络是100Mb/s 网，传输效率比较低。于是小艾制定了一个局域网 升级成千兆网的计划，经过网络改造以后，局域网文件传输时间从2个小时降到了20分钟。

(3)针对测试用例执行很多但是并没有发现几个缺陷的问题，小艾要求各测试项目组 与开发人员加强沟通，筛除掉了50%的无关紧要的测试用例。

**13.8** **测试驱动的软件设计变革**

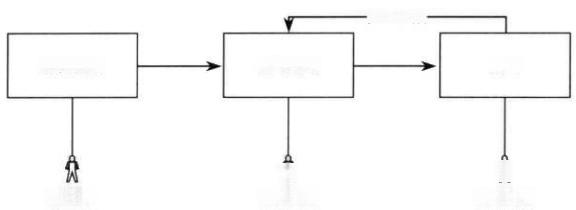
测试人员埋头完成测试任务仅仅是测试人员的基本工作职责， 一个优秀的测试人员不 仅能高质量地完成基本工作，而且能对软件研发生命周期的上下游环节有一定的贡献。测 试人员同样可以对推动软件设计的变革。

**13.8.1** **测试人员没有主动权吗**

在测试人员的心目中，可能大部分人都会认为测试人员与开发人员、架构师团队是图 13-17所示的这个关系。

在这个思维模式下，测试人员所做的事情就是当开发人员宣称可以进入测试时测试人 员正式开始做测试，测完把缺陷报告给开发人员，等待开发人员修补缺陷，然后测试人员 进行验证。在这样的一个工作模式下，测试人员往往觉得测试团队很不受重视，因为测试 人员是被缺陷驱动的测试执行者，测试人员日复一日地辛辛苦苦进行测试，但是始终缺乏 成就感。

第13章 涅槃!华山论剑：测试架构师的诞生



—报告缺陷一

编写代码

众

开发人员

测试人员

架构设计

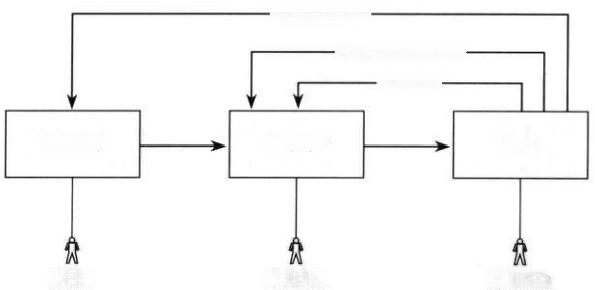
架构师

测试

图13-17 测试人员与开发人员、架构师团队的关系

导致这样一个工作模式存在的一个更深层次的原因是在很多测试团队中，测试人员的 技术能力不是很扎实，很多时候是测试人员认为自己不懂编程，所以就选择从事测试工程 师的职位了。在这种思维模式主导下，很多测试人员确实能够发现不少缺陷，但是往往对 为什么会导致这个缺陷知其然不知所以然，从而导致治标不治本，缺陷虽然开了很多，但是 貌似问题还是继续发生，甚至缺陷解决了很多但是新开的更多。更严重一点的后果是由于 测试人员不理解缺陷产生的根本原因，所以有可能漏掉一些与这个缺陷相关的一系列问题。

实际上，在一个成熟的研发组织里，测试人员也有推动软件设计变革的机会，前提是 测试人员具备较强的技术能力与开放的思维模式。如图13-18所示，好的测试人员不仅仅 是发现缺陷然后报告缺陷，而且可以分析缺陷产生的深层次原因，甚至可以从代码层面分 析出开发人员代码质量问题，从质量的角度去推动开发质量的提高。另一方面，测试人员 还可以影响产品的架构设计，例如性能相关的部署架构，因为测试人员可以具有比开发人 员更好的全局观，测试人员是从整个产品的角度去看这个产品。



—帮助改进设计 —

帮助提高开发质量一

—报告缺陷一

编写代码 测 试

开发人员 测试人员

架构设计

架构师

图13-18 测试人员对架构设计与开发质量的贡献

从菜鸟到测试架构师——一个测试工程师的成长日记

所以，测试人员的思维方式往往束缚了自己，从而影响测试价值的发挥。测试与开发 都属于研发团队的一部分，测试人员同样需要具备较强的技术能力，也更需要摆脱思维定 势的思维模式，这样才能让测试团队的更多价值得以体现。

**13.8.2** **性能引发的软件设计变革**

应用系统的性能是客户体验的一个非常重要的组成部分，现在人们越来越倾向于在电 子商务网站购买物品，如果一个电子商务网站在用户点击之后10s 才能返回页面给用户， 那么用户一定不能接受这样的体验，从而转向竞争对手的网站，直接造成网站用户流失。

另一方面，性能问题也会引发功能问题。例如，某订票网站在订票高峰时段由于大规 模用户并发请求，订票系统瘫痪了，用户的钱被扣除之后页面没有响应了，导致用户订票 失败。这时性能问题就直接导致功能问题了，因为网站的功能不能正常运行了。

为了避免这样的问题产生，需要进行性能驱动的软件架构变革设计。分布式计算已经 成为现今大型应用的基本需求的一部分。如果一个应用只支持在单机上运行，无法进行系 统水平方向的扩展，那么无论如何，对开发人员的程序进行改进与优化也是很难应付超大 规模的用户并发访问的。

在一个产品研发团队中，软件的性能方面的架构设计一般由架构师团队负责，测试架 构师作为架构师团队的一部分，理所当然地需要参与到这个架构设计讨论中。相应地，性 能测试是测试团队职责的一部分，性能测试人员也要对软件架构有一定的了解，这样才能 给架构师团队更多有建设意义的性能改进建议。

测试团队尤其是性能测试团队人员需要深刻理解生产环境的架构，并从软件生产环境 的角度来审视自己测试的这一部分是否真的能够在客户环境中正确运行。

下面来看一个简单的例子。假设开发人员做了一个用户访问计数系统，部署在 IBM WebSphere 应用服务器上，这个系统的功能是用户每访问一次计数器 visitorCount 就自动 加1,并在页面上显示你是访问本网站的第几位访客。这个 visitorCount 数值存在于应用 服务器中，如图13-19所示。

测试人员安装了一个测试环境，有一个 IBM WebSphere应用服务器，部署成功了。测 试人员进行了多角度的测试，例如基本功能测试、边界测试、并发测试等，怎么测试都 没问题。可是在生产环境真的没有问题了吗?我们看下面的例子，这是生产环境的部署 情况。

第13章 涅槃!华山论剑：测试架构师的诞生

A

用户

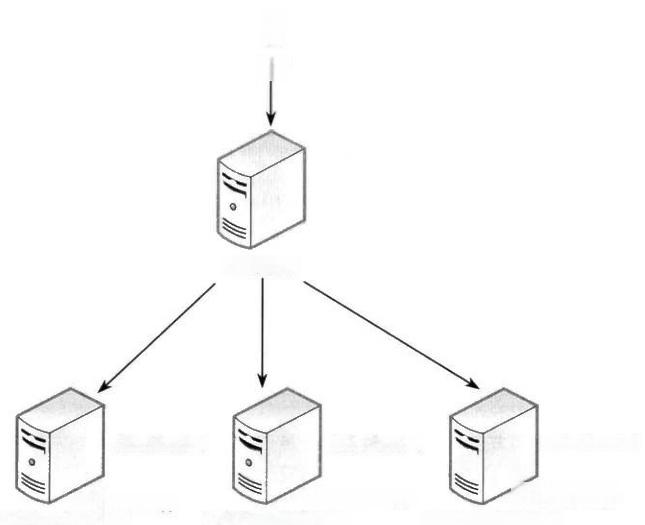
visitorCount=1

VisitorCountApp

WebSphere 应用服务器

图13-19 计数系统单节点部署

为了支撑更多的并发用户访问，系统采用了分布式部署架构，架设了一台 Web 服务 器及三台IBM WebSphere 集群节点服务器， Session 管理采用了Session Affinity 机制，应 用服务器负载均衡采用了Round Robin方式，如图13-20所示。



穴

用户

Web服务器

visitorCount=1 visitorCount=1

0

VisitorCountApp VisitorCountApp VisitorCountApp WebSphere集群节点1 WebSphere集群节点2 WebSphere集群节点3

visitorCount=1

图13-20 计数系统分布式部署架构

从菜鸟到测试架构师——一个测试工程师的成长日记

在这样的部署架构下，问题出现了：

(1)第1个用户访问该系统，请求被派发到了WebSphere 集群节点1上， visitorCount 自动加1,visitorCount=1 。 用户在页面上看到的是“您是访问本网站的第1位访客”。

(2)第2个用户访问该系统，请求被派发到了WebSphere 集群节点2上，由于这个节 点是被第一次访问，所以visitorCount 在第2个用户访问之前仍然是0,在第2个用户访 问之后visitorCount 自 动 加 1 ,visitorCount 也是1。用户在页面上看到的仍然是“您是访 问本网站的第1位访客”。

可以看到，虽然有两个用户访问了本系统，但是第1个节点服务器统计的 visitorCount=1, 第2个节点服务器统计的visitorCount=1, 第3个节点服务器统计的visitorCount=0 。 该应 用系统虽然在单机部署架构下能很好地工作，但是在一个分布式的部署架构下竟然不能正 确地计算出结果!

因此，测试人员必须要深刻理解所测系统的架构，才能避免遗漏问题给客户。另一方 面，测试人员也可以为所测系统的架构设计方案提供积极建议。测试架构师需要在这方面正 确地引导测试人员，确保测试人员测试过的应用系统在生产环境中能够真正正常地工作。

**13** **.9软件系统的RAS保证**

Reliability (可靠性)、Availability (可用性)、 Serviceability (可服务性)三者的综合 通常简称为RAS, 这一术语是IBM 公司在发布IBM 大型机系统时提出的。

服务器的RAS 已成为衡量服务器“含金量”的重要因素之一，IBM 在几十年的服务 器研究开发中积累了丰富的经验，形成了一整套服务器 RAS 设计的思想体系。其设计体 系贯穿了整个服务器，并对重要部件进行全方位多层次的保护，确保服务器拥有行业领先 的 RAS。

RAS 的概念不仅适用于硬件领域，也已经被引入到了软件领域。软件系统的RAS 可 以理解为：

Reliability (可靠性)指的是软件系统能够正确地运行，尤其是系统在特定环境下 能够正常工作，能够侦测与避免错误发生。



1 参考文献：<http://en.wikipedia.org/wiki/Reliability,_availability_and_serviceability>

第13章 涅槃!华山论剑：测试架构师的诞生

Availability (可用性)指的是软件系统实际正常运行的时间占应当正常运行总时 间的比率，例如一个具有99.99%可用性的系统其宕机时间只有0.01%。许多重要 的业务系统都需要提供7×24(即一天24小时， 一周7天)不间断地服务，如银 行、电信、保险、政府等部门的业务系统。这些业务系统停止运行就意味着巨大 的经济损失，更重要的是，将失去用户的信任。

Serviceability (可服务性)指的是当系统出现问题时能够比较容易地查找到问题 产生的根源。可服务性可以包含两方面的含义：一方面是出问题时客户能够方便 地判断问题所在，另一方面是软件厂商的技术支持人员能够方便地分析问题并解 决问题。

从软件系统的质量保证的角度来看，可靠性、可用性往往可以通过在系统测试阶段设 计一些测试用例来确保，但是可服务性是测试团队常常忽略的一个领域。可服务性给客户 带来的是从问题产生直到问题解决整个过程的端到端的体验，可服务性做得不好也许不会 直接影响到产品的基本功能，但是会对客户满意度产生很大影响。一个产品团队要想给客 户交付一个卓越的产品，产品可服务性是不可忽视的。

所以在一个成熟的产品研发团队中，软件测试架构师也需要将 Serviceability Testing (可用性测试)纳入到产品测试活动中，并积极鼓励测试人员去发现更多的可服务性缺陷。

下面列举了一些常见的可服务性缺陷：

系统无法运行但是日志中没有任何错误信息提示，从而导致很难快速了解到底系 统出了什么问题。

错误信息不准确，客户无法正确理解错误信息。

错误信息误导了客户，例如明明是网络断了但是系统报告了一个其他错误。

软件系统没有收集系统错误日志的功能，技术支持人员不得不频繁地询问系统操 作人员出错时的上下文信息。

为了提高产品的可服务性，从而提高客户满意度，测试架构师可以要求测试团队增加 下面的这些重点测试用例：

**1.Logging** **(日志检查)**

例如：系统正常运行时日志中是否有错误信息；系统出错时日志中是否有清晰的易于

从菜鸟到测试架构师——一个测试工程师的成长日记

理解的错误信息提示。

**2.Tracing (错误跟踪)**

例如：在打开trace 的情况下，当系统出错时Trace log文件中是否能够详细地列举出 系统在出错前后都有什么样的操作步骤被执行过。

**3.** **问题诊断工具**

如果产品附带了问题诊断工具，需要确保当系统产生错误时问题诊断工具是否能够自 动地搜集必要的信息以便技术支持人员诊断问题并解决问题。

**13.10** **学习笔记——测试架构师之小艾观**

小艾当了测试架构师之后，发现要当好这个角色真的不容易，需要充当一个多面手， 具备很好的综合素质才能胜任，技术与软技能一个都不能少。当然，这个角色使小艾考虑 问题的角度也不再局限于原来在一个小项目团队中的那一亩三分地的视野了，促使小艾从 更宏观的角度去思考整个产品的质量问题。

任何一个项目都有一定的资源约束，软件本身的复杂性与质量保证的特性也使得没有 一个完美的可量化指标能够证明经过一系列测试活动之后可以100%杜绝缺陷的存在，即 使投入10倍于现有人力，也并不一定能推导出软件质量会有10倍的改善，当然过少的资 源原则上也会导致更多的缺陷被遗漏。因此从某种意义上来说软件质量没有最好，只有更 好，测试架构师需要能够在一个复杂的资源约束条件下，制定一个周密的质量保证计划， 尽可能地避免将缺陷遗留给客户。

在一个产品被发布之后，测试架构师可以与技术支持部门保持沟通，与技术支持人员 一起分析客户那里产生的质量问题的根源，从而制定质量的持续改进行动计划，持续不断 地推动质量的进一步提高。

一年以后，小艾所在的产品研发部门逐渐地发展壮大了，在小艾的指导下，原来的产 品测试团队的技能有了很大提高。后来研发部门的产品线又进行了扩充，增加了几个其他 产品的研发任务，小艾被任命为产品线的首席测试架构师，小艾开始了他的首席架构师之 旅……



**博文视点** **·** **IT出版旗舰品牌**

技术凝聚实力 ·专业创新出版

**试读热评——**

●一直从事软件测试的执行工作，但是从来没有想过系统地学习，以前的知识都是从软件工程相关的书上得来的， 现在有这样一本书系统地阐述软件测试，对我很有帮助。

●这本书详细介绍了各种测试类型和相关的流程，像安装测试、移植测试、功能测试、性能测试、多国语言测试等，使 我对企业级软件测试的整体框架有了更多了解，为我现在从事的软件测试工作提供了很多参考和帮助。

●介绍软件测试的书籍很多，然而大多都侧重于理论，这本书中各位专家总结了实际项目中的经验，从理论联系实 际的角度来讲解，对从事软件测试工作的同仁们有很大的启发。

●从事软件测试工作以来，一直希望能得到系统化的理论指导而不是只了解自己所做的这一小块区域。这本书涵 盖的内容不仅包含了我所了解到的所有的测试类型，甚至对我之前一无所知的类型也有所涉及，这在我意料之 外，同时又深感受益良多。

●我是一名有一年工作经验的软件测试工程师，一年来一直在进行黑盒测试，之前对于测试只是严格遵循已有的测 试案例。拿到这本书后我才了解到测试原来是这样一个庞大的体系，我在这本书里获得了很多关于测试的理论知 识，相信对于我以后的测试工作会具有指导性作用。

●从事软件测试工作已经3年多了，一直忙着各种项目而没有时间停下来好好梳理和总结。看到这本书，一下子就 被它新颖的构思和精辟的理论经验总结吸引了，很有收获，也很感动，就像在帮自己梳理自己的昨天和今天，也仿 佛在小艾身上看到了自己美好的明天，憧憬着涅槃，去华山论剑!推荐给每一位从事测试工作，并且希望能在这 个领域有所建树的同行们。

●我是一名开发工程师，一直对测试团队如何保证软件质量感到好奇。本书很好地帮助我了解测试团队的角色分 工与工作方式，我现在知道如何找到对的测试人员进行沟通来提高开发质量和解决问题了。

●目前国内很多软件产品或者项目的测试部分还处在起步或者初级阶段，亟需一些指导性的书籍。这本书很系统全 面，可以为测试团队建立完善的测试架构和流程提供详尽的参考。

●作为一名即将毕业的大学生，我打算选择软件测试工作作为我的第一份职业。通过这本书，我对软件测试工作有 了一个系统的了解，同时对未来的职业发展道路有了一个比较清晰的思路。

●IBM 专家写的测试秘籍读后受益匪浅，助我大大提升了测试功底，对测试流程有了更深的了解。是一本不得不读

的好书!



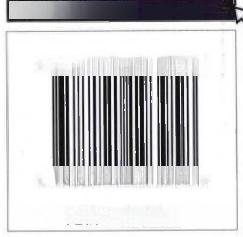
***新浪微博*** **@博文视点Broadview**

***weibo.com***

**策划编辑：刘** **皎**

**责任编辑：高洪霞**

**封** **面** **设** **计** **：** **李** **玲**



**上架建议：软件测试**

ISEN978-7-121-19395-8

917871211193958l>

**定价：58.00元**