开发者测试

王兴亚 王智钢 赵源 陈振宇 编著

开发者讨厌给自己程序写测试，更讨厌不给自己程序写测试的人。

—— 陈振宇

机械工业出版社
China Machine Press

软件工程技术丛书

全国大学生软件测试大赛指导用书

开发者测试

王兴亚 王智钢 赵源 陈振宇 编著

DEVELOPER TESTING

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

开发者测试 / 王兴亚等编著 . —北京：机械工业出版社，2019.1
（软件工程技术丛书）

ISBN 978-7-111-61681-8

I. 开… II. 王… III. 软件 – 测试 IV. TP311.55

中国版本图书馆 CIP 数据核字（2019）第 000815 号

全书共分为 8 章及一个附录，主要内容包括开发者测试概述、程序静态分析、白盒测试、程序插桩与变异测试、单元测试、集成测试、JUnit 基础、JUnit 深入应用、慕测科技——开发者测试平台等与开发者测试相关的知识、技术和平台。书中涵盖了开发者测试的四个重要方面：1）开发者测试出现的背景与意义；2）开发者所应掌握的基本和高级程序分析方法（如程序流程分析、符号执行）以及软件测试技术（如白盒测试、单元测试、集成测试、变异测试、程序插桩）；3）开发者所应掌握的软件测试分析辅助工具（如 JUnit、JaCoCo、PITest）；4）用于开发者测试教学、竞赛的慕测平台。全书通过多个 Java 示例代码阐释了各个方法和技术，以便读者理解和学习。

本书得到江苏高校品牌专业建设工程项目 PPZY2015B140 的支持。本书适合高等院校相关专业的学生及教师阅读，也适合软件开发人员、测试人员及未来希望从事软件开发、测试的其他专业人员参考。

开发者测试

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：佘　洁　　责任校对：李秋荣
印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2019 年 2 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：12.75
书　　号：ISBN 978-7-111-61681-8　　定　　价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

前　言

当前，信息需求的持续增长和信息技术的快速发展加快了软件产品的研发速度，同时也大大增加了软件产品的测试压力。以互联网、移动应用等产品为例，众多软件公司普遍采用微小改进、快速迭代、反馈收集、及时响应等手段来提高软件的迭代速度，缩短软件产品的发布流程。显然，仅仅依赖测试人员已经难以满足市场和客户对产品质量的需求，这就要求开发人员也深入参与到软件测试过程中，与测试人员共同完成软件产品的质量保证工作。在本书中，我们定义由开发者承担的与代码相关的软件测试工作为开发者测试。

本书从开发者测试出现的背景与意义、开发者所应掌握的基本和高级程序分析方法以及软件测试技术、开发者所应掌握的软件测试分析辅助工具、用于开发者测试教学和竞赛的慕测平台等多个方面对开发者测试进行系统性介绍。相信通过本书的学习，读者可以对开发者参与测试的必要性、开发者测试所涵盖的内容有初步的认识和了解，同时能够结合本书的示例及平台锻炼自己的测试能力。

本书适用于两类不同的读者：1）在高等院校学习和工作的教师和学生，本书有助于他们理解和认识测试工作承担者的责任，并为他们学习和锻炼自身的测试能力提供方向和平台；2）软件产业的开发人员、测试人员和管理人员，本书有助于他们认识开发者在测试工作中的重要性和所应承担的工作内容，以及开发者所应具备的测试技能。

本书讲述的方法是通用的，可以用于测试任何类型的计算机软件。但是，为了使读者更好地理解和学习本书的开发者测试方法，本书提供了大量 Java 示例代码以及面向 Java 的程序分析、测试工具。这些示例和工具可以在任何支持 Java 的操作系统（如 Windows、Linux、Mac）、开发环境（如 Eclipse、IntelliJ、Sublime Text）中开发、测试和运行。

本书共包含8章及一个附录，除第7章与第8章外，其他章节的内容互不相关，因而读者可选择其中部分章节进行阅读。

第1章：开发者测试概述。本章在研究和分析开发者与软件测试关系的基础上，介绍了开发者测试的定义、背景与意义。同时，本章还从静态测试与动态测试、白盒测试与黑盒测试、不同测试工具间的对比中分析得到开发者测试所涉及的方法、技术与工具。此外，本章还讨论了开发者测试技术未来的趋势，并介绍了支持开发者测试教学与竞赛的慕测平台。

第2章：程序静态分析。本章对软件静态测试的基础——程序静态分析方法进行了介绍。通过代码评审、结构分析等方法可以有效地检测出程序中的逻辑错误，而程序流程分析（如控制流分析、数据流分析）则可以更细粒度地反映程序中语句间、变量间的关联。此外，本章还介绍了辅助程序正确性证明的静态/动态符号执行方法，便于读者了解更高级的程序分析方法。

第3章：白盒测试。白盒测试要求软件内部的逻辑结构透明可见，因此更适合由软件项目的开发者来承担。本章介绍了两类主要的白盒测试方法，包括以程序内部逻辑结构为基础的逻辑覆盖测试方法和以程序路径为基础的路径覆盖测试方法。与此同时，本章还比较了不同白盒测试方法的测试强度，并介绍了用于度量程序复杂度的环复杂度方法。

第4章：程序插桩与变异测试。本章介绍了用于获取程序运行时信息的程序插桩方法，以及用于度量测试用例集缺陷检测能力的变异测试方法。对于程序插桩方法，本章详细介绍了插桩位置、类型、数量的选择方法；对于变异测试方法，本章详细介绍了变异算子的设计与选择方法。同时，本章还介绍了工具JaCoCo和PITest，以便读者体验Java程序的运行时信息收集和变异测试过程。

第5章：单元测试。单元测试是对软件基本组成单元（如方法、函数、过程）的测试。在测试过程中要完成初始状态的创建、测试结果的验证、测试资源的释放等工作，这些工作适合开发者使用代码控制开展。本章在介绍单元测试框架的基础上，进一步阐述了单元测试的各项内容，使读者能针对不同的测试对象分

析、建立相应的测试模型。

第 6 章：集成测试。通过单元测试的软件模块并不能保证在整合后依然运行正确，因此需要做集成测试以进一步验证。本章介绍了集成测试过程、集成测试所面向的缺陷类型以及分析方法，并详细介绍了多种集成测试策略。同时，本章还讨论了不同集成测试策略的优缺点，并对它们各自的适用场景进行了分析，测试人员可据此选择合适的集成测试策略。

第 7 章：JUnit 基础。工欲善其事，必先利其器。JUnit 是开发者开展单元测试的一把利器。本章对 Java 单元测试框架的基本功能（如注解、测试类与测试方法、错误与异常处理、批量测试）进行了详细的介绍，使读者对 JUnit 的功能和适用范围有了详细的了解。本章还穿插了数个 JUnit 示例程序，帮助读者更快、更方便地学习 Java 单元测试。

第 8 章：JUnit 深入应用。在前一章介绍 JUnit 基本功能的基础上，本章对 JUnit 的高级功能进行了介绍，包括用于提高测试代码开发效率的匹配器功能，面向 Controller 和 Private 函数的测试功能、Stup 测试功能和 Mock 测试功能。同时，本章还介绍了 JUnit 与常用 Java 开发框架（如 Ant、Maven）的集成方法，读者可据此配置来构建更方便的 Java 单元测试环境。

附录：慕测科技——开发者测试平台。实践练习是提高开发者测试能力的有效方法。本附录介绍了支持开发者测试教学的慕测平台，并说明了面向教师的账号注册、班级管理、考试管理等功能。同时，还对由慕测平台提供技术支持的全国大学生软件测试大赛进行了介绍，该赛事为软件测试专业的宣传及开发者测试理念的普及做出了重要贡献。

目　录

前　言

第 1 章　开发者测试概述 …… 1

1.1　开发者与软件测试 …… 1

1.1.1　测试和调试 …… 1

1.1.2　开发者测试 …… 3

1.1.3　PIE 模型 …… 4

1.2　开发者测试方法与技术 …… 6

1.2.1　静态测试与动态测试 …… 6

1.2.2　黑盒测试与白盒测试 …… 8

1.2.3　失效重现 …… 9

1.3　开发者测试工具 …… 9

1.3.1　静态测试扫描工具 …… 9

1.3.2　测试覆盖分析工具 …… 12

1.4　开发者测试趋势 …… 14

1.4.1　软件开发和运营困境 …… 14

1.4.2　DevOps 介绍 …… 16

1.4.3　DevOps 中的开发者测试 …… 17

1.5　慕测开发者测试 …… 19

1.6　小结 …… 27

习题 1 …… 28

第 2 章　程序静态分析 …… 29

2.1　程序静态分析概述 …… 29

2.1.1 代码评审 …… 30
2.1.2 结构分析 …… 31
2.2 程序流程分析 …… 32
2.2.1 控制流分析 …… 32
2.2.2 数据流分析 …… 33
2.3 符号执行 …… 35
2.3.1 静态符号执行 …… 36
2.3.2 动态符号执行 …… 38
2.4 编程规范和规则 …… 39
2.5 程序静态分析工具 …… 42
2.5.1 工具简介 …… 42
2.5.2 工具安装与评估 …… 42
2.6 小结 …… 48
习题 2 …… 48

第 3 章 白盒测试 …… 50

3.1 逻辑覆盖测试 …… 50
3.1.1 语句覆盖 …… 52
3.1.2 分支覆盖 …… 54
3.1.3 条件覆盖 …… 56
3.1.4 条件 / 判定覆盖 …… 58
3.1.5 修正条件 / 判定覆盖 …… 61
3.1.6 条件组合覆盖 …… 63
3.2 路径覆盖测试 …… 65
3.2.1 环复杂度 …… 65
3.2.2 基本路径覆盖 …… 66
3.2.3 主路径覆盖 …… 69
3.2.4 循环结构测试 …… 72

3.3 小结 …… 74
习题 3 …… 74

第 4 章 程序插桩与变异测试 …… 77

4.1 程序插桩 …… 77
4.1.1 程序插桩概述 …… 77
4.1.2 程序插桩示例 …… 78
4.1.3 程序插桩工具 JaCoCo …… 79
4.2 变异测试 …… 84
4.2.1 变异测试概述 …… 84
4.2.2 变异测试方法 …… 85
4.2.3 变异测试工具 PITest …… 89
4.3 小结 …… 91
习题 4 …… 91

第 5 章 单元测试 …… 94

5.1 单元测试概述 …… 94
5.2 单元测试框架 …… 95
5.3 单元测试内容 …… 96
5.3.1 算法逻辑 …… 96
5.3.2 模块接口 …… 97
5.3.3 数据结构 …… 97
5.3.4 边界条件 …… 98
5.3.5 独立路径 …… 98
5.3.6 错误处理 …… 99
5.3.7 输入数据 …… 99
5.3.8 表达式与 SQL 语句 …… 100
5.4 慕测单元测试实例 …… 100

5.5 小结 …… 102
习题 5 …… 103

第 6 章 集成测试 …… 105

6.1 集成测试概述 …… 105
6.1.1 集成测试过程 …… 106
6.1.2 集成测试缺陷类型 …… 107
6.2 集成测试分析 …… 110
6.3 集成测试策略 …… 112
6.3.1 一次性集成与增量式集成 …… 112
6.3.2 自顶向下与自底向上集成 …… 114
6.3.3 基于调用图的集成 …… 117
6.3.4 其他集成测试策略 …… 118
6.4 小结 …… 124
习题 6 …… 124

第 7 章 JUnit 基础 …… 127

7.1 一个 JUnit 实例 …… 127
7.2 注解 …… 132
7.3 测试类与测试方法 …… 135
7.3.1 Assert …… 135
7.3.2 TestCase …… 139
7.3.3 TestResult …… 141
7.3.4 TestSuite …… 142
7.4 错误与异常处理 …… 143
7.4.1 错误和异常 …… 143
7.4.2 异常处理 …… 144
7.5 批量测试 …… 146

7.5.1 参数化测试 …… 146
7.5.2 打包测试 …… 149
7.6 小结 …… 150
练习 7 …… 150

第 8 章 JUnit 深入应用 …… 152

8.1 匹配器 …… 152
8.2 JUnit 测试进阶 …… 154
8.2.1 Controller 测试 …… 154
8.2.2 Stup 测试 …… 155
8.2.3 Mock 测试 …… 159
8.2.4 Private 测试 …… 162
8.3 JUnit 集成 …… 165
8.3.1 JUnit-Ant 集成 …… 165
8.3.2 JUnit-Maven 集成 …… 169
8.4 小结 …… 171
练习 8 …… 171

附录 慕测科技——开发者测试平台 …… 172

参考文献 …… 192

第 1 章
Chapter 1

开发者测试概述

早期软件规模小且复杂程度低，软件测试常常包含于调试工作中，由软件开发人员完成。在当今规模化和工程化的软件研发中，为了提高生产效率和专业化程度，相关工作逐步细分，出现了专门的软件测试岗位。

为了获得更高的用户忠诚度，扩大产品市场份额，在当前移动互联网迅速普及的背景下，众多软件公司纷纷采用了微小改进、快速迭代、反馈收集、及时响应等手段来迅速改进软件产品，满足用户需求。软件的持续快速迭代需求大大压缩了软件开发的发布流程，使得一部分测试任务开始迁移，由软件开发人员担任的这部分与代码相关的软件测试工作，我们统称为开发者测试。开发者测试包括传统的单元测试、集成测试、接口测试，甚至部分与系统测试相关的任务。

1.1 开发者与软件测试

开发者需要对自己开发的程序代码承担质量责任。在软件质量管理机制下，一般要求开发者首先自行对自己编写的代码进行审查和测试，并保证提交的代码已达到一定的质量标准。开发者测试中的单元测试和集成测试主要采用白盒测试方法，要求测试人员对软件代码非常熟悉。这样的测试任务由软件开发人员来做效率会更高。

1.1.1 测试和调试

在软件开发过程中，开发者需要对程序进行测试和调试。测试和调试极其相

关但含义完全不同。简单来说，测试是为了发现缺陷，调试是为了修复缺陷。调试往往需要依赖已有的测试信息或者补充更多测试信息，需要先找出缺陷根源和缺陷的具体位置，再进行修复以消除缺陷。而从职责上说，测试只需要发现缺陷，并不需要修复缺陷。在软件开发过程中，开发者需要同时肩负这两种职责，对自己开发的程序进行测试，发现缺陷并对其进行调试以修复缺陷。调试的过程如图 1-1 所示。

图 1-1　调试的过程

调试时如果已经识别或者找到测试中所发现缺陷的产生原因，就可以直接修复，然后进行回归测试。如果没有找到缺陷的产生原因，可以先假设一个最有可能的原因，并通过附加测试来验证这样的假设是否成立，直到找出原因为止。

调试工作是程序员能力和水平的一个重要体现。软件开发调试有时难度很大，原因如下：1）失效症状和缺陷原因可能相隔很远，高度耦合的程序结构加重了这种情况；2）失效症状可能在另一缺陷修复后消失或暂时性消失；3）失效症状由不太容易跟踪的人为错误引发；4）失效症状可能是由不同原因耦合引发的。因此，程序员有时会因为在调试程序时找不到问题所在，而使软件开发工作陷入困境。

程序调试方法多种多样，更多时候是依赖程序员的经验及其对程序本身的理解。调试方法的具体实施可以借助调试工具来完成，如带调试功能的编译器、动态调试辅助工具“跟踪器”、内存映像工具等。

回溯法是指从程序出现不正确结果的地方开始，沿着程序的运行路径向上游

寻找错误的源头，直到找出程序错误的实际位置。例如，程序有 5000 行，测试发现最后输出的结果是错误的，采用回溯法，可以先在第 4500 行插桩，检查中间结果是否正确。若正确，则错误很可能发生在第 4500 ～ 5000 行之间。若不正确，则在第 4000 行插桩，以此类推，直到找出程序错误的具体位置。

1.1.2　开发者测试

从履行职责、提高效率、保护源代码、方便实现等角度来说，开发者需要完成的测试工作主要集中在单元测试和集成测试阶段。程序代码开发出来之后，开发者先对自己开发的代码进行单元测试，然后再把多个已经通过单元测试的模块按照设计书组装起来进行集成测试。当然，在实践中也会出现后期的系统测试需要开发者配合甚至主导的情况。

静态测试与动态测试都是开发者需要掌握的测试方法，在实践中一般将两者结合起来使用。开发者对自己开发的程序代码进行检查，这是静态测试；而开发者运行代码，给定输入数据，检查程序能否正常运行并给出预期结果，则是动态测试。对静态测试存疑的代码部分，加强动态测试进行结果验证，是实践中常用的开发者测试策略。

白盒测试是开发者最主要的测试方法，也是在软件测试工作上体现开发者优势的地方。然而，并不是说开发者测试不需要黑盒测试方法。恰恰相反，我们建议开发者在实践中对程序代码进行等价类和边界值分析，这有助于提高开发者测试的效率和质量。而在集成测试或者配合一些复杂模块的测试中，开发者也可能会用到灰盒测试等方法。

在开发者测试中，手工测试与自动化测试都会用到。随着软件技术的发展，软件测试的自动化程度会越来越高。开发者在测试中应尽可能通过自动化测试工具来提高测试工作效率。但并不是所有测试工作都能够自动化完成，也不是所有场景都适用自动化测试。

为了提高软件质量和缩短软件项目总工期，测试常常与开发同步进行。研发人员应综合运用多种软件测试方法和技术，针对不同的测试场景合理选择测试方

法和工具，并在测试时尽可能采用自动化测试工具来提高软件测试的效率。

总体上，开发者测试一般采取先静态后动态的组合方式：先进行静态结构分析、代码评审，再进行代码的覆盖性测试。利用静态分析的结果作为导引，通过代码评审和动态测试的方式对静态测试结果进行进一步的确认，使测试工作更为有效。代码覆盖测试是白盒测试的重点，通常可采用语句覆盖、分支覆盖等。对于软件的重点模块，可使用逻辑覆盖、路径覆盖、数据流覆盖等更复杂的准则。在不同的测试阶段，测试重点有所不同。在单元测试阶段，以代码审查和语句覆盖为主；在集成测试阶段，需要增加接口测试和模块集成结构分析等。

1.1.3　PIE 模型

软件测试的主要目的之一是发现缺陷。动态测试工作中通常会出现复杂而有趣的现象。假设某个程序中有一行代码存在缺陷，在该软件的某次运行中，这个存在缺陷的代码行并不一定会被执行。即使这存在缺陷的代码被执行，若没有达到某个特定条件，程序状态也不一定会出错。即只有在运行错误代码，并达到某个特定的条件，程序状态出错并传播出去被外部感知后，测试人员才能发现程序中的缺陷。

软件测试的一个基本模型称为 PIE（Propagation-Infection-Execution）模型。PIE 模型对于理解软件测试方法、测试过程、缺陷定位和程序修复等都具有重要作用。在介绍 PIE 模型之前，我们首先理解缺陷在不同阶段的不同名称及其含义：

- Fault（故障）：故障是指静态存在于程序中的缺陷代码，有时也称之为程序缺陷（Defect）。
- Error（错误）：错误是指程序运行缺陷代码后导致的错误状态。
- Failure（失效）：失效是指程序错误状态传播到外部被感知的现象。

针对缺陷不同阶段的性质，我们可以构建一个 PIE 模型来解释缺陷产生的整体过程。PIE 提示我们，发现一个缺陷需要满足以下三个必要条件：

1）Execution（运行）：测试必须运行到包含缺陷的程序代码。

2）Infection（感染）：程序必须被感染出一个错误的中间状态。

3）Propagation（传播）：错误的中间状态必须传播到外部并被观察到。

上述三个条件是缺陷被检测出来的必要条件，三个必要条件组合成检测出缺陷的充分条件，即充要条件。我们不难理解，一个测试满足条件 1 不一定能满足条件 2，即测试运行到包含缺陷的代码，但不一定能感染出错误的中间状态。一个测试满足条件 2 不一定能满足条件 3，即测试能感染出错误的中间状态（当然也运行到了包含缺陷的代码），但不一定能成功传播出去并被测试人员发现。

为了进一步理解这些现象，以图 1-2 中的示例程序 MY_AVG 来进行说明。程序语句 s_4 存在缺陷，循环控制变量 i 的初值应为 0，而不是 1。

s_0	public static void MY_AVG(int[] numbers) {
s_1	int length = numbers.length;
s_2	double V_avg, V_sum;
s_3	V_sum = 0.0;
s_4	for(int i=1; i<length; i++) {　　**// Fault: i 初始值应为 0**
s_5	V_sum += numbers[i];
s_6	}
s_7	V_avg = V_sum / (double) length;
s_8	System.out.println("V_avg:"+V_avg);
s_9	}

图 1-2　一个示例程序 MY_AVG

在程序的某次运行中，调用了上述代码段 MY_AVG，并输入测试数据 0，4，5，预期（正确）输出结果是 3。程序运行到了 s_4（条件 1 满足），中间变量 V_sum 应该为 0 + 4 + 5 = 9。由于缺陷导致数组第一个数字“0”被遗漏，中间变量 V_sum 为 4 + 5 = 9（条件 2 不满足）。但由于 0 的累加不影响最终结果，最终的平均值为 3，与预期输出结果一致。

我们将上述程序做一下微调，缺陷依然是循环初值 1，如图 1-3 所示。

s_0	public static void MY_AVG(int[] numbers) {
s_1	int n;
s_2	double V_avg, V_sum;
s_3	V_sum = 0.0;
s_4	n = 0;
s_5	for(int i=1; i<length; i++) { // **Fault: i 初始值应为 0**
s_6	V_sum += numbers[i];
s_7	n++;
s_8	}
s_9	V_avg = V_sum / (double) n;
s_{10}	System.out.println("V_avg:"+V_avg);
s_{11}	}

图 1-3 简单修改后的示例程序 MY_AVG

在程序的某次运行中，调用了上述代码段 MY_AVG，并输入测试数据 4，3，5，预期（正确）输出结果是 4。程序运行到了语句 s_4（条件 1 满足），中间变量 V_sum 应该为 4 + 3 + 5 = 12。由于缺陷导致数组第一个数字“4”被遗漏，中间变量 V_sum 为 3 + 5 = 8（条件 2 满足）。但个数变量 n 的累加也减少了 1，即 n 原来应该为 3，现在是 2（条件 2 满足）。这两个错误的中间状态叠加后，V_avg = 12/3 = 8/2 = 4，导致最终结果正确（条件 3 不满足）。

PIE 模型表明发现 Bug 并不是一件容易的事情。要全面发现软件 Bug，不仅需要针对特定需求和软件特性进行测试设计，还需要学会利用不同的软件测试方法，如有效地结合使用白盒测试和黑盒测试方法。

1.2 开发者测试方法与技术

软件测试方法有很多分类。本节仅介绍与开发者测试相关的分类。软件测试依据是否需要运行程序可以分为静态测试与动态测试，依据是否需要了解软件内部结构可以分为黑盒测试和白盒测试。

1.2.1 静态测试与动态测试

静态测试不运行被测程序，而是手工或者借助专用的软件测试工具来检查软