HZ BOOKS
华章教育

经典原版书库

软件测试
原理与实践

[爱尔兰] 斯蒂芬·布朗（Stephen Brown） 乔·蒂莫尼（Joe Timoney） 范氏钗（Thoa Pham） 汤姆·莱萨特（Tom Lysaght） [中国] 叶德仕（Deshi Ye） 著

（英文版·第2版）

Principles and Practice
Second Edition

Stephen Brown Joe Timoney Thoa Pham Tom Lysaght Deshi Ye

机械工业出版社
China Machine Press

经 典

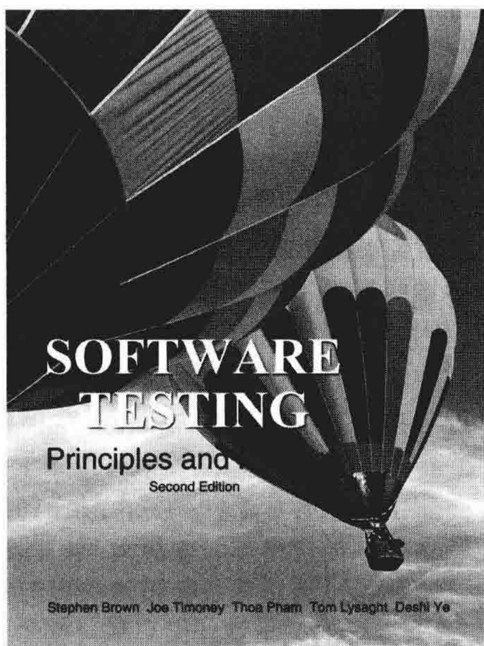
软件测试

原理与实践

（英文版·第2版）

Software Testing

Principles and Practice (Second Edition)



RFID

［爱尔兰］ 斯蒂芬·布朗（Stephen Brown） 乔·蒂莫尼（Joe Timoney）范氏钗（Thoa Pham）汤姆·莱萨特（Tom Lysaght） ［中国］ 叶德仕（Deshi Ye） 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

软件测试：原理与实践（英文版·第2版）/（爱尔兰）斯蒂芬·布朗（Stephen Brown）等著．—北京：机械工业出版社，2019.4（2019.8重印）
（经典原版书库）
书名原文：Software Testing: Principles and Practice, Second Edition

ISBN 978-7-111-62406-6

Ⅰ．软…　Ⅱ．斯…　Ⅲ．软件－测试－英文　Ⅳ．TP311.562

中国版本图书馆CIP数据核字（2019）第059488号

本书版权登记号：图字　01-2019-1168

Authorized English language edition from the Original second edition, entitled Software Testing: Principles and Practice by Stephen Brown, Joe Timoney, Thoa Pham, Tom Lysaght, Deshi Ye.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanic, including photocopying, recording, or by any information storage retrieval system, without permission of by Stephen Brown, Joe Timoney, Thoa Pham, Tom Lysaght, Deshi Ye.

English language edition published by China Machine Press.

Copyright © 2019 by China Machine Press.

This edition is manufactured in the People's Republic of China, and is authorized for sale and distribution Worldwide.

本书英文版由Stephen Brown, Joe Timoney, Thoa Pham, Tom Lysaght, Deshi Ye.授权机械工业出版社在全球独家出版发行。未经Stephen Brown, Joe Timoney, Thoa Pham, Tom Lysaght, Deshi Ye.预先许可，不得以任何方式抄袭、复制或节录本书的任何部分。

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：姚　蕾	责任校对：殷　虹
印　　刷：北京市荣盛彩色印刷有限公司	版　　次：2019年8月第1版第2次印刷
开　　本：170mm×242mm　1/16	印　　张：19.75
书　　号：ISBN 978-7-111-62406-6	定　　价：69.00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）88378991　88379833　　投稿热线：（010）88379604

购书热线：（010）68326294　　读者信箱：hzjsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所　韩光/邹晓东

此为试读，需要完整PDF请访问：www.ertongbook.com

出版者的话

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson、McGraw-Hill、Elsevier、MIT、John Wiley & Sons、Cengage 等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出 Andrew S. Tanenbaum、Bjarne Stroustrup、Brian W. Kernighan、Dennis Ritchie、Jim Gray、Afred V. Aho、John E. Hopcroft、Jeffrey D. Ullman、Abraham Silberschatz、William Stallings、Donald E. Knuth、John L. Hennessy、Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近 500 个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com

电子邮件：hzjsj@hzbook.com

联系电话：（010）88379604

联系地址：北京市西城区百万庄南街 1 号

邮政编码：100037

华章科技图书出版中心

Preface

This book is based on a series of lectures given at Maynooth University and Zhejiang University. It provides a textbook for a number of courses, describing the fundamentals of software testing. The material has been developed over the past ten years, and reflects both the experience of 20 years in industry from one of the authors, and the authors' joint experience in lecturing.

There is no one standard textbook on software testing, and this book is the result of many years of extracting and interpreting test techniques from a wide and varied number of sources. These include testing classics such as *The Art of Software Testing* by Myers, *Software Testing* by Roper, *Testing Object- Oriented Systems* by Binder, and *Introduction to Software Testing* by Ammann and Offutt; standard software engineering textbooks such as *Software Engineering* by Pressman, and *Software Engineering* by Sommerville; software process books such as *Software Testing in the Real World* by Kit and *extreme Programming explained* by Beck; and ISO and IEEE standards related to software quality and testing.

Software testing is a challenging task–it is as important for businesses and government as it is for research institutions. It is still as much an art as a science: there are no accepted standards or norms for applying the various techniques, and interpretation is required. There is no well established research on the effectiveness of the different approaches. The techniques are easy to understand, but generally difficult to apply to real-world software. By providing extensive worked examples, this book aims to provide a solid basis for both understanding, and applying, various test techniques

In the second edition, many of the examples have been updated, and the sections on Integration Testing and System Testing have been expanded. The examples have been updated to work with the latest version of the tools used, and a companion website is provided (`www.softwaretestingbook.org`).

The software tools used in this book are merely representative–they have not been selected as the best example of each, but rather as good examples of the range of capabilities that such tools exhibit. The key goal of the book is that the reader should understand the principles of software testing, and be able to apply them in practice. The book does not endorse or recommend any particular tool. Only a subset of the tool features are covered in this book: the reader should refer to the tool-specific documentation for more details.

前　言

本书的内容基于爱尔兰国立大学梅努斯分校和浙江大学的一系列课程讲稿。该书覆盖了软件测试的基本原理，可以作为许多课程的参考教材。本书的内容历经十年发展，既反映了其一位作者二十年的工业界经验，也融合了多位作者在教学方面的共同经验。

在软件测试领域，目前还没有统一的标准教科书，而本书是通过对各种不同的软件测试技术进行多年的提炼、阐释而形成。这些测试技术的来源包括一些经典的测试书籍如 Myers 的《软件测试的艺术》、Roper 的《软件测试》、Binder 的《面向对象系统测试》、Ammann 和 Offutt 的《软件测试基础》；标准的软件工程教材如 Pressman 的《软件工程：实践者的研究方法》、Sommerville 的《软件工程》；软件过程类书籍如 Kit 的《现实世界中的软件测试》、Beck 的《极限编程》以及软件质量及测试相关的 ISO 和 IEEE 标准。

软件测试是一项具有挑战性的任务，它对科研机构、企业及政府具有同等的重要性。软件测试既是门科学也是门艺术。针对如何应用不同的软件测试技术，目前还没有广泛认同的标准，因此对各种软件测试技术的阐释是必须的。针对各种不同技术的有效性，目前还没有成熟的研究成果。总的来说，各种测试技术容易理解，但如何将其应用到软件产品中是困难的。本书旨在通过广泛丰富的实例，为读者对理解和如何应用各种不同的软件测试技术提供一个坚实的基础。

第 2 版对第 1 版的内容进行了全面更新和修订，新增和扩充了集成测试和系统测试两章。示例已更新，以使用新版本的工具，并提供了一个配套网站（www.softwareetestingbook.org）。

本书中使用的软件工具仅仅是这些工具中的代表——我们选择这些工具并不是因为它们是最佳工具，而是这些工具能够较好地展示我们想要说明的功能。本书的主要目标是让读者了解软件测试的原理，并能在实践中应用。本书不支持或推荐任何特定的工具。书中只介绍了工具的一部分特性：读者应该参考特定工具的详细文档。

Contents

Preface ii

1 Introduction 1
- 1.1 The Software Industry 1
  - 1.1.1 Software Testing and Quality 2
  - 1.1.2 Software Testing and Risk Management 3
- 1.2 Mistakes, Faults and Failures 3
  - 1.2.1 Software Faults 4
  - 1.2.2 Software Failures 5
  - 1.2.3 Need for Testing 6
- 1.3 The Role of Specifications 8
- 1.4 Overview of Testing 9
  - 1.4.1 Testing in the Development Process 9
  - 1.4.2 Test Automation 9
- 1.5 The Theory of Testing 10
  - 1.5.1 Exhaustive Testing Example 11
  - 1.5.2 Implications 11
  - 1.5.3 When To Finish Testing 12
- 1.6 Book Structure 13
  - 1.6.1 Order of Testing 14
  - 1.6.2 Programming Language 14
  - 1.6.3 Level of Detail 14
  - 1.6.4 Code Examples 14
- 1.7 Notes on Terminology 14

2 Principles of Software Testing 15
- 2.1 Static Verification 15
  - 2.1.1 Walk-throughs 16
  - 2.1.2 Code Reviews/Inspections 16
  - 2.1.3 Formal Methods and Testing 18
- 2.2 Dynamic Verification 19
- 2.3 Black-Box and White-Box Testing 19
  - 2.3.1 Errors of "Omission" and "Commission" 21
- 2.4 Test Approaches 22
  - 2.4.1 Black-Box Testing 22
  - 2.4.2 White-Box Testing 23
  - 2.4.3 Fault Insertion 24
- 2.5 Test Design Process 24

2.5.1 Analysis 25
2.5.2 Generating Test Cases 25
2.5.3 Generating Test Data 26
2.5.4 Implementing Tests 26
2.6 Analysis of Software Specifications 27
2.6.1 Parameters 27
2.6.2 Parameter Ranges 27
2.6.3 Equivalence Partitions 28
2.6.4 Boundary Values 29
2.6.5 Combinations of Values 30
2.6.6 Sequences of Values 38
2.7 Analysis of Software Components 40
2.7.1 Control Flow Graphs 40
2.7.2 Decisions and Conditions 43
2.7.3 Paths 44
2.7.4 Data Flows 47
2.8 Analysis of Targets for Fault Insertion 47
2.8.1 Offutt's 5 Sufficient Mutations 47
2.9 Test Artefacts 48

3 Unit Testing **50**
3.1 Techniques 50
3.2 Usage 51
3.3 Black-Box Techniques 52
3.3.1 Equivalence Partitioning (EP) 52
3.3.2 Boundary Value Analysis (BVA) 53
3.3.3 Testing Combinations of Inputs 55
3.3.4 Testing Sequences of Inputs/State-Based Testing 56
3.3.5 Testing with Random Data 56
3.3.6 Error Guessing/Expert Opinion 58
3.4 White-Box Techniques 59
3.4.1 Statement Coverage (SC) 60
3.4.2 Branch Coverage (BC) 61
3.4.3 Condition Coverage (CC) 62
3.4.4 Decision Condition Coverage (DCC) 64
3.4.5 Multiple Condition Coverage (MCC) 65
3.4.6 Modified Condition Decision Coverage (MCDC) 66
3.4.7 Path Coverage 67
3.4.8 Dataflow Coverage (DU Pairs) 68
3.5 Fault Insertion 69
3.5.1 Strong Mutation Testing 69
3.6 Test Ranking 70

4 Unit Testing Examples **71**
4.1 Example One: fits() 71
4.1.1 Description 71
4.1.2 Specification 72
4.1.3 Note on Black-Box Testing 72
4.1.4 Equivalence Partitioning 72

4.1.5 Boundary Value Analysis . . . 77
4.1.6 Combinational Testing . . . 78
4.1.7 Random Testing . . . 81
4.1.8 Testing Sequences of Inputs . . . 82
4.1.9 Elimination of Duplicate Black-Box Tests . . . 82
4.1.10 Important Note on White-Box Testing . . . 83
4.1.11 Source Code . . . 84
4.1.12 Statement Coverage . . . 84
4.1.13 Branch Coverage . . . 85
4.1.14 Condition Coverage . . . 87
4.1.15 Decision/Condition Coverage . . . 88
4.1.16 Multiple Condition Coverage . . . 90
4.1.17 Modified Condition Decision Coverage . . . 90
4.1.18 Path Coverage . . . 92
4.1.19 Dataflow Coverage (DU Pairs) . . . 93
4.1.20 Elimination of Duplicate Tests . . . 95
4.1.21 Test Implementation . . . 96

4.2 Example Two: premium() . . . 97
4.2.1 Description . . . 97
4.2.2 Specification . . . 97
4.2.3 Equivalence Partitioning . . . 97
4.2.4 Boundary Value Analysis . . . 101
4.2.5 Combinational Testing . . . 102
4.2.6 Random Testing . . . 105
4.2.7 Elimination of Duplicate Black-Box Tests . . . 105
4.2.8 Source Code . . . 107
4.2.9 Statement Coverage . . . 107
4.2.10 Branch Coverage . . . 108
4.2.11 Condition Coverage . . . 110
4.2.12 Decision/Condition Coverage . . . 111
4.2.13 Multiple Condition Coverage . . . 113
4.2.14 Modified Condition Decision Coverage . . . 116
4.2.15 Path Coverage . . . 117
4.2.16 Dataflow Coverage (DU Pairs) . . . 119
4.2.17 Elimination of Duplicate Tests . . . 121
4.2.18 Test Implementation . . . 121

5 Unit Testing Object-Oriented Software **123**

5.1 Characteristics of Object-Oriented Software . . . 124
5.2 Effects of OO on Testing . . . 124
5.3 Object-Oriented Testing Models . . . 125
5.3.1 Conventional Models . . . 125
5.3.2 Combinational Models . . . 126
5.3.3 State Machine Models . . . 127
5.3.4 Specification & Design Models . . . 129
5.3.5 Built-In-Test . . . 130
5.4 Example 1 . . . 131
5.4.1 Class CarTax . . . 131

5.4.2 Black-Box Testing in Class Context . . . 131
5.4.3 Equivalence Partitioning . . . 132
5.4.4 Boundary Value Analysis . . . 134
5.4.5 Combinational Testing . . . 134
5.4.6 White-Box Testing in Class Context . . . 134
5.4.7 Combinational Testing at the Class Level . . . 135
5.4.8 State-Machine Testing . . . 137
5.4.9 Specification/Design Testing . . . 141
5.4.10 Built-In Testing . . . 141
5.5 Example 2 . . . 143
5.5.1 Classes Shape and Square . . . 143
5.5.2 Specification/Design Testing . . . 143

6 Integration Testing 145
6.1 Principles . . . 145
6.1.1 Analysis . . . 146
6.1.2 Test Cases . . . 147
6.1.3 Test Data . . . 147
6.1.4 Test Implementation . . . 147
6.2 Example . . . 147
6.2.1 The Premium System . . . 148
6.2.2 Sequence Diagram . . . 149
6.2.3 Source Code . . . 150
6.2.4 Analysis . . . 150
6.2.5 Test Cases . . . 152
6.2.6 Test Data . . . 152
6.2.7 Test Implementation . . . 153
6.2.8 Interpretation of Test Results . . . 153
6.3 External Integration Testing . . . 153
6.4 Unit Testing during Software Integration . . . 154
6.4.1 Drivers and Stubs . . . 154
6.4.2 Top-Down Integration . . . 154
6.4.3 Bottom-Up Integration . . . 155
6.4.4 Incremental Feature/Continuous Integration . . . 155

7 System Testing 156
7.1 Introduction . . . 156
7.1.1 System Test Categories . . . 156
7.2 Usage . . . 158
7.2.1 System Level Functional Testing . . . 158
7.2.2 Test Cases . . . 160
7.3 GUI Testing Example . . . 160
7.3.1 Example System . . . 161
7.3.2 Testing the Interface Behaviour . . . 167
7.3.3 Testing the Interface Navigation . . . 168
7.3.4 Testing Software Features . . . 170
7.3.5 Testing Use Cases and Scenarios . . . 173
7.3.6 Testing a Web-Based GUI . . . 174
7.4 Field Testing and Acceptance Testing . . . 175

此为试读，需要完整PDF请访问：www.ertongbook.com

7.4.1 Field Testing 175
7.4.2 Acceptance Testing 175

8 Software Test Automation **176**
8.1 Unit Test Automation 177
8.1.1 TestNG 177
8.1.2 Basic TestNG Features 177
8.1.3 Other TestNG features 179
8.1.4 Documenting Automated Tests 181
8.1.5 Automated Unit Testing in an IDE 182
8.2 Automated Unit Test Example: fits() 182
8.2.1 Testing fits() with Inline Data 182
8.2.2 Parameterised Tests for fits() 183
8.3 Coverage Measurement Automation 186
8.3.1 EclEmma Example 186
8.3.2 Lazy Evaluation 187
8.4 Automated Object-Oriented Unit Testing 188
8.4.1 CarTax Example 188
8.4.2 Automating Inheritance Testing 188
8.5 Automated System Testing 190
8.5.1 Swing-Based GUI Example 191
8.5.2 Web-Based GUI Example 197
8.6 Selenium IDE (Integrated Development Environment) 205
8.7 Other Test Automation Tools and Techniques 205
8.7.1 Customising the Test Runner 205
8.7.2 Stress Testing 206
8.7.3 Mutation Testing 207

9 Testing in the Software Process **209**
9.1 Test Planning 210
9.2 Software Development Life Cycle 211
9.3 The Waterfall Model 211
9.4 The V-Model 212
9.5 Incremental and Agile Development 213
9.5.1 Incremental Development 214
9.5.2 Extreme Programming 215
9.5.3 SCRUM 217
9.5.4 DevOps 218
9.5.5 Process-Related Quality Standards and Models 219
9.6 Repair-Based Testing 219
9.6.1 Specific Repair Test 219
9.6.2 Generic Repair Test 219
9.6.3 Abstracted Repair Test 220
9.6.4 Example 220
9.6.5 Repair-Based Test Suites 220

10 Advanced Testing Issues **221**
10.1 Philosophy of Testing 221
10.2 Test Technique Selection 222

10.3 Design For Testability (DFT) 222
10.4 Overlapping and Discontinuous Partitions 223
10.4.1 Overlapping Input Partitions 223
10.4.2 Overlapping Output Partitions 223
10.4.3 Inband Error Reporting 224
10.5 Handling Relative Values 225
10.5.1 Classic Triangle Problem 226
10.6 Pairwise Testing 227
10.7 Modified Combinational Testing 228
10.8 Improved Condition Coverage 229
10.9 Testing with Floating Point Numbers 229
10.10 Testing with Complex Data Structures 231
10.11 State-Based or Sequential Testing 231
10.12 Inserting Data Faults 235
10.13 Automated Statement and Branch Testing 235
10.14 Automated Random Testing 235
10.15 OO Dataflow Testing (DU Pairs) 237
10.16 Verifying State in Object-Oriented Testing 239
10.17 Advanced OO Testing 240
10.17.1 Multiplicity in UML Class Diagrams 241
10.18 Text-Based Value Representation 242
10.19 Unit Testing of GUI Components 243
10.20 Some issues in Testing Web-Based Applications 244
10.21 Responsive UI Design Testing 244
10.22 Testing Concurrent and Parallel Software 244
10.22.1 Unit Testing 245
10.22.2 System Testing 245
10.22.3 Static Analysis 245
10.22.4 Tools 245
10.23 Testing Embedded Software 246
10.24 Testing Network Protocol Processing 248
10.24.1 Text-Based Protocols 248
10.24.2 Binary Protocols 249
10.24.3 Protocol Stacks and DFT 249
10.25 Including Testing In The Build Procedure 249
10.26 Research Directions 250

APPENDICES

A Terminology **252**

B Software Testing Tools **254**

C Exercises **255**
C.1 TestNG and Eclipse 255
C.2 Unit Test - Exercise 1 257
C.3 Unit Test - Exercise 2 258
C.4 Unit Test - Exercise 3 259
C.5 Unit Test - Exercise 4 259

C.6 Unit Test - Exercise 5 260
C.7 Unit Test - Exercise 6 261
C.8 Unit Test - Exercise 7 262
C.9 Unit Test - Exercise 8 262
C.10 Unit Test - Exercise 9 263
C.11 Exercise 10 - Test Projects 264

D Source Code **265**

D.1 Chapter 4 - Unit Testing Examples 265
D.1.1 CharterFlight.java 265
D.1.2 CharterFlightTest.java 266
D.1.3 Insurance.java 267
D.1.4 InsuranceTest.java 268
D.2 Chapter 5 - OO Testing 269
D.2.1 CarTax.java 269
D.2.2 CarTaxTest.java 270
D.2.3 CarTaxWithAssertions.java 271
D.2.4 CarTaxWithAssertionsTest.java 273
D.2.5 Shape.java 273
D.2.6 Square.java 274
D.2.7 ShapeTest.java 274
D.2.8 SquareTest.java 274
D.3 Chapter 6 - Integration Testing 275
D.3.1 PremiumCalculator.java 275
D.3.2 Validator.java 276
D.3.3 PremiumCalculatorIntegrationTest.java 276
D.4 Chapter 7 - System Testing 277
D.4.1 FlightBooker.java 277
D.4.2 SimpleSwingTestFramework.java 279
D.4.3 FlightBookerBehaviourTest.java 282
D.4.4 FlightBookerNavTest.java 283
D.4.5 FlightBookerFeatureTest.java 285
D.4.6 FlightBookerScenarioTest.java 286
D.4.7 FlightBookerWebTest.java 288
D.5 Chapter 8 - Software Test Automation 292
D.5.1 Demo.java 292
D.5.2 DemoTest.java 293
D.5.3 DemoTestM.java 293
D.5.4 DemoTestA.java 294
D.5.5 DemoTestB.java 294
D.5.6 Demo2.java 295
D.5.7 Demo2Test.java 295
D.5.8 testng-1.xml 295
D.5.9 CharterFlightTestInline.java 296
D.5.10 CustomRunner.java 296
D.6 Chapter 10 - Advanced Issues 298
D.6.1 Lamp1.java 298
D.6.2 Lamp2.java 298

D.6.3 Lamp1Test.java 298
D.6.4 Lamp2Test.java 299

Select Bibliography **300**

Chapter 1

Introduction

1.1 The Software Industry

The software industry has come a long way since its beginnings in the 1950's. The independent software industry was essentially born in 1969 when IBM announced that they would stop treating their software as a free add-on to its computers, and instead would sell software and hardware as separate products. This opened up the market to external companies that could produce and sell software for IBM machines.

Software products for mass consumption arrived in 1981 with the arrival of PC-based software packages. Another dramatic boost came in the 1990's with the arrival of the World Wide Web, and in the 2000's with mobile devices. In 2010 the Top 500 companies in the global software industry had revenues of $492 billion. The industry is extremely dynamic and continually undergoing rapid change as new innovations appear. Unlike some other industries, for example transportation, it is still in many ways an immature industry. It does not, in general, have a set of quality standards that have been gained through years of hard-won experience.

Numerous examples exist of the results of failures in software quality and the costs it can incur. Famous incidents include the failure of the European Space Agency's Ariane 5 rocket, the Therac-25 radiation therapy machine, and the loss of the Mars Climate observer in 1999. A study by the US Department of Commerce's National Institute of Standards and Technology in 2002 estimated that the cost of faulty software to the US economy was up to $59.5 billion per year.

However, many participants in the industry do apply quality models and measures to the processes through which their software is produced. Software Testing is an important part of the Software Quality assurance process, and is an important discipline within Software Engineering. It has an important role to play throughout the software development lifecycle, whether being used in a Verification and Validation context, or as part of an actual test-driven software development process such as Extreme programming.

Software Engineering as a discipline grew out of the "Software Crisis". The term Software Crisis was first used at the end of the 1960's but it really began to have meaning through the 1970's as the software industry was growing. This reflected the increasing size and complexity of software projects combined with the lack of formal procedures for managing such projects.

This resulted in a number of problems:

- Projects were running over-budget
- Projects were running over-time
- The Software products were of low quality
- The Software products often did not meet their requirements
- Projects were chaotic
- Software maintenance was very difficult

If the software industry was to keep growing, and the use of software was to become more widespread, this situation could not continue. The solution was to be in formalizing the roles and responsibilities of software engineering personnel. These software engineers would plan and document in detail the goals of each software project and how it was to be carried out, they would manage the process via which the software code would be created, and they would ensure that the end result had attributes to show that it was a quality product. This relationship between quality management and software engineering meant that software testing would be integrated into its field of influence. Moreover, the field of software testing was also going to have to change if the industry wanted to get over the "Software Crisis".

While the difference between debugging a program and testing a program was recognized by the 1970's, it was only from this time on that testing began to take a significant role in the production of software. It was to change from being an activity that happened at the end of the product cycle, to check that the product worked, to an activity that takes place throughout each stage of development, catching faults as early as possible. A number of studies comparing the cost of early vs late defect detection have all reached the same conclusion: the earlier the defect is caught, the less the cost of fixing it.

The progressive improvement of software engineering practices has led to a significant improvement in software quality. The short-term benefits of software testing to the business include improving the performance, interoperability and conformance of the software products produced. In the longer term, testing reduces the future costs, and builds customer confidence.

Nowadays, many software development processes have integrated software testing within their activities. And approaches such as "Test Driven Development" (TDD) use testing to lead the code development. In this approach, the tests are developed (often with the assistance of the end-user or customer) before the code is written.

1.1.1 Software Testing and Quality

Proper software testing procedures reduce the risks associated with software development. Modern programs are often very complex, having thousands of lines of code. And they often implement a solution that has been defined in very abstract terms, described as a vague set of requirements lacking in exactness and detail. Quality problems are further compounded by external pressures on developers, from business owners, imposing strict deadlines and budgets in order to reduce both the time to market and associated production costs. These pressures can result in inadequate software testing, leading to reduced quality. Poor quality leads to increased software failures, increased development costs, and increased delays in releasing software. More severe outcomes for a business can be a loss of reputation, leading to reduced market share, or even to legal claims.

International Standard ISO 9126[1] *Software Engineering–Product Quality* is structured around six main attributes. These are given with their characteristics in Table 1.1.

Table 1.1: Software Quality Attributes ISO 9126

Attribute	**Characteristics**
Functionality	Suitability, accurateness, interoperability, compliance, security
Reliability	Maturity, fault tolerance, recoverability
Usability	Understandability, learnability, operability
Efficiency	Time behavior, resource behavior
Maintainability	Analysability, changeability, stability, testability
Portability	Adaptability, installability, conformance, replaceability

Attributes that can be measured objectively, such as performance and functionality, are easier to test than those which require a subjective opinion, such as learnability and installability.

1.1.2 Software Testing and Risk Management

Software testing can be viewed as a risk-management activity. The more resources that are spent on testing, the lower the probability of a failure, but the higher the testing cost. One factor in risk management is to compare the expected cost of failure against the cost of testing. The expected cost is estimated as follows:

$$expected\text{-}cost = (risk\text{-}of\text{-}failure) * (cost\text{-}of\text{-}failure)$$

For a business there are short and long term costs associated with failure. The short term costs are primarily related to fixing the problem, but may also be from lost revenue if product release is delayed. The long term costs are primarily the costs of losing reputation, and associated sales.

The cost of testing needs to be in proportion to both income and the cost of failure (with the current state of the art, it is arguable that all software is subject to failure at some stage). The effectiveness of testing can generally be increased by the testing team being involved early in the process. Direct involvement of customers/users is also an effective strategy. The expected cost of failure is controlled through reducing the probability of failure–through sound engineering development practices, and quality-assurance (testing is part of the quality assurance process). Software testing can be addressed as an optimisation process: getting the best return for the investment.

1.2 Mistakes, Faults and Failures

Leaving aside the broader relationship between software testing and the attributes of quality for the moment, the most common application of testing is to search for defects present in a piece of software and/or verify that particular defects are not present in that

[1]Recently updated in the ISO 25000 family of standards

此为试读，需要完整PDF请访问： www.ertongbook.com