软件测试策略、设计及其自动化实战

——— Linux、Android、Windows、Web 的全面软件测试

于 艳 / 编著

西安电子科技大学出版社
http://www.xduph.com

软件测试策略、设计及其自动化实战

——Linux、Android、Windows、Web的全面软件测试

于艳 编著

西安电子科技大学出版社

内 容 简 介

本书主要介绍软件测试的策略、建模、设计与不同平台的自动化技术。全书分为两大部分：第一部分包括第1～5章，主要介绍前沿的测试理论、测试流程、测试策略模型、测试建模、测试设计和用例设计、探索性测试、测试质量度量与改进、自动化基础知识、自动化框架的开发等，可帮助学习者成长为优秀的测试架构师；第二部分包括第6～11章，针对Linux、Android、Web、Windows下的用户界面UI、命令行CLI、单元接口等，分别论述了测试方法、当前流行的自动化技术与工具以及实际案例与实践总结，可以提升实际项目的自动化覆盖率，帮助学习者成长为全面的自动化测试专家。

本书涵盖了软件测试各个方面的知识，介绍了各种先进的前沿技术，涵盖了理论和实际案例，内容全面。

本书适合App测试人员、移动设备测试人员、Web测试人员、驱动测试人员、Linux测试人员、Windows测试人员、自动化测试人员、质量保证人员等阅读学习，也适合作为软件测试课程和测试培训的参考资料，还适合作为大专院校相关专业和培训学校的教材。

图书在版编目(CIP)数据

软件测试策略、设计及其自动化实战——
Linux、Android、Windows、Web的全面软件测试/于艳编著. —西安：西安电子科技大学出版社，2019.6
ISBN 978-7-5606-5262-7

Ⅰ. ①软… Ⅱ. ①于… Ⅲ. ①软件—测试 Ⅳ. ①TP311.55

中国版本图书馆CIP数据核字(2019)第037425号

策划编辑 戚文艳
责任编辑 祝婷婷 阎彬
出版发行 西安电子科技大学出版社(西安市太白南路2号)
电 话 (029)88242885 88201467 邮 编 710071
网 址 www.xduph.com 电子邮箱 xdupfxb001@163.com
经 销 新华书店
印刷单位 陕西天意印务有限责任公司
版 次 2019年6月第1版 2019年6月第1次印刷
开 本 787毫米×1092毫米 1/16 印张 24
字 数 569千字
印 数 1～3000册
定 价 55.00元
ISBN 978-7-5606-5262-7/TP
XDUP 5564001-1

前　言

笔者从事一线的软件测试工作已经十余年，在此期间，阅读了大量的测试文献资料，参与了很多不同类型的测试项目，从实践中逐渐构建出自己的知识体系和经验。本书的主要内容就是这些知识体系和经验的总结。希望本书能帮助测试人员快速建立测试理念，掌握切合实际的综合性技能，通过理论结合实践的方式解决实际工作中的问题，提升工作效率和质量。

笔者在以往面试过程中发现：一方面，大部分应聘人员对测试策略、建模、设计只知道皮毛，且不会结合实际项目灵活应用，导致测试架构师职位常常空缺很久也不能招聘到合适的人员。尽管有的人拥有多年测试经验，但是除了业务知识以外，对测试本身并没有进行思考和总结，一旦离开熟悉的业务领域就又变成了新人，之前的经验很难复用，而且即使在自己熟悉的业务领域，由于本身的测试核心能力不足，所以发展也受到了限制。另一方面，应聘人员欠缺自动化技术，测试金字塔中往往只懂上层不懂底层，而且随着软件越来越复杂，某个领域的测试往往会涉及其他领域的各种技术，通过了解不同领域的测试技术和自动化，开拓思维，有助于进一步提升核心能力，更好地实现整体把控，解决工作中遇到的各种问题。以手机终端测试为例，其自动化技术会涉及 Android 系统各个架构下对应的不同的自动化技术，如 UI 自动化、Framework 的仪表盘技术、HAL 的单元接口自动化等。在实际项目中还可能会涉及 Linux、Web 以及 Windows 的自动化技术，以便更好地提升自动化覆盖率，这一部分测试技术会涉及驱动、OS 和上层应用等。针对这些问题本书介绍了一些可以借鉴的方法，期待读者依据这些方法，根据被测项目的实际特点，构建满足项目实际需求的测试策略、设计和自动化等。

本书的组织结构如下：

第一部分包括第 1 章到第 5 章，介绍了前沿的测试必备理论、测试策略模型、测试与设计建模、自动化测试理论与自动化框架解决方案等。其中，第 1 章主要介绍了最新的软件测试概念、测试类型、质量属性、开发生命周期模型、漫游测试、软件测试分类、测试过程阶段等。第 2 章介绍了通过 HTSM 模型、ACC 模型及 TEmb 方法输出测试策略。第 3 章概述了各类测试模型、测试设计技术与用例设计技术，介绍了如何进行测试设计、建模和输出用例，并介绍了缺陷分析法、软件质量评估与质量管理等。第 4 章介绍了自动化测试概述、测试金字塔、自动化测试工具实现原理、自动化测试脚本技术、自动错误预防(AEP)机制等。第 5 章给出了实现跨平台自动化的整体解决方案，包括自动化下载源码和 MD5 校验、自动化编译与打包、自动化分发测试工具、自动化执行测试、自动化上传结果到 Testlink、自动化对比不同版本的源码并发送邮件、性能监测、自动化画图、精准测试等。

第二部分包括第 6 章到第 11 章，介绍各个领域常用的自动化技术、测试方法以及对应的系列工具。第 6 章介绍了白盒测试方法、接口测试方法、Gtest 单元测试框架及覆盖率工具、Python 的 Unittest 单元测试框架以及其他语言的单元测试框架等。第 7 章介绍了 Linux 测试类型、CLI 命令行的自动化、Linux GUI 自动化以及常用的系列工具等。第 8 章介绍了 Android 系统架构及各层架构下的各个自动化技术，如 Android 上层 UI、Frame-

work 层、底层如 HAL 和 Kernel 层的自动化技术、App 测试方法以及常用的系列工具等。第 9 章介绍了 Windows 的自动化技术、自动化工具、猴子与模糊测试以及常用的系列工具等。第 10 章介绍了 Web 测试方法与工具、UI 自动化以及常用的系列工具。第 11 章介绍了 Web 性能策略与测试设计、性能测试方法以及 JMeter 性能工具等。

本书不要求读者掌握特定的背景知识，读者可以思考本书的测试技术方法，并应用于实际测试项目，评估其效果，通过评估和思考，掌握原理和细节，演化成新的测试技术方法。希望本书可以帮助初学者迅速了解软件测试全过程与相关技术，同时也能够帮助中高级工程师系统梳理测试技术并构建自己的测试体系，从而升级为测试架构师或者自动化测试专家。

感谢所有曾经支持和帮助过我的人。特别感谢研华科技的开发经理梁继超参与本书部分章节的审查与修订。感谢我的爸爸、妈妈和那些默默关心我的人，写这本书花费了很长的时间，是你们让我坚持自己的理想。感谢我的女儿梁澜馨，谢谢你给妈妈带来很多快乐，谢谢你对妈妈写作无法陪伴你的理解和支持！还要感谢西安电子科技大学出版社戚文艳编辑的悉心帮助和指导。

很高兴和大家分享十余年的经验、思考与总结。由于笔者水平有限，很多内容都是自己的经验总结，难免会出现错误，欢迎各位读者不吝指正。如果在阅读本书过程中有任何问题或者建议，欢迎随时发送邮件到 451193604@qq.com，笔者将尽量给您答疑解惑。

编　者

2019.2

目　录

第 1 章　软件测试必备理论 …… 1
1.1　软件测试知识 …… 1
1.1.1　软件测试 …… 1
1.1.2　软件缺陷 …… 1
1.1.3　软件测试应遵循的七原则 …… 2
1.2　软件测试方法 …… 3
1.2.1　功能测试法 …… 3
1.2.2　性能测试法与流程 …… 3
1.2.3　负载测试法 …… 4
1.2.4　压力测试法与驱动压力测试案例 …… 4
1.2.5　安全性测试法与案例 …… 6
1.2.6　UI 与 UE 测试法 …… 8
1.2.7　国际化、本地化、全球化测试法 …… 9
1.2.8　安装与卸载测试法 …… 10
1.2.9　兼容性测试法与案例 …… 10
1.2.10　故障转移与恢复性测试法 …… 11
1.2.11　容量测试法 …… 12
1.2.12　可靠性测试法与案例 …… 12
1.2.13　可访问性测试法 …… 12
1.3　软件产品质量模型 …… 13
1.3.1　功能性 …… 13
1.3.2　可靠性 …… 13
1.3.3　易用性 …… 14
1.3.4　效率 …… 15
1.3.5　可维护性 …… 15
1.3.6　可移植性 …… 15
1.3.7　质量属性与测试类型的对应关系 …… 16
1.3.8　质量属性的定制化使用 …… 16
1.4　软件开发生命周期模型 …… 17
1.4.1　大棒模型 …… 17
1.4.2　边做边改模型 …… 17
1.4.3　瀑布模型 …… 17
1.4.4　V 模型 …… 17
1.4.5　快速原型模型 …… 18
1.4.6　螺旋模型 …… 18
1.4.7　敏捷软件开发 …… 18
1.4.8　持续集成与常见问题说明 …… 19
1.5　漫游测试的方法与管理 …… 20
1.5.1　常用的漫游测试法 …… 20
1.5.2　探索性软件测试法 …… 20
1.5.3　漫游测试的选取与定制化 …… 25
1.6　软件测试分类 …… 26
1.6.1　黑盒、白盒、灰盒测试 …… 26
1.6.2　静态、动态测试 …… 27
1.6.3　单元、集成、系统、验收测试 …… 27
1.7　测试过程阶段 …… 28
1.7.1　测试计划与控制阶段 …… 28
1.7.2　测试设计阶段 …… 29
1.7.3　测试执行阶段 …… 29
1.7.4　评估结束准则和测试报告阶段 …… 30
1.7.5　测试结束活动 …… 31
第 2 章　测试策略模型 …… 32
2.1　测试策略概述 …… 32
2.1.1　测试活动步骤说明 …… 32
2.1.2　测试策略概述 …… 32
2.1.3　回归测试 …… 32
2.1.4　BVT 测试与冒烟测试的区别 …… 33

2.2 启发式测试策略模型(HTSM) …… 33
2.2.1 启发式测试策略模型(HTSM) …… 33
2.2.2 启发式测试策略模型(HTSM)的定制化 …… 36
2.3 Google ACC 建模 …… 37
2.4 测试策略 TEmb 方法 …… 40
2.4.1 通用元素 LITO …… 41
2.4.2 产品特性与案例说明 …… 42
2.4.3 风险分析 …… 43
2.4.4 组合测试策略的机制与案例 …… 46
第 3 章 测试建模、设计技术与质量管理 …… 48
3.1 测试设计技术 …… 48
3.1.1 功能列表 …… 48
3.1.2 功能交互分析 …… 48
3.1.3 输入输出模型 …… 49
3.1.4 状态机模型 …… 49
3.1.5 组合分析模型 …… 50
3.1.6 错误推测法 …… 51
3.2 测试用例设计方法 …… 52
3.2.1 等价类划分法 …… 52
3.2.2 边界值分析法 …… 53
3.2.3 判定表因果图法 …… 55
3.2.4 测试用例及其检查点 …… 55
3.3 测试建模技术 …… 56
3.3.1 测试模型概述 …… 56
3.3.2 基于模型的测试(MBT) …… 56
3.3.3 常用的基于模型的测试工具 …… 57
3.4 如何进行测试设计与建模 …… 58
3.4.1 了解目标和项目环境信息 …… 58
3.4.2 基于 5W1H 的需求分析 …… 58
3.4.3 MFQ 测试设计模型 …… 59
3.4.4 PPDCS 测试建模步骤 …… 59
3.5 缺陷分析法 …… 60
3.5.1 缺陷分析方法 …… 60
3.5.2 ODC 缺陷分析法 …… 62
3.5.3 四象限分析法 …… 64
3.5.4 Gompertz 模型分析法 …… 64
3.5.5 根本原因分析(RCA) …… 65
3.6 软件质量评估 …… 65
3.6.1 测试覆盖率评估 …… 66
3.6.2 测试过程评估 …… 66
3.6.3 质量评估 …… 67
3.6.4 版本质量评估 …… 67
3.7 质量管理 …… 68
3.7.1 软件质量管理三部曲 …… 68
3.7.2 能力成熟度模型(CMMI) …… 69
3.7.3 全面质量管理(TQM) …… 69
3.7.4 QC 旧七工具 …… 70
3.7.5 QC 新七工具 …… 72
3.7.6 事后回顾(AAR) …… 72
3.7.7 项目回顾会议 …… 73
第 4 章 自动化测试必备理论 …… 74
4.1 自动化测试知识 …… 74
4.1.1 软件测试自动化概述 …… 74
4.1.2 测试金字塔(Test Pyramid) …… 75
4.1.3 蛋筒冰激凌模式 …… 76
4.2 自动化测试工具知识 …… 77
4.2.1 测试工具分类 …… 77
4.2.2 单元、接口测试工具的实现原理 …… 77
4.2.3 UI 测试工具的实现原理 …… 77
4.2.4 Web UI 测试工具的实现原理 …… 78
4.2.5 性能测试工具的实现原理 …… 78
4.3 自动化测试脚本技术 …… 79
4.3.1 线性脚本技术与启发 …… 79
4.3.2 结构化脚本技术与启发 …… 79
4.3.3 共享脚本技术与启发 …… 80
4.3.4 数据驱动脚本技术与启发 …… 80
4.3.5 关键字驱动脚本技术与启发 …… 81
4.3.6 自动化成熟度等级 …… 82
4.3.7 自动化脚本衡量标准与提升法 …… 82
4.4 自动错误预防(AEP)机制 …… 83
第 5 章 一键式测试自动化框架 …… 84
5.1 一键式测试自动化概述 …… 84
5.1.1 Python 概述 …… 84
5.1.2 手工测试流程 …… 84
5.1.3 一键式测试流程自动化 …… 85
5.1.4 环境准备 …… 85
5.2 配置文件及其读取介绍 …… 86
5.3 自动化下载源码和 MD5 校验 …… 88

5.3.1 SVN 自动化下载程式 …… 88
5.3.2 FTP 自动化下载待测标的物 …… 90
5.3.3 MD5 自动化校验 …… 97
5.3.4 自动化脚本调用执行 …… 101
5.4 自动化编译和打包 …… 102
5.4.1 Windows 下的自动化编译 …… 102
5.4.2 Windows 下自动化编译的调用步骤 …… 104
5.4.3 Linux 下的自动化编译打包 …… 104
5.5 自动化分发测试工具 …… 105
5.5.1 自动化传输文件到 DUT 端 …… 105
5.5.2 自动化脚本调用步骤 …… 106
5.6 自动化执行测试 …… 107
5.6.1 远程调用自动化 …… 107
5.6.2 自动化执行测试 …… 107
5.7 自动化上传测试结果到 Testlink …… 107
5.7.1 Testlink API 介绍 …… 107
5.7.2 上传测试结果到 Testlink 的实现代码 …… 109
5.7.3 实现自动化上传结果到 Testlink …… 110
5.8 自动化对比不同版本的源码并发送邮件 …… 111
5.8.1 自动化对比源码 …… 111
5.8.2 自动化对比源码调用步骤 …… 112
5.8.3 自动化发送邮件 …… 112
5.8.4 自动化发送邮件调用步骤 …… 114
5.9 自动化框架 …… 114
5.9.1 框架总调用 …… 114
5.9.2 精准测试技术 …… 115
5.10 性能监测 …… 116
5.10.1 Psutil 介绍 …… 116
5.10.2 Psutil 的实例展示 …… 117
5.11 自动化画图 …… 125
5.11.1 基于 Excel 的自动化画图 …… 125
5.11.2 基于 Highcharts 的自动化画图 …… 127
5.11.3 基于 Gnuplot 的自动化画图 …… 134
第 6 章 单元自动化测试 …… 136
6.1 单元测试的白盒测试 …… 136
6.2 接口测试设计技术 …… 138
6.3 Python 的 Unittest 框架 …… 142
6.3.1 Unittest 概述与案例 …… 142
6.3.2 管理测试用例与案例 …… 144
6.3.3 discover 方法与案例 …… 148
6.3.4 跳过测试法与案例 …… 149
6.3.5 HTMLTestRunner 生成测试报告与案例说明 …… 150
6.3.6 HTMLTestRunner 集成测试报告与案例说明 …… 151
6.4 跨平台 C++ Googletest 框架 …… 152
6.4.1 Googletest 概述 …… 152
6.4.2 参数化介绍 …… 153
6.4.3 Android 中的 Gtest 测试框架 …… 154
6.4.4 Android Gtest 案例 …… 157
6.4.5 GCOV 与 LCOV 代码覆盖率测试 …… 162
6.4.6 GCOV 和 LCOV 的使用方法与案例 …… 163
6.5 其他语言的单元测试框架 …… 165
6.5.1 Java 的单元测试框架与案例介绍 …… 165
6.5.2 C# 的单元测试框架 Nunit …… 166
第 7 章 Linux 测试 …… 167
7.1 Linux OS 测试类型 …… 167
7.1.1 Linux OS 压力测试与案例 …… 167
7.1.2 Linux OS 稳定性测试与案例 …… 169
7.1.3 Linux OS 性能测试与案例 …… 170
7.1.4 Linux OS 兼容性测试 …… 172
7.1.5 Linux API 测试 …… 173
7.1.6 Linux 的其他测试 …… 173
7.2 命令行类的 CLI 自动化测试 …… 173
7.2.1 CLI 自动化技术 …… 173
7.2.2 shell 编程实现自动化案例 …… 173
7.2.3 Telnet 自动化登录案例 …… 176
7.2.4 FTP 自动化登录与上传文件案例 …… 177
7.2.5 SSH 自动化登录案例 …… 178
7.3 Linux GUI 自动化 LDTP 测试 …… 180
7.3.1 LDTP 自动化框架技术 …… 180

7.3.2 LDTP 的安装 …… 181
7.3.3 LDTP 具体使用案例说明 …… 181
7.3.4 LDTP 获取应用程序信息 …… 182
7.3.5 ldtpeditor 录制脚本 …… 182
7.3.6 Gedit 案例讲解 …… 182
7.3.7 Firefox 案例讲解 …… 183
7.4 Linux GUI 自动化 X11::GUITest … 184
7.4.1 X11::GUITest 的安装 …… 184
7.4.2 X11:GUITest 案例讲解 …… 185
7.4.3 Recorder 安装与使用 …… 187
7.5 LTP 内核测试工具介绍 …… 187
7.5.1 LTP 介绍 …… 187
7.5.2 STAF 介绍 …… 189
7.5.3 STAF 与 LTP 的集成 …… 189
7.6 Linux 常用系列工具 …… 190
7.6.1 Linux CPU 性能分析系列工具 …… 190
7.6.2 Linux 内存分析系列工具 …… 191
7.6.3 存储系统分析工具 …… 193
7.6.4 网络性能工具 …… 196
7.6.5 磁盘 I/O 分析系列工具介绍 …… 199
7.6.6 静态分析工具 cppcheck 与案例 …… 199
7.6.7 性能测试工具 lmbench …… 201
7.6.8 GPU 测试工具 …… 201
7.6.9 Screentest 测试工具 …… 202
7.6.10 浏览器测试系列工具 …… 202
7.6.11 Docker 环境搭建 …… 204
第 8 章 Android 测试 …… 205
8.1 Android 技术 …… 205
8.1.1 Android 架构 …… 205
8.1.2 Android 各架构的自动化技术 …… 206
8.1.3 Android 开发环境搭建 …… 207
8.2 Monkey 自动化工具 …… 208
8.2.1 Monkey 介绍 …… 208
8.2.2 Monkey 语法与实际指令 …… 209
8.3 MonkeyRunner 自动化工具 …… 210
8.3.1 MonkeyRunner 介绍 …… 210
8.3.2 MonkeyRunner API …… 211
8.3.3 MonkeyDevice API …… 211
8.3.4 MonkeyImage API 与案例 …… 212
8.3.5 MonkeyRunner 录制与回放 …… 213
8.3.6 快捷键与案例 …… 214
8.3.7 MonkeyRunner 案例说明 …… 215
8.3.8 EasyMonkeyDevice 介绍与案例 …… 216
8.4 基于 Framework 的 Instrumentation 自动化工具 …… 218
8.4.1 Junit 单元自动化框架介绍与案例 …… 218
8.4.2 Instrumentation 介绍 …… 219
8.4.3 hierarchyviewer 捕获控件信息 …… 219
8.4.4 创建 Instrumentation 自动化测试程序案例 …… 221
8.5 基于 UI 的 UI Automator 测试工具 …… 224
8.5.1 UIAutomator 介绍 …… 224
8.5.2 UI Automator Viewer 获取 UI 元素信息 …… 225
8.5.3 UI Automatior UiObject API …… 226
8.5.4 UI Automatior UiDevice API …… 227
8.5.5 UI Automatior UiSelector API … 229
8.5.6 UI Automatior UiCollection API … 230
8.5.7 UI Automatior UiScrollable API … 230
8.5.8 UI Automatior UiWatcher API …… 231
8.5.9 UI Automator TestCase …… 232
8.6 基于 UI 的 Robotium 自动化工具 …… 232
8.6.1 Robotium 介绍 …… 232
8.6.2 基于源码的 Robotium 自动化与案例 …… 233
8.6.3 基于 APK 的 Robotium 自动化与案例 …… 234
8.6.4 UI 控件查看工具 …… 236
8.6.5 Recorder 录制工具 …… 236
8.7 基于 UI 的 Appium 自动化工具 …… 237
8.7.1 Appium 介绍 …… 237
8.7.2 Appium 安装 …… 237
8.7.3 Appium 的设置界面 …… 238
8.7.4 Appium Inspector 与案例说明 …… 241
8.7.5 UI AutomatorViewer 工具与案例说明 …… 243

8.7.6 Appium 自动化案例说明 …… 245
8.7.7 Pycharm 介绍 …… 248
8.7.8 HTMLTestRunner 生成测试报告案例 1 …… 250
8.7.9 HTMLTestRunner 生成测试报告案例 2 …… 251
8.7.10 Webview 控件识别 …… 256
8.8 Android App 测试方法 …… 257
8.8.1 Android 应用的硬件特性测试 …… 257
8.8.2 Android 应用的内存测试 …… 258
8.8.3 Android 应用的流量测试 …… 259
8.8.4 Android 不同网络下的测试 …… 260
8.8.5 Android 应用的弱网测试与网络模拟工具 NEWT …… 262
8.8.6 Android 应用的多任务测试 …… 269
8.8.7 Android 应用的消息提示测试 …… 270
8.8.8 Android 应用的 Dalvik 与 ART 测试 …… 270
8.8.9 Android 应用的耗电量测试 …… 270
8.8.10 Android 应用的特性测试 …… 271
8.8.11 Android 应用的兼容性测试 …… 271
6.8.12 Android 应用的安全性测试 …… 271
8.8.13 Android 应用的安装卸载测试 … 272
8.8.14 Android 应用的用户体验测试 …… 272
8.8.15 Android 应用的性能测试 …… 272
8.8.16 Android 应用的启动时间测试 …… 273
8.8.17 机器学习在 App 启动时间应用案例 …… 273
8.8.18 Android 应用的其他测试 …… 276
8.8.19 Android 应用的典型问题 …… 276
8.8.20 代码扫描测试 …… 277
8.8.21 云测试平台 …… 277
8.9 Android 常用测试系列工具 …… 278
8.10 CTS 测试 …… 280
8.10.1 CTS 介绍 …… 280
8.10.2 搭建测试环境 …… 280
8.10.3 CTS 运行 …… 281
8.10.4 结果分析 …… 282
8.10.5 CTS Verifier 运行 …… 283
8.11 Android 开发系列工具 …… 283
8.11.1 adb 工具介绍 …… 283
8.11.2 DDMS 介绍 …… 284
8.11.3 静态代码扫描工具 Findbugs …… 285
8.11.4 静态代码扫描工具 Lint …… 286
8.11.5 Android 的内存泄露工具 (MAT) …… 288
8.11.6 HTTP 抓包工具 Fiddler …… 289
8.11.7 App 性能检测工具 GT …… 289
第 9 章 Windows 测试 …… 290
9.1 Windows UI 自动化测试工具介绍 …… 290
9.2 Windows UI 自动化测试技术 …… 290
9.2.1 Windows API 技术 …… 290
9.2.2 MSAA 技术 …… 291
9.2.3 UI Automation 技术 …… 292
9.2.4 基于 Reflection 反射的 UI 测试技术 …… 292
9.2.5 自动化常遇到的问题总结 …… 293
9.3 常用工具 AutoIt 介绍 …… 293
9.3.1 AutoIt 介绍 …… 293
9.3.2 Au3Info 获取信息工具 …… 295
9.3.3 AutoIt 案例说明 …… 296
9.3.4 Aut2exe 工具介绍 …… 297
9.4 Coded UI Test …… 297
9.4.1 Coded UI 测试介绍 …… 297
9.4.2 Coded UI Test 案例 …… 298
9.5 猴子测试与模糊测试 …… 301
9.5.1 猴子测试 …… 301
9.5.2 模糊 Fuzz 测试 …… 301
9.6 Windows 系列工具 …… 303
第 10 章 Web 测试 …… 304
10.1 测试方法与工具 …… 304
10.1.1 Web 功能测试 …… 304
10.1.2 Web 链接测试及工具 …… 305
10.1.3 Web 兼容性测试及工具介绍 …… 306
10.1.4 W3C 测试 …… 307
10.1.5 Web 安全性测试 …… 307
10.1.6 Web 代码合法性测试 …… 307

10.1.7 Web 的 UI、UE 测试 …… 308
10.1.8 契约测试 …… 308
10.1.9 Web 的 API 接口测试 …… 309
10.2 自动化测试工具 Selenium …… 309
10.2.1 Selenium 介绍 …… 309
10.2.2 Selenium IDE …… 310
10.2.3 Selenese 命令 …… 312
10.2.4 Selenium RC 介绍与不同语言的使用说明 …… 313
10.2.5 WebDriver 介绍与案例 …… 315
10.2.6 定位页面元素与对应脚本说明 …… 316
10.2.7 基于 Python 的 WebDriver 案例说明 …… 320
10.2.8 元素等待方法与案例说明 …… 322
10.2.9 Unittest 的案例说明 …… 324
10.2.10 HTMLTestRunner 的案例说明 …… 325
10.2.11 PageObject 页面对象设计模式 …… 327
10.2.12 结合 Junit 的 Java 案例说明 … 327
10.2.13 Selenium Grid 的安装与案例说明 …… 330
10.2.14 Jenkins 与 Selenium 的集成 …… 334
10.2.15 验证码的常用处理方式 …… 335
10.2.16 Web 自动化使用 AutoIt 工具 …… 335
10.2.17 Web 自动化使用 Sikuli 工具 …… 335
10.3 Web 系列测试工具介绍 …… 336
10.3.1 Browsershots 工具 …… 336
10.3.2 HD-Tach 工具 …… 337
10.3.3 Firefox 的 Firebug …… 337
10.3.4 Firefox 的 FirePath …… 338
10.3.5 Chrome 开发人员工具 …… 339
10.3.6 IE 开发人员工具 …… 340
10.3.7 Web 性能测试工具介绍 …… 341
第 11 章 Web 性能测试 …… 342
11.1 Web 性能测试技术 …… 342
11.1.1 Web 性能测试术语 …… 342
11.1.2 Web 性能数据的计算方式 …… 344
11.1.3 Web 性能结果分析 …… 344
11.2 Web 性能测试类型 …… 345
11.3 Web 性能测试策略与设计 …… 350
11.3.1 基于风险分析的 Web 性能测试策略 …… 350
11.3.2 Web 测试设计与注意点 …… 351
11.4 Web 性能测试工具 JMeter …… 352
11.4.1 JMeter 介绍 …… 352
11.4.2 JMeter 安装与目录结构 …… 353
11.4.3 运行 JMeter …… 354
11.4.4 JMeter GUI 界面介绍 …… 356
11.4.5 JMeter 常用组件介绍 …… 357
11.4.6 JMeter 的执行顺序 …… 361
11.4.7 JMeter 作用域 …… 362
11.4.8 JMeter 的参数化测试 …… 362
11.4.9 创建 Web 测试计划 …… 363
11.4.10 使用 JMeter 代理录制性能测试脚本 …… 367
11.4.11 使用 Badboy 录制性能测试脚本 …… 369
11.4.12 使用 Chrome 插件录制性能测试脚本 …… 371
参考文献 …… 373
推荐语 …… 374

第 1 章　软件测试必备理论

1.1　软件测试知识

1.1.1　软件测试

IEEE 将软件测试定义为在规定的条件下，使用手工或者自动化手段来运行或测试某个系统，对其是否满足设计需要进行评估的过程。1979 年 Glenford J. Myers 提出软件测试是为了发现错误而执行程序的过程。这个定义紧扣测试的基本活动，即执行程序和寻找错误，符合多数测试人员的工作内容。但这里要特别注意，测试不仅仅是找缺陷，也包括了改进研发流程，预防缺陷，提供对产品质量相关的信心与信息等。实际上当测试人员把工作的重点放在寻找更多的缺陷的时候，很可能会急功近利，因为发现缺陷会占用较多时间，放慢测试脚步，占用测试人员的思维，忽略其他一些基本的功能，很容易导致测试遗漏。

Cem Kaner 教授提出，软件测试是一种技术调查，其目的是向关系人提供有关产品(软件、系统或服务)质量的实验信息。他的定义认为测试是一种服务，服务的内容是提供产品质量的实验信息。质量是对某些人而言的价值，不同的人对质量有不同的评判标准，对信息有不同的需求。除了发现缺陷外，测试人员通过多种方式来向不同的关系人提供信息，不仅能为客户服务，而且能帮助客户避免获得具有较多缺陷的软件，还能向测试经理提供调查报告，向程序员提供代码的质量反馈(一般包含更多技术细节，从而让开发人员修改代码，提高产品质量)，向产品经理提供技术支持(侧重于用户体验和产品价值)，向运维人员提供已知的缺陷和解决方案建议等。Cem Kaner 教授特别提出技术调查，即系统地进行广度调查和深度调查，从各个方面收集信息，帮助项目关系人做出决定。测试人员的职责不是质量保证，测试人员没有权利控制项目计划、预算、开发人力、产品范围、开发模型、客户关系等，但是却能尽力提供有价值的、实时的有关产品真实状态的信息，暴露对产品、项目及业务价值的威胁，即持续向客户报告的是风险而不只是缺陷。

1.1.2　软件缺陷

IEEE 729 - 1983 将缺陷定义为：从产品内部看，缺陷是软件产品开发或维护过程中存在的错误和问题；从产品外部看，缺陷是系统所需要实现的某种功能的失效或违背。简而言之，软件缺陷是指程序、系统中存在问题，不能达成和产品设计书的一致性，不能满足用户的需求。

软件测试人员需要理解相关项目关系人，包括客户、用户、产品经理、开发人员、管理人员等对软件的期望，知道软件通过何种途径提供服务。需要考虑从不同关系人的视角考察软件，来发掘危害软件价值的问题。如果只从单一关系人的角度考察，则很可能会错过对其他人而言很重要的问题。测试人员不能仅仅依赖产品需求规格书，还应该站在用户角度考察产品，了解目标与需求价值，才能真正发现用户需求和软件需求遗漏和错误的问题。

测试过程中，测试人员应该早期参与测试，通过参与需求审查及早发现需求问题，以及通过代码审查等发现代码规范、代码架构优化等问题。例如通过代码审查发现开发人员往往只关注了功能实现，忽略了异常错误处理代码，没有对异常错误情况做处理，所以需要重视这部分的测试。总之，需要尽可能早地发现缺陷，越早发现缺陷，修复越快，成本越低。一般而言，通过相似项目的历史缺陷信息的度量和分析，可以帮助测试人员发现规律，找出问题，提前预防。

1.1.3 软件测试应遵循的七原则

软件测试有七个通用的原则，每一条原则都是宝贵的知识积累，需要关注。

(1) 测试无法显示潜藏的软件缺陷。测试可以显示软件存在缺陷，但不能证明软件没有缺陷。测试会降低软件中存在没有被发现的缺陷的可能性，但是即使没有发现缺陷，也不能证明软件是完全正确的。

(2) 穷尽测试是不可行的。测试新手可能认为，拿到软件后就是要进行完全测试，找出所有缺陷，确保软件完美无缺。但实际上，进行所有(如各种输入、预置条件、输出的组合)测试是不可行的，所以需要基于风险分析和优先级(参见第 2 章测试策略)进行风险的测试而不是穷尽测试。

(3) 软件测试应尽早介入。越早发现缺陷，修改的代价越小；越晚发现缺陷，修改的代价越高，且成倍数增长。为了尽早发现缺陷，在软件或者系统开发生命周期中，测试活动应该尽可能早地介入。

(4) 缺陷集群性(Pareto 原则)。生活中的害虫和缺陷很类似，两者都成群出现，如果发现一个，则附近可能会有一群。软件测试的工作分配比例应与预期的、后期观察的缺陷分布模块相适应，符合 Pareto 原则，即 80 - 20 原则，即 80%的缺陷发生在 20%的模块中。

(5) 杀虫剂悖论。与农药杀虫一样，老用一种农药，害虫就有了抵抗力，农药就再也发挥不了效力。采用相同的测试用例重复进行测试，最后将不能再发现新的缺陷。同理，为了避免杀虫剂悖论，测试用例需要定期审查更新和修改，同时需要不断增加不同的测试用例，来找出更多潜藏的软件缺陷。

(6) 测试依赖于软件测试背景。不同上下文的测试是不同的，例如涉及安全的关键性软件和电子商务网站的测试就不同。可以参考 2.4.2 节的产品特性，设计测试策略。

(7) 不存在缺陷就是有用系统的谬论。如果系统无法使用，或者不能满足用户的需求和期望，那么找到和修改缺陷也是没有帮助的。而且也并非所有软件缺陷都要修复，需要根据风险决定哪些缺陷要修复，哪些不需要修复。需要尽可能多地了解客户的需求，定位产品的特性。

1.2　软件测试方法

本节介绍通用的软件测试方法，工作中建议从 1.3 节介绍的六大质量属性出发，根据实际项目产品特性、风险点，分析和构建适合项目需求的测试类型和测试点。

1.2.1　功能测试法

功能测试(也称为行为测试)以产品的需求规格说明书、设计文档和测试需求列表为依据，测试产品的功能实现是否符合产品的需求规格。需求规格是从最终用户的角度，对程序行为特征进行描述的。功能列表属于常见的功能测试模型，通过输出功能列表的方式，将产品的功能分解为层次结构，既能抽象出主要功能区域，又能提供必要的细节，为功能覆盖提供指导和参考。在应用此方法时，可以参考 3.1.2 节，综合考虑功能列表的多个元素，来测试功能之间的组合和交互。

功能测试重点关注的是对软件产品功能行为的测试，测试时应从实际用户的角度出发，按照功能的使用进行用户场景划分，划分的粒度要明确统一，划分后的测试场景要能覆盖特性功能，各场景之间不存在功能重叠。功能测试一般属于黑盒测试，常用的测试用例设计方法有等价类划分、边界值分析、因果图判定表、正交试验设计、错误推测法等(第 3 章将对这些方法进行详细介绍)。由于不可能测试所有东西，因此可以参考第 2 章的测试策略，根据风险分析对需求进行优先级划分，来决定对测试范围、每个功能投入多少关注等。

1.2.2　性能测试法与流程

性能测试的定义分为广义和狭义两个方面。在狭义上，性能测试主要涉及常规的性能指标，通过模拟生产运行的业务压力或用户使用场景来测试系统性能是否满足性能的要求，一般属于正常范围下的测试。在广义上，性能测试则可以分为压力测试、负载测试、稳定性测试、配置测试等，这在后面的章节中会做具体介绍。

就狭义的定义而言，一般来说，产品的需求规格中会给出性能指标值，测试人员可依此来验证产品是否满足需求规格，需求规格中对性能的要求和定义，会直接影响性能测试的范围，包括测试深度和测试广度。但实际上，很多时候需求规格都很简单，需要挖掘隐式需求。

性能测试的一般流程是：首先，测试出产品的最佳性能值，此时需要注意测试指标之间的内在关系，在测试一个指标时，其余的指标最好保持不变，避免对测试造成影响。其次，分析影响性能的因素，测试其对性能的影响程度。例如，分析哪些因素会导致性能下降，然后分别对其进行测试，观察测试结果是否符合预期，从而得出哪些因素对系统性能的影响大，哪些因素影响小，是否符合预期。另外，还要测试各个因素下性能趋势是否符合预期，其最坏值是否合理，是否会成为系统的性能瓶颈，是否需要调优。一般情况下，测试单个因素对性能的影响时，其余因素应保持不变，而在某些场景下，也需要考虑多个因素组合下的测试。最后，以场景为单位，测试每个场景下的性能，从而更好地评估产品在用户

使用环境中的性能表现，这对用户更有意义。实际测试中，不可能对所有功能都进行性能测试，一般需要基于产品特性，根据风险分析进行慎重的选择。基于风险分析的测试策略会在第 2 章详细介绍。

1.2.3 负载测试法

负载测试是指通过对被测系统不断加压，直到超过预定的指标或部分资源已经达到了一种饱和状态不能再加压为止。它可测试系统在资源超负荷情况下的表现，用来发现设计上的错误或者验证系统的负载能力。其中，负载测试中涉及的资源包括软件配置、硬件配置、路由、TCP/IP 端口、CPU、内存、Flash、带宽、线程、句柄、锁、缓冲器、磁盘等软件正常工作所需要依赖的装置及部件。对负载测试的理解如下：

(1) 负载测试站在用户的角度，观察在一定条件下软件系统的性能表现。测试人员可以通过工具获取测试场景中的资源使用情况，也可以通过分析实际用户的真实系统中的数据来统计负载信息。

(2) 负载测试的预期结果是用户的性能需求得到了满足。此指标一般体现为响应时间、事务处理速率、交易容量、并发容量、资源使用率等。

1.2.4 压力测试法与驱动压力测试案例

压力测试是指当系统已经达到一定的饱和程度(如 CPU、磁盘等已经处于饱和状态)时观察系统是否会出现错误，以此判断系统的业务处理能力。

一般而言，压力测试包括以下三种测试方法：

(1) 持续以超过规格的负载进行测试，查看系统的可恢复性。测试内容一般包括并发性能、疲劳测试、大数据量。

(2) 耐力测试(endurance test)，即以常量负载(constant - load)进行长时间测试。建议使用边界值进行测试，可以发现如内存溢出、文件碎片化、数据查找变慢(随着数据集不断扩大)、系统性能逐渐出现下降甚至服务停止等情况。

(3) 弹性测试，即使用持续突发形态下的负载进行压力测试，保证规格值之内的业务都能被正确处理且系统运行情况正常。

其中，对于第(1)种测试，系统可能无法保证所有业务都能被正确处理，甚至系统有可能崩溃，但即使崩溃，也不能说明存在产品缺陷，因为只要负载超过规格足够多，系统就一定会崩溃。但是系统崩溃后希望能立即主动可恢复，所以持续超过规格用来作为测试可靠性的恢复测试法。第(2)和第(3)种测试在云测试中常常会用到，首先增加负载至超出边界(扩展)，然后将负载降低到边界以下(缩短)。例如负载上限是 100 个并发用户，可以测试 100、99、101 个并发用户，持续测试一天，预期结果是性能对于用户保持一致，无论用户数实际是否超过了边界值。又例如系统最多支持 1000 个用户同时发送邮件，可以设置 60 分钟一个周期，前 30 分钟为 1400 个用户同时发送邮件，后 30 分钟为 600 个用户同时发送邮件，持续测试 1 天。

以驱动类项目为例，表 1 - 1 至表 1 - 4 列出了驱动类项目涉及的压力测试点。读者可以根据产品特性，继续增减测试点。

表 1-1　接口 API 压力测试

压力内容	驱动进行长时间的测试，包括： (1) 驱动最大性能下的长时间测试； (2) 用户数据下的长时间测试； (3) 同驱动多线程下的长时间测试(如 Can 驱动通过多线程实现收发功能，并运行 7 天的长时间测试)； (4) 不同驱动多线程、多进程的长时间测试(单个驱动可通过多线程实现业务功能，接着同时运行不同驱动的进程，如串口、Brightness、DIO 等，进行长时间测试)； (5) 支持最多数目的驱动卡片或设备下，同时进行长时间测试(如 PC 能插入的最多的驱动卡片，每个驱动卡片同时进行长时间测试)； (6) 长时间用户场景的测试，可借助提供给用户的 utility 和 example 进行长时间下的稳定性测试，例如，出现过函数频繁调用 printf 打印信息，而导致无响应的问题
检查点	(1) 驱动功能正常，符合要求； (2) CPU、内存等正确，无内存泄露，可正常开关机
测试工具	(1) 基于单元测试框架，开发各个驱动的自动化测试工具，进行测试； (2) 开发资源监测工具，提供分析
通过标准	运行 7 天，符合驱动相应标准

表 1-2　频繁调用接口 API 测试

压力内容	(1) 频繁调用接口 API，如频繁调用 open/close(通过频繁点击界面调用接口的压力会稍小，建议直接使用 API 调用方式，加压测试) (2) 对于 API 的所有参数的所有可取值进行频繁遍历测试； (3) 多线程并发＋频繁操作
检查点	(1) 返回值正确； (2) get 与 set 值保持一致； (3) 无内存泄露
测试工具	(1) 开发自动化测试工具进行测试； (2) 多线程也可通过源码审查或静态工具检查(例如原子性，利用 Java 的 synchronized、pthread_mutex_lock 进行同步)
通过标准	通过检查点

表 1-3　频繁安装卸载驱动

压力内容	(1) 频繁 insmod/rmmod； (2) 频繁 Install/Uninstall； (3) Windows 的设备管理器中，频繁禁用/使能，例如频繁 Enable/Disable 串口的master 和 slave 后，发现串口驱动的 port 不出现
检查点	(1) 驱动安装、卸载等操作正常，无错误； (2) 驱动运行正常
测试工具	编写自动化脚本进行测试
通过标准	运行 100 次，通过检查点

表 1-4 高负载下的压力测试

压力内容	系统高负载下的测试，根据产品依赖的资源，在占用系统 CPU、内存、磁盘空间等低资源下，进行压力测试： (1) 系统高负载下＋单个驱动的测试； (2) 系统高负载下＋驱动的多线程、多进程争抢资源下的测试； (3) 系统高负载下＋全部驱动＋低资源下的 Burnin 测试
检查点	系统不会崩溃等
测试工具	开发自动化测试程序脚本工具，进行测试
通过标准	默认运行 7 天，无异常

1.2.5 安全性测试法与案例

查找产品的潜在软肋，可以避免黑客利用该软肋对产品进行攻击。安全测试的方法分为白盒、黑盒和灰盒三种。其中，白盒测试需要使用包括源代码在内的所有可用的资源，而黑盒测试只访问软件的输入和观察到的输出结果，介于两者之间的是灰盒测试，它在黑盒测试的基础上可通过对可用二进制文件的逆向工程获得额外的分析信息。

1. 白盒测试法

白盒测试包括各种不同的源代码分析方法，可以人工完成，也可以利用自动化工具完成，这些自动化工具包括编译时检查器、源代码浏览器或自动源代码审核工具。

案例一：字符串复制没有考虑目的缓冲区的大小是否能容纳源缓冲区，导致复制时将目的缓冲区以外的内容进行非法覆盖。由于字符串缓冲区被破坏，导致程序崩溃，如果通过恶意输入触发这种情况，则可造成拒绝服务让合法用户无法正常使用，如果函数返回地址被恶意代码覆盖，则函数返回时控制权便会转移恶意代码，执行任意代码。以 Stack Overrun为例，申请了一段内存，而填入的数据大于这块内存，填入的数据就覆盖掉了这段内存之外的内存了，代码如下：

```
void fun(char * input)
{
    char buf[100];
    strcpy(buf, input);
}
```

没有进行长度检查，如果黑客通过操作参数 input，则可能重写返回地址，从而产生安全性问题。当复制的数据大于 Stack 声明的 Buffer 时，会导致 Buffer 被 Overwritten，从而产生基于 Stack 的 Buffer Overrun。当 Stack 声明的变量位于函数调用者的返回地址时，返回地址会被攻击者重写，此时就可以利用 Buffer Overrun(BO)执行恶意代码进而控制计算机。在 C 语言中，假设调用上面的函数 fun，则程序的 Stack 如图 1-1 所示。首先，参数会放到栈中去，接着是这个函数执行完的下一个指令的地址，即返回地址，接着是 EBP、registers，再接着是这个函数的本地变量的内存空间。上面函数 fun 申请了 100 个字节的空间。当 BO 发生时，数据就会覆盖掉 Buffer 之后的内存，关键的部分是 Return address 可以被覆盖。那么黑客就可以把 Return address 的值修改成这个 Buffer 的一个地址，比如起始地址 buf[0]的地址，而这个 buf 里边填入黑客自己的代码。这样当这个函数退出的时候，