HZ BOOKS
软件工程技术丛书

测试系列

软件测试过程改进

Software Testing in the Real World

Improving the Process

Edward Kit 著　　李新华　陈丽容　马立群　等译

机械工业出版社 China Machine Press
中信出版社 CITIC PUBLISHING HOUSE

Software Testing in the Real World
Improving the Process

软件测试过程改进

"我确实非常喜欢这本书，如果让我写一本测试方面的书，恐怕也很难有所超越。本书对测试过程的阐述尤其出色。"

——Greg Daich，Science Applications International公司高级软件工程师，"软件技术支持中心测试组"成员

"该书易读易懂，适合对软件测试改进感兴趣的人阅读……该书一定能发挥重要作用，帮助我们不断做出改进……我建议大家不妨一'试'。"

——Bill Hetzel，Software Quality Engineering公司总裁

"该书从实用的角度入手，融合了测试领域的最新进展。"

——Barry Boehm，"南加州大学软件工程中心"主任

本书为改进软件测试过程提供了一套行之有效的方法，内含许多测试技巧和指南，适合于组织内部制定策略，使测试过程不断得到改进，对于规模不同，过程成熟度各异的组织都适用。

本书就当今测试领域的方法、工具、技术和组织等问题进行了探讨。本书注重实践，针对大多数组织内部软件工程成熟度较低这一问题，在成本、风险、标准、计划、测试任务和测试工具等方面提供了指导。

测试和质量保证专家、开发和项目管理人员以及开发人员，将从这些测试改进的实用技巧以及具体的软件测试工具信息中获益匪浅。

作者简介：

Edward Kit 是一位国际软件咨询专家，有20多年软件工程方面的经验。他是Software Development Technologies公司的总裁和创始人。他的客户包括微软、惠普、Sun、苹果、诺基亚等各大公司。他在美国、日本、印度、澳大利亚和欧洲提供过咨询和教育服务，曾在贝尔实验室和Tandem Computers公司工作过。

封面设计 陈子平

ISBN 7-111-11635-6

9 787111 116356

HZ BOOKS

华章图书

网上购书：www.china-pub.com

北京市西城区百万庄南街1号 100037

读者服务热线：(010)68995259，68995264

读者服务信箱：hzedu@hzbook.com

ISBN 7-111-11635-6/TP · 2801

定价：29.00 元

软 件 工 程 技 术 丛 书

测试系列

Edward Kit 著

李新华 陈丽容 马立群 等译

软件测试过程改进

Software Testing in the Real World

Improving the Process

机械工业出版社 China Machine Press

中信出版社 CITIC PUBLISHING HOUSE

本书系统全面地介绍了软件测试方法，为改进软件测试过程提供了一套行之有效的方法。内容包括软件测试成熟度、改进测试过程的基本框架、验证及确认测试、测试工具、测试管理技术等。附录中给出了软件工程和测试的相关标准、验证及确认测试审查清单示例、测试工具选择等。书中包含了许多测试技巧和指南，帮助组织内部制定战略，改进测试过程。书中所讨论的技术对于规模不同、过程成熟度各异的组织都适用。

本书结构清晰、内容丰富，适于软件测试人员、产品开发人员、管理人员、质量保证人员、系统分析人员、工具开发人员以及其他与软件测试有关的工作人员阅读，也可作为高等院校计算机专业师生的参考书。

图书在版编目（CIP）数据

软件测试过程改进/基特（Kit, E.）著；李新华等译. -北京：机械工业出版社，2003.3
（软件工程技术丛书-测试系列）
书名原文：Software Testing in the Real World: Improving the Process
ISBN 7-111-11635-6

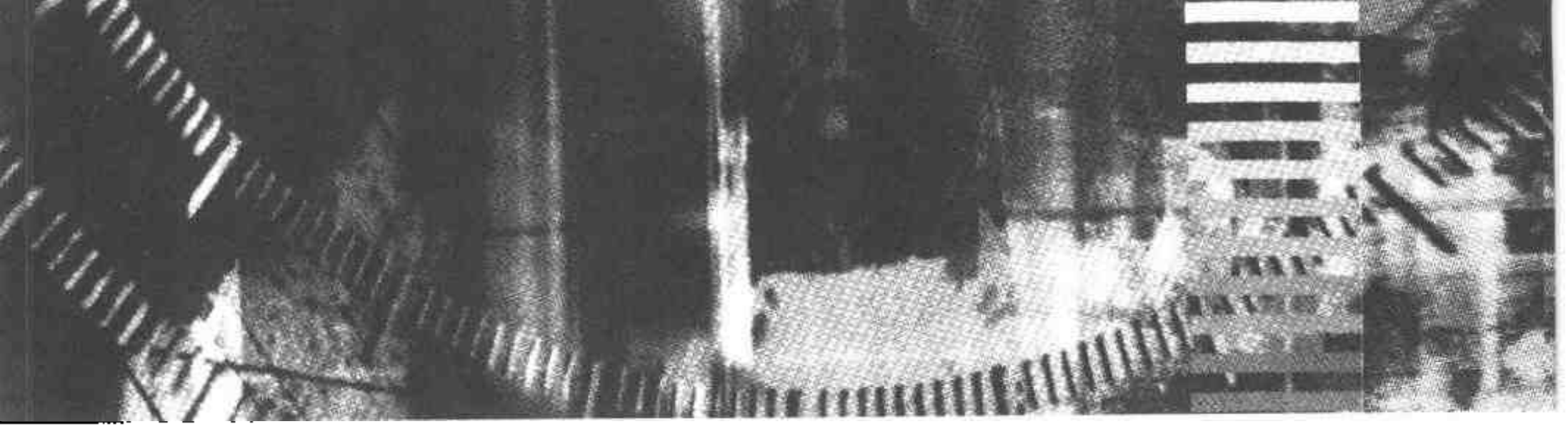
Ⅰ. 软… Ⅱ. ①基…②李… Ⅲ. 软件-测试 Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字（2003）第 009748 号

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：邱李华 刘立卿

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2003 年 4 月第 1 版第 1 次印刷
787mm×1092mm 1/16 · 13.25 印张
印数：0 001-5 000 册
定价：29.00 元



序

多年来，我有幸与多位测试领域的著名专家共事过，而 Ed Kit 更是其中的佼佼者。他是一位软件专家、测试专家与质量管理专家，也是一位优秀的研讨会组织者和咨询专家。他在许多实践方面对测试技术的发展做出了卓越的贡献。我本人与 Ed Kit 相处、共事十载，可谓受益匪浅。本书是 Ed Kit 对测试领域的又一重要贡献，使测试界的同仁可以进一步分享他的真知灼见。

本书汇集了 Ed Kit 从事测试工作的亲身经验，强调测试的过程和改进办法。书中既有实践经验，也有不少好的窍门，当你阅读本书时，可以从实际应用深入到基本思想，快速浏览潜在的改进过程（从第 3 章“白纸方法”开始），书中提供了许多技巧和练习，并且包含了八个附录，附录中包括了与测试相关的标准概要以及测试资源和测试工具的参考。多数读者还将发现，本书包括了许多易于理解和应用的实用思想。

第二部分主要是为新手或在初创的组织机构内工作的人员设计的。其中的章节对测试的内涵进行了阐述并对过程的改进提供了框架。尽管不少内容看起来是很平常的“常识”，但真正要将其付诸实施却一点都不平常。本书非常有助于从实用的角度对软件测试做出改进。

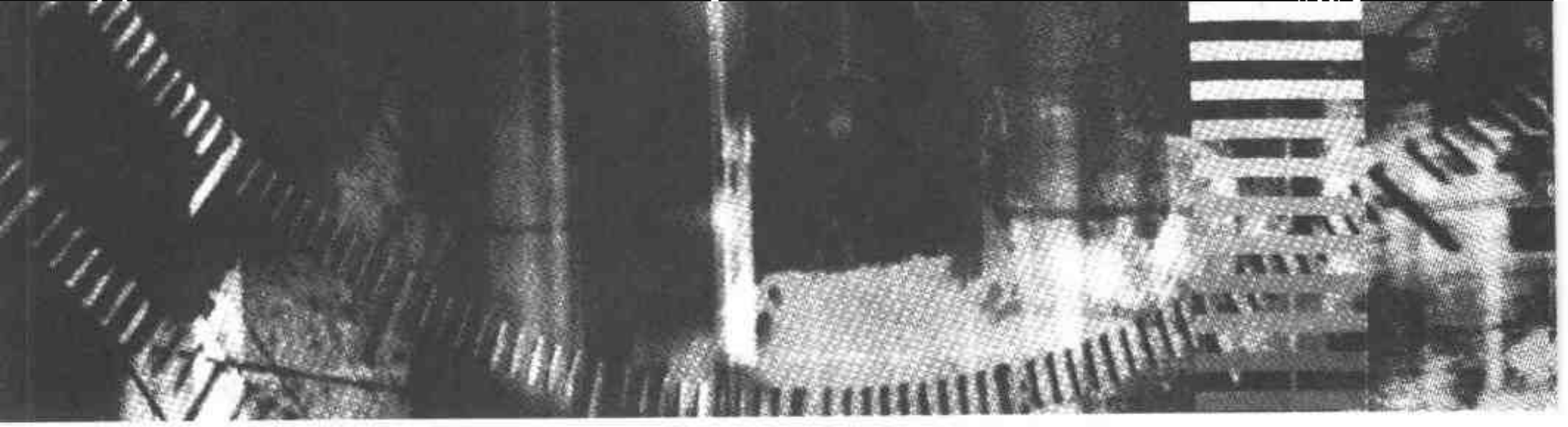
本书简明易懂，非常适合那些想对软件测试做出改进的人员阅读。

许多读者，尤其是业内的开发人员、测试人员和管理人员应该为本书的出版感到高兴，不妨“测试”一下本书。不少读者也许读过出版已达十年之久的拙作《*The Complete Guide to Software Testing*》。本书在许多方面实际是对拙作的更新，不少方面我一直也想做出修改，只可惜时间有限、力不从心。我还想了解各位读者的兴趣所在以及成功的测试经验。如能分享各位的测试结果，当不胜感谢。

Bill Hetzel[⊖]

软件质量工程公司总裁

⊖ Bill Hetzel 博士是软件测试学科的先驱，1972 年世界首次软件测试会议的召集人。



前　言

软件测试。
它既令人兴奋，也令人烦恼。
既令人羡慕不已，也令人望而却步。
要想在预算内按时交付高质量的软件，测试是必不可少的。

本书是专为测试过程的改进而撰写的，面向测试专家、产品开发人员、管理人员、质量保证专家、维护人员、项目管理人员、审计人员、系统分析人员、工具开发人员以及其他与软件测试有关的工作人员。通过阅读本书可以进一步了解测试工作，从而使开发人员和非测试领域的专业工作者进一步提高专业水平。

为了了解测试并对测试工作有所改进，必须从广义的角度来观察测试过程，即应该将人员、方法、工具、测试和领导艺术集成到测试软件产品的过程中。本人从事测试实践、教学和咨询工作 20 余年，发现人们为了改进测试过程反复提到以下一些关键性问题：

方法问题

- 哪些是测试的主要方法，我们什么时候用到这些方法，怎样实施这些方法？
- 怎样避免缺陷转移并提高发现缺陷的能力？
- 怎样理解并利用有关的测试标准和术语？
- 怎样生成有意义的测试计划及其他重要的测试文件？
- 怎样发现并确定软件测试过程中需要改进的地方？

领导艺术问题

- 怎样组织软件测试工作，从而使其可以度量、可以控制？
- 从何处可以寻求有关软件测试的帮助？
- 怎样确保在中、长期内改进工作的可支持性？
- 怎样组织测试工作才能高效地协调人们之间的相互关系？
- 怎样扬长避短？

工具和技术问题

- 使测试过程自动化的最重要的时机是什么？
- 怎样利用度量手段了解并控制软件测试过程？
- 怎样挑选测试工具和销售商并有效地利用测试工具？
- 怎样将风险分析技术用做测试决策的基础？

根据我的经验，在方法、领导艺术和技术方面采取一种平衡的策略是改进测试最有效的方法。

大多数的软件开发组织由于软件过程的不成熟而蒙受损失。因此，专业工作者和管理人员通常会问："我们应该怎么办呢？"。本书为有效地改进软件测试过程提供了一个工具箱。既然是工具箱，就不是简单地从 A 到 Z 教人们如何去做的某个方法，而是一系列可供选择的技术，可以单独使用也可综合采纳，帮助人们制订并实现各类改进目标。

第一部分是用于明确方向的。这一部分介绍了软件测试中的六个基本问题，软件测试的历史以及改进测试过程的一个简单、实用的方法。从一般到具体，对软件工程和测试这些在新生事物不断涌现的时代应运而生的新兴学科进行了阐述。

第二部分讨论根据实际情况确定具体目标，并说明这些目标与各种正式的与不太正式的测试定义、测试目的关系，以及与对于专业工作者的日常工作来说非常重要的各类现行标准之间的关系。

介绍了测试过程的基本形式，阐明了成本和风险方面的一些重要的现实问题，这些问题涉及到决策过程中的测试对象、时间和方法。这一部分还对验证和确认这些测试活动所涉及的有关方面进行了探讨，从而为进一步深入地探讨验证和确认做好了准备。对规划制订、软件工程成熟度目标、配置管理、标准和工具等问题进行了阐述和定义。

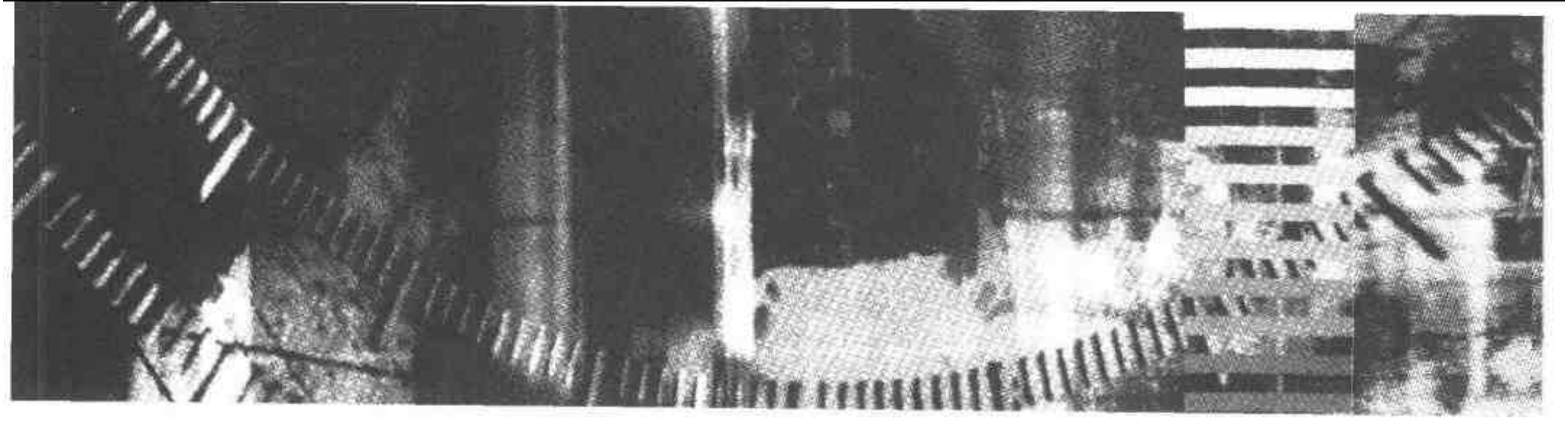
第三部分介绍基本的验证和确认任务，包括对测试成本的规划与控制，对资源的充分利用等。阐述了如何发挥工具在各种测试活动中的作用，说明了工具的分类，并就如何获得工具提供了指南。探讨了如何合理并建设性地利用度量，并指出了应如何正确地利用度量结果。

第四部分介绍涉及测试工作的组织和管理问题，这是测试成功与否的决定因素。概括了当今一些公司的最优方法，然后集中讨论了如何在中、短期内不断完善测试工作。

本书的正文尽量包括许多定义、范例和参考文献。附录包含了各种补充材料（由于太多，无法收集在本书的正文内）。这些材料有范例清单、练习、计划制订文件和范例工作文件。还收集了正式标准、规范及其他一些有用的信息源，如期刊、工具、会议、时事通讯、用户群和加注的参考书目。

Ed Kit

1995 年 9 月于加州圣何塞



目 录

序

前言

第一部分 软件测试过程成熟度

第 1 章 软件测试的六个要点 …… 3

1.1 要点 1：测试过程的质量决定测试工作的成败 …… 3

1.2 要点 2：使用早期软件生存周期测试技术可避免缺陷转移 …… 3

1.3 要点 3：软件测试工具的时代已经到来 …… 4

1.4 要点 4：改进测试过程必须有专人负责 …… 4

1.5 要点 5：测试是一个专业技术学科，要求富有经验的专门技术人员 …… 4

1.6 要点 6：培养破旧立新的、积极的合作精神 …… 5

第 2 章 技术与实践 …… 6

2.1 一门新兴学科的短暂而丰富的历史 …… 6

2.2 我们的现状如何 …… 8

2.3 测试应该如何定位 …… 8

2.4 参考文献 …… 9

第 3 章 白纸方法 …… 10

第二部分 测试过程改进框架

第 4 章 树立注重实际的观点 …… 15

4.1 我们的目的是什么 …… 15

4.2 关于错误你知道多少 …… 15

4.2.1 什么和为什么 …… 16

4.2.2 错误在哪里 …… 16

4.2.3 错误的成本 …… 16

4.3 关于测试的一些定义 …… 17

4.4 优秀测试人员的测试态度 …… 18

4.4.1 测试者寻找错误 …… 19

4.4.2 测试是破旧立新 …… 19

4.4.3 测试人员找出错误，对事不对人 …… 19

4.4.4 测试提高产品价值 …… 20

4.5 测试人员怎样工作 …… 20

4.6 现在能做什么 …… 20

4.7 参考文献 …… 21

第 5 章 重要选择：测试什么、何时测试、怎样测试 …… 22

5.1 风险及风险管理 …… 22

5.2 尽早开始测试 …… 24

5.3 测试过程的基本形式：验证和确认 …… 25

5.4 测试、开发生存周期及合同 …… 25

5.5 有效测试 …… 27

5.6 测试的效益 …… 27

5.7 现在能做什么 …… 28

5.8 参考文献 …… 28

第 6 章 重要方法：测试的框架 …… 29

6.1 计划 …… 29
6.1.1 验证计划中要考虑的因素 …… 30
6.1.2 确认计划中要考虑的因素 …… 30
6.2 软件工程成熟度和 SEI …… 30
6.2.1 SEI 过程成熟度等级 …… 31
6.2.2 过程成熟度是怎样影响测试的 … 32
6.3 配置管理 …… 33
6.3.1 什么是配置管理 …… 34
6.3.2 在 CM 方面测试所关心的问题 … 34
6.4 标准 …… 35
6.4.1 IEEE/ANSI 标准 …… 36
6.4.2 ISO 9000、SPICE 及其他标准 … 37
6.5 正式文档 …… 38
6.6 测试件 …… 38
6.7 度量 …… 39
6.8 工具 …… 40
6.9 现在能做什么 …… 41
6.10 参考文献 …… 41

第三部分 测 试 方 法

第 7 章 验证测试 …… 45
7.1 验证的基本方法 …… 45
7.1.1 验证的组织形式 …… 46
7.1.2 审查：关键元素和阶段 …… 47
7.1.3 走查 …… 47
7.1.4 走查：关键元素 …… 47
7.1.5 伙伴检查 …… 48
7.2 发挥验证的作用 …… 48
7.2.1 验证要做什么 …… 48
7.2.2 审查单：验证工具 …… 49
7.3 验证不同阶段的文档 …… 50
7.3.1 验证需求 …… 50
7.3.2 好的需求规格说明的特性 …… 50
7.3.3 验证功能设计 …… 52
7.3.4 验证内部设计 …… 53
7.3.5 验证代码 …… 54
7.4 从验证中获取最大收益 …… 55
7.4.1 作者 …… 55
7.4.2 开发小组 …… 56
7.4.3 审查小组 …… 56
7.4.4 高收益的验证 …… 56
7.5 验证的三个成功因素 …… 57
7.5.1 成功因素 1：过程责任人 …… 57
7.5.2 成功因素 2：管理部门的支持 …… 57
7.5.3 成功因素 3：培训 …… 57
7.6 建议 …… 57
7.7 参考文献 …… 58
第 8 章 确认测试 …… 59
8.1 确认概述 …… 59
8.1.1 覆盖 …… 60
8.1.2 基本测试策略 …… 61
8.1.3 确认任务与测试覆盖 …… 61
8.1.4 测试基础 …… 61
8.1.5 确认策略 …… 62
8.2 确认方法 …… 63
8.2.1 基于功能测试的黑盒方法 …… 63
8.2.2 基于内部测试的白盒方法 …… 68
8.3 确认活动 …… 71
8.3.1 低层测试 …… 72
8.3.2 高层测试 …… 73
8.3.3 再测试 …… 80
8.3.4 累进测试和回归测试 …… 80
8.3.5 测试执行的筹划 …… 81
8.3.6 测试点 …… 82
8.4 确认测试的推荐策略 …… 82
8.5 参考文献 …… 83
第 9 章 控制确认成本 …… 84
9.1 使测试实施成本最小化 …… 84
9.1.1 预运行启动成本 …… 84
9.1.2 执行成本 …… 85
9.1.3 后运行成本 …… 85
9.1.4 降低测试实施成本的建议 …… 85
9.2 降低测试的维护成本 …… 86
9.3 降低确认测试件开发成本 …… 87

9.4 测试件库 …… 88
9.5 建议 …… 89
第 10 章 测试任务、可交付文件及其在生存周期中的对应阶段 …… 90
10.1 主测试计划 …… 90
10.2 验证测试任务和可交付文件 …… 92
10.2.1 制定验证测试计划 …… 92
10.2.2 验证执行 …… 93
10.3 确认测试任务和可交付文件 …… 94
10.3.1 制定确认测试计划 …… 95
10.3.2 测试结构设计 …… 96
10.3.3 测试件开发——设计和实施细节 …… 97
10.3.4 测试执行 …… 100
10.3.5 测试评估 …… 102
10.3.6 何时停止 …… 102
10.4 用户手册 …… 104
10.5 产品发布标准 …… 104
10.6 IEEE/ANSI 测试文件概述 …… 105
10.6.1 测试计划和规格说明的文件结构 …… 105
10.6.2 测试报告的文件结构 …… 106
10.7 把任务和可交付文件对应到生存周期 …… 106
10.7.1 概念阶段 …… 106
10.7.2 需求阶段 …… 107
10.7.3 功能设计阶段 …… 107
10.7.4 内部设计阶段 …… 107
10.7.5 编码阶段 …… 108
10.7.6 集成和测试阶段 …… 108
10.7.7 运行/维护阶段 …… 109
10.8 参考文献 …… 109
第 11 章 软件测试工具 …… 110
11.1 测试工具的分类 …… 110
11.1.1 评审与审查工具 …… 111
11.1.2 制定测试计划的工具 …… 111
11.1.3 测试设计和开发工具 …… 112
11.1.4 测试执行和评估工具 …… 113
11.1.5 软件测试支持工具 …… 115
11.2 工具采购 …… 115
11.3 参考文献 …… 116
第 12 章 度量 …… 117
12.1 通过度量获得答案 …… 117
12.2 有用的度量 …… 118
12.2.1 度量复杂度 …… 118
12.2.2 度量验证效率 …… 119
12.2.3 度量测试覆盖 …… 119
12.2.4 度量/跟踪测试执行状态 …… 119
12.2.5 度量/跟踪事故报告 …… 119
12.2.6 基于事故报告的测试度量 …… 120
12.3 其他的相关度量 …… 120
12.4 建议 …… 121
12.5 参考文献 …… 121

第四部分 测试管理技术

第 13 章 测试的组织方法 …… 125
13.1 测试的组织和改组 …… 125
13.2 结构设计因素 …… 127
13.3 测试功能的组织方法 …… 127
13.3.1 方案 1：测试是各人的责任 … 127
13.3.2 方案 2：测试是各小组的责任 …… 128
13.3.3 方案 3：用专用资源进行测试 …… 129
13.3.4 方案 4：将测试组织置于 QA 中 …… 130
13.3.5 方案 5：隶属于开发的测试组织 …… 130
13.3.6 方案 6：集中管理的测试组织 …… 131
13.3.7 方案 7：没有测试技术中心的集中式测试组织 …… 132
13.4 选择正确的方案 …… 133
13.5 参考文献 …… 133

第 14 章 目前的做法、发展趋势和挑战 …… 135
14.1 图形用户界面：有哪些新东西 …… 135
14.2 应用测试 …… 135
14.3 测试人员与开发人员的比例 …… 136
14.4 软件度量和实践基准研究 …… 136
14.5 参考文献 …… 138
第 15 章 获得可持续收益 …… 139
15.1 实现收益 …… 139
15.2 获得帮助 …… 139
15.2.1 软件测试书籍和业务通讯 …… 140
15.2.2 咨询和培训服务 …… 140
15.2.3 软件测试研讨会 …… 140
15.3 后续工作 …… 141
15.4 参考文献 …… 141

第五部分 附 录

附录 A 软件工程和测试标准 …… 145
附录 B 验证审查单 …… 150
附录 C 验证练习 …… 167
附录 D 确认练习（答案） …… 181
附录 E 参考书目（包括软件测试工具一览表） …… 184
附录 F 信息资源：会议、期刊、通讯、DOD 规范 …… 188
附录 G 专用工具和工具选择 …… 191
附录 H 改进实施示范清单 …… 194

第一部分

软件测试过程成熟度

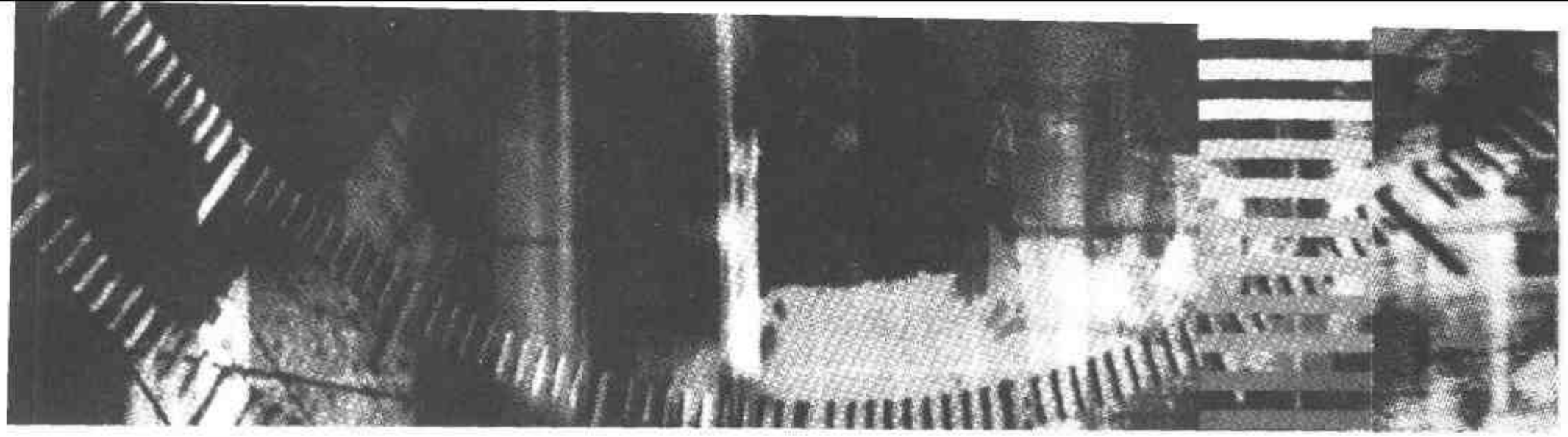
“追求永无止境”

——H.James Harrington

第 1 章 软件测试的六个要点

第 2 章 技术与实践

第 3 章 白纸方法



第1章

软件测试的六个要点

本章讲述设计新的或更新旧的软件测试过程的基础。利用软件测试过程，可集成人员、方法、度量、工具和设备对软件产品进行测试。作为本书基础的软件测试过程的要点如下：

（1）测试过程的质量决定测试工作的成败。

（2）使用早期软件生存期测试技术可避免缺陷转移。

（3）软件测试工具的时代已经到来。

（4）改进测试过程必须有专人负责。

（5）测试是一个专业技术学科，要求富有经验的专门技术人员。

（6）培养破旧立新的、积极的合作精神。

1.1 要点 1：测试过程的质量决定测试工作的成败

从根本上说，软件开发过程的质量决定了软件系统的质量，同样的，测试过程的质量决定软件测试的质量和有效性。

测试工作有它自身的周期。测试过程从产品的需求阶段开始，此后，与整个开发过程并行开展，换句话说，伴随着开发过程的每一个阶段，都有一个重要的测试活动。

在一个开发过程不成熟的组织中工作的测试小组将比其他的小组忍受更多的痛苦，但是无论开发机构的成熟状态如何，测试小组能够而且也应该致力于改进小组自身的内部过程。不成熟的测试组织中不成熟的测试过程将导致没有收益的、混乱的、令人沮丧的环境，生产出质量低下、不令人满意的产品。对不成熟测试组织中的测试过程进行富有成效的改进，将促进整个组织开发过程的改进。

1.2 要点 2：使用早期软件生存周期测试技术可避免缺陷转移

有一半以上的错误是在需求阶段引入的。如果错误在它被引入的阶段即被发现，并且对程序进行有效的测试，避免错误从任何一个开发阶段转移到它的后继阶段，这样，错误成本将降到最小。

尽管许多人意识到了这一点，但在实际中，我们常常缺乏手段，无法在错误被引入的阶段立即发现它，而是拖到很后面，常常要拖到功能和系统测试阶段，此时我们已经进入了“混乱带（The Chaos Zone）”。如果我们不能利用那些已得到证明了的可应用到早期开发过程中的测试技术，那么，很可能就失去了提高测试效率的最佳时机。

例如，我们应该学会对诸如需求一类的关键文件进行评审。即使是一个不成熟的组织也能够进行有效的审查，在至少二十年的历史中，审查被不断地证明是一个成功者，成本/收益比一次次地证明（文献已证明）了这一点。

1.3 要点3：软件测试工具的时代已经到来

经过多年的观察、评估、甚至是等待，现在我们可以说测试工具的时代已经到来。有许许多多、不同种类的工具可供选择，而且他们中间有许多是成熟的、强健的产品。

例如，利用捕获/回放工具，可以完成一天二十四小时的无人参与的测试运行，缩短测试周期，实现测试自动化。对于每一个主要平台都有一类工具：有些工具需要人工干预的更多一些；有些工具更适合客户/服务器类型；有些工具在技术上是精雕细琢的；另一些则是比较简单的。

另一类基本工具是结构覆盖工具，用于确定软件是否已被充分测试，这种工具可特别指出产品的哪部分在测试中已被实际执行到。指望用户成为代码的第一位执行者的想法已经不再能接受了。

掌握获得工具的策略，并采用一个适当的步骤选择工具无疑是很重要的。然而这种步骤往往是基于某些常识，并不需要系统地去做。在获得工具方面，听从独立专家的忠告可能是非常有好处的。

1.4 要点4：改进测试过程必须有专人负责

当测试小组感到一筹莫展时，就应着手在几个关键地方进行改进了，譬如完善规格说明，强化评审和审查。

管理人员应该指定一名设计师或一个核心小组优先考虑潜在的改进问题，领导测试改进工作，而且必须明确支持他们的工作。虽然这不是什么特别的尖端科学，但却费时耗力。假如我们能在包括有效的测试计划和设计在内的整个测试过程中使用工具，肯定可以获得巨大的帮助。

说到底，软件测试是一个过程，需要有人负责该过程的改进工作，必须有专人计划和管理测试过程的改进进程。

1.5 要点5：测试是一个专业技术学科，要求富有经验的专门技术人员

软件测试已经成为一种专业——一种职业的选择，是一个可留下足迹的领域。软件测试过程发展十分迅速，形成了一个需要专业人员的学科。今天，任何组织要在软件测试领域获得成功，它必须拥有足够的具有丰富软件测试经验的行家，而且，他们应能从管理者那里得到大力支持。

测试不是一个入门级的工作，也不是其他工作的垫脚石。很多人发现，一旦测试工作真正开展起来，它具有的挑战性将超过产品开发。它应该被组织起来，切实遵守用户至上的宗旨，而不是处于辅助的，容易被产品开发部门或其他组织所左右的地位。

测试应该是独立的，公平的，有组织的，公平分享对产品质量所做贡献的认可和回报。

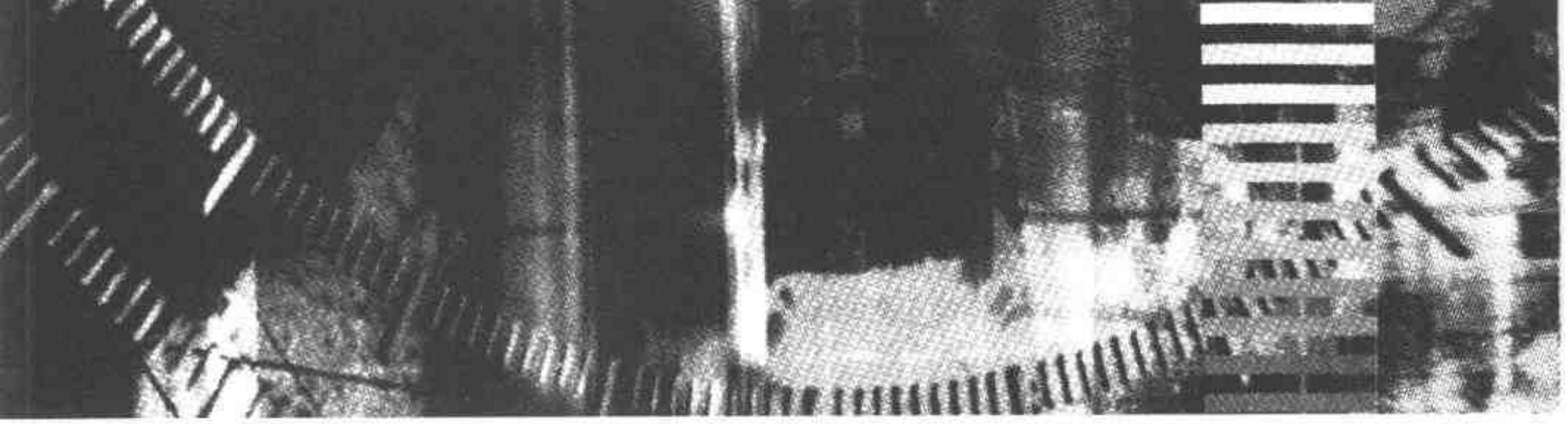
1.6 要点6：培养破旧立新的、积极的合作精神

测试要求专门的训练，要求富有创造力。优秀的测试，也就是目前设计和执行成功的测试——发现产品中的缺陷的测试——要求有真正的灵活性，但同时也可以看成是破坏。实际上，以可控的、系统的方式破坏某些东西需要相当大的创造力。优秀的测试者是井井有条地将产品拆开，找出它的弱点，促使它发挥或超水平发挥其性能。

我们的产品能否正常工作？何时会发生故障？我们知道错误总是存在，故障也是难免的，但是，故障在合理的范围内吗？如果考虑产品的临界情况，故障是可以接受的吗？我们将各种重要的可能性都考虑在内了吗？以上这些信息都应该由测试者提供，只有当我们对诸如此类的问题进行了令人满意的探讨后，才能将产品交付用户。

树立正确的“破旧立新”的思想，对于测试成功会产生意义深远的影响。如果我们的目的是指出产品做了不应该做的，没有做应该做的，那么，这样的测试就踏上了成功之路。尽管现在所做的离规范的测试标准还相去甚远，但我们的专业工作人员和管理人员在进行“破旧立新”的测试工作中，所取得的成绩确实是令人惊奇的。

成功的测试要求有系统的方法，要求我们特别关注下面几个关键因素：计划、项目和过程控制、风险管理、审查、度量、工具、组织以及专业精神。在整个开发过程中，测试人员为保障产品质量发挥了积极作用，做出了重要贡献，这点是我们要牢记在心的。



第2章

技术与实践

几个世纪以来，人们在通过各种复杂方法认识和控制客观世界的过程中，始终伴随着痛苦的尝试和惨痛的失败。最终产生了基本的技能、技术指导和最佳实践，并成为特殊领域里专业技术人员的第二本能。

在人类各学科的漫长演化过程中，软件工程还处于萌芽状态。当我们意识到这点时，记住软件开发学科有多么年轻就显得十分重要。直到六十年代后期，“软件工程”一词才开始有了运用基础法则进行系统研究的含义。

此外，技术与实践之间还存在着鸿沟。例如，软件测试已成为大学里软件工程课程的基本成分，工业部门的技术培训正在开展，但这种培训还常常是随意的。已发表的文献（那些真正需要的人不一定能看到）提出了一些前沿的方法，用以更好地开发软件，尤其是达到所期望的软件质量，但是有很多先进方法停留在研究阶段，不能在实际领域里得到检验而有许多研究忽略了现实问题，例如投资的回报。

即使那些得到了很好的检验的方法也没有被工业部门广泛应用，软件系统的开发仍然过分昂贵，充满了严重错误，而且常常滞后，产品维护的费用远高于开发费用。

与此同时，我们的系统变得日益复杂，50%以上的开发工作落在了测试身上。那些负责测试的人员还要为搭建一个切实可行的工作平台，配上适用的测试工具付出极大的努力，为争取一个独立的地位，一定的人力物力资源保证而不懈奋斗。

2.1 一门新兴学科的短暂而丰富的历史

提到“软件工程”一词，总是具有通过努力实现可靠性、功能性、可预测性、节约开支、提高效率的含义，软件测试就涉及到这些方方面面。

在早期的软件开发中，测试的含义比较狭窄，将测试等同于“调试”，纠正软件中已知的故障，常常由开发人员自己完成这部分工作。对测试的投入极少，测试介入也很晚，常常是等到形成代码，产品已基本完成时才进行测试，这种最坏的情况至今依然存在。

直到1957年，软件测试才开始与调试区别开来，作为一种发现软件缺陷的活动。由于一

直存在着“为了使我们看到产品在工作，就得将测试工作往后推一点”的思想，测试仍然是后于开发的活动。潜意识里，我们的目的是使自己确信产品能工作。大学里关于测试的讨论也很少。计算机科学课程中研究数值方法和算法设计，但不包括软件工程，也不包括测试。研究集中在编译、操作系统和数据库，然而它们都不能对研究实际领域里的测试问题提供帮助。

到了20世纪70年代，尽管对“软件工程”的真正含义还缺乏共识，但这一词条已频繁出现。1972年在北卡罗来纳大学举行了首届软件测试正式会议，随后，一批专著相继出版（Hetzel,1973; Myers 1976, 1979）。

Myers注意到人类实现自我的本性对测试工作的重要影响，将测试定义为“为发现程序错误而执行程序的过程”。他指出如果我们的目的是表明程序没有错误，那么将发现不了几个错误，但如果我们的目的是指出错误的存在，那么将发现许多错误。正确的目标和心态对测试成功有着深远的影响（见第4章）。

Myers及其他几位在七十年代的工作是测试过程发展的里程碑。然而，在实际工业领域，软件测试仍然是可有可无的，在时间和经费紧张的情况下，首先砍掉的项目就是软件测试。测试工作总是开展得太晚，以至于从一开始就没有足够的时间使测试工作得以正常进行。管理人员甚至很愿意压缩测试过程，“因为没有时间测试，不管怎样都得交付产品了”（也就是说，所得到的回报似乎要超过所冒的风险），这话是不是听起来很耳熟，因为这种现象在许多组织仍然普遍存在。

直到上世纪80年代早期，“质量”的号角才开始吹响。软件开发人员和测试人员开始坐在一起探讨软件工程和测试问题。制定了各类标准，包括IEEE（Institute of Electrical and Electronic Engineers）、美国的ANSI（American National Standard Institute）标准以及ISO（International Standard Organization）国际标准，这些标准过于庞大，根本无法在日常工作中将它们全部消化。不管怎样，它们中确实包括了一些重要的指南和合同书的底线，提供了非常宝贵的参考。

上世纪90年代，测试工具终于盛行起来。人们普遍意识到工具不仅仅是有用的，而且要对今天的软件系统进行充分测试，工具是必不可少的。我们已经看到各种工具在测试中发挥的重要作用。今天几乎所有的软件开发公司都在研制工具，工具已成为测试过程的重要组成部分。但不管怎样，工具毕竟是工具，它们本身是不能解决实际问题的。

开发与测试的发展

	1960	1970	1995
软件规模	小	适中	超大
软件复杂性	低	中等	高
开发队伍规模	小	中等	大
开发方法及标准	特别	适中	复杂
测试方法及标准	特别	早期	正在形成
独立测试组织	很少	有些	许多
测试重要性的认可	很少	有些	重要
测试专业从业人数	很少	很少	许多

尽管在过去三十年中取得的成绩是巨大的，但在大多数公司里软件过程（包括测试过程）还很不成熟。而且，需软件解决的问题的复杂程度越来越高，平台也越来越复杂，我们已无法找到更有效的方法、工具和能胜任工作的专业人员，毫无疑问，测试者的工作将越来越艰难。

2.2 我们的现状如何

对于软件工程实践成熟度概念的正确认识是测试成功的基础，对现状的了解十分重要，我们需要对所处的开发环境的成熟程度及其对测试过程的影响做出客观的评价。

20 世纪 80 年代，在卡内基-梅隆大学的软件工程研究所（Software Engineering Institute（SEI））为美国国防部开发了软件过程评估方法和能力成熟度模型。在 1987 年的技术报告中提出了五个等级的能力成熟度模型，这是一个组织对自己的软件成熟程度进行评估的模型。

事实上，大多数组织的软件过程开发环境还相当不成熟，根据是否真正采用成熟的软件过程来衡量，全世界大约 75%的软件开发处于 1~5 级中的 1 级。当我们谈到实际领域里的软件测试时，这个 1~5 级中的 1 级就表明了我们实际工作环境的状况，而我们还不得不在这样的环境中开展测试工作。

令人高兴的是 SEI 的模型还告诉了我们如何提高成熟度等级，已经有公司达到了 5 级，只不过它们是少数中的少数（详细情况参见第 6 章的 SEI 和软件工程成熟度，以及第 14 章的发展趋势和实践文献）。

2.3 测试应该如何定位

如果把测试看作是编码完成后进行的活动，那么开发生存周期将由需求、设计、实现、最后是测试而组成。只有等到编码完成之后才开展一些功能测试。这样做无疑很容易发现错误，但是开发者与测试者常常互视为对手。

一旦认识到测试并不只是调试，那么测试不只是邻近开发结束时进行的活动就明白不过了。事实上，开发早期进行的测试是整个测试工作成功的关键，测试有它自己的生存周期，贯穿于整个开发生存周期都可开展积极有效的测试工作（参见第 5 章）。

我们把测试看作发现软件错误的手段，这里用的是软件一词而不是代码，软件还应该包括相关文档，例如规格说明、设计文档和用户手册等。

工程软件质量概念的理论含义不只是可靠性和高效率，还应该是一种与用户需求和用户满意度更加紧密相关的东西，那么如何把当今多数组织所进行的测试与质量保证（QA）联系起来呢？

QA 与过程开发不是一回事，QA 的功能一般不包括过程的推进，也就是说，QA 是一个过程专家，用于建议实现过程改进的方法。文献中通常定义 QA 的功能为：

- 监督软件产品及其开发过程；
- 保证软件产品及其开发过程符合已有的标准和过程；
- 保证在产品、过程和标准上的不足得到管理人员的注意。

相反，测试执行深入的分析、测试，以及对软件产品或系统的总体评价，测试工作忠诚于客户而不是开发者，这是被广泛认可的，然而在许多组织中测试仍然在形式上从属于开发。

从属于开发就意味着测试人员的声音常常是听不到的，当测试人员完成工作，报告了测试结果后，管理者只是简单地表示感谢，无论如何，产品照样交付。因为测试人员在组织中人微言轻，他们的意见很少甚至根本得不到重视。另外，组织的用人政策也能从一个侧面反映出这点，经常是将一些层次低，可有可无的人员安排到测试岗位上。

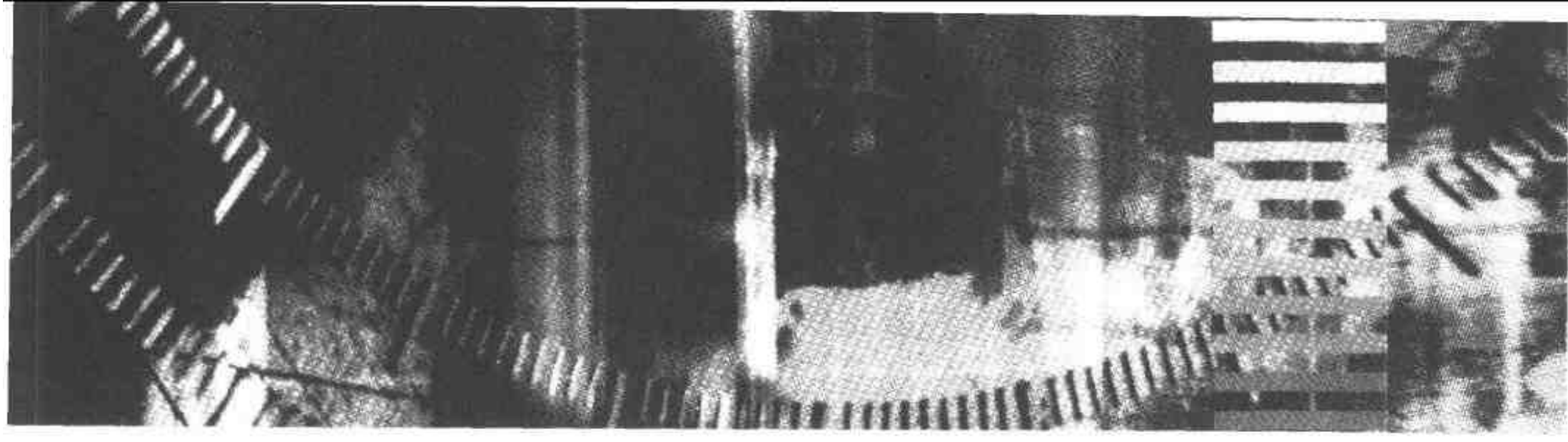
不管怎样，事物总在发展。测试的重要性，在组织中独立行使测试职能的重要性已得到普遍认可，至少理论上如此，必须任用那些技术上胜任的人员，并且给予他们足够的回报（参见第13章），测试工作的成果将成为组织的宝贵财富（参见第6章）。

2.4 参考文献

Hetzel, W.（1973）. *Program Test Methods*. Englewood Cliffs, NJ: Prentice-Hall.

Myers, G.J.（1976）. *Software Reliability: Principles and Practices*. John Wiley.

Myers, G.J.（1979）.*The Art of Software Testing*. John Wiley.



第3章

白纸方法

在以后的章节中将展开讨论测试的要点，许多技术与思路对任何组织都是可采用的。现在拿出一张白纸，将你认为在你的组织中对软件测试过程起关键作用的潜在的改进列出清单，这些改进可以有多种形式：一个新思路；一个新技术或所要研究的新工具；一个软件测试过程的基本变更等。充分考虑目的、实用性和成本效益，在阅读本书时对清单进行反复检查、更新以及调整顺序。为了使你能尽快开始这项工作，下面将给出几个建议，更多的将在以后几章中给出。

潜在的改进

- 调查研究要对程序进行有效审查所应采取的措施。
- 着手确定什么工具能起到最大作用。
- 从今天开始培养“破旧立新”的测试态度。

在本书的最后，附录 H 将给出一个清单，上面列出了全部“要做的改进”，它们是由一个 11 人小组提出的，这 11 个测试专业人员来自不同类型的组织，他们的组织有着不同的软件工程过程成熟度等级。在参加一个软件测试课程班后，他们在清单上列出了他们认为在自己的组织内切实可行的改进，该软件测试课程正是本书的基础。

附录 H 中的第二个清单是在一个中等规模的软件公司举办的一个现场测试课程讲座时编制的，该清单有两个小标题：“软件测试功能内部优先考虑的改进”和“软件测试功能之外优先考虑的改进”，在表中将改进想法按测试集内部和外部分开有它的好处。可按照以下几方面来评估要实施的项目：

- 在组织中实施的难度；
- 实施所要求的资源；
- 对测试过程改进的盈利；
- 真正实现该项目所需要的短期、中期或是长期的努力。

在可能的想法被收集起来后，这种评估可以做为第二步进行。

应该有一个长期目标，真正的进步常常需要两年、五年、十年甚至更长时间的努力，我们已经知道有几个公司在做长期计划，有的正在实施第三个十年计划。

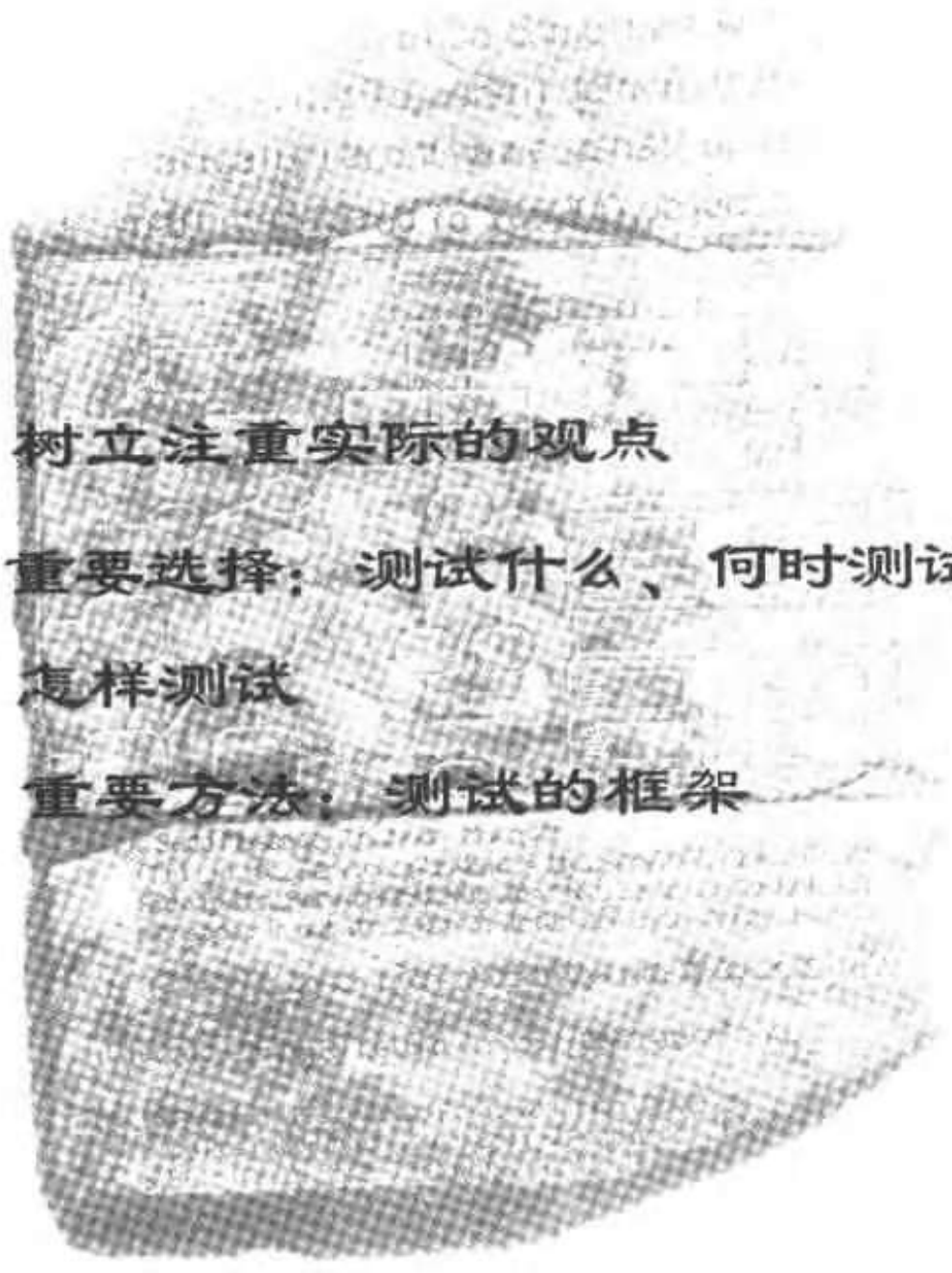
保持提出这样一个正确的问题："我们现在所做的的确是对以往工作的改进吗？"，如果回答"是"，而且接下去每一次的回答都是"是"，那么我们的工作就是一种持续的改进，其结果是以有限的资源取得令人惊讶的成绩。一旦所做的改进有目共睹，来自组织中管理层及其他部门的支持就会很快得到提高

第二部分

测试过程改进框架

“必胜的信心固然重要，但充分的准备更是至关重要”

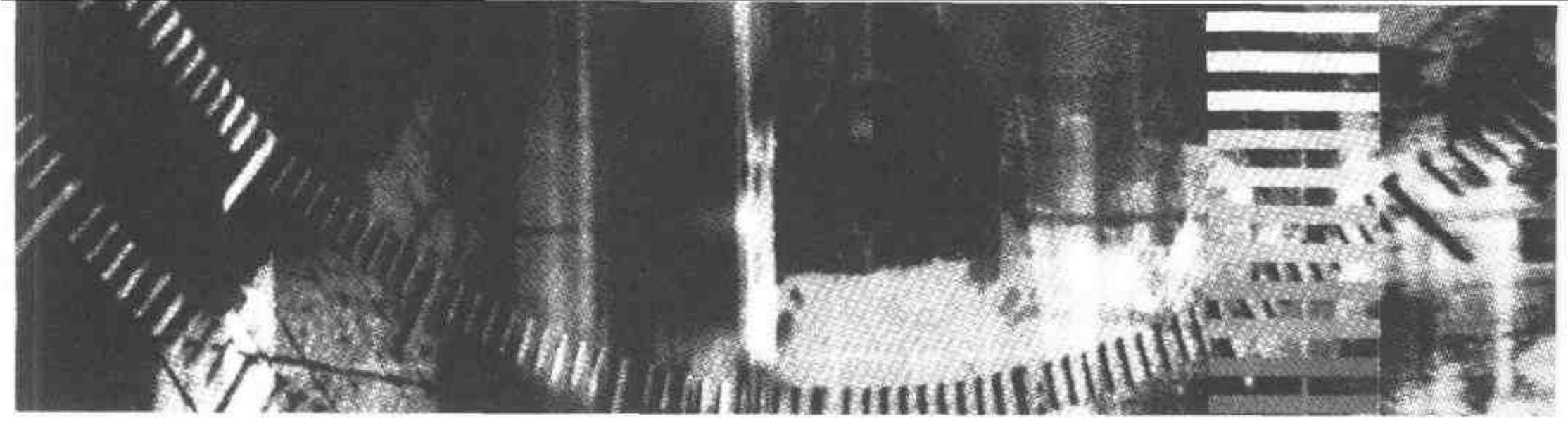
——Joe Paterno



第4章 树立注重实际的观点

第5章 重要选择：测试什么、何时测试、怎样测试

第6章 重要方法：测试的框架



第4章

树立注重实际的观点

软件是人写的，是人就会犯错误。即使标准的商业软件里也存在错误，只是严重程度不同而已。我们无法完全避免错误，但通过努力可将它们尽早定位，尤其是对那些严重错误。

我们需要了解并能采用合适的测试技术，但是，对于一些能够使测试更加有效的非技术性问题有所认识也同样重要。我们不仅要做好的测试，而且要做聪明的测试。

对测试过程及其所包含的要素给出明确定义十分重要，做什么，为什么做以及怎样做，这些问题都是彼此交流、互相合作的基础。这些定义构成了一种通用语言，提供了进一步研究的基础，可帮助我们获得测试工作的重点，不再只是扮演“救火员”的角色。

作为测试人员，强烈的责任感和基本的敬业精神对测试工作的成功有着意义深远的影响，进一步还将推进整个开发过程的改进。

4.1 我们的目的是什么

测试的最终目的是什么？最终的底线就是令用户满意。最后掏钱的是他们，如果我们要在商业上有立足之地我们就得解决他们的问题，我们根本的目的必须是他们的高兴和满意。高质量是我们的追求目标，但质量并不是一个抽象的概念，我们开发的系统是要能用的，而且要能成功地使用，决不是束之高阁的摆设。如果质量是一个在现实中有意义的、有用的目标，那么就得有用户的参与。

测试人员不可能负责全部的质量程序，但不管怎样，许多测试决策最终应该以用户的满意为基础，在大型的、复杂的开发组织中，在各种各样的软件开发项目中，这一点常常容易被忘记。用户希望系统做什么？使用时系统能按用户的要求正确工作吗？这些是测试决策需要考虑的实际问题。

4.2 关于错误你知道多少

故障（fault）、失效（failure）、错误（error）、缺陷（defect）、问题（issue）、隐错（bug）和过失（mistake），如果测试人员的全部工作就是以查找它们为目的，那么对这些概念有所了

解是十分重要的。

4.2.1 什么和为什么

错误隐藏在软件开发过程的产品中，这里的产品不仅仅指代码，还包括产品需求、功能规格说明、草案及最终的用户手册、草案及最终的数据表、技术支持等许多开发过程不同阶段的其他产品。

软件生产可以看成是一系列不完善的转换过程，每一次转换产生一个产品或交付，如果在将一种表示形式转换成另一种表示形式时，不能进行完整和准确的转换，或者不能完全匹配问题的解决方案，就会引入软件错误。

软件错误是人为错误。所有的人类活动，尤其是复杂的活动都难免犯错，必须接受这个事实，将测试的重点集中在查找错误上，并采用最有效的方式。

以下是几个经常遇到的含义非常接近的词汇（IEEE/ANSI, 1990[Std 610.12-1990]）：

- 过失（mistake）：人为产生的不正确结果。
- 故障（fault）：在程序中的一个不正确的步骤、过程、或数据定义，是过失发展的结果。（可能导致失效。）
- 失效（failure）：一个不正确的结果，故障（例如崩溃）引起的结果（表现）。
- 错误（error）：导致不正确结果的全部。

过失是人犯下的，是人做的一件错事。编程时突然电话铃响了，或者是一时马虎按错了键，这些过失都有可能将一个故障或隐错带进产品中。

失效是这些故障的表现形式，即使故障或隐错没有引起失效，它们也是客观存在于文档或代码中的，测试人员的任务就是找出它们。

当出现了失效，比如：系统崩溃、用户得到一个错误的信息、取款机给出错误的钱数等等，按照以上定义，这些导致不正确结果的全体都是错误。

没有几个人能十分准确的使用这些词条，实际上也没有必要。本书中我们将主要使用缺陷（defect）和错误（error）两个词。在测试过程中，我们将找到的错误有不同的类型，明白这点非常重要。

4.2.2 错误在哪里

通过对错误分布情况的仔细分析，可以帮助我们将测试的主要精力更好地集中到最有价值的地方，图 4-1 给我们最大的启发就是错误集中在开发过程的早期。

开发早期的错误通常是很多的，而且令人讨厌的是它们还会转移到后期。这和制造业的装配线类似，如果一个坏零件或次品被允许上线，从这点开始，包含它的组件就是“坏”的，如果该组件下了线，并出了厂门，情况就会更糟糕，就得为那个坏零件付出代价。换句话说，错误不是自封闭的，当它们转移到后面的组件中时，往往会以新的形式出现。

4.2.3 错误的成本

所有的错误都是要付出代价的。没有被发现的错误，以及那些在开发过程中很晚才发现的

错误都是成本最高的，没有被发现的错误将在系统中迁移、扩散，最终导致系统失效。直到很晚才发现的错误常常造成代价昂贵的返工。那些没有被发现的错误导致系统失效，造成严重的财产损失，有时还会带来法律上的麻烦，系统将终身为此付出高昂代价。

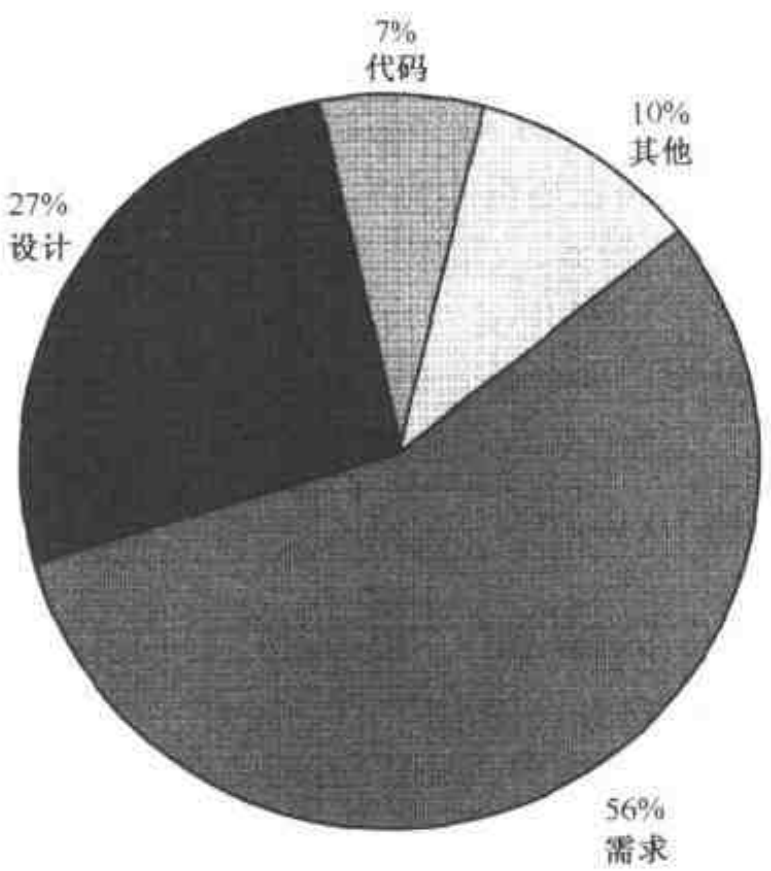


图4-1 缺陷分布情况。数据摘自Dick Bender（1993）的“Writing testable requirement”，图中的“需求”一词包含了功能设计（© 1993, 1994 Software Development Technologies）

这意味着测试是贯穿开发全过程的工作，也意味着对最终产品的测试仅仅是软件质量大战中的一个战役，而且不是代价最高的战役。今天，40%～70%的软件开发时间和资源都花在查错和纠错上。不幸的是大多数组织还没有一套办法来准确计算成本，不管怎样，在使用测试资源方面的任何有意义的改进都能极大地降低开发成本。

4.3 关于测试的一些定义

关于测试的定义有多种，也没必要为它们之间的差异争论不休，重要的是如何使用这些定义，集中精力解决那些测试中真正需要解决的问题。不过，对各种不同定义的了解是必要的。

不同时期关于测试的定义

（1）确信程序做了它应该做的事（Hetzel，1973）。
（2）为找出错误而运行程序或系统的过程（Myers，1979）。
（3）查出规格说明中的错误，以及与规格说明不符的地方。
（4）一切以评价程序或系统的属性、能力为目的的活动（Hetzel，1983）。
（5）对软件质量的度量（Hetzel，1983）。
（6）评价程序或系统的过程。

（7）验证系统满足需求，或确定实际结果与预期结果之间的区别。

（8）确认程序正确实现了所要求的功能。

这些定义说法不一，但都很有用。有一些如定义（2）强调测试过程中所作的事情（Myers, 1979），其他的定义则更侧重于测试的更一般的目标，如评估质量（5），用户满意（1，3），而其他一些如定义（7）、（8）将重点放在预期结果上。

例如，如果用户满意是我们的目的（应该如此），这种满意或构成满意的东西就应该体现在需求中，如果这些需求通过书面表述得十分完整，我们就拥有了很大的优势。我们可以问问自己："规格说明到底有多完整？得到正确的实施了吗？"

确认预期结果与实际结果一致的定义（7）是很有价值的，因为它注意到了一个事实，那就是在测试时需要明确预期结果，然后将它们与实际发生的结果相比较，这种比较是测试的基础之一。

IEEE/ANSI 关于测试的定义

（1）在特定的条件下运行系统或部件，观察或纪录结果，并对系统或部件的某些方面进行评价的过程（IEEE/ANSI，1990[Std 610.12-1990]）。

（2）对软件进行分析，找出其现有状况与要求状况之间的差异（即 bug），并对软件的特性进行评价的过程（IEEE/ANSI，1983[Std 829 -1983]）。

ANSI 在正式标准中使用了这两个不同的定义，它们都有点不令人满意，似乎都是面向确认的。例如，在第一个定义中，诸如"在特定的条件下运行..."的说法本身就隐含了只有在某些东西能运行了测试才能开始。

在本书中，我们希望测试的定义不排除对像规格说明这类文档的测试。一种称之为验证的测试就能够针对开发过程中的任何中间产品进行，而另一种称之为确认的测试则只能通过运行代码来完成（见第 5 章）。

管理人员对测试有他们自己的认识和定义，这也从一个侧面反映了他们对测试人员的重视程度。他们希望产品是安全可靠的，在正常和不利的条件下都能正常实现其功能，他们也希望产品正是用户所需的，只有这样，他们才能将产品交付用户并赚到钱。

质量对管理者来说也是十分重要的，为了公司的未来，他们需要对自己的产品树立信心，对经济利益的考虑在领导决策时总是占有十分重要的地位，而且他们要确保不被投诉。

对测试者来说，什么是最重要的呢？什么是一个能把测试人员的注意力集中到测试工作的本质的定义呢？

针对测试人员的最好的定义

测试以发现错误为目的，测试是努力发现产品中每个可以想象到的故障或弱点的过程。（Myers，1979）

4.4 优秀测试人员的测试态度

对测试人员来说测试究竟意味着什么？

4.4.1 测试者寻找错误

如果测试人员的工作就是发现产品中的每个可想象到的故障或弱点，那么一个好的测试就是有可能发现尚未被发现的错误的测试，一个成功的测试就是发现了以前没有被发现的错误的测试。

集中精力揭露错误是一个优秀测试人员应持的基本态度，这就是我们的工作，也是体现自我价值的基础，在发现错误的日子里我们会感觉良好，当超额完成了一天或一周的找错任务时，我们会激动不已，我们为找出错误而庆祝——为了优秀的产品。

4.4.2 测试是破旧立新

测试是主动的、富于创造性的破坏工作，它需要极强的想象力、毅力和强烈的使命感，需要对复杂结构的弱点系统地定位并证明其失效。这就是为什么测试自己的产品尤其困难的原因之一。人们本能地不愿意看到自己的产品中出现错误。

当我们测试自己的产品时，立足点是它是能工作的，是按照我们构造它的方式工作的。在测试自己的产品时，我们十分清楚边界和问题在哪里，但我们并不认为这是什么严重的问题，也不认为任何人在以后都会遇到这些问题（或由此产生的后果）。作为产品的开发者，我们的内心深处认为产品是正常工作的，所以我们对产品的测试并不太努力。这时最需要的是另外的一些人，他们持有的态度是："我就是要破坏掉这个东西、我知道有错而且要找出错误，这就是我的工作，我以此谋生。"

4.4.3 测试人员找出错误，对事不对人

错误存在于产品之中，而不在生产它的人身上。带着"为了破坏而测试"的态度，我们针对的并不是某个人或某个开发队伍，我们只是寻找由这些开发者生产的产品中的错误。

开发人员要理解测试人员并不是通过寻找产品中的错误来"打击"他们，而是帮助他们，因为开发人员自己没有时间、技术或者工具，没有测试自己产品的公正态度。建立良好的关系非常重要，测试人员完全可以主动地说："你们是开发领域的专家，已经尽了最大努力，我们正在花时间学会如何测试，我们要找出工具所不能及的地方，我们将慢慢成为测试的行家，我们一起努力将会贡献出更好的产品。"

关键是使得开发人员与测试人员形成一个团队，测试者扮演的角色就是找出错误并把它们摆在桌面上，开发人员对这些往往是心存戒心。测试人员要寻找开发人员生产的产品中的缺陷，从这点来说，彼此互为对手，另一方面，为了生产出更优秀的产品这个共同的目标，大家成为一个团队中的伙伴，在对手和伙伴之间达到平衡是非常重要的。如果开发和测试的功能都能得到很好地发挥，对于组织来说将获益匪浅。独立性固然重要，但在实际工作中团队精神的培养常常是十分有益的。

当召集开发和市场人员开会时，要始终围绕产品中的问题开展讨论，而不去针对生产这些产品的人。

4.4.4 测试提高产品价值

包括测试、开发、市场以及管理在内的每个人都需要充分认识到，通过测试尽早地发现错误并把它们摆到桌面上可以提高产品的价值，可以使开发人员避免在隐藏着错误的基础上生产产品，可以使市场人员交付的产品的确为用户所需，也可以使管理人员获得他们所追求的最高质量和最大效益。

发现错误，帮助开发人员把工作做得更好，最终看到公司拿出优质产品，以此我们可获得极大的成就感和积极的团队精神。目的就是要生产出高质量的产品，避免遭受不成功带来的痛苦。如果不成功，后果之一是导致用户不满意，投诉我们，这样我们都将丢饭碗。

如果管理者能积极培养测试和开发人员之间的合作意识，事情会好办得多，不过首先往往是测试人员能清楚地认识到他们对产品质量的贡献（参见第 13 章）。

4.5 测试人员怎样工作

已经有了“测试人员的态度”，我们该怎样寻找错误呢？

- 检查内部结构和设计？
- 检查功能性的用户界面？
- 检查设计目的？
- 检查用户需求？
- 运行代码？

答案是这些事情都得做，也许还会更多。

实际工作中，在用户或其他部门对进度的压力下，高层管理者划定的底线常常是不切实际的，与以用户满意为最终目标是背道而驰的。要生产出没有缺陷的软件，尽管从未有人做到过，但管理者还是常常为此感到压力很大。生产力可能低下，可能继承了一些还没有明确或充分考虑的过程，使我们很难知道从何入手，在这个背景下，实际测试者和管理者都会问一个问题：“我们首先做什么？”

答案是从你所在的具体组织的实际情况出发，一步一个脚印。在开发和测试过程中不必追求质量管理大全中的每一条来获取切实的改进。在开展下一步工作之前，应该在潜在的改进清单（参见第 3 章和附录 H）中增加一些条目以下列出了从本章得出的一些可能的内容。

4.6 现在能做什么

以下这些建议可以添加到第 3 章描述的“白纸”中。

- 对你所在组织关于测试的定义进行讨论并达成共识，以本章的定义作为起点。
- 当得到需求时，组织一个开发人员和测试人员参加的会议，对需求展开讨论。
- 努力得到管理者的支持，在测试和开发人员之间形成良好的团队合作精神。
- 使管理者认识到测试必须尽早开始，测试不是事后活动。
- 使管理者认识到测试每一个产品需要合适的人员梯队。
- 邀请开发者或开发部门的领导讨论产品开发风险。

● 访问用户，了解他们即将使用产品的方式，为如何开展测试提供思路。

4.7 参考文献

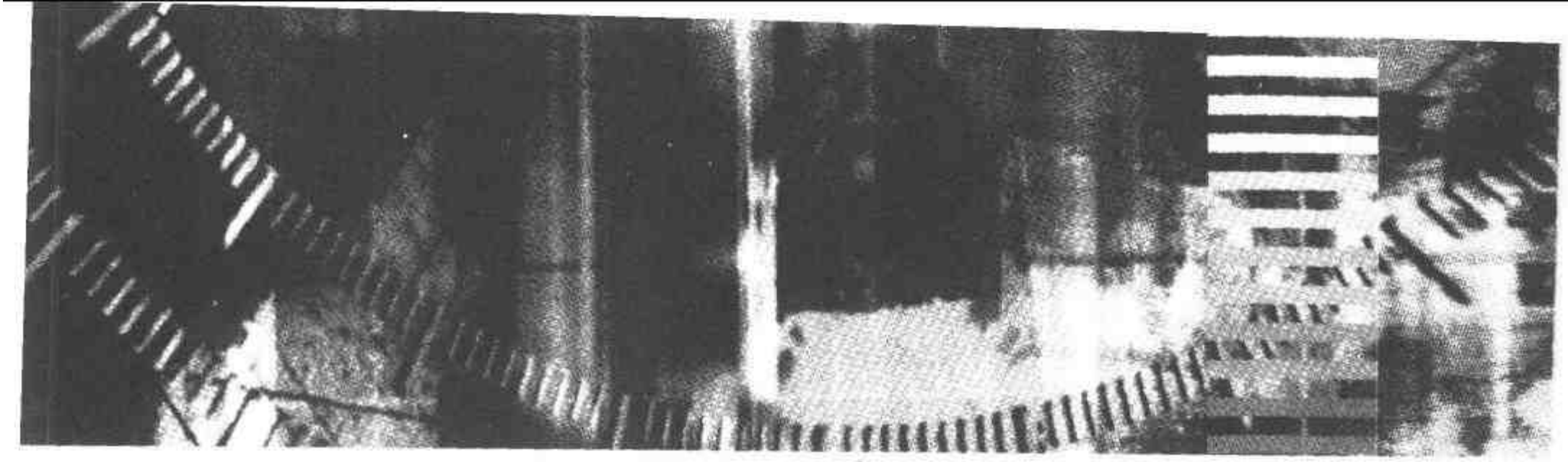
Bender, R（1993）. "Writing testable requirements", *Software Testing Analysis & Review（STAR）Conference Proceedings.*

Hetzel, W.（1973）. *Program Test Methods.* Englewood Cliffs, NJ: Prentice-Hall.

IEEE/ANSI（1983）. IEEE Standard for Software Test Documentation,（Reaff. 1991）,IEEE Std 829-1983.

IEEE/ANSI（1990）. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990.

Myers, G.J.（1979）. *The Art of Software Testing*. John Wiley.



第5章

重要选择：测试什么、何时测试、怎样测试

优秀的测试人员总是能够在以往实现或运行过的测试的基础上，设计出更多的测试。穷尽测试将意味着许多产品根本无法投入市场，因为测试将不可能完成，即使做完了，那也是代价大得惊人。

开发组织常常要花 50%甚至更多的时间在找错和纠错上，但是在大多数的组织中，对于这个巨大资源的使用既没有正式的文件，也缺乏有力的依据，实际中需要做出选择，但选择的基础究竟是什么呢？

一个选择就是确保测试是恰当的——确保那些最关键的项目被测试到，而不将有限的测试资源浪费在无关紧要的地方。另一个选择就是尽早开展测试——集中精力在错误刚刚被带入的阶段就发现它，避免错误向后扩散转移。

测试有着与开发周期并行的生存周期，包含了验证测试和确认测试，测试应该尽可能地在开发生存周期中的最有效位置加入进去。

测试资源不可能在短期内有很大的增长，所以，我们必须尽可能好地使用它。

5.1 风险及风险管理

为什么我们做以下事情之前会犹豫不决？

- 跳伞；
- 使用试验药物；
- 在某公司的股票上投资$20 000；
- 为一架新飞机的首航领航；
- 修理高压线路；
- 借给朋友甚至亲戚$10 000；

- 拆卸炸弹；
- 雇一个少年照看孩子。

这些事情都是危险的，做它们需要冒险，在做之前需要估计出有多大风险，采取一切可能的防范措施以避免最坏的情况发生。

风险就是不愿易看到的事情，例如丧失生命、大量的财政损失等发生的可能性。当开发的系统无法正常工作时，其后果可以从有点不舒服一直到大的灾难，对这些系统的测试应该包含对风险管理的认识。

我们不可能什么事情都做，必须有妥协，但是我们不可能去冒难以承受的高风险，必须要问的几个关键问题是："谁将使用产品？用它干什么？如果它出问题危险是什么？它的工作结果是什么？经济上有损失吗？会失去用户的满意吗？会丢命吗？"

对每一个产品，我们必须做效益最高的测试，保证产品足够的可靠，足够的安全，并能满足用户/客户的需求。这些事情听起来容易，但实际情况可能是几行代码就需要数百个测试用例，如果一个用例需要执行半秒钟，那么对于一个主要产品运行完所有可能的用例仍有可能花上几十年。

这并不意味着我们将放弃测试，只是意味着不可能有足够的时间对每一件事情进行全面测试，因此必须冒一定风险做出决定。当以风险作为测试选择的基础时，我们做的就是要选择系统中将产生最严重后果的部分，并集中注意力对它们进行测试。

另一个选择测试重点的基础是使用频率，如果系统中的某部分被经常使用，但其中有错，那么该部分的频繁使用将使得出现失效的可能性大大提高。

集中测试那些系统或程序中最可能出错的部分也是合理的。为什么在软件的某一片断中存在特别大的风险，对此可能有许多理由，要紧的是能看清它们，并基于观察做出有实际依据的决定。理由可能来自系统那部分的生成方式：可能是因为以一定的方式组成开发队伍，由一个缺少经验的人编写了复杂的代码块，导致在系统的这部分存在很大风险；可能时间太紧；可能开发资源不足；还可能经费预算太紧。

实际上，这种风险分析常常是基本交流中的一件事。当我们与开发经理和产品开发者交谈时，问他们最担忧的是什么，如果回答是："哎，进度安排根本无法实现，系统的这部分太复杂，无法保证能做好"，那么作为测试者就应该把这些话当成应该给予特别关注的提示（见图5-1）。

对于每一个要进行的测试活动，都要对潜在的所有类型的失效情况做出判断，以确定测试目标的优先级。在10个不同的测试活动（4个验证和6个确认）中，可根据产品的规模、复杂性和重要性决定要开展的活动，哪些（如果有）可以跳过去，但无论如何，要把测试的风险控制在可接受的范围内。

风险不仅是决定测试管理的基础，同时也是决定日常测试活动的基础。图5-1清楚地告诉我们这不是一件简单的事情，但并不意味着一切对风险进行估计的尝试都应该抛弃，相反，我们应该充分利用本节提出的各种因素，以此为其起点对目前测试工作的拨款进行更加合乎实际的估计。

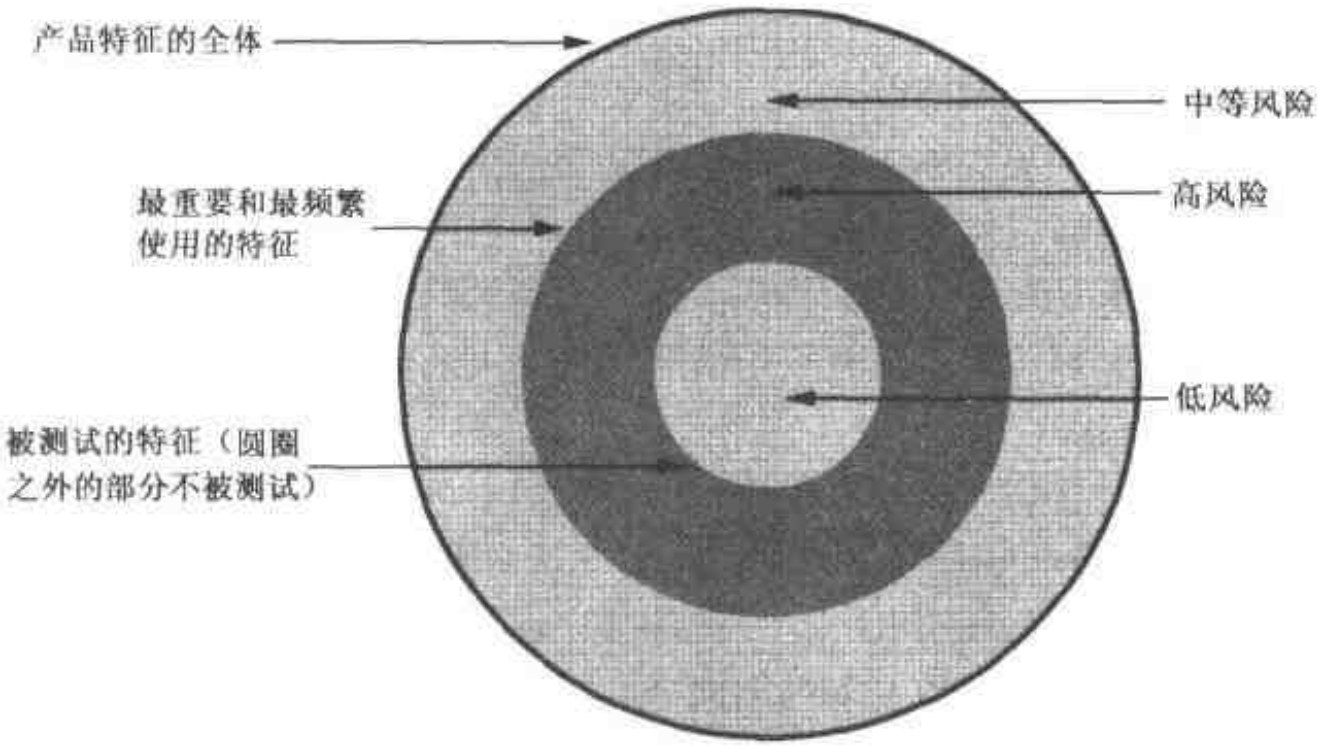


图5-1 作为选择测试基础的系统风险。图中的关键问题包括“被频繁使用的特征”，此确定的准确性如何？它们与实际的接近程度如何？如果它们切合实际，那么相关的风险部分划分就可能合理，如果它们不能反映实际情况，那么“高风险”区域就可能比图中画出的大得多（© 1993, 1994 Software Development Technologies）

5.2 尽早开始测试

软件开发过程中的每一个开发阶段都有自己的产品，对它们进行测试可了解该阶段工作的成败。在早期这些产品就是需求和规范，可阅读它们并与其他文档进行比较。后期就有了可运行的代码，当然对代码也可以进行纸面上的审查。可能的话应尽早测试，因为许多重要的错误是在早期引入的，事实证明，一半以上的缺陷常常是在需求阶段带进的。

尽早测试可避免缺陷转移。我们常常会错过许多发现缺陷的大好机会，而将它们留给了下一阶段，如果到这个时候才发现它，那么消除该缺陷的付出将大得多。缺陷转移到的阶段越靠后，付出的代价就越高。错误在它刚刚带入时就被发现，该错误的成本最小，如果到交付给用户之后才被发现，该错误的成本将达到最大。

在一个压力很大的项目中以及不太理想的组织中，面对糟糕的需求我们能做些什么？使用第7章的评审方法可对它们进行验证，初步审查可以从小规模开始，过去二十多年的实践证明这些方法可有效发现大部分错误。即使在正规的验证开展之前，有些非正式的方法也能提供很多帮助。

简洁和良好的交流很重要，用户对于想要什么会改变主意，实际上这也是过程的一部分。但是，做为测试者和开发者，我们更需要注意的问题就是怎样记录他们的需求，怎样评审他们，怎样使他们得到满足。

我们要坚持召开有关人员（包括产品开发人员，用户，测试人员及市场人员）参加的会议，讨论需求和规格说明，并对讨论过程进行详尽记录，这种交流常常是容易被忽略的。更多的交流将针对语言和术语的定义，例如：“这是可交付文件，谁对这份文件负责，对这份文件我们做了什么改动，由于它很重要，我们打算照这样来测试它”。到这个时候，我们已开始触及到了真正的软件工程过程和成熟度问题的基础。

5.3 测试过程的基本形式：验证和确认

测试可以分为两种基本形式。假设测试是在早期开始的，这些测试的定义不仅包含代码测试，而且包含产品的文档和其他非执行形式的测试。

验证，按照 IEEE/ANSI 的定义，是为确定某一开发阶段的产品是否满足在该阶段开始时提出的要求而对系统或部件进行评估的过程。

我们有清晰完整的需求吗？有一个好的设计吗？按照设计生产出的产品是什么？验证就是对诸如需求规格说明、设计规格说明和代码之类的产品进行评估、评审、审查和桌面检查的过程。如果是针对代码，其含义就是代码的静态分析——代码评审，而不是动态执行代码。验证测试可应用到开发早期一切可以被评审的事物上，以确保该阶段的产品正是我们所期望的。

有人把验证称为“人工”测试，因为它通常涉及对文档的书面阅读，相反，确认则通常是在计算机上运行软件。

确认，按照 IEEE/ANSI 的定义，是在开发过程中或结束时，对系统或部件进行评估以确定其是否满足需求规格的过程。

正式的确认包括实际软件或仿真模型的运行，确认是“基于计算机的测试”过程，它经常暴露错误的现象。

本书的中心论题是包含验证和确认的测试。

定义：测试=验证+确认

验证和确认是互补的，发现错误的效果会由于它们中的一个或另一个没有完成而受到损失，它们是为捕获不同类型问题而设计的过滤器。

历史上测试一直是主要针对确认，而且这种情况还将继续，这并不是说我们应该停止做确认，而是要更加清楚怎样去做，并怎样结合验证一起做。我们必须保证在适当的时候，对适当的产品进行验证和确认。

5.4 测试、开发生存周期及合同

由于组织的类型和环境的不同，存在不同类型的软件生存周期模型，但它们大体是一致的。图 5-2 给出了一个典型的包含测试在内的开发生存周期模型。

图5-2 一般的软件开发过程。这是一个典型的开发生存周期模型，IEEE/ANSI软件验证和确认计划（Standard 1012-1986）在“测试”和“运行和维护”之间还有一个称为“安装和检测”的阶段。IEEE/ANSI并不要求一种特定的模型（© 1993, 1994 Software Development Technologies）

由于历史的原因，最初命名为“测试”的阶段只是生存周期中的一个时间段，其间软件产品的部件被评估和集成，并进行确认测试。设计阶段常常分为功能设计和内部设计。

开发软件生存周期的过程——IEEE/ANSI,1991(Std 1074-1991)

> 标准定义了组成开发和维护软件所必需的过程中的一系列活动，覆盖了贯穿整个生存周期的管理和支持过程，还有从概念摸索直到退出使用的软件生存周期的各个方面，给出了相关的输入和输出信息。如果在软件生存周期的最初阶段就使用这些标准，那么利用这些过程以及其中的活动可最大程度地满足用户利益。标准要求定义用户的软件生存周期，并表明它与典型软件生存周期的对应关系。标准不打算定义或隐含一个自己的特别的软件生存周期。

尽管存在各种不同的生存周期模型，但它们都会涉及一些重要的组织问题：“谁是真正做测试的？测试是承接转包的工作吗？测试与开发是同一个承包者吗？”关键的一点是对于每一个开发阶段，都有一个与之相对应的测试阶段。

图 5-3 中的 Dotted-U 模型更详细地给出了开发周期与测试周期的结合情况。

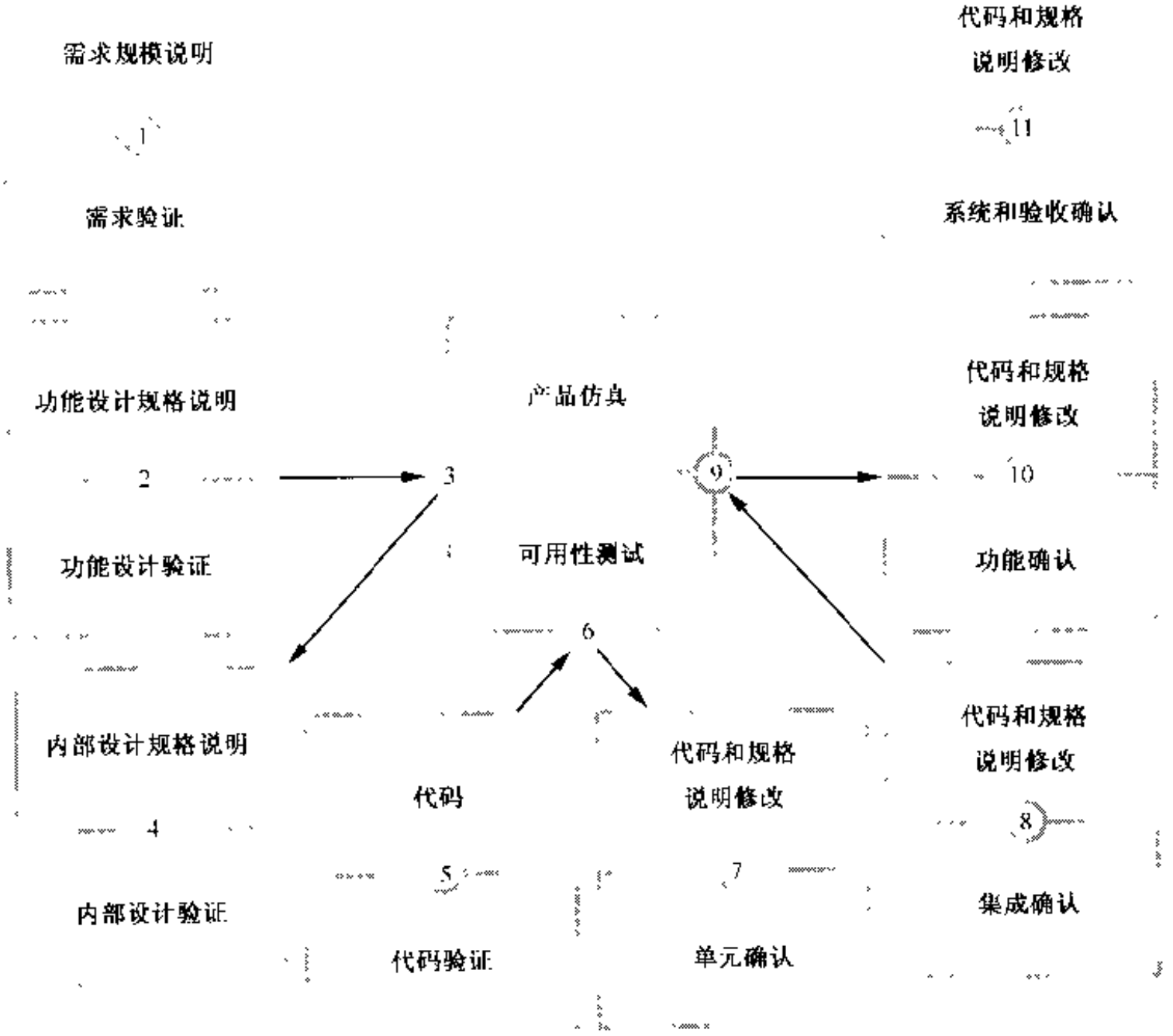


图5-3　SDT（软件开发技术）Dotted-U模型表明了开发和测试过程的结合，本书将清楚地描述图中给出的各部分（© 1993，1994 Software Development Technologies）

注意到开发和测试在各自周期中各个阶段的一一对应关系，每个开发的主要交付都要通过测试机构的测试（验证或确认）。在开发生存周期中有一个需求阶段，相应地在测试生存周期中就有一个需求的验证阶段，设计验证应该与设计开发阶段同步进行，等等。还应注意到任何测试活动的计划可能发生在图中所标出的活动之前。图中覆盖了 10 个功能性测试活动（四个验证和六个确认）。

5.5 有效测试

有效测试可排除错误。在任何特定的情况下，我们如何知道应该做多少测试？应该做全面测试还是部分测试？

测试有如下基本形式：

（1）全面测试　起点不晚于需求阶段，一直进行到验收测试。

（2）部分测试　从功能设计完成后的任何时候开始，并尽可能少地影响需求和功能设计。

（3）末端测试　高度面向确认的测试，对需求和功能设计没有影响。

（4）审计级测试　对计划、过程和产品的充分性、正确性以及是否符合标准进行梗概性审计（Lewis, 1992）。

问题的关键再一次落到了风险管理身上，为了做出更有效的风险管理决策，我们有必要对关键软件做出清楚的定义。

关键软件，按照 IEEE/ANSI（1990 [Std 610. 12-1990]）的定义，是指那些失效可能影响到安全，或者可能造成巨大的经济或社会损失的软件。

对于关键软件，或者是使用范围大，应用方式繁多的软件，要采用全面测试。

对于使用范围小，规模小的非关键软件（失效的后果不会导致灾难），可采用部分测试。

为了使有效测试成为可能，我们需要软件开发过程产生以下产品：

- 需求规格说明（完整测试需要）；
- 功能设计规格说明（完整、部分和末端测试都需要）；
- 内部设计规格说明（最有效的完整和部分测试都需要）。

从测试的角度出发，完整测试必须具备相当清晰和完整的需求规格说明，为了改进过程，测试人员所能做的第一件事就是坚持得到他们需要的那种规格说明。

部分和末端测试至少需要功能设计规格说明，如果要使测试更有效，还需要内部设计规格说明，因为内部的信息可以告诉我们产品是怎样构造出来的。

5.6 测试的效益

有效测试就是能成功找出错误的测试，但是，从经济角度考虑它并不总是值得去做的事情。我们已经讨论了错误成本（如果让错误留在产品中所付出的代价），但是在实际中，我们会面临并且需要明确许多关于测试工作的成本效益问题。

我们知道测试的真正花销吗？它是怎样影响开发时间的？我们知道测试在整个开发资源中所占的比例吗？测试工具为我们节省时间了吗？工具本身需要花销吗？我们是最大地挖掘它们的潜力，还是将它们束之高阁？我们将测试资源用在寻找那些有最大经济风险的错误了吗？如

果我们观察测试所开发的产品对工作目标的总体影响，我们的测试花费有效吗（见图 5-4）？

当我们开始在现实组织中回答这些问题时，将发现在认识和行动之间存在很宽的鸿沟。换句话说，许多人非常清楚这些问题的答案，但他们也知道那是远远不可能做到的，因为大家都在忙于“救火”。

这些问题常常被认为是领导考虑的。在实际中测试过程的成本效益问题极少被考虑。（有关成本效益及其度量的详细介绍，请见第 12 章。）

> “错误查找越有效，产品生存期内节省的开发和维护成本越多。几个实例表明，部分测试能节省 1.5 倍的花费，全面测试甚至能节省 2 倍的花费。这些数据是否准确，是否得到公认并不重要，重要的是它们使得测试能不同程度地产生效益这个假设得到了具体表现。”
>
> （Lewis, 1992; p.280）

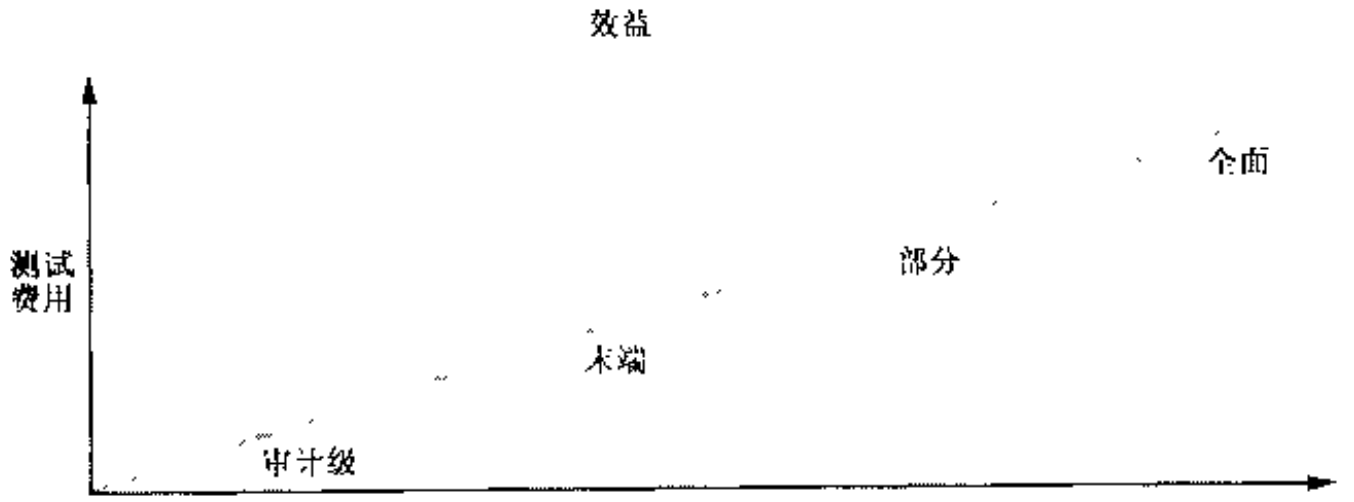


图5-4 错误查找越有效，产品生存期内节省的开发和维护成本越多（© 1993, 1994 Software Development Technologies）

5.7 现在能做什么

- 以本章中那些与你的环境关系最密切的因素为基础，开始做风险估计。
- 估计一下你是否能在全面和部分测试之间进行适当组合。
- 与开发人员进行交谈，了解他们对系统担忧的地方（哪怕仅仅是“感觉”）。
- 将开发人员和测试人员，最好还能有客户和市场人员召集在一起，改进需求规格说明。
- 通过进行更多（更早）的验证测试估计组织的收益。

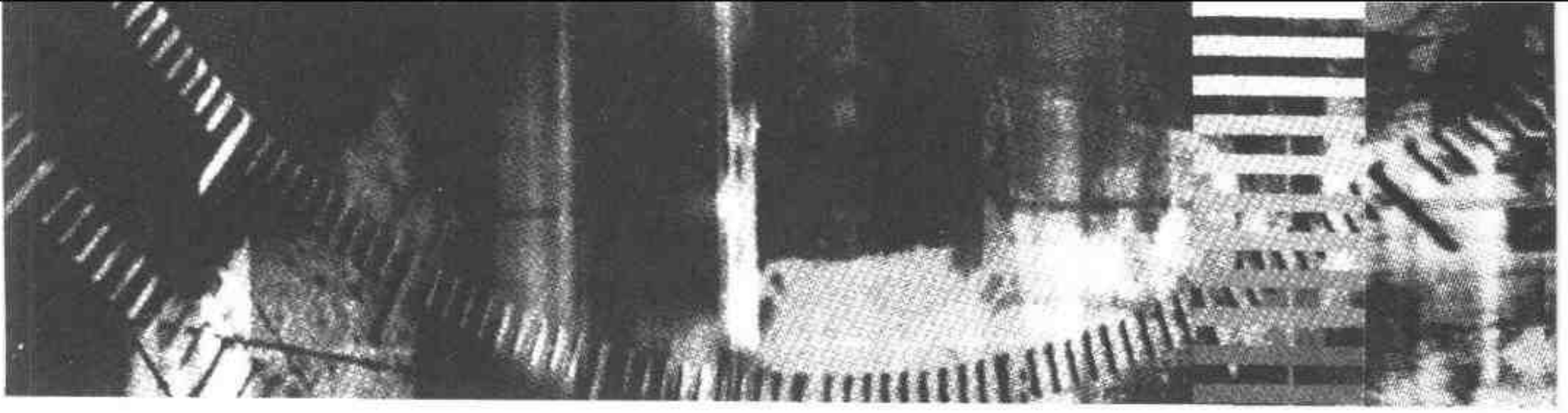
5.8 参考文献

IEEE/ANSI(1986). IEEE Standard for Software Verification and Validation Plans, (Reaff. 1992), IEEE Std 1012-1986.

IEEE/ANSI(1990). IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990.

IEEE/ANSI(1991). IEEE Standard for Developing Software Life Cycle Processes, IEEE Std 1074-1991.

Lewis, R.(1992). *Independent Verification and Validation*. John Wiley.



第6章

重要方法：测试的框架

通常，任何在软件开发项目中被广泛使用的软件工程方法对于软件测试工作也是适用的，因此，那些诸如计划和分析、正式文档、配置管理、度量、标准、工具和过程都能用于测试。

随着工程项目的日益复杂，开发和测试组也变得日益庞大，这些方法对于项目的成功更是至关重要。

这些想法的大多数都能在过程成熟度的概念中有所体现，这些方法被使用得越多越系统，软件过程的成熟度就越高。本书的后面将详细讨论大多数方法。

6.1 计划

没有计划，我们就不可能知道：

- 我们想做什么
- 我们打算怎样做
- 要花多长时间
- 需要的资源
- 成本

我们很容易看到一个中等规模（更不必说大规模）的软件开发工程，由于计划没有做好，而变得混乱不堪失去控制，计划是任何测试活动的第一步。

如果测试被当成软件工程的一个重要组成部分，测试者将必须花时间去分析产品，在设计测试之前评审产品需求，在实施测试之前对测试进行设计。事实上，设计测试的过程对检查规格说明中的缺陷很有帮助，因此，一个好的测试过程能改善规格说明。换句话说，我们必须计划，而且一旦开始计划，其他各种类型的问题都会出现。

哪类标准是适用的？我们打算作审查和评审吗？为了找出一个特定产品中的最关键问题应该使用什么工具？我们有时间和经费完成应该做的测试吗？作为测试者我们要对我们的过程进行何种评审？我们有优先权吗？

人们总是希望在下一个阶段再进行测试，特别是在有时间压力时更是如此，这样测试人员

就只能是面对代码开始找其中的错误。无论如何，在早期的测试阶段上投资是合算的，将会得到数倍的回报，同样，把时间花在早期的计划上也是从不会浪费的，而且将会大大缩短总时间周期。在每一个验证活动中，计划和执行都是两个不同的阶段，无论是需求验证、功能设计验证、内部设计验证还是代码验证，情况都是如此。

6.1.1 验证计划中要考虑的因素

每一类型的验证（需求、功能设计、内部设计、代码）都要考虑以下问题：

- 将进行的验证活动
- 使用的方法（审查、走查等）
- 产品中要验证的和不要验证的范围
- 没有验证的部分所承担的风险
- 产品需优先验证的范围
- 资源、进度、设备、工具和责任

6.1.2 确认计划中要考虑的因素

确认测试活动包括单元测试（由开发者完成）、集成测试（由开发者完成）、可用性测试、功能测试、系统测试和验收测试，与之相应的工作有将全部确认活动看成一个整体而进行的高层计划，以及测试件结构设计。

对每个确认活动，我们必须做的有：

- 详细计划
- 测试件的设计和开发
- 测试运行
- 测试评价
- 测试件维护

确认计划中要考虑的问题有：

- 测试方法
- 设备（用于开发针对测试运行的测试件）
- 测试自动化
- 测试工具
- 支撑软件（开发和测试共享）
- 配置管理
- 风险（预算、资源、进度和培训）

（关于计划及上测试计划的更多细节将在第 10 章中讨论）

6.2 软件工程成熟度和 SEI

软件工程研究所（SEI）是联邦拨款的研究和开发中心，由美国国防部主办，隶属于卡内基-梅隆大学（Carnegie Mellon University）。它是由国会于 1984 年创立的，主要解决两方面的

问题，一是受过专门培训的软件专业人员的短缺，二是根据进度在预算范围内生产出高质量的软件。

SEI 的任务是促进软件工程技术向前发展，从而改进依赖于软件的系统质量。SEI 着眼于软件过程，认为改进过程是提高软件产品质量的关键，而改进过程意味着要将工程规范引进到软件的开发和维护中。这与本书的基本前提（软件测试过程的重要性）是相一致的。

6.2.1 SEI 过程成熟度等级

为了评估开发机构按照现代软件工程方法开发软件的能力，SEI 定义了五个过程成熟等级，它们成为过程模型的一部分，该模型被称为能力成熟度模型（CMM）：

等级 1：初始级	（混乱）	不可预测，难以控制
等级 2：可重复级	（可重复）	可重复以前熟悉的任务
等级 3：已定义级	（标准）	过程得以描述，被充分理解
等级 4：定量管理级	（度量）	过程得以度量、控制
等级 5：优化级	（优化）	着重过程改进

很少有组织达不到一级的，该级的特征是具有不可预见、难以控制的过程。要达到二级，必须能重复以前熟悉的任务，换句话说，必须有一套办法实施项目控制、项目计划、评审和交流。

由于许多测试人员的日常工作并不涉及一般的软件过程改进和成熟度，SEI 的 CMM 描述的目标和活动对于每个致力于改进日常测试过程的人员来说就具有十分重要的意义。事实上，第三级中与测试相关的活动包括：

等级 3，活动 5：按照项目定义的软件过程开展软件测试

（1）顾客和最终用户在适当的时候参与制定和评审测试准则。

（2）采用有效的测试方法进行软件测试。

（3）测试的充分性由以下因素决定：

（i）测试级别（例如：单元测试、集成测试、系统测试等）。

（ii）测试策略（例如：黑盒测试、白盒测试）。

（iii）测试覆盖（例如：语句覆盖，分支覆盖）。

（4）对于软件测试的每个级别，建立并使用测试准备就绪准则。

（5）每当被测的软件或其环境改变时，在每个有关的测试级别上进行回归测试。

（6）在测试计划、测试过程和测试用例准备使用之前，对它们分别进行同行评审。

（7）对测试计划、测试过程和测试用例进行管理和控制。

（8）每当配给需求、软件需求、软件设计或代码改变时，应适当修改测试计划、测试进程和测试用例。

等级 3，活动 6：按照项目定义的软件过程，计划并开展软件的集成测试

（1）根据软件开发计划制定软件集成测试计划，并写成文档。

（2）负责软件需求、软件设计以及系统测试和验收测试的人员，参与评审集成测试用例和测试过程。

（3）对照指定版本的软件需求文档和软件设计文档，开展软件集成测试。

等级 3，活动 7：计划并实施软件的系统测试和验收测试以表明软件满足其需求

（1）尽早地安排足够的用于软件测试的资源，以做好充分的测试准备。

（2）在测试计划中应包括系统测试和验收测试。顾客和最终用户参与评审并批准测试计划。

（3）由一个独立于软件开发者的测试小组来制定并准备测试计划和测试用例。

（4）对测试用例建立文档，并在测试开始之前，由顾客和最终用户参与评审并批准测试用例。

（5）对于配给需求和软件需求文档的基准软件和基准文档，开展软件测试。

（6）用文档记载在测试期间发现的各种问题，并跟踪到底。

（7）将测试结果写成文档，并作为判定软件是否满足其需求的根据。

（8）对测试结果进行管理和控制。

6.2.2 过程成熟度是怎样影响测试的

非常明显，测试预算中非技术性（管理）开销的总量与过程成熟度等级总体上成反比（见图 6-1）。

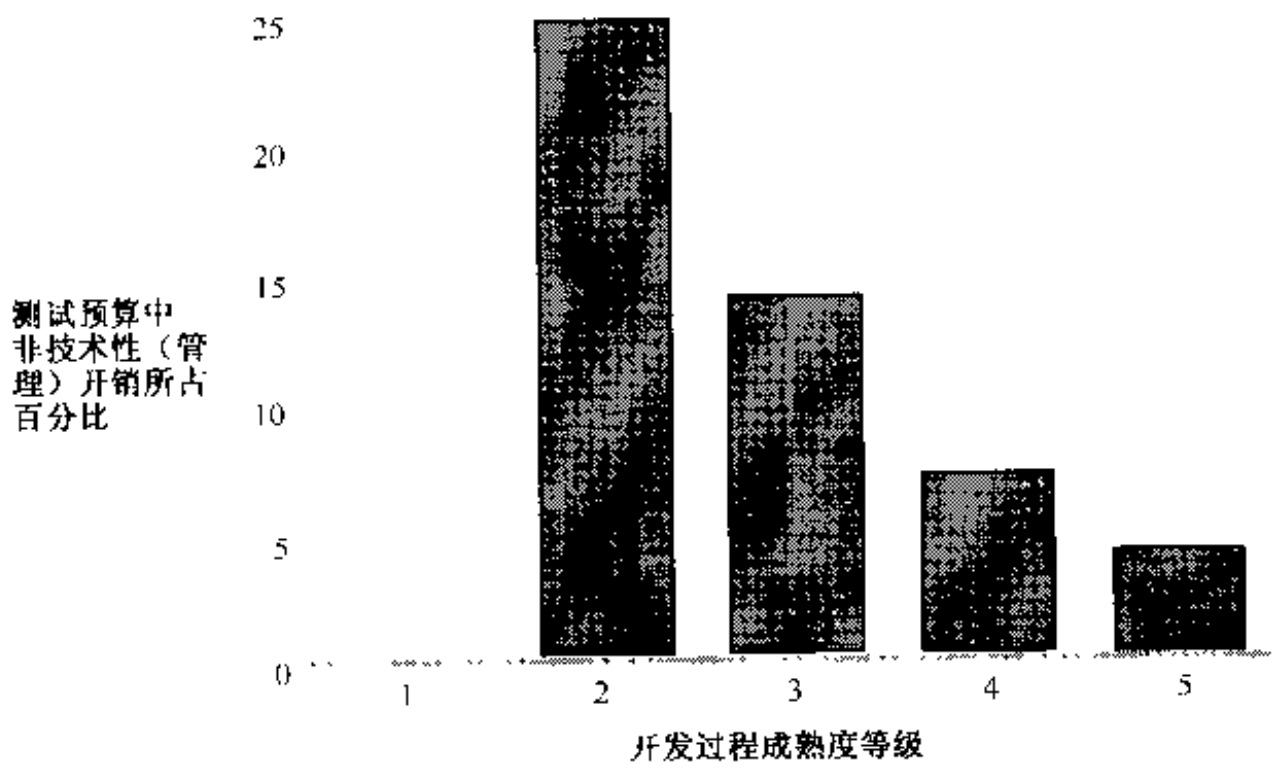


图6-1 过程成熟度等级对测试的影响（Lewis,1992）。当开发部门处于不知道下一步将走向何方时，测试部门则不能对自己的活动进行计划，于是将更多的时间花在管理问题上而不是技术问题上（© 1993，1994 Software Development Technologies）

等级 1 的组织由于其行为具有很大的不确定性，导致输出是不可预见的，甚至于开发部门不知道下一步将走向何方，测试部门不能计划自己的活动，于是将更多的时间花在处理一些非

技术性的问题上。

SEI 模型中有专门的咨询服务机构指导按要求开展软件过程评估（SPA），SPA 是由 SEI 提出的一种方法，帮助组织评估自己的软件过程成熟度，并明确要改进的关键所在。他们观察一个组织正在使用的方法和手段，并推荐一些为改进过程应该马上进行的活动。

SPA 以提问为基础，包括一系列针对特别专题的提问和在获得的信息上展开的后续提问，对问题的回答根据它们涉及内容的重要程度进行加权（参见 Humphrey, 1989）。

关于 SEI 评估的更多细节

SEI 的评价问题分为七个部分，下面是一些问题的例子，可应用于处在等级 1，以等级 2 为目标的组织。

组织：

对每个包含软件开发的项目是否具有软件配置控制功能？

资源、人员和培训：

对新任命的软件开发管理人员是否制定了所需的软件项目管理培训计划？

技术管理：

是否有办法保持对最新软件工程技术的了解和掌握？

文档标准和过程：

是否有一套正式的程序用来管理在建立合约之前的每一个软件开发评审。

过程测量：

软件人员配备概况反应的是否是实际情况而不是人员配备计划？

数据管理和分析：

是否对每个项目的评审效率进行分析？（等级 4 中的问题）

过程控制：

是否有办法控制软件需求的改变？

6.3 配置管理

配置管理（CM）问题起源于混乱和缺乏管理。这些问题可能会浪费大量时间；它们往往在最糟糕的时候发生；他们总是令人沮丧而且是完全没必要的。

如果

……我们无法识别与某目标码模型对应的源代码；

……我们无法确定某目标模型是由哪一个版本的 COBOL 编译器产生的；

……上个月被纠正的一个错误突然又出现了；

……我们无法识别在一个特定版本的软件中源代码的更改；

……多个开发人员对同一个源模型同时进行修改，而且有些修改丢失了；

……共享的源代码改变了，所有共享该代码的程序员没有得到通知；

……公共（运行期）代码的接口发生改变，没有告知所有的使用者；

……发生了变更，但对受其影响的代码版本没有作相应的变更……

……那么，所缺乏的就是配合管理。

6.3.1 什么是配置管理

配置管理是利用技术和管理的方法从以下四个方面加以指导、控制和监管：

（1）配置标识

（i）为标识所有程序文件（源程序、目标列表）和修订专用文档的基准和修订级别进行的一些常规约定；

（ii）用于识别“建立”的项目（包括源文件、目标文件、工具和所用的修订级别、数据文件等）的派生记录。

（2）配置和更改控制

（i）维护已定义和已批准的基准；

（ii）避免不必要的或无关紧要的更改；

（iii）促进有价值的更改。

（3）配置状态报告

记录和跟踪问题报告、更改请求、更改次序等；

（4）配置审计

（i）标准执行的不定期审计；

（ii）临近项目结束时的定期审计。

CM 是管理和调整变更（change）的关键（参见图 6-2），对于一个参与人员较多、变更量较大的项目它是至关重要的。概念十分简单，但实际操作却常常十分复杂。它应用于程序、测试和数据，而且非常重要，也可应用于代码和得到代码过程中的所有文档。

CM 回答诸如以下的问题：

- 什么是当前的软件配置？
- 它的状态如何？
- 我们如何控制对配置的变更？
- 对软件已经做了那些变更？
- 其他人的变更对我们的软件有影响吗？

6.3.2 在 CM 方面测试所关心的问题

测试在 CM 方面关心的问题有：

- 有效管理自己的确认测试和它们的修订级别；
- 把测试的版本与适当的被测软件的版本联系起来；
- 确保问题报告能准确地标识软件和硬件配置；
- 确保通过开发能建立正确的东西；
- 确保被测试的东西是正确的；
- 确保将正确的东西交付客户。

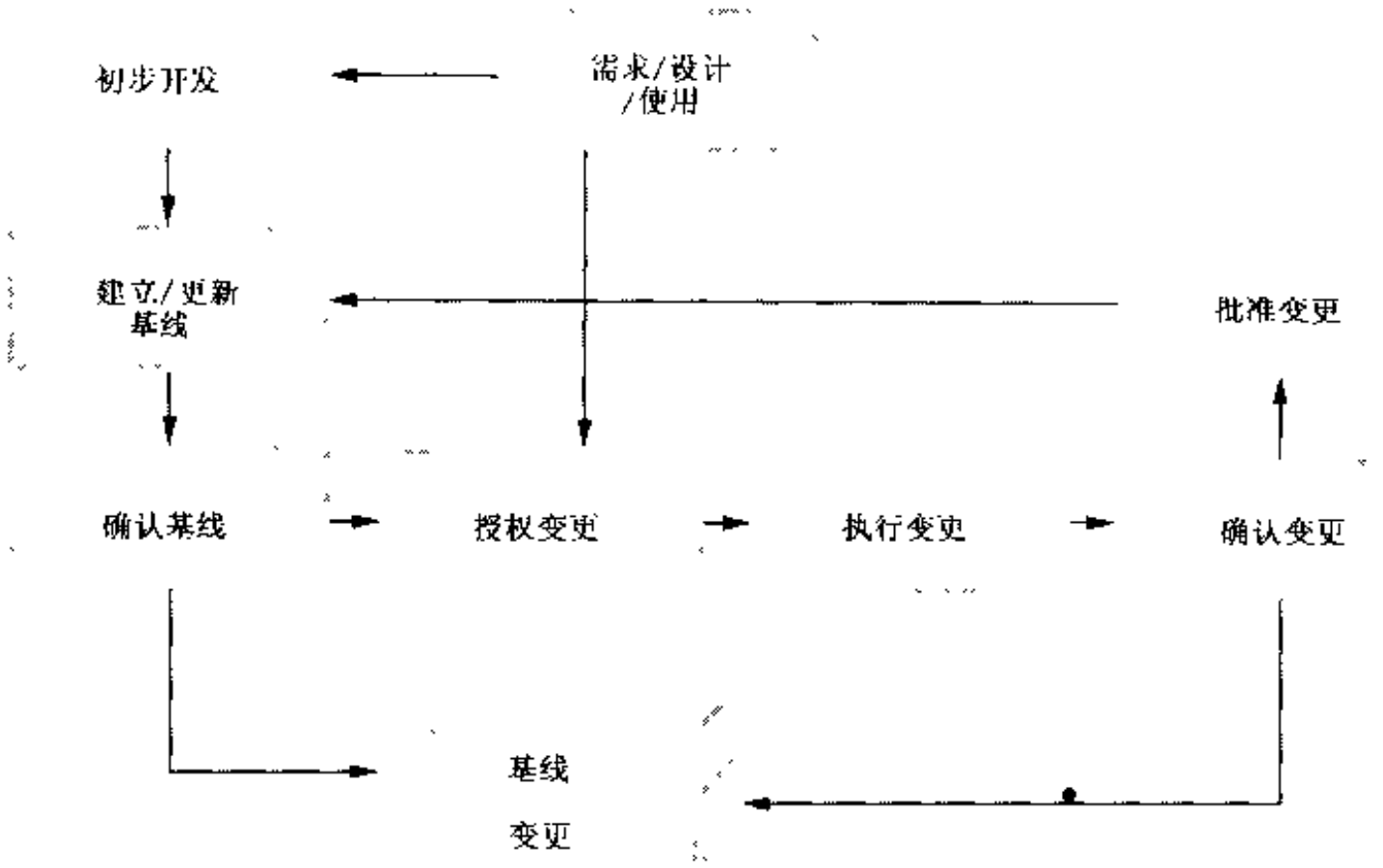


图6-2 CM概括（© 1993.1994 Software Development Technologies）

以上测试所关心的问题很重要，因为我们必须清楚地知道下面所发生的几件事情：我们正在测试的东西对吗？从谁那里得到我们测试的东西？如果我们对它进行测试，而且通过了测试，我们怎样知道它正是最终要交付客户的？怎样控制对它的变更？何时让人做变更，他们又是在何处做的变更？怎样确保进行更改采用的方式不至于产生负面影响？变更由谁控制？这些都是CM的问题（参见 Beresoff,1980; Humphrey, 1989; IEEE/ANSI, 1987[Std 1042-1987], 1990[Std 828-1990]）。

有专门讨论配置管理的课程和书籍，其基本活动就是标识系统的组成成分，并跟踪它们的状态。重要的是这不是一个特别的问题，而是用于指导自身的学科，成为指导我们“在这个组织中应如何工作”的一部分知识。它和其他致力于过程成熟度的实践一起得到初始化、维护、升级和改进。

6.4 标准

为什么需要标准？使用标准能简化交流，提高相容性和一致性，消除对同一个问题的不同解答（它们常常是不同的、甚至是矛盾的）。无论是“官方”的标准，还是仅仅只是达成的共识的标准，在我们与客户和供货商交谈时，都是非常重要的，但是当我们在组织内部的各部门之间讨论问题时，往往容易低估标准的重要性。标准同时也提供了至关重要的延续性，可避免重复性工作。标准可用来保留经过检验的实践活动，以弥补组织内部不可避免的人员变动。

有些标准对测试实践者尤其重要，对于像需求之类的文档，它们能提供一个基准，使得测试人员和做确认工作的人员能按一定框架寻找他们期望的东西。更明确地，标准将告诉我们什么是关键的测试文档，例如测试计划。

标准不仅是开发的实际需要，还是各种合同的基础，在不得已时它还是诉讼的基础。诉讼中提到的问题之一就是软件是否是按照当前工业部门通用的标准开发的，这就意味着我们不

仅需要知道标准是什么，而且需要清楚如何应用它们。

6.4.1 IEEE/ANSI 标准

许多与软件测试有关的重要的标准是由电气与电子工程师协会（Institute of Electrical and Electronics Engineers（IEEE））制定的。IEEE 是一个创建于 1884 年的跨国组织，由遍布全球的许多专业协会组成。软件测试标准由 IEEE 协会的技术委员会和 IEEE 标准部门的标准办调委员会发布。

这些标准经过一个获得实际专业人士共识的过程后产生，这个过程包括详细的讨论和不同委员会的成员之间的争论，它是标准产生的重要环节。另一个重要环节就是适时地提供标准，从项目批准到标准批准大约需要三年。

重要的美国软件测试标准

- IEEE 软件测试文档标准，Reaff. 1991（IEEE/ANSI Std 829-1983）
- IEEE 软件单元测试标准，Reaff. 1993（IEEE/ANSI Std 1008-1987）

其他有关软件测试的标准

- IEEE 软件验证和确认计划标准，Reaff. 1992（IEEE/ANSI Std 1012-1986）
- IEEE 软件评审和审计标准（IEEE/ANSI Std 1028-1988）
- IEEE 软件质量保证计划标准（IEEE/ANSI Std 730-1989）
- IEEE 软件工程术语词汇表标准（IEEE/ANSI Std 610.12-1990）

最早的也许仍然是最重要的软件测试标准之一，软件测试文档标准（Standard for Software Test Documentation）最初于 1982 年被 IEEE 批准，1983 年被 ANSI 批准，并于 1991 年被重新确定。该标准描述了基本软件测试文档的内容和格式，测试人员经常首先以它为基础和模板形成一些重要测试文档，比如测试计划、测试设计规格说明和测试总结报告等。

其他的重要标准，如软件单元测试标准（Standard for Software Unit Testing），详细说明了软件单元测试的系统化方法，为单元测试的具体实施提供指南。标准按层次给出了一个测试过程所包含的阶段、活动和任务，定义了每个活动的最小任务集合。

在每个文件的题目中，IEEE 都会使用以下三个名称中的一个：

（1）标准，表示“必须使用”；

（2）推荐实践，表示“应该使用”；

（3）指南，表示“斟酌使用”。

这些标准提出的一些反映软件测试的实际状态的建议，通过标准可以从有关专家那里得到非常有价值的忠告，这些专家潜心研究多年，所研究的正是新旧软件工程都需要面临的关键问题。

许多被 IEEE 批准的标准，还要继续被美国国家标准协会（ANSI）批准为标准，它是美国非官方标准系统的协调机构。当标准被证明得到了多数人的一致认可后，ANSI 将批准它为美国国家标准。ANSI 保证那些可能受到实质影响的利益集团有机会参与标准的制定和条款的解

释，他们的意见能得到很好的考虑。

更多的关于美国标准的细节

IEEE/ANSI 软件工程标准可以通过 IEEE 购买。

IEEE 的联系方式：

IEEE

445 Hoes Lane

PO Box 1331

Piscataway，NJ 08855-1331 USA

Freephone：（1）800 678 IEEE

From outside USA：（1）908 981 0060

IEEE 计算机协会中涉及软件工程的两个主要专业分会是：

（1）软件工程标准专业委员会（SESS），负责批准和重新确定软件工程标准

（2）软件工程技术委员会（TCSE）

联系方式：

Elliot Chikofsky

PO Box 400

Burlington，MA 01803 USA

软件工程标准的发展还远未结束，不仅新的标准在不断的增加，而且现有的标准也是至少每五年需要重审一次。

6.4.2 ISO 9000、SPICE 及其他标准

ISO 9000 系列标准专门针对组织的质量管理系统，ISO 9001 是质量管理国际标准的基础，ISO 9000-3 是一本讨论如何将 ISO 9001 应用于软件的指南。TickIt 是英国的认证体系，用于对按 ISO 9001 生产软件的组织进行认证。

由于不同国家的不同部门一直在致力于制定不同的标准，有人担心有些组织是在设置贸易障碍，而不是在促进国际合作。一家公司也许改进自己的过程以通过某一个模型的认证，过后却发现新的市场要求按另一套标准进行评估。已经有人担忧 ISO 9000 在提高产品质量上起不了多大作用，而且极少能用来改进软件过程。

WG10：软件过程评估，是国际标准化组织（ISO）中的一个重要工作小组，与 ISO 9000 无关，该小组开始了一个称为“软件过程改进和能力确定（Software Process Improvement and Capability Determination）”（SPICE）的项目，制定了一套相关标准和指南，目的是发布不同国家和不同组织都能使用的软件过程评估标准。软件工程研究所（SEI）与这个工作小组密切配合，能力成熟度模型（CMM）就是该所对 ISO 的贡献。

有关其他标准的细节

SPICE 的联系方式：

Software Engineering Institute
Carnegie Mellon University
Pittsburgh，PA 15213-3890
Tel：（1）412 268 5813
被国际同行互相引用的已出版的英国标准列在英国标准协会（British Standard Institute）的“标准目录”中
联系方式：
British Standards Institute（BSI）
Linford Wood
Milton Keynes MK14 6LE
Tel：（44）1908 221 166

书中的附录 A 列出了所有当前流行的有关软件工程的 IEEE/ANSI 标准，并对每一条标准做了简短的评述，另外还包含了有关 ISO 9000 和 SPICE 的出版物的一些细节。

6.5 正式文档

开发人员（和测试人员）最为头疼的事情也许就是文字工作了，但它对于任何成功的项目都是至关重要的。正式文档的格式、整体的内容和时间安排都是事先规定好的，是可交付的。

将决定写成文字需要文档，书写的过程中需要成百条细致规定，帮助我们将自己的思想明确地表达出来，文档还是与其他人交流的媒介物，使彼此的交流具有一致的基础。开发和测试小组提交的文档是管理部门的重要输入，便于管理部门理解和把握他们的工作。

软件生存周期中的每一个阶段都需要交付一个或多个文档，开发部门除了代码以外，还需交付三份重要文档：

（1）需求规格说明　不要在批准后就抛弃它们，有些组织的确是这样做的。

（2）功能设计规格说明　不要抛弃功能设计规格说明，而开始将用户手册作为外部接口的权威性说明，有些组织的确这样做。

（3）内部设计规格说明　必须坚持开发部门提交一份内部设计规格说明，有些组织常常不这样做。

这些文档对于一个成熟的开发组织来说是必需的，而且要维护它们，直到产品退出使用。还需要有适当的配置管理，任何变更都应该在所有受其影响的文档中有所体现，这些文档的变更是受控的，并且要与代码的变更同步。

测试工作在自身的生存周期中也将形成相应的正式文档，他们都将包含在测试件（testware）中。

6.6 测试件

一个组织交付的产品是其工作的重要体现。硬件开发工程师生产硬件，软件开发工程师生产软件，软件测试工程师生产测试件。

测试件是通过验证和确认两种测试方法生产出来的，测试件包括验证审查单、验证错误统计、测试数据、以及测试计划、测试规格说明、测试过程、测试用例、测试数据和测试报告等支撑文档。

之所以取名测试件是因为从它开始使用起就有了自己的生命，像软件一样，它应该处于配置管理系统的控制之下，得到保存和忠实的维护。与通常的看法不同，测试并不是匆忙中一带而过的活动，在完成后不留下任何有形的东西。同软件一样，测试件也是可以重用的，而且每次使用不再产生再开发的费用，具有重要价值。

像软件和硬件需要维护一样，测试件的维护也十分重要。生成测试件是测试工作的重要组成部分，测试件有指定的寿命，可以控制和管理，是公司的宝贵财富。一旦测试件被建成，它就必须受到某种控制而不至于丢失，如果某一测试人员离开了公司，其他人完全可以维持并继续其过程。

给测试件命名，有助于消除测试组织没有自身产品的传统误解，同时也赋予测试人员对自己所做工作的所有权。专业测试人员有自己的产品，有自己的测试库、代码和软件，测试人员的特别交付就是测试件。测试件的概念使得测试人员之间、组织之间的交流更便捷。

测试件的引用将贯穿全书。尽管测试件管理与查找错误没有直接的联系，但它对测试效率有很大影响。第 7 章将给出有关验证测试件的更多细节，第 9 章将详细讨论生存周期中每一阶段的测试交付及测试件管理。

6.7 度量

没有度量，就不可能知道我们是在前进、倒退还是原地踏步。SEI 软件过程评估调查表的主要部分用在研究过程测量上，这并不是巧合。

过程测量问题实例

- 是否搜集了软件错误的统计数据？
- 为测试小组保留了实际与计划测试用例的数据图表吗？
- 是否对由测试暴露的软件问题追踪到底了？
- 是否度量并记录了设计和代码审查覆盖率？

成熟的开发和测试组织对过程中的所有重要元素都能提供测量标准，这不仅是因为他们需要对其性能、资源的使用、以及工作效率进行监督，在此基础上做出决策，而且还因为在任何层面上，即使提出有关统计数据采集的问题也有助于过程改进。

尽管度量很容易让人觉得是不必要的俗套，但测试方面有一些重大的问题只能通过在组织内开发度量才能获得答案，从而使组织在未来作出更合理的决策。

度量可以告诉我们什么

- 对于特定产品，测试的实际工作量有多大？
- 我们的验证工作效率有多高？
- 我们的确认测试有多全面？

- 产品的质量如何：
  ——在测试过程中（客户使用之前）？
  ——在客户的产品使用之中？
  ——与其他产品相比？
- 产品（大约）有多少错误：
  ——在测试开始之前？
  ——与产品接触一段时间之后？
- 我们何时停止测试？

度量能告诉我们程序复杂度的实际情况。它有助于计划测试工作、预测错误的数量以及错误的位置。对检测出来的错误的数量和类型的度量，可以使我们准确掌握验证的效率以及开发过程中的不足之处。

对确认测试覆盖的度量可以提供确认测试彻底性和综合性的定量评估。跟踪测试执行状态能监视测试类的汇聚点并提供何时终止测试的定量信息。

度量并跟踪事故报告（按严重程度）：

- 提供产品质量的一个主要指标；
- 提供产品发布的重要标准；
- 与用户对产品的满意程度有关；
- 作为剩余错误数量的一种预测；
- 规范化后，提供产品质量相对于其他产品的度量；
- 能提供对测试效率的度量（客户报告的事故与测试报告的事故比较）。

记住：

“不能度量的东西不能控制。”

——Tom DeMarco, 1982

（关于有益的度量及其实施的详细情况，见第 12 章。）

6.8 工具

与其他工具一样，测试工具能提高工作效率。现在有各式各样的工具，在测试过程的每一阶段为我们提供帮助。如果想充分利用好工具，有几个问题值得一问：

（1）工具怎样介入并支持我们的测试过程？

（2）我们知道怎样计划并设计测试吗？（工具是替代不了思考、计划和设计的。）

（3）谁负责实施新工具的培训工作？

（4）谁在组织内部不断地推广并支持工具的使用？

换言之，象所有的测试活动一样，工具的使用必须与开发过程的全局相配合。

“简单地将工具摆在测试问题面前是解决不了问题的。”

——Dorothy Graham, *The CAST Report*

工具的分类方式包括：

- 按使用工具的测试活动或任务（如：代码验证、测试计划、测试执行）；
- 按功能描述关键词，换言之，就是工具执行的具体功能（如：捕获/回复、逻辑覆盖、比较器）；
- 按分类的主要领域，换言之，工具的少数几个高层分类或组合，每一类包含的工具在功能或其他特征方面相类似（如：测试管理、静态分析、仿真器）。

为适合本书的编写，并且为了强调测试具有其本身的生存周期，在第 11 章中工具将按以下标题进行讨论：

- 评审和审查
- 测试计划
- 测试设计和开发
- 测试执行和评估
- 测试支持

对工具的了解与有效地评估、挑选并实施这些工具是有区别的。无论开发过程的成熟度有多高，实施正确的工具挑选和评估策略是很重要的。（关于工具、工具的分类、工具的获得和实施的详细情况，参考第 11 章。）

6.9 现在能做什么

- 在组织中提高对标准的认识，复制一两份关键的标准，从软件测试文档标准着手，利用这些标准帮助我们书写测试计划，这方面的获益将是显而易见的。
- 对组织中工具的使用情况进行初步评价，是否已最大可能地发挥了作用？至少有一条通道来听取专家对工具的建议和意见。
- 获得一组 SEI 成熟度评估问题集，并将其中的子集应用到组织的相关部分，即使是问些问题也是有益的，它可使你清楚地了解到自己所处的位置以及迫切要做的事情。认真考虑一下如何从过程评估专家那里获得独立的帮助。
- 在组织中提高对测试件的认识，它是应该得到承认、培育、维护和发展的主要资产。
- 加深对配置管理的认识，组织中有专人负责吗？如果你的组织还没有正式实行配置管理，先在你自己的领域里非正式地使用，并将获得的收益展示出来。

6.10 参考文献

Beresoff, E.H., Henderson, V.D. and Siegel, S.G.（1980）. "Software configuration management: a tutorial, "*IEEE Tutorial: Software Configuration Management*, IEEE Cat. No. EHO 169-3, 27 October, 24-32.

DeMarco, T,（1982）. *Controlling Software Projects*. Yourdon Press.

Humphrey, W.S.（1989）. *Managing the Software Process*. Reading, MA: Addison-Wesley.

IEEE/ANSI（1987）. IEEE Guide to Software Configuration Management,（Reaff, 1993）, IEEE Std 1042-1987.

IEEE/ANSI （1990）. IEEE Standard for Software Configuration Management Plans, IEEE Std

828-1990.

计算机科学技术报告和 SEI 技术报告的宣传机关：

Research Access Inc.

3400 Forbes Ave., Suite 302

Pittsburgh, PA 15213

Tel. （1） 800 685 6510

另见：

Software Engineering Institute, Tel. （1） 412 268 7700

第三部分

测 试 方 法

“一鼓作气，一竿子到底”

——中国谚语

第 7 章 验证测试

第 8 章 确认测试

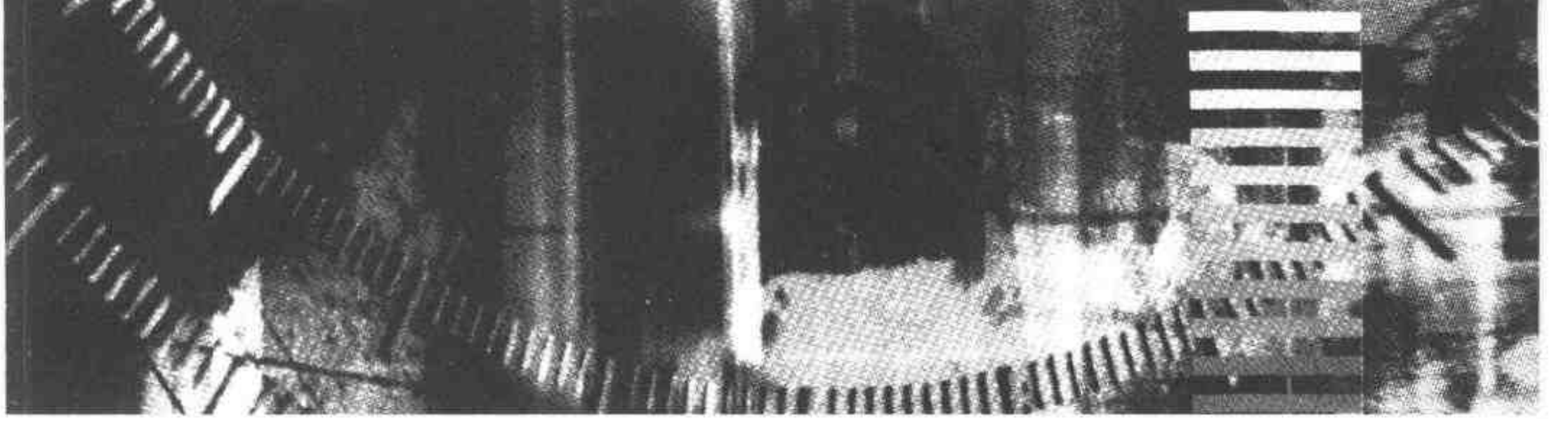
第 9 章 控制确认成本

第 10 章 测试任务、可交付文件及其在生存周期中的对应阶段

第 11 章 软件测试工具

第 12 章 度量

-



第7章

验证测试

验证活动是测试生存周期中的一个阶段，在每个验证活动中测试的目的都是为了发现尽可能多的错误，测试小组应积极参与开发人员主持的审查和走查，并开展验证工作，在开发的早期尤其如此。

另外，测试组应该开发自己的验证“测试件”，对于各种类型的文档形成通用的、测试特有的审查单，使得验证技术成为组织的财富。当然，测试件本身就像其他所有软件一样，也应该得到验证。

测试应该利用验证实践进一步改善对基础理论的跨学科交流，同时普遍提高开发环境的成熟度。

应该开展尽可能多、尽可能深入的验证，实践证明，无论是从短期还是长期来看，验证都是一条最可靠、效益最高的质量改进之路。

7.1 验证的基本方法

验证是对工作产品进行人工检查或评审。评审的方法有：审查、走查、技术评审等等不一而足，所指也不尽相同，但一般认为审查是最正规的方式。

验证方法的基本特征

	审　查	走　查	伙伴检查
主持人	非该软件的编制人员	任何人	没有
参与人员	3~6 人小组	多一些人	1～2 人
准备	有	只有主持人	无
数据收集	有	不要求	无
输出报告	有	不要求	口头评论
优点	有效	能使更多人熟悉产品	费用低
缺点	短期成本	查出的错误较少	查出的错误较少

7.1.1 验证的组织形式

正式的评审、技术评审以及审查是用于表达更加结构化的验证的几种不同说法，在以下各节中，我们提到最多的是审查，就是因为审查是最结构化的，但这并不意味着将其他方法排除在外或怀疑其他方法的价值（见图 7-1）。

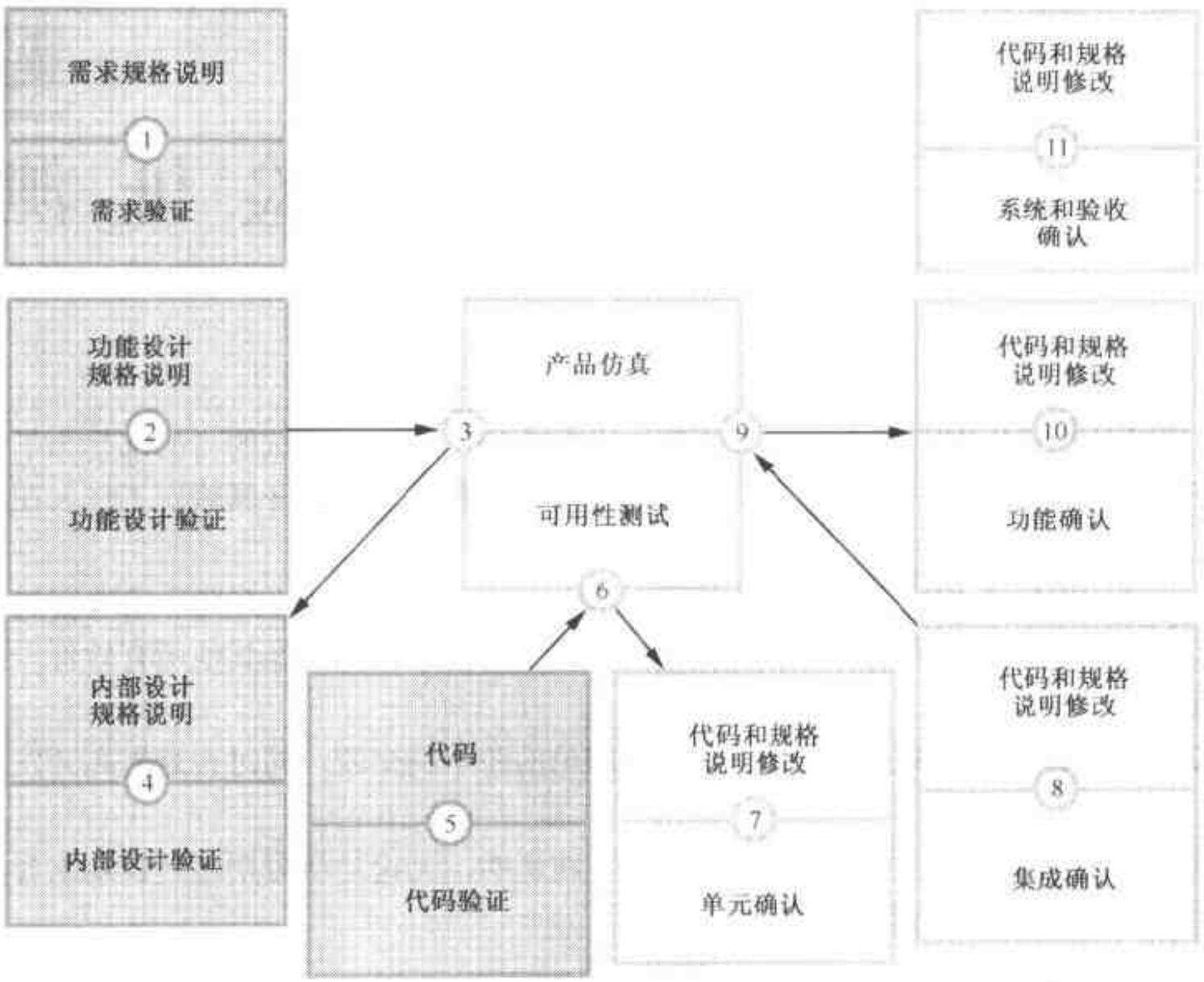


图7-1 SDT（软件开发技术）Dotted-U模型，验证测试（© 1993,1994 Software Development Technologies）

这些方法的中心“事件”是举行的一次会议，会上对受审产品的错误进行讨论和确认。

正式方法——主要因素

（1）评审小组的每个成员以开放、积极的方式参与活动，通过传统、惯例和评审活动的书面规定对评审的参与进行管理。

（2）提供一份关于产品状态的书面报告，与项目有关的每个人，包括管理人员，都可以获得报告。

（3）评审小组对书面报告内容的质量负责。

（Freedman and Weinberg,1990）

审查的一个特点是在开会之前，所有的参与人员应做好充分准备，审查组负责人给参加者布置任务，告诉他们各自应扮演的审查角色，以及阅读资料时应重点注意的地方。如果审查工

作开展得很成功，大多数错误是在准备阶段被发现的，而审查会议通常会发现一些重大缺陷。

7.1.2 审查：关键元素和阶段

目标：

- 发现缺陷，收集数据；审查会不对供选择的解决方案进行评审；
- 交流工作产品的重要信息。

元素：

- 定期的、有组织的会议，要求每个参与者事先做好准备；
- 3~6 人小组，一个公正的领导；
- 主持人是“阅读者”而不是生产者。

输入：

- 被审查的文档；
- 有关的原始资料；
- 一般的和“裁剪的”审查单。

输出：

- 审查汇总/报告；
- 有关错误类型的数据。

关键阶段：

- 简要介绍/开始—个人准备—审查会议—校订/重复工作—结束/再审查—收集数据；
- 有些审查还包括一些非正式的分析会议。

审查会议的主持人不能是生产者或被审查文档的编写者，这样不仅可迫使一些文档编写者之外的人员深入阅读材料，而且可使会议参加者听到更多的不同声音，会议上其他的审查人员将提出一些不同的观点。

无论参加何种形式的评审，测试人员对产品所持的观点与开发者是不同的，“它将如何工作？如果以这种方式测试，将发生什么？”作为测试人员，在测试过程中要不断地考虑将发现什么问题，注意力往往集中在那些一般不容易被注意到的地方，这就是为什么测试人员成为审查和评审过程的一部分是如此重要的原因——他们给会议带去独特的声音。

任何重要的工作产品能够也应该被审查——项目计划、工程绘图、用户手册、测试计划等。附录 B 包含了一个可用于任何文档的一般审查单。

7.1.3 走查

由于缺乏准备，走查不像审查那么正式，参与者只是简单地参加会议，由主持者（一般是产品的作者）准备，参与人员在会议前不需做更多的准备工作。

7.1.4 走查：关键元素

目标：

- 发现缺陷，熟悉材料。

元素：

- 定期的会议，只有主持人必须准备；
- 2~7 人小组，由生产者或作者领导；
- 主持人通常是生产者。

输入：

- 被检查的元素，可使用的标准。

输出：

- 报告。

因为主持人是生产者，其他参加人员又没有多少负担，使得走查的覆盖面可以比审查和评审都大，从而给更多的人提供了一个熟悉了解材料的机会。

有时，走查不以发现错误为目的，而只是为了交流。软件可能是“继承”来的，我们并不确切地知道它到底是什么，于是组织走查，把一些人关在屋子里，对它逐页进行通读。如果能发现一些错误那是最好，但我们的主要目的是使自己更加了解产品。

由于走查的主持人是生产者，使得检查的客观性受到影响，这是走查的不足。

从发现问题的角度出发，审查中参与者的准备工作以及正式评审往往更能探测到问题的本质。

7.1.5 伙伴检查

任何形式的测试，哪怕是毫无方法可言，只要是由作者以外的其他人来做，而且以发现错误为目的，那就总比不测试要强的多。当不合适或不可能得到要审查或检查的材料时，也许这时就有理由不测试了。

无论如何，就算简单地将文档交给别人，要求他们仔细阅读，偶而也能发现几个我们自己无法发现的错误。有研究表明这种桌面检查可能十分有效，发现产品中的错误而不是人的错误，桌面检查不失为这方面的一种极好训练。

7.2 发挥验证的作用

7.2.1 验证要做什么

我们必须问自己：

- 通过这些评审到底发现了多少错误？
- 有多少是由评审发现的？又有多少是在后面的确认测试中发现的？
- 有多大比例的错误留到了测试结束，由客户发现的？

进行验证活动的顺序是：工作产品规模小的在先、大的在后，简要的在先、详细的在后，验证成本低的在先、高的在后，潜在的收益大的在先、小的在后。这意味着如果资源或进度的限制不能进行某些验证活动（而且总是会出现这种情况的），那么省去验证活动的顺序与上面要验证的顺序相反。

需求验证最有可能为软件开发节省成本。它可以检测出许多不足，否则这些不足会进入开发周期的后期，那时再纠正不足代价就要高得多，因为问题转移到了其他阶段。另外，50%以上的缺陷实际上都是在需求阶段引入的。

一般来说，我们建议进行正式的评审和审查。它们的短期费用较高，但只要处理得当，收益总是会超过成本。它们会检测出很高比例的错误，而固有数据收集方法是开发过程一种十分有用的度量。这些方法的成败与效益是可以度量的。

由于审查很费时而且要求精力高度集中，它们一般处理量相对较小的材料。像所有的测试活动一样，大型工作产品的验证是不可能穷举的，因此一般需要进行风险评估、做出取舍。

最好能将验证方法进行“混合和匹配”。有时人们这样说也是合理的：“这部分代码最为关键，是系统的心脏，因此我们要仔细进行审查。我们要挑选几个人，对规格说明的每一行都进行审查。”对重要文件进行采样也是很有用的。对样本进行审查可以估计文件的质量以及未审查部分包含错误的数量。

另一方面，我们也可以说：“这部分代码不太重要，不是系统的心脏，用不着审查。”对于这种情况，我们进行不太正规的评审、桌面审查、走查或这些形式的变体。取舍是不可避免的，风险分析就可以发挥作用了（见第 5 章）。

验证的效果总的来说要高于确认测试。它可以发现一些在确认测试过程中几乎不可能检测出来的缺陷。最重要的是，利用验证我们可以尽早发现和检测出缺陷。

在大多数组织中，验证/确认的缺陷分配为 20/80，验证甚至更低。作为一般策略，我们应该加大验证的比例，这主要涉及观念的改变。找一个容易很快出结果的领域进行验证，以便获得人们的支持，激起他们的热情。然后再往前走一步，慢慢地改变比例。

7.2.2 审查单：验证工具

审查单是验证的重要工具，尤其是对于像审查一类的比较正式的验证。针对各种类型的验证都有相应的审查单，例如，有需求审查单、功能设计规格说明审查单、内部设计规格说明审查单、测试计划审查单等。对于一切要进行检查的项目都可以开发相应的审查单。

一般的审查单

在附录 B 中包含了以下审查单：

- 需求（规格说明）验证审查单
- 功能设计验证审查单
- 内部设计验证审查单
- 一般代码验证审查单（1）
- 一般代码验证审查单（2）
- “C”代码验证审查单
- “COBOL”代码验证审查单
- 一般文档验证审查单

开发与测试部门常常有不同的审查单，测试用的审查单更趋向于面向产品的可靠性和可用性，而开发用的审查单通常更加注重可维护性，以及代码标准指南之类的事情。

重要的是利用一般的审查单，并在其基础上针对特殊目的和特别项目开发属于自己组织的审查单，这些审查单应该有明确的目的，并充分反应组织在验证测试方面现有的成熟水平。

审查单是测试件的重要组成部分。为了最大程度的发挥验证的作用，它们应得到认真的保管、改进、发展和更新——并且有专人负责。它们是验证测试的重要工具；是培训的重要手段；可确保不同项目和不同人员开展的验证工作得到延续；能记录组织的验证工作前进的步伐。

7.3 验证不同阶段的文档

7.3.1 验证需求

在需求阶段，要保证用户的需要在被转化为设计之前能得到完全理解，由于很难将需要和想要区分出来，因此开发需求十分困难，而且在以后的阶段中它们还会不断变化，一直以来需求都是软件链中最薄弱的一环。我们说到需求到底意味着什么？对需求验证时要提些什么问题？

用户需要的能力是什么？我们试图为用户提供什么？用户需要什么？问题的答案就是需求要陈述的。IEEE/ANSI 的定义为：

需求是用户为解决一个问题或达到一个目标所需要的一种能力条件。

需求是一个系统或系统组成部分为满足一个合同、标准、规格说明、或其他正式规定的文件所必须达到或具备的能力条件。

需求可以是正式编制的文档，也可以是明确用户需求的非正式通信，可以是明确的，也可以是隐含的，但总是存在的，这个阶段的测试工作的目的就是保证在开展下一步工作之前能完全吃透用户的需要。

面临的一个困难是严格将需求与它们的解决办法区分开来，在开发过程早期产生的文档中常常将二者混淆，另一个困难就是要清楚的认识到，即使是在最好的情况下，需求也是要不断变化的。

早期，作为测试人员我们习惯说，“必须冻结需求，必须达成需求不能更改的协议，而且要保证不改。”然而，这是完全不现实的，因为实际项目的需求就是要变化的。我们要做的是通过需求验证和各种形式的配置管理（见第 6 章）来管理和控制它的变更，为以后的测试打下坚实的基础。

7.3.2 好的需求规格说明的特性

以下是一个好的需求规格说明应该具有的特性：

- 明显
- 清楚（不含糊）
- 完整
- 前后一致（必须对相互矛盾的需求做出取舍）
- 合理（可实现）
- 可度量（可量化、可测试）
- 可修改

- 可跟踪
- 相关的需求明确

IEEE/ANSI 软件需求规格说明指南是一份很有用的文件，它能帮助客户确定他们想要的东西；它能帮助提供商掌握他们应该提供什么，它能帮助与需求打交道的人们了解他们要达到的目标。

IEEE/ANSI 提供的帮助

IEEE/ANSI 标准 830-1993，软件需求规格说明推荐实行（Recommended Practice for Software Requirement Specifications），可以帮助

- 软件客户准确描述他们想要的东西；
- 软件供应商准确把握客户的需求；
- 个人实现以下目标：

——为他们各自的组织开发标准软件需求规格说明；

——为具体的软件需求规格说明规定形式和内容；

——开发其他的本地支持项目，如需求质量清单或需求编写人员手册。

需求文件的质量往往是生产者、其技术、他们是否知道怎样撰写需求以及他们是否熟悉标准的函数。需求的质量是组织成熟度水平的重要标志之一。如果没有生存周期，如果没有对需求文件的必要性达成一致意见，没有就需求文件的撰写者、内容、甚至可接收的需求框架达成一致意见，要进行开发是很困难的，而要进行合理的测试则是十分困难的。

我们对需求进行评审时，应该注意基本功能，但不能局限于此。譬如，我们需要一些定义。需求往往包含一些重要的术语，而每个人都理所当然地认为，它们的意思我们是清楚的。

需求审查单——项目示例

以下内容选自一份通用需求验证审查单：

- 准确、不模棱两可、清楚

每个项目准确不含糊；只有一种解释；可以理解每一项目的意思；规范容易理解。

- 前后一致

规格说明中的前后项目不矛盾。

- 具有相关性

每个项目与问题及最终的解决方案具有相关性。

- 可测试

在项目开发及验收测试期间，应能确定该项目是否得到满足。

- 可跟踪

在项目开发及测试期间，应能在开发的不同阶段跟踪每个项目。

附录 B 提供了一份更完整的需求验证清单。

需求可能会经常涉及到一些重要和基本的问题，像安全性、可用性、可维护性、以及一些不明确或不好解释的性能。当我们从可测试性角度出发阅读需求时，所有的含糊不清都要被问

个究竟。一个“很好的性能”到底由什么构成，对此很难测试，我们所能做的是回过头去问：“你用“很好”一词意思是什么，是 2 秒的响应时间，还是 24 小时，或者其他？”可测试性的概念很接近于可度量性，一般来说，需求越是可量化，对其最终导出系统的成功而进行的测试就越简单。测试人员也可给需求编写者提供一个框架，告诉他们哪些是必须要有的，比如性能规格说明、可用性的测量、可维护性的标准等等。

对于需求，测试者的立场是找出那些可能是问题的地方，它是清楚、完整、连贯、合理的吗？可实现吗？可追踪吗？从测试的角度可度量吗？验证需求是一块沃土，改进开发过程以及以后的测试都将在这块土地上开花结果。一个“我该如何测试？”的提问，常常可使自己从产品设计者那获得更理想的解决办法，以及关于测试什么，如何测试方面的好建议。

需求验证练习

- 以附录 C 中预定系统的简单需求规格说明为例，从一个人的“伙伴检查”到完整规格说明的审查，充分表明需求验证的有效性，记住成功的验证是要花时间的。
- 使用附录 B 中的需求验证审查单，并增加一些在今后的文档中应该检查的新项目。
- 记录练习所花的时间。
- 如果将所发现的问题遗留到开发的以后阶段，甚至留给用户，对由此发生的时间/费用进行估计。

答案：练习后面是对文档的总评，它是四个测试专业小组对文档进行大约 30 分钟的评审后所发表的意见。

需求已被审查；变更已经完成；缺陷得到了处理；需求也已由评审组签字批准。这样，我们就为测试设计、变更谈判、以及确认打下了良好的基础。通过参与需求的验证，使我们对产品有一定的认识，知道它是怎样工作的，可以开始计划确认测试了，我们已处于赢家地位。

7.3.3 验证功能设计

功能设计是将用户需求转换成一组外部（人类）接口的过程。该过程的输出是功能设计规格说明，从外部对产品行为进行描述。对用户能见到的必须进行描述，而对用户不能见到的则应避免描述。功能设计规格说明最终转换成内部设计以及用户手册。它不得包含内部信息、内部数据结构、数据图表或流程图；它们属于内部设计规格说明，是过程的下一步。

验证功能设计与验证需求有什么不同？如果说需求是最重要的，则功能设计就是次最重要的，就是因为它处于过程的早期。

功能设计审查单——项目示例

以下内容选自一份通用功能设计验证清单：

- 如果在某个地方对一个术语清楚地进行了定义，则尽可能用该定义取代该术语。
- 如果用词语对一个结构进行了描述，则尽可能对所描述的结构画一幅图。
- 如果指定了一项计算，则动手算出至少两个例子并作为范例包括在规格说明内。
- 搜索确定性语句时，尽可能按照需要对有关层次进行搜索，以获得计算机所需要的

确定性。

- 注意模糊性词语，如一些、有时、经常、通常、一般地说、习惯上、最、或大部分。

附录B提供了一份更完整的功能设计验证清单。

功能设计验证的目的是确定用户需求是如何在功能设计中得到具体体现的，可追踪性的概念在这里开始得到运用。我们已经有了需求，这个阶段将再次使用它作为验证功能设计的原始文件，需求中的每一段都要在功能设计规格说明中得到体现，如果不是这样的话，要么是它被彻底遗漏了（何处有记录？），要么是有人忘记做了。

最常见的功能设计规格说明的失败是不完整。优秀的审查或评审人员不是仅阅读那些摆在他们面前的东西，而是不停地问："遗漏了什么？"不断地想哪些是应该写在纸上的。设想一下，要是我来写文档，我会在正读到的这部分之前包含些什么内容，应该怎样功能性地描述它们？应该如何照顾到最终的用户？通过假想自己是编写文档的设计者，可以使我们发现一些遗漏性错误，这些都是最重要的错误。

需求本身也有许多源文件，例如标准、信函、会议记录等等，它们应该是可追踪的源头。

功能设计规格说明验证练习

- 以附录C中的销售系统功能设计规格说明为例，对其进行功能规格说明验证，表明验证的有效性。
- 使用附录B中的功能设计验证审查单。

答案：练习后面是对文档的总评，它是四个测试专业小组对文档进行大约 30 分钟的评审后所发表的意见。

7.3.4 验证内部设计

内部设计是将功能设计转化为详细的数据结构、数据流和算法集合的过程，其输出是内部设计规格说明，它表明了产品是如何建造的。为体现不同的抽象层次，可能会有多个内部设计规格说明，可能的话，应该验证它们中的每一个。

内部设计审查单——项目示例

下面是从一个典型的内部设计验证审查单中抽出来的：

- 设计文档是否包含了有关初步设计的步骤的描述，或是否有该步骤的说明？
- 是否有计算机系统的用户界面的模型？
- 是否有被推荐计算机系统的高层功能模型？
- 主要实施方案及评估是否在文档中得到反映？

附录B提供了一份更完整的内部设计验证审查单。

IEEE/ANSI 在软件工程标准文档中提供了软件设计描述的推荐实践，它包括信息、格式及材料组织方式的确定。

IEEE/ANSI 提供的帮助

IEEE/ANSI 标准 1016-1987，IEEE 关于软件设计描述的推荐实践（Reaff,1993），详细说明了必须包含的信息内容，推荐了软件设计描述的编排方式，参见附录 A。

内部设计规格说明对测试人员来说是十分宝贵的，由于清楚了产品将如何建造，系统将如何集成，测试者便可以设计出更多的基于内部的测试用例。

使用审查单对内部设计进行审查，向上可追踪到功能设计和需求，看看所用的算法及组合方式是否合适，在寻找错误的同时，我们一直在思考，“到底应该如何测试？”

例如，如果在内部设计中要用到一个表格，测试人员可以问一些与测试相关的问题：“表格有多大？为什么有 25 个项目？怎样填充表格？怎样充满表格？表格为空时将发生什么？”

这样的提问将激发一些极好的测试思路。一旦开始阅读内部设计文档，对产品的各类限制就变得更加清晰，边界条件也显露出来了，它们可提醒我们注意那些可能的失效条件，以及其他各类内部需考虑的问题。从测试的角度出发，如果有内部设计规格说明，并对它们进行审查，通过审查获得的信息，可使我们对产品的认识更加深入和全面。

7.3.5 验证代码

编码是将详细设计规格说明转换为特定的代码集的过程，该过程的输出是源代码本身。许多公司常常以此作为走查和审查工作的起点，也许它是一个很舒服的起点，因为许多工作都已经完成，但它肯定不是最有效的起点。当人们准备对代码进行走查和审查，并享受其过程时，我们还得将他们拽回去检查那些在代码之前形成的文档。如果针对一个糟糕的规格说明和错误的需求进行编码，那么我们就已经浪费了大量的时间。

代码验证包括以下活动：

（1）将代码与内部设计规格说明进行比较。

（2）对照特定语言审查单检查代码。

（3）使用静态分析工具对句法规则是否被满足进行检查。

（4）验证代码中的名词与在数据字典和内部设计规格说明中的相一致。

（5）寻找新的边界条件、可能的性能瓶颈、以及其他可能需要追加确认测试的内部需考虑的条件。

有些公司进行正式的代码评审，每个参加者预先得到代码，通读它，寻找错误，一起开会，一步一步地走代码，测试人员在其中帮助找出错误，并要求得到解释。而在另外一些公司，对代码的检查也许很不正式，只是简单的伙伴检查，几个人互相交换着看代码，将错误在上面标记出来，小范围内这种方法也许有效。

代码审查单——项目示例

下面是一般代码验证审查单中的典型标题：

数据引用错误

是否引用了未说明或未初始化的变量？

数据说明错误
是否存在变量重名问题?
计算错误
目标变量（保存计算结果的变量）的分配空间是否小于右边的表达式?
比较错误
是否为了对类型或长度不一致的数据或变量进行比较而运用了转换法则?
控制流程错误
是否会过早退出循环?
接口错误
如果某个模块有多个入口点，是否引用的某个参数与当前入口点不相关?
输入/输出错误
在程序的输出文本中是否有语法错误?
可移植性
数据是如何被组织的（例如，紧缩结构）?
附录B提供了完整的一般及特殊的代码验证审查单。

我们应该总是以测试的观点思考问题，无论何时看到一段代码，我们都将考虑一些新的测试，那些仅靠阅读需求或功能设计规格说明无法想到的测试。首次运行代码就常常能发现导致失效的错误，而开发人员做直线检查通常发现不了这种错误，他们提交代码做测试，但那条路径首次被执行时整个系统就崩溃了。

如果是正式地开展代码审查，就应该使用组织中现有的审查单，或在开始时使用一般的审查单（见附录B），并在其基础上开发出定制的审查单。

7.4 从验证中获取最大收益

7.4.1 作者

通常，要将自己的产品交出来供大家检查，人们总会本能地持抵触情绪。应当本着互相协作的团队精神："我们是寻找产品中缺陷的，而不是要在某部分中发现错误来攻击个人。"作为作者，我们应该清楚地意识到由于自己所处的位置，会使得有些事情的确难以处理，但无论如何，这种形式的评审将促进我们的工作，最终获益的是我们自己。

作为审查小组中的一员，要避免对书写风格的讨论，我们是评估产品，而不是生产者。"你如果采用另一种方式编写的话，也许要好些。"这类的讨论只会转移我们的注意力，风格问题并不会对内容产生实质性的影响，只是书写的形式问题。当然，关于术语的定义或阐述的合理性问题，最终会影响内容。

无论验证采取什么样的形式，对别人都应该采取得体、理智和体贴的态度。进行适当的准备，在审查会议上提出问题，但别在会上解决问题。

如果我们在测试组织内能成功地培养出这样的态度（成为榜样则要经过长期的努力），就会使企业文化发生重大的转变。

7.4.2 开发小组

评审和审查有助于加强彼此交流，提高工作积极性，从某种意义上说，它利用了人们的荣誉感，甚至是窘迫感，试想一下，如果人们知道他们的产品要被审查，他们肯定将工作得更加出色。

反馈可能是非常有积极意义的，如果在数次评审会上，生产者／作者都能感觉到“这看起来不错，它确实是个好方法”，那么它尽管不是被正式通过，也将成为组织中普遍看好的做法。反馈能以建设性方式加强人们的表现意识，仅仅因为我们知道自己的材料将接受极为详尽的评审，在半年到一年的时间内，我们将会减少产品中的缺陷。

另一个重要的问题是交流。我们不能单纯从所发现的错误衡量审查的成功，审查将改进软件开发过程。审查的一个重要贡献就是人们开始了交流和讨论，能更多的了解对他们工作有益的东西，例如，通过加入审查小组，使测试人员开始了解产品的基本信息，审查的交流价值不容忽视。

验证促进的交流不仅是在开发队伍内部的，还包括整个组织内的，早期的验证更是这样。如果正在做需求验证，当然希望市场人员、测试人员、开发人员都能参加。验证提供了一个在更大范围内互相理解、彼此交流的绝好机会。

7.4.3 审查小组

由于没有适合于任何类型工作产品的具体方法可循，审查工作是困难的，不同种类的产品需要不同的专门技能和思维方式，需要吹毛求疵的思考方法，它甚至胜过最好的审查检查单。

审查还要求注意力高度集中，时间长了会十分疲劳。它需要查找遗漏的能力，而大多数人往往只关注眼前所看到的，最好的审查员和评审员必须要问“遗漏了什么？”或“什么是这儿应该有却没有的？”优秀的评审员，尤其是优秀的审查小组成员，是组织的宝贵财富，他们的名字耳熟能详，他们在不断地满足各类验证活动的需要。

7.4.4 高收益的验证

验证就其成就感和重要性来说，比确认测试有过之而无不及。它能够在尽可能早的时候发现缺陷，从而大大减小缺陷的成本。那么验证自身是高收益的吗？通过验证尽早发现错误，我们的节省比花费多吗？简单地回答就是验证虽然不是免费的，但确实是高收益的，这点已反复被事实所证明。

因此，我们应该尽量验证全部关键文档，或者至少验证一部分。对于大规模的工作产品，我们几乎不可能完成全部代码审查，以及正式的规格说明和设计审查，有一部分可能使用走查代替正式审查，有些项目将采用桌面检查，总之，要根据实际情况做出决策。

如果有一个很好的配置管理系统（见第 6 章），我们还可以提高验证的效益，必须有人保证我们不会在错误的时间对错误的东西进行验证，我们所验证的应该与确认测试以及最终的交付是同一个东西。

7.5 验证的三个成功因素

7.5.1 成功因素 1：过程责任人

如果我们还没有开展验证，或者希望改进自己的工作方式，那么很需要一个过程倡导者，他可以是任何人，只要对此感兴趣即可，他需要投入很大精力，对过程的进展负责。他可以是开发组织中的某个人，清楚地了解审查工作的好处，也可以是质量保证人员，可以是过程专家，或者是过程工程组中的一员，可以是全职也可以兼职。如果公司规模很大，就应该安排专人组织正式的验证过程，而对于规模较小的公司，则是既不必要也不可能的。重要的是必须有人成为方法和工作的倡导及推进者，并能在中长期内不断地得到支持。

7.5.2 成功因素 2：管理部门的支持

提倡采用审查的往往是管理部门。无论是否引入了过程的概念，使管理者对审查及其收益有大致了解是非常重要的。要是他们不清楚长期的获益情况，期望得到他们在经费及其他方面的支持是不切实际的。他们有各种压力，有各类紧急事情要处理，让他们理解将资源投入早期测试工作是很合算的，的确不是件容易的事。

你应该将审查的早期成果展示出来，以得到更多的支持，可能的话，应该收集那些早期通过审查发现的错误的有关数据，并将它们与将错误遗留到下一阶段将发生的费用进行比较。说服那些对此不十分热心的人参加一个会议，会上针对特定的工作产品展开讨论，以一个小的现场审查表明审查工作在他们组织中对他们的产品是如何有效的，让他们知道对一个新过程的收益进行有效调节是完全可能的。

7.5.3 成功因素 3：培训

评审和审查中的培训是重要的，包括对参与者如何进行评审和审查的特殊培训，还包括成本、效益、以及人事和企业文化等专题，每个将被审查其文档的人也应该受到培训，一个注重人才培养的优秀团队对新员工具有很大的吸引力。

SIRO 提供的帮助

软件审查和评审组织（Software Inspection and Review Organization, SIRO）——该组织负责交流有关基于小组的软件检查方面的新观念和新信息，促进审查和评审技术的发展，为评审和审查方面的资源和研究报告提供交流场所。

联系方式：

SIRO

PO Box 61015

Sunnyvale，CA 94088-1015

7.6 建议

首先要推荐的是审查，实践表明它是最有效、最切实可行的验证方法。审查也许是软件开

发历史上最古老的方法，已经被成功使用了至少 25 年，尽管审查技术已经得到很大发展，但你还是不会推荐你的组织使用未经考验的前沿方法进行审查。几个人在一间屋子里，对材料进行系统、客观地阅读，是审查最基本的特性。

实际上验证和确认测试之间存在交叉，在验证中我们很容易开始考虑确认测试中的问题。大多数只做功能和系统测试的组织存在“混乱带”（The Chaos Zone），这期间，将所有的代码抛给测试部门，而此前他们对要测试的东西一无所知，只能仓促上马，因为是在一个未知领域里开展工作，压力很大，而早期的验证完全可以成为测试人员熟悉产品发现错误的绝好机会。

作为开始，可以对几个关键材料进行审查，通常一个组织有各种类型的项目，它们处于不同的阶段，有些项目可能已经进行很长时间了，以至于对它们开展验证已没有意义。

最好别兴师动众，从某一时刻开始对所有文档都进行正式审查。最好的办法是首先将那些高风险的、在许多不同的项目中起决定性作用的材料挑选出来，我们的起点可能是评审所有新的需求规格说明，也可能是评审关键项目中所有新的更改的代码。如果能得到同行在度量和跟踪方面的热心支持，那么审查工作就能在更宽范围内的文档中展开。

将本章中对需求和功能设计的验证练习的结果，展示给那些对审查工作不十分了解的领导或同事，一定要包含所用的时间资源，以及对于将被发现的错误遗留到后面阶段所造成后果的估计。

7.7 参考文献

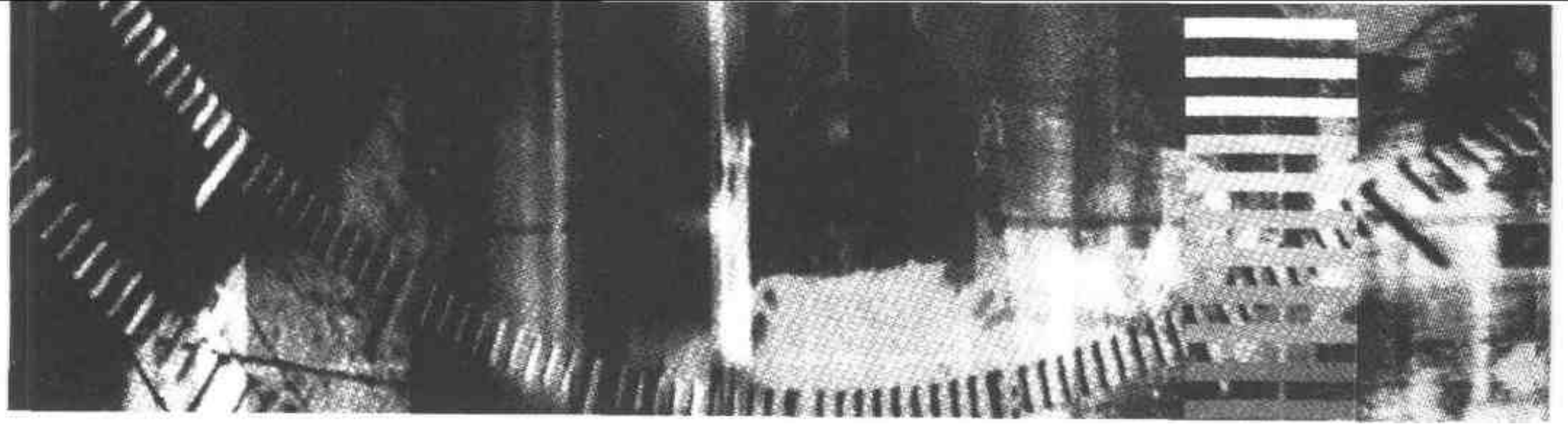
Bender, D.（1993）. "Writing testable requirements," *Software Testing Analysis & Review（STAR）Conference Proceedings.*（This provides a set of practical guidelines for writing testable requirements.）

Fagan, M.E.（1976）. "Design and code inspection to reduce errors in program development," *IBM Systems Journal,* 15（3）.

Freedman, D.P. and Weinberg, G.M.（1990）. *Handbook of Walkthroughs, Inspections. and Technical Reviews.* New York: Dorset.

Glib, T. and Graham, D.（1993）. *Software Inspection.* Wokingham: Addison-Wesley.

IEEE/ANSI（1988）. IEEE Standard for Software Reviews and Audits, IEEE Std 1028-1988.



第8章

确 认 测 试

8.1 确认概述

我们从对所有确认测试适用的八条基本原理开始：

（1）测试可用于显示错误的存在而不是错误的不存在。

（2）测试最困难的问题之一是不知道何时终止。

（3）避免使用未经计划、不能重复使用且用后即扔的测试用例，除非该程序是真正的用后即扔的程序。

（4）测试用例必不可少的一部分是给出预期输出或结果。仔细比较每一测试的实际结果和预期结果。

（5）测试用例必须考虑无效和预期之外、有效和预期内的输入条件。“无效”定义为有效条件之外的条件，并且被测程序的诊断也是如此。

（6）测试用例必须能生成理想的输出条件。经验较少的测试人员倾向于只从输入的角度思考。经验丰富的测试人员能确定生成预先设计的输出所要求的输入。

（7）除单元和集成测试以外，程序不应由开发该程序的个人或组织测试。出于成本方面的考虑，往往要求开发人员进行单元和集成测试。

（8）没有发现的错误数与已经发现的错误数成正比例。

IEEE/ANSI 的定义如下：

确认是在开发过程之中或结束时评估系统或组成部分的过程，目的是判断该系统是否满足规定的要求。

八条基本原理非常有用，但在实践当中怎样判断一个程序是否确实符合需求呢？有两个办法可以解决问题：

（1）开发可以判断产品是否满足在需求规格说明中注明的用户需求的测试。

（2）开发可以判断产品的实际行为是否按照功能设计规格说明中描述的那样与预期行为相匹配的测试。

尽管内部设计和代码是从功能设计导出的，但一般用不着对它们了解就可以判断最终产品是否符合需求。

在 IEEE/ANSI 应用中，注意“需求”这个词既包括用户需求也包括功能接口。实际上应该将它们当作两个不同的文件制定和保留。高层需求必须从客户和市场的角度进行规定，而功能规格说明应从工程的角度制定。许多公司面临的一个主要问题是在市场与工程规格说明责任之间定义一条清楚的界线。通过分别定义这两份重要文件的格式和内容，许多组织就组织的责任问题达成了一致意见，如营销负责制定需求规格说明，而软件工程负责功能设计规格说明（见图 8-1）。

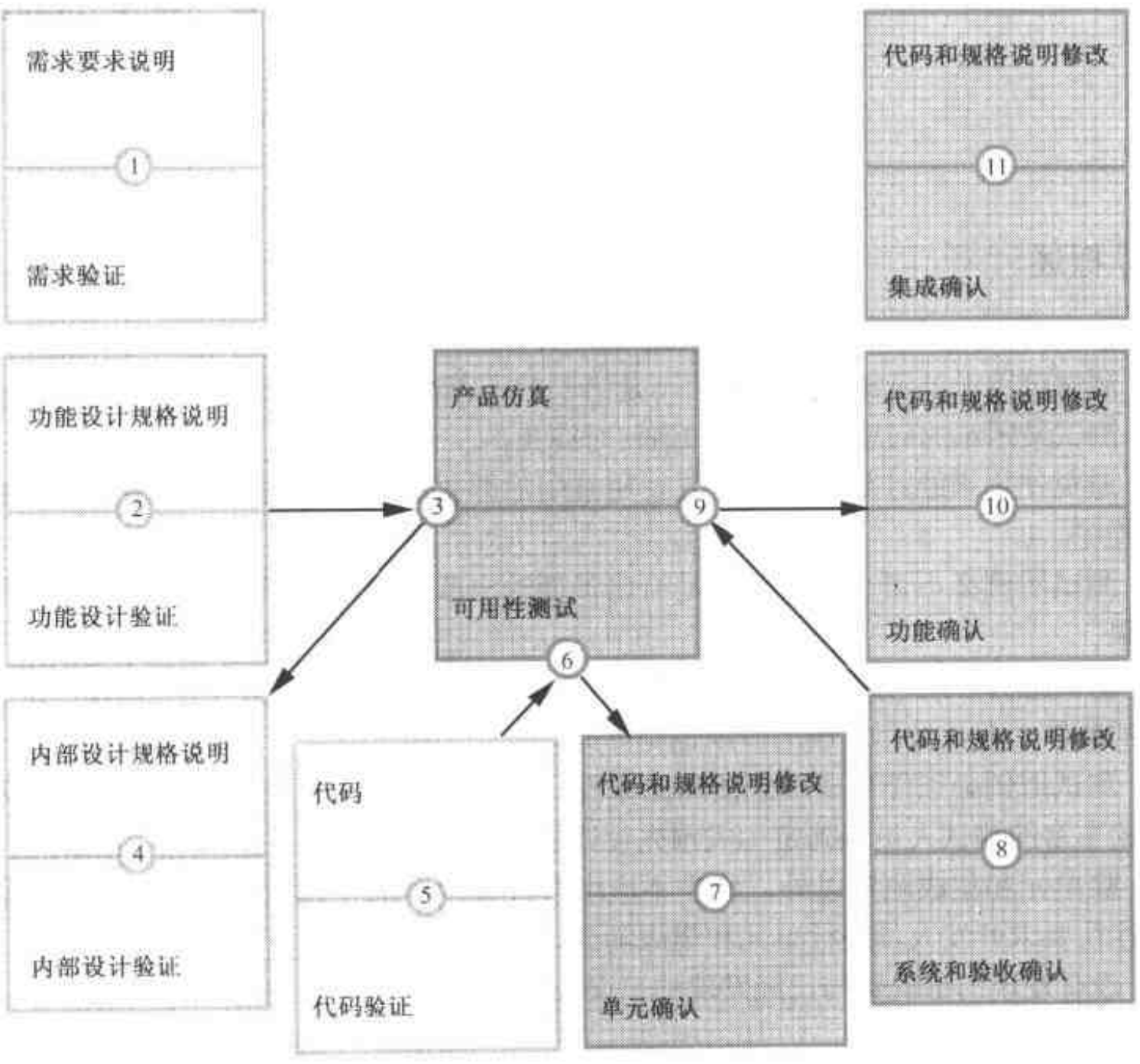


图8-1 SDT（软件开发技术）Dotted-U模型，确认（©1993,1994 Software Development Technologies）

8.1.1 覆盖

我们怎样度量产品的被测程度？“被测试”的量是什么？在多大程度上我们的测试用例覆盖了产品？怎样定量度量我们的测试工作？

某测试用例对程序 P 的测试将：

- 涉及（覆盖）P 的一些需求；
- 利用（覆盖）P 的一些功能；
- 执行（覆盖）P 的一些内部逻辑。

我们必须确保在每一层次上都有足够的测试。

对 P 的被测试性进行的测量是 P 的测试用例的集合提高需求覆盖性、功能覆盖性和逻辑覆盖性的程度。

8.1.2 基本测试策略

黑盒测试和白盒测试是两种基本的测试策略。它们是策略而不是技术或分析方法。

黑盒测试基于功能设计规格说明，不考虑内部程序结构。黑盒测试是针对最终用户、外部规格说明而进行的测试。进行黑盒测试时对产品的内部情况一无所知。在实践中，在不太了解代码的情况下，基于需求和功能规格说明，测试或至少制定详细的测试计划是非常重要的。了解代码会改变对需求的看法，而测试设计不应该过早地被这方面的知识所“污染”。

黑盒测试不能测试隐藏的功能（即已经实施但功能设计规格说明中没有描述的功能），与之相关的错误在黑盒测试中也发现不了。

白盒测试必须知道内部程序结构，它基于内部设计规格说明或代码。它们无法检测不存在的功能（即那些在功能设计规格说明中做了描述，但内部规格说明或代码没有支持的功能）。

8.1.3 确认任务与测试覆盖

穷举测试是不可能的，任何程序的测试都应该是不完整的，通过从所有可能的测试用例中确定最有可能检测出最多错误的子集，将这种不完整性的负面影响降到最低水平，这样，我们可以通过有限的测试，发现尽可能多的错误。

测试覆盖由三部分组成：需求覆盖、功能覆盖和逻辑覆盖。IEEE/ANSI 清楚地阐述了前两者的重要性，但事实上逻辑覆盖也是很重要的，原因有两条：

（1）可以间接地提高功能覆盖；

（2）对逻辑路径的测试是必要的，而逻辑路径是无法从外部功能看出来的（例如，一个数学函数可以根据输入变元的值采用完全不同的算法）。

一般所提到的覆盖是指语句层次上。最简单的形式是被测程序中语句的百分比：“80%覆盖”意思是程序中 80%的语句被测试。在一个涉及编译程序测试的项目中，作者发现如果覆盖为 75%，我们发现了 x 个错误，然后将覆盖从 75%提高到 85%，会新发现 x 个错误。同样，从 85%提高到 90%，我们又发现了二至三倍的 x 个错误！一般地说，发现新错误的概率与没有覆盖的代码总量成反比。越接近 100%的覆盖率，我们搜寻的就越有可能是以前没有覆盖的地域，也就越有可能从每行代码中发现更多的缺陷。

8.1.4 测试基础

测试的基础是源材料（被测产品），它们是测试的刺激因素。换言之，就是被认为存在错误的域：

- 基于需求的测试是基于需求文件的。
- 基于功能的测试是基于功能设计规格说明的。
- 基于内部的测试是基于内部设计规格说明或代码的。

基于功能和内部的测试对于未满足需求的情况不起作用。基于内部的测试不能检测功能上的错误。

8.1.5 确认策略

如果有黑盒和白盒两种测试，同时考虑到需求、功能和逻辑测试覆盖方面的需要，怎样确定我们的整体策略呢？

- 基于需求的测试必须采用黑盒策略。可以在不知道内部设计规格说明或代码的情况下对用户需求进行测试。测试基于需求文件，但通过功能设计规格说明而制订。
- 基于功能的测试应该采用黑盒策略。采用功能设计规格说明设计基于功能的测试是必要的也是充分的。需求和内部设计规格说明对于基于功能的测试没用。
- 基于内部的测试只能采用白盒策略。可采用功能设计规格说明制订测试。

有两个基本要求，适用于所有的确认测试：

（1）明确结果

（2）可重复性

测试用例的必要部分之一是给出预期的输出或结果。如果没有事先定义预期结果，就很容易将实际结果视为正确。又是 Myers 看到了我们的感觉和认知机制对于测试的重要性："眼睛总是看见它想见的东西。"（Myers，1979：p.12）

测试用例应该是可重复的。换言之，每次在相同的软件/硬件配置上运行时应该能产生一样的结果。如果实际和预期结果不同，开发人员在纠错过程中应能重现这种情况。但可重复性也不是总能实现的，如软件和/或硬件在处理异步过程时就是如此。下面是一些 IEEE/ANSI 的定义：

测试

（ i ）是一种活动，在该活动中一个系统或组成部分在特定条件下被运行，结果被观察或记录，并对该系统或组成部分的某方面进行评估。

（ ii ）是一个或多个测试用例的集合。

IEEE/ANSI 的第二个关于"测试"的定义需要增加项目，以便区分一个测试用例内的低层测试。

测试用例

（ i ）是为特定的目的开发的一组测试输入、执行条件和预期结果。

（ ii ）从头至尾作为一个单元执行的最小实体。

一个测试用例可以执行任何数量各不相同的分测试。

测试步骤

（ i ）是用于特定的测试用例的设置、执行和结果评估的详细指令。

（ ii ）一个测试用例可用于多个测试步骤（见图 8-2）。

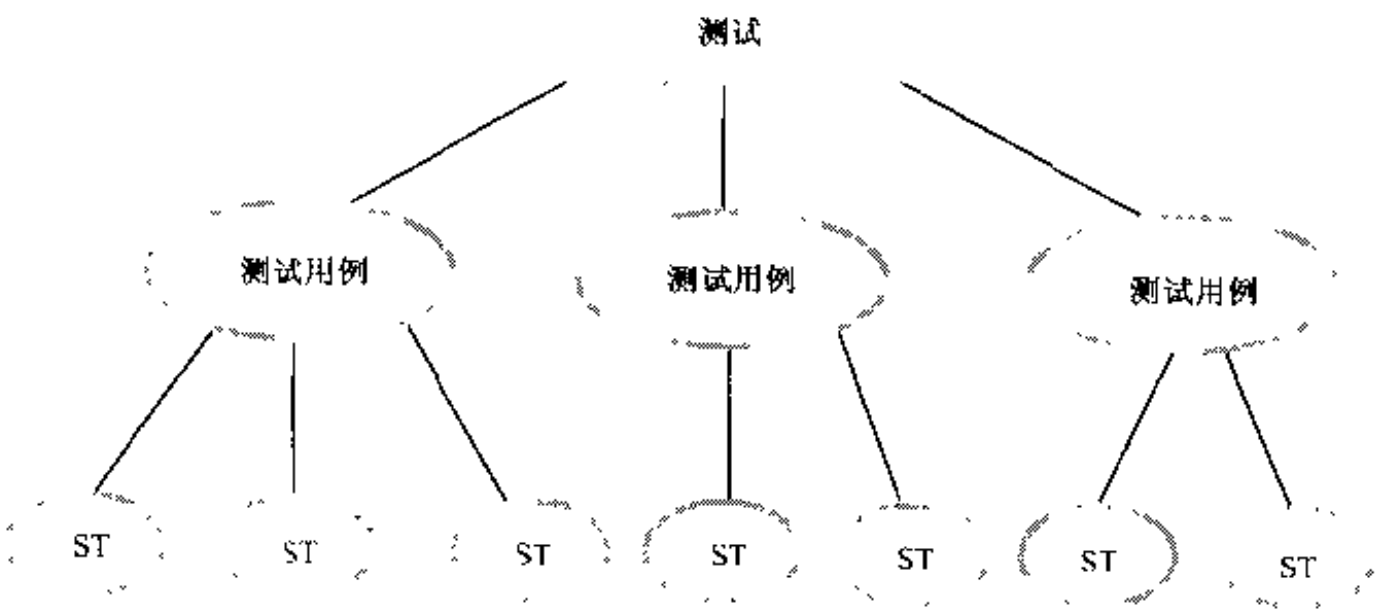


图8-2 测试、测试用例和分测试（©1993,1994 Software Development Technologies）

8.2 确认方法

与纯粹的随机方法相比，经过验证的测试设计方法为确定测试提供了一种更加合理和有效的手段。

本节为设计高效测试介绍了一些详细的技术和分析方法。测试文献中对每种方法都介绍过。下面描述的方法是最常见、最实用且最有用的。每种方法各有所长和缺点（即可能和不能检测出的错误）。

8.2.1 基于功能测试的黑盒方法

常用的方法有：

- 等价类划分
- 边值分析
- 错误猜测

不常用的方法有：

- 因果图
- 句法测试
- 状态转换测试
- 图形矩阵

1. 等价类划分

等价类划分是一种系统过程，以所获得的信息为基础，确定一组要接受测试的输入条件，而每一类输入条件又可能代表（或覆盖）一大组其他的测试。如果对被测产品进行划分，那么产品对于该类的所有成员表现是基本相同的。

目的是减少覆盖这些输入条件的所需测试用例的数量。

有两个不同的步骤。第一步是确定等价类（EC），而第二步是确定测试用例。

（1）确定等价类

对每一外部输入：

（i）如果输入指定了有效值的范围，则定义一个有效 EC（在该范围内）和两个无效 EC（在该范围两端之外各一个）。

例如：如果输入要求 1～12 月中的一个月，则为 1～12 月定义一个有效 EC 和两个无效 EC（月<1 和月>12）。

（ii）如果输入指定有效值的数量（N），则定义一个有效 EC 和两个无效 EC（没有和多于 N）。

例如：如果输入需要至少三本但不超过八本书的题目，那么定义一个有效 EC 和两个无效 EC（<3 和>8 本书）。

（iii）如果输入指定一组有效值，则定义一个有效 EC（在该组有效值内）和一个无效 EC（在该组有效值外）。

例如：如果输入需要 TOM、DICK 或 HARRY 这些名字之一，那么定义一个有效 EC（采用有效名字之一）和一个无效 EC（采用 JOE 这个名字）。

（iv）如果有理由认为程序处理每个有效输入时各不相同，那么每个有效输入定义一个有效 EC。

（v）如果输入指定“必须是”的情况，则定义一个有效 EC 和一个无效 EC。

例如：如果输入的第一个符号必须是数字，则定义一个有效 EC，第一个符号是数字；定义一个无效 EC，第一个符号不是数字。

如果有理由认为程序不是以完全相同的方式处理 EC 中的各个元素，则将 EC 再分为更小的 EC。

（2）确定测试用例

（i）为每一 EC 赋一个惟一的数字。

（ii）编写新的测试用例，尽可能多地覆盖还没有覆盖的 EC，直到测试用例覆盖了所有的有效 EC。

（iii）编写一个测试用例，覆盖一个，且仅覆盖一个还没有覆盖的无效 EC，直到测试用例覆盖了所有的无效 EC。

（iv）如果在同一测试用例中测试了多个无效 EC，那么可能永远不会执行有些测试，因为第一个测试会屏蔽其他的测试或终止测试用例的执行。

提示：任何测试用例的必要部分之一描述预期结果，即使对采用无效输入的测试也是如此。

等价类划分通过识别许多相等的条件极大地降低了要测试的输入条件的数量。它不测试输入条件组合。

2. 等价类划分练习

等价类划分练习看起来容易，但需要练习。作为训练，我们设计了一套练习和答案。阅读下面的程序描写，然后完成随后的工作图表。

GOLFSCORE（高尔夫球记分）功能设计规格说明

GOLFSCORE 是一个程序，计算高尔夫球赛参赛选手的得分，以下列假设和记分规则为基础：

假设：

（1）比赛场地的数量可以在 1～8 之间。

（2）参赛者的数量可以在 2～400 之间。

（3）每位球员在每个场地上比赛一次。

（4）球员的得分是每一球场上的得分之和。

（5）每一高尔夫球场有 18 个球洞，每一球洞的规定击球次数是 3、4 或 5。

每一球洞的记分规则为：

击球次数	得分
超过规定击球次数	0
达到规定击球次数	1
比规定击球次数少 1 次	2
比规定击球次数少 2 次	4
比规定击球次数少 2 次以上	6

输入

GOLFSCORE 的输入是一个格式化的文本文件，依次包含如下记录：

（1）球场记录。每一高尔夫球场一份记录。每份记录包含球场的名称以及 18 球洞中各个球洞的规定击球次数。

第 1 列：　　空白

第 2～19 列：　　球场名称

第 21～38 列：　　球洞 1～18 的规定击球次数（规定击球次数是整数 3、4 或 5）

（2）分界符记录。表示球场记录的结束。

第 1 列：　　非空白

第 2～60 列：　　空白

（3）球员记录。每球场每球员一份记录（可以是任何顺序）。每份记录包含球员的姓名、球场的名称以及 18 个球洞中各球洞的实际击球次数。

第 1 列：　　空白

第 2～19 列：　　球场名称

第 22～39 列：　　球员姓名

第 41～58 列：　　18 个球洞的击球次数

（按每球洞计，击球次数是非零的数字）

（4）分界符记录。表示球员记录的结束。

第 1 列：　　非空白

第 2～60 列：　　空白

输出

GOLFSCORE 生成下列输出报告，显示：

（1）球员的姓名、他们在每一球场的得分、他们的总分以及他们在比赛中的最后排名，按总分从高到低排列。

（2）与报告（1）相同，但按球员姓名的字母顺序排列。

（3）与报告（1）相同，以球场为单位，按该球场的记分从高到低排列。

每份报告为每位球员包含一份输出记录。

工 作 图 表

外部输入条件	有效等价类	无效等价类

该练习有许多种答案。附录 D 提供了一种答案。

边值分析

边值分析是等价类划分的一种变体和改良，主要有两方面的区别：

第一，不是在等价类中选择一个元素作为代表，而是在挑选元素时应使得 EC 的边界受到测试。边界总是最容易发现错误的地方。

第二，不只关注输入条件，也注意输出条件。可以输出什么？输出的类是什么？我们应该提供什么样的输入才能确保获得一个有用的类组能代表符合要求的输出？

边值分析的原则是：

- 如果输入指定一组有效值，则为该组的两端编写测试用例，并为紧靠两端之外的条件编写无效输入测试用例。

例如：如果输入要求一组 0.0 至 90.0 度之间的一个实数，则为 0.0、90.0、-0.001 和 90.001 编写测试用例。

- 如果输入指定一系列的有效值，则为这些值的最小数和最大数以及这些值以外上下各一个数编写测试用例。

例如：如果输入要求至少 3 本但不能超过 8 本书的题目，则为 2、3、8 和 9 本书编写测试用例。

- 对各输出条件采用上述原则。

边值分析不象看起来那么简单，因为边界条件有时很微妙，很难识别。该方法不测试输入条件组合。

边值分析练习

采用前面为 GOLFSCORE 设计的同一功能设计规格说明，列出要测试的边界值。记住输入和输出条件都要考虑。附录 D 提供了一种答案。

错误猜测

错误猜测是一种特别的办法，以直觉和经验为根据确定可能发现错误的测试。基本思路是列出可能出现的错误或可能出错的情况，然后根据列表开发测试。我们以前遇到过的最容易出错的情况是什么？错误的历史能提供答案。过去出错的地方很有可能以后也出错。

可以一试的项目包括：

- 空或零表/串
- 零实例/事件
- 串中的空白或空字符
- 负数

Myers（1979）从事的一项研究表明程序中剩余错误的概率与已经发现的错误是成比例的。仅仅这一条，就为高效率的错误猜测提供了丰富的空间。

因果图

因果图是一种挑选高效测试用例以检查组合输入条件的系统方法，它是将自然语言规格说明转化成形式语言规格说明的一种严格的方法，它揭露规格说明中的不完整性和二义性。

因果测试的导出

因果测试可以从以下步骤导出：

- 将规格说明分解成可以操作的块。
- 鉴别因和果。
- 制作（Boolean）因果图。
- 注明限制，说明不会出现的因和/或果的组合。
- 按顺序跟踪图中的状态条件，将图转换成有限项判定表。表中的每一列代表一个测试

用例。

- 判定表中的各列被转化成测试用例。

因果图检查输入条件组合，可以提供非冗余、高效的测试。它在功能（外部）规格说明验证方面非常有用，因为它能显示规格说明中的错误。但实施起来很困难并且费时。如果有工具将图转换成判定表，则该方法就实用得多。

句法测试

句法测试是一种为程序生成有效和无效输入数据的系统方法。它对使用隐藏语言定义数据的程序实用（操作系统和子系统的互动命令）。如果是形式化开发的显式语言，如编译程序语言，这种测试一般没有效果。

句法测试总的来讲是一种机关枪办法，依靠许多的测试用例，对语义测试（用功能测试）无效。句法测试的关键是学会怎样识别隐藏语言。

句法测试的步骤是：

- 鉴定目标语言（显式还是隐藏）。
- 形式化定义语言句法。
- 通过覆盖定义图先测试有效事例。
- 一层一层、从上到下，设计测试，一次只一个错，一次一层。
- 测试无效事例。
- 使测试的生成和执行自动化。

状态转换测试

状态转换测试是一种分析方法，采用有限状态机，为有许多相似但稍有不同的控制功能的程序设计测试。它主要是一种功能测试工具，对功能设计验证也很管用。

图形矩阵

图形矩阵是一个简单的用图形编排数据的方法。图形表现为一个正方矩阵，每行代表图形的一个节点（节点=1，…，n）。每列代表图形中一个节点（节点=1，…，n）且 M（I，J）定义节点 I 和节点 J 之间的关系（如果有的话）。（矩阵一般很稀疏，因为在某些节点之间往往没有联系。）

图形矩阵用于证明一些关于图形方面的事情，还可用于开发算法。

8.2.2 基于内部测试的白盒方法

一旦启用白盒测试，便可通过一系列的技术确保系统的内部各部分获得充分的测试并且有足够的逻辑覆盖。

逻辑覆盖有四种基本形式：

（1）语句覆盖

（2）判定（分支）覆盖

（3）条件覆盖

（4）路径覆盖

白盒方法的定义和比较

白盒方法如图 8-3 所示。如要实现条件覆盖，则要求测试覆盖特征 1 和 3，而不要求覆盖特征 2 和 4。如要实现多重条件覆盖，则要求测试覆盖特征 1 和 4，这类测试会自动覆盖特征 1 和 2。

	语句覆盖	判定覆盖	条件覆盖	判定/条件覆盖	多重条件覆盖
1 每个语句至少执行一次	Y	Y	Y	Y	Y
2 每个判定中的所有可能结果至少出现一次	N	Y	N	Y	implicit
3 一个判定中每个条件的所有可能结果至少出现一次	N	N	Y	Y	implicit
4 每一判定中所有可能的条件输出组合至少出现一次	N	N	N	N	Y

图 8-3 白盒方法的定义和比较。图中的每一列代表白盒测试的一种方法，每一行(1-4)定义一种测试特征。对于特定的方法（列），行中的“Y”意思是该方法要求该测试特征“N”表示没有要求。“Implicit”意思是该测试特征暗中通过该方法的其他要求获得（©1993，1994 Software Development Technologies）

穷举路径覆盖往往是不现实的。但有关于其他三种基本形式的实用方法，可以提高逻辑覆盖的程度.

白盒覆盖举例

为澄清这些覆盖方法之间的不同，考虑下面的 Pascal 过程。这个例子的目的是列出一组测试（多组输入数据）满足各白盒覆盖方法的标准。

```
The liability procedure
Procedure liability (age, sex, married, premium);
begin
   premium:=500;
   if((age<25)and(sex = male)and(not married))then premium:=premium+1500;
else(if(married or(sex = female))then
         premium:=premium-200;
    if((age>45)and(age<65))then
           premium:=premium-100;)
end;
```

三个输入参数是年龄（整数）、性别（男或女）和婚否（真或假）。记住下面各条：

- 语句覆盖：每个语句至少执行一次。
- 判定覆盖：每个语句至少执行一次；每个判定的所有可能结果至少出现一次。
- 条件覆盖：每个语句至少执行一次；一个判定中每个条件的所有可能结果至少出现一次。
- 判定/条件覆盖：每个语句至少执行一次；每个判定的所有可能的结果至少出现一次；一个判定中每个条件的所有可能的结果至少出现一次。
- 多重/条件覆盖：每个语句至少执行一次；每一判定中所有可能的条件输出组合至少出现一次。

下面提供了责任（保险）过程的逻辑覆盖方法。在后续表格中使用下面的符号。

每行中的第一列表示练习程序中特定的“IF”语句。例如，“IF-2”意思是样本程序中的第二个 IF 语句。

最后一列在括号中显示测试用例的编号。例如“(3)”表示第 3 个测试用例。测试用例编号后面的信息是测试数据本身的缩略形式。例如“23 F T”的意思是年龄=23，性别=女，婚否=是。

表格中的星号（*）意思是“通配符”或“任何有效输入”。

语句覆盖	年　龄	性　别	婚　否	测试用例

该程序中只有两个语句，任何输入组合都会覆盖两个语句。

判定覆盖	年　龄	性　别	婚　否	测试用例
IF-1	<25	Male	False	(1)23M F
IF-1	<25	Female	False	(2) 23 F F
IF-2	*	Female	*	(2)
IF-2	>=25	Male	False	(3) 50 MF
IF-3	>=45	Female	*	(2)
IF-3	>45，<65	*	*	(3)

① 该输入对于 IF-3 是不必要的，但它有必要确保 IF-1 是假[if age<25 and married is false]以使 IF-1 的 else 分支（因此 IF-3）将被执行。

条件覆盖	年　龄	性　别	婚　否	测试用例
IF-1	<25	Female	False	(1) 23 F F
IF-1	>=25	Male	True	(2) 30 M T
IF-2	*	Male	True	(2)
IF-2	*	Female	False	(1)
IF-3	<=45	*	*	(1)
IF-3	>45	*	*	(3) 70 F F
IF-3	<65	*	*	(2)
IF-3	>=65	*	*	(3)

注：这些测试用例没有执行 IF-1 和 IF-3 的 then 分支和 IF-2 的（空）else 分支。

判定/条件覆盖	年　龄	性　别	婚　否	测 试 用 例
IF-1（判定）	< 25	Male	False	(1) 23 M F
IF-1（判定）	< 25	Female	False	(2) 23 F F
IF-1（条件）	<=25	Female	False	(2)
IF-1（条件）	>=25	Male	True	(3) 70 M T
IF-2（判定）	*	Female	*	(2)
IF-2（判定）	>=25	Male	False	(4) 50 M F
IF-2（条件）	>	Male	True	(3)
IF-2（条件）	*	Female	False	(2)
IF-3（判定）	<=45	*	*	(2)
IF-3（判定）	>45，<65	*	*	(4)
IF-3（条件）	<=45	*	*	(2)
IF-3（条件）	>45	*	*	(4)
IF-3（条件）	<65	*	*	(4)
IF-3（条件）	>=65	*	*	(2)

注：上表实际上是所有的判定（来自判定覆盖表）与所有的条件（来自条件覆盖表）的合并　降低了测试用例的数量

多重条件覆盖	年　龄	性　别	婚　否	测 试 用 例
IF-1	< 25	Male	True	(1) 23 M T
IF-1	< 25	Male	False	(2) 23 M F
IF-1	< 25	Female	True	(3) 23 F T
IF-1	< 25	Female	False	(4) 23 F F
IF-1	>=25	Male	True	(5) 30 M T
IF-1	>=25	Male	False	(6) 70 M F
IF-1	>=25	Female	True	(7) 50 F T
IF-1	>=25	Female	False	(8) 30 F F
IF-2	*	Male	True	(5)
IF-2	*	Male	False	(6)
IF-2	*	Female	True	(7)
IF-2	*	Female	False	(8)
IF-3	<=45，>=65	*	*	impossible
IF-3	<=45，<65	*	*	(8)
IF-3	>45，>=65	*	*	(6)
IF-3	>45，<65	*	*	(7)

8.3　确认活动

确认活动可以分为：

（1）低层测试

（ i ）单元（模块）测试

（ ii ）集成测试

（2）高层测试

（i）可用性测试
（ii）功能测试
（iii）系统测试
（iv）验收测试

8.3.1 低层测试

低层测试涉及对程序的各个组件的测试，一次一个或组合起来测试。这需要对程序内部结构的详细了解，因此最适合开发人员进行测试。

低层测试的形式包括：

- 单元（模块）测试
- 集成测试

单元（模块）测试

单元或模块测试是测试一个程序的各个组件（子程序或过程）。目的是发现模块接口规格说明与实际行为的差异。

单元测试可以处理测试的组合，简化开发人员对错误的诊断和修改，允许使用并行测试，换言之，就是可以同时测试多个组件。

单独测试一个特定模块（X）需要：

（1）一个驱动程序模块，它将测试用例转换成 X 输入变量的形式并打印或解释 X 的输出结果。

（2）使用零个或者更多的桩（stub）模块，对被 X 调用的模块进行仿真，要求每个桩模块都是在运行层次上直接从属于 X，如果 X 是终端模块（即它不调用其他任何模块），则不需要桩模块。

集成测试

集成测试是将多个部件组合并测试的过程，集成测试的主要目的是发现部件之间接口的错误。

集成测试的执行有几个层次，我们可以集成和测试一个程序的不同模块、一个子系统的不同程序、一个系统的不同子系统、或网络中的不同系统等等。此外，集成测试还有很多其他方案。

在非增量型的“莽撞”（big bang）集成中，将所有的部件组合形成程序，然后对这个集成的结果进行测试。实践证明，这是一个效率很低的方法，由于错误可能牵扯到任何部件，使得调试非常困难。

增量型集成是将要测试的部件与已经测试过的一组部件结合，有两种增量型集成：自底向上和自顶往下。增量型集成有许多长处，由于驱动程序或桩模块少，工作量也就少一些。那些涉及部件之间接口不匹配的错误较早就能被发现。纠正错误也更快，因为发现的错误一般与最新添加的组件有关。测试也更详尽，测试新组件时会为以前集成的组件提供进一步的功能和逻辑覆盖。

自底向上集成的步骤包括：

- 从层次结构中的终端模块（不调用其他模块的模块）开始。
- 为每一模块生成驱动程序模块。
- 下一个要测试的模块是其从属模块（它调用的模块）都已被测试的模块。
- 一个模块被测试后，其驱动程序被一个真正的模块（下一个要测的模块）和其驱动程序所取代。

自顶往下集成的步骤包括：

- 从执行层次中顶上的模块开始。
- 生成桩模块，有些可能需要多个版本。
- 桩模块与他们首次出现相比往往更复杂。
- 下一个要测试的模块是至少有一个从属（调用）模块已被测试的模块。
- 一个模块被测试后，其桩之一被一个真正的模块（下一个要测试的模块）和其要求的桩所代替。

自底向上集成的不足是在加上最后一块模块之前程序是不完整的。自顶往下集成优点是早期就有程序的轮廓版本，允许演示。但自顶往下集成的桩很昂贵。因此，它们没有明显的优劣之分：一种方案的长处是另一种方案的短处，而对测试的选择往往基于对开发的选择。

我们可以借此机会考虑一下风险管理问题。有效的办法之一是混合采用自底向上和自顶往下，基于风险优选模块集成；例如，与高风险功能有关的模块可以在该过程中早点进行集成测试，而与低风险功能有关的模块可以晚点测试。尽管人们倾向于先做容易的事情，但该混合方案提出的建议却正相反。

还有一些其他的增量集成策略，一般是自底向上和自顶往下集成的变体（详细情况可参考本章后面的参考文献）。

8.3.2 高层测试

高层测试涉及对整个产品的测试。为保证测试的客观性，测试最好在开发组织之外由独立的测试小组进行。高层测试的形式包括：

- 可用性测试
- 功能测试
- 系统测试
- 验收测试

可用性测试

随着计算机的普及，软件产品的用户群也越来越大。用户的要求越来越高，产品的购买也越来越基于可用性。目的是让软件适合于用户的实际工作风格而不是强迫用户的工作风格适应于软件。可用性软件测试这一思想以前就有，现在是实施的时候了。

可用性测试涉及用户对产品的使用并观察他们的反应。与 Beta 测试不同，尽管也涉及到用户，可用性测试应该尽早在开发周期中实施。应尽早让真正的客户参与进来，甚至屏幕还只

是画在纸上时就参加。功能设计规格说明是开始的前提。

和所有测试一样，对目标进行定义是非常重要的。用户找到他们想要的东西有多容易？浏览菜单有多容易？在生存周期内应设法进行两到三次可用性测试（见图 8-4）。

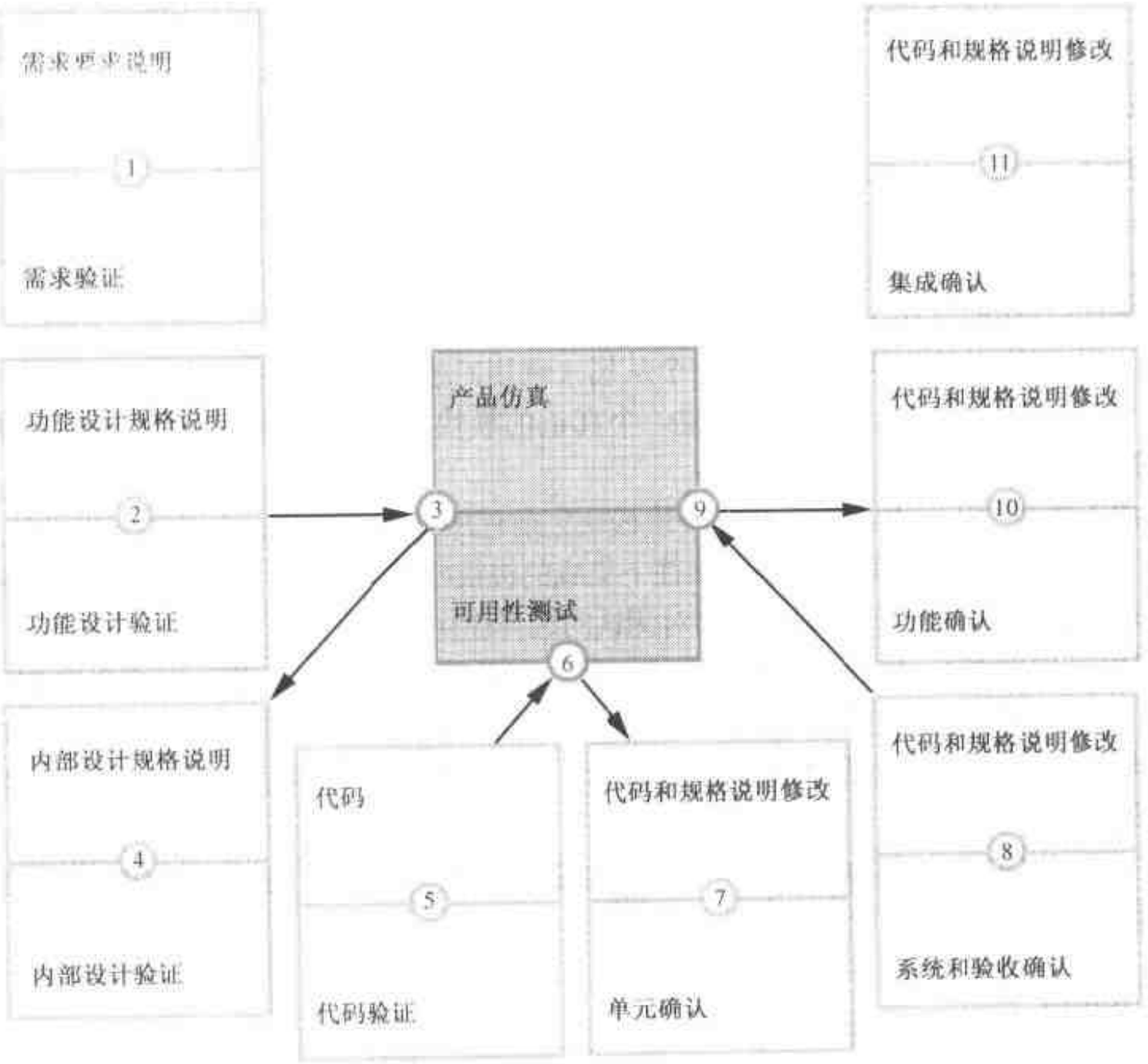


图8-4 SDT（软件开发技术）Dotted-U模型，可用性测试

（©1993,1994 Software Development Technologies）

可用性测试是确定产品用户界面与用户的人类工程需求之间的差异。用户文件测试是基本的组成部分之一。可用性测试从预期的用户那里收集具体问题的有关情况。它往往涉及对产品的表现而不是对功能的评估。

以前，可用性测试是系统测试的组成部分之一，因为它是基于需求的。现在其重要性及其在开发周期中的渗透已经将它提升到了更加显著的位置。

可用性测试是一种确认活动而不是验证活动，因为它需要一个真正的用户通过仿真或实际形式执行最终产品而与其互动。可以测试的可用性特征包括：

- 可访问性：用户可以较为容易地进入、浏览并退出吗？
- 反应性：用户可以在需要的时候，以一种明确的方式做他们想做的事情吗？
- 有效性：用户可以最少的步骤和时间做他们想做的事情吗？

- 可理解性：用户理解产品结构、求助系统和文档吗？

可用性测试的过程如下：

- 定义测试的目标。
- 准确定义并聘请被测对象。
- 对测试进行计划并准备好各种所需的材料。
- 让被测对象进入放有工作站的工作室或实验室，还可以架设摄像机并设置通往观察室的单向窗户。
- 进行测试。
- 使用摄像机和/或观察人员，记录下被测对象的每一个词和动作。
- 专家和开发人员对结果进行分析并提出修改意见。

可用性测试的类型有：

- 不受形式限制的任务：用户未计划的任务。
- 结构化过程脚本：事先定义、撰写好的脚本，内含用户可遵照执行的一步一步的指令。
- 纸屏幕：原型或实体模型还没有做好之前，研究人员为用户充当计算机的角色。
- 原型实体模型：采用前期原型而不是最终产品。
- 在用户办公室进行现场试运行：在用户办公室采用原型或最终产品。

如果研究人员或产品本身的自动模块按脚本引导被测对象并提问的话，则可用性测试方法可能产生强加于人的效果。如果用户单独工作，或者可用性专业人员呆在单向窗户后面不出声的话，就不会有这种强加于人的效果。

功能测试

功能测试是发现程序的功能规格说明与其实际行为之间的差异的过程。如果检测出了差异，有问题的可能是程序也可能是规格说明。所有基于功能测试的黑盒方法都适用（见图 8-5）。

功能测试在产品提供给客户之前由测试小组进行。只要产品具备足够的功能可以执行某些测试，或在完成单元或集成测试之后，就可以开始功能测试。

功能覆盖度量按以下方式进行：

对程序 P 的特定测试用例的执行将检测（覆盖）P 的一些外部功能。对 P 的可测性的度量是 P 的测试用例组合形成的功能覆盖度。功能覆盖可以用功能覆盖矩阵度量。黑盒测试方法可以提高功能覆盖。

功能测试的步骤如下：

- 分解并分析功能设计规格说明。
- 将功能划分成逻辑组件，对每一组件列出详细的功能。
- 对每一功能，采用解析的黑盒方法确定输入和输出。
- 开发功能测试用例。
- 开发功能覆盖矩阵。
- 执行测试用例并度量逻辑覆盖。
- 根据功能和系统测试的组合逻辑覆盖的提示，开发补充功能测试。

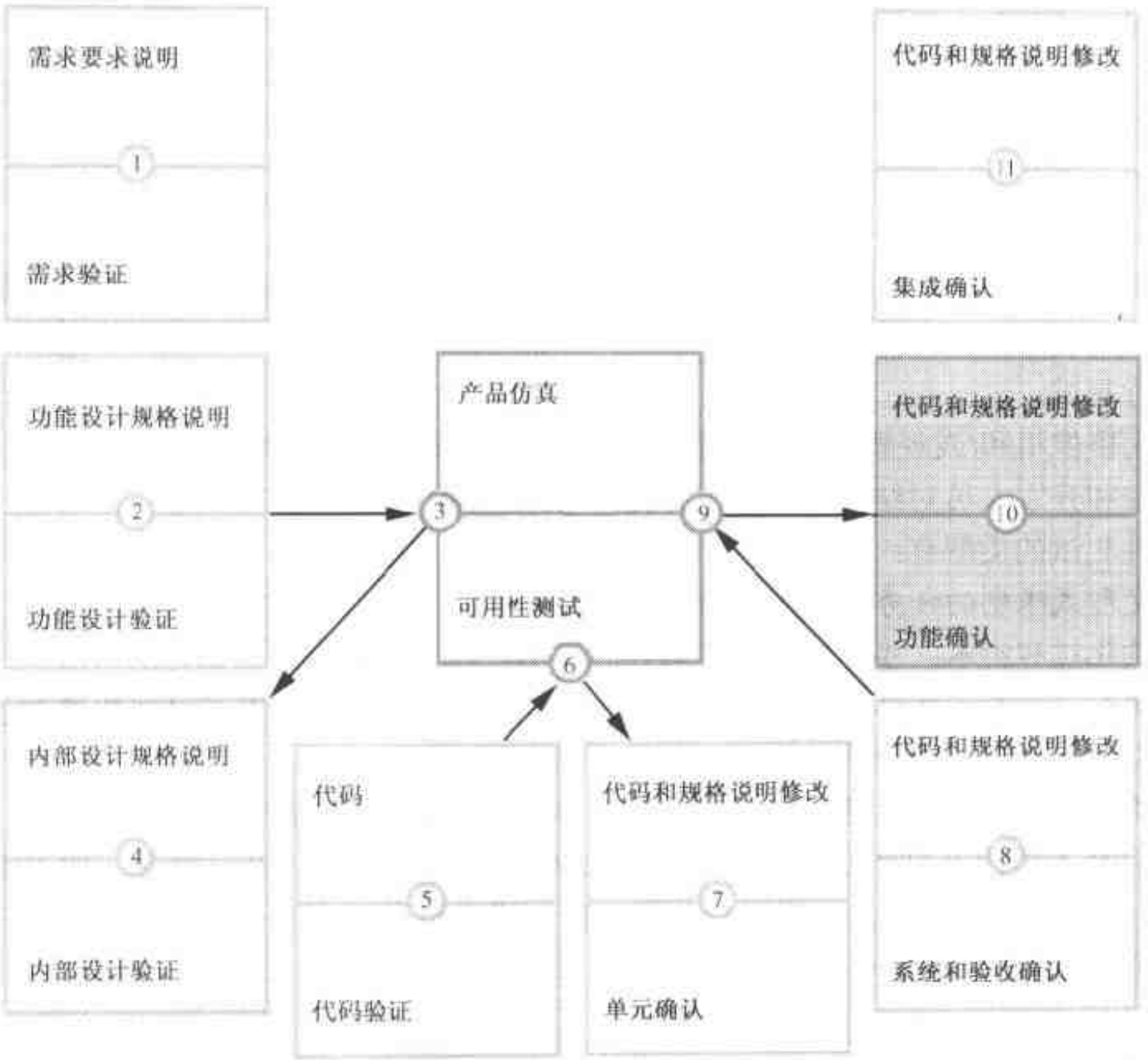


图8-5 SDT（软件开发技术）Dotted-U模型，功能测试

（©1993,1994 Software Development Technologies）

功能覆盖矩阵（见图 8-6）很简单，是一个矩阵或表格，列出要测试的具体功能、测试每一功能的重点和包含每一功能测试的测试用例。

系统测试

系统测试是最容易被人误解并且是最难的测试活动。尽管名叫系统测试，但它却不是对于被完整地集成的系统或程序进行功能测试的过程。如果采用功能测试，则系统测试就是冗余的。系统测试是表明程序或系统没有满足需求规格说明所确定的原有需求和目标的过程。

系统测试是很难的。不可能有测试用例的设计方法，因为需求和目标没有，也不应该，用精确的术语描述程序的功能。需求必须具体，以便可以测试，但也必须一般，以便给功能设计以自由。严格、通用的技术方法所必需的具体性，对于需求规格说明来说是不存在的（见图 8-7）。

因为没有方法，系统测试需要很多的创造。我们应该从用户的角度进行思考，尽量考虑他们要解决的问题。设计系统测试时要分析需求规格说明，而将其形式化时则要分析功能设计规格说明或用户文件。这是测试用户文件的一种理想办法，但往往很不实用，因为必须形成系统测试用例时手册一般还没有准备好。

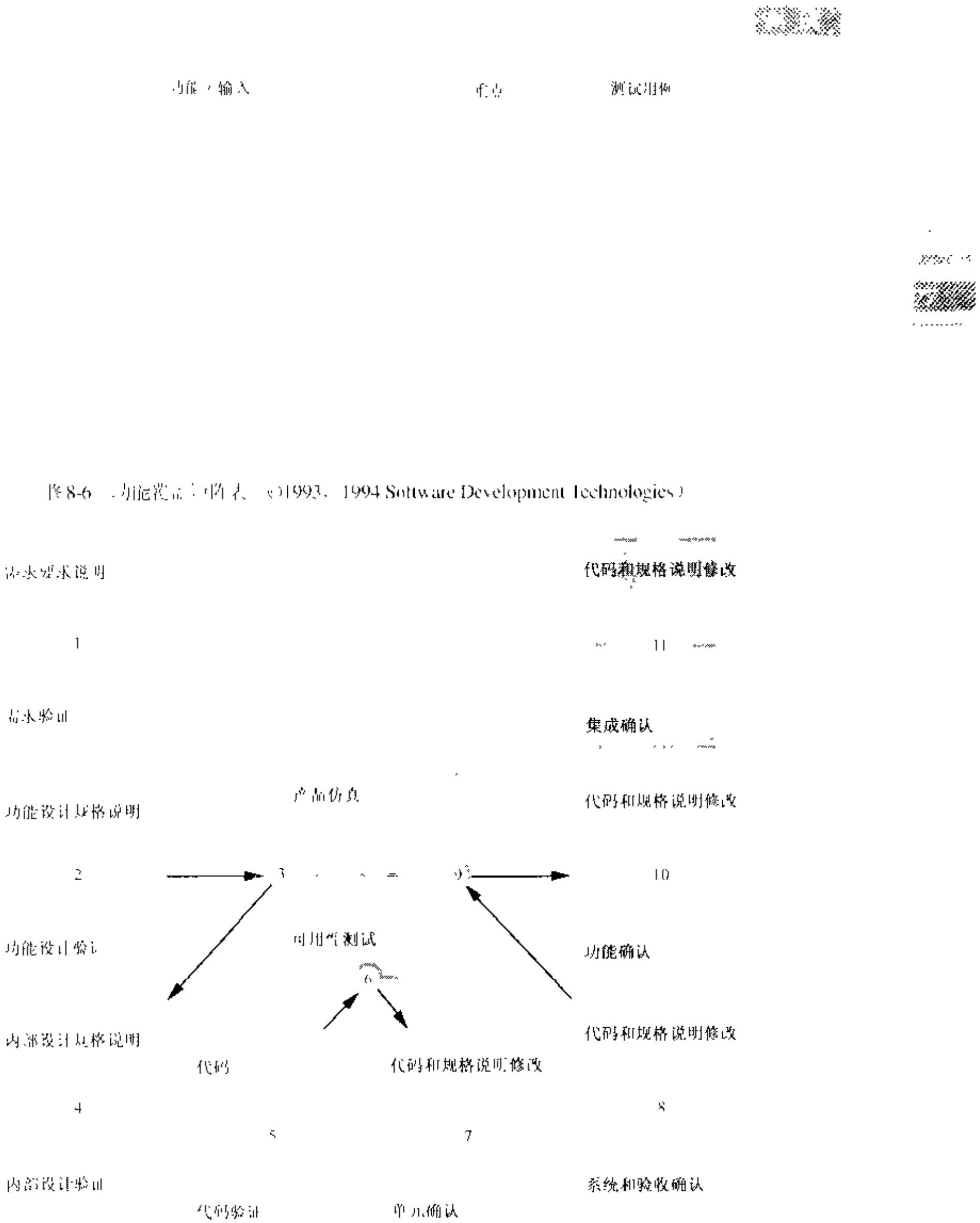


图8-6　功能设计中的术语（©1993、1994 Software Development Technologies）

图8-7　SDT（软件开发技术）Dotted-U模型：系统测试

（©1993、1994 Software Development Technologies）

既然没有一种专门的方法可供使用，我们可以针对各种不同类型的系统测试构造用例。需求覆盖可以表述如下：

程序 P 的某个测试用例的执行将检查（覆盖）P 的某些需求。P 的可测性可用 P 的测试用例集合所形成的需求覆盖来度量。

黑盒测试方法用于提高需求覆盖。需求覆盖可以用需求覆盖矩阵或需求跟踪矩阵进行度量。

系统测试的类型/目标如下：

- 容量测试：判断程序是否能处理所要求的数据、请求等容量。
- 负载/强度测试：确定负载峰值条件，此条件下程序不能在要求的时间间隔内处理要求的负载。
- 安全测试：显示程序的安全性要求是否被破坏。
- 可用性（人类因素）测试：确定对于用户来说困难或不方便的操作。要测试文件、设备和手动过程。
- 性能测试：确定程序是否满足性能需求。
- 资源应用测试：确定程序使用资源（存储器、磁盘空间等）的水平是否会超过需求。
- 配置测试：确定按需求配置软件或硬件时程序是否运行正常。
- 兼容性/转换测试：确定程序的兼容性目标是否符合，转换过程是否可以工作。
- 可安装性测试：确定安装过程中会导致不正确结果的方式。
- 恢复性测试：确定系统或程序出现故障之后是否满足恢复性需求。
- 服务能力测试：确定其服务能力是否满足需求。
- 可靠性/可获得性测试：确定系统是否满足可靠性和可获得性要求。

系统测试由测试小组在客户可获得产品之前进行。只要产品具备一定的功能可以执行一些测试或者在单元和集成测试完成之后就可以开始系统测试。可以与功能测试并列进行。由于测试一般要依赖功能接口，最好将系统测试推迟，直到功能测试表明预定的可靠性水平，如 40% 的功能测试完成之后再进行。

系统测试的步骤如下：

- 分解并分析需求规格说明。
- 将需求划分成逻辑类，对每一组件列出详细的需求。
- 对于每一类系统测试：
  - —— 对每一相关需求，确定输入和输出。
  - —— 开发需求测试用例。
- 开发需求覆盖矩阵，该矩阵为一个表，表中的元素描述一个特定的为需求覆盖增值的分测试、该分测试的优先权以及该分测试中出现的具体测试用例。
- 执行测试用例并度量逻辑覆盖。
- 根据组合覆盖信息的指示，开发补充测试。

验收测试

验收测试是将最终产品与最终用户的当前需求进行比较的过程。一般在测试小组满意地完

成可用性、功能和系统测试之后由客户或最终用户进行一定一般在指定时间内以生产方式运行并操作软件（见图 8-8）。

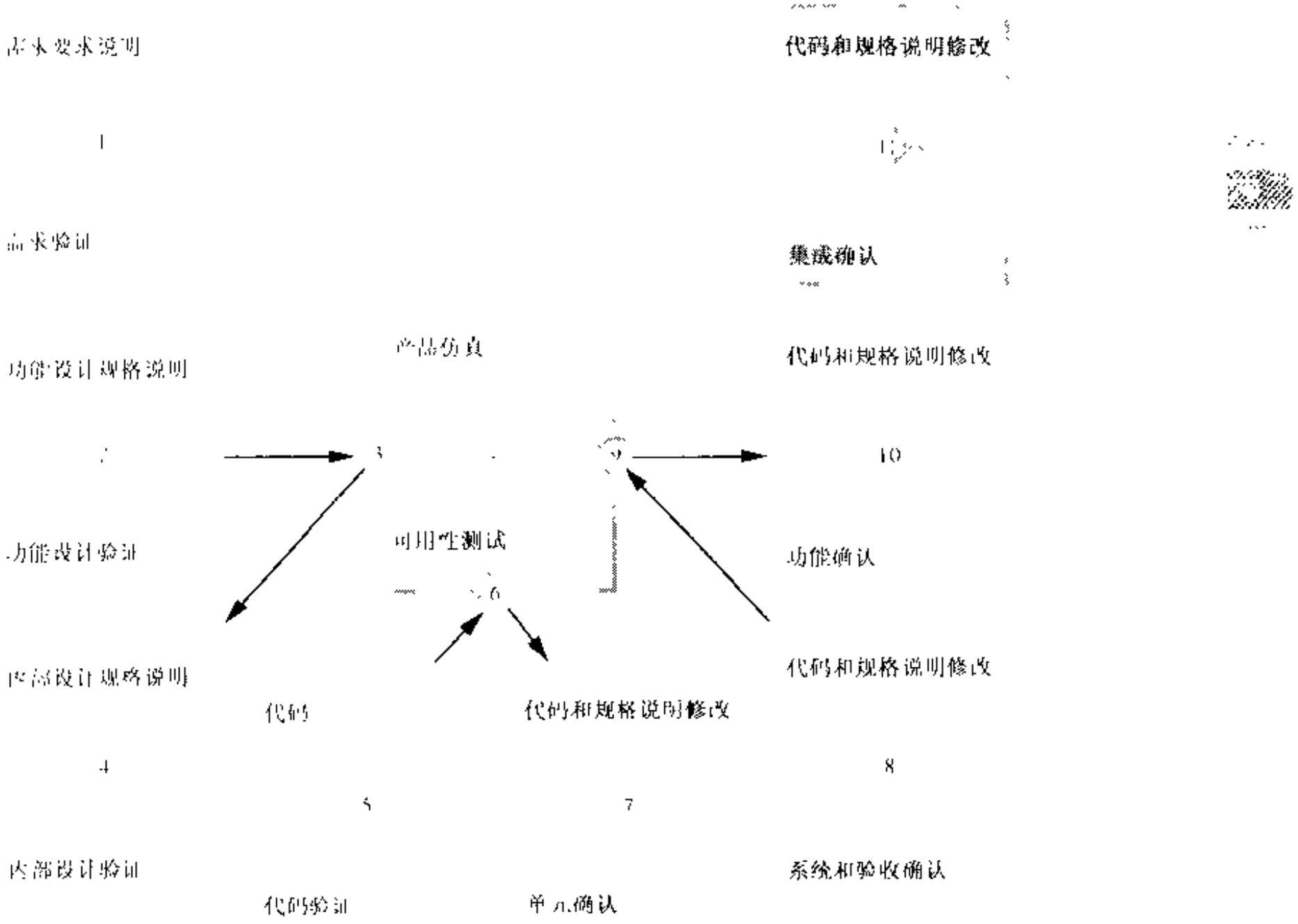


图8-8 SDT（软件开发技术）Dotted-U软件验收测试

（©1993，1994 Software Development Technologies）

如果软件是按合同开发的，则验证测试由签定合同的客户进行一合同规定了验收标准。如果产品不是按合同开发的，开发组织可以安排其他形式的验收测试——ALPHA 和 BETA 测试

ALPHA 和 BETA 测试可分别用作验收测试，不过常常是两者同时都用，BETA 在 ALPHA 后。两者都涉及在指定时间内以生产方式运行并使用软件。ALPHA 测试一般在开发公司内，但在开发组织外由最终用户进行。BETA 测试一般在公司之外，在不是所有客户都能获得软件之前，由经过挑选的真正的用户群进行

实施 ALPHA 和 BETA 测试的第一步是定义测试的基本目标：累进测试和/或回归测试。累进测试是测试新代码以确定是否包含错误的过程。回归测试是对程序进行测试以确定修改是否在未修改的代码中引进了错误（回归）。

ALPHA 和 BETA 一般作为回归而不是累进测试更有效。ALPHA 和 BETA 都要求仔细挑选用户。用户应该有使用产品的积极性。用户能提供良好的硬件和软件配置覆盖。出于支持方面的考虑，测试地点也很重要

必须对测试的时间予以充分的考虑。必须有一定的长度，以便获得所需的反馈。时间长度的确定要根据新软件是否具备相关的学习曲线或是否需要用户进行新的开发。

建议制定双边计划和契约。用户同意测试的时间长度以及在该时间长度内指定的应用量和类型，并且同意及时报告问题、进展和状况。提供商同意提供特殊的支持。如果发现非常严重的缺陷，有些组织提供奖励。

8.3.3 再测试

为什么要进行再测试？因为经常使用的软件产品必须不断更新，而产品的每一新版本都必须再测试。

8.3.4 累进测试和回归测试

除非真正是用后即扔，大多数的测试用例开始都是累进测试用例，而最终成为产品生命中的回归测试用例。回归测试不是一种新的测试活动。它是重新执行某些或所有为具体的测试活动开发的测试。可以为每一活动进行回归测试（如：单元测试、可用性测试、功能测试、系统测试等）。

交付测试的一项产品的新版本是下列情况之一：

（1）新一轮测试的初始输入，最终成为该产品的一项主要的新版本。

（2）该初始输入的后续升级，而不管客户是否可以获得（即符合发布标准）与否。

理论上，后续升级与初始输入所要求的再测试是一样多的，而两者都可能需要认真的回归。将它们区别对待会有风险。通过配置管理小心处理变化并且掌握变化对产品的影响，可以降低这种风险。

两种完全不同的测试政策是：

（1）测试组织本着合作的精神同意测试尚未准备发布的产品的原始、甚至未完成版本。在此，测试过程一般是不正规的。宣布产品准备发布后，一般要求正规的再测试。这种政策有可能成为测试组织滥用的目标，他们将时间都花在这种方式上，而不是建设自动回归测试库。

（2）测试件组织只接受那些准备向客户发布的产品版本进行测试。这样做太严格，但有时是必须的。

在有些组织中，只有在产品成功地通过了预先定义的一组验收测试之后，测试功能才接收产品进行测试。这类验收测试是经过仔细挑选的，以便确保后续测试活动的效果。这意味着产品的基本功能是相当完整的，执行其他测试时一般不会遇到较大的干扰。

尽管对产品开发小组很有帮助，但对未完成产品进行测试往往需要测试开发人员的技术和专业知识。如果测试开发人员将时间都花在测试的执行上，他们就没有时间开发更多的测试了。在接手测试准备发布的产品之前，对方方面面都作出考虑，包括进度和费用，是非常重要的。许多组织对产品交付测试有严格的标准（即测试组织的接收标准）。

8.3.5 测试执行的筹划

要设计能检测出错误的测试用例时，还要考虑一些测试执行方面的问题：

- 是采用许多小测试用例还是采用几个大的测试用例
- 测试用例的相互依赖
- 测试执行的主机环境
- 测试点

一个测试用例包含一个或多个分测试，而每一分测试的目标都是检测错误。一个分测试可能向被测产品传送：

- 只是有效输入
- 只是无效输入
- 有效和无效输入的组合

如果测试用例引起故障，会出现一系列的情况，视测试用例本身、测试环境和被测产品而定。测试用例的执行可能被提前终止，这样一来，后续分测试就不被执行了。这意味着只有在改正错误或修改测试用例后，才能执行后续分测试。另一种可能是测试用例的执行继续进行下去，就象没有出现故障一样。

可以在几个层面上屏蔽测试条件：

（1）测试用例之间：如果一个测试用例（X）依靠另一个测试用例（Y），那么由于 Y 引起了故障而使 Y 不能满足 X 的依靠时，X 的剩余部分就不能被执行。

（2）测试用例以内、分测试之间：在一个测试用例内，如果一个分测试引起了故障，剩余的分测试就不能被执行。

（3）分测试以内、输入之间：在一个分测试内，如果一个输入引起故障，剩余的输入就不能被测试。

屏蔽测试条件举例

- 在测试一个程序接口时，分测试调用产品的过程之一（XYZ）：
  CALL XYZ（无效、有效、无效、有效、无效）
- XYZ 以一定的顺序检查输入参数的有效性，顺序对于测试人员来说一般是未知的。
- 检测到第一个无效参数时，XYZ 将中止或将一个误码返回给调用者，从而提前结束执行，剩下的其他无效参数都未检查到。

注意，上面的例子永久性的屏蔽掉了输入条件——甚至一个有效输入如果在被测产品中引起故障，也可以暂时屏蔽其他的输入条件，直到错误被改正为止。

测试执行指南

- 对于使用有效输入的分测试，采用大测试用例
- 对于使用一个无效输入的的分测试，每个分测试采用一个测试用例
- 一个测试用例可以依赖为一个程序但不应依赖另一个测试用例。

- 开始执行每一个测试用例时，主机系统的状态必须一致。
- 新文件不应累积。
- 工作模式不应发生变化。该指南与每一个测试用例的可重复性相联系。如果测试用例引起故障，要诊断原因是非常困难的。如果基础环境失去控制，故障的起因可能进一步被掩盖起来。该指南对测试用例的分测试层也适用。

8.3.6 测试点

有时要让测试用例为综合测试一项产品生成必需的条件是困难的。为克服这些障碍，测试可以要求开发对产品增加测试点。测试点是产品中的一个永久性的点，它：

- 以手工或程序方式讯问可以由测试人员通过外部手段设置的变量的值。
- 按变量值的提示，进行一项或多项具体行动。
- 如果变量置为缺省值，就什么都不做。

- 基于功能测试的主要黑盒方法
  - 等价类划分
  - 边值分析
  - 错误猜测
- 基于内部测试的白盒方法
  - 语句覆盖
  - 判定覆盖
  - 判定/条件覆盖
  - 多重条件覆盖

图8-9 方法概括（©1993,1994 Software Development Technologies）

测试点用于终止系统，以使测试恢复过程、引入时延、调用测试人员提供的过程以及在一个通道、控制器或装置中生成输入/输出错误条件用于恢复过程的测试。

测试类型	执行人员
● 低层测试	
—— 单元（模块）测试	开发
—— 集成测试	开发
● 高层测试	
—— 可用性测试	独立测试组织
—— 功能测试	独立测试组织
—— 系统测试	独立测试组织
—— 验收测试	客户

图8-10 活动概括（©1993,1994 Software Development Technologies）

8.4 确认测试的推荐策略

- 为功能测试系统地确定、设计并开发基于功能的测试，为可用性和系统测试采用黑盒

方法和基于需求的测试。

- 运行测试，并使用自动工具度量它们的集合内部逻辑覆盖。
- 评审并分析测试结果。
- 如果某一区域显示不成比例的大量错误，则强化该区域的测试。

基于外部的测试应该由不知道被测程序的内部结构的人进行。掌握了内部结构会使测试出现偏向，避免的办法是请别的人设计不同类型的测试，或者让同样的人在基于外部的测试过后设计基于内部的测试。

8.5 参考文献

黑盒测试和白盒测试分析方法方面的文献以及单元测试、集成测试和系统测试方面的文献：

Beizer, B.（1984）.*Software System Testing and Quality Assurance*. Van Nostrand Rheinhold.

Beizer, B.（1990）.*Software Testing Techniques*. Van Nostrand Rheinhold.

Myers, G.J.（1976）. *Software Reliability: Principles and Practices*. John Wiley.

Myers, G.J.（1979）.*The Art of Software Testing*. John Wiley.

可用性方面的文献

Myers,G.J,Brad, A. and Rosson, Mary Beth（1993）."Survey on User Interface Programming," *CHI'92 Conference Proceedings*. ACM Conference on Human Factors in Computing Systems.

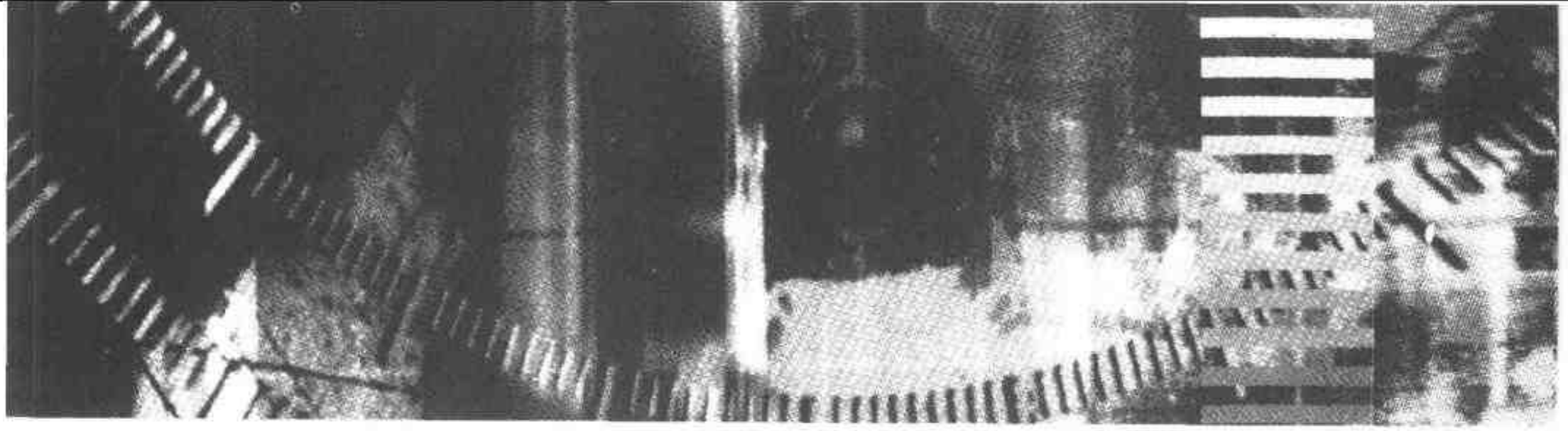
Rosenbaum, Stephanie（1993）."Alternative Methods for Usability Testing," *Software Testing, Analysis, & Review*（*STAR*）*Conference Proceedings*.

还可参考

- SIGCHI (ACM Special Interest Group on Human Factors in Computing Systems)
- Usability Professionals Association

可用性测试方面的专业组织：

- American Institutes for Research, Palo Alto, CA
- IBM User Interface Institute
- Mead Data Central, Inc, Miamisburg, OH
- Multimedia Research, Bellport, NY
- Usability Sciences Corp., Dallas, TX



第9章

控制确认成本

假如我们拥有的资源是无限的，那么我们便可以完成我们想要进行的所有测试。但软件项目的实际情况是，我们几乎总是要么缺时间，要么缺钱，或者两者都缺。这种情况很令人头疼，因为我们既不知道如何系统地降低成本，也不知道测试的实际成本有多高。

只有通过度量（见第 12 章）才能了解到实际成本。但在一般的测试实践中准备建立的控制之下，我们怎样才能获得我们已有的成本——特别是确认成本？本章探讨在整体测试战略中建立起成本控制要考虑的主要问题。

重要的因素之一是测试件。测试件是一系列主要的测试工作产品（可交付的产品）。测试件的主要目标是使测试产量极大化，就是说要使检测出错误的能力最大化，检测的次数最小化。

测试件还有些目标与错误检测无直接的联系，但对于成本控制却是非常重要的。这些目标包括使测试实施成本、测试维护成本和测试开发成本最小化。

9.1 使测试实施成本最小化

作为累进测试的一次性测试实施成本并不重要。在产品的生存周期内为该产品的每一新版本进行重复测试而发生的经常性成本与预期的需求相比才是真正要考虑的问题，通常也是非常重要的问题。

测试实施的成本组成部分包括：

- 预运行启动成本
- 执行成本
- 后运行成本

9.1.1 预运行启动成本

在此，我们关心的是使时间总量，劳动力总量，尤其是使从事各类关键工作所需的熟练劳动力的总量最小化。所谓关键性工作包括硬件配置、软件配置、测试环境的建立（文件恢复和初始化）以及测试的确定。

9.1.2 执行成本

这里，我们的目标是使总的执行时间和所需的专用设备尽可能地减少。执行时间包括值班管理时间（要求用户或操作员进行手工操作的测试执行过程中任一部分，在此过程中有必要尽量减少时间、劳动力和技巧）和非值班管理时间。

如果需要重新运行测试时怎么办？是否每次都得全部重新运行一遍？做出不同的选择就会获得不同的成本控制效果，决策是在风险与成本的对立之间进行。

完全重复测试（将所有测试全部重新运行一遍）将风险降到最低，但提高了测试执行的成本。部分重复测试（有选择性地重新运行部分测试）能减少测试执行成本，但同时加大了风险。

为部分重复测试选择正确的测试，其成本也许同样很高，必须将其与低测试执行成本进行比较，权衡利弊。这要看具体环境而定。如果自动化程度很高，有时重新运行所有的测试，其成本效益也是很不错的。

为部分重复测试选择测试的办法有两种：

（1）确定并挑选一些测试，对由于发生变化而受到影响的每一事物进行重新测试；

（2）确定并挑选一些测试，对与变化有密切和直接联系的事物进行重新测试。

注意，第一种办法的风险要小些。而第二种是一种很主观的办法，并且需要对产品十分了解。一般的建议是，只有在不允许完全重复测试的情况下才采用部分重复测试。

9.1.3 后运行成本

在此，我们的目的是进行测试结果的分析和文档编制、测试环境的拆除以及原有环境的恢复，使所需时间和熟练劳动力的总量减少到最低限度。

9.1.4 降低测试实施成本的建议

重新配置硬件既耗时又费钱。预运行时，可以考虑采用为测试环境永久配置的专用硬件。在允许的范围内，尽可能使软件和测试环境的配置自动化，使测试的确定和选择过程自动化。

测试的实施应尽可能自动化，以减少手工辅助。如果测试过程的某部分需要照管，最好请初级技术人员而不是测试开发人员。另一方面，由于测试开发人员经常面对测试值班管理任务，他们会想办法使其自动化！

涉及后运行时，在允许的范围内，测试结果与预期结果的比较应自动化，以降低分析成本。使测试自动化的方法有两个：

（1）使用测试工具（采用现有工具与制作新工具相比，效益总是要高一些）；

（2）将自动化集成到测试用例本身。

推荐进行自动化的两个候选过程是：

（1）执行过程中需要的手工互动（由用户或系统操作员进行）；

（2）测试结果的检验。

前面的两种办法对于两个过程都适用。

利用工具使测试自动化

可以采用捕获/回放工具使测试执行过程自动化。在后运行阶段，应尽可能地自动化。考虑到此时时间的紧迫，在完成所有测试后花三天时间以手工方式检查所有的测试结果，之后才知道软件是否通过了测试，是毫无意义的。这一阶段如果实现了自动化，还可以消除人为的失误。

为使测试结果的检验自动化，可在执行过程中采用捕获/回放或脚本驱动模拟器并在执行过程后采用比较器。第 11 章进一步介绍了测试工具的情况以及捕获/回放工具的具体信息。

使手工互动自动化

对于任何软件的测试，都应使手工互动自动化。对于具备高度互动用户界面的测试产品，对于多用户软件的同时使用，以及对于具备图形用户界面（GUI）的软件来说，手工互动自动化是必需的。

手动的基本种类包括操作员行为和用户行为。操作员行为使测试不断执行下去，如装磁带、开始或继续测试的执行；用户行为是脚本驱动的互动测试所需要的。

而所有测试最好都使测试结果的检验自动化，因为除可以消除手工劳动以外，还强调了对预期结果进行预定义的必要性。根据测试用例设计的不同，测试结果可以通过两种办法进行检验：

（1）执行过程中——如果实际结果与预期结果不相符，会打印出异常信息。

（2）执行后——在执行过程中将实际测试结果写到文件上，执行后与预期结果进行比较。

9.2 降低测试的维护成本

在软件产品的生存周期内必须始终保持重复测试。如软件组件对于产品本身一样，所有测试件的组成部分都应该置于配置管理系统的控制之下。

维护工作主要包括：

- 对于报告并经确认的每一个问题都增加一项测试；
- 添加累进测试以测试新的变化；
- 定期检查所有的测试用例，以获得效果的连续性。

怎样判断测试效果的连续性呢？

（1）每一测试用例都应是可执行的：即被测试的产品在功能上不应有任何变化，因为这种变化会阻碍按计划执行具体的测试。

（2）基于需求和基于功能的测试应该是合法的：即产品需求或功能性的变化不应该使测试用例产生误导（检测出并不存在的错误）或不起作用（检测不出原先可以检测出的错误）。

（3）每一测试用例应该不断地增加价值：即相对于另一测试用例来说不应该是完全冗余的，但继续使用它往往是因为它的成本效益高，而不是因为它增值多少。

为保护顾客对于产品的投资，许多软件生产厂家承诺在新版本中提供产品的向上兼容功能。测试人员可以利用这一承诺，只要进行那些使用了在功能规格中描述的接口的测试。这适

用于所有测试，甚至包括基于需求的测试和基于内部的测试。这类测试可以为产品的所有新版本充当不需维护且十分有用的重复测试。

出于可用性方面的原因，软件生产厂家往往随着产品新版本的更替而在产品的用户界面中进行一些重大且互不兼容的修改。这类修改往往是对产品的互动界面而不是程序界面做出的。即使将再培训成本（一般很小）考虑在内，这类修改对于用户来说也是有益的。

但这类修改也使测试人员有更多的活要干（当然也提供了职业保障）。必须开发新的测试。必须对旧测试进行检查，必须修改或废弃受到影响的测试。如果对产品界面做出了一对一的实实在在的修改，应有良好的工具或转化程序可以自动将对应的测试做出修改。

白盒测试也有黑暗的一面。有的产品界面不承诺以后的兼容性（陈述或暗示），涉及这类产品界面的测试用例属于高维护测试用例——直接涉及到内部过程，以及/或者检查或设置内部数据项目。必须尽量减少这类测试用例的数量，因为在使用之前必须对他们重新进行检查才能确保他们仍然是有效的。使用这类测试时要十分小心，但需要时还是可以用的。同时，尽量获得对未来维护兼容性的承诺。

测试维护的原则

- 不要为了使测试某程序更容易而修改该程序（如测试挂钩），除非是永久性修改。
- 如果测试用例必须涉及产品的某一内部项目，则应尽可能使该项目从开发起就成为永久性的项目。

9.3 降低确认测试件开发成本

对于确认测试，测试件包括测试用例和测试数据以及支持文件，如测试计划、测试规格说明、测试步骤和测试报告。

一般来说，开发高质量的测试件需要的方法与适用于高质量的软件工程方法是一样的。我们必须计划好每一阶段并了解每一个需求才开始设计。编码开始前必须进行详细的设计，并遵守现有的编码标准。我们必须对测试件进行审查和走查，就像对待其他软件产品一样。测试用户需要用户文件。总之，应把测试件当作重要的商务财产对待，将其置于配置管理系统的控制之下。

获得测试件的基本方法非常简单，包括制作、购买或再利用。只要可能，就应该对测试件进行重用，这就是为什么测试件的维护如此重要的原因。

只要有货，就购买测试件。对基于标准的某些软件，可以十分容易地购买到合适的测试件（如，C、UNIX、COBOL 和 Pascal 的确认测试就是如此）。有时，可以从同行公司购买有用的测试库。只有绝对需要时才制作测试件，因为这通常是最没有成本效益的解决方案。可以在公司内部制作，也可以承包给外部的承包商。开发测试件的另一个需要考虑的重要方面是自动测试执行成本与非自动测试执行成本之间的对立，以及现有测试工具的利用。

有些软件产品或产品组件通过脚本驱动的模拟器很适合于测试。真正的测试和预期的结果以文本文件的形式作为数据记录（脚本）编写并存储。编写一个程序（模拟器），该程序为每一测试读数据，解释并执行测试，然后将测试结果与预期结果进行比较。

以数据而不是以代码编写测试可以降低开发（和维护）成本。测试的数量越多，节省的成本就越大。这对于程序界面的测试十分有用（见图 9-1）。

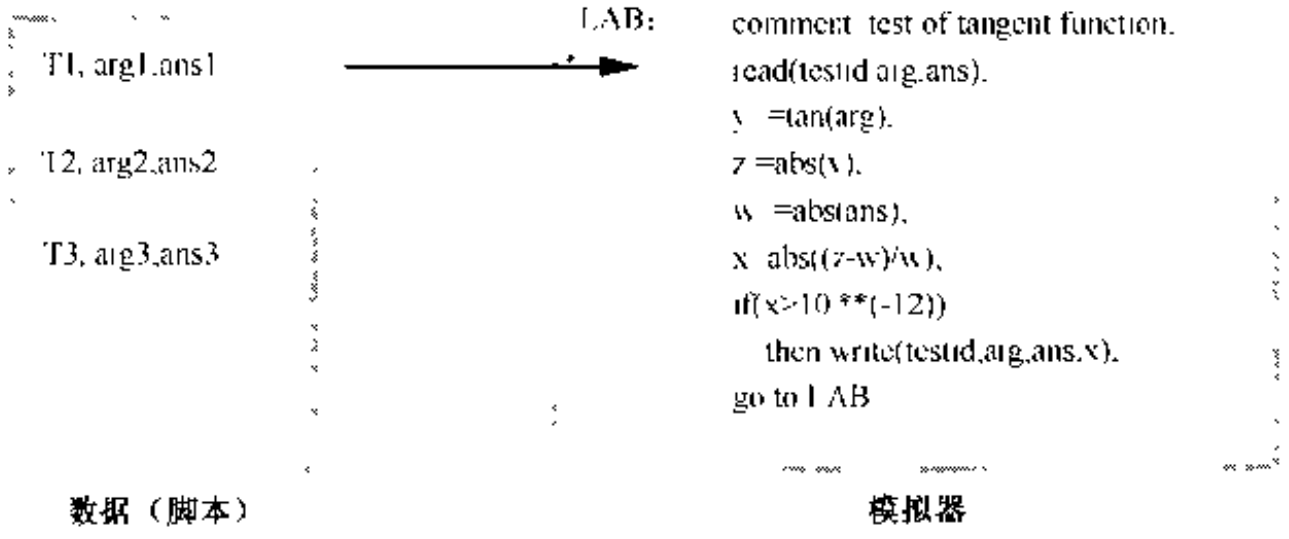


图9-1 脚本驱动的模拟测试举例。实（浮点）数极少完全相等。因此算法中对相等测试进行了描述（本清其十进制数前12位是否相等）（©1993，1994 Software Development Technologies）

9.4 测试件库

由于必须经常存取测试件，应以库的形式将测试用例进行安排、分类，以方便存取。在测试件的生存周期内，这类库在测试执行和维护方面可以节省大量的成本。使用方便的测试件库，不仅方便制作者，别人用起来效果也很好。构建测试件库时，应尽可能使所有测试用例或其子集的挑选和执行都非常简便（见图 9-2）

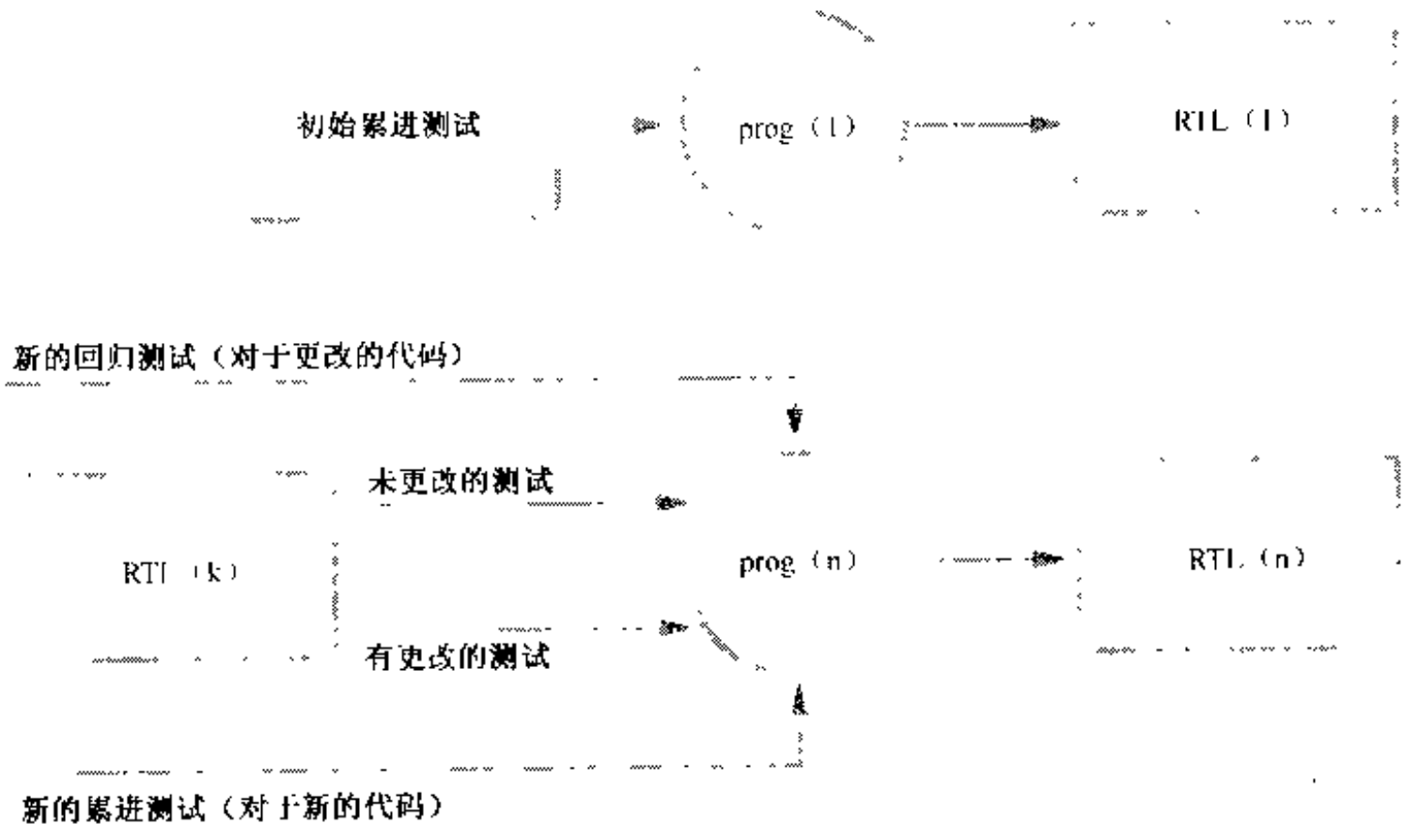


图9-2 回归测试库的演变（©1993，1994 Software Development Technologies）

定义一种结构和命名约定，对于组织的具体测试要求是最为合适的。为方便别人使用测试库，将通常是作为一个单元运行的相关测试用例收集成（已命名）组。一个测试库可以包含许

多组，某个特定的测试用例可以属于一个或多个组。

库的结构和/或分组以及命名约定必须能清楚地确定并区分：

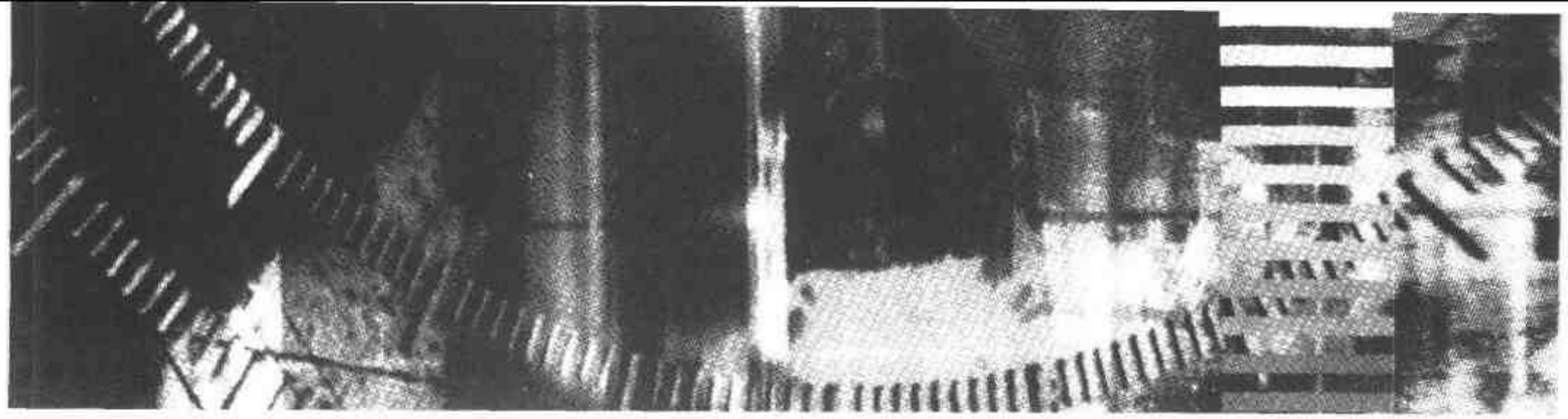
- 需要相同环境的测试用例和分组；
- 使用前必须检查的高维护测试；
- 测试用例或分组的性质（分类、目的、测试基）。

测试库应包含：

- 索引或示意图；
- 库结构以及分组和命名约定描述；
- 对于每一测试分组：
  - —— 组和分组标准描述；
  - —— 每一组内测试用例的名称；
  - ——（各组的）测试步骤文件；
  - —— 各组内执行所有测试用例的命令文件。
- 对于每一测试用例：
  - —— 测试用例和测试步骤文件；
  - —— 所有相关文件（来源、目标、数据等），包括执行测试用例的命令文件和生成全部导出文件的命令文件。

9.5 建议

- 检查一下目前测试费用的支出情况。
- 评估一下为测试环境永久配置的专用硬件的使用情况。
- 将测试结果与预期结果的比较自动化，以降低分析成本。
- 对利用捕获/回放工具使测试执行自动化的情况进行调研。
- 将所有测试件组成部分置于配置管理系统的控制之下。
- 制订计划对所有测试用例定期进行评估，使效果保持连续性。



第10章

测试任务、可交付文件及其在生存周期中的对应阶段

效率高的测试是经过计划的。成功的测试需要有一定的方法，包括条例、结构、分析和度量。没有组织、没有目标、任意随便的测试是注定要失败的。

每一测试活动都有一项或多项输入（开发或交付文件），并由一项或多项任务构成。每项任务提供一项或多项输出（测试可交付文件）。

本章阐述与每一测试活动有关的具体任务和可交付文件，介绍有助于这些任务和可交付文件的各种标准。提供可交付文件的大纲，适用的话，还为文件注明了 IEEE/ANSI 标准。这些任务和可交付文件对应到软件生存周期的具体阶段，显示出对应的时间和位置。

时间和资源总是对实际测试有所限制。在测试的每个层面上（活动、任务、子任务等），为测试工作的各方面确定优先要考虑的问题是非常重要的。确定优先问题的基础是风险。

10.1 主测试计划

测试任务概述如下：

（1）主测试计划制订

（2）验证测试

（i）计划

（ii）执行

（3）确认测试

（i）计划

（ii）测试件开发

（iii）测试执行

（iv）测试件维护

主测试计划风险管理要考虑的问题包括：

- 测试产品的大小和复杂性；
- 产品的关键性，关键软件是那些软件的失效可能影响到安全，或者可能造成巨大的经济或社会损失的软件（IEEE/ANSI，1986[Std 1012-1986]）；
- （SEI）开发过程成熟水平；
- 测试形式（全面、部分、末端、审计级）；
- 人员、经验和组织。

（关于这些问题的背景，参考第 5 章）

通常采用三级优先方案（LO、MED、HI）就足够了。开展工作时，HI 级项目先进行；然后是 MED 级；最后，如果有时间就进行 LO 级。

制定主测试计划的目的是在上层文件中弄清全貌，包括主进度表、主要资源利用、主生存周期以及要考虑的质量保证问题。我们要进行什么类型的测试？有多少验证工作？进行什么样的确认？要进行验收测试吗？我们需要什么样的整体策略？

这种“全貌”性的可交付文件之一是软件验证和确认计划。按照 IEEE/ANSI 标准 1012-1986 进行软件验证和确认可以为每一阶段提供综合性的评估，保证在软件的生存周期内尽早地查出并修改错误，减少项目风险、成本和进度影响。从而可以提高软件质量和可靠性。快速评估按建议做出的修改和产生的结果，改进软件过程的透明度。

可交付文件：软件 V&V 计划（大纲）

（IEEE/ANSI，1986[Std 1012-1986]）

- 目的
- 参考文件
- 定义
- 验证和确认概况
  —— 组织、主进度表、资源概要、责任、工具、技术和方法
- 生存周期验证和确认
  —— 每个生存周期阶段的任务、输入和输出
- 软件验证和确认报告
  —— 对所有 V&V 报告的内容、格式和时间分配进行描述
- 验证和确认管理步骤
  —— 政策、步骤、标准、惯例、协议

IEEE 1012-1986 标准内含图表和说明，可以帮助作者制定计划。该计划的总目标是对将要应用到项目中去的 V&V 活动进行描述。IEEE 标准文件定义了计划的轮廓，主要用于对计划的内容进行对照检查。软件 V&V 计划包括在关键软件最低文档需求内，附属于总的软件质量保证计划（SQAP）。

软件质量保证计划处于最高层，用于对有关文件的检验。它提醒人们有这样一个软件 V&V 计划。软件质量保证计划面向关键软件的开发和维护；该标准的次级标准之一可以用于非关键软件。软件质量保证计划的目的是向用户、开发人员和公众显示，采取了具体措施确保软件的

质量。里面包含的内容有配置管理以及与供应商和承包商之间的关系。软件质量保证计划由软件质量保证小组或合适的代表机构制定。

可交付文件：软件质量保证计划（大纲）

（IEEE/ANSI，1989[Std 730-1989]）

- 目的
- 参考文件
- 管理
- 文档编制
- 标准、惯例、协议和度量
- 生存周期验证和确认
- 评估和审计
- 测试
- 问题报告和纠正措施
- 工具、技术和方法
- 代码控制
- 媒体控制
- 供应商控制
- 记录收集、维护和保存
- 培训
- 风险管理

10.2 验证测试任务和可交付文件

概括起来，验证活动包括

（1）活动

（i）需求验证

（ii）功能设计验证

（iii）内部设计验证

（iv）代码验证

（2）验证任务（以活动为单位）包括：

（i）制定计划

（ii）执行

10.2.1 制定验证测试计划

要进行需求验证吗？采用什么样的方法：全方位的正式审查，还是走查或桌面检查？要验证工作产品的哪些区域？要验证全部需求吗？只验证百分之五十的代码吗？不验证的风险是什

么？资源调度情况如何？有哪些责任？

制定验证测试计划要考虑的问题包括：

- 要进行的验证活动；
- 采用的方法（审查、走查等）；
- 工作产品需要或不需要验证的具体区域；
- 不对工作产品某些区域进行验证所涉及的风险；
- 对工作产品的验证范围进一步进行挑选；
- 资源、进度、设备、工具、责任。

每个验证活动都要编写验证测试计划。对于要进行的每个验证活动（即需求验证、功能设计验证、内部设计验证、代码验证），都制定相应的验证测试计划。制定计划的目的是对验证的方法、工作产品中要验证和不要验证的域、有关的风险和优先领域以及其他标准的计划制定信息做出详细的描述。该文件没有相应的 IEEE/ANSI 标准。

可交付文件：验证测试计划（大纲）

- 测试计划标识
- 引言
- 验证活动（要求对……）
- 要验证和不要验证的域
- 任务
- 责任
- 人员配备和培训
- 进度
- 风险和意外事故
- 审批

10.2.2 验证执行

任务：

- 审查、走查、技术评审

可交付文件：

- 审查报告（每次审查一份）
- 验证测试报告（每一验证活动一份）

每次审查后提交一份报告。审查了什么？谁在审查？进度提前了多少？出错率有多高？在什么地方出现了缺陷？有多严重？对于该产品我们得出什么样的结论？对出现的差错修改后是否需要重新审查？审查是否失败？为什么？有多少返工？

可交付文件：审查报告（大纲）

- 审查报告标识

- 测试项目和版本
- 参与人员
- 被审查材料的规模
- 审查小组的准备时间
- 软件元素的处理
- 返工的工作量及返工结束日期预测
- 缺陷清单
- 缺陷概要（分类整理的缺陷数量）

审查报告大纲从“IEEE/ANSI 软件评估和审计标准（Std 1028-1988）”的 6.8 节导出，该标准为报告内容规定了最低要求。

还有一个与验证执行有关的可交付文件：验证测试报告。该测试报告是验证活动的概要。我们的目标是验证所有与软件有关的文档，但最终我们验证了多少？出现了哪些需要解决的内部问题？哪些已经完成？哪些还没有完成？可以把该报告当作一种执行概要，可用于提高管理层对测试过程的意识，引起他们对有关问题的注意。

可交付文件：验证测试报告（大纲）

- 验证测试报告标识
- 验证类型
- 测试项目和版本
- 概述
- 偏差（与阶段开始时的要求的偏差）
- 内部问题（作为独立实体）
- 验证步骤记录（会议、审查、特别的行动等）
- 关键性和风险评估概述
- 公开行动项目清单

每个工作产品应有一个验证测试报告（不是每个审查会议一个）。目的是概括为这一验证活动进行的所有验证步骤（任务）。该文件没有相应的 IEEE/ANSI 标准。

10.3 确认测试任务和可交付文件

确认活动概括如下：

- 单元测试（属开发部门）
- 可用性测试
- 功能测试
- 系统测试
- 验收测试

确认任务：

- 将所有确认活动作为整体，制定高层计划
- 测试件结构设计
- 以活动为单位
  - 制定详细计划
  - 测试件开发
  - 测试执行
  - 测试评估
  - 测试维护

10.3.1 制定确认测试计划

我们准备采用什么样的测试方法？采用何种测试自动化和工具？预算有多大？需要什么样的支持软件和培训？怎样进行配置管理？

制定确认测试计划要考虑的问题包括：

- 测试方法
- 设备（用于测试件开发还是测试执行）
- 测试自动化
- 支撑软件（开发与测试共享）
- 配置管理
- 风险（预算、资源、进度、培训等）

这些方面既适用于整体测试也适用于每个确认活动。

主确认测试计划必须是为整个确认测试工作制定的，必须为每一确认测试活动（单元测试、集成测试、可用性测试、功能测试、系统测试和验收测试）制定一个或多个详细的确认测试计划。

制定主确认测试计划的目的是对整个确认测试工作进行概括。计划内容必须确定具体的确认测试活动、每一测试活动大致需要的资源、每一活动粗略的进度情况、总的培训要求和风险。

可交付文件：主确认测试计划

（IEEE/ANSI，1983[Std 829-1983]）

目的：

- 规定测试活动的范围、方法、资源和进度

大纲：

- 测试计划标识
- 引言
- 测试项目
- 要测试的特征
- 不要测试的特征
- 方法

- 项目通过/不通过的标准
- 中止标准和恢复要求
- 测试可交付文件
- 测试任务
- 环境要求
- 人员配备和培训要求
- 进度
- 风险和偶然事件
- 审批

每一确认测试任务至少要有一个详细的（确认测试）计划。制定详细计划的目的是阐述应怎样进行确认活动：

- 单元测试（属开发部门）
- 集成测试（属开发部门）
- 可用性测试
- 功能测试
- 系统测试
- 验收测试

上述测试计划大纲既可用于主确认测试计划也可用于详细的确认测试计划。大纲中的每一项可用于详细的测试计划，计划对测试内容进行分类，规定要/不要测试的特征。换言之，对于详细的测试计划，每一项仍然适用，只是层次更低、更详细而已。

10.3.2 测试结构设计

测试结构设计是主确认测试计划的重要补充。结构设计是将测试作为整体设计出有意义、高效率结构的过程。怎样组织测试？它们是基于需求、基于功能还是基于内部结构的测试？怎样将它们分类？如果我们将不同的测试一起执行的话，将这些测试进行归类的逻辑关系是什么？怎样构建测试库？

每个重要的软件产品有且只有一个测试结构规格说明。可以将其视为整个测试库的根文件（路线图）。

结构设计需要考虑的主要方面包括：

（1）测试基础（基于需求、功能、还是内部结构测试）；

（2）测试的分类和分类规则；

（3）测试库的结构和命名规约；

（4）一定的执行期间内对测试的分类。

可交付文件：测试结构规格说明

- 测试规格说明标识
- 引言

- 测试库定位
- 测试库存储和访问规定
- 测试库结构/组织
- 标准
- 所有文件的分类和命名规约

10.3.3 测试件开发——设计和实施细节

测试件的设计目标是：

- 尽量检测出错误
- 降低测试开发成本
- 降低测试执行成本
- 降低测试维护成本

测试件是软件，因此优秀的软件设计和软件工程技术也适用于测试件开发。

测试件开发的任务包括：

- 详细设计
- 实施

在此，我们探讨测试设计规格说明的制定。要测试什么？哪些方面要优先考虑？对于相关的项目组，怎样集中高层测试设计？怎样具体实施这些测试？

详细设计是为软件特征或软件特征组指定测试方法细节并确定测试用例的过程。细节设计要考虑的问题包括：

- 实现测试开发目标；
- 符合测试结构；
- 设计各个测试用例。

详细设计的可交付文件包括测试设计规格说明和测试用例规格说明。

详细测试设计的基本步骤

- 确定要测试的项目；
- 以风险为基础确定哪些项目为优先项；
- 对于相关的测试项目组，开发高层测试设计；
- 根据高层设计，设计各个测试用例。

测试项目确定是指确定所有要测试的目标项目的过程。测试项目确定的第一步是对需求和功能设计规格说明仔细研究、分解和分析。采用黑盒方法，为基于功能的测试开发出目标测试项目清单。测试项目是各分测试的对象。根据我们的经验和智慧，为各种相关类型的系统测试而进行的基于需求的测试，开发出另一目标测试项目清单。在有些测试文献中，测试项目称为测试目标，而测试项目的集合叫做项目清单。

可以利用我们手头的所有资源，为各种类型的系统测试而进行基于需求的测试，开发出不同的测试项目清单。对测试设计、专用工具或硬件和环境也许需要专门的方法。对于每种类型

的系统测试，首先考虑该类型是否适用。如果适用，则根据对这种类型系统测试的具体要求开发出测试项目清单。

系统测试类型（基于需求的测试）：

- 容量
- 可用性
- 性能
- 配置
- 兼容性/转换
- 可靠性/可获得性
- 负荷/应力
- 安全性
- 资源利用
- 可安装性
- 恢复
- 可维修性

重复一点，我们不可能面面俱到，但我们不能忽视主要的项目，并且（根据风险）挑选出应该优先考虑的目标测试项目清单。对目标测试项目清单进行提炼时采取的步骤包括：

（1）如果清单是独立开发出来的，则对两份清单进行比较，删除多余的测试项目。

（2）以进度、资源和不测试各项的风险为基础，对各清单上的测试项目进行优选（LO、MED、HI）。

（3）对于每一清单，生成一个覆盖矩阵，（确定具体的测试用例之后）显示测试项目和测试用例之间的映射（即哪些测试用例覆盖哪些测试项目）。

（4）对于关键软件，生成一个需求跟踪矩阵。

评估风险时，必须意识到某种未测试情况对于某一客户可能不会造成什么后果，但对于另一客户可能是灾难性的，这是由于各自不同的使用模式造成的。

对于关键软件，IEEE/ANSI 软件验证和确认计划标准（Std 1012-1986）要求进行：

- 需求可跟踪性分析
- 设计可跟踪性分析
- 代码可跟踪性分析

以上分析体现在需求跟踪矩阵中，采用需求跟踪矩阵的目的是保证对于关键软件不要漏掉什么项目（见图 10-1）。最终对于每一需求都可通过映射方式找到符合该需求和功能设计的测试用例。

需求跟踪矩阵将每一需求与其在功能设计中的目的、在内部设计中的支持、在代码中的支持和该需求的测试集联系起来。如果出现修改，则必须重新验证出处。

对于要测试的典型的实用产品，通常有许多测试设计规格说明。测试设计规格说明是一种概括性文件，帮助确定测试用例。同一测试用例可以通过多个测试设计规格说明确定。每一测试设计规格说明有一个或多个测试用例规格说明。实践中，许多组织将这两个文件合二为一，

省去了更详细的测试用例规格说明。

需　求	功能设计	内部设计	代　码	测　试
饭店有两个订购站	管理屏幕 #2	45 页	12485 行	34、57、63
服务员可以从任一个站预订	订购屏幕	19 页	6215 行	12、14、34、57、92
顾客可能要求另开账单	订购屏幕	39 页	2391 行	113、85
顾客可以从多个站开账单	账单打印	138 页	49234、61423 行	74、104

图10-1　需求跟踪矩阵示范（©1993，1994 Software Development Technologies）

可交付文件：测试设计规格说明

（IEEE/ANSI，1983[Std 829-1983]）

目的：

- 指出测试方法的改进之处，确定设计和相关测试所覆盖的特征。还要确定测试用例和测试步骤（如果有的话）以便完成测试，并指定特征通过/不通过标准。

大纲：

- 测试设计规格说明标识
- 要测试的特征
- 方法提炼
- 测试用例确定
- 特征通过/不通过标准

可交付文件：测试用例规格说明

（IEEE/ANSI，1983[Std 829-1983]）

目的：

- 定义测试设计规格说明确定的测试用例。测试用例规格说明提供输入的具体值和预期的输出值。规格说明还标明由于采用特定的测试用例而引起的对测试步骤的限制。测试用例与测试设计分离开，以便用于多种设计，也可重新用于其他情况。

大纲：

- 测试用例规格说明标识

- 测试项目
- 输入规范
- 输出规范
- 环境要求
- 特殊步骤要求
- 交互用例的相关性

实施是将每一测试用例规格说明翻译成可执行测试用例的过程。实施可交付文件包括：

- 测试用例、测试数据
- 测试步骤规格说明
- 已完成的功能覆盖矩阵
- 已完成的需求覆盖矩阵
- 对于关键软件，已完成的需求跟踪矩阵

测试步骤规格说明一步一步解释如何设置，如何开始测试，如何监视测试的运行，以及测试中止后如何重新开始测试。写好规格说明是非常重要的，如何运行好一个测试库，不能光装在脑袋里，应该以书面形式表达出来。

可交付文件：测试步骤规格说明

（IEEE/ANSI，1983[Std 829-1983]）

目的：

- 确定运行系统需要的所有步骤并运用指定的用例以便实施相关的测试设计。步骤与测试设计规格说明分开，因为步骤必须一步一步地执行，不应包含一些无关的细节。

大纲：

- 测试步骤规格说明标识
- 目的
- 特殊要求
- 步骤的编排

10.3.4 测试执行

测试执行概述如下。任务包括：

- 测试用例选择
- 运行前设置、执行、运行后分析
- 记录活动、结果和事件
- 判断故障是由产品错误还是由测试本身的错误引起的
- 测量内部逻辑覆盖

测试执行是执行所有或挑选的测试用例并对结果进行观察的过程。有些组织（那些具备较为成熟的测试过程的组织）要求产品在进入测试（执行）之前必须达到一定的标准。制定

标准的理论根据是避免对还没有准备好测试的产品进行未成熟测试，不浪费开发及测试人员的时间。

举例：进入测试（执行）的要求

- 产品基本上已经完成。
- 提交测试的产品是向顾客发布的候选产品。
- 该产品版本有关的验证活动都已经完成。
- 代码推导链中的所有规格说明对于该产品版本而言都已批准、定稿。
- 已经执行了测试小组挑选的一组“验收测试”且没有出现事故。验收测试是测试库中合理的子集，执行验收测试而未出现事故则表明该产品具备一定的可靠性，可以进行完整的测试。

测试执行的主要可交付文件包括测试记录、测试事故报告和逻辑覆盖报告（工具输出）。

在开发和测试周期内可能进行多次可用性测试。可用性测试被视为确认活动，因为它在产品执行环境中（真正的、仿真的或实体模型）聘请了真正的用户。既然如此，可能会有许多的可用性测试期，而每期都必须提供测试记录报告和数量不定的测试事故报告。

测试记录保留了测试执行的细节。保留什么？怎样记录？保留多长时间？要说明某个问题是一天能解决的问题还是涉及到很大程度的返工，弄清上述情况是很有用的。

可交付文件：测试记录

（IEEE/ANSI，1983[Std 829-1983]）

目的：

- 按时间顺序记录测试执行的有关细节。

大纲：

- 测试记录标识
- 描述
- 活动和事件记录项目

事故报告是缺陷（即问题、错误）报告的别名。其中最重要的部分是事故描述，这一部分不应只描述出现的情况，而应该将预期结果与实际结果进行比较。

可交付文件：测试事故报告

（IEEE/ANSI，1983[Std 829-1983]）

目的：

- 用文件记录需要进一步调查的测试执行事件。

大纲：

- 测试事件报告标识
- 概述

- 事故描述
- 影响

不管来源如何（测试人员、内部用户、客户等），任何事故报告都必须由提供者标注严重程度。报告的严重性表明对用户运行环境的实际和预期影响。必须建立严重性的标准定义。建议严重性程度的分级为三级以上，五级以下。

在确定优先服务，判断产品发布前是否准备就绪以及在定义各种质量度量方面，严重性概念都是非常重要的。从开发的角度来看，某一已经确定的问题的严重性有两方面的作用：出现该问题的概率和该问题真正出现时产生的影响。从经历过该问题的用户的角度来说，通常只有后一种情况才有关系。

10.3.5 测试评估

测试评估概括如下：

- 测试覆盖评估
- 产品错误评估
- 测试有效性评估

测试覆盖评估是对产品测试用例集合全面性的评价，并决定是否要开发补充测试的过程。全面性评价是基于当前各个层次的测试覆盖性：功能覆盖（采用功能覆盖矩阵）、需求覆盖（采用需求覆盖矩阵）、逻辑覆盖（采用覆盖测量结果）。这项工作的具体可交付文件是合适的补充测试（通常是基于内部的）。

测试执行之前就要弄清功能覆盖和需求覆盖。只有在测试执行提供了逻辑覆盖方面的情况后才能全部完成测试覆盖分析。如果需要增加测试，就要回到测试开发的“详细设计”阶段。

产品错误评估是根据测试执行对产品质量进行评价，并决定是否要开发补充测试的过程。通常代码的20%最有可能出现错误。越快将这20%定位越好，因为这样就知道还剩下多少错误。评价基于被检测错误的数量，其性质和严重性，出错产品的域以及错误检测率。这项工作的具体可交付文件是可能需要的补充测试。

测试有效性评估是相对于测试完成标准，对当前测试工作的整体有效性进行评价，以及决定是停止测试还是增加并继续测试的过程。有效性评价基于测试覆盖评估和产品错误评估。

在此，主要问题有：

- 决定停止还是继续测试？
- 如果决定继续测试，则需要补充哪些测试？
- 如果决定停止测试，怎样撰写测试概要报告？

10.3.6 何时停止

测试完成的传统标准是时间到点了（即分配的测试时间用完了），或完成了所有的测试又没有检测出错误。

但有意义且实用的完成标准还是有的。完成标准应该基于以下各因素：

（1）成功地采用了具体的测试用例设计方法。

（2）每一类覆盖的覆盖率。

（3）错误检测率（即每一单元的测试时间检测出的错误数）低于指定的限度。基于错误检测数量的标准必须注明错误严重性程度。

（4）检测出错误的具体数量（估计存在的错误总量的比率），或消耗的具体时间。

测试执行状态跟踪模型

计算各类测试用例的数量：

- 计划的：计划开发的测试用例。
- 可用的：可用于执行的计划测试用例。（可用的测试用例少于或等于计划测试用例。）
- 执行的：已执行的可用测试用例。（已执行的测试用例少于或等于可用测试用例。）
- 通过的：已执行的测试用例，在最近的执行过程中没有检测出错误。（通过的测试用例少于或等于已执行的测试用例。）

将这四组数字对应于时间（星期、天等等）绘制出来，分析趋势。

如果将测试用例分成以上四类，可以建立基于各产品的矩阵，按时间跟踪或根据通过的测试用例/计划的测试用例之类的比例进行跟踪。举个例子，我们可以决定只有当通过与计划之比超过 97%时才允许发货。测试执行状态图可以提供各类有用曲线，通过对这些曲线的分析可以获得实用的测试管理信息（见图 10-2）。

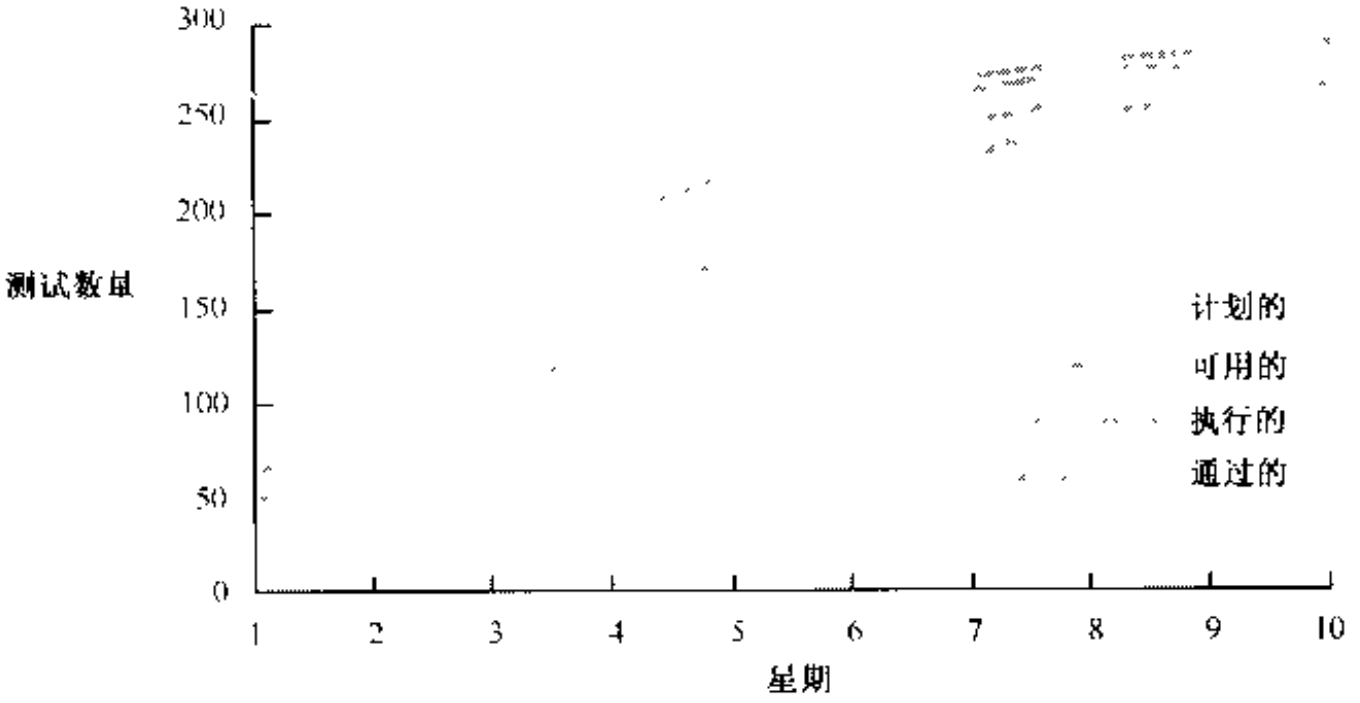


图10-2　测试执行图示范（©1993，1994 Software Development Technologies）

最终文件是测试总结报告，对确认活动结果作出概述：计划内容、从覆盖角度已完成的工作、发现了多少缺陷、其严重性程度如何。也可以将其视为测试活动管理报告。

可交付文件：测试总结报告

（IEEE/ANSI，1983[Std 829-1983]）

目的：

- 总结与一个或多个测试设计规格说明相关的测试活动结果并提供基于这些结果的评

估。

大纲：

- 测试总结报告标识
- 概要
- 偏差
- 综合评价
- 结果概述
- 评估
- 活动概述
- 审批

10.4 用户手册

交付给客户的每一软件产品都包括可执行的代码和用户手册。产品文档的重要性决不低于代码，因为文档的质量是事关产品成败的重要因素。对用户来说，如果用户手册上说要做什么事情，用户照着去做，却不起作用，这时即使代码是正确的也解决不了问题。软件和测试文献一般都不涉及对手册的测试，这也许是因为手册通常都是由技术作家（不是开发人员或测试者）根据生存周期、可交付文件以及没有标准可参考的技术术语撰写的。

“用户文件评估”在 IEEE/ANSI 软件验证和确认计划标准（Std1012-1986）中被定义为任选任务。在开发周期的任何阶段都会出现这方面的任务。用户文件可能包括用户手册或指南，根据对项目的合理性而定。

必须在开发过程中检查文件草稿以确保文件的正确性、可理解性和完整性。必须将手册当作产品的重要部分来对待。必须采用验证（包括计划和报告）的概念和方法对手册进行综合测试。

可考虑采用关键性评审过程。如果认为手册与代码同样重要的话，就应该进行正式审查，审查的方面包括：正确性、可理解性和完整性。如果认为手册没有代码重要，则采用不太正规的办法。对手册进行评审时，可使用附录 B 中的通用文件验证审查表。

如果不折不扣地遵守了功能设计规格说明，便可以用它作为手册的测试基础，因为手册测试的主要目标就是找出两者之间的差异。手册中的范例都应该作为手册测试的一部分接受测试，以判断它们运行起来与描述的是否一样。

10.5 产品发布标准

尽管大多数组织都制定了一定的发布标准，但这些标准往往很不正规。随着公司的成熟，发布标准也越来越具体并可度量。向客户发布新的产品版本（任何版本、不只是第一次发布的版本）的要求应该不光是圆满完成确认测试。典型的发布标准包括：

- 最终产品的所有组成部分，包括用户文件，都准备齐全并经过了测试；
- 定义并明确了软件发布和支持政策；
- 软件制作/分销已准备就绪；

- 软件支持已准备好；
- 客户知道如何报告问题

10.6 IEEE/ANSI 测试文件概述

下面我们概括了用于测试计划和规格说明的所有文件，以及这些文件相互之间、与各种测试活动和标准之间的关系。见图 10-3 和图 10-4。

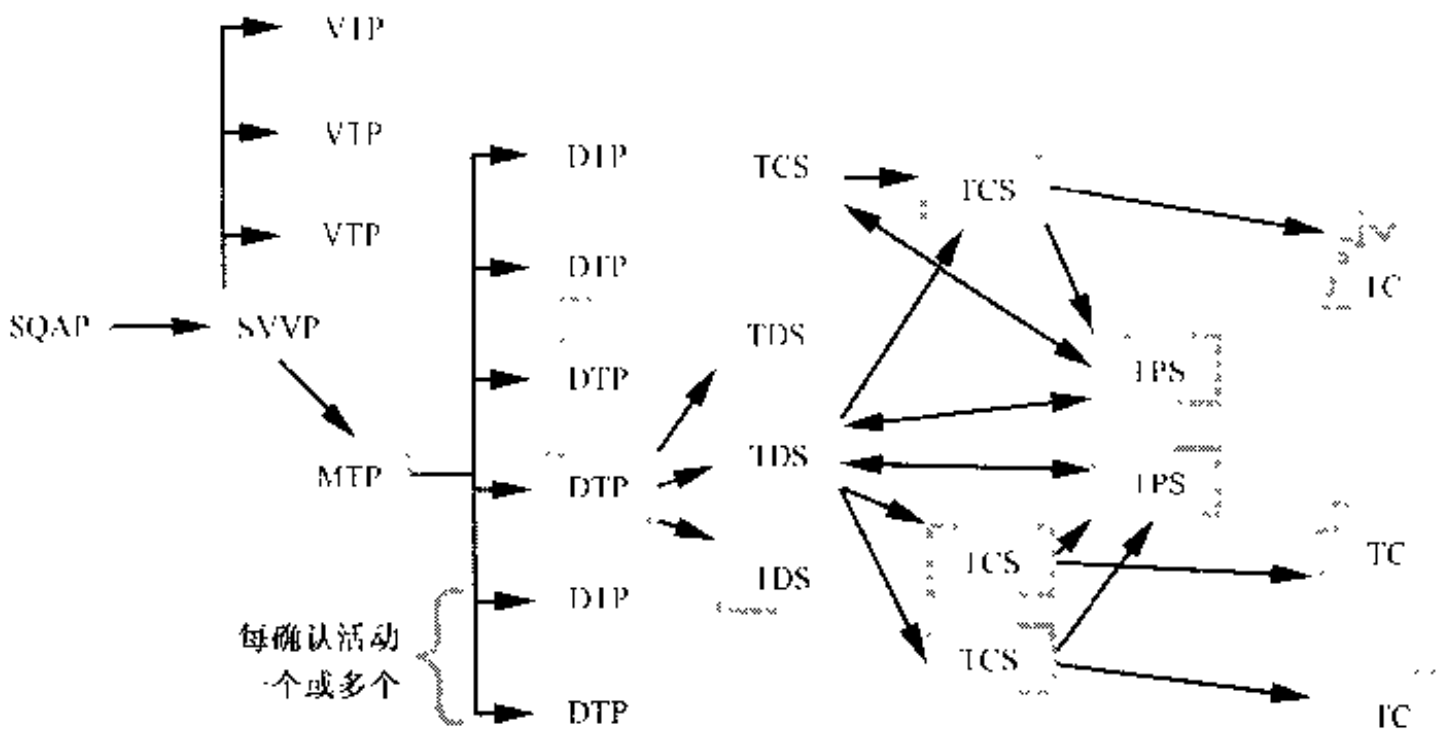


图10-3 测试计划和规格说明的文件结构（©1993，1994 Software Development Technologies）

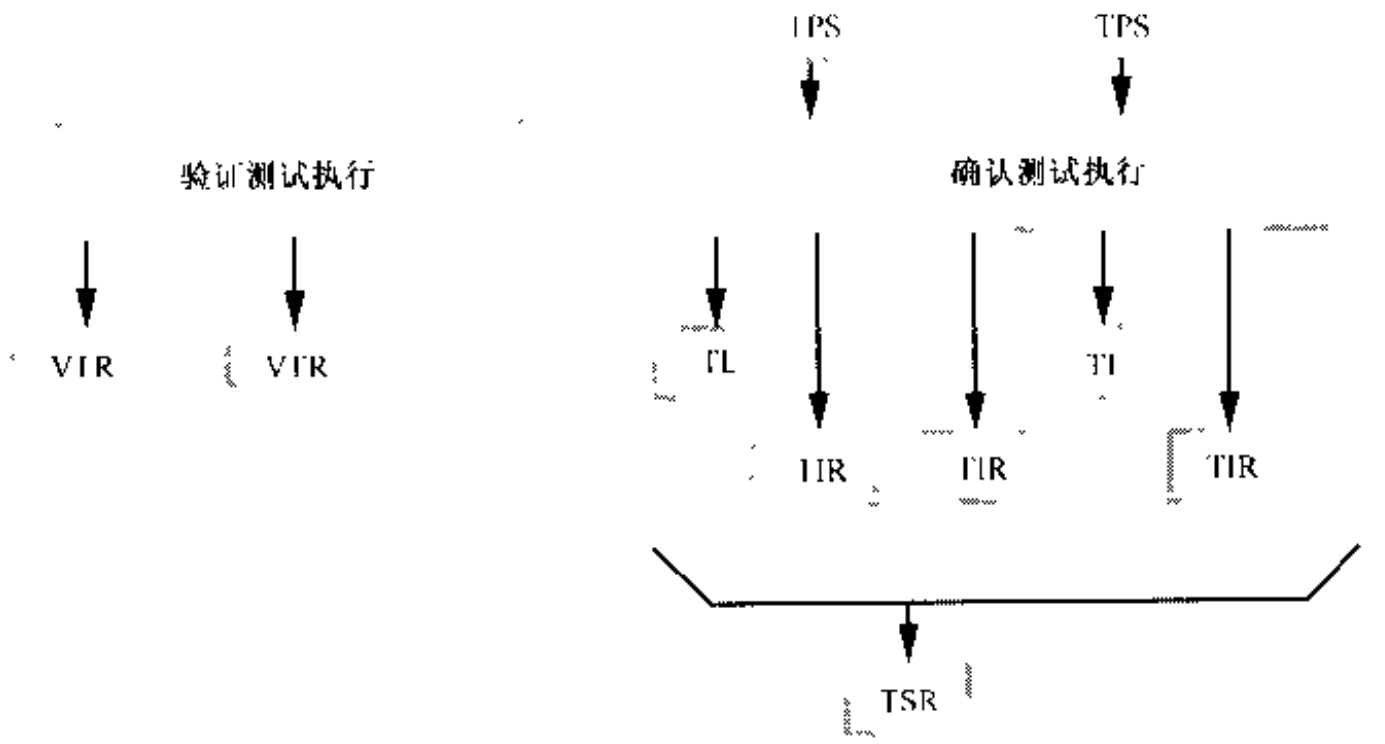


图10-4 测试报告的文件结构（©1993，1994 Software Development Technologies）

10.6.1 测试计划和规格说明的文件结构

SQAP：软件质量保证计划（IEEE/ANSI，1989[Std 730-1989]）。每软件测试产品一个。

SVVP：软件验证和确认计划（IEEE/ANSI，1986[Std 1012-1986]）。每 SQAP 一个。
VTP：验证测试计划：每验证活动一个。
MTP：（主确认）测试计划（IEEE/ANSI，1983[Std 829-1983]）。每 SVVP 一个。
DTP：（详细确认）测试计划（IEEE/ANSI，1983[Std 829-1983]）。每活动一个或多个。
TDS：测试设计规格说明（IEEE/ANSI，1983[Std 829-1983]）。每 DTP 一个或多个。
TCS：测试用例规格说明（IEEE/ANSI，1983[Std 829-1983]）。每 TDS/TPS 一个或多个。
TPS：测试步骤规格说明（IEEE/ANSI，1983[Std 829-1983]）。每 TDS 一个或多个。
TC：测试用例。每 TCS 一个。

10.6.2 测试报告的文件结构

VTR：验证测试报告。每验证活动一个。
TPS：测试步骤规格说明。（IEEE/ANSI，1983[Std 829-1983]）。
TL：测试记录。（IEEE/ANSI，1983[Std 829-1983]）。每测试期一份。
TIR：测试事故报告。（IEEE/ANSI，1983[Std 829-1983]）。每事故一个。
TSR：测试总结报告。（IEEE/ANSI，1983[Std 829-1983]）。一个。

10.7 把任务和可交付文件对应到生存周期

本节将每一测试输入、任务和可交付文件对应到软件生存周期的相应阶段，这些阶段包括：

- 概念
- 需求
- 功能设计
- 内部设计
- 编码
- 集成和测试
- 运行/维护

生存周期模型中的不同阶段可能在很大程度上会重复，但阶段的结束必须按一定顺序进行（可交付文件的审批、签字等）。许多测试任务可能会横跨多个阶段，但在最早的起始阶段就已经显示出来了。测试可交付文件在最后的完成（到期）阶段显示。在此讨论的生存周期是以全套测试为前提的（测试介入的时间不迟于需求阶段）。

10.7.1 概念阶段

（1）测试输入
　　非正式项目讨论
（2）测试任务
　　（ i ）整体规划
　　（ ii ）尽量收集产品和项目的情况
（3）测试件可交付文件

没有

10.7.2 需求阶段

（1）测试输入

（i）软件质量保证计划（任选，参考 SQA）

（ii）需求（来自开发）

（2）测试任务

（i）计划（验证和确认计划分别制定）

（ii）对需求进行分析

（iii）对需求进行审核

（iv）开始确定、分列并设计基于需求的测试，同时还制作需求覆盖或跟踪矩阵

（3）测试件可交付文件

（i）软件 V&V 计划

（ii）验证测试计划（针对需求）

（iii）验证测试报告（针对需求）

10.7.3 功能设计阶段

（1）测试输入

功能设计规格说明（来自开发）

（2）测试任务

（i）功能设计验证、确认计划

（ii）分析功能设计规格说明

（iii）审核功能设计规格说明

（iv）开始进行可用性测试

（v）开始确定、分项并设计基于功能的测试，同时还制作功能覆盖矩阵

（vi）开始实施基于需求和基于功能的测试

（3）测试件可交付文件

（i）（主确认）测试计划（IEEE/ANSI，1983[Std 829-1983]）

（ii）验证测试计划（针对功能设计）

（iii）验证测试报告（针对功能设计）

10.7.4 内部设计阶段

（1）测试输入

内部设计规格说明（来自开发）

（2）测试任务

（i）内部设计验证计划

（ii）分析内部设计规格说明

（iii）审核内部设计规格说明

（iv）开始确定、分项并设计基于内部的测试

（3）测试件可交付文件

（i）（详细确认）测试计划（IEEE/ANSI，1983[Std 829-1983]）

—— 一个或多个确认活动

（ii）验证测试计划（针对内部设计）

（iii）验证测试报告（针对内部设计）

（iv）测试设计规格说明（IEEE/ANSI，1983[Std 829-1983]）

10.7.5 编码阶段

（1）测试输入

代码（来自开发）

（2）测试任务

（i）代码验证计划

（ii）分析代码

（iii）验证代码

（iv）设计基于外部的测试

（v）设计基于内部的测试

（3）测试件可交付文件

（i）测试用例规格说明（IEEE/ANSI，1983[Std 829-1983]）

（ii）需求覆盖或跟踪矩阵

（iii）功能覆盖矩阵

（iv）测试步骤规格说明（IEEE/ANSI，1983[Std 829-1983]）

（v）验证测试计划（针对代码）

（vi）验证测试报告（针对代码）

（vii）验证测试件（功能和系统测试）

10.7.6 集成和测试阶段

（1）测试输入

（i）用户手册草稿

（ii）要测试的软件

（iii）用户手册定稿

（iv）测试项目传输（IEEE/ANSI，1983[Std 829-1983]）

（2）测试任务

（i）制定计划

（ii）审查由开发部门进行的模块和的集成测试

（iii）进行功能测试

（iv）进行系统测试

（v）审查用户手册草稿和最终版本

（3）测试件可交付文件

（i）测试记录（IEEE/ANSI，1983[Std 829-1983]）

（ii）测试事故报告（IEEE/ANSI，1983[Std 829-1983]）

（iii）测试总结报告（IEEE/ANSI，1983[Std 829-1983]）

10.7.7 运行/维护阶段

（1）测试输入

（i）（见注）

（ii）已确认的问题报告（任何来源）

（2）测试任务

（i）监视验收测试

（ii）为确认的问题开发新的确认测试

（iii）对所有测试的持续有效性进行评估

（3）测试件可交付文件

升级的测试库

注：软件生存周期是一个重复过程。产品首次发布后，如果对产品进行了修改，则要求开发和测试活动回到与修改相对应的生存周期阶段。例如，如果产品增加了新功能（不是因为修改了需求而出现的新功能），则要求制定新的功能设计规格说明，而过程必须回到功能设计阶段并自此按顺序继续下去。换言之，不会因为产品已经向客户发布了，就将所有开发和测试活动都移交到运行/维护阶段。

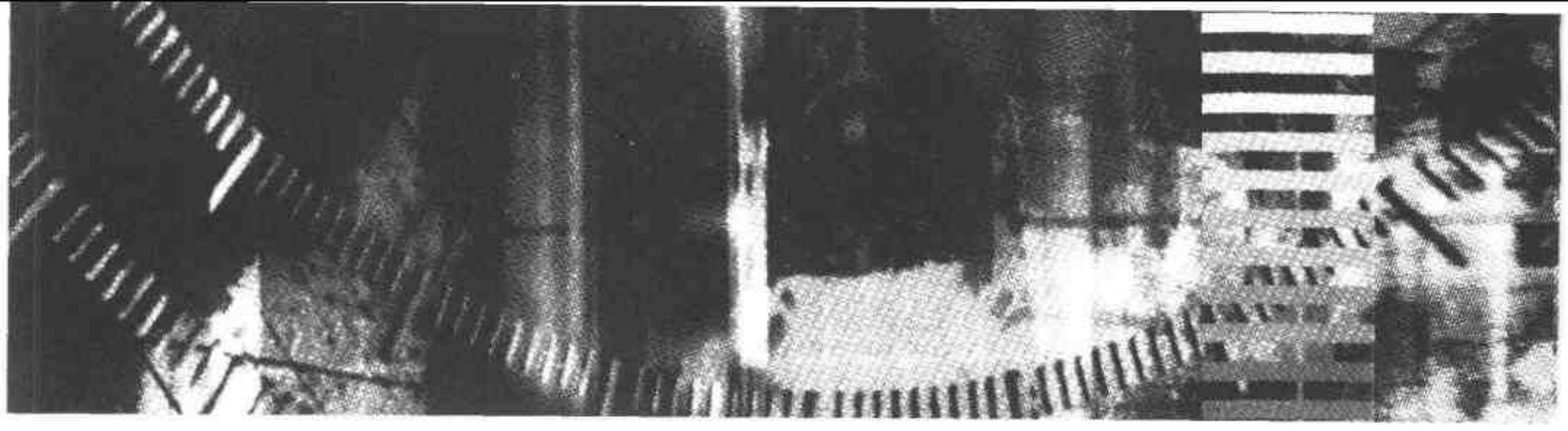
10.8 参考文献

IEEE/ANSI（1983）.IEEE Standard for Software Test Documentation,（Reaff. 1991）,IEEE Std 829-1983.

IEEE/ANSI（1986）. IEEE Standard for Software Verification and Validation Plans,（Reaff.1992）,IEEE Std 1012-1986.

IEEE/ANSI（1988）.IEEE Standard for Software Review and Audits, IEEE Std 1028-1988.

IEEE/ANSI（1989）.IEEE Standard for Software Quality Assurance Plans, IEEE Std 730-1989.



第11章

软件测试工具

采用测试工具可以降低测试工作的难度，提高测试工作的效率和生产力。一点都不奇怪，参加测试课程培训的人，要实现的首要目标之一就是：我应该为我的组织购买什么样的工具？

有各种各样的计算机辅助软件测试（CAST）工具，可用于测试过程的许多方面。它们的应用范围和质量相差很大，提供辅助的程度也各异。

要让工具在测试工作中发挥作用，制定统一的评估、采购、培训、实施和维护计划是必不可少的。在这方面，掌握一定的专业知识是非常有益的。

11.1 测试工具的分类

测试工具的分类方法有如下几种：

（1）按工具使用的测试活动或任务划分 —— 在此，活动包括代码验证、测试计划、测试执行；

（2）按描述性功能关键词 —— 工具实现的具体功能，如捕获/回放、逻辑覆盖、比较器；

（3）按主要的划分区域 —— 工具的少数几种高层分类或归类。

每一类工具在功能或其他特征方面具有相似之处。这方面的例子包括：测试管理工具、静态分析工具和仿真器。

本章主要讨论与工具有联系的活动，即：

- 评审与审查
- 制定测试计划
- 测试设计与开发
- 测试执行与评估
- 测试支持

我们采用这种分类方法是出于两方面的原因。首先，它最接近于测试人员的观点——测试人员正在做什么以及什么时候做。第二，它与测试标准一致，而且基于测试标准。

附录G列出了一些以描述功能分类的具体工具。总的来说，在测试过程的前端，可获得

的专门工具比后端要少些。

11.1.1 评审与审查工具

评审与审查工具是用于对需求、功能设计、内部设计和代码进行评审、走查与审查的工具。有些工具是用于规格说明的，但专用于代码的工具要多得多。

所要求的工具类型包括：

- 复杂度分析
- 代码理解
- 句法与语义分析

复杂度分析

有经验的程序员知道，80%的问题是由于20%的代码引起的。复杂度分析有助于发现这最紧要的20%。《McCabe复杂度度量》（McCabe Complexity Metrics）于1982年首次发表在NBS（国家标准局）的出版物《结构测试：采用回路复杂度度量的软件测试方法》上。

复杂度度量方法能确定复杂域中的高风险。回路复杂度度量基于程序中的一系列判定。对于测试人员来说这种度量是非常重要的，因为它提示了为有效地避免故障所需要的测试量（包括审查）。换言之，那些标明为较复杂的代码域是必须补充测试进一步审查的候选域。还有其他种类的复杂度度量法（例如McCabe和Halstead提供的），一般都是程序测试成本/进度和程序中的缺陷的指示器。

代码理解

代码理解工具能帮助我们了解不太熟悉的代码。这些工具能帮助我们理解相关性、跟踪程序逻辑、观看程序的图形表达并确认死代码。利用这些工具可以成功地确定需要特殊关照的域，例如需要审查的域。

准备代码审查会议要花大量的时间。需要广泛的分析、理解和逆向工程，采用代码理解工具可以使这些工作更加容易。

句法和语义分析

句法和语义分析工具进行广泛的错误检查，找出编程员忽略的错误，有时在正式测试之前或在正式测试的过程当中用于标识潜在的缺陷。

这些工具是与语言（有时是方言）相关的。以C语言为例，由于存在各种方言，这些工具通常可以通过方言配置。工具可以对代码进行句法分析，对错误进行记录并提供结构信息。句法分析器可以发现语义错误，并指出句法上不连贯的地方。

11.1.2 制定测试计划的工具

制定测试计划的目的是定义测试活动的范围、方法、资源（包括工具）和进度。测试计划为整个测试过程提供基础，测试是一种脑力劳动，制定计划是必不可少的。工具不能代替思维。

尽管捕获/回放工具非常有用（见下面的 11.1.4 节“测试执行和评估”），但它们并不能取代周密的测试计划和设计。

也许在这方面最管用的是标准。IEEE/ANSI 软件测试文档标准（Std 829-1983）对测试计划的目的、大纲和内容进行了描述。标准的附录收集了一些商业数据处理范例。尽管有些工具包含了测试计划模板，但有些公司认为采用下面的办法也很有用：即让人将 IEEE/ANSI 标准内的测试计划大纲输入到可访问的编辑文件中。

有些很有用的商业工具，可以为全部产品测试确定具体的项目人员和进度需求。人们往往抱怨进度是由上面的管理层事先确定了的，采用这类工具可以反映项目的客观情况。

制定测试计划所需要的工具类型包括：

- 测试计划文件编制模块
- 测试进度和人员安排估计
- 复杂度分析器

有助于评审和审查的工具对制定测试计划也同样有用，即可以确定复杂产品域的工具也可以用于确定对计划制定产生影响的域，计划制定用于基于基本风险管理的补充测试。

11.1.3 测试设计和开发工具

测试设计是详细说明软件特征或特征组合的测试计划所指定的整体测试方法，以及确定并选择相关测试用例的过程。测试开发是将测试设计转换成具体的测试用例的过程。

像制定测试计划一样，对最重要、最费脑筋的测试设计过程来说，工具起不了多大的作用。但测试执行和评估类工具，如捕获/回放工具，是有助于测试开发的，也是实施计划和设计合理的测试用例最有效的手段。

测试设计和开发需要的工具类型包括：

- 测试数据生成器
- 基于需求的测试设计工具
- 捕获/回放
- 覆盖分析

同样，标准是能发挥作用的。IEEE/ANSI 软件测试文档标准（Std 829-1983）对测试设计规格说明的目的、大纲和内容进行了描述，标准的附录还收集了一些商业数据处理范例。

尽管有些工具包含了测试设计规范模板，但许多公司认为采用下面的办法也很有用：即让人将 IEEE/ANSI 标准内的测试设计规范大纲输入到可访问编辑文件中。

有一种测试数据生成工具非常有用，它能使基于用户定义格式的测试数据生成自动化，例如，能自动生成具体由用户指定输入事务的所有排列。

基于需求的测试设计工具至今还没有获得广泛的实际应用。根据故障需求可能占错误成本的 80%这一假设，可使用基于因果图理论的高度规范的方法，设计测试用例以确保实现的系统符合正式规定的需求文件。对于那些希望采用规范、严格且有条有理的方法的人，这种方法很合适。

11.1.4 测试执行和评估工具

测试执行和评估是执行测试用例并对结果进行评估的过程。这一过程包括选择用于执行的测试用例、设置环境、运行所选择测试、记录执行活动、分析潜在的产品故障并测量这项工作的有效性。评估类工具对执行测试用例和评估结果这一过程起辅助作用。

测试执行和评估所要求的工具类型包括：

- 捕获/回放
- 覆盖分析
- 存储器测试
- 仿真器及性能

捕获/回放

有经验的测试人员知道，最烦人的事情莫过于不断的重复运行手工测试。测试人员求助于捕获/回放工具使测试执行自动化，换言之，就是根据需要，可以几个小时、一个晚上或一天24小时不间断运行测试而不需值班管理。

捕获/回放工具可以捕获用户的操作，包括击键、鼠标活动，并显示输出。这些被捕获的测试，包括已被测试人员确认的输出，为今后的产品修改测试构成基线。需要时，工具可以自动回放以前捕获的测试，并通过与以前存储的基线进行比较而对结果进行确认。因此，当修复故障并进行产品增强而对产品作出修改时，测试人员不需要通过手工反复不断地重新运行测试。

捕获/回放工具可以分为本机式和非侵入式两种。本机式（有时称为侵入式）捕获/回放是在一个系统内进行的。捕获/回放工具和被测试的软件处于同一系统，即测试工具处于受测试系统的“本地”。有时称为侵入式是因为捕获/回放软件在一定程度上会影响操作性能，当然对于大多数软件测试来说，这种影响是没关系的。

非侵入式捕获/回放需要为测试工具增加硬件系统。通常主机系统（包含被测试软件）与捕获/回放工具之间有专门的硬件连接，从而可以使捕获/回放系统以对主机软件透明的方式发挥功能。最好的非侵入式工具独立于平台和操作系统。

有三种形式的捕获/回放，排在前面的最便宜，排在后面的最贵：

（1）本机式/软件侵入式（在被测试系统内的软件层面上产生干扰）；

（2）本机式/硬件侵入式（只在硬件层产生干扰）；

（3）非侵入式（不产生干扰）。

使用最多的类型是本机/软件侵入式。非侵入式通常用于被测试产品本身是集成硬件和软件系统的，不能再容纳更多的内部硬件或软件，例如，实时嵌入式系统。大部分的软件测试没有这种限制，因此本机/软件侵入式也就成了大多数组织通常采用的性价比高的解决方案。

覆盖分析

覆盖分析器对测试质量提供定量测量。换言之，覆盖分析器可以发现对软件的测试是否充

分。这种工具对所有软件测试组织都是必不可少的，它告诉我们接受测试产品的哪些部分已被当前测试所执行（覆盖）。这类工具还会告诉我们软件产品具体有哪些部分还没有覆盖到，需要进一步的测试。

有些公司认为，不必要覆盖所有语句，即不必在发布之前执行产品内的全部语句。他们似乎觉得让客户首次执行代码没有什么不妥。也许这些公司采取了另外的策略。如果我们不对覆盖进行测量，就还没有把握测试人员的工作要领。

几乎所有结构工具都将源代码载入预处理程序，实现对覆盖信息的跟踪。问题是现有的新源程序比原来的大，因此我们的目标模块在规模上会增加。另一潜在的问题是性能可能受到影响，因为现有的程序与原来的不一样。但是，软件发布的最后版本不包括上面的预处理步骤，因此也就不会在规模和性能方面受到影响。

覆盖形式多种多样，包括语句、判定、条件、判定/条件、多条件和路径。至少，首先应该确保程序中的每一条语句受到过测试，而且对于所有可能出现的结果每一判定至少作出过一次。

存储器测试

不管是叫做边界检验器、存储器测试器、运行时错误检测器还是漏洞检测器，这类工具一般来说应能检测：

- 存储器问题
- 重写并/或重读阵列界
- 已分配但未释放的内存
- 读出并使用未初始化的存储器

错误在生产过程显现出来之前可以将其确认，它们可能引起严重的问题。详细的诊断信息可以跟踪并消除错误。

尽管存储器测试工具往往是语言专用及平台专用的，但还是有些销售商生产的工具可用于最常见的环境。这方面最好的工具是非侵入式的，且使用方便、价格合理。也就是实用的一类工具，特别是将低质量的应用考虑在内时，情况尤其如此。

测试用例管理

我们会渐渐感觉到需要测试用例管理工具。我们使用捕获/回放工具建立测试并使其自动化。然而有一天我们一觉醒来发现有成千上万的测试，杂乱无章，需要管理。

最好的测试用例管理器能：

- 提供用户界面用于管理测试；
- 对测试进行整理以方便使用和维护；
- 启动并管理测试执行期，运行用户选择的测试；
- 提供与捕获/回放及覆盖分析工具的无缝集成；
- 提供自动化的测试报告和文件编制。

为什么要将测试用例管理另归一类，即为什么不将其集成到现有的捕获/回放工具中去？

不幸的是我们还做不到这一点。但可喜的是已有几家处于领先地位的工具销售商他们正在积极努力以实现这一目标，再过一年左右就会出现一些成型的产品。

仿真器和性能

仿真器取代与被测试软件交互作用的软件或硬件。有时对于某些测试来说，他们是能获得的惟一实用的方法：例如，当软件与无法控制或不能获得的硬件设备接口时就是这样。仿真器经常用于测试电信应用程序、通信存取方法、控制程序和网络。

利用仿真器还可以检查系统性能。一般来说，性能工具有助于确定软件和系统性能。实际当中，有时很难找到区分仿真器与性能工具的分界线。

最后，还有些工具可用于自动多用户客户/服务器加载测试和性能测量。这些工具可用于生成、控制并分析客户/服务器应用的性能测试——在这些应用处于在线之前。

11.1.5 软件测试支持工具

测试支持工具不是测试过程的主体，它们对整体测试工作提供全面的支持。如果这些工具的质量差或没有，专业测试人员就会很难受。

测试支持需要的工具类型包括：

- 问题管理
- 配置管理

问题管理

问题管理工具有时称为缺陷跟踪工具、故障管理工具、事故控制系统等，用于在整个软件产品生存周期中对缺陷和强化管理的记录、跟踪并提供全面的帮助。

尽管许多公司花费大笔经费开发内部的问题管理系统，但现在已有工具销售商开发跨多种平台的系统。最好的问题管理工具很容易根据特定的环境进行定制，并具备如下标准特征：

- 能十分容易并迅速地提交和更新缺陷报告；
- 能十分容易地生成预定义或用户定义的管理报告；
- 能十分容易并有选择性地自动通知用户对缺陷状态的修改；
- 能很容易地根据用户提问提供对所有数据的安全访问。

配置管理

配置管理（CM）是对文件修改以及其他紧要事物进行管理、控制和协调的关键。CM 工具协助版本控制并构建管理过程（见第 6 章）。

除问题管理和配置管理之外，还有许多与测试无关的工具，对测试过程提供支持。这些工具包括项目管理工具、数据库管理软件、电子表格软件以及字处理器。

11.2 工具采购

工具采购问题主要涉及良好的管理常识，实施起来并不特别困难。购买工具，绝大多数情

况下都是有针对性的为了解决短期问题，结局往往是将其束之高阁，又多了一件价值昂贵的设备而已。

决定购买工具时应该进行成本/收益分析，不管多简单都应如此。工具销售商往往热衷于介绍他们的工具能做什么，如何能解决我们的具体问题。我们必须问的是："成本有多高？"

关于成本，重要的是确定实际成本——总成本，甚至生命期成本。确定实际成本主要靠大概估计，但比不做要好得多，而且产品价格通常只是最基本的成本，附加成本还包括挑选、安装、运行、培训、维护和支持，以及改组过程的总成本。

与需要组织更换测试过程相比，支持已有测试过程的工具在人力和财力方面实施起来要容易得多。

首次采用的范围必须确定。新工具是在一个小组内，甚至在挑选的几个人内采用，还是立即在整个组织范围内采用？

早期的管理支持很关键，必须确保实际使用新工具的人员可以集中精力，尽早组织起来，投入运行并拿出成果。

精心挑选并实施相互支持且具有协同效果的工具也是提高生产力的一个重要方面。

采购工具之前的提问

为高效率地实施工具项目，建议对下面的问题作出回答：

- 工具怎样介入并支持测试过程？
- 知道怎样计划并设计测试吗？（工具不能代替思考、计划和设计。）
- 谁负责新工具的培训工作？
- 谁连续负责推广并支持工具在组织范围内的使用？

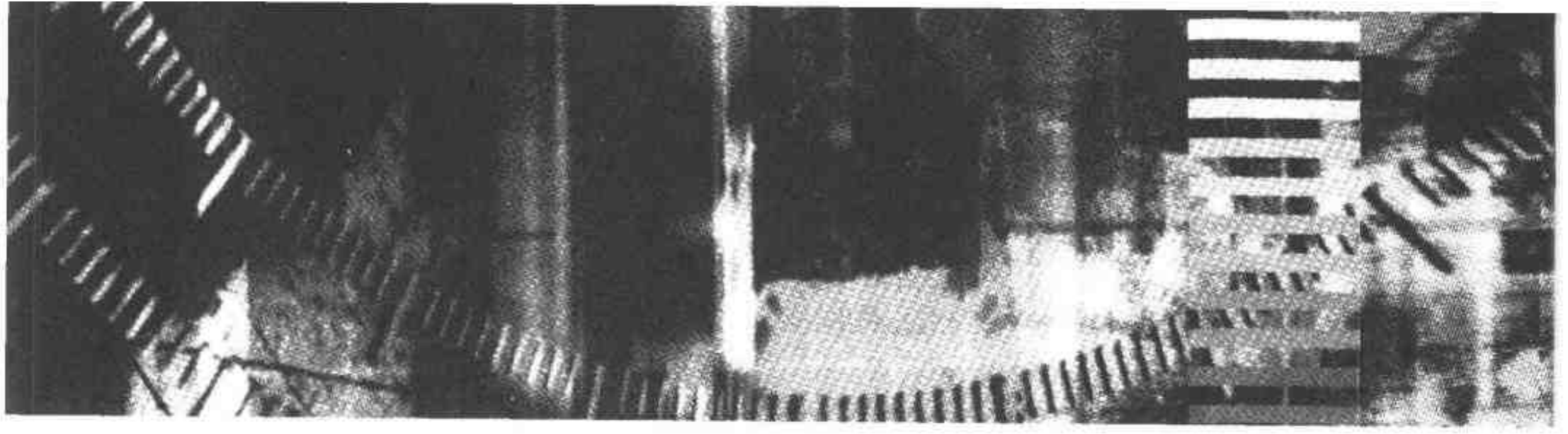
难就难在从合适的销售商购买合适的工具。提供并保持对现有测试工具及其能力的评估是一项十分重要的工作。市面上有好几百种测试工具，开发公司在规模、已建立的客户库、产品成熟度、管理深度、以及对测试和工具的理解方面差别很大。

关于工具及工具销售商选择过程更详细的情况，参考附录 G。

早期专家提供的建议是非常宝贵的，从评估过程并确定哪些工具有用这一阶段一直到实施阶段尤其如此。毫无疑问，使用工具可以使测试更容易、更有效、更高产。通过适当的步骤，我们可以避免将昂贵的新工具束之高阁。

11.3 参考文献

IEEE/ANSI（1983）. IEEE Standard for Software Test Documentation,（Reaff.1991）,IEEE Std 829-1983.



第12章

度　量

测试工作有一些大事情，这些事情包括：产品质量、风险管理、发布标准、测试过程的效率以及何时停止测试。

度量能提供答案。但一旦我们开始考虑要度量的对象，就会不知所措，因为几乎每样东西都可以度量。但度量每样东西是不现实的（与我们不能测试每样东西的原因相同），我们必须为度量进行优选，根据是哪些度量最关键并且会真正投入实际应用。

为度量而度量会浪费精力。我们应该问一问："度量有用吗？我们从中能得到什么好处？假如我们从特定的域获得了可靠的度量结果，我们能对某一重要的问题给出满意的回答吗？"也许，只有等进行了具体的度量之后我们才能提出一些重要的问题。

12.1　通过度量获得答案

我们的计划和随后的活动要有效果的话，就必须以可靠的事实为基础。测试要花多长时间？测试的效率有多高？测试库有多好？做验证而不是确认值得吗？确认测试有多充分？

根据以往的经验，宣布产品"准备好了，可以测试"时，该产品可能处于什么样的状态？可能会发现何种错误，有多少，在哪儿？测试后，有多少错误还可能遗留在产品内？与市场上的其他产品相比，该产品在测试和生产中表现如何？

对验证检测出来的错误数量和类型进行度量可以确定验证的效率。验证是昂贵的，尽管我们可能相信验证的效率高，但我们必须提供理由使别人相信这一点。在确认过程中我们发现了多少本应该是在验证过程中发现的错误？

通过度量确认测试覆盖（需求、功能和逻辑覆盖），可以对确认测试的全面性和综合性进行定量评估。我们测试了产品的多大部分？测试库有多好？

度量/跟踪测试执行状态可以集中显示关键测试范畴（计划、可用、执行、通过），并提供定量信息确定何时必须终止测试。计划了多少测试？有多少可用？多少已被执行？通过了多少？何时可以（合理地）停止下来？

在测试开始之前，对测试工作量进行规划以及对错误数量进行估计时，程序复杂度为我们

提供了有效答案。

对事故（经过严格分类）报告的度量和跟踪是产品质量的重要指标。它为产品的发布提供了一种客观标准，可以预测出剩余的错误数量，一般与用户对产品的满意程度相关。规范化后，它还可以对产品与产品之间的相对质量提供度量。

如果不对事故报告进行跟踪，公司就不能以负责的态度对问题进行处理。事故会不断积累，最终导致公司不再能制定计划对它们进行管理或将它们降低到合理的程度。对于这类事态的发展，度量可以提供早期警告。

客户而不是测试提供的事故报告（经过诊断被确认的错误）为测试的有效性提供了另一种度量。测试人员掌握客户发现的缺陷是非常重要的，因为它们是改进测试库的宝贵基础（见 Hetzel，1993）。

12.2 有用的度量

12.2.1 度量复杂度

一般地说，程序的组件越复杂，包含的错误也就越多，同时也就越需要测试工作集中力量将它们找出来。这方面，有许多现成的工具，非常不错，而且也没有必要了解这些工具的具体工作原理，就可以有效地发挥它们的作用。

对于程序复杂度，有几种著名的度量方法。大多数的复杂度度量只能在编完程序之后才能计算。这类度量对于一些确认任务和缺陷预测是非常有用的。

一种简单的复杂度度量是对代码行（LOC）或源语句的度量，可以通过几种方法进行计算，这实际上是一种长度度量。某一特定的行可能是空白、注释、一条或多条可执行的语句和/或数据声明。还有一个问题就是比较用不同的源语言编写的程序 LOC。规范 LOC 的最简单的办法就是生成与汇编语言等价的程序，然后再计算 LOC。我们必须决定如何计算 LOC，然后在组织内制定标准。花费大量的时间讨论计算方法并不十分必要，重要的是采用相同的工具并通过相同的方法进行精确地度量。

除计算代码行外，另一办法是功能点数。与代码行一样，功能点数也是对规模、工作量和复杂度的度量。与代码行不同的是，功能点数是根据功能规格说明的细节，从用户的角度导出的，永远独立于编程语言。从测试过程的角度来看，功能点数一直用于估计每一功能点数所需要的测试用例，以及对每一功能点数的缺陷进行度量。

实践中，是度量代码行还是度量功能点数，对此常常引起激烈的争论。喜欢功能点数的人认为代码行是一种陈旧、不准确的方法，有许多明显的缺点。而功能点数的不足之处是太抽象，与软件开发商的实际生产没有密切关联，需要培训才能学会计算，对于非专业人员来说又要涉及一个十分复杂的过程。

就目前情况来看，代码行和功能点数在软件生产中都占有一席之地，值得考虑。从长远来看，功能点数或其变体将会变得容易理解、计算及使用，有可能最终完全取代对代码行的度量。

另一种复杂度度量方法是 McCabe 复杂度度量，它等于程序中的判定数（+1）。N 条分支等价于 N-1 个判定。子程序之间的复杂度是添加的。其结果是复杂度数，如果超过一定的限制，就表示需要特别注意，比如进行审查或补充进行确认测试。许多优秀的工具具备自动计算 McCabe 复杂度的能力。

Halstead 的度量法用于计算程序长度（不要与代码行混淆）。可以在编写程序之前对程序长度进行预测，对于很多的程序来说，预测和实际值非常接近。还有些公式可用于预测故障数、编程工作量和时间。

12.2.2 度量验证效率

验证活动效率是一种很重要的度量，因为验证十分很昂贵。验证测试报告必须包含每一验证活动检测出的具体错误清单（见第 10 章）。利用随后从确认活动获得的错误数据，我们可以计算出验证检测出来的错误与忽略的（未检测到的）错误。

12.2.3 度量测试覆盖

利用需求覆盖/跟踪矩阵和功能覆盖矩阵，可以以手工方式度量基于需求的和基于功能的测试覆盖。而逻辑覆盖（实际上）只能采用自动工具进行度量。度量语句覆盖是当今采用的现成的覆盖工具中最常用的做法，一般也是较为合理的起点。

12.2.4 度量/跟踪测试执行状态

测试执行跟踪最简单的办法是采用电子数据表。表格的列包括时戳和四个测试类：计划的、可用的、执行的、通过的。每一行是对每一类测试用例的定期（如每周、每天）观察。前后观测之间的时间应该很短，以使有足够的观测次数，可以观察到发展趋势。可以以图形的方式自动提供电子表格，以便更容易理解发展趋势。通过的测试与计划的测试的预定比（如 98%）通常用作发布的标准之一。

12.2.5 度量/跟踪事故报告

事故或“错误”报告可以为软件提供许多宝贵的质量度量基础。为充分发挥事故报告的潜力，事故报告过程及其自动支持系统应遵守以下原则：

（1）必须有一种且只有一种报告事故的方法。冗余的报告机制会惹出一些不必要的劳动。

（2）所有的事故报告只有一个主库。如果数据破碎、冗余或不完整，要获得准确、完整且及时的信息是很困难的。

（3）必须采用正规机制报告每一事故。对于未报告的事故是永远也不会调查的。非正式的报告往往会被遗漏，且状态得不到跟踪。

（4）事故报告必须严格确定事故发生之处的软件配置。用户必须能够从运行的系统中动态地获得所有必需的版本信息。从运行的系统中获得的版本标识是可以相信的。

（5）产品的每一用户，不仅是客户，都必须花时间对事故进行报告。这里所说的用户包括内部用户、开发人员、测试人员和管理人员。

（6）产品新版本纳入基线后，便可以进行正式测试并准备发布（处于集成和测试阶段），随后出现的所有事故都必须正式进行报告。发布前检测出来的问题与发布后检测出来的问题同样重要。

（7）必须对每一事故报告进行调查，然后将其归纳为以下类别之一：

（i）用户/操作人员错误

（ii）无法重复报告的问题

（iii）信息不够

（iv）文件编制（用户手册）错误

（v）请求修改/强化

（vi）已确认的产品错误

（vii）其他

（8）由于原始事故报告描述问题的症状（即问题对于用户的表现形式），报告必须包含足够的篇幅用于真实错误的精确描写。许多质量度量是基于已确认的错误数目而不是事故报告，必须能计算出具体产品中已确认的错误数目，而不是错误的症状。这里的产品也不一定是事故报告中确认的产品。应该有多个事故报告，每一个报告描述不同的症状，它们都对同一错误进行描述，但不要求在同一报告中对该错误进行冗余描述。

（9）对于已确认的问题，报告也必须包含足够的篇幅记录错误的根原因归类。优秀的软件工程组织已培育出一种文化，对每一确认的错误都进行根原因分析，然后将其用于错误预防程序之中。根原因往往被严格分成一系列的标准原因。

（10）事故报告系统应能提供完整的跟踪信息，从报告发起的时候开始，到报告正式结束。为对过程进行管理，必须开发状态转移图，以便显示事故报告中所有可能的状态、引起状态改变的事件以及被授权对状态进行修改的组织。

（11）只有与报告相关的所有工作都结束之后，报告才算结束。对于已确认的错误，在报告结束之前必须向客户提供纠正办法。

12.2.6 基于事故报告的测试度量

每 1 000 代码行确认的错误个数（错误个数/KLOC）是常用的错误密度度量方法，为产品质量提供了指标之一（另一种办法是每一功能点数确认的错误个数）。这也是一种产品之间的质量比较。已确认的错误数量本身也是对剩余错误数量的预测，因为在一个程序中未发现的错误数量是与已发现的错误数量成正比的。

发布标准的关键因素之一是已确认但未修改的错误数量（各严重性等级），他们构成向客户进行产品发布的可接收性极限。将用户和客户报告的确认错误数量与测试报告的数量进行比较可以表明测试度量的效率。用户/客户报告的错误还可以为新的测试用例提供基础。

12.3 其他的相关度量

对于软件工作者来说，还有许多其他的度量。常见的有：

（1）错误检测出来之前的存在时间；

（2）对报告的故障进行修复的反应时间；
（3）通过根原因类检测出来的错误的比率与频率；
（4）改正错误的效率；
（5）错误成本：
（i）失效对用户造成的损失
（ii）调查及诊断成本
（iii）修复成本
（iv）重新测试成本
（v）发布成本

12.4 建议

- 对于组织需要采用的三种主要的测试度量达成一致意见。
- 具备一种度量/跟踪测试执行状态的能力，测试状态基于主要的测试状态模型（计划的、可用的、已执行的、通过的）。
- 确定一种度量程序复杂度的工具。
- 定义与测试相关的主要的发布标准并确定怎样实现度量。

12.5 参考文献

Beizer, B.（1990）.*Software Testing Techniques*. Van Nostrand Reinhold.

Capers Jones.（1991）.*Applied Software Measurement*. McGraw-Hill.

Hetzel, W.（1993）.*Making Software Measurement Work*. QED Publishing Group.

Humphrey, W.S.（1984）. *Managing the Software Process*. Reading, MA: Addison-Wesley.

Kit, E.（1986a）.*Testing C Compilers*, Computer Standards Conference.

Kit, E（1986b）.*State of the Art, C Compiler Testing*. Tandem Computers Technical Report.

IEEE/ANSI（1988a）.IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE Std 982.1-1988.

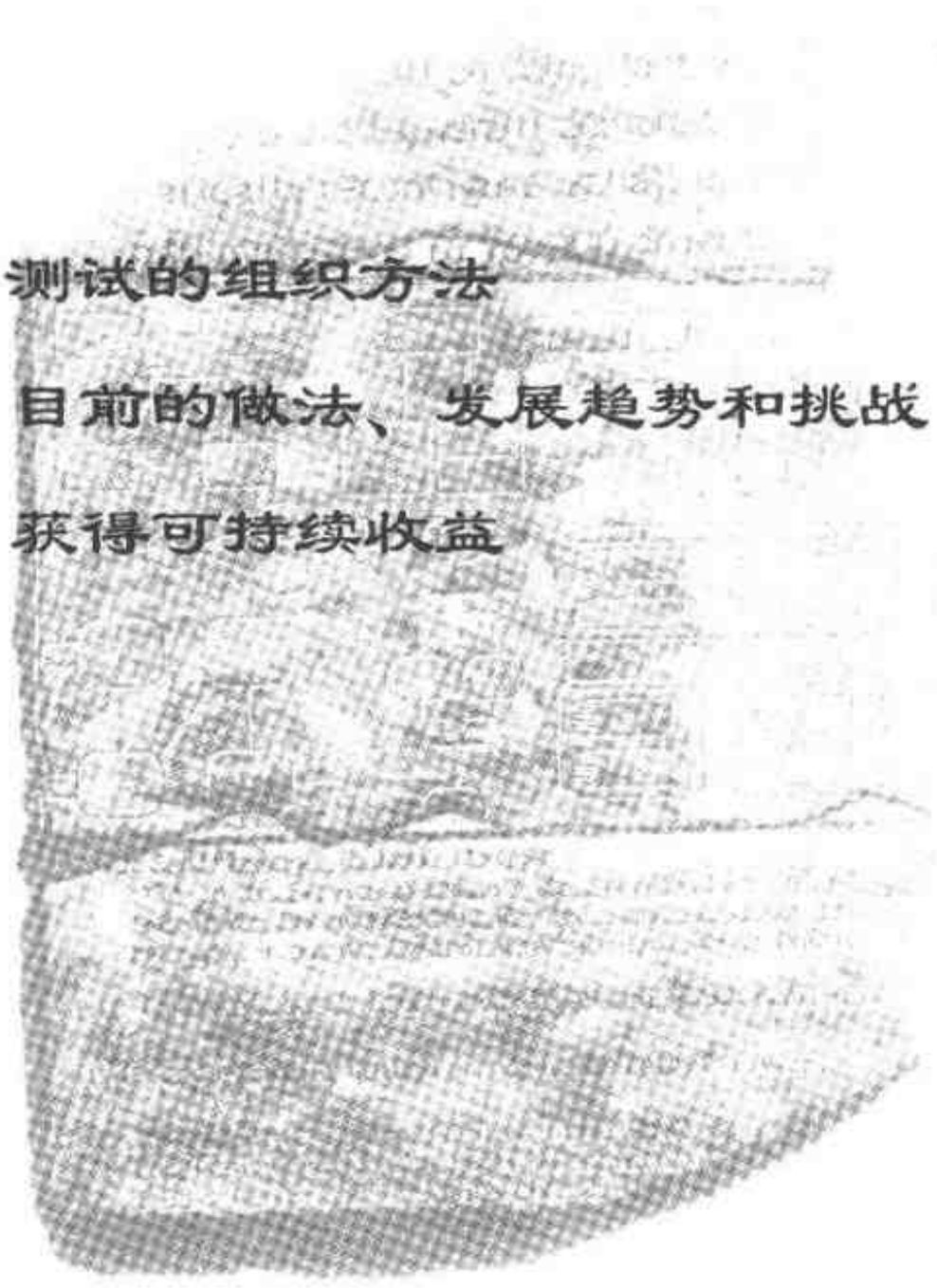
IEEE/ANSI（1988b）.IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE Std 982.2-1988.

第四部分

测试管理技术

“失去测试技术上的竞争就失去了软件质量上的竞争，这也意味着失去了软件领域的竞争。失去了软件领域的竞争即意味着失去了计算机市场乃至高技术市场的竞争，而这进一步意味着在整个后工业化的社会“战争”中的失利。”

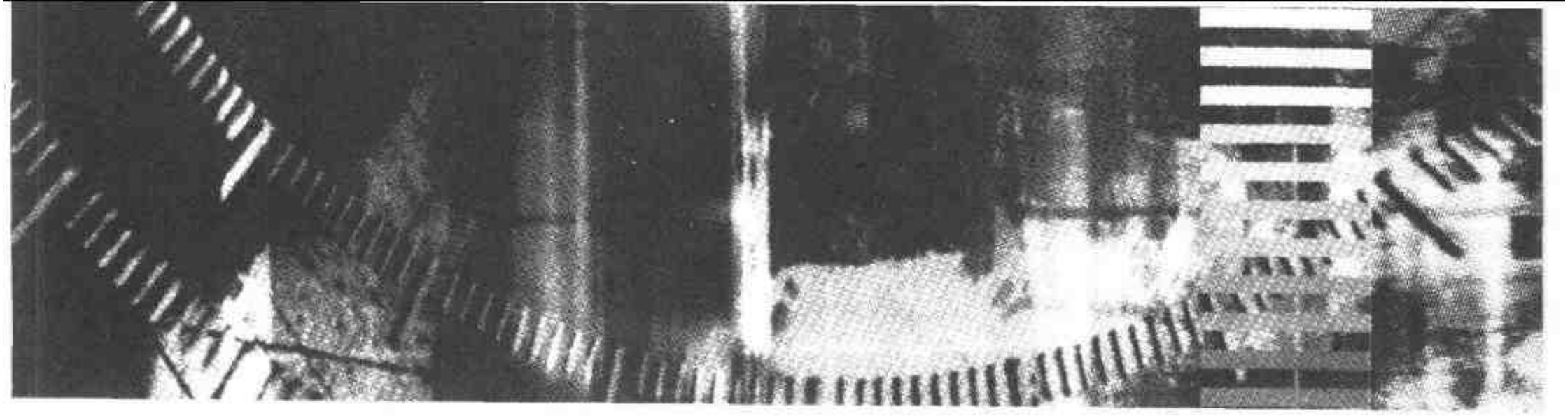
——Boris Beizer



第 13 章　测试的组织方法

第 14 章　目前的做法、发展趋势和挑战

第 15 章　获得可持续收益



第13章

测试的组织方法

大部分人都生活在不断变化的组织内。我们生活在没有边际的结构之中。软件测试尤其如此，各个公司纷纷尝试不同的组织方法——在一定程度上是因为结构是解决问题的最有效的办法，即可以最有效地协调好人们之间的相互关系。

组织为什么会存在？结构设计中最重要、最基本的构件是什么？实践当中组织软件测试的具体方法有哪些优点和缺点？如何决定现阶段在我们公司采用哪种办法？

组织的存在是因为人们的群体行为应该是可以预测而且是可靠的。为了达到目的，人们有必要相互合作。而组织的产生就是为了支持，或更确切地说，是为了控制人们从事的活动。好的组织可以减少组织与个人的需求之间的不可避免的相互冲突。

例如，有些软件组织已经成功地营造出了一种积极的文化氛围，鼓励并奖励产品开发人员到独立的测试开发小组去工作，反之亦然。当然，最好的文化氛围是鼓励那些随着时间的推移完全轮换一遍的人——开始几年做开发，接着做几年测试，然后再回到开发。

我们认识几位比较特殊的人，在 10 年左右的时间内重复了两轮工作岗位。令人吃惊的是，公司的经理们将这些人视为产品开发和测试开发方面非常宝贵的资源。还有必要谈职业保障吗？在开发方面，有比知道怎样设计测试库的人更优秀的人才吗？这样的开发人员知道怎样设计可以通过测试的软件，因为他可以从测试人员的角度进行思考。而且，掌握了开发人员的思维方式的测试设计人员更优秀？

但不幸的是有不少管理人员只是将测试视为一种培训，为开发服务，用不着安排最好的开发人员。这是非常糟糕的。我将自己列为这样的一类管理人员：不会聘请某个人去从事测试工作，除非这个人能胜任开发小组的工作。优秀的测试管理人员会鼓励这样的一个人获得几年的产品开发经验，然后再调到测试小组去工作。太偏激了吗？我们却认为一点都不偏激！

13.1 测试的组织和改组

一旦我们亲身经历过几次改组，我们就会理解对改组进行仔细计划的必要性。我们必须面对这一事实——改组会带来不同程度的压力、干扰、沮丧和混乱。下面的一段话被人们引用了

2200 年，从中我们可以看出，多少年来人们是如何认识并表达改组问题的：

> “我们严格训练……但似乎每次我们开始形成团队时就要改组……后来我渐渐地意识到我们通过改组来适应新的形势，要营造一种进步的假象，这是一种好方法，而与此同时，又在制造混乱、低效和沮丧。
>
> — Petronius Arbiter，公元前 210 年

时至今日，Petronius 的思想还如此贴切，真是不可思议。

组织是一个系统，将材料、知识和方法组合起来，把各种不同的输入转换成有价值的输出。组织结构是以一定的模式对责任、权威和关系进行的安排，组织通过这种结构发挥功能。结构形式包括报告等级制度、职业描述和目标。

测试组织（通常叫做测试小组）是一种资源或一系列的资源，专门从事测试活动。随着公司的扩大，必须要有专门、独立的测试功能。只有不持偏见的人才能提供不持偏见的度量——测试在度量软件质量方面真正有效果，就必须独立进行。

组织结构像软件。两者都没有真正完全结束的时候。随着时间的推移，对结构的要求（外部的和内部的）在改变——现有结构满足这些要求的能力下降——使得组织的再设计成为一种不断重复的活动。同样，随着时间的推移，对软件系统的要求也在改变，与组织的情况一样，不管开始时设计得多么完美，如果最终想继续留用的话，软件就必须不断升级。我们必须抛弃一种幼稚的幻想，即一旦我们理解了软件系统要解决的问题，我们就可以安下心来独自设计并测试软件。

测试管理是很困难的。测试小组的管理人员必须具备：

- 理解并评价软件测试过程、标准、政策、工具、培训和度量的能力；
- 维护一个测试组织的能力，该组织必须坚强有力、独立自主、办事正规且没有偏见；
- 招收并留住杰出的测试专业人才的能力；
- 领导、交流、支持及控制的能力；
- 关照测试小组的时间。

高级管理人员还必须具备招收并留住杰出的测试管理人员的能力。事实上，这也正是容易出大错的地方。高级管理人员没有充分理解上述要求，不知道如何根据这些要求评价潜在的测试管理人员。其结果是往往将不合适的人员晋升到了测试管理层！

如果改组会影响测试，高级管理人员应该了解这种改组对测试管理人员满足上述要求的能力所带来的影响。进行改组时如果没有充分考虑对测试的影响，就会出许多乱子。错误的软件测试结构可能带来的危险包括：

- 测试的独立性、正规性和思想倾向会被削弱或消除。
- 测试人员不参与有回报的项目。
- 测试工作人员不够。
- 测试人员结构不合理，初级人员太多。
- 测试的管理人员级别太低。
- 没有测试知识、培训、工具和过程方面的优势。

- 测试没有能力阻止发送质量低下的产品。
- 缺乏集中、持续的改进过程。
- 管理人员缺乏管理测试小组的能力。
- 不强调质量问题。
- 测试管理人员由于没有获得事业上的进步而沮丧。

制订改组计划时，上述各项可以用做需要考虑问题的清单。也可以将它们转换为提问，例如：这一变换会削弱测试的独立性吗？会影响测试工作获得高质量资源的能力吗？等等。

13.2 结构设计因素

结构设计因素出奇的少，他们包括：

（1）高耸还是平缓——在首席行政长官和搬运工人之间设立许多层次（这是一种高耸的组织结构），或设立很少几个层次（平缓的组织）。在过去的十年之中，平缓的组织更受欢迎，处在一个高耸的组织之中，管理人员觉得特别容易丢工作。

（2）市场还是产品——组织的结构设置可以面向不同的市场或不同的产品。

（3）集中还是分散——组织可以集中，也可以分散。对于测试组织来说，这是一个关键问题。本章后面，我们将深入探讨这一问题。

（4）分级还是分散——可以将组织分级，即按权力和级别对组织一层一层地分级。也可以分散，即扩散或排列开来。

（5）专业人员还是工作人员——组织应该拥有一定比例的专业人员和工作人员。

（6）功能还是项目——组织可以面向功能或项目。

将这些设计因素组合起来，就可以获得多种实用的结构设计。有时在同一家公司内实施各种设计方案。

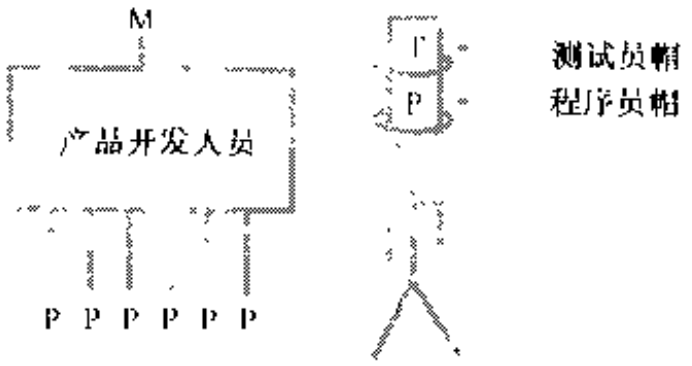
13.3 测试功能的组织方法

将上述组织方面的基本设计因素组合起来，可以构成许多不同的测试结构。实际应用当中，有七种典型的测试组织方案，他们反映出一个成熟的开发组织的进化历程。下列结构方案假设单元测试应该由产品开发进行。因此，下面的材料谈到的测试活动与单元测试无关，如：功能测试、系统测试等。

13.3.1 方案 1：测试是各人的责任

方案 1 是实践当中常用的方法，对测试考虑较少的小公司来说尤其如此。如图 13-1 所示，有一群产品开发人员，其主要责任是开发产品。在该模式中，这些产品开发人员同时也负责测试各自的代码。这些人很不幸，他们必须尽力而为，头戴两顶帽子：程序员帽和测试员帽。他们负责功能测试、系统测试以及其他各种测试。

该方案的问题在于它违反了一个基本假设：测试必须独立进行，这样才能真正有效地对软件质量进行度量。程序员是偏向于他们自己开发的产品的，因而对各自的错误视而不见，这是人的本性。

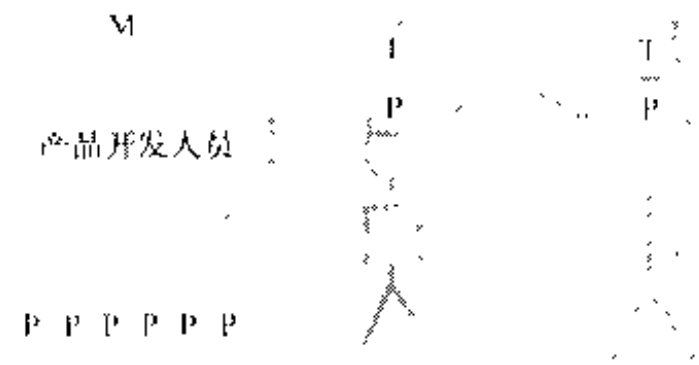


优点	缺点
表面看来是最自然的解决方案(从原有产品开发小组自然演变过来)。	从本质上讲，程序员不可能对自己的程序进行有效测试。

图13-1　方案1：测试是各人的责任

13.3.2　方案 2：测试是各小组的责任

方案 2 克服了方案 1 中明显的不足：即从本质上讲，程序员是不能对自己的程序进行测试的。克服这一不足的办法是在小组内部让产品开发人员相互测试代码。从图 13-2 可以看出，每个人仍然戴两顶帽子：右边的人负责开发他们自己的模块，同时必须测试其同事的模块。



优点	缺点
解决了程序员不能测试自己代码的问题。	程序员能力(多余的一顶帽子)。 方法、技术、培训。

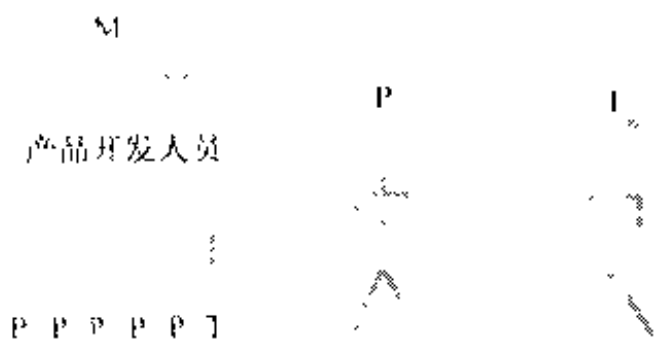
图13-2　方案2：测试是各小组的责任

现在的问题是这些人必须花时间了解软件测试专业人员的工作，并掌握产品开发的过程、标准、政策、工具、培训和度量等。对于典型的软件工业项目，这样要求实在是太高了。这有点象建筑工地上的一位普通工人，要求他在同一工地上既是电工师又是木工师。当然这也不是不可能，也可以做到——只是可能性不大，也没多大意义。

实际上，这些人只会戴一顶帽子——那顶他们主要负责且管理层对他们进行评价的帽子——即产品开发人员帽。只要时间和技术上允许，他们会花时间测试别人的代码。但在负责产品开发的同时，他们一般不会很专心去学测试方面的专门技术、工具和方法。现实如此。

13.3.3 方案 3：用专用资源进行测试

方案 3 选择产品开发人员并安排他们做新工作，即测试开发，从而解决了开发人员能力不够的问题。从图 13-3 可以看出，小组中的每个人现在只需要戴一顶专业帽子。棘手的事情是小组的管理人员必须选择合适的人员从事测试工作。



优点	缺点
解决了开发人员的能力问题（现在只戴一顶专业帽子了）。 一个小组。	管理问题。 管理层能提供测试过程、标准、政策、工具，培训和度量吗？

图13-3 用专用资源进行测试

在初期，我们注意到根本就没有测试组织。某一天，管理人员一觉醒来说："好，现在我明白了，我们得有专人从事测试工作。让谁去做测试人员比较合适呢？"这种情况的确发生过。以前作者在讲授一门测试课时，一位开发经理举起手来说："好，现在我知道测试是怎么一回事了。我要安排我们的第一位全职独立测试人员到位！"妙极了，教师对他的教学能力暗自高兴。

"是的"，该经理继续说，"反正这个人做开发也不合适。他们一直表现不佳，自己感觉也不好，他们对项目没做多少贡献，我得把他们解雇，但这事的确有些难办，就让他们去做测试吧！这不就两全其美了吗？"

当然，他这样做损失不会小。现在头戴多顶帽子的人是一线的产品开发经理。除指导产品开发小组外，这个人还必须就测试过程、测试标准、测试政策、测试工具、测试培训、测试度量、专业测试人员的聘用等方面提供指导。这样一来，是办不成什么事情的。

很明显，必须成立测试小组——专门从事测试活动的一套资源，由一位测试经理管理。没有正式的组织，就必须为每一项目建立一套测试过程和工具。建立了单独的测试小组之后，该组织就可以连续为所有的项目服务——为管理层提供独立、不带偏见的高质量的信息。

Bill Hetzel 在其《软件测试指南大全》（1988）一书中写道：

独立的测试组织十分重要，因为

- 没有这样一个组织，建立的系统就不会理想，
- 有效的度量对于产品质量控制是十分重要的，
- 测试协调需要全职、专门的人员投入。

建立正规的测试组织，可以解决方案 3 的问题。从图 13-4 可以看出，测试组织的重要性得以实现，组织由测试经理领导。现在的问题是，在组织上将测试小组放在什么位置？

图13-4 测试组织

13.3.4 方案 4：将测试组织置于 QA 中

一般的做法是将新建立的测试组织作为质量保证的一部分，在此，QA 还对开发过程进行审计（见图 13-5）。QA 经理也许不了解软件测试。测试小组的管理能力非常重要。测试是由开发以外独立的组织进行的，所以必须投入精力鼓励并营造一种积极的小组氛围。人们开始发出疑问，谁管质量？开发管理层抱怨，他们不再拥有生产高质量的产品所需的整套资源。

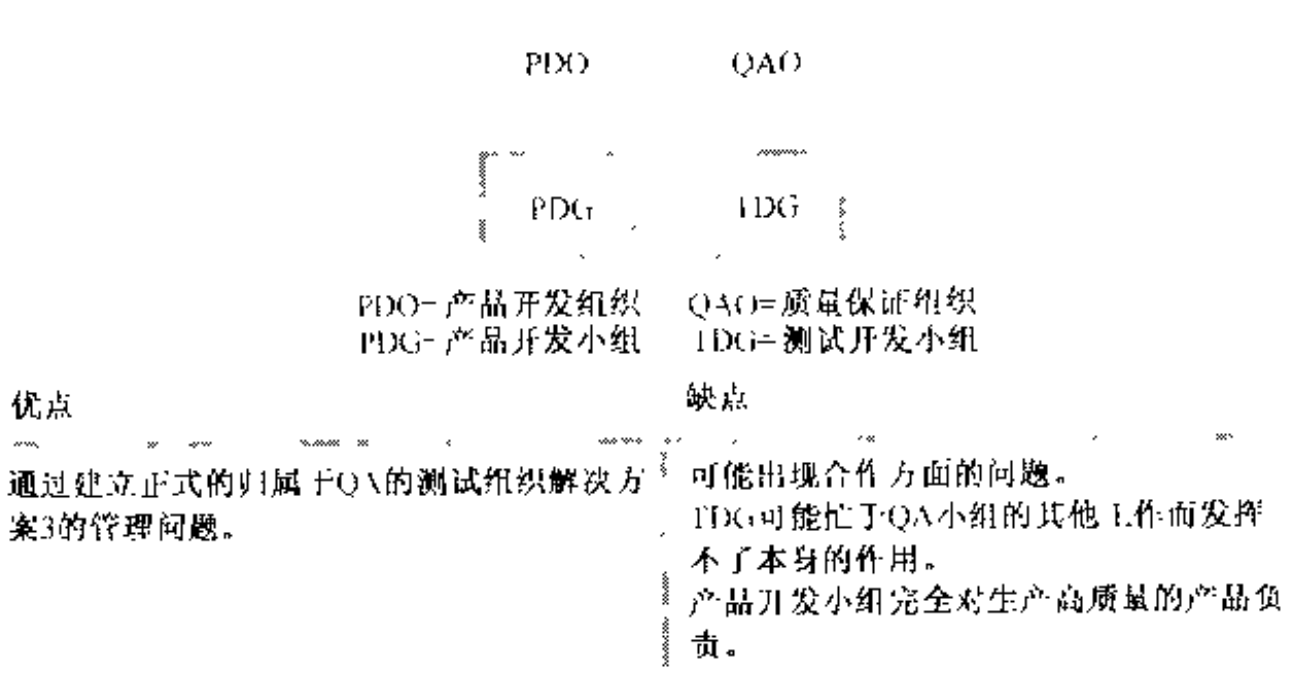


图13-5 方案4：QA中的测试组织

13.3.5 方案 5：隶属于开发的测试组织

通过将测试组织归属于开发组织（见图 13-6），方案 5 主要解决方案 4 中的小组合作和质量归属问题。根据这一方案，测试小组归二线产品开发经理领导。

不理想的是，对大多数二线产品开发经理要求太高了。如果副总裁这样的高级管理人员能理解高效、独立的测试功能的重要性，并愿意将能干的管理人员安排在测试小组经理的岗位上，

问题就简单得多。无论如何，一切都指望高级经理了，他现在是头戴两顶帽子：管理负责产品开发的管理人员以及管理负责软件测试的管理人员。

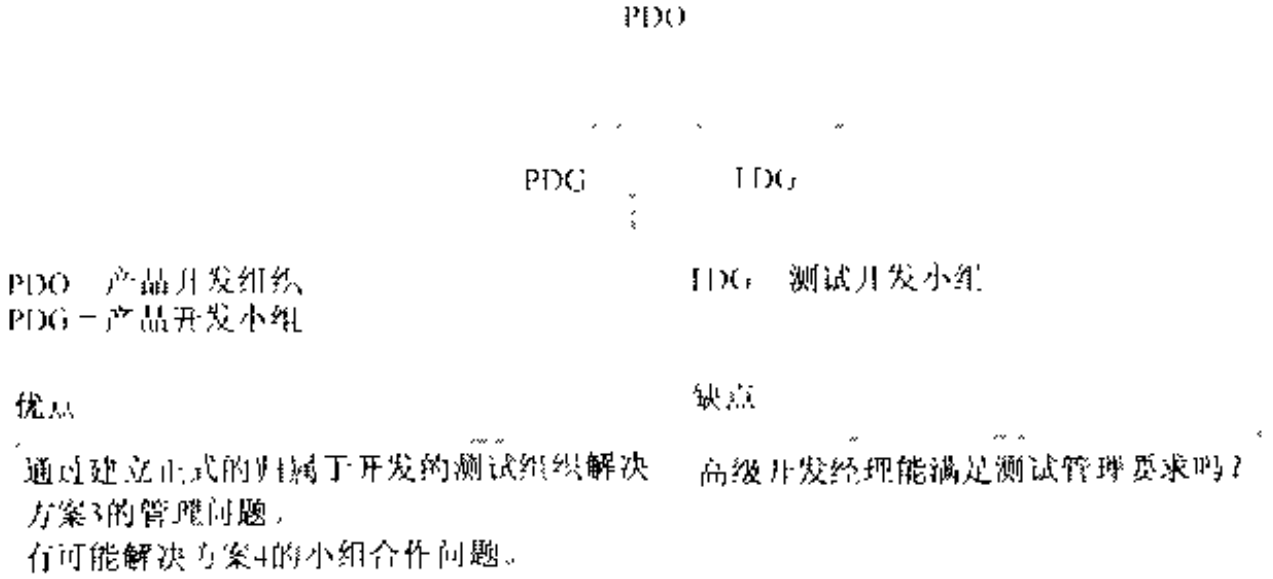


图13-6 方案5：隶属开发的测试组织

随着组织的扩大，测试组织聘用的人也越来越多，也就需要多个测试小组了。出现了新的问题，对于多个测试小组是集中管理还是分散管理？

13.3.6 方案6：集中管理的测试组织

方案6通过建立中心测试组织解决方案5中的高层管理问题，中心测试组织归属并服务于产品开发部门（见图13-7）。可以看出，这为高级测试经理/主任提供了难得的机会，可以对组织施加重大的影响。例如，高级测试经理可以：

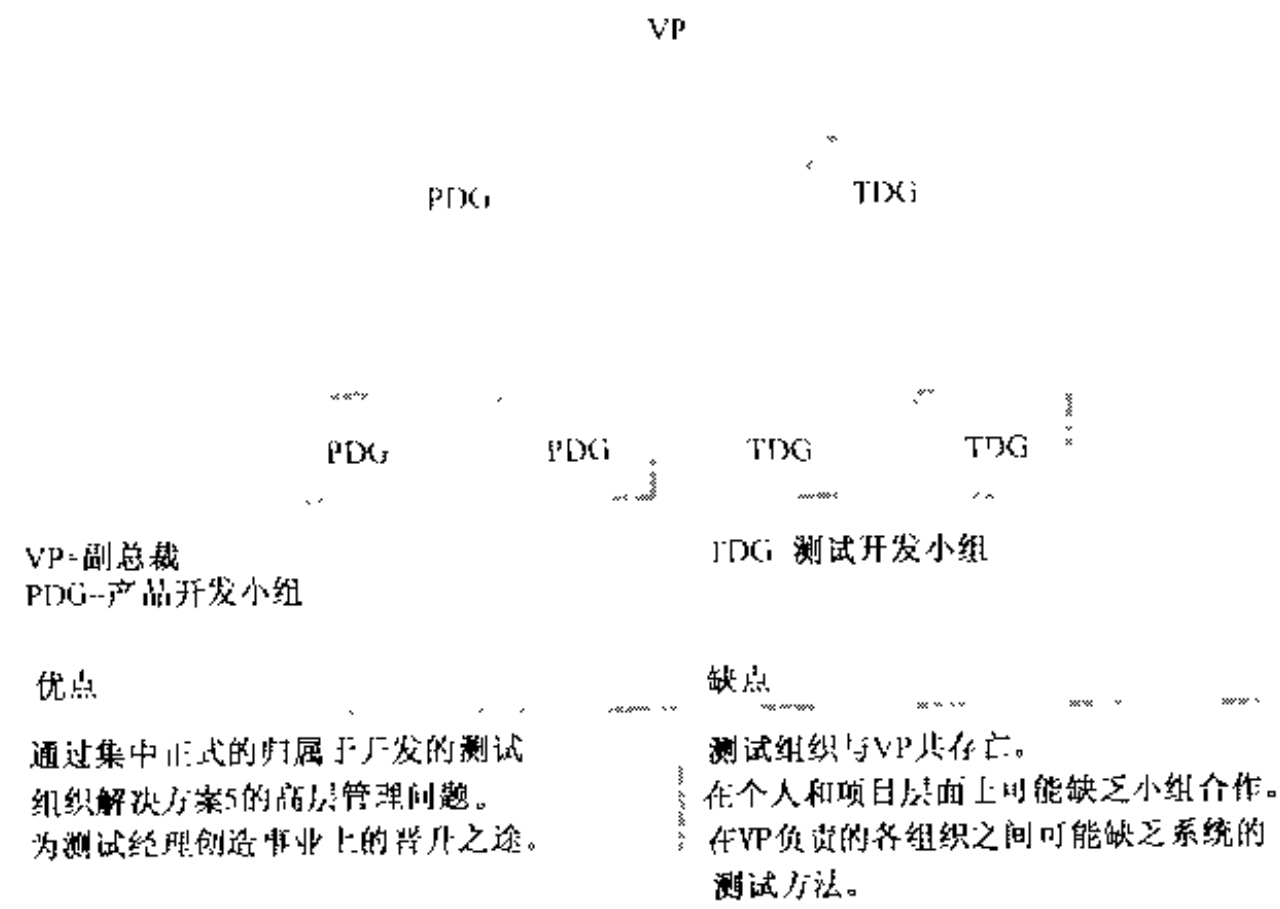


图13-7 方案6：集中管理的测试组织

- 管理共享的测试资源（包括人和设备）以减缓需求并更好地管理公司的风险；
- 协调各测试小组所有测试人员的系统培训；
- 向所有测试人员推广使用高质量的测试工具；
- 物色并聘用优秀的一线测试经理；
- 为一线测试经理提供真正的指导。

此外，一线测试经理在事业上也有盼头，可以通过努力成为高级测试经理。

现在风险集中起来了，如果我们假设高级测试经理很优秀，那么测试组织将与领导产品开发和测试开发组织的副总裁共存亡。当涉及到软件测试的人头、资金和奖励时，副总裁必须提供必要的支持。

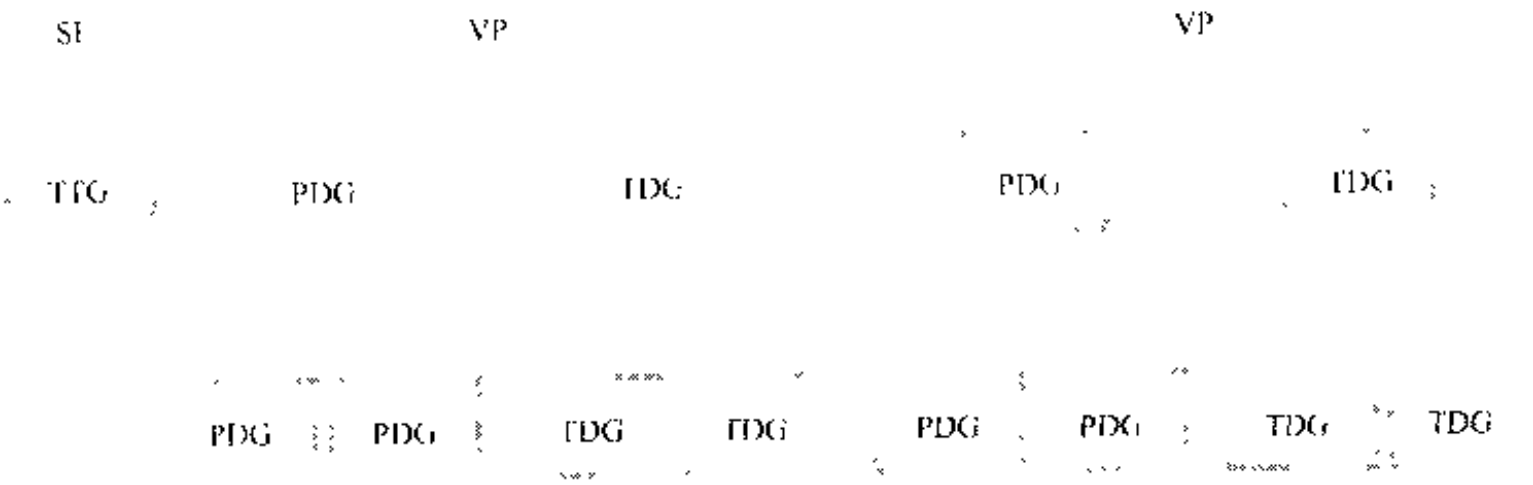
只要副总裁的思路正确，实践当中这一切可以而且往往确实也是运行正常。如果副总裁给非常胜任工作的高级测试经理提供合理的资源、用人得当且授权合理，这一方案会非常出色。

成立课题组，由独立的测试组织提供资源，与产品开发人员并肩作战，一块行动、相互交流、一起生活，这样一来就可以减少原来提到的小组合作时的不利因素。如果需要就产品质量问题进行认真的讨论，高级测试经理可以实事求是地向副总裁汇报。这对于副总裁来说是非常重要的，因为他无法从产品开发管理人员那儿获得全部情况。

随着公司的扩大，包含多个部门，采用这一方案的公司会开始觉得在副总裁负责的各组织之间缺乏系统方案和最优方法。这些问题可以通过下一个也是最后一个方案解决。

13.3.7 方案7：设有测试技术中心的集中式测试组织

通过建立作为软件工程功能一部分的测试技术小组，方案7解决协调性问题（见图13-8）。新的测试技术小组负责：



优点	缺点
将作为软件工程一部分的测试技术小组集中起来，解决方案6的协调性问题。	测试组织与VP共存亡。 在个人/项目层上可能缺乏合作。

图13-8 方案7：集中管理和测试技术

- 领导并管理测试过程，测试生产力改进工作；
- 促进并协调测试培训项目；
- 协调测试工具项目的计划和实施；
- 根据需要编制测试过程、标准、政策和指南文件；
- 发现并采用测试小组内的最优方法；
- 推荐、确定并实施主要的测试度量。

13.4 选择正确的方案

从采用各种结构设计因素的不同方案中引进哪种方案呢？作为起步，下面我们提供了一套示范选择标准，可作为评估不同方案的基础。在多大程度上该组织结构：

- 提供快速决断能力；
- 提倡合作，尤其是产品开发和测试开发之间的合作；
- 支持一个独立、正规、不带偏见、精干、人员配置合理并有成就的测试组织；
- 帮助协调测试与质量责任之间的平衡；
- 协助在本章前面提到的测试管理要求；
- 为测试技术提供所有权；
- 可利用现有资源，特别是人；
- 对职工（包括经理）的道德和事业产生积极影响？

快速决断可以加快反应速度。对于不能快速作出决断的组织结构，可以成立特别核心小组并/或定义特别的快速决断过程，即由高层管理人员每日召开短会，集中处理项目的有关问题并做出决定。

改组时可以按以下步骤进行：

（1）详细描述现有组织；

（2）定义并挑选主要的选择标准（见上面各条）；

（3）用文件记录新出现的其他方案；

（4）采用选择网格（见下文）对候选方案进行评估；

（5）确定新组织；

（6）实施新组织。

决定矩阵选择网格是一种质量工具，在采用选择标准的同时可用于评估各种方案。选择标准一旦确定，便可以将它们编号或进行标记，然后填入决定矩阵。

对候选方案也进行编号并垂直填入矩阵，如图 13-9 所示。评估每一方案时，对照每一选择标准进行赋值，然后填入矩阵。该数值选自事先定义的数值范围，如 1 至 5，5 表示高等级，意思是该组织在很大程度上满足被评方案的现有标准。同样，3 表示中等，而 1 则代表低等。

将得分水平相加，得每一方案的总分。也可以将该方法改进，对选择标准赋予权重。

改组应仔细计划实施。最后一条建议是：在上述过程中，记住让参与者加入进来。

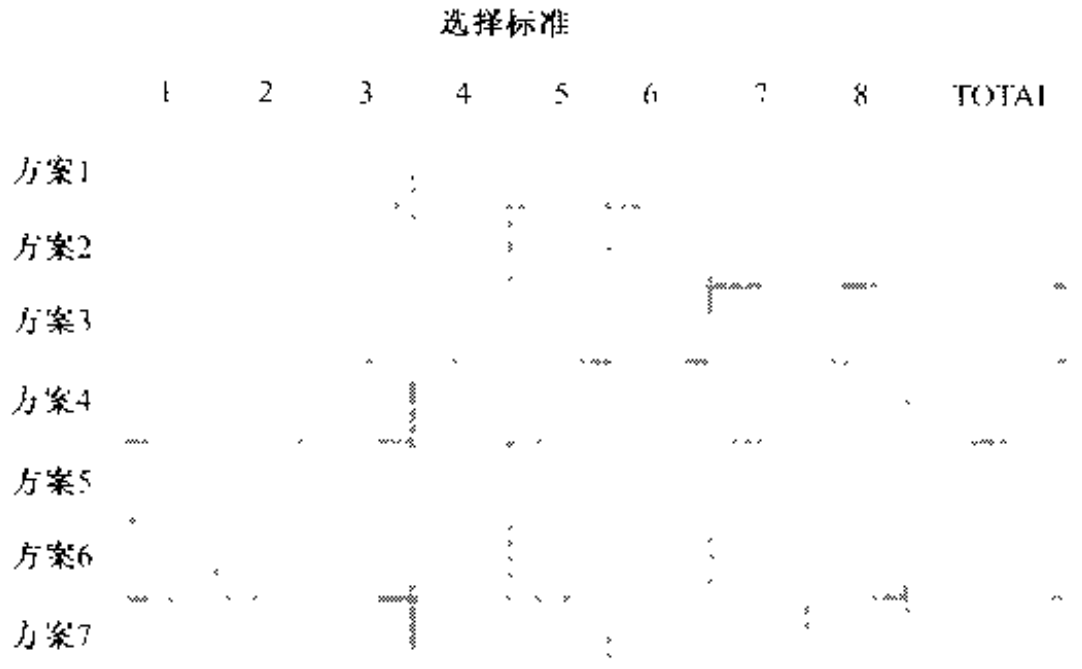


图13-9 决定矩阵选择网格

13.5 参考文献

Beizer, B. (1992). "Losing It, An Essay on World Class Competition," *Software Quality World*, 4(3).

Goodman, Paul, Sproull & Associates (1990). *Technology and Organizations*. San Francisco, CA: Jossey-Bass.

Gelperin, D. And Hetzel, W. (1989). *STEP: Introduction and Summary Guide*. Jacksonville, FL: Software Quality Engineering.

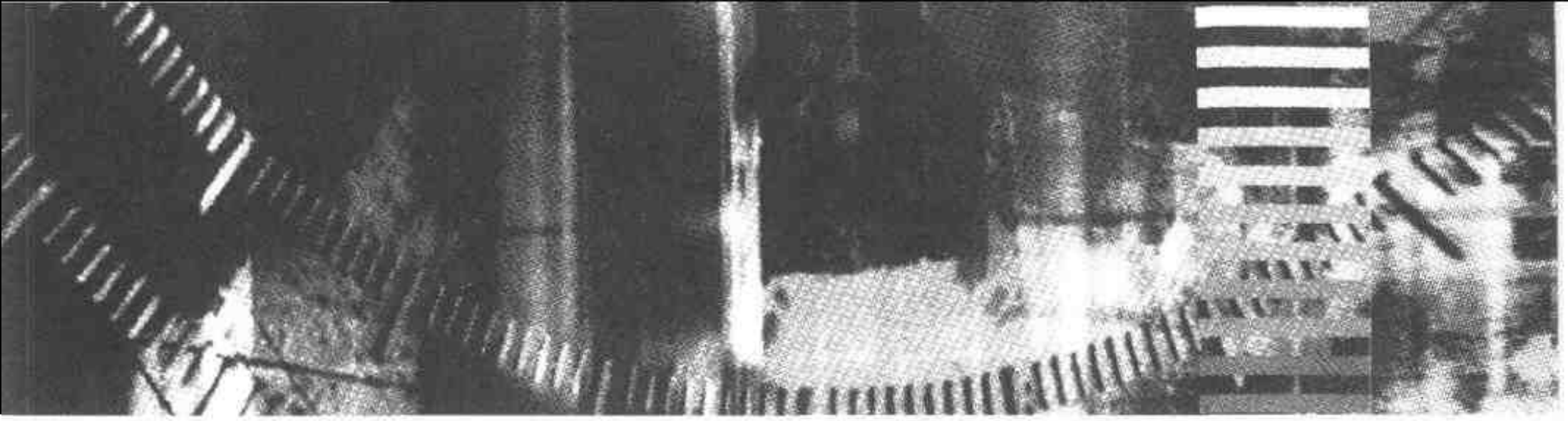
Hetzel, W. (1988). *The Complete Guide to Software Process*. Wellesley, MA: QED Information Sciences.

Humphrey. W.S. (1989). *Managing the Software Process*. Reading, MA: Addison-Wesley.

Kaner. C. (1988). *Testing Computer Software*. Blue Ridge Summit, PA: TAB Books.

Kit, E. (1992). "Approaches to Organizing the Independent Test Function," *Software Testing Analysis & Review(STAR) Conference Proceedings*.

Silverman, M.(1984). *The Technical Manager's Survival Book*. New York: McGraw-Hill.



第14章

目前的做法、发展趋势和挑战

随着技术和开发工具的进步，出现了采用图形用户界面（GUI）的复杂软件以及客户机/服务器应用结构。尽管这些进步提高了开发生产力，但生产出来的软件很难测试，对测试的新要求降低了对开发生产力所带来的收益。商业测试工具的自动化在一定程度上弥补了这一不足。

回顾软件工程的短暂历史，测试人员与开发人员之比发生了巨大的变化（测试人员增多），这意味着现在人们认识到了测试的重要性，组织现在也愿意为合适的测试投资了。与此同时，测试人员的人数在增加，用于测试的时间和测试的效率也在增多和提高。

同时，对该行业最优做法的研究表明，严格意义上的非技术因素——良好的管理、通信、人员和控制——与最先进的方法、CASE 工具和其他灵丹妙药相比对成功的测试更重要。

14.1 图形用户界面：有哪些新东西

有两种用户界面：

（1）基于文字的用户界面（传统型）；

（2）图形用户界面。

图形用户界面的特点是采用了一种新的输入装置（鼠标）和高分辨率的立体彩色图像。图形用户界面测试要求的自动化程度越来越高，但这在图形用户界面环境下比基于文字的环境要复杂得多。幸运的是，新一代捕获/回放工具提供了解决问题的办法（参考第 11 章）。

14.2 应用测试

掌握产品的真正用途可使我们集中精力测试几种关键域。我们选择确认测试时，事实上已经在潜意识地对潜在的应用作出判断。应用测试是解决这一问题更为正规的方法。

应用测试是一个过程，包括：

- 基于对使用模式的估计的初始测试；
- 对实际应用（产品完成并发挥功能后）进行度量（采集数据），并开发运行剖面；
- 基于运行剖面，调整需要优先考虑的方面，开发新的测试，并进行再测试。

应用测试反映硬件和软件配置方面所预测的运行应用，包括测试的运行频率、测试的运行序列和系统负载。测试强调错误检测，即在运行应用当中最容易出现的错误。

应用测试最好在单元和集成测试之后进行，因为这时大量的功能已经到位，并已完成了需求确认——如果确认是单独进行的话。实践当中，应用测试一般在基于实际用户的实际应用开发出运行剖面后进行，这意味着应用测试只适用于第一次发布之后的后续产品发布。

应用测试的优点是面向客户：对测试资源和进度进行优化以极大地满足客户的要求。难办的是如何低成本高效率地获得运行剖面，这也是许多公司至今尚未正式进行应用测试的原因。但许多著名的软件公司在优选值得开发的测试时，的确在考虑已知或预计的产品应用。

应用测试是以前在 Institute for Zero Defect Software 供职的 Frank Ackerman(1992,1993)率先推出的。

14.3 测试人员与开发人员的比例

测试人员与开发人员的比例是一个很有趣的指标，反映出从事测试活动人数与从事软件产品开发人数之比。在软件工程学科短暂而又风云变幻的历史中，测试人员的比例在急剧增加。

历史上，对主机系统来说，测试人员与开发人员的比是 1:5-10，即每五到十个产品开发人员只有一个测试人员。最近公布的数字有：

- Microsoft，1992　　2:3
- Lotus(for 1-2-3 Windows)　　2:1
- 4 大公司平均（1992）　　1:2

（注：Microsoft 和 Lotus 参考 Marvin(1993)。Microsoft 的人员比例是对于 Microsoft 全球产品部而言。四大公司指的是 Microsoft、Borland、WordPerfect 和 Novell，参考 Norman（1993）。）

上述数字应该高于平均数，即比大多数公司实际聘用的测试人员要多。通过软件开发技术公司所授测试课程进行的非正式调查，表明大多数公司的测试人员与产品开发人员之比达不到一比二。比较典型的比例在 1:3 和 1:4 之间。被调查的大多数测试专业人员，如果其公司的比例在 1:5 或者更高(如 1:7、1:10、甚至 1:20)，一般都会感觉到测试的资源不够。

14.4 软件度量和实践基准研究

这是一项十分重要的研究，由 Xerox 公司和软件质量工程公司共同发起，以促进人们对软件高品质的了解。该项研究的目的是确定软件技术领域具有世界水平的项目，并对这些项目的工程、管理和度量实践特征进行描述。软件质量工程公司提供了一份完整的研究报告（1991）。

该项研究的设计包括两个阶段。第 1 阶段确定“最佳”项目。对七十五家软件组织进行了调查，要求他们对 52 个问题做出回答（15 个涉及组织，37 个涉及他们的实际做法）。第 2 阶段对“最佳”项目的特征进行描述。调查了十个“最佳”项目（7 家公司、509 人），要求他们对 104 个问题做出回答（17 个涉及态度、87 个涉及实践和度量），并对他们进行现场实践评估。

“最佳”项目的挑选基础是：

- 被认为可以生产高质量的产品
- 采用的做法更佳

- 最近实施或进行最终测试时可对项目小组进行访问

换言之，最佳做法是从主观判断具有最高质量的项目逆向得来的——这些项目是组织最引以为荣的。

第 2 阶段挑选公司的基础是：

- 软件是组织任务的重要组成部分
- 第 1 阶段调查得高分的（高于第 75 个百分位）
- 名气以及公众对公司成就的认知程度

第 2 阶段的公司包括：

- AT&T
- 杜邦
- GTE
- IBM
- NCR
- 西门子
- 施乐

该项研究的主要发现包括：

（1）最佳项目强调严格的计划，并从发展的角度对状态进行密切跟踪和报告；

（2）最佳项目依靠管理上的基本原则（合作、交流和控制），而不是技术和最先进的方法；

（3）最佳项目强调评议、审查以及严格独立高水平的测试；

（4）度量用于跟踪进度、质量和问题；

（5）七家公司都有计划、要求和设计规范；

（6）最佳项目采用了推荐做法——为获得更规范且更结构化的过程而进行的投入在结果上是显而易见的；

（7）最佳项目在不同的领域（调查方面）有所突出，并强调生存周期的不同阶段；没有哪个项目在所有领域都处于领先地位。

主要的度量工作包括：

- 调度性能
- 代码缺陷
- 测试结果
- 测试缺陷
- 发布后的缺陷
- 公开问题的数量
- 修改错误的时间
- 问题跟踪
- 代码行
- 过程兼容

没有灵丹妙药。成功的关键是抓基础：

- 良好的管理
- 相互交流
- 优秀的人才
- 严格控制
- 连续的度量

14.5 参考文献

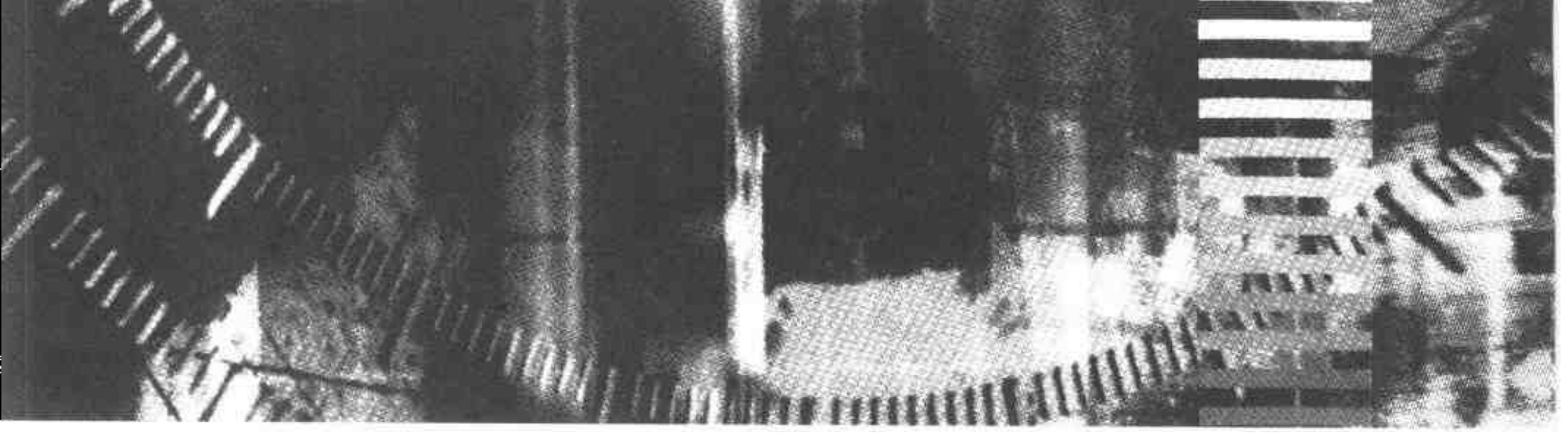
Ackerman, F. (1993). "Usage Testing," *Software Testing Analysis & Review (STAR) Conference Proceedings 1992.*

Ackerman, F. (1994). "Constructing and using Operational Profiles," *Software Testing Analysis & Review (STAR) Conference Proceedings 1993.*

Norman, S. (1993). "Testing GUIs is a sticky business," *Software Testing Analysis & Review (STAR) Conference Proceedings 1992.*

Software Quality Engineering (1991) Report 908.

Tener, M. (1993). "Testing in the GUI and Client/Server World," *IBM OS/2 Developer,* Winter.



第15章

获得可持续收益

只有坚持不懈地采用成功的软件工程实践，公司才能在当今的市场上竞争、生存并立于不败之地。必须进行必要的修改和投入以便改进现有技术和方法；否则就会满盘皆输。多年的咨询经验，加上对 Xerox 的研究（见第 14 章），清楚地表明如果我们强调计划、交流、跟踪和度量，就会极大地提高生产高质量产品的可能性。

15.1 实现收益

引起变化是很困难的事情。无论进行小改变还是大改变，是战术上的还是战略上的改进，对于那些涉及文化演变或革命的变化，需要花时间对情况进行评估并对可持续收益进行计划。你面对的是行人道上一条小小的裂缝还是一个很深的窟窿？对于不同的情况，采取的战略是不一样的。当面对一个很深的窟窿，采纳小步走以便实现连续改进的建议可能导致灾难性的后果！记住一句中国谚语：

“两步过鸿沟是危险的。”

改变一个组织是困难的。改变必须是积极的、有计划的并且可被控制，尽量减少干扰和文化上的冲突。判断是否已做好了迎接变化的准备：组织目前感觉有多艰难？目前客户感觉有多痛苦？对应于自然抵抗变化的程度，内部对现状不满意的程度有多高（Migdoll，1993）？

必须定义变化的好处，并说明要进行的改进。要向整个组织的管理人员和专业人员宣传变化的益处。对已实现的变化的效率进行度量并予以公布。人们应该知道他们参与（并为之付出代价）的东西是起作用的。

最后，管理人员必须起到指导作用。如果做法不当，管理层将自食其果：收到客户的投诉信，失去一家大客户，被起诉等。如果管理层能发现问题的所在并提出解决问题的办法，那么可持续改进的又一关键方案就到位了。在热情消失的困难时期，管理层的支持是很重要的。

15.2 获得帮助

早期的软件测试专家没有软件工程标准、测试工具、课程、书籍和研讨会可供利用。但现

在情况已发生了变化。

15.2.1 软件测试书籍和业务通讯

有时最好的起步是得到一本好书。如果你花得起钱买两本书，先买一本当代很经典的测试书：Glenford Myers（1979）著的《软件测试的艺术》。如果公司还另花得起 50 美元的话，就为公司办件好事，买一本 Watts Humphrey (1989) 著的《软件流程管理》，但别留着一个人用。把它交给管软件开发的人，因为他们是改进开发过程最合适的人，而且作为回报，对你的软件测试工作也十分有利。有关软件工程和软件测试方面的详细资料，参考附录 E。

供测试经理和专业人员参考的期刊和业务通讯包括在附录 F 中，这些刊物是测试界内实践和经验方面的宝贵信息资源。

15.2.2 咨询和培训服务

采用外部咨询有助于变化。对于有政治问题和压力的组织来说，独立的咨询专家可以提供一种中立、不带偏见且没有政治风险的观点。当人们对实际问题靠得太近时，咨询专家可以站在全行业的高度提出意见，包括从不同组织获得的经验。

通过培训，各个层次的人可以获得技术和概念方面的完整知识。组织承诺为提高职工的专业水平进行投资，并决心在行业内处于领先地位，是否组织培训是主要指标之一。

改进开发过程的培训和咨询服务可满足世界各地的客户要求。通常，这些服务是直接为软件产品开发人员、软件测试开发人员和管理人员提供的。挑选培训和咨询服务提供商时，可采用下列评估项目：

- 他们有经过验证的开发和测试经验吗？
- 他们可以提供经过验证的且实用有效的解决方案吗？
- 他们可以进行技术评估，以便审查现有的软件开发和测试方法吗？
- 他们如何确定并挑选最需要改进的领域？
- 他们提供整套的验证和确认服务吗？
- 他们具备一支稳定有经验的专家队伍为你提供定制设计吗？
- 他们有技术权威，即具备技术、交流和指导才能的有经验的高级软件工程师和管理人员吗？
- 他们具备对你的软件类型适用的软件技术专家做顾问吗？
- 他们是永久性的技术权威、作者、研讨会发言人和国家软件技术标准制定方面的权威吗？
- 他们有感到满意的广泛的客户群体吗？

要了解为你提供服务的专家或顾问的资历情况，包括从业的年头。（参考附录 G。）

15.2.3 软件测试研讨会

软件测试研讨会的与会人员包括那些负责测试和评估活动的人，他们是专业人员、管理人员、质量保证专家、系统分析人员、项目管理人员和程序员。公司往往选送他们的高级人士参加研讨会，因此研讨会是难得的交流场所。

软件测试研讨会一般包括：

（1）会前辅导班，专家们集中讨论几个议题，时间在半天至两天之间，视研讨会和议题而定。典型的议题包括：软件测试概况、软件审查、面向目标的系统测试、客户机/服务器测试、GUI 测试。

（2）全体会议，专家和工业领袖们就技术问题和行业趋势进行探讨，向与会人员提供建议。

（3）测试工具展示会，一般占一或二天时间，有 20 到 30 家销售商参展，展示软件测试方面的产品和服务。大部分销售商都将展示他们的最新软件测试工具。

（4）分组讲座，多个分组会同时进行，与会人员选择自己感兴趣的专业讲座，如管理、工具、方法、过程改进等。

会费各不相同，通常在 300 至 1500 美元之间，视研讨会本身以及是否包括会前辅导班而定。与会人数也不一样，通常在 200 至 600 人之间。

至于各测试研讨会的具体信息，参考附录 F。

15.3 后续工作

实施改进是一种“永远是以后再说”的承诺。不要害怕制定长期的计划，但同时也要追求小规模、稳健而且可感觉到的短期收益，这样日积月累就会出现大的改观。遇到一筹莫展的情况，可倒回去参考本书的基础部分——并可参考经过验证的工程学基本原理。

必须有人对整个改进过程负责，对所有的改进过程进行计划和管理，定期与其他改进过程负责人联系并检查工作情况。如果公司内部没有合适的人选，可找一位优秀的咨询专家担当此任，以确保工作不断取得进展。记住，如果一段时间停滞不前，则意味着相对于别人你在后退，因为他们在不断地改进。

怎样知道我们取得了进展？通过度量找答案。例如，可以度量文件的可获得性这类较简单的东西。如果能够在可度量、可重复的基础上作出比较，则意味着你已经走上了正确的过程控制和成熟之路。

真正的进展等于可持续的收益——这种收益不会随着人员的变化、开发项目和客户类型的不同以及出现不可避免的改组而出现波动。收益会增加 5%或少一点，但改进尽管起步较难却是可以叠加的，每进一步会使下一步变得更容易。

15.4 参考文献

Humphrey, W.S.(1989). *Managing the Software Process*. Reading, MA: Addison-Wesley.

Migdoll, M.(1993). “Improving Test Practices,” *Software Testing Analysis & Review (STAR) Conference Proceedings*.

Myers, G.(1979). *The Art of Software Testing*. John Wiley.

第五部分

附　录

附录 A　软件工程和测试标准

附录 B　验证审查单

附录 C　验证练习

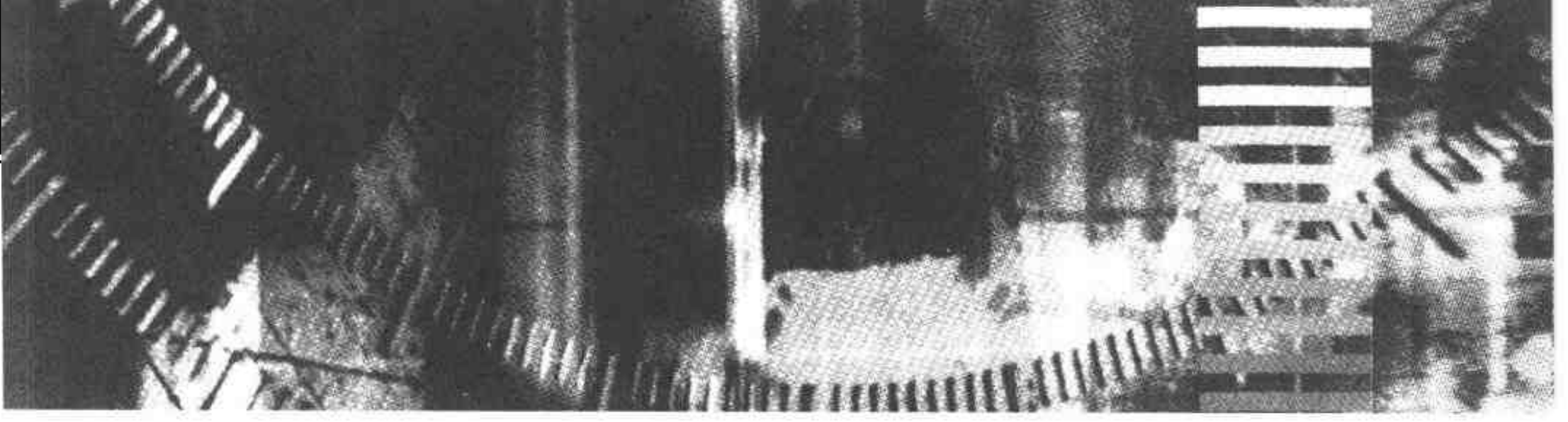
附录 D　确认练习（答案）

附录 E　参考书目（包括软件测试工具一览表）

附录 F　信息资源：会议、期刊、通讯、DOD 规范

附录 G　专用工具和工具选择

附录 H　改进实施示范清单



附录A

软件工程和测试标准

主要软件测试标准

IEEE/ANSI Standard 829-1983

IEEE Standard for Software Test Documentation (Reaff.1991)

该标准定义了八份覆盖整个测试过程文件的内容和格式。测试计划规定了测试活动的范围、方法、资源和进度。定义了测试项目、测试任务、负责各任务的人员以及与计划有关的风险。该标准显示了这些文件制定时的相互关系以及它们与测试过程的关系。

IEEE/ANSI Standard 1008-1987

IEEE Standard for Software Unit Testing (Reaff.1993)

软件单元测试是一个过程，包括测试计划的制定、测试套件的开发以及按要求对测试单元进行度量。度量意味着将样本数据施加到单元上，并将单元的实际性能与单元需求文件中规定的要求性能进行比较。

该标准定义了一种系统且文档化的单元测试集成方法。该方法采用了单元设计、单元实施信息以及单元需求，以便确定测试的完整性。该标准描述了一个由阶段、活动和任务等层级组成的测试过程。此外，它还定义了每一活动的最起码的任务数量，当然每一活动的任务是可以增加的。

其他与软件测试有关的标准

IEEE/ANSI Standard 1012-1986

IEEE Standard for Software Verification and Validation Plans (Reaff.1992)

该标准有三方面的用途：

（1）为关键和非关键软件的软件验证和确认计划（SVVP），提供有关格式和内容方面的统一和基本的要求。

（2）为关键软件定义最基本的验证和确认（V&V）任务，以及按要求必须包括在 SVVP 之内的输入和输出。

（3）推荐任选的 V&V 任务，用于就特定的 V&V 工作对 SVVP 进行合理的修改。

IEEE/ANSI Standard 1028-1988

IEEE Standard for Software Reviews and Audits

软件评审和审计是软件开发周期中软件产品评估的基本组成部分。该标准对评审或审计人员的评估行为提供指南。包括对关键和非关键软件都适用的过程，以及对评审和审计的执行所要求的步骤。

IEEE/ANSI Standard 730-1989

IEEE Standard for Software Quality Assurance Plans

该标准将法律责任作为理论基础，面向关键软件的开发和维护，在这方面如果出现故障会影响安全或引起巨大的经济或社会损失。主要目标是描述特定项目所有的计划和系统行为，以便使人确信该软件产品符合现有的技术要求。

该标准为软件质量保证计划规定了格式和基本内容。

IEEE/ANSI Standard 610.12-1990

Standard Glossary of Software Engineering Terminology

该标准是 IEEE Standard 729 的修改和重新规定。该标准包含 1 000 多条术语的定义，构成了软件工程的基本词汇。它以美国国家标准学会（ANSI）和国际标准化组织（ISO）的术语为基础，提倡软件工程和相关领域词汇的准确性和一致性。

其他的软件工程标准

IEEE/ANSI Standard 828-1990

IEEE Standard for Software Configuration Management Plans

该标准在格式上类似于 IEEE Standard 730，但限于处理软件配置管理问题。该标准确定了配置标识、配置控制、配置状态注册和报告以及配置审计和评论方面的要求。这些要求的实施提供了一种记录、交流并控制软件产品项目演变的手段。它为软件产品项目在其开发和维护生存周期的进化过程中的完整性和连贯性提供了保证。

IEEE/ANSI Standard 830-1993

IEEE Recommended Practice for Software Requirements Specifications

该指南描述了软件需求规格说明方面的一些优秀方案。为使读者作出正确选择，该标准还提供了大量的辅导材料，该指南包括了优秀软件需求规格说明的特征，以及规格说明的方法和相关格式。

IEEE/ANSI Standard 982.1-1988

IEEE Standard Dictionary of Measures to Produce Reliable Software

该标准为所选度量提供了定义。这些度量以生产出可靠的软件为目的，适用于整个软件开发生存周期，该标准没有指定或要求采用任何一种度量。目的是描述各个度量和它们的用途。

该标准的补充标准是 982.2，它为 IEEE Standard 982.1 中度量的使用提供指南。它为该行业提供所需的信息，以便充分利用 IEEE Standard 982.1。

IEEE/ANSI Standard 990-1987

IEEE Recommended Practice for Ada as a Program Design Language (Reaff.1992)

该推荐实践为基于 Ada 编程语言句法和语义的程序设计语言（PDL）特征提供了许多建议，这些建议反映了各种最新的优秀方案。在该文件中，它们称为 Ada PDL。

IEEE/ANSI Standard 1002-1987

IEEE Standard Taxonomy for Software Engineering Standards(Reaff.1992)

该标准描述软件工程标准分类的形式和内容。解释了各种类型的软件工程标准，它们的功能和外部关系以及各种功能在软件生存周期中所起的作用。组织可以采用该分类作为标准的开发或评估计划的方法之一。也可以作为标准分类或编制标准手册的基础。

IEEE/ANSI Standard 1016-1987

IEEE Recommended Practice for Software Design Descriptions(Reaff.1993)

软件设计描述是对软件系统的描绘，可用作交流软件设计信息的媒介。该推荐实践介绍了软件设计的文件编制。它为软件设计描述指定了必要的信息内容和推荐的组织。

IEEE/ANSI Standard 1042-1987

IEEE Guide to Software Configuration Management (Reaff.1993)

该指南的用途是为与 IEEE Standard 828 兼容的软件配置管理（SCM）实践的计划提供指导。该指南的重点是 SCM 计划的过程，同时也为软件配置管理的理解提供了广泛的视角。

IEEE/ANSI Standard 1045-1992

IEEE Standard for Software Productivity Metrics

该标准定义了度量和报告软件过程生产力的框架，重点在定义如何度量软件过程生产力，以及在提供生产力结果时报告哪些内容。该标准的使用对象是那些想度量生产代码和文件编制产品的软件过程生产力的人。

IEEE/ANSI Standard 1058.1-1987

IEEE Standard for Software Project Management Plans (Reaff.1993)

该标准指定软件工程管理计划的格式和内容。它没有指定开发或项目管理计划应采用的

过程或技术，也没有提供项目管理计划的例子。相反，该标准为组织建立起自己的开发项目管理计划的方法和过程提供了基础。

IEEE/ANSI Standard 1061-1992

IEEE Standard for a Software Quality Metrics Methodology

该标准为建立质量要求以及确定、实施、分析和确认软件质量度量的过程和产品提供了方法。该标准没有指定具体的度量方法，但包含了度量的例子以及应用标准的完整示范。

IEEE/ANSI Standard 1063-1987

IEEE Standard for Software User Documentation (Reaff. 1993)

该标准为用户文件的结构和信息内容规定了最低要求，主要适用于技术内容而不是风格。该标准的使用者可以在其组织内部开发具有自己风格的使用手册，实现该标准的要求。

IEEE/ANSI Standard 1074-1991

IEEE Standard for Developing Software Life Cycle Processes

该标准定义了构成软件开发和维护的必要过程的一系列活动，包括整个生存周期的管理和支持过程，以及软件生存周期从概念的形成到淘汰的各个方面。还提供了相关的输入和输出信息。利用这些过程以及它们的组成活动，可使用户在采用该标准生存周期时获得最大的好处，并将其映射到典型的软件生存周期中。制定该标准的目的不在于定义或推荐自己的软件生存周期。

IEEE/ANSI Standard 1209-1992

IEEE Recommended Practice for the Evaluation and Selection of CASE Tools

该推荐实践为个人和组织提供了评估和选择计算机辅助软件工程（CASE）工具的过程。该推荐实践介绍了支持软件工程过程包括项目管理过程、开发过程和集成过程的工具的评估和选择。

IEEE/ANSI Standard 1219-1992

IEEE Standard for Software Maintenance

该标准定义了软件维护的过程。它规定了软件维护活动的计划、执行和文件编制的处理、控制和管理要求。

IEEE/ANSI Standard 1298-1992

IEEE Software Quality Management System, Part1: Requirements

这是澳大利亚标准 AS 3563.1-1991。该标准规定了对软件开发人员质量管理系统的要求。它确定了开发人员准备设计、开发并维护的质量管理系统的各个因素，目的是确保软件符合合同、购货订单或其他契约的要求。

SPICE 产品

入门指南（IG）。该指南描述 SPICE 产品的组合，并为制定行业标准时选择和使用这些标准提供指导和要求。

过程评估指南（PAG）。定义采用基础实践指南（BPG）和评估工具（AI）时怎样实施评估。

基础实践指南（BPG）。在高层次上定义优秀软件工程的主要实践。

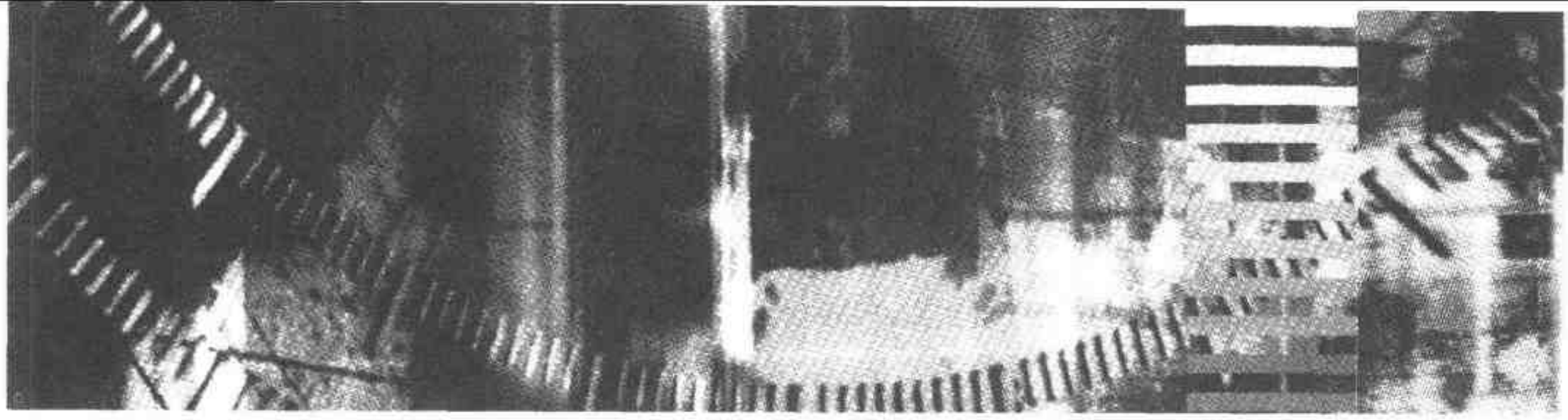
评估工具（AI）。对表格的制作和使用提供指南，这些表格用于收集接受评估的过程的有关数据，有助于确认基本实践已经到位。

过程改进指南（PIG）。为采用 PAG、BPG、和 AI 进行软件过程改进提供指导。

过程能力确定指南（PCDG）。为采用 PAG、BPG 和 AI 进行供应商评估和/或认证提供指导。

评估人员培训和资格评定指南（ATQG）。为采用 SPICE 标准开发评估人员培训项目提供指导。

（注：写作本书时，一些 SPICE 产品仍处于起草、评审和试用阶段）。



附录B

验证审查单

本附录包括下列审查单：

- 需求（规格说明）验证审查单
- 功能设计验证审查单
- 内部设计验证审查单
- 一般代码验证审查单（1）
- 一般代码验证审查单（2）
- “C”代码验证审查单
- “COBOL”代码验证审查单
- 一般文档验证审查单

需求（规格说明）验证审查单

以下内容来自波音计算机操作需求审查单，该审查单发表在 D.P.Freedman 和 G.M.Weinberg 所著的《走查、审查和技术评审手册》第三版，294~295 页，1990。经 Dorset House 出版社同意使用。保留版权。感谢波音计算机服务公司的确认。

（1）完整性。所有的条款必须详细说明所列问题的解决方案。

（2）正确性。每一个条款必须是没有错误的。

（3）精确、无二义性、有条理。每一项条款精确不含糊，只能有一种解释；每项条款含义易于理解，说明容易读懂。

（4）一致性。在需求规格说明中任何一项条款不能与其他条款相冲突。

（5）相关性。每项条款都应与问题和其解决方案有关。

（6）可测试性。在程序开发和接受测试的过程中，依据条款可以判定程序是否满足条款的要求。

（7）可追踪性。每项条款都可以在它的问题环境中追踪到它的起源。

（8）可行性。每项条款在规定的开销和时间限制下，利用现有的技术、工具、资源和人

员，可以实现。

（9）没有无根据的详细设计内容。需求规格说明是解决问题所必须满足的需求的陈述，不能被设想的问题解决方案所掩盖。

（10）可管理性。需求的表达必须遵循一个原则，即每一项条款的变更不会与其他条款发生冲突。

对于已完成的需求规格说明的更改要受控；对于已存在的需求的每次计划更改要进行跟踪。并且，计划的更改所造成的冲突要可以评估。

功能设计验证审查单

以下内容来自由 D.P.Freedman 和 G.M.Weinberg 所著的《走查、审查和技术评审手册》第二版，303~308 页，1990 年。经 Dorset House 出版社同意使用，保留版权。

通过改变规格说明的形式揭示规格说明中含糊、错误和/或曲解内容的一些方法

（1）通过改变语句中的强调方式来暴露可能出现的二义性。

（2）如果一个术语（概念）在别处已明确定义，当遇到该术语时，试着用定义替换该术语。

（3）当一个结构被用文字描述时，试着勾画被描述的结构的结构图。

（4）当一个结构被用图形描述时，试着以强调不同方面的形式重新勾画该结构的图形。

（5）当遇到一个等式时，试着用文字形式表达等式的含义。

（6）当文字中指定或隐含一个计算时，试着用等式表达该计算。

（7）当遇到指定的计算时，至少手工计算两个例子，并把例子写入规格说明。

（8）寻找含有必然性（CERTAINTY）意义的描述，找到证据。如 ALWAYS、EVERY、ALL、NONE 的词往往预示着无需证明的必然。

（9）查找确定性语句时，应根据需要按下 SEARCH BACK，一层层往回搜索，直到满足计算机需要的确定性。

（10）注意那些具有劝说性的词汇，如 CERTAINLY（当然）、THEREFORE（因此）、CLEARLY（清楚地看到）或 ANY FOOL CAN PLAINLY SEE（任何傻瓜都可以看到）。

（11）注意含义不明确的词，例如一些（SOME）、有时（SOMETIMES）、经常（OFTEN）、通常（USUALLY）、平常（ORDINARILY）、通常（CUSTOMARILY）、大多数（MOST）或主要地（MOSTLY）。

（12）当给定了功能设计清单但不全面时，确定对后来各项有一个全面的理解。注意等等（ETC.）、等等（AND SO FORTH）、等等（AND SO ON）或例如（SUCH AS）等词。

（13）在试图阐明功能清单时，同（12）中一样，我们有时会规定一个规则，确信该规则不能包含未规定的假设。

（14）查找功能清单，找出没有例子或例子太少或例子彼此雷同的并说明这些规则。

（15）当心那些含义含糊的动词，如处理（HANDLED）、处理（PROCESSED）、丢弃（REJECTED）、跳过（SKIPPED）或除去（ELIMINATED）。

（16）被动语态的结构也是个陷阱。因为被动语态的未指明执行者，很容易忽略谁在做这项工作。

（17）特别注意那些没有参考的比较（COMPARATIVES WITHOUT REFERENTS）。

（18）通常只有作者明白代名词（PRONOUNS）的所指，而读者不明白。

内部设计验证审查单

以下内容来自由 D.P.Freedman 和 G.M.Weinberg 所著的《走查、审查和技术评审手册》第二版，313~315 页，1990 年版。经 Dorset House 出版社同意使用，保留版权。感谢波音计算机服务公司的确认。

使用本审查单的前提：假设至少从两个层次描述内部设计，并且有独立的文档。这两个层次为：初步设计（概要设计）和相继的详细设计。

波音公司计算机程序初步设计文档评审审查单

（1）是否清楚地规定了初步设计的目标？

（2）初步设计文档中是否包含了初步设计的过程，或是否有针对初步设计过程的参考说明？这个过程应该包含了下列内容：

（a）对使用设计技术的描述。

（b）对设计表示方法的解释。

（c）对使用的测试过程、测试用例、测试结果和测试分析的描述。

（d）对使用的评估过程和评估标准的描述。

（3）有没有一个计算机系统将要提供的功能清单？

（4）有没有一个计算机系统的用户界面模型？模型应该提供如下内容：

（a）对用户可用的语言的描述。

（b）可供你在自己的桌面上仿真计算机系统的使用过程的足够细节。

（c）关于用户界面的机动性、适应性、可扩展性等信息。

（d）给用户的指南、技术支持等信息。

（e）对用户可用功能和使用这些功能的途径的描述。

（f）对该计算机系统的易使用性的正确评价。

（g）对于要使用的计算机系统的用户过程的阐述和实际操作的详细要求。

（5）有没有计算机系统所有其他接口的模型和/或描述。

（6）是否存在一个对目标计算机系统的高层次的功能模型？该模型应该具有下列内容：

（a）一个操作说明。

（b）一个对测试过程、测试用例、测试结果和判断模型正确与否的测试分析的解释。

（c）判断需求是否满足要求的需求评估模型。（在初步设计中不涉及对细节的定性的和定量的分析）。

（d）对所有可供选择的方案，以及否决每一个可供选择办法的原因的讨论。

（7）在文档中是否有主要实施的选择方案和对它们的评估的描述？对其中的每一个可供

选择的方案，应该包括下列内容：

（a）一个全面、精确、无二义性的模型，确定模块、模块的输入输出集合，模块操作流程和执行模型操作流程的准则。

（b）确定模型是否满足需求的评估标准，评估标准可以包括：

- 性能
- 存储需求
- 结果的质量
- 易用性
- 可维护性
- 适应性
- 通用性
- 技术先进性
- 简单性
- 机动性
- 可读性
- 可移植性
- 模块性

对模型的检查要包括功能模型和相关的数据模型两个部分。

（c）评估将要执行的可选方案的费用、时间和其他资源需求，这些评估将伴随着下列内容：

- 对使用的评估技术的描述
- 进行评估时所用数据的来源
- 与每项评价相关的置信度因素

（d）用硬件实现和软件实现模块的鉴别（有些是由软硬件结合起来实现的），这同样包括对购买、购买和修改、或建立一个模块的建议。每个建议都要有相应的信息支持。

（8）关于可选方案概要设计队伍有没有向实施者提出建议？

（9）概要设计队伍的建议是否得到足够的支持？

（10）在概要设计文档中和概要设计评审阶段提出的资料是否能给你足够的信心，使你相信你要实现的计算机系统达到你希望使用系统的需求？

一般代码验证审查单（1）

数据引用错误

（A）数组、字符串、记录、变量记录、常量和变量的引用

（1）对于所有涉及到的数组，每个下标值是否都是整数，并且在维数定义的上下限之内？

（2）标识字符串时，是否超越字符串的边界？标识运算或脚本时，是否出现“差一个”数组的错误？

（3）是否引用了未说明或未初始化的变量？

（4）某些实际中使用常量的地方，使用字母代替是否会更好（例如边界检查）？

（B）存储属性

（1）存储区域是否有别名？是否总能赋予它正确的属性？

（2）变量值的类型或属性是否与编译器或解释器所期望的不同？

（3）如果使用了指针或引用变量，引用的存储器是否具有编译器/解释器所希望的属性？

（4）如果一个数据结构在多个函数或过程中被引用，在不同过程中对该数据结构的定义是否一致？

（C）内存分配

（1）是否为引用指针分配了内存？

（2）在机器使用过程中，如果内存分配单元小于内存地址单元，是否会有显式或隐含的地址问题？

（D）其他

数据说明错误

（A）变量和常量的属性

（1）如果变量在声明时被初始化，初始化是否正确且与存储类型相匹配？

（2）变量是否分配正确长度、类型和存储类？

（B）变量和常量的使用

（1）是否存在变量重名问题？这未必是错误，但是，这是一个程序中名字混乱的信号。

（2）所有的变量是否都在它们所在的程序块中被显式的声明了？如果不是，变量与封闭程序块的关系是否能被理解？

（C）其他

计算错误

（A）数据类型描述

（1）是否有计算使用的变量的数据类型相互矛盾？

（2）是否存在混合模式的计算？

（3）是否有计算使用的变量数据类型相同，但长度不同？

（4）目标变量（保存计算结果的变量）的分配空间是否小于右边的表达式？

（B）算术、关系、布尔和指针赋值错误

（1）数值计算中是否有中间结果上溢或下溢的可能？

（2）是否可能除数/模为零？

（3）如果指定的机器用二进制表示变量，计算结果是否会不精确？

（4）是否非法使用了整数计算，尤其是除法？

（C）设计赋值错误

（1）变量的值是否会超出它的合理范围？例如，应检查变量 PROBABILITY 的值以确保在闭区间[0，1.0]之内。

（2）对于包含多个操作符的表达式，设定赋值和运算的先后顺序是否正确？

（D）其他

比较错误

（A）算术

（1）在指定的以 2 进制为基础的机器中，是否存在分数或浮点数之间的比较？

（2）编译器/解释器计算布尔表达式的方式是否会影响程序？

（B）布尔

（1）对于包含多个布尔操作的表达式，设定赋值的先后顺序是否正确？

（2）每一个布尔表达式是否规定了它应该的状态？每一个布尔操作的运算数都是布尔型？

（C）转换

（1）是否为了对类型或长度不一致的数据或变量进行比较而运用了转换法则？

（D）比较运算符

（1）比较运算符都正确吗？

（E）其他

控制流程错误

（A）入口/出口条件

（1）如果程序语言中包含语句组概念，有显式的结尾（END）吗？结尾与其相应的语句组匹配吗？

（2）如果需要，程序、模块、子程序或循环体有最终结尾吗？

（3）是否会过早退出循环？

（4）是否有这种可能，由于入口条件的原因，某个循环永远不能执行？这是否是一个疏忽？

（5）如果一个程序包含有多个分支，索引变量是否保证能执行到每个分支？

（B）循环

（1）“无限循环”的结果是什么？（例如，while 语句条件永远为真）

（2）是否会有“差一个循环”的错误？

（C）不完整的判定

（1）有不完整的判定吗？（例如，某个输入参数的期望值为 1 或 2，逻辑假设是否会认为如果不是 1 则一定是 2？这个假设有效吗？）

（D）其他

接口错误

（A）参数和自变量

（1）被调用模块接收到的参数的个数和属性（如：类型和大小）是否按照设计文档与每一个调用该模块的模块传送的参数相匹配？参数的顺序正确吗？这个规则同样适用于用户定义和建立的函数和函数库。

（2）如果某个模块有多个入口点，是否引用的某个参数与当前入口点不相关？

（3）常量是否像自变量一样被传递？如果在接收的子程序中对常量参数分配了值，常量的值将会被改变。

（B）赋值

（1）子程序是否会更改只作为输入值的参数？

（2）每个参数的单位是否与相应的自变量的单位相匹配？（如：度还是弧度？）

（3）如果说明了全局变量，它们在所有的引用模块中是否具有相同的定义和属性？

（C）其他

输入/输出错误

（A）文件

（1）如果文件被显式声明了，它们的属性是否正确？

（2）所有的文件是否在使用前都首先被打开？

（B）错误

（1）文件尾（EOF）和输入/输出错误是否都进行了检查和处理？

（2）在程序输出文本中是否存在语法错误？

（C）格式错误

（1）格式说明是否与输入/输出语句中的信息相一致？

（D）存储区域

（1）输入/输出存储空间和记录空间是否匹配？

可移植性

（A）存储类/大小

（1）数据是如何被组织的（例如，紧缩结构）？

（2）是否约定了符号集（即整理次序）？

（3）是否有约定好的系统参数集（HP-UX 系统返回编码）？

（4）是否考虑了将产生的代码类型（如：生成不同的目标代码）？

（B）其他

其他检查项

（A）清单

（1）如果生成了交叉引用清单，从中找出从未被引用或个别被引用的变量。

（2）如果构造了属性清单，利用它确保不会分派预想不到的缺省属性。

（B）警告

（1）如果程序编译成功，但是出现了“警告（warning）”或“报告（informational）”信息，检查每一条信息，这可能意味着你所做工作的有效性存在问题。

（C）健壮性

（1）程序或模块是否十分健壮？也就是说是否检查输入的有效性？

（2）程序中是否遗漏了某个函数？

（3）是否有内存错误（例如 heap 和/或 stack）。

（D）其他

一般代码验证审查单（2）

以下内容来自由 D.P.Freedman 和 G.M.Weinberg 所著的《走查、审查和技术评审手册》第二版，337~338 页，1990 年。经 Dorset House 出版社同意使用。

评审代码时要记住的一般问题

功能

（1）能不能用简单语言表达某个概念（特定的想法）？在实现的代码中它是否已用简单的语言表达？

（2）在全部功能中，每个功能是否有其明确的位置？是否表达清楚？

（3）程序是否被适当地保护，这样即使可能被误用，也可以可靠地实现其功能。

形式

（1）无论采用什么风格，作为一个整体程序是否清楚整洁？

（2）对于各层次的读者来说是否都是有意义的？

（3）在程序内部或程序之间有没有重复的代码段？

（4）注释是否有用？是否只是不良代码的一种托词？

（5）详细程度是否一致？

（6）是否使用了标准实践？

（7）是否进行了适当的初始化，并且在程序结束后是否进行了必要的释放？

经济利益

（1）是否存在毫无利益的多余操作？

（2）无论从内部或外部规格说明来考虑，存储器使用是否一致？

（3）修改需要多少费用？（考虑一项将来最可能的修改）

（4）系统简单吗？

C 语言代码验证审查单

这是一个由开发部门为软件开发制定的审查单，多数条款与可维护性、易读性而不是可靠性相关。同时，多数条款可以由一个用户编译器或语法检查实用程序强制检查。开发人员不必为可以自动进行的检查而反复研究它们。

功能性

（1）每个程序单元是否只有单一功能？

（2）是否存在某段代码应该放在一个独立的函数中？

（3）程序代码和性能需求一致吗？

（4）代码和详细设计相吻合吗？（问题既可能在编码中，也可能在设计中。）

数据使用

（A）数据和变量

（1）所有变量名都是小写吗？

（2）内部名字长度是否都限制在 8 个字符以内？

（3）外部名字长度是否都限制在 6 个字符以内？

（4）所有的赋初值都使用了“=”？

（5）声明分成了内部组和外部组吗？

（6）除了显式的声明外所有声明都有注释吗？

（7）每个名字仅被一个函数使用吗？（单字符变量除外）

（B）常量

（1）所有常量名用大写字母吗？

（2）所有的常量定义都是通过“#define”语句进行的吗？

（3）被多个文件使用的常量是否放入 INCLUDE 头文件中？

（C）指针类型

（1）指针是否声明了并作为指针使用了（不是作为整数）？

（2）指针类型变量是否指向应指的类型？（除非指针为 NULL）？

控制

（1）“else_if”和“switch”使用的清楚吗？（通常“else_if”比较清楚，但是“switch”可以用于非互斥的情况并且更快）

（2）“goto”和“labels”是否只在绝对必要的情况下使用，并且使用时有清楚的注释代码？

（3）如果可能，使用“while”而不是“do-while”了吗？

连接

（1）“INCLUDE”文件的使用是否符合工程标准？
（2）是否避免了“INCLUDE”文件的相互嵌套？
（3）除非特别必要并说明全局连接，所有的数据是否局限在一定的活动范围（内部静态或外部静态）内？
（4）所有的宏都用大写字母命名了吗？

计算

（A）操作符的词法规则
（1）一元操作符是否与它们的操作数相邻？
（2）主要操作符“->”、“.”、“()”周围有空格吗？（不应该有空格）
（3）赋值符和条件操作符左右总是有空格吗？
（4）逗号和分号后面跟着空格吗？
（5）关键字后面有空白吗？
（6）函数名后的“(”是否与标识符相邻？
（7）空格是否用于表示优先顺序？如果优先顺序复杂，是否使用圆括号表示其优先顺序？（特别是对于位操作）

（B）求值的顺序
（1）针对优先级，圆括号使用是否恰当？
（2）除了下列的情况，代码执行是否依赖于求值的顺序？
（a）expr1,expr2（表达式1，表达式2）
（b）expr1?expr2: expr3（表达式1？表达式2：表达式3）
（c）expr1&&expr2（表达式1&&表达式2）
（d）expr1‖expr2（表达式1‖表达式2）
（3）移位（shift）使用的是否恰当？
（4）代码的执行结果是否依赖于求值顺序？（例如 i=i++;）

维护

（1）子程序和好的文档中非标准的用法是否被分离开？
（2）每个程序单元是否只有一个出口？
（3）程序单元是否易于修改？
（4）程序单元是否尽可能独立于具体的设备？
（5）如果可能，是否使用系统标准定义的类型头部（否则通过“INCLUDE”使用工程标准的头部）？
（6）避免使用“int”（使用标准的定义类型代替）了吗？

清晰性

（A）注释

（1）程序单元头部是否提供了丰富完整的信息？

（2）有没有提供有效的注释帮助我们理解程序代码？

（3）程序单元中的注释是否可以提供丰富的信息？

（4）与逻辑段落相关的注释行的内容是否与程序语句有关系？

（5）函数的数组和变量是否有说明？

（6）在程序头部的开发历史的说明中是否发布了更改方面的注释信息？

（B）规划布局

（1）对程序代码的布局是否使程序的逻辑明显？

（2）循环部分是否缩进并可以看出循环体与其上下代码的分离？

（C）词法控制结构

（1）是否使用一个标准的、工程范围的（至少一致的）词法控制结构模式？例如：

```
{
}
```

或

```
while (expr) {
stmts;
}
```

等等

COBOL 语言代码验证审查单

以下内容来自 IBM COBOL 程序审查单，发表于 D.P.Freedman 和 G.M.Weinberg 所著的《走查、审查和技术评审手册》第三版，339~344 页，1990 年。经 Dorset House 出版社同意使用。同样感谢国际商用机器公司的认可。

标识符部分

（1）函数在 REMARKS 段的文字是否为程序的完整序言？

环境部分

（1）对于文件，每个 SELECT 语句是否明确定义了外部（依赖于系统的）规格说明？

数据部分——文件节

（1）文件定义（FD）是否和在环境部分中相应的 SELECT 语句具有一致的顺序？

（2）记录和数据项的命名是否与其使用含义相符？

（3）每个文件定义（FD）是否包括相关的注释：

（a）文件使用（记录模式、块大小、记录长度、嵌入的关键字等）

（b）活动的数量（升级频度、每次程序运行的时间等）

（c）与其他数据项的交互关系（记录中是否包含 OCCURS...DEPENDING ON 子句对象？无论在程序的哪个位置，记录的长度是否依赖于上述对象等？）

（4）文件被分类或者被合并了吗？

（5）在给定操作或系列操作中是否一直对文件活动进行统计？

（6）在程序中规定的完整的文件属性和它们规定的一些动态属性（例如块大小、最大记录长度）之间存在着平衡关系，这种正确的平衡是否被打破了？也就是说，如果一个文件在指定的程序中设计得灵活，是否需要定义得这样灵活？

数据部分——工作存储器和连接节

（1）数据项的命名是否与它们的使用相符？

（2）每个数据项（除了对基本项的明显的使用如下标等以外）是否包括相关的注释：

（a）特性（固定或可变长度、最大可用长度等）

（b）与其他数据项的交互（数据项中是否包含或依赖于 OCCURS...DEPENDING ON 子句等？）

（c）程序中使用的范围（只在某个确定部分中使用，还是在整个程序段中使用等？）

（3）具有任何一种统一特性的所有数据项是否按照一定的格式放在一起了？

（a）使用（算术工作区域、文件记录的工作区域等）

（b）公共目的（用于处理特殊文件的所有内容等）

（c）属性（信息文字、常量等）

（4）作为常量使用的工作存储项是按规定指定的吗？

（5）按照指定的顺序要求的数据项是否排列正确？

（6）就简化数据引用而言，而非根据层号的正常层次，在数据描述中使用 REDEFINE/RENAME 是否合理且已文档化？

过程部分

（1）主要函数部分是否包括块注释？（例如：程序段落、节、片段）

（2）模块的注释是否足够详细？

（3）注释是否准确并且内容有意义？

（4）程序代码是否与在模块的注释段中的文档化的说明基本符合？

（5）每个段落、节或片段是否有一个相似的目的，从而证明将一组代码放在一起的必要性？

（6）每个执行段落或部分是否支持它完成的功能和它所代表的全部逻辑？

（7）在分段的程序中，分段的必要性是否清楚？

（8）每个段是不依赖其他段独立存在，还是非常依赖其他程序片段？

格式部分

（1）IFTHENELSE 和 DO 语句组结合得是否合适？

（2）IF 语句嵌套是否恰当地缩进了？

（3）注释是否准确并且内容有意义？

（4）是否使用了有意义的标号？

（5）复杂动词（例如 SORT/MERGE、OPEN/CLOSE）的子句是否恰当地缩进并清楚地置于动词之下？

（6）所有的 GO TO 使用是否符合安装规范？

外部连接

（1）所有的初始入口和最终出口是否正确？

（2）每一个入口点定义正确吗？

（3）在连接部分的 ENTRY 语句的 77 或 01 项每个参数都被引用了吗？

（4）动词 STOP、RUN/GOBACK/EXIT PROGRAM 的使用是否正确？

（5）每一个对其他模块的外部调用：

（a）所有要求的参数传到了每个被调用模块中吗？

（b）传输的参数值设置得正确吗？

（c）在模块的最终退出之前，是否关闭所有的文件？

逻辑

（1）所有设计都实现了吗？

（2）代码是否做了设计规定的工作？

（3）设计正确完整吗？

（4）域中字符的个数是否被测试并设置正确？

（5）每个循环都执行到且执行次数正确吗？

程序语言的使用

（1）使用了最佳动词或动词组吗？

（2）对于安装时定义的 COBOL 语言的限制子集的使用是否贯穿于模块始终？

（3）是否关注 COBOL 中关于内务处理的需求？（例如：设置可变长度目标域的长度，然后再对该域执行 MOVE）？

内存使用

（1）在首次使用之前对每个域进行适当的初始化了吗？

（2）指明了正确的域了吗？

（3）如果设置并循环使用某个存储区域，对它的处理是否恰当？

（4）当需要动态分配域时，它是否被静态初始化了（也就是说，采用了定义中的 VALUE 子句）？

（5）在数据项的定义中，REDEFINES 子句的使用是否与数据项在代码中的所有使用相一致？

（6）如果使用了 MOVE 的 CORRESPONDING 选项和算术动词，是否能和将影响目标域的数据定义绝对区分开来？

测试和分支

（1）正确的条件被测试了吗（IF X=ON 与 IF X=OFF）？

（2）正确的变量是否用于测试了（IF X=ON 与 IF Y=ON）？

（3）用于数据项的测试，每个条件名被定义在该数据项下 88-level？

（4）每个简单的 GO TO 或 GO TO...DEPENDING ON 分支语句是否正确并至少执行一次？

（5）在 IF 语句中，多数测试执行到的分支语句是否是 THEN 子句？

性能

（1）程序逻辑最优吗（也就是用最短的语句最有效地实现）？

（2）在检索逻辑处使用下标是否能更有效、更合理，反之如何？

（3）文件中的错误定义（ERROR DECLARATIVE）编码是否可能覆盖 I/O 错误？

（4）一般的错误/异常（error/exception）程序是否提供：

（a）对于算术语句有 ON SIZE ERROR？

（b）对于 start/read/write/rewrite 语句有 INVALID KEY？

（c）对于 search/release/sequential READ 语句有 AT END？

（d）对于 STRING/UNSTRING 有 ON OVERFLOW？

可维护性

（1）使用的列表控制是否增强了可读性（例如，EJECT，SKIPx）

（2）程序段和 SECTION 名是否和代码中的逻辑意义相一致？

（3）每个 PERFORM 段是否以 EXIT 段结尾？

（4）使用 ALTER 语句，而不是采用某种转移/条件分支流程控制，是否完全合理？

（5）包含的所有空 ELSE 是否恰当？

复制程序的使用

（1）经过标准化处理以便安装的每个数据项定义和处理段落，是通过 COPY 程序在模块中生成的吗？

（2）在 COPY 语句代码中，COPY 语句的 REPLACE 选项用于改变数据项的命名是否有合理的原因？

普通文档验证审查单

以下内容可以用于评审用户手册或其他任何文档，该内容发表在 D.P.Freedman 和 G.M.Weinberg 所著《走查、审查和技术评审手册》第三版，356~359 页，1990 年。经 Dorset House 出版社同意使用。

评审文档时要寻找的缺陷清单

（1）文档生存周期的各个阶段是否都考虑到了？
- （a） 有没有用户反馈方面的条款？
- （b） 有没有变更方面的条款？
- （c） 系统的变更是否会导致文档编制的困难或开销改变？
- （d） 有没有文档分发方面的相关规定？
- （e） 有没有文档分布更改方面的相关条款？
- （f） 文档易于再生吗？
- （g） 拷贝受限或受控吗？
- （h） 是否有专人补充文档？
- （i） 用户和开发者是否就文档的意图进行了约定？
- （j） 有没有使技术支持人员了解当前情况的有关规定？
- （k） 有没有阅读/进入/存储这些资料的可用工具？
- （l） 文档是否有适当的批准？
- （m） 文档是否在总计划中处于该处的位置？
- （n） 文档中有没有指出可能后继的其他文档？

（2）文档的内容充分吗？
- （a）主题覆盖
  - （ i ）所有主要的主题完整吗？
  - （ ii ）是否排除了所有不相关的主题？
  - （iii）主题完整，但是在细节、设想、论据和未知数方面是否完整？
  - （ iv ）技术水平是否和文档水平相符合？
  - （ v ）谁是读者？
- （b）正确性
  - （ i ）论据（事实）无误？
  - （ ii ）没有矛盾吗？
- （c）依据，根据
  - （ i ）有没有支持描述的相应依据？
  - （ ii ）依据真实吗？
  - （iii）有没有关于文档目标的清楚描述，所有目标相符合吗？
  - （ iv ）陈述有权威性吗？

（3）文档中的资料清晰吗？

（a）例子清楚吗？

（i）在需要的地方有例子吗？

（ii）例子和其对应的内容相关吗？

（iii）有助于理解吗？

（iv）例子易误解吗？

（v）例子有错误吗？

（vi）例子的效果完全发挥了吗？

（b）图表、图片或其他可视资料清楚吗？

（i）在需要的地方有图表、图片或其他可视资料吗？

（ii）图表、图片或其他可视资料和其对应的内容相关吗？

（iii）提供的图表、图片或其他可视资料有助于理解吗？

（iv）图表、图片或其他可视资料易误解吗？

（v）图表、图片或其他可视资料有错误吗？

（vi）图表、图片或其他可视资料的效果完全发挥了吗？

（vii）信息的数量适当吗？

（c）使用的技术明确吗？

（i）所有的文档协调无冲突吗？

（ii）规格标准相互符合吗？

（iii）如果合适，是否有术语表？

（iv）定义正确吗？

（v）定义清楚吗？

（vi）术语表全面吗？

（vii）是否包括了太多的技术术语？

（d）书写风格清楚吗？

（i）段落内容仅与相关的思想有关，与其他无关吗？

（ii）大的逻辑单元是否由小标题分开？

（iii）对于读者来说，模糊的索引是否太笼统了？

（iv）对典型读者来说是否说得太简单了？

（v）内容是否太平淡，容易使人入睡？

（vi）有没有摘要？

（4）文档提供了充分的引用帮助了吗？

（a）如果合适，是否有一个内容表？

（b）内容表是否在合适的位置？

（c）如果合适，是否有索引？

（d）索引是否在合适的位置？

（e）内容表正确吗？

(f) 索引正确吗?

(i) 引用的页号准确吗?

(ii) 是否有不同类型用户要查找的各种相关内容的入口?

(iii) 入口是否在正确的标题下?

(iv) 是否有可替换标题的入口，使用不同术语进入它?

(v) 同一个术语的主要和次要入口是否有区别?

(vi) 项目分解得是否充分，或对于单个项目的引用是否太多? 过多的子目录是否必要。

(vii) 是否有多余的入口?

(g) 有没有必备的出版物的参考书目?

(i) 如果没有先导内容，可以肯定吗?

(ii) 在试图开始阅读文档之前，是否可以找到参考书目?

(iii) 引用是否提供足够的内容，以便读者查到相关内容的参考资料?

(iv) 有没有帮助读者选择合适文档的注释信息?

(h) 有没有一个相关出版物的参考书目，其中包含了更多的信息?

(i) 如果是信息的惟一来源，可以肯定吗?

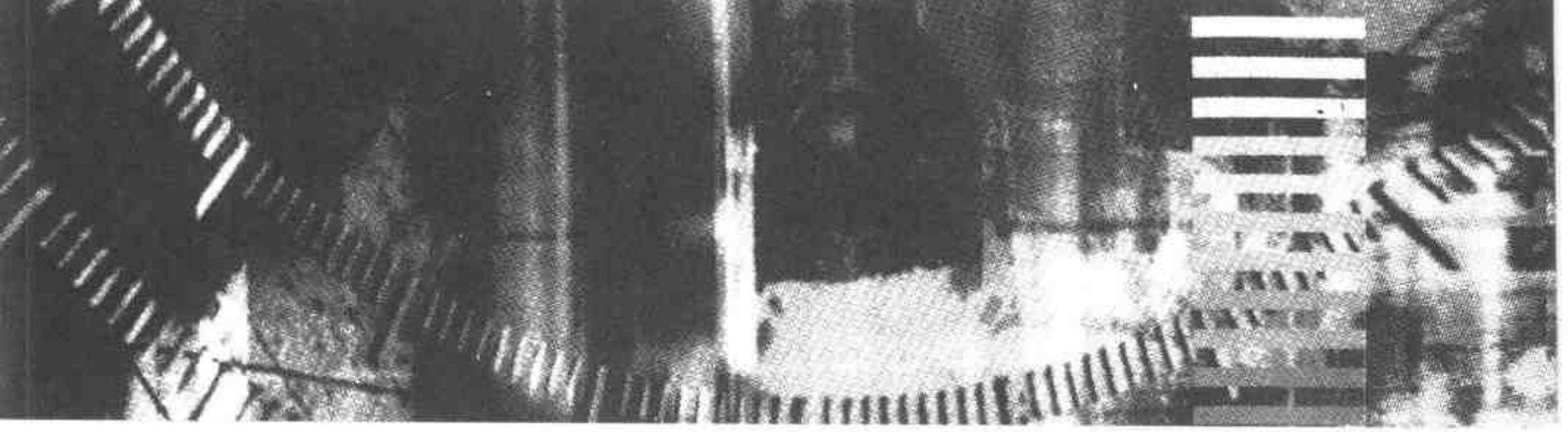
(ii) 引用是否提供足够的内容，以便读者查到相关内容的参考资料?

(iii) 有没有帮助读者选择合适文档的注释信息?

(i) 文档本身的组织是否有利于轻松地查找信息?

(i) 页编码切合实际吗?

(ii) 页编码完整吗?



附录C

验 证 练 习

需求验证练习

下面的文档是一个预定系统的需求规格说明。假设你在准备一个审查会，运用第 7 章需求验证一节关于审查和评审的指导方针，试着评审这个需求规格说明。

记住成功的验证需要充裕的时间。即使一个好的检查员检查一页也需要一小时时间，事实证明这么做是值得的。

运用附录 B 中的需求验证审查单，找寻任何意思模糊、冗长、不可测试的条款。基于这份详细验证，如果你认为在将来的文档还有其他需要的检查项目，将其加到审查单中。

记录你进行验证的时间。

如果你发现的错误被遗留到开发的下一个阶段或甚至交给用户，估计更改所需要的时间和花费。

下面是这个验证练习的一个方案。

Lili'uokalani 皇后营地预定系统需求规格说明

位于夏威夷 Kona 的 Lili'uokalani 皇后儿童活动中心，为了夏威夷土著儿童的利益和娱乐，为他们在 Papawai 海滩保留了家庭营地。

目标

开发一套在 IBM 个人计算机上使用的野营地预定系统。

预定策略

（1）有三种不同的场所，可以容纳的人数如下：

（ i ）主（Main）营地，最多容纳 50 个露营者。

（ ii ）中路（Middle Road）营地，最多容纳 30 个露营者。

（iii）小块（Point）营地，最多可容纳 30 个露营者。

（2）每种场所指定了露营区域：

（i）主营地 5 个区域。

（ii）中路（Middle Road）营地 3 个区域。

（iii）小块（Point）营地 3 个区域。

（3）如果需要为专门的机构服务，任何区域都可以对公众关闭。

（4）每次预定不得超过 10 人。

（5）预定提前量不得超过一个月，从预定到拿到钥匙的时间不得超过一个月。

（6）预约单必须包括：来客人数、成人和孩子的姓名。至少要有 30%的儿童，其中一半以上必须是夏威夷土著儿童或有夏威夷血统的儿童，并已在 Queen Lili'uokalani 注册。

（7）预定限于：

（i）学校上课期间连续 7 天。

（ii）学校节假日期间 4 日/3 晚。

（8）取消预约至少要提前 2 天。放弃预约的结果是得到警告。第二次警告中止 3 个月的海滩特权，第三次警告中止 6 个月的海滩特权。

（9）一把钥匙的押金 10 美元。退还钥匙时返还押金。取钥匙必须是在工作日，并且比预定的时间提前不得超过一周。钥匙从你使用的第一个工作日起由你负责保管。不按时归还钥匙将罚没押金，钥匙丢失将会导致 3 个月的停止使用并罚没押金。

软件需求

（1）用户类型：单用户系统，一个用户类型，即预定管理系统。

（2）功能环境和界面：容易使用的、菜单驱动的系统，在没有文档的情况下一个计算机初学者在 4 个小时之内可以掌握。预定系统的操作环境符合微软的 Windows 环境。

用户必须能够：

（a）登录和退出应用；

（b）增加、修改和删除夏威夷族的儿童名单；

（c）执行备份和恢复所有数据；

（d）为露营地预约和取消预约，不必以夏威夷族土著儿童的名字进行预定；

（e）进入警告或推迟通知功能；

（f）删除超过用户指定时间的旧的预约；

（g）打印、显示目前有效的警告和中止特权；

（h）按日期、名字、地点和区域打印或显示预约情况的报告；

（i）记录钥匙的押金和返还情况。

（3）文档：不要求用户文档。要求联机帮助。

（4）兼容性：必须运行在 DOS 6.0 和 Windows 3.1 上。

（5）安全性：个人计算机不一定被指定专门用于本预定系统，因此，预定系统和它的底层数据库必须受到保护，避免非法进入。

（6）可安装/可配置：无需任何文档，用户可以安装软件，并支持用户选择打印机配置。必须支持下列打印机：

（ i ）HP LaserJet 4M

（ ii ）IBM 4019 Laser Printer

（iii）Epson LQ Series

（7）性能：假定计算机系统被指定只运行这一个任务，除了数据相关（排序、搜索、报告）的性能外，所有的用户操作不能超过 3 秒钟。

（8）可靠性/可恢复性

（9）有用性

（10）可测试性

需求规格说明练习——解决方案

下面的文本后的注释是 10 个练习者用了大约 30 分钟时间进行评审的研究结果。

位于夏威夷 Kona 的 Lili'uokalani 皇后儿童活动中心，为了夏威夷土著儿童的利益和娱乐，为他们在 Papawai 海滩保留了家庭营地。

目标

开发一套在 IBM 个人计算机上使用的野营地预定系统。

预定策略

（1）有三种不同的场所，可以容纳的人数如下：

（ i ）主营地，最多容纳 50 个露营者

（ ii ）中路（Middle Road）营地，最多容纳 30 个露营者

（iii）小块（Point）营地，最多可容纳 30 个露营者

人头数？帐篷数？或其他？

每个露营地的价格如何？价格是否相同？

（2）每种场所指定了露营区域：（没有约束条件？）

（ i ）主营地 5 个区域

（ ii ）中路（Middle Road）营地 3 个区域

（iii）小块（Point）营地 3 个区域

分布策略是什么？一个区域满额或再到下一个区域，还是均匀分布？每个区域可以容纳多少个露营者？所有区域大小是否相同？是男女共用区域吗？

（3）如果需要为专门的机构服务，任何区域都可以对公众关闭。

如何通知已经预定露营地的人们？

已经预定露营地的人们怎么办？

能将人们转到其他区域吗？

以什么样的优先顺序进行上述工作？

日期、通知、退款?

（4）每次预定不得超过 10 人 （没有约束条件？）

有没有预约的最小人数要求?

是每个场地的人数吗?

（5）预定提前量不得超过一个月。两次预定之间必须有一个月的等待时间，从钥匙归还之日算起。

意思是两个月吗?

限制是由露营者，还是由预定人规定的?

（6）预约单必须包括：来客人数、成人和孩子的姓名。至少要有 30%的儿童，其中一半以上要是夏威夷土著儿童或有夏威夷血统的儿童，并已在 Queen Lili'uokalani 注册。

只要求儿童是夏威夷土著儿童或有夏威夷血统的儿童?

夏威夷血统是如何定义的?

在那儿生活了 2 年，出生在夏威夷?

它们必须有关系吗?

你检查吗?

在哪儿以及怎样建立数据库?

（7）预定限于：

（i）学校上课期间连续 7 天

（ii）学校节假日期间 4 日/3 晚。

这是否确定了所有的日期?

按哪些学校的时间表?

（8）取消预约至少要提前 2 天。放弃预约的结果是将得到警告。第二次警告将中止 3 个月的海滩特权，第三次警告将中止 6 个月的海滩特权。

第一次警告没有处罚吗?

“海滩特权”是否意味着他们可预定露营地的位子?

有没有不接受取消预定的相关规定?

（9）一把钥匙的押金 10 美元。退还钥匙时返还押金。取钥匙必须是在工作日，并且比预定的时间提前不得超过一周。钥匙从你使用的第一个工作日起由你负责保管。不按时归还钥匙将罚没押金，钥匙丢失将会导致该露营者 3 个月的停止使用并罚没押金。

如果取消预定会出现什么情况?

取消整个预定呢?

软件需求

（1）用户类型。单用户系统，一个用户类型，即预定管理系统。

（2）功能环境和界面：容易使用的，菜单驱动的系统，在没有文档的情况下一个计算机初学者在 4 个小时之内可以掌握。

预定系统的操作环境符合微软的 Windows 环境。

用户必须能够：

（a）登录和退出应用；什么意思？

（b）增加、修改和删除夏威夷族的儿童名单；详细说明？

（c）执行备份和恢复所有数据；软件支持什么类型的设备？

（d）为露营地预约和取消预约，不必以夏威夷族上著儿童的名字进行预定；

（e）进入警告或推迟通知功能；用名字？如果下次预定用其他名字怎么办？通知包括什么内容？

（f）删除超过用户指定时间的旧的预约；需要通知客户吗？

（g）打印、显示目前有效的警告和中止特权；是按每个人名字进行的吗？

（h）按日期、名字、地点和区域打印、显示预约情况的报告；

（i）记录钥匙的押金和返还情况。对于悬而未决的怎么办？

如果露营区域满员怎么办？

对于机构如何处理，要打印吗？

（3）文档：不要求用户文档。要求联机帮助。

（4）兼容性：必须运行在 DOS 6.0 和 Windows 3.1 上。

哪一种平台？

本套系统所需的最小内存是多少？

（5）安全性：个人计算机不一定被指定专门用于本预定系统，因此，预定系统和它的底层数据库必须受到保护，避免非法进入。

定义。谁是非法的？

其他应用如何？（我们应该模拟真实的环境）

不可验证或只是不可存取。

（6）可安装/可配置：无需任何文档，用户可以安装软件，并支持用户选择打印机配置。

必须支持下列打印机：

（ i ）HP LaserJet 4M

（ ii ）IBM 4019 Laser Printer

（iii）Epson LQ Series

需要多大空间？

（7）性能：假定计算机系统被指定只运行这一个任务，除了数据相关（排序、搜索、报告）的性能外，所有的用户操作不能超过 3 秒钟。

（8）可靠性/可恢复性

如何定义的？规则是什么？

当磁盘满了后会怎么样？

（9）有用性

如何定义的？规则是什么？

（10）可测试性

如何定义的？规则是什么？

关于需求规格说明的更多的疑问、问题和难题

（1）预定单中的人员如果只由儿童组成（没有成年人），可以吗？

（2）程序必须为每个露营场所或区域建立等待列表功能吗？

（3）一个 7 天预定单是指 7 个晚上还是 6 个晚上？

（4）谁会因放弃预定而受到处罚（发布警告）？预定单中的每个人，只是成年人，只是儿童，或只是进行预定的个人？

（5）在预定的等待期间，应该选用谁的名字？预定单中的每个人，只是成年人，只是儿童，或只是进行预定的个人？

（6）如果被警告了四次或四次以上会怎么样？

（7）是由用户或系统决定学校上课期间和限定于四天之内的预定？

（8）对于专门机构的预定规则和其他预定者一致吗？（例如，预定最早不得提前超过一个月）

（9）对于任何给定的露营区域，预定是严格的按照日历日期计算，隐含着以小时计算（例如中午到中午），或预定的时间限制在指定时间内（如星期三下午 3 点到星期天上午 10 点）？

（10）如何指定（命名）每个场地的区域？如何准确地识别它们？

（11）程序需要的最小内存和磁盘空间是多大？

功能设计验证的练习

下面的文档是一个销售代理系统的功能设计说明。假设你要准备一个审查会，运用第 7 章验证功能设计一节关于审查和评审的指导方针，试着评审这个功能设计说明。

运用附录 B 中的相关清单，找寻任何意思模糊、冗长、不可测试的条款。基于这份详细验证，如果你认为在将来的文档中还有其他需要检查的项目，将其加到审查单中。

记录你进行验证的时间。

如果你发现的错误被遗漏到开发的下一个阶段或甚至交给用户，估计一下进行更改所需要的时间和花费。

下面是这个验证练习的一个方案。

本练习的前提条件是了解窗口界面，下拉式菜单以及通过鼠标选择菜单项，正如大多数 Macintosh 和 PC-Windows 应用程序一样。

ABCSALES 的功能设计说明

开始显示：ABC 公司

ABC 公司销售 6 种基本产品：

（P1）X 型寻呼机（购买价 80 美元，出租 5 美元/月）

（P2）Y 型寻呼机（购买价 120 美元，出租 10 美元/月）

（P3）本地寻呼服务（12 美元/月/台，不限制使用次数）

（P4）G 型蜂窝电话（购买价 500 美元，出租 35 美元/月）

（P5）H 型蜂窝电话（购买价 270 美元，出租 20 美元/月）

（P6）本地蜂窝电话服务（60 美元/月/台，不限制使用次数）

对于租用机器消费者必须使用 ABC 公司的服务，购买机器则不用。ABC 公司的服务业务并不严格局限于购买和租赁 ABC 设备的客户。

ABC 公司有两种销售人员：

- 销售代表（SR）：销售所有产品
- 销售经理（SM）：销售所有产品并管理销售代表

ABC 公司销售委托方案

销售代表（SR）

一个销售代表每月保底薪金为$2200，设备配额为 20 部电话、10 台寻呼机，服务性配额为 20 个新的客户服务合同（任何寻呼机和电话的客户服务都可以）。不论是购买还是租赁性质都可以满足设备配额要求。

在设备配额之内，每一台（个）设备，销售代表得到销售额 4%的佣金，超过配额之外，每台设备佣金为 7%。每介绍谈成一个新的租赁客户，在配额内可得报酬 13%，在配额外可得报酬 18%；如果客户在后续的月中继续租用设备，销售代表每月可得报酬 10%。

在服务配额之内，对于每个新的服务单元销售代表得到每月服务费的 10%，超过配额之外，每介绍一个新的服务性客户，销售代表得到每月服务费的 15%；如果客户继续接受寻呼或通话服务，销售代表可得客户服务费的 8%。每个销售代表每月最多可以得到$700 的剩余服务佣金。如果他或她每月业绩超过服务配额的 150%，销售代表可得到$300 的奖金。

销售经理（SM）

一个销售经理每月保底薪金为$2900，设备任务额为 10 部电话、5 台寻呼机，没有服务性任务额。

销售经理的设备（出售和租赁的剩余）佣金与销售代表相同。除此之外，销售经理可获得佣金为所有销售代表出售和租赁设备总产值的 1%加上销售代表租赁剩余部分的 1%。销售经理可获得他或她所得新服务合同额的 14%。得到所有销售代表服务合同剩余部分费价的 2%。销售经理没有剩余的服务费提成，如果他管辖的所有销售代表都完成了月配额，销售经理得奖金$400。

一般程序功能

ABCSALES 是一个独立的功能齐全的系统，包括如下的性能和功能：

（1）ABCSALES 系统可读/写两个数据库 SALES 和 PERS，SALES 保留了每笔销售交易的完整历史记录（销售以及启动和终止租赁、服务的合同）。PERS 包括目前的销售经理和销售代表的姓名。（SALES 和 PERS 开始为空，其修改和维护由 ABCSALES 来进行）。

（2）对于每一笔销售交易，ABCSALES 接收用户输入以完整地定义本次交易，并相应地更新 SALES 数据库。

（3）ABCSALES 在数据库 PERS 中填加、删除销售代表和销售经理，并处理 SALES 数据库中的交易的所属关系。（确保所有的交易都有连续的所属关系）

（4）根据用户的命令，ABCSALES 系统可基于 SALES 和 PERS 数据库生成报告，根据销售代理计划的规定，显示所有现有销售代表的总收入和销售经理当前月的销售收入。

详细的功能描述

ABCSALES 是一个单用户系统，运行在 IBM-PC 机上，Windows 操作系统环境。

运行 ABCSALES 系统用鼠标双击程序管理窗口的 ABCSALES 图标，随后 ABCSALES 窗口就会显现出来。

ABCSALES 系统菜单条包括两种菜单：标准控制菜单（在 ABCSALES 系统窗体左上角）和命令菜单，下列命令通过命令菜单可以使用：

- 销售人员（增加或删除销售经理、销售代表，和/或处理交易的所属关系）
- X 交易（定义新的销售交易）
- 报告（生成报告）
- 退出（结束 ABCSALES 系统运行）

销售人员

当选择销售人员（Salespers）菜单命令时，将出现一个名为销售人员的窗口，包含下列内容：

（1）现有的所有销售代表名单。清单中每一个输入项为 20 个字符的域，记录当前销售代表的名字。

（2）一个由 20 个字符组成的域，记录当前销售经理的名字。

（3）在销售人员窗口右上角附近有一组共五个按钮，可以提供如下功能：

（a）选择 OK 按钮返回 ABCSALES 系统。SM 域中必须有一个名字。

（b）选择 ADD 按钮增加一个新的 SR。当选择 ADD 按钮后，出现一个名为 SRadd 的窗口。该窗口包括一个含有 20 个字符的域用于填写新 SR 的名字。与当前其他人员相比较而言新的 SR 名字必须是惟一的。SRadd 窗口包括两个按钮：OK 执行添加、CANCEL 则取消执行，不论 OK 还是 CANCEL 返回 Salespers 窗口后都会更新 SR 清单。

（c）XFERXACT 将属于所选的销售人员的交易（在数据库 SALEs 中的）所有权转移到其他销售人员的名下，以确保交易所有权的连续性。当按下该按钮后，提示用户从当前的 SR 和 SM 名单中选择销售人员，之后返回 Salespers 窗口。

（d）DELETE 删除选择的销售人员（SR 或 SM）。当销售人员在 SALES 数据库中还有属于他或她的交易时不能删除之。用户返回 Salespers 窗口，该窗口包含更新 SR 和 SM 名单的功能。

（e）NEWSM 使一个选定的 SR 就职 SM。如果已存在一个 SM，按下此按钮时，当前的 SM 就移至 SR 名单中。用户返回 Salespers 窗口，该窗口包含更新 SR 和 SM 名单的功能。

X 动作

当选中 Xaction 命令时，将显示一个名为 Xaction 的窗口。该窗口中包括一个程序自动生成的交易标识符（8 位整数，左边用零补齐）来惟一标识该交易，另外还有如下的数据域，让用户来定义交易的细节：

（1）交易日期（月/日/年，月、日、年各填写两位，如果月、日、年是一位数字，左边用零补齐）

（2）租赁或服务终止的交易标识号（如果交易是要终止租赁或服务，用户要键入启动该租赁或服务时的交易标识号；对于该终止，除了日期外屏幕上其他数据域都被忽略）

（3）销售人员的名字

（4）产品（本项必须填写，从显示的 6 项产品清单中单击来选择一项产品）

（5）如果所选产品是设备，用户必须选择进入销售或租赁

在 Xaction 的窗口的右上角附近有一组共三个按钮，提供如下功能：

- ENTER 进入交易并返回 ABCSALES 窗口
- CLEAR 重新初始化所有数据域（允许用户重新填写空白的 Xaction 的窗口）
- CANCEL 取消交易输入并返回 ABCSALES 窗口

CLEAR 和 CANCEL 忽略所有的数据域，但 ENTER 在将交易加入 SALES 数据库之前要执行许多有效性检查。如果发现错误，交易不能入库，并且向用户提供适当的信息或窗口。下面是执行 ENTER 时要进行的检查：

（1）日期是有效的，并且不会比今天的日期晚。

（2）对于终止交易，其交易标识号必须是 SALES 数据库中存在的，并且该交易必须是租赁或服务合同。

（3）销售人员域中必须包含在 PERS 数据库中存在的当前销售人员（SR 或 SM）的姓名。

报告

当选中 Report 命令时，将显示一个名为 Report 的窗口。该窗口中包括每个销售人员的如下信息：

（a）销售人员的姓名和头衔（SR 或 SM）

（b）月基本工资

（c）配额内的设备收入（出售的和新租赁的）

（d）配额之外的设备收入（出售的和新租赁的）

（e）配额内服务合同的收入

（f）配额之外的服务合同的收入

（g）设备租赁的剩余收入

（h）服务合同的剩余收入

（i）对于 SM，来自 SR 的收入

（j）奖金

（k）月总收入

退出

通过选择下列两者之一退出 ABCSALES 系统：

（a）ABCSALES 命令菜单的退出（Exit）命令，或

（b）ABCSALES 控制菜单中的关闭（Close）命令。

评审功能设计说明练习的解决方案

下列的文字注释是 10 名测试练习者进行评审后的综合结果，共用了 30 分钟时间。

ABCSALES 的功能设计说明

开始显示：ABC 公司

ABC 公司销售 6 种基本产品：

（P1）X 型寻呼机（购买价 80 美元，出租 5 美元/月）

（P2）Y 型寻呼机（购买价 120 美元，出租 10 美元/月）

（P3）本地寻呼服务（12 美元/月/台，不限制使用次数）

（P4）G 型蜂窝电话（购买价 500 美元，出租 35 美元/月）

（P5）H 型蜂窝电话（购买价 270 美元，出租 20 美元/月）

（P6）本地蜂窝电话服务（60 美元/月/台，不限制使用次数）

对于租用机器消费者必须使用 ABC 公司的服务，购买机器则不用。ABC 公司的服务业务并不严格局限于购买和租赁 ABC 设备的客户。

ABC 公司有两种销售人员：

- 销售代表（SR）：销售所有产品
- 销售经理（SM）：销售所有产品并管理销售代表

ABC 公司销售委托方案

销售代表（SR）

一个销售代表每月保底薪金为$2200，设备配额为 20 部电话、10 台寻呼机，服务性配额为 20 个新的客户服务合同（任何寻呼机和电话的客户服务都可以）。不论是购买还是租赁性质都可以满足设备配额要求。

在设备配额之内，每一台（个）设备，销售代表得到销售额 4%的佣金，超过配额之外，每台设备佣金为 7%。每介绍谈成一个新的租赁客户，在配额内可得报酬 13%，在配额外可得报酬 18%；如果客户在后续的月中继续租用设备，销售代表每月可得报酬 10%。

在服务配额之内，对于每个新的服务单元销售代表得到每月服务费的 10%，超过配额之外，每介绍一个新的服务性客户，销售代表得到每月服务费的 15%；如果客户继续接受寻呼或通话服务，销售代表可得客户服务费的 8%。每个销售代表每月最多可以得到 $700 的剩余服

务佣金。如果他或她每月业绩超过服务配额的 150%，销售代表可得到$300 的奖金。

本段难于阅读和追踪，做成表格或许有帮助。

销售经理（SM）

一个销售经理每月保底薪金为$2900，设备任务额为 10 部电话、5 台寻呼机，没有服务性任务额。

销售经理的设备（出售和租赁的剩余）佣金与销售代表相同。除此之外，销售经理可获得佣金为所有销售代表出售和租赁设备总产值的 1%加上销售代表租赁剩余部分的 1%。销售经理可获得他或她所得新服务合同额的 14%。得到所有销售代表服务合同剩余部分要价的 2%。销售经理没有剩余的服务费提成，如果他管辖的所有销售代表都完成了月配额，销售经理得奖金$400。

一般程序功能

ABCSALES 是一个独立的功能齐全的系统，包括如下的性能和功能：

（1）ABCSALES 系统可读/写两个数据库 SALES 和 PERS，SALES 保留了每笔销售交易的完整历史记录（销售以及启动和终止租赁、服务的合同）。PERS 包括目前的销售经理和销售代表的姓名。（SALES 和 PERS 开始为空，其修改和维护由 ABCSALES 来进行）。

（2）对于每一笔销售交易，ABCSALES 接收用户输入以完整地定义本次交易，并相应地更新 SALES 数据库。

（3）ABCSALES 在数据库 PERS 中填加、删除销售代表和销售经理，并处理 SALES 数据库中的交易的所属关系（确保所有的交易都有连续的所属关系）。

（4）根据用户的命令，ABCSALES 系统可基于 SALES 和 PERS 数据库生成报告，根据销售代理计划的规定，显示所有现有销售代表的总收入和销售经理当前月的销售收入。

详细的功能描述

如果进行详述，用图表怎么样？

ABCSALES 是一个单用户系统，运行在 IBM-PC 机上，Windows 操作系统环境。

运行 ABCSALES 系统用鼠标双击程序管理窗口的 ABCSALES 图标，随后 ABCSALES 窗口就会显现出来。

ABCSALES 系统菜单条包括两种菜单：标准控制菜单（在 ABCSALES 系统窗体左上角）（用来做什么的？）和命令菜单。下列命令通过命令菜单可以使用：

- 销售人员（增加或删除销售经理、销售代表，和/或处理交易的所属关系）
- X 交易（定义新的销售交易）
- 报告（生成报告）
- 退出（结束 ABCSALES 系统运行

销售人员

当选择的销售人员（Salespers）菜单命令是，将出现一个名为销售人员的窗口，包含下列

内容：

（1）现有的所有销售代表名单。清单中每一个输入项为 20 个字符的域，记录当前销售代表的名字。

可以有多少个销售代表（SR）？如果用户项的字符数多于 20 个会怎么样？

（2）一个由 20 个字符组成的域，记录当前销售经理的名字。

可以有多少个销售经理（SM）？如果用户项的字符数多于 20 个会怎么样？

（3）在销售人员窗口右上角附近有一组共五个按钮，可以提供如下功能：

（a）选择 OK 按钮返回 ABCSALES 系统。SM 域中必须有一个名字。

如果 SM 域为空会如何？

（b）选择 ADD 按钮增加一个新的 SR。当选择 ADD 按钮后，出现一个名为 SRadd 的窗口。该窗口包括一个含有 20 个字符的域用于填写新 SR 的名字。与当前其他人员相比较而言新的 SR 名字必须是惟一的。SRadd 窗口包括两个按钮：OK 执行添加、CANCEL 则取消执行，不论 OK 还是 CANCEL 返回 Salespers 窗口后都会更新 SR 清单。

如果名字不惟一会如何？

（c）XFERXACT 将属于所选的销售人员的交易（在数据库 SALES 中的）所有权转移到其他销售人员的名下，以确保交易所有权的连续性。当按下该按钮后，提示用户从当前的 SR 和 SM 名单中选择销售人员，之后返回 Salespers 窗口。

如果“来自”和“转到”一样会如何？

（d）DELETE 删除选择的销售人员（SR 或 SM）。当销售人员在 SALES 数据库中还有属于他或她的交易时不能删除。用户返回 Salespers 窗口，该窗口包含更新 SR 和 SM 名单的功能。

（e）NEWSM 使一个选定的 SR 就职 SM。如果已存在一个 SM，按下此按钮时，当前的 SM 就移至 SR 名单中。用户返回 Salespers 窗口，该窗口包含更新 SR 和 SM 名单的功能。

没有 CANCEL 按钮？

只有一个 SM？

X 动作

当选中 Xaction 命令时，将显示一个名为 Xaction 的窗口。该窗口中包括一个程序自动生成的交易标识符（8 位整数，左边用零补齐）来惟一标识该交易，另外还有如下的数据域，让用户来定义交易的细节：

（1）交易日期（月/日/年，月、日、年各填写两位，如果月、日、年是一位数字，左边用零补齐）

可以空出左边吗？

（2）租赁或服务终止的交易标识号（如果交易为终止租赁或服务，用户将键入启动该租赁或服务时的交易标识号；对于该终止，除了日期外屏幕上其他数据域都被忽略）

（3）销售人员的名字

（4）产品（本项必须填写。从显示的 6 项产品清单中单击来选择一项产品）

（5）如果所选产品是设备，用户必须选择进入销售或租赁。

“must”是如何强制执行的？

在 Xaction 的窗口的右上角附近有一组共三个按钮，提供如下功能：

- ENTER 进入交易并返回 ABCSALES 窗口
- CLEAR 重新初始化所有数据域（允许用户重新填写空白的 Xaction 的窗口）
- CANCEL 取消交易输入并返回 ABCSALES 窗口

这是这些操作的准确顺序吗？

CLEAR 和 CANCEL 忽略所有的数据域，但 ENTER 在将交易加入 SALES 数据库之前要执行许多有效性检查。如果发现错误，交易不能入库，并且向用户提供适当的信息或窗口。下面是执行 ENTER 时要进行的检查：

（1）日期是有效的，并且不会比今天的日期晚。

每天的日期是如何定义的？

（2）对于终止交易，其交易标识号必须是 SALES 数据库中存在的，并且该交易必须是租赁或服务合同。

（3）销售人员域中必须包含在 PERS 数据库中存在的当前销售人员（SR 或 SM）的姓名。

报告

当选中 Report 命令时，将显示一个名为 Report 的窗口。该窗口中包括每个销售人员的如下信息：

（a）销售人员的姓名和头衔（SR 或 SM）

（b）月基本工资

（c）配额内的设备收入（出售的和新租赁的）

（d）配额之外的设备收入（出售的和新租赁的）

（e）配额内服务合同的收入

（f）配额之外的服务合同的收入

（g）设备租赁的剩余收入

（h）服务合同的剩余收入

（i）对于 SM，来自 SR 的收入

（j）奖金

（k）月总收入

退出

通过选择下列两者之一退出 ABCSALES 系统：

（a）ABCSALES 命令菜单的退出（Exit）命令，或

（b）ABCSALES 控制菜单中的关闭（Close）命令。

一般的观点和疑问：

- 在本说明中错误处理被忽略了。
- 如果销售人员没有完成他或她的配额指标怎么办？
- 在说明中，通篇都是对佣金算法的文字叙述，如果使用如图片或公式、表格、图表等表达方式将会加强我们对算法的理解。
- 在功能设计说明书中贯穿了计算方法的描述，试图解释如何完成计算，而实际上解释得并不清楚。
- 在第三页中“相应地更新 SALES 数据库”，“相应地”是什么意思？
- 另一个例子是“按照用户的要求”，针对本系统用户的要求是指什么？
- 在第四页中，说明书中提到一个“小”窗口，多“小”算是小窗口？
- 没有充分地描述控制菜单。
- 像 XFERXACT，在操作过程中关机或机器断电怎么办？

关于规格说明的更多的疑问、问题

（1）ABCSALES 系统似乎约定了 SM 的个数为一。如果 SR 清单很大，在 Salespers 窗口中放不下，该窗口需要滚动吗？多少个 SR 可以向一个 SM 报告？同时，Salespers 命令不允许指定一个当前 SR 队伍之外的人员担任 SM。

（2）设备指标额是否填在设备出售或租赁的实际位置？如果是，当销售人员销售（a）寻呼机超过电话或（b）租赁超过出售时有没有什么奖励？

（3）对于代理预付费用不足月的情况，租赁或服务合同是否有效？

（4）SM 自己创造的服务合同的产值的剩余部分是什么？

（5）SM 是否会从 SR 交易的新服务合同中提取佣金？

（6）在使用 Xaction 命令终止租赁或服务合同时，没有检查以确认（a）将要终止的合同有效吗？（b）正在终止合同的销售人员是否有该合同的所有权？

（7）没有能力强迫租赁 ABC 设备必须使用 ABC 的服务。

（8）注意报告结构或交易所有权的修改，在任何一个数据库中都没有提到维修历史记录。对于在月的当中更换 SM 或交易所有权，ABCSALES 是否用分离和预付佣金的方法来表达？

（9）在 Salespers 命令中：

（a）说明书中没有说明当 SM 域为空时按下 OK 按钮系统会怎么样？

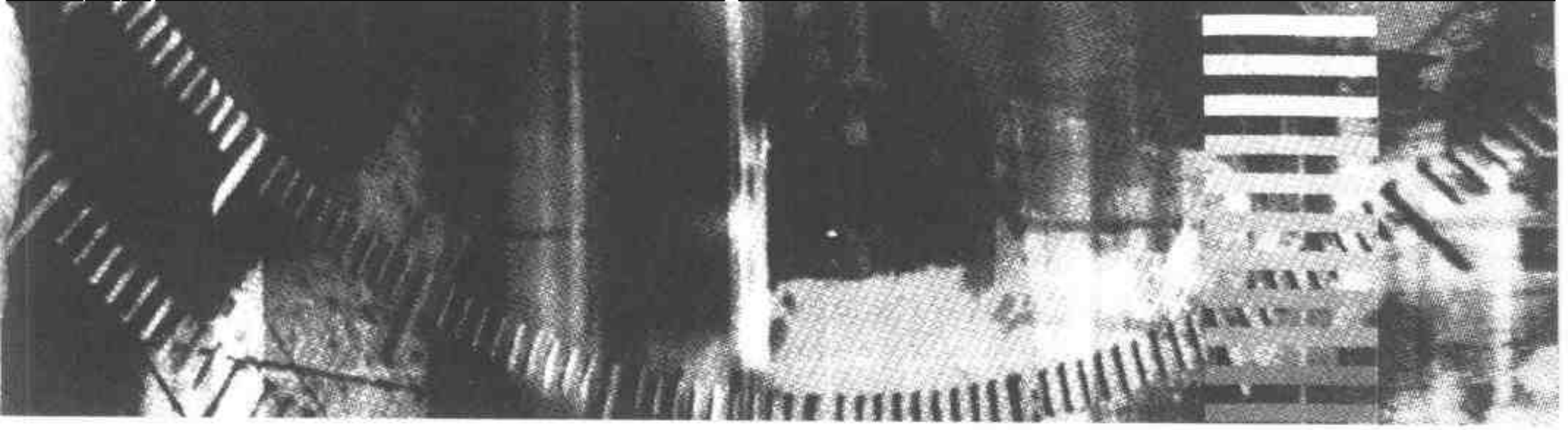
（b）ADD 功能提供用户一个取消或执行 ADD 的可能，然而，在 XFERXACT、DELETE 和 NEWSM 功能中没有这项。

（c）当检查到用户错误时所有功能中都缺乏关于程序行为的确切描述。

（d）没有指明由 20 个字符组成的销售人员的名字的精确格式或语法。

（10）Report 命令没有说明报告在哪打印，报告格式、包括的域的宽度也没有指出。

（11）Xaction 命令中，当按下 ENTER 按钮进行有效性检查时，没有定义任何错误信息发布的规则。



附录D

确认练习（答案）

针对第 8 章给出的练习，下面是一个可能的答案。

等价类划分练习解答

外部输入情况	有效等价类	无效等价类
球场记录的个数	4	<1，>8
球场记录中的球场名称	非空	空
球场记录中的规定击球次数	整数	非整数
球场记录中的整数规定击球次数	4	<3，>5
高尔夫球员的人数	20	<2，>400
每个球员的球员记录数	=球场数	不等于球场数
球员记录中的球场名称	非空	空
球员记录中的非空球场名称	球场记录中已定义的	球场记录中未定义的
球员记录中的球员姓名	非空	空
球员记录中的击球次数	非零数字	0，非数字
分界符记录	第一列非空	第一列为空

边界值分析练习答案

输入条件

（1）空的输入文件
（2）输入文件中没有球场记录
（3）1 个球场记录
（4）8 个球场记录

（5）9 个球场记录
（6）球场记录中空的球场名
（7）球场记录中的由 1 个字符组成的球场名
（8）球场记录中的由 18 个字符组成的球场名
（9）对于第 1 个球洞（1）和最后一个球洞（18）：
　（a）球场记录中的规定击球次数非整数
　（b）球场记录中的规定击球次数=2
　（c）球场记录中的规定击球次数=3
　（d）球场记录中的规定击球次数=5
　（e）球场记录中的规定击球次数=6
（10）分界符记录中第 1 列为空
（11）分界符记录中第 1 列非空
（12）分界符记录中第 2 至 60 列为空
（13）分界符记录中第 2 至 60 列非空
（14）输入文件中没有球员记录
（15）1 个球员
（16）2 个球员
（17）400 个球员
（18）401 个球员
（19）球员记录中的球场名称在球场记录中已定义
（20）球员记录中的球场名称在球场记录中未定义
（21）某个球员的球员记录数比球场数少 1
（22）某个球员的球员记录数与球场数相等
（23）某个球员的球员记录数比球场数多 1
（24）某个高尔夫球员在同一个球场中有两个以上球员记录
（25）球员记录中某个球员的名字为一个字符
（26）球员记录中某个球员的名字为 18 个字符
（27）对于第 1 个球洞（1）和最后一个球洞（18）：
　（a）击球次数不是数字
　（b）击球次数=0
　（c）击球次数=1
　（d）击球次数=9

输出条件

注意：部分输出情况会根据输入情况不同而做相应的变化（例如，#14，16，17），不需要进行额外的测试。

（28）所有的高尔夫球员总得分相同

（29）所有的高尔夫球员总得分各不相同

（30）某个高尔夫球员得到了最低分（0）

（31）某个高尔夫球员得到最高分（6 乘以球场数的 18 倍）

（32）每个球场中，该球场的所有球员得分相同

（33）每个球场中，该球场的所有球员得分各不相同

（34）每个球场中，某个球员得了该球场中的最低分（0）

（35）每个球场中，某个球员得到该球场中的最高分（6 乘以 18）

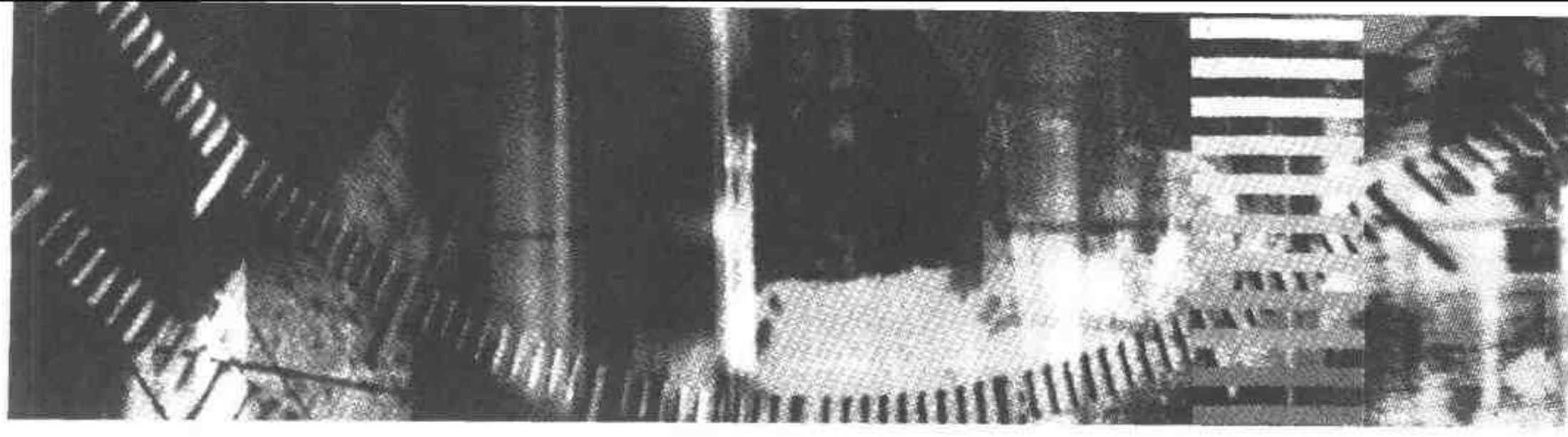
（36）部分但不是全部的球员得到的总分相同（检查排序的正确性）

（37）球员个数正好占报告页中的一页，没有剩余的空间给其他球员（确认没有打印多余的页）。

（38）球员的个数比测试（37）中多一个。

（39）某个球员的名字在排列序列中具有可能的最低值。

（40）某个球员的名字在排列序列中具有可能的最高值。



附录E

参考书目（包括软件测试工具一览表）

1. Software Testing Techniques, Boris Beizer

（Van Nostrand Reinhold Company, Inc.出版，1983 年，1990 年第二版，508 页。）
该书强调单元级测试并介绍对个体程序员最适用的覆盖技术。Boris 解释说：

“不管用于测试一个系统的方法多高明，文件多完整，结构如何构建，还有开发计划，项目评论……不管整套的技术多么先进，如果没有对单元级软件、单个的子程序进行正确测试的话，所有的一切都将归于零，项目也会失败。”

介绍的许多技术是基于作者为一家电信软件制造商担任测试和质量保证部主任时的一手经验。

评论：Boris Beizer 是一位资深的软件开发专家，他的作品反映出他丰富的经验。该书不乏各种先进的测试技术，可以满足广大经验丰富的程序员、逻辑设计人员和计算机设计人员的需求，他们有必要完全熟悉详细的单元测试技术。该书不是为软件测试专家撰写的软件测试入门，也不是为非程序员编写的，那不是作者写这本书的初衷。

2. The Complete Guide to Software Testing, Bill Hetzel

（John Wiley 出版，1984 年，1988 年第二版，280 页。）

该书阐述怎样有效地进行软件测试以及怎样对这项工作进行有效的管理。该书的内容还包括怎样定义测试、怎样度量测试以及如何确保测试的有效性。该书是为软件专业人员或管理人员撰写的，不要求读者事先拥有这方面的知识。它包括测试方法概述以及丰富的参考书目。

评论：该书不失为测试方面一本扎实、实用且基础性强的好书。1984 年首次出版，并没有收录一些最新出现的技术，如 GUI 测试、测试自动化和面向对象的测试等。但该书对许多软件测试理论做了精辟的阐述，而这些基础理论至今未变，因此该书仍为首选的三本书之列。

3. Managing the Software Process, Watts S. Humphrey

（Addison-Wesley Publishing Company 出版，1989 年，486 页。）

该书为改进软件开发过程提供了实用性指南。它为程序管理人员和程序员阐述了如何确定当前过程的质量、如何对过程进行改进以及从何处开始。该书是卡内基-梅隆大学软件工程研究所系列从书中的一本，介绍评估软件组织的技术以及通过评估发现最需要改进的管理问题

评论：我们知道你在想什么，这本书不是谈软件测试的，那么为什么将它列在这儿呢？理由是，测试是一个过程，而这本里程碑似的书详细阐述了管理任何软件过程的步骤，包括测试。此外，该书还包括一些很好的测试和审查方面的材料。不仅如此，关心软件测试的人都应该了解软件工程研究所（SEI）和该书详细介绍的五个层次的过程成熟度。

4. The Art of Software Testing, Glenford J. Myers

（John Wiley 出版，1979 年，177 页。）

不少人认为，Glenford Myers 撰写的这本书是迄今为止软件测试方面的第一本好书。它首次从实用而不是从理论的角度阐述了软件测试的目的和本质。它包括测试技术以及测试方面重要的心理学问题。正如前言所述，该书有三类主要的读者，即专业程序员、项目管理人员和程序专业的学生。自 1979 年起，出现了专业软件测试人员，因此也应该将他们列入上述名单。

评论：这是一本经典著作，也是第一本，常被人们引用。就凭这，尽管它出版有些年头了，仍值得一读。将这本书辅之以一本当代的著作，就可以获知经典和当代测试领域的全貌了。

测试方面的其他书籍

Abelow, D. (1993). "Wake up! You've Entered the Transition Zone," *Computer Language*, March.

Ackerman, A.F. (1992). "Usage Testing," *Software Testing Analysis & Review (STAR) Conference Proceedings*.

Ackerman, A.F. (1994). "Constructing and Using Operational Profiles," *Software Testing Analysis & Review (STAR) Conference Proceedings* 1993.

Beizer, B.(1984). *Software System Testing and Quality Assurance*. Van Nostrand Reinhold.

Bender, D.(1993). "Writing Testable Requirements," *Software Testing Analysis & Review (STAR) Conference Proceedings*

Beresoff, E.H., Henderson, V.D. and Siegel, S.G. (1980). "Software Configuration Management: A Tutorial," *IEEE Tutorial: Software Configuration Management*, IEEE Cat No. EHO 169-3, 27 October, pp. 24-32.

Bogart, T.G. (1993). "Fundamentals of Software Testing," *Software Testing Analysis & Review (STAR) Conference Proceedings*.

Brooks, F.P. (1975). *The Mythical Man-Month*. Reading, MA: Addison-Wesley.

Craig, R. (1992). *Software Testing Practices Survey*, Software Quality Engineering.

Fagan, M.E. (1976). "Design and Code Inspection to Reduce Errors in Program Development," *IBM Systems Journal*, 15(3).

Freedman, D.P. and Weinberg, G.M. (1990). *Handbook of Walkthroughs, Inspections, and Technical Reviews*. New York: Dorset House.

Gilb, T. and Graham, D.(1993). *Software Inspection*. Wokingham: Addison-Wesley.

Hetzel, W. (1973). *Program Test Methods*. Englewood Cliffs, N.J.: Prentice-Hall.

Hetzel, W. (1993). *Making Software Measurement Work*. QED Publishing Group.

Hetzel, W. and Craig, R.(1990). *Software Measures and Practices Benchmark Study*. Research Reports TR-900 and TR-903, Software Quality Engineering.

Kit, E. (1986). *Testing C Compilers*, Computer Standards Conference.

Kit, E. (1986). *State of the Art, C Compiler Testing*. Tandem Computers Technical Report.

Lewis, R.O. (1992). *Independent Verification and Validation*. John Wiley.

McConnell. S. (1993). "From Anarchy to Optimizing," *Software Development*, July.

Migdoll, M. (1993). "Improving Test Practices," *Software Testing Analysis & Review (STAR) Conference Proceedings*.

Myers, G.J. (1976). *Software Reliability, Principles and Practices*. John Wiley.

Myers, G.J., Brad A. and Rosson, Mary Beth (1992). "Survey on User Interface Programming," *CHI'92 Conference Proceedings*. ACM Conference on Human Factors in Computing Systems.

Norman, S. (1993). "Testing GUIs is a sticky business," *Software Testing, Analysis and Review (STAR) Conference Proceedings*.

Rosenbaum, S. (1993). "Alternative Methods for Usability Testing," *Software Testing Analysis & Review (STAR) Conference Proceedings*.

Tener, M. (1993). "Testing in the GUI and Client/Server World," *IBM OS/2 Developer*, Winter.

1994 IEEE Software Engineering Standards Collection, IEEE Computer Society, 1994.

软件测试工具一览表

Testing Tools Reference Guide, Software Quality Engineering, 800 423-8378 or 904 268-8639.

描述了数百种商用软件测试工具。

Software Test Technologies Report, Software Technology Support Center, Hill Air Force Base, Tel.(1) 801 777 8057.

软件测试术语概括，工具分类，以及评估和挑选软件测试工具的方法。

Software TestWorld CD-ROM, Edward Kit, Software Development Technologies, Tel. (1) 408 252 8306, fax (1) 408 252 8307.

一种互动 CD-ROM 应用，包含重要的软件测试产品和服务，有工具、标准、培训、咨询、研讨会和文献以及所选工具的互动演示。

Software Management Technology Reference Guide, Software Maintenance New, Inc. Tel. (1) 415 969 5522.

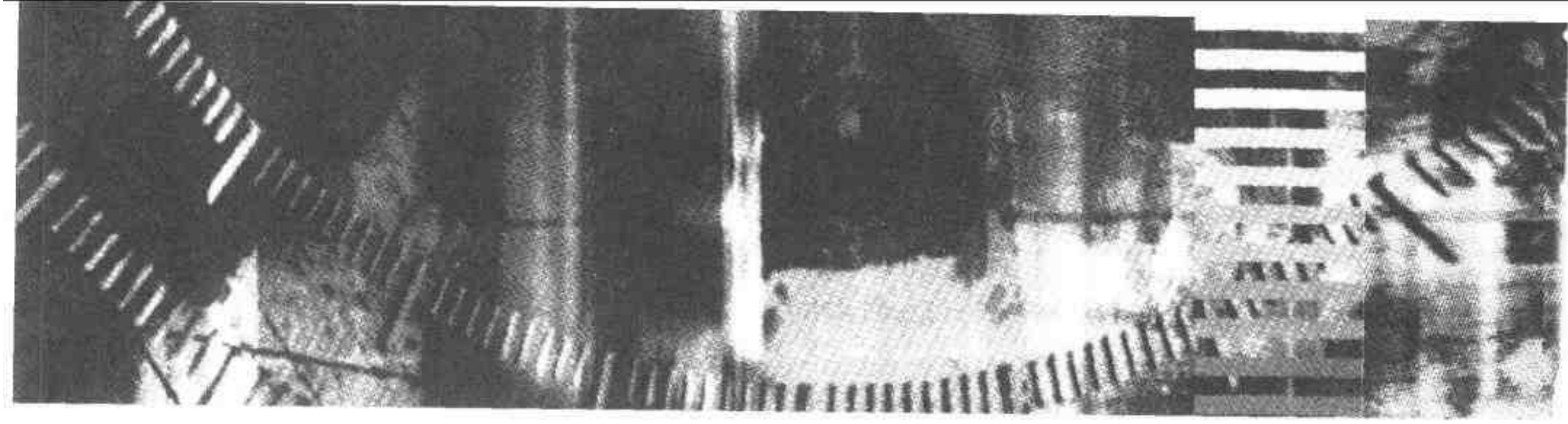
描述当前可获得的支持软件管理的软件和服务。包括测试问题、工具和资源方面的章节。

The CAST Report, Dorothy Graham and Paul Herzlich, Cambridge Market Intelligence Ltd, Letts of London House, Parkgate Road, London SW11 4NQ, Tel. (44) 171 924 7117, fax (44)171 403 6729.

支持软件测试包括静态分析、故障排除和测试管理的可以获得的工具。

Ovum Evaluates: Software Testing Tools, Ovum Ltd., Tel. (44) 171 255 2670.

26 种用于使系统自动化和验收测试的重要工具的详细评估。



附录F

信息资源：会议、期刊、通讯、DOD 规范

测试会议

International Software Testing Analysis & Review (STAR)

STAR 由一般大会、分组会议、工具展览、以及会前讲座组成，STAR 的小册子上写着：

“STAR 的对象是那些在组织中负责测试和评价活动的人们……其宗旨是促进和帮助实践者和管理者，使他们在软件测试方面的工作更加富有成效。

一个典型的美国会议有 600 多人参加，他们来自美国各地，还有其他国家，包括英国、加拿大、墨西哥、法国、挪威、德国和瑞士等。与在美国召开的 STAR 会议相衔接的还有欧洲会议（EuroSTAR），它由英国计算机协会软件测试专业组主办。

Software Quality Engineering 的联系电话

Tel. (1) 904 268 8639 (US)

International Conference & Exposition on Testing Computer Software

该会议包括会前全天的研讨班、一般大会、分组会议和工具展览。

USPDI 的联系方式：

1734 Elton Rd., Suite 221

Silver Springs MD 20903

Tel. (1) 301 445 4400

International Software Quality Week (SQW)

SQW 由会前讲座、大会、分组会议、工具销售展以及讨论班组成。会议册子上写着：

"Software Quality Week 集中关注软件测试技术、质量控制、风险管理、软件安全性及测试自动化方面的进展

Software Research, Inc. 的联系电话

Tel. (1) 415 957 1441

International Conference on Software Testing

该会议是一个针对实践者的活动，会议上测试/质量控制专业人士共同分享在整个系统开发生命周期中的改进测试和质量控制的方法，实践者分享他们的测试过程、测试管理和度量策略以及最有效的测试技术，测试工具销售展览与会议同时进行，使与会者能看到展示的工具，并和工具支持人员进行面对面的交流

质量保障研究所的联系方式：

Tel. (1) 407 363 1111

Fax. (1) 407 363 1112

Quality Assurance Institute, Inc.

7575 Dr. Phillips Boulevard

Suite 350

Orlando, FL 32819

有关测试的期刊和通讯

- *Testing Techniques Newsletter*: Software Research, Inc., 625 Third St., San Francisco, CA 94107-1999. (Free)
- *Software Testing, Verification and Reliability*: (ISSN 0960-0833): a quarterly journal published by Wiley & Sons Ltd, Baffins Lane, Chichester, West Sussex, PO191UD, UK.
- *Software Quality Management Magazine*: a quarterly magazine available through Software Quality Engineering, 3000-2 Hartley Road, Jacksonville, FL 32257. Tel. (1) 904 268 8639.
- *Software Quality World* (ISSN 1042-9255): a semi-monthly newsletter published by ProQual, Inc., P.O.Box 337, Medfield, MA 02052-0003. Tel. (1) 508 359 7273.
- *Software Management News*, B10-Suite 237, 4546 El Camino Real, Los Altos, CA 94022. Tel. (1) 415 969 5522.

SEI 订户程序

SEI 订户程序有超过 1000 的来自工业、政府和研究部门的个人订户，其目的是使得其成员保持对 SEI 的事态和最新研究成果的了解，出版物包括 BRIDGE、SEI 周刊和 SEI 技术评论，其中 SEI 技术评论汇集了 SEI 在一年内的关键技术工作，订户对所有 SEI 报告和年度专题论文集均可享受 10%的折扣。

联系客户关系部，请致电：（1）412 268 5800

军方/DOD 的有关规范

DOD-STD-2167A. *Defense System Software Development*, February, 1988. 包括下列标准：

- MIL-STD-499A(USAF). *Engineering Management*, May, 1974.
- MIL-STD-1521B(USAF). *Technical Review and Audits for Systems, Equipment, and Computer Software*, June, 1985.
- MIL-STD-480B. *Configuration Control – Engineering Changes, Deviations, and Waivers*, July, 1988.
- MIL-STD-481A. *Configuration Control – Engineering Changes, Deviations, and Waivers (Short Form)*, October, 1972.
- MIL-STD-482. *Configuration Status Accounting Data Elements and Related Features*, December, 1970.
- MIL-STD-490A. *Specification Practices*, June, 1985.
- DOD-STD-2168. *Defense System Software Quality Program*, April, 1988.

通过以下方式可得到联邦和军方的标准、规范和手册的复印件：

DOD Single Stock Point

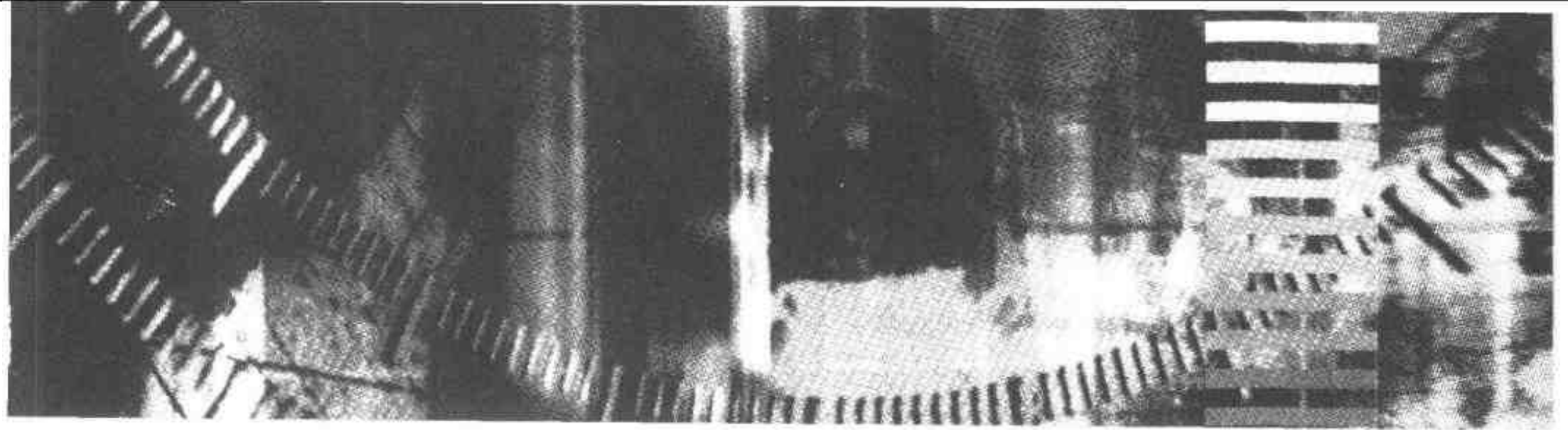
Commanding Officer

US Naval Publications and Forms Center (Attn: NPFC 1032)

5801 Tabor Avenue

Philadelphia, PA 19120

Tel. (1) 215 697 2667



附录G

专用工具和工具选择

工具选择

以下工具评价过程描述了软件开发技术公司（一个领先的测试工具咨询机构）在编译 Software TestWorld™ 中所使用的方法。Software TestWorld™ 是一个当前可用的测试工具的 CD-ROM 最新版，这些方法与一个精心选择的过程联系起来，为每一个试图获得合适工具的人提供了重要的指南。

包含在 Software TestWorld 中的选择过程是严格以过去 15 年中不断发展的需求为基础的，它是一个“整体产品”方法，包括了对销售商的测试工具的评价、管理小组、商业实践、培训和技术支持，以下是选择过程的概要介绍：

- 通过邮件或电子邮件得到工具的印刷品和手册，对它们进行评阅。
- 浏览一些主要的贸易杂志，注意新产品方面的广告和文章。
- 参加重要的会议，观看工具展览，并与那些展示新产品的工具销售商交谈。
- 考查产品的详细资料，包括销售商的“白皮书”，并将信息保存在工具数据库中。
- 利用软件测试工具目录（见附录 E），对可提供的支持软件测试的工具有个概要了解。
- 与销售商联系，要求得到一个工具演示版，运行它并测试它，其中存在的缺陷之多、质量之差常常令人吃惊，究竟有人对它进行过测试吗？
- 如果产品通过了以上的质量测试，就向销售商要产品评价包。可以与之做生意的销售商都有产品评价程序。有些产品销售商提供辅导，有些提供产品，但一般地说，所有销售商均提供足够的工具和材料，可以对产品进行技术评估。在这一阶段，SDT 允许在实验室机器上安装产品，并对这些过程和真正的用户文件进行评估。此外，不管是否需要帮助，SDT 还拨打技术支持热线对支持过程的有效性进行评估。
- 可能的话，亲临工具销售商的总部。会见管理人员，检查设备，并且讨论价格、客户基础、营销战略、技术支持、测试参考、未来产品和培训。
- 符合 SDT 要求的销售商可以开始与 SDT 公司合作，参加 Software TestWorld 新版本的

销售。

联系方式：

Software Development Technologies

Tel.(1) 408 297 1911, fax: (1) 408 297 1993

具体工具

下面给出的工具清单并不十分详尽，主要工作是给出了对测试工具以及它们能力的评价。

软件开发技术

软件测试工具一览表——7.5.5 版

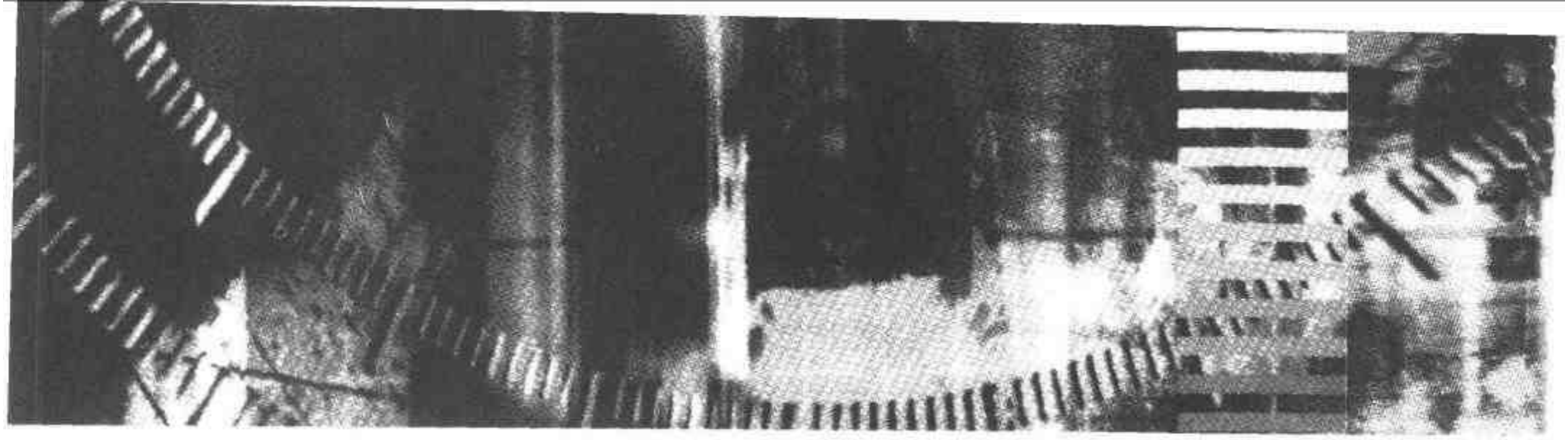
类　型	工　具	销 售 商	Java	Win	UNIX	MF
测试计划、管理和控制						
进度估计	1 Knowledge PLAN	Software Productivity Research	√	H	√	√
	2 SLIM	QSM	√	H	√	√
需求管理	3 icCONCEPT RTM	Integrated Chipware, Inc.	√	H	H	√
	4 DOORS	QSS Inc.	√	H	√	√
	5 Requisite Pro	Rational Software Corp	√	H	√	√
测试管理	6 QA Director	Compuware		H	H	√
	7 SQA Manager	Rational Software Corp		H		
	8 TestDirecter	Mercury Interactive Corp		H		
配置管理	9 CA-Endevor	Computer Associates		√	H	H
	10 Changeman	Serena International				H
	11 ClearCase	Rational Software Corp		H	H	
	12 Continuus/CM	Continuus Software			H	
	13 PVCS Version Manager	Intersolv		H		√
问题管理	14 Continuus/PT	Continuus Software			H	
	15 Clear DDTS	Rational Software Corp	√	√	H	√
	16 PVCS Tracker	Intersolv		H		√
评审和审查						
技术评审管理	17 ReviewPro	Software Development Technologies	√	H	H	√
复杂性分析	18 Visual Quality	McCabe & Associates		H	H	√
代码理解	19 AcquaProva	CenterLine Development Systems			H	
	20 Visual Reengineering	McCabe & Associates		H	H	√
测试设计及开发						
数据库生成	21 CA-Datamacs/II	Computer Associates				H
	22 TestBytes	Mercury Interactive Corp	√	H	√	√
测试执行及评价						
单元测试	23 JaveSpec	Sun Microsystems, Inc.	H	√	√	√

（续）

类　型	工　具	销 售 商	Java	Win	UNIX	MF
Java 捕获/回放	24 JavaStar	Sun Microsystems, Inc.	H	√	√	√
本机捕获/回放	25 QAHiperstation	Compuware		H		H
	26 QA Partner	Segue Software Inc.	√	H	H	H
	27 QA Playback	Compuware		H		H
	28 QC Replay	CenterLine Development Systems			H	
	29 SQA Robot	Rational Software Corp		H		
	30 WinRunner	Mercury Interactive Corp		H		√
	31 XRunner	Mercury Interactive Corp			H	
非交互捕获/回放	32 Ferret	Azor, Inc.	√	√	√	√
	33 TestRunner/NI	Qronus Interactive	√	√	√	√
Web 捕获/回放	34 AutoTester Web	AutoTester, Inc.		H		
	35 Silk	Seque Software, Inc.		H		
	36 SQA Robot	Rational Software Corp		H		
	37 Web Test	Mercury Interactive		H		
覆盖分析	38 JavaScope	Sun Microsystems, Inc.	H	√	√	√
	39 Visual Testing	McCabe & Associates		H	H	√
内存测试	40 BoundsChecker	Compuware		H		
	41 HeapAgent	MicroQuill Software Pub Inc.		H		
	42 Purify	Rational Software Corp		H	H	
客户机/服务器	43 Automated Test Facility	Teradyne, Inc.		H		
	44 LoadRunner	Mercury Interactive Corp		H	H	
	45 QualityWorks	Segue Software Inc.		H	H	√
性能/仿真	46 LoadRunner	Mercury Interactive Corp		H	H	
	47 PerformanceStudio	Rational Software Corp			H	
	48 TPNS	IBM				H

图例：H=主机的　√=可行的

注：上表中列在评审和审查中的工具也可用于测试计划。



附录H

改进实施示范清单

下面的实施改进表摘自参加测试课程的代表编辑的改进表，该测试课程是形成本书的基础，课程学员来自不同规模、不同成熟层次的组织。

术语的标准化：

- 使用 IEEE/ANSI 定义
- 使用验证和确认
- 使用测试件

产品需求规格说明：

- 力求取得完整、可测试的规格说明
- 为验证需求准备审查单
- 对当前的需求开始评审
- 为改进测试覆盖，书写一份规格说明/需求指南

测试工具：

- 寻找新的自动化工具，包括覆盖工具
- 对工具 WinRunner 进行调研
- 评价工具
- 确定代码复杂性工具

文档：

- 为确认测试计划生成标准化文档
- 生成验证计划和验证报告
- 生成并使用测试计划大纲

- 明确测试交付（测试计划）
- 测试文档库

混杂的工作：

- 开始内部设计规范的验证
- 确定适当的测试点
- 劝说开发者互相测试代码
- 学会并实行对一般和特殊项目的风险分析
- 对可用性测试进行调研
- 对来自公司专业小组成员的支持进行调研
- 对软件工程研究所（SEI）进行调研

测试库：

- 组织
- 生成新的强化测试

软件测试的内部改进

目的：

- 改善交流
- 使测试库自动化，缩短周期时间
- 尽早发现错误
- 改进问题分析
- 生成可重用的测试库

优先考虑的改进：

（1）使用捕获/回放工具使回归测试自动化
（2）测试文档标准化（测试计划、测试过程等）
（3）覆盖分析工具/复杂性工具/白盒测试
（3）尽可能早地开展 QA：（两个子项）
　　（ i ）早期的项目计划
　　（ ii ）早期的技术计划
（3）后期整体项目评价
（4）处理测试小组所有权
（5）提高对 QA 的认识
（6）改进设备管理（资源分配、调度）

软件测试之外优先考虑的改进：

（1）营销和功能规范
（2）版本改进控制和发布控制
（3）开发单元/集成测试（尽早发现故障）
（4）及时对项目和规范进行技术评价
（5）后期项目评价
（6）常见故障数据库
　　（i）通报已发现的故障
　　（ii）QA 介入客户故障
（7）硬件可靠性
（8）第三方项目管理

[General Information]
书名=软件测试过程改进
作者=
页数=196
SS号=11048874
出版日期=

封面
书名
版权
前言
目录
正文