软件测试基础

Foundations of Software Testing

【美】Rex Black 著
丹丹 王华 译

FOUNDATIONS OF SOFTWARE TESTING

人民邮电出版社
POSTS & TELECOM PRESS

软件测试基础

FOUNDATIONS OF SOFTWARE TESTING

【美】Rex Black 著
郑丹丹 王华 译

人民邮电出版社
北京

图书在版编目（CIP）数据

软件测试基础 /（美）布莱克（Black, R.）著 ；郑丹丹，王华译. -- 北京 ：人民邮电出版社，2013.5
ISBN 978-7-115-30791-0

Ⅰ. ①软… Ⅱ. ①布… ②郑… ③王… Ⅲ. ①软件－测试 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2013)第012340号

内 容 提 要

本书从软件测试原理、贯穿生命周期的测试、静态技术、测试设计技术、测试管理、测试的工具支持等几个方面介绍了软件和系统测试的基本技术、工具和概念。

本书的主要读者对象为软件测试领域技术人员、ISTQB 初级考试备考人员/培训班学员、软件工程/测试学习者，以及本科院校软件工程相关专业的师生。

软件测试基础

◆ 著 ［美］Rex Black
译 郑丹丹 王 华
责任编辑 杨 凌

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：787×1092 1/16
印张：17
字数：353 千字 2013 年 5 月第 1 版
印数：1－4 000 册 2013 年 5 月北京第 1 次印刷

著作权合同登记号 图字：01-2012-6398 号

ISBN 978-7-115-30791-0

定价：49.00 元

读者服务热线：(010)67132692 印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第 0021 号

译者序

软件测试是软件产品开发和部署中探究得最少的领域之一。软件测试几乎被视为软件部署前一项次要的活动，甚至只是走个形式而已。只要改变这种态度，并提供优秀的软件测试基础学习课程或教材，就能显著降低发布新软件时经常发生的问题，同时降低这些问题可能带来的风险。

对于想要成为职业软件测试人员的人来说，ISTQB（国际软件测试认证委员会）的课程和认证体系无疑为他们规划了一条清晰的职业学习路线。这套课程分为 3 个级别：基础级、高级和专家级。其中，高级分为测试管理和测试分析两个模块，专家级则分为测试过程改进、测试管理、测试自动化（开发中）和安全性测试（开发中）4 个模块。每个级别和每个模块都有对应的资格认证考试。

本书的作者曾经是 ISTQB 和 ASTQB（美国软件测试认证委员会）的主席，而这本书的编写正是针对 ISTQB 的基础级考试大纲，覆盖了基础级课程和考试的内容。除了对知识主体的讲述外，书的每一章中都提供了对应的练习，让读者能从真实项目的角度来应用所讲的概念、技术，考试样题和模拟题则按照大纲对每个知识点的掌握程度的要求来设计，让答题者能对自己的水平进行评价。因此，不管是从事软件相关工作，希望了解基础测试概念和技术的人，还是准备参加 ISTQB 考试的人，都可以选择此书。

很荣幸有机会参与这本书的翻译。在此要感谢（广州）赛宝认证中心的文燕、熊文杰、李雪几位同事，也感谢王春雨先生和高维女士，感谢他们在本书的整理和校译过程中提供了帮助。由于译者专业知识有限，虽然本书已经经过详细的评审，仍不免有翻译不当或不准确之处，敬请读者批评指正。

译者

2012 年 12 月于广州

目　录

第1章　测试基础 …… 1

1.1　为什么需要测试 …… 1
1.2　什么是测试 …… 5
1.2.1　练习 …… 9
1.3　测试的基本原则 …… 11
1.3.1　练习 …… 18
1.4　基本测试过程 …… 19
1.4.1　练习 …… 23
1.5　测试的心理学 …… 23
1.5.1　练习 …… 28
1.6　职业道德规范 …… 28
考试样题与模拟考试习题 …… 30
1.1　为什么需要测试（K2） …… 30
1.2　什么是测试（K2） …… 31
1.3　测试的基本原则（K2） …… 32
1.4　基本测试过程（K1） …… 32
1.5　测试的心理学（K2） …… 32
第1章　跨章节问题 …… 33
模拟考试1 …… 33
模拟考试2 …… 34

第2章　贯穿软件生命周期的测试 …… 36

2.1　软件开发模型 …… 36
2.1.1　练习 …… 42
2.2　测试级别或阶段 …… 43
2.2.1　练习 …… 52
2.3　测试类型或目标 …… 53
2.3.1　练习 …… 60
2.4　维护测试 …… 61
2.4.1　练习 …… 64
考试样题与模拟考试习题 …… 65
2.1　软件开发模型（K2） …… 65
2.2　测试级别或阶段（K2） …… 66
2.3　测试类型或目标（K2） …… 67
2.4　维护测试（K2） …… 68
第2章　跨章节问题 …… 69
模拟考试1 …… 69
模拟考试2 …… 70

第3章　静态技术 …… 72

3.1　静态技术和测试过程 …… 72
3.1.1　练习 …… 74
3.2　评审过程 …… 74
3.2.1　练习 …… 81
3.3　静态分析的工具支持 …… 83
3.3.1　练习 …… 85
考试样题与模拟考试习题 …… 86
3.1　静态技术和测试过程（K2） …… 86
3.2　评审过程（K2） …… 87
3.3　静态分析的工具支持（K2） …… 88
第3章　跨章节问题 …… 89
模拟考试1 …… 89
模拟考试2 …… 90

第4章　测试设计技术 …… 91

4.1　测试开发过程 …… 91
4.1.1　练习 …… 102
4.2　测试设计技术的种类 …… 109
4.2.1　练习 …… 111
4.3　基于规格说明的技术或黑盒技术 …… 112
4.3.1　练习 …… 128
4.4　基于结构的技术或白盒技术 …… 137
4.4.1　练习 …… 143
4.5　基于经验的技术 …… 144
4.5.1　练习 …… 149
4.6　选择测试技术 …… 150
4.6.1　练习 …… 152
考试样题与模拟考试习题 …… 152
4.1　测试开发过程（K3） …… 152
4.2　测试设计技术的种类（K2） …… 154
4.3　基于规格说明的技术或

黑盒技术（K3） …… 155
4.4 基于结构的技术或白盒技术（K3） …… 157
4.5 基于经验的技术（K2） …… 159
4.6 选择测试技术（K2） …… 159
第4章 跨章节问题 …… 160
模拟考试1 …… 160
模拟考试2 …… 163

第5章 测试管理 …… 167

5.1 测试组织 …… 167
5.1.1 练习 …… 173
5.2 测试策划和估算 …… 175
5.2.1 练习 …… 182
5.3 测试进度监控 …… 184
5.3.1 练习 …… 191
5.4 配置管理 …… 197
5.4.1 练习 …… 200
5.5 风险与测试 …… 201
5.5.1 练习 …… 203
5.6 缺陷或事件管理 …… 204
5.6.1 练习 …… 212
考试样题与模拟考试习题 …… 213
5.1 测试的组织（K2） …… 213
5.2 测试策划和估算（K2） …… 214
5.3 测试进度监控（K2） …… 216
5.4 配置管理（K2） …… 218
5.5 风险与测试（K2） …… 218
5.6 缺陷或事件管理（K3） …… 219
第5章 跨章节问题 …… 220
模拟考试1 …… 220
模拟考试2 …… 222

第6章 支持测试的工具 …… 225

6.1 测试工具的类型 …… 225
6.1.1 练习 …… 230
6.2 有效使用工具：潜在的收益和风险 …… 231
6.2.1 练习 …… 237
6.3 组织中工具的引入 …… 238
6.3.1 练习 …… 242
考试样题与模拟考试习题 …… 243
6.1 测试工具的类型（K2） …… 243
6.2 有效使用工具：潜在的收益和风险（K2） …… 244
6.3 组织中工具的引入（K1） …… 244
第6章 跨章节问题 …… 245
模拟考试1 …… 245
模拟考试2 …… 246

附录A Omninet：网络无处不在 …… 248

市场需求文档 …… 248

1. 范围 …… 248
1.1 术语的首字母缩写和缩写词 …… 248
1.2 适用文档 …… 249
2. 要求的发布日期 …… 249
3. 需求描述 …… 249
3.1 一般技术要求 …… 249
3.2 管理 …… 250

附录B Omninet：网络无处不在 …… 253

系统需求文档 …… 253

功能系统需求 …… 253
可靠性系统需求 …… 255
易用性系统需求 …… 256
效率系统需求 …… 257
可维护性系统需求 …… 257
可移植性系统需求 …… 258
设计模型 …… 259
Omninet系统架构 …… 259
付款数据处理决策表 …… 259
信息服务亭模块流程图 …… 260
信息服务亭状态转换图 …… 260
信息服务亭状态转换表 …… 260
信息服务亭操作系统/浏览器/连接速度配置正交表 …… 260

附录C 考试样题答案 …… 262

第1章

测试基础

第1章，测试基础，包含以下6节。

1. 为什么需要测试
2. 什么是测试
3. 测试的基本原则
4. 基本测试过程
5. 测试的心理学
6. 职业道德规范

1.1 为什么需要测试

学习目标

LO-1.1.1 通过具体的例子来描述软件中的缺陷会以什么样的方式损害个人、损害环境或者损害公司的利益（K2）。

LO-1.1.2 区分引起缺陷的根本原因及其影响（K2）。

LO-1.1.3 通过举例的方式说明为什么需要测试（K2）。

LO-1.1.4 描述为什么测试是质量保证（quality assurance）的一部分，通过举例说明测试是如何提高软件质量的（K2）。

LO-1.1.5 举例说明术语错误（error）、缺陷（defect）、故障、失效以及对应的术语错误（mistake）和缺陷（bug）的定义并比较它们的区别（K2）。

本节将说明以下关键概念：

- 缺陷如何造成伤害或损失；
- 缺陷及其影响；
- 测试的必要性；
- 测试在质量保证中的角色。

ISTQB 术语

缺陷（bug）：参见 defect。

缺陷（defect）：可能会导致软件组件或系统无法执行其被要求执行的功能而产生瑕疵，例如错误的语句或数据定义。如果运行中遇到缺陷，可能会导致组件或系统的失效。

错误（error）：人为地产生不正确结果的行为。

失效（failure）：组件或系统与预期的交付、服务或结果存在的偏差。

故障（fault）：参见 defect。

错误（mistake）：参见 error。

质量（quality）：组件、系统或过程满足指定需求或用户/客户需要及期望的程度。

风险（risk）：将要造成负面结果的因素，通常用影响和可能性表示。

软件无处不在，一个人在其生活的许多方面都要接触软件。有时软件明显存在，有时则不然。当我们身处银行或在使用 ATM 时，我们能看到运行中的软件系统；但当我们开着现代化的轿车时，则不太会想到，与 20 世纪 80 年代之前建造的航天器相比，现在大多数轿车上安装的软件数量更多，运算能力更强。

虽然我们会轻易地忘记软件正在运行，但却能轻松地想起我们碰到的那些软件不起作用的经历。如果软件含有缺陷，很多不好的事情就会发生在各种人身上。

先从开发或购买软件的公司或组织说起。对于一个公司，无论是作为软件的生产者或是因购买软件而使其客户蒙受损失的采购商，其名誉都会遭受损害。

由于生产中出现的失效，公司可能要承担高昂的或不可预测的维护成本。

由于问题在开发后期才发现，因此可能出现预期之外的发布周期延迟。

无论是在发布前还是发布后出现缺陷，当出现大量缺陷时，可能造成利益相关者缺乏信心，无论他们是公司的员工还是客户。

当然，某些情况下软件缺陷过多还可能导致法律纠纷。

环境方面，软件被越来越多地嵌入到控制工业过程、工厂、发电厂还有汽车的系统中。如果这些应用程序不正常运作，就可能造成过度污染和资源浪费。比如，这样的软件可能使系统超过需要、额外地多烧汽油或多用电力。

缺陷也会给个人、社会和国家带来威胁。如果因为软件失效而导致其雇主的经济放缓，那么个人可能会失业。

某些情况下，人们可能因为问题软件的缺陷而丧命。最近在得克萨斯州发生了这样一个事故，一个人在电梯门关闭时想要挤进电梯，却被电梯门卡住了，而且具有讽刺意味的是事故发生在一家医院里。这个人只有半个身子进了电梯，因为电梯的控制软件没有在门半开时完全阻止电梯的运行，结果导致悲剧发生。

人们可能会丧失民事权利，比如当电脑化的投票机失效时。

很多人因为银行和大学所用的软件被缺陷影响而蒙受隐私受侵犯、身份信息被盗等损失。因为软件已在战争环境中得到普遍使用，可能因为缺陷造成使命甚至战争的失败。战士个人也常常携带着或依赖于有软件运行的系统。

缺陷来自哪里，又会造成什么后果呢？

缺陷在软件系统中出现不是因为小虫仙们在数据中心飞来飞去把缺陷放入电脑和代码里（英文中 bug 也有小虫的意思）。坦白地说，缺陷存在于系统里是因为有人把它们放进去了。有人出了一个错，结果往系统中引入了一个缺陷（bug 或 defect）。

缺陷有可能在生命周期中的任何时候被引入。可能引入到需求或设计说明书中；可能引入到代码中，不管是业务逻辑或是用户界面；可能引入到文档中，不论是电子文档还是纸质文档。

系统中一旦引入了缺陷，可能会造成失效，也可能不会。若系统出现失效，必须是执行或运行了有缺陷的那部分系统，这时缺陷可能引起系统的不当行为。假设有正确的前提条件，系统可能无法完成它在这些条件下该做的事情。如果客户、用户或其他利益相关者当场或后来看到这个失效的情况，则可能会对系统质量不满意。

注意：

这里我们多次用了“可能”这个条件词。在很多情况下，因为没有看到缺陷造成的行为，即使系统里存在缺陷也没有发现。之所以没看到，仅仅是因为这种行为要在执行包含缺陷的代码前有特定顺序的动作，或要在特定数据提供给这部分代码，或同时满足这两个条件时才发生。

那么，所有这些缺陷和失效的背后是什么呢？

有多种原因导致产生缺陷。

程序员、业务分析员、系统分析员和其他参与者，包括测试人员，都可能犯错误。他们犯错，有些错误会引入缺陷。人们在承受时间压力的条件下可能会犯更多的错误。目前时间压力在软件和系统开发项目中普遍存在。如果你要解决一个复杂的问题，或处理历经数年变得复杂了的代码，或处理复杂的架构时，就更容易犯错。

很多情况下，系统的很多不同部分，或综合系统（即由多个系统组成的大系统）需要一起运作。

综合系统和实现系统的技术经常都在变化，但又必须紧密配合。

想想这些日子人们对面向服务的架构都做了些什么。不断有人尝试在已有的主机代码的基础上实现原本用其他代码已经做到的服务，这些通常是在主机上运行的旧 COBOL 代码，有时候这些代码已经在数据中心放了二三十年甚至更久。

其次，我们有很多系统以端到端的形式交互，在事务能正确完成之前必须由数个系统处理。单个操作可能是简单的，但整个一系列的操作则可能变得复杂，而这种复杂可能会

带来对一个人或多个人的不利期望，这会导致不同操作的接口或排序产生缺陷。

很多失效是由如前面所说的缺陷引起的。不过，环境条件也可能引起失效。例如，执行代码的系统过热可能会造成内存、存储或 CPU 的间歇性失效。软件的误用，不管是故意的还是无意的，也可能引起失效。

有人可能会争论说系统应该能处理所有可预见的情况和使用。不过，考虑到系统可能面对几乎无限的条件和输入，在某些情况下，误用引起的失效是无法完全避免的。

既然已经考虑到缺陷和失效，我们就可以理解系统面临的一些风险了。

到目前为止我们所讨论的风险都是关于质量的。我这里所说的“质量”宽泛地指“系统是否准备好发布给客户或可用于过程中的下一个步骤了”，下一部分我用到“质量”这个重要的词时会更精确一些。

对于软件项目来说，还有一些其他风险和制约是必须考虑的。例如，无法实现部分所需功能的风险，发布延迟的风险，受到预算和资源制约的风险。测试对于这些方面的风险不太起作用，反而会受到这些风险的影响。我们后面也会更多地谈到这方面的问题。

然而，如果把测试看成管理质量风险的一种方法，它确实会以各种方式起到这样的作用：一种是通过提供信息，在质量风险方面指导项目；另一种是让测试关注于最重要的质量风险；还有一种是找到重要的问题让人们可以去修改。

因为法律法规、符合性问题或合同的原因，我们可以或必须进行测试。例如，萨班斯·奥克斯利法案适用于美国的很多金融机构。对于某些特定类型的网站，可能会要求让残疾人也可以使用。

好了，让我们回到“质量是什么”这个问题。对“质量”有两个常见的、正式的、但是又相互冲突的定义：一个是“使用的适合性”，另一个是“对要求的符合性”。

我之所以说这两个定义相互冲突是因为，“使用的适合性”表明软件是否有质量问题应由使用、采用、采购或要求这个软件的人来确定。

而“对要求的符合性”，相反地，只要求系统按照某些文档所说的那样工作。那么文档中对正确性的定义本身是不是正确，又成为另外一个问题了。

测试是如何与质量相关联的？前面我说到了测试是管理质量风险的一种方法，让我们再深入一点。

假设我们执行完一组测试后只找到了很少的缺陷。在这种情况下，我们对系统的信心应该会更高些。

假设我们执行了一组测试，所有的测试都通过了。在这种情况下，只要测试的设计是合理的，仍然存在的质量风险水平就低了。

假设我们执行了一组测试，而测试没通过，那么这些测试给了我们提高系统质量的机会。最后，如果覆盖度是足够的，这组测试总体为我们良好地评估了系统的质量。

当然，这又让我们想到一个问题——“我们真的正在测试正确的东西吗？”想想对你

的系统而言重要的而且能产生一个适合使用的系统的质量特性，比如性能、易用性、可靠性和准确性这些质量特性。你已经想通这些特性到底是什么了吗？对于这些，目前有足够的测试吗？你怎么知道呢？

在本书的后面部分，我们会看到能确定这些问题答案的方法。

搞清楚测试、质量保证和质量改进三者之间的关系和区别是很重要的。测试组常常被错误地称为“质量保证组”或“QA 组”。尽管测试可能是也应该是一个更大的质量保证策略的组成部分，但测试在保证质量方面所起的作用也就仅仅相当于每天早晨量体重来确认体重减轻一样。只有在整个软件生命周期中、在整个组织中充分关注质量，我们才能保证质量。

仅有质量保证本身也是不够的。从长远来说，我们想要提高质量，因此我们需要寻找有待改进的领域并弄明白如何适当地改进，而测试能为质量改进提供重要的输入。比如，当我们找到很多缺陷时，如果我们对缺陷恰当地分类，程序员对它们恰当地分类，就能采取一些步骤在未来降低这类缺陷的数目。

1.2　什么是测试

学习目标

LO-1.2.1 认识测试的总体目标（K1）。

LO-1.2.2 举例描述软件生命周期中不同阶段的测试目标（K2）。

LO-1.2.3 区分测试与调试（K2）。

ISTQB 术语

调试（debugging）：发现、分析和去除软件失效根源的过程。

需求（requirement）：为让用户解决问题或达到目的，系统必须满足的条件或者能力。通过系统或者系统的组件的运行以满足合同、标准、规格或其他指定的正式文档定义的要求。

评审（review）：对产品或产品状态进行的评估，以确定与计划的结果之间存在的误差，并提供改进建议。例如，管理评审（management review）、非正式评审（informal review）、技术评审（technical review）、审查（inspection）和走查（walkthrough）。

测试用例（test case）：为特定目标或测试条件（例如，执行特定的程序路径，或是验证与特定需求的一致性）而制定的一组输入值、执行入口条件、预期结果和执行出口条件。

测试（testing）：包括了所有生命周期活动的过程，有静态的也有动态的。涉及计划、准备和对软件及其相关工作产品的评估，以发现缺陷来判定软件或软件的工作产品是否满足特定需求，证明它们是否符合目标。

测试目标（test objective）：设计和执行测试的原因或目的。

本书将讨论以下关键概念：

- 测试的共同目标是什么？
- 测试在软件开发、软件维护和软件运行活动中的目的是什么？
- 在找到缺陷、建立信心、提供信息和预防缺陷方面的目的是什么？

多数情况下，测试有以下几个常见的一般测试目标。

进行测试可能是为了找到缺陷，然后为程序员提供修复这些缺陷所需要的信息。程序员一般不会修复所有的缺陷，但我们的信息应该至少帮助他们修复最重要的缺陷。

进行测试可能是为了使人们对系统质量水平有信心。公司采购应用软件的时候，在数据中心部署应用软件之前，经常会做验收测试来获取信心。

进行测试可能是为了预防缺陷，这对你可能有些陌生。不过，如果测试人员参与评审并且测试设计与系统实施并行完成的话，测试会预防缺陷。这些早期的测试活动在测试和开发活动之间建立良性的反馈循环，在生命周期早期将缺陷清除。

进行测试可能是为了提供信息，以深入了解对于被测系统的质量而言，什么是真正重要的。我们准备好发货了吗？我们什么时候能准备好发货？还有哪些风险领域是需要解决的？已知的问题中哪些是最可怕的？

一旦我们开始有了那样的信息，即能让我们深入了解质量水平，我们就可能会有下一个目标：帮助管理人员在大的项目目标的背景下理解质量。只有当管理人员理解我们的发现、理解这些发现对项目意味着什么的时候，才能说我们作为测试人员真的帮助项目管理了质量风险。

当然，并不是所有的目标都在这里列出来了。你可能还会想到你自己项目里的其他目标——这样也很好。

重要的往往不是选择了哪些测试目标，而是要让那些目标保持一致。这个地方经常出错。作为专业测试人员的你可能没有让自己的计划和行动与管理人员定的测试目标一致，你可能发现自己被迫使自己的计划和行动与管理人员所暗示但没有明说的测试目标一致，这导致当你猜测错误时出现问题。更糟的是，可能所有管理人员并没有一组单一的、一致的测试目标，而是不同管理人员有不同的、可能还相互冲突的测试目标。

这些一致性问题很常见，而且对于测试组而言是非常危险的问题。

测试目标可能随着我们所处的阶段或测试级别而变化，以找到缺陷这个测试目标为例进行说明。

在单元或组件测试中，你会想在所测系统的单个部分完全集成到系统中之前，找到单个部分里的缺陷。

对于集成或串测试，你会想在所测系统的这些部分组合到一起时，找到每对或每组组件的关系和接口中的缺陷。

如果是系统测试，你会想找到所测系统作为一个整体，在总体或特定行为、功能和响应上存在的缺陷。

对于验收或试运行测试，我们一般不想找到任何缺陷，因为如果找到缺陷，就会成为问题。我们想证明产品已经准备好可以部署或发布了，或者评估质量并提供部署或发布的风险的信息。

对于维护测试，我们可能专门找一种特定的缺陷，即那些在变更的开发中引入的缺陷。

最后，在运行测试里，我们又想找某些种类的缺陷，尤其是跟系统非功能特性相关的缺陷，通常是在运行环境中，比如可靠性或可用性。

由此可见，一个一般的目标必须基于应用该目标的背景调整为特定的目标。

当然，我们都想成为有效和高效的软件测试人员，但我们常常草率地使用“有效”和“高效”这些词。看看我们能不能更严谨一点。

“有效”是指形成确定的、决定性的或期望的结果，令人印象深刻。因此，要成为有效的测试人员，我们必须选择适宜的目标和期望的结果。要让所有人都认同我们是有效的，就要取得所有人对目标的认同。

“高效”是指高产地形成期望的影响，尤其是没有浪费的高产。因此，若想要成为高效的测试人员，我们必须适宜地分配资源。资源包括时间（既指工作量也指工期），也包括金钱和要花钱的那些东西。

重要的是记住一点，在优化一组活动的有效性或效率的时候，可能会对更大范围的工作的有效性和效率带来损害。比如，假设我们决定等到需求规格说明书完成之后再开始设计测试。这样可能会提高我们自己的效率，因为在说明书变更时我们不用对测试返工。不过，我们会失去在需求规格说明书编写过程中找出里面的缺陷的机会，我们会失去预防缺陷的机会。

这是局部优化的经典例子。另一个经典例子是，当程序员说“我不需要做单元测试，因为后面会有测试人员找出缺陷”，这是真的，但测试人员不一定能找到本来单元测试能找到的所有缺陷，而且测试人员找缺陷的成本更高。

所以，应该从整个开发或维护过程的大背景去考虑有效性和效率，而不仅仅是从你自己的测试过程来考虑。

把测试和调试区分开来是非常重要的。

测试，正如我们已经看到的，会有多个目标，其中一个目标就是找出缺陷。当然，一般来说，测试中实际发现的不是缺陷。我们发现已发生的失效，通常是由缺陷引起的。

调试是当我们认为失效是由缺陷引起时所触发的特定活动。调试涉及识别缺陷的根本原因，然后移除根本原因，修复代码。最后，恰当的调试还会包括一定程度的单元测试来检查修复的正确性。

当缺陷修复后的内容再回到发现该缺陷的测试人员手里时，这个测试人员进行一次确认测试来确保之前观察到的失效已经解决了。确认测试（confirmation test）在 ISTQB 术语

表中也叫做再测试（re-test）。

注意，在这个过程中，不同的人有不同的职责：测试人员负责测试，程序员负责调试。

图 1.1 表示的是测试找到失效、程序员调试失效、测试人员确认测试的过程。

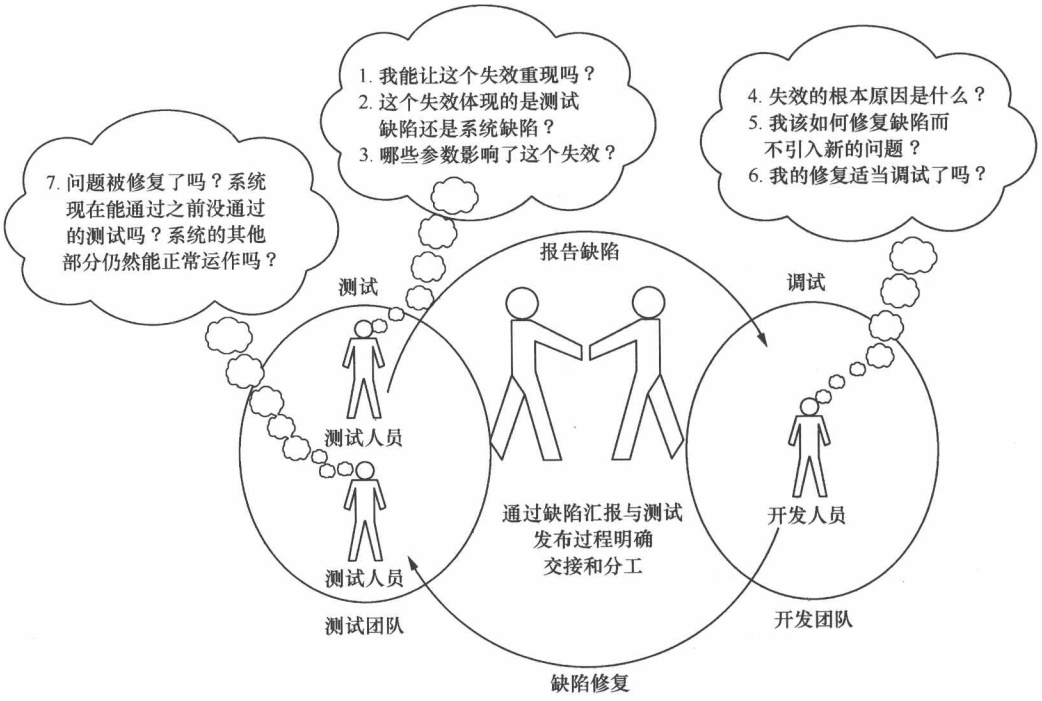

图 1.1　发现－调试－确认

具体地说，测试人员执行过程中 1～3 的步骤，即图 1.1 的上部分。测试人员查看问题不是间歇性的或重复性的。测试人员检查以区分失效是由系统缺陷引起的，还是由不好的测试用例、不好的测试数据等引起的。测试人员还查看不同的配置或数据值是否会改变失效的行为，这个信息会帮助程序员进行调试，所以测试人员把它写进缺陷报告中。

缺陷报告随后交接给开发组。如图 1.1 中部所示，这时的交接必须予以良好定义，否则就会有一种我称为“打乒乓球”的风险。在程序员能采取行动前，缺陷报告可能会在两个组之间来回传递，相互说明。

一旦程序员接受了缺陷报告，即认为缺陷需要去修复，需要去识别根本原因。他尽量在修复的时候不产生更多麻烦，然后对修复的内容进行单元测试。

通过测试发布过程，这里有另一次交接，我们后面会讨论这个。不过，与缺陷报告的交接一样，这时的交接也必须定义好。

最后，测试人员对修复了缺陷的系统进行确认测试和回归测试，报告的结果最好在这里就关闭了。不过，如果缺陷修复没能修改好整个失效，导致报告再次打开，也可能发生我称之为“循环的循环”的情况。

很明显，“打乒乓球”和“循环的循环”都对测试团队、开发团队和整个项目团队的有效性和效率不利。这个找缺陷－调试－确认的过程很重要，在一个项目中可能要数十次甚至数百次地经历这个过程，必须把它做对。

前面我提到过测试组的一个常见问题：不一致的期望。当人们认为测试仅仅是测试系统，通常只需要在项目尾声进行时，就出现了一个不一致的期望。

还有很多其他重要的测试活动，例如：

- 策划和控制；
- 选择测试条件；
- 设计测试用例；
- 检查测试结果；
- 评价出口准则；
- 报告测试结果；
- 结束测试任务。

我们后面会谈到这些。

1.2.1　练习

练习 1

某程序接收的输入为 3 个整数，代表一个三角形 3 条边的长度。程序应该给出 3 种输出中的一种：如果三条边都不相等，应该输出“不等边三角形”；如果有两条边相等，应该输出“等腰三角形”；如果三条边都相等，应该输出“等边三角形”。

你要写一组有效和高效的测试用例。“有效”是指覆盖重要的测试条件并找到重要的缺陷，“高效”是指用尽量少的测试。

由你自己决定测试用例的详细程度和怎样写测试用例的文档。一个测试用例通常包括测试人员行为、数据和所期望的结果，不过这里的行为一般就是指输入你设计的测试数据。

这个练习很有名，是在第一本关于软件测试的书里找到的。这本书是 Glenford Myers 所著的 The Art of Software Testing（《软件测试的艺术》）。在这本书中，Myers 给出了这个看起来很不起眼的例子，然后展示了多种可能的解答。

看着你的答案问你自己几个问题。

首先，你是不是既包括了有效的输入也包括了无效的输入？大多数从事测试工作一段时间的人都擅长用无效输入来测试，有时候会过分强调这部分测试而忽略了重要的有效条件。

其次，你要先执行哪些测试？有些人喜欢从各种无效的输入开始，强加很多的错误条件。这种方式对于稳定的应用软件是可以的，但对于不够稳定的应用软件，通常最好还是

先执行一些基本的有效用例。

练习 1 答案

来看看我的答案。我结合了等价划分和边界值分析来设计这个方案。

测试人员动作和数据	预期结果
错误输入类	
3 个测试，每个测试分别用一个非数字输入 *a*、*b* 或 *c*	错误提示
3 个测试，每个测试分别用一个实数输入 *a*、*b* 或 *c*	错误提示
不等边三角形类	
2、3、4	不等边
maxint–2、maxint–1、maxint（maxint=最大合法整数）	不等边
1、2、3	错误提示
5、3、2	错误提示
1、2、4	错误提示
–2、3、4	错误提示
0、1、2	错误提示
maxint–1、maxint、maxint+1	错误提示
等腰三角形类	
2、2、1	等腰
9、5、5	等腰
maxint、maxint–1、maxint	等腰
1、2、1	错误提示
2、5、2	错误提示
3、1、1	错误提示
2、–2、1	错误提示
2、2、0	错误提示
maxint、maxint+1、maxint	错误提示
等边三角形类	
1、1、1	等边
maxint、maxint、maxint	等边
maxint+1、maxint+1、maxint+1	错误提示
1、1、–1	错误提示
0、0、0	错误提示
用户界面测试	
a、*b* 和 *c* 分别缓冲区溢出一次	错误提示
尝试用一个空格作为 *a*、*b* 或 *c* 输入的首个字符	错误提示或忽略空格
尝试用一个空格结束 *a*、*b* 或 *c* 的输入	错误提示或忽略空格
a、*b* 或 *c* 留空（每次一个）	错误提示

我们这里只关注了功能性和用户界面的测试，后面的练习中会谈到可能还要考虑的其他系统质量方面的风险。

注意，我的方案只是很多可能的正确答案中的一种。Myers 的书中介绍了一些判断一组好的测试的通用规则，Myers 也提供了对这个练习的解答。因为他的书是在绿屏主机程序的时代写的，所以他不用覆盖某些我的方案包括了的用户界面测试。

这里我用了大概 40 个测试。但是如果对不同类不使用类似的错误条件的话，我可以把测试数降到 21 个。这样做需要两个简单的假设。

首先要注意的是，只有任意两条边的长度之和都比第三条边长时，才能形成一个三角形，代表三条边长度的 3 个整数也是如此。因为可能的逻辑错误我可以减去 6 个测试，3 个第三边的长度比另外两边之和要长的，3 个第三边的长度与另外两边长度之和相等的。

其次，要注意一个聪明的程序员会写一个功能，确认输入给程序的是一个整数，然后在每个输入字段重用这个功能。所以，我们只要测试一个字符、一个实数、一个数字 0、一个负整数、一个最大值+1 的数、一个缓冲区溢出、一个前面空格、一个后面空格、一个空白，总共 9 种输入错误。

除了本书外，可能还需要参考其他书籍来得出一组完整的测试。我也是在重读了 Myers 对练习的讲解后才确定了我的方案的。

这指出了本书的一个重要的主题思想——不参考资料做测试设计和实施常常导致测试有缺漏，尤其是在时间很紧、进度压力大的情况下做测试设计和实施，这种缺漏的可能性就更大，即使是有经验的测试人员也一样。这就更需要从一开始就仔细地分析、设计和实施测试，而不是仅仅依赖于像探索性测试或随机测试这种响应型的测试方法。大概每个测试项目都会做响应型测试，但不是作为主要的测试方法。

1.3　测试的基本原则

学习目标

LO-1.3.1 说明测试的 7 项原则（K2）。

ISTQB 术语

穷尽测试（exhaustive testing）：测试套件包含了软件输入值和前置条件的所有可能的组合方式的测试方法。

本节将解释以下测试原则：

- 测试揭示缺陷的存在；
- 穷尽测试的不可能性；
- 早期测试的好处；