

软测之魂

核心测试设计精解

肖利琼◎著

掌握核心竞争力 成为不可替代的测试精英



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

软测之魂 核心测试设计精解

- ◀ 找bug，就好像挖地雷，每走一步，都得小心谨慎，否则一不小心，一个bug就在你眼皮底下悄悄地溜走了
- ◀ 相信自己，bug是找不完的，bug不是没有了，只是暂时我没发现
- ◀ 坚持！严重的偶发bug定可重现
- ◀ 并不是所有的bug都能解决或需要解决，测试要坚持，又要舍得放弃

七位测试权威赞誉推荐

思科-网迅资深QA总监 朱少民
腾讯互联网测试部助理总经理 吴凯华
深圳迈瑞超声软件部测试经理 袁娟
北京昱达环球培训总监 CSTQB资深专家 崔启亮
《软件测试精要》作者 董杰
软件测试类图书资深作者 蔡为东
TIB自动化测试工作室 陈能技



策划编辑：张春雨
责任编辑：许 艳
封面设计：侯士卿

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。



上架建议：软件测试

ISBN 978-7-121-12296-5



9 787121 122965 >

定价：49.00元

软测之魂

核心测试设计精解

肖利琼◎著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING



内 容 简 介

测试界有不少朋友都认为，测试设计不就是测试用例的设计吗？这样的回答是不全面的，用例设计只是其中的一项重要内容。测试设计是一个过程，它贯穿在软件生命周期的各个环节，是测试工作的核心、灵魂所在。

本书以测试设计为主线，首先介绍了软件测试行业在过去十多年来的发展变化，如今，实实在在地发生在我们身边的一起起软件质量事故，无不昭示着软件测试行业朝阳的到来。如何把握测试技术，把测试工作做得精透，成为测试行业的佼佼者，是很多读者朋友关心的话题。本书接下来首先明确了测试的目标，然后介绍了测试设计的各个环节，包括测试架构的设计、测试需求分析与测试策略制定、测试方案的设计、用例的设计、测试执行流程设计、测试输出的管理设计、测试过程的控制方法设计等。最后，以追逐软测之理念进行延展，旨在帮助读者理解站在测试工作之上看测试，如何超越自我进行测试创新，为走出一条属于自己的测试精华之路提供指引。

本书是作者从事一线测试工作13年来的测试经验与智慧结晶，适合对软件测试有一定了解，特别是有一定实际测试经验的测试工程师。同时，本书也可以作为高校、软件测试专业培训机构的参考教材，让学生在学习理论知识的同时，学习企业中的工程实践案例，有针对性地认识与把握测试的核心技术，以增强自身的就业竞争力。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（CIP）数据

软测之魂：核心测试设计精解/肖利琼著. --北京：电子工业出版社，2011.2

ISBN 978-7-121-12296-5

I. ①软... II. ①肖... III. ①软件—测试 IV. ①TP311.5

中国版本图书馆 CIP 数据核字(2010)第 223227 号

责任编辑：许 艳

印 刷：北京中新伟业印刷有限公司

装 订：

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编：100036

开 本：787×980 1/16 印张：19.75 字数：410.8 千字

印 次：2011 年 2 月第 1 次印刷

印 数：4 000 册 定价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010) 88258888。

肖老师在深圳工作，我在合肥工作，我们相距很远，但因为同行而认识。更准确地说，是因为我写的《全程软件测试》，这本书成了我们相互认识的纽带。2008年4月，肖老师买了这本书，就书中的问题和我交流，逐渐我们熟悉了，成为朋友。所以，当她开始写本书的时候，自然想到了我，我也很高兴地接受了邀请，为她的新书写序。

肖老师在测试领域工作的时间比我还早三年，到如今已经在测试领域辛勤耕耘了十三年，可以说是国内软件测试的老前辈，见证了国内软件测试从无到有、从小作坊到专业化的测试发展过程，自己也成长为一位资深的测试工程师、测试经理。她的确热爱软件测试，不仅在测试领域一干就是十几年，而且时时刻刻想着这个行业，参加各种软件测试论坛和其他活动，总想为这个行业做点什么。为此，三年前就开始策划写一本软件测试实践方面的书，将自己的经验拿出来和大家分享。

人们经常说，软件测试工作中测试用例设计最能体现测试工程师的水平。如果测试人员对需求理解不够，设计的测试用例可能就不对，所设计的测试用例覆盖面就不全；如果测试人员对产品技术实现不能把握，就不能设计出有针对性的、高效的测试用例；如果测试人员对测试方法和技术掌握不深的话，就不知从哪里下手来设计测试用例。做好测试设计工作，测试执行的工作就相对容易了。只有正确地、全面地设计好测试用例，才能保证测试的执行正确、有效和完整，最终保证发布的产品达到所要求的质量。测试计划、测试执行等固然重要，但测试设计是测试工作的核心。本书正是围绕测试设计来展开讨论的，但又不局限于测试用例的设计，从测试管理设计、流程设计、测试组织结构设计，到测试架构设计、测试方案设计、测试执行设计和测试输出设计等，让设计无处不在，为我们展现了测试的美妙境界。

本书是作者十几年测试工作的结晶，从书中很多的实例可以见证。在每一节，差不多都会有一个实例，这不仅让我们的阅读更轻松，而且更容易理解作者要阐述的实践方法或解决问题的思路。本书语言通俗易懂，希望读者在阅读中获得快乐和知识。

最后，相信读者通过阅读本书受益多多，并能像作者一样，更加热爱测试，在测试领域一干就是十年、二十年，将我国测试水平带向另一个辉煌的未来。

朱少民
资深 QA 总监
思科-网迅（中国）软件有限公司

业界专家的评论

我用了一天时间把本书读了一遍，在印象中已出版的测试类书籍很少有专门讲测试设计的，本书吸引我的就是作者新颖的设计方法和浅显易懂的讲解，每个原理都是娓娓道来，读的过程是非常舒服的。宛如看到作者的良苦用心，也看到了作者非常深厚的测试设计经验和丰富实践，读完后很有体会和感触，也有很大的收获。测试设计作为软件测试里最主要和关键的一个工作环节直接决定了后期测试质量的结果，相信本书对测试爱好者的测试技能提升会有非常大的帮助，推荐广大测试爱好者也来仔细阅读学习。

——腾讯公司互联网测试部 助理总经理、质量管理通道分会会长 吴凯华

本书作者是用心感悟的软件系统测试专家，在十多年的测试实践中深刻认识和理解了软件测试的过程，并总结出指导测试过程的最佳实践方法。对于很多测试工程师来说，它是一本实用的软件测试书籍，通过生动的实例使初学者更容易理解，此书对于理解软件测试方法和过程很有帮助。

——深圳迈瑞生物医疗电子股份有限公司 超声软件部测试经理 袁娟

测试设计是软件测试活动的创造性工作，是提高测试执行的有效性和效率的技术保证。随着我国软件测试行业的发展，软件测试方面的书籍不断丰富，但是专注于软件测试设计技术的书却凤毛麟角，本书的出版可以填补这方面的空白。本书全面论述测试设计的各个环节，以及相应的设计技术，理论与实践相结合，辅以典型测试案例分析。无论您是软件测试架构师、测试工程师，还是测试经理、项目经理，都能从书中得到有益的收获。

——北京昱达环球科技有限公司培训总监 CSTQB 资深专家 崔启亮

测试设计是我们测试者的作战思想，与测试工具一起影响着测试工作的质量。在测试工具和自动化测试书籍占主流的环境下，这样一本专门描述测试设计的书籍，可以很好地帮助广大测试人员获得测试设计领域的知识和经验。本书通过翔实的实战案例阐述了所谈到的测试设计技术，有利于读者轻松理解书中所分享的测试设计思想和方法，且容易运用于自己的实践中。如果你是一位测试经理，本书内的测试管理类经验能让你少走一些弯路，走在正确的路上；如果你是一位测试工程师，本书中的实践经历能让你拓展测试视野，测试专业能力获得新的提升！

——《软件测试精要》作者 董杰

本书内容翔实，论述充分，旁征博引，深入浅出，无论是初入测试行业的新人，还是经验丰富的资深测试人员，都能从中获益。

——软件测试类书籍作者 蔡为东

本书是作者多年软件测试行业工作经验的总结，比较系统、全面地揭示了软件测试的本质、软件测试工作的全过程，不失为一本能引领测试人员真正进入测试领域、正确开展测试工作的好书。

——TIB 自动化测试工作室 (<http://www.cnblogs.com/testware>) 陈能技

早在 1997 年，像很多测试行业的朋友一样，在还不清楚软件测试是做什么时，稀里糊涂地走上了软件测试这个岗位。回首过去，从没想到测试会成为一个行业，也没想到在自己的职业生涯中能坚持这么长时间一直从事这个工作。说来也是很幸运的，能有机会见证这个行业在某些方面的点滴变化，如测试专业书籍的变化，从原来没有专业书到现在琳琅满目。同时也被这个行业中的不少人和事感动着，如近几年涌现了一批批国内的测试专家，他们奉献着自己的实践经验，无论是通过出版专业书籍，还是在网上建立个人博客，无不洋溢着对测试事业的热爱。同时，近几年，伴随着国内软件信息产业的快速发展，出现了一批批从事软件测试服务业的公司或培训机构，使得测试行业从悄然形成，到如今让人感到到处都是一片朝气蓬勃，前景无限美好的朝阳景象。

深圳有一个关山月美术馆，周末常与家人或朋友去看画展，展出的画有国画大师关山月本人的，也有国内外其他名流画家的。每次看完画展回来，都有一番感触。软件测试也是门艺术，一门很美的艺术，只是它的表现手法是以 Bug 来展现的。早在 25 年前，美国软件测试大师 G. J. Myers 在其经典著作《软件测试的艺术》就提到这一点。有感于国内测试领域在近几年的发展，以及目前存在的一些浮躁气氛、认识的误区，作为在测试江湖中摸爬滚打了十多年的自己，是否也可做点什么，如把这些年的经验总结、心得体会写出来与读者分享，让后来者少走些弯路，岂不是一件好事？

有了想法，要把想法变为现实，并非易事。一开始，想得很简单，以为积累了足够的素材，然后把把这些素材整理一下即可成书。由于平时看的专业书多为教科书类，这种书有一个显著特点就是概念多，理论味道太浓，与工程实践有一段距离，常读着读着会让人打瞌睡。但相反，故事情节跌宕起伏的小说类读物却常使我们挑灯夜战不知所累。于是就书的表现风格上想有一个区别于常规的打破，然后在接下来的样章中使用小说式的风格讲述关于软件测试的概念、测试的设计理念等。但这样一来，测试技术最核心的严谨性又得不到体现，后来在与编辑的反复商榷后，改为始终以读者为中心。通过生动的实际案例，结合易读易记的图表来展示书的风格，使读者能在轻松阅读的同时，又不失掌握技术的严谨性。

当书的大纲、样章确定后，就开始了长时间的写作之路。就像我们平时在公司做项目一样，起初我制订了一个漂亮的写作计划，然而不到两周的时间，进度与计划就相背离了，后来就越差越远了。这其中，与公司工作忙，每天能挤出进行业余写作的时间很有限（或者没有）有关。有时确实是太累了，不仅是体力，还有时感到心有余而力不足，比如 1 小时原计划写 1000 字左右，却往往是两小时过去了才理出个头绪。从开始积累写作素材到写作完成花了近 3 年的时间，个人感觉是挺长的。也曾多次想打退堂鼓，但已与出版社签了合同，到期要交稿，诚信是很重要的，有了压力，也就有了动力。记得在写作的中途，正值到武汉进行校园招聘，只好带着稿件每有空时就接着写。整个写作过程发生的事还有

第 1 章 朝阳中的软件测试.....	1
1.1 关于软件测试	1
1.1.1 书中一角到书山一角的跨越	2
1.1.2 捉虫子与挖金矿	3
1.2 Bug 就在我们身边.....	5
1.2.1 惠普 100 款笔记本软件曝严重漏洞	6
1.2.2 奥运门票销售系统被迫关闭	6
1.2.3 美 F-22 机群系统瘫痪，软件质量威胁国家安全	7
1.3 把握测试岗位	8
1.3.1 测试入门	9
1.3.2 优秀测试	11
1.3.3 卓越测试	13
1.4 测试基础简要	14
1.4.1 软件测试基本概念	14
1.4.2 软件测试目的	15
1.4.3 软件测试策略	15
1.4.4 软件测试方法	17
1.4.5 软件测试流程	18
第 2 章 找 Bug 的核心思维与境界	20
2.1 情有独钟的思维模式	20
2.1.1 逆向思维	20
2.1.2 发散性思维	23
2.2 测试的第一重境界：围着 Bug 转	26
2.2.1 独上高楼——发现 Bug	29
2.2.2 为伊消得人憔悴——定位 Bug	31
2.2.3 蓦然回首——关闭 Bug	34
2.3 测试的第二重境界：站在 Bug 之上	36
2.3.1 测试的价值不仅仅是发现 Bug	37
2.3.2 测试的服务链	42
2.4 测试的第三重境界：挑战零缺陷	43
2.4.1 缺陷的防与堵	44

目 录

2.4.2	“零缺陷”文化	46
2.4.3	“零缺陷”后的误区	47
第 3 章	测试设计景观	48
3.1	放眼设计	49
3.2	解读测试设计	50
3.3	测试管理中的隐形指挥棒：测试组织模式的设计	53
3.3.1	以开发为核心的组织模式	54
3.3.2	以项目经理为核心的组织模式	56
3.3.3	独立的测试组织模式	58
3.4	提高测试效率的有力武器：测试流程之设计	59
3.4.1	认识测试流程	60
3.4.2	让大家一起忙起来	61
3.4.3	软件运行得犹如蜗牛在爬行	63
3.5	好钢用在刀刃上：测试技术应用之合适设计	64
3.5.1	通信的心跳在狂蹦	65
3.5.2	解开用例失效之谜	66
第 4 章	测试架构的设计	69
4.1	思索测试架构	69
4.1.1	认知测试架构	69
4.1.2	测试架构设计不仅仅在技术上	72
4.2	让每个测试人员都看到希望	72
4.2.1	回顾与思考微软的测试职业发展路线设计	73
4.2.2	架构合适的测试技术发展梯队通道	78
4.2.3	架构合适的测试管理方向发展轨道	80
4.3	万里航行总舵手——业务测试架构的设计	82
4.4	测试建设之基石——测试框架的设计	84
4.4.1	相框与测试框架	84
4.4.2	化抽象为具体——测试框架内容	85
4.4.3	突破起点——搭建测试框架的方法	88
第 5 章	测试需求分析与测试策略制定	91
5.1	从测试需求开始	91

目 录

6.6.1	代码更改追溯分析法原理介绍	142
6.6.2	应用案例	144
第 7 章	话说用例的设计	146
7.1	漏测一个提示界面，不仅损失 158 万元	146
7.2	逆境中的用例设计	148
7.3	技术攻关：高效用例设计方法	151
7.3.1	隐式边界	151
7.3.2	分类法	155
7.3.3	反常规操作法	160
7.3.4	倒推法	162
7.3.5	用例设计的综合策略	165
7.4	用例有效、无效的正确认识	166
7.5	用例的价值	168
7.6	设计可复用的用例	170
7.7	用例重构	173
7.8	用例设计规范诞生	176
第 8 章	测试执行流程设计	178
8.1	需求测试	178
8.1.1	需求内审中的测试需求	180
8.1.2	需求外审中的测试需求	182
8.1.3	测试设计过程中的测试需求	182
8.1.4	需求测试检查点	183
8.1.5	需求测试中的几个问题	186
8.2	内部版本发布测试	187
8.2.1	版本发布恶梦	187
8.2.2	小议冒烟测试	189
8.2.3	版本发布信息传递	191
8.3	回归测试	193
8.3.1	确定回归内容	193
8.3.2	基于用例的回归测试方法	193
8.3.3	基于 Bug 的回归测试方法	197
8.4	交叉测试	198

8.4.1	交叉测试的特点	199
8.4.2	交叉测试模式	201
8.4.3	交叉测试后的进一步思考	204
第 9 章	测试输出管理设计	205
9.1	Bug 管理	205
9.1.1	“Bug 单”的故事	207
9.1.2	Bug 管理工具的选择	208
9.1.3	Bug 生命周期设计	209
9.1.4	约束的力量——Bug 管理规范	213
9.1.5	Bug 库的应用杂谈	218
9.1.6	处理不可重现的 Bug	221
9.2	用例管理	223
9.2.1	用例管理工具选择	223
9.2.2	用例结构与元素的设计	226
9.2.3	用例维护的设计	230
9.3	测试文档模板设计	231
9.3.1	测试计划模板设计	233
9.3.2	测试方案模板设计	234
9.3.3	测试报告模板设计	235
9.4	测试总结管理设计	238
9.4.1	写总结的好处	238
9.4.2	测试工作日志	239
9.5	测试知识库设计	241
9.5.1	沉淀测试知识库	241
9.5.2	测试知识库的管理	242
9.5.3	学以致用打折吗	244
第 10 章	控制测试过程的实用方法	245
10.1	把握测试工作启动的起点	245
10.1.1	测试人员何时投入项目合适	245
10.1.2	项目测试启动会	248
10.2	测试设计的评审	250

目 录

10.2.1	三级评审机制	251
10.2.2	自审检查单的诞生	252
10.2.3	设计检查单——提高设计质量的有效工具	253
10.3	测试版本的控制	255
10.3.1	版本发布众生相	256
10.3.2	版本接收/停止测试准则	257
10.3.3	测试与版本号	259
10.4	测试配置管理	260
10.4.1	测试也需“电子眼”	260
10.4.2	测试配置的构建与应用	261
10.5	漏测分析：测试流程改进的助推器	263
10.5.1	漏测的定义与漏测分析的意义	263
10.5.2	漏测问题收集	265
10.5.3	漏测分析计划	266
10.5.4	漏测分析实施	266
10.5.5	漏测措施执行跟踪	267
第 11 章	追逐软测之理念	269
11.1	开拓测试管理新思维：测试环境创新	270
11.2	畅想：测试团队的发展之路	272
11.2.1	散兵游勇年代	273
11.2.2	测试游击队	274
11.2.3	测试部落	276
11.2.4	以项目经理为核心的组织模式	279
11.3	测试设计理念至上	281
11.4	挑战测试新技术	282
11.5	测试是不可或缺的“一条腿”	285
11.6	通向“罗马”的测试之路	286
11.6.1	识别自己——英雄不问出处	286
11.6.2	选择一条适合自己的测试康庄大道	288
附录 A	专业名词解释	292
附录 B	参考书目和资源	297

第 1 章

朝阳中的软件测试

欢迎阅读本书的第 1 章，本章将以对软测¹行业有特别影响的专业书籍及网络资源的变化为背景，介绍十多年来测试行业的快速发展及软测工作是做什么的，其核心又是什么。随着信息产业的蓬勃发展，如今软件已渗透到我们工作、生活的方方面面，软件到处都是，它不可能完美无缺，Bug 无处不有。由于软件漏洞或缺陷而引发的事故经常发生，软件测试这个把握软件质量最后一个关卡的工作日益受到业界的重视，测试行业由此也风生水起，已展现出朝阳行业的端倪。如何把握测试工作，在测试行业中赢得一席之地，本章后两节分享了如何把握各阶段测试工作的一些见解，同时也浓缩地介绍了测试基础知识的精华，以期对初学者能指点迷津。

1.1 关于软件测试



¹除非特别说明，本书提到的软测是“软件测试”的简称。

朝阳冉冉升起，万物生光辉，好一幅灿烂的朝阳图景，令人神往。你可知，国内的软测行业正如图中的朝阳，正朝着美好前景蓬勃发展……

1.1.1 书中一角到书山一角的跨越

1997 年的早春，笔者加入了 X 公司的软件研发部，当时，软件测试是做什么的都还搞不懂，却被老板安排到测试员的岗位。虽然在大学里修过张海藩的《软件工程》，但记得软件测试只是其中的一个章节，如图 1-1（左）所示，对其中的细节也不是特别有印象。为了把工作做好，第一天工作结束后就到书店找关于这方面的书。找遍了书店，结果根本没有这方面的书。问问其他同学、朋友也基本上没人知道软件测试到底是做什么的。

十多年过去了，如今到任意一个相对有一定规模的科技书店，带着“软件测试”这个关键词，一定能找到一系列的软件测试类图书。软件测试只是软件工程书中一角的现象再也不会回来了，软测知识体系的形成已有了质的跨越，如图 1-1 所示。

但是，我们国内的软件测试行业，还处于初步发展阶段。由美国 Glenford J. Myers 等著的在业界鼎鼎有名的《软件测试的艺术》一书，其第一版早在 1979 年就出版了，而那时我们国家的计算机发展还处于实验室的研究阶段。换句话说，早在 30 年前，国外一些专家的认识就已经很深刻了，积累了丰富的实战经验，各方面技术配套应用齐全，走在专业技术发展的前沿。从另一个角度看，我们国内的软测行业要走的路还很长，目前我们已从十多年前的萌芽阶段过渡到了初步发展阶段，特别是近年来随着软件产业的发展，已到了软件测试不往前发展都难的境况。

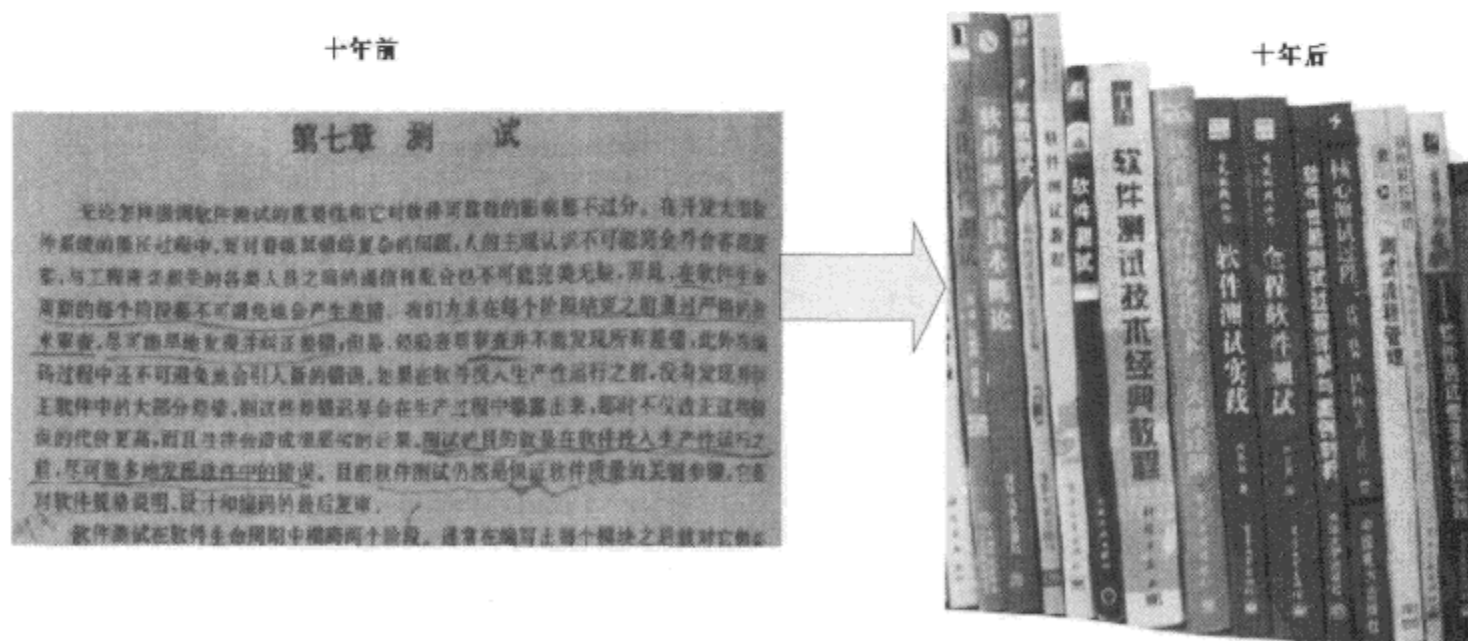


图 1-1 软测知识体系的跨越

1.1.2 捉虫子与挖金矿

有一种职业，它以捉虫子为主要目的，但并不是捉什么动物身上的寄生虫，也不是捉什么植物身上的虱虫，而是专捉隐藏于软件中的臭虫，这便是软件测试。对于软件测试的工作，著名软件测试专家、清华大学教授郑人杰说：“软件测试工作是对质量的把关，其中包含技术及管理等方面的工作，工作相对稳定，对年龄没有限制。而且随着项目经验的不断增长和对行业背景的深入了解，会越老越吃香。”

如今在互联网上，输入“软件测试工程师职业发展”关键词，用百度或谷歌搜索一下，会看到“国家紧缺 IT 人才”、“软件测试人才缺口 30 万”、“软件测试黄金职业”，等等，尽是吸引人眼球的强有力的词汇。受国际金融海啸的影响，2008 年国内外很多大公司利润大幅度下滑，华尔街不断传来让人难以置信的消息，雷曼兄弟投资银行宣布破产，花旗、通用、IBM、微软这些跨国巨头公司纷纷裁员，以缩减开支，类似的负面网络新闻就像雪片般漫天飞舞。然而就在这金融海啸席卷全球每一角落的环境下，国内的软测行业却独树一帜，招聘需求依然灿烂有加。

一方面是人才的紧缺，另一方面是企业用人的急需。市场上的资源供不应求，且存在较大的缺口，一个合格的软件测试工程师难求，更何况一个资深的将领级人才。软件测试人才俨然成了职场的香饽饽。很多公司亮出年薪 10 万、20 万元仍招不到合适的人，正可谓求贤若渴，这是网上活跃的新闻。这也说明了软测行业不往前发展都难，软测职业，已成为职场的新宠。

有道是“风景这边独好！”近年来，网上进行的悬赏捉虫活动越来越多。常常一上网，就跳出某某企业邀请你参加有奖捉虫活动的消息，如图 1-2 所示的内容就是其中之一。



图 1-2 公开悬赏寻找 Bug 海报图

捉一条虫子奖几百元、几千元不等，多劳多得，软测工作犹如挖金子的行当。金子真有那么容易挖吗？这使笔者想起了大家都熟悉的黄金矿工网络游戏，如图 1-3 所示。

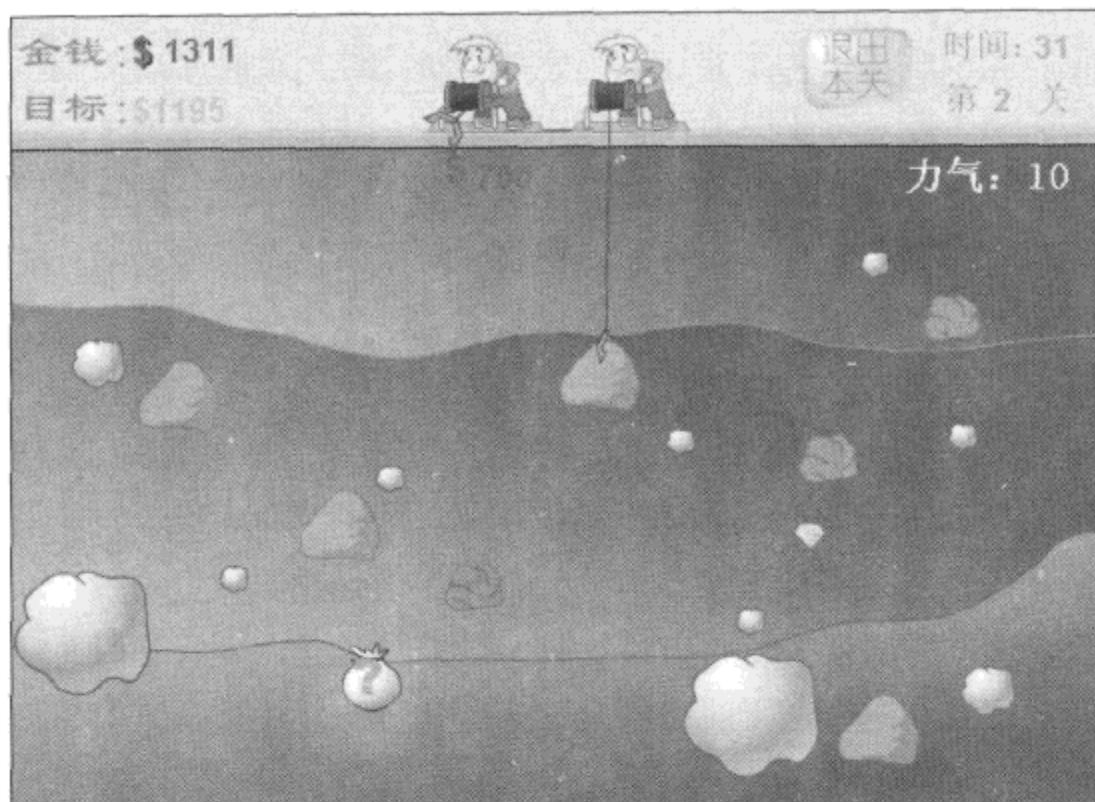


图 1-3 挖金矿游戏

矿中的金子，有大金子、小金子，它们的含金量是不同的，还有钻石，级别就更高了。要挖到不同含金量的金子，挣到更多的钱，顺利闯关，是需要考虑不同的攻关策略的。例如，挖到石头尽快炸掉，以赢得时间。一轮完毕，可用少量的金币购买幸运草、大力水、石头收藏夹等工具，使它们在接下来的一场攻关中能助你一臂之力。黄金矿工双人版，是双人游戏，犹如团队的工作，相互之间密切配合很重要。比如钻石的含金量是比较多的，但它通常藏于乱石之中，在有限的时间内，仅靠一个人的力量来完成挖钻石的工作是不太可能的，这时两人可以先合力把石头挖掉，再由一人挖钻石。这是一种更快地获得钻石的方法或策略设计，也叫计谋。同样的道理，以捉虫为核心目的的软件测试工作，如果设计得好，可使你捉到大虫（严重以上 Bug²）、罕虫（极端条件下发生的 Bug），让你在软件测试中大显身手，成为捉虫高手。

如图 1-4 所示，软件中存在不同类别的 Bug，隐藏得越深越难以发现。

² Bug：原意是“臭虫”或“虫子”，是指电脑系统的硬件、软件中存在的错误。

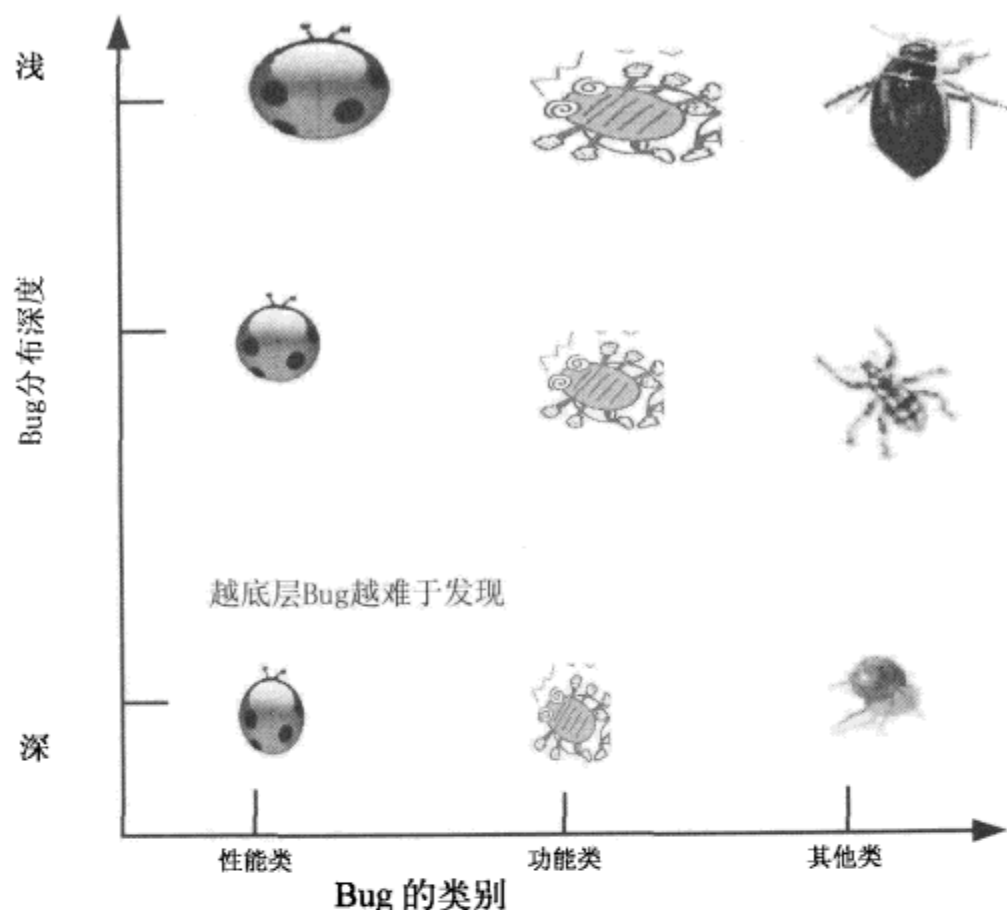


图 1-4 Bug 的分布深度与发现的难易程度关系示意图

1.2 Bug 就在我们身边

在 1.1 节中，我们了解到国内软测行业在近年来的发展变化。这个职业是做什么的，它的发展前景如何，也可从中找到答案。那么，为什么说它是朝阳中的行业，它的背景又是什么，通过本节介绍，不仅可以回答这些问题，而且可理解这个行业存在的必然性及重要意义。

随着信息技术的日益发展，特别是在最近十几年，通信与互联网的迅速崛起，人们的生活已发生了翻天覆地的变化。多年前很流行的一句话“不懂电脑的人是文盲”，如今在很大程度上已成为事实。网购、网游、网上交友、网上开店、网上银行等，每一个主题都异常活跃，现代人的生活已离不开网络。我们在享受着信息化高速公路带来的高品质生活的同时，却也不时会遭遇“黑客”的攻击。如今，没被“病毒”黑过的电脑已是物以稀为贵了。电脑被黑，是因为操作系统或运行其上的应用软件存在漏洞，此漏洞就是设计的缺陷，其中有一部分是软件中存在 Bug。网络软件最常见的是存在安全性的问题，但就问题而言，其他类型的软件，如通信设备、医疗仪器、航空飞机等其中内置的嵌入式软件系统，

同样存在软件漏洞。

信息交流如此发达的今天，软件可以说无处不在，软件是由人设计的，没有 100%完美的软件，所以说“只要有软件在，Bug 就会存在”。本节接下来与大家一起分享几个著名的软件质量事故，它们实实在在地发生在我们生活的周围，小到影响我们的日常生活，大到影响国家安全。

小贴士：

软件的 Bug 和漏洞区别：

Bug 是错误，程序设计中的错误；漏洞是设计的系统中可以被黑客利用的部分，这其中有一部分是程序错误 Bug 造成的，一部分是设计的缺陷。漏洞是外圆，Bug 是内圆，外圆包括内圆，两者之间重叠于内圆。

1.2.1 惠普 100 款笔记本软件曝严重漏洞

2007 年 12 月 19 日，据媒体报道，惠普日前发布了一款补丁程序，修复了 100 款笔记本电脑所预装的“惠普信息中心”软件存在的一个重大漏洞。该漏洞是由赛门铁克安全人员发现的。据安全人员称，该漏洞存在于惠普笔记本（包括康柏品牌）所预装的“惠普信息中心（HP Info Center）”软件的 ActiveX 控件中。利用 HPInfoDLL.dll 这个 ActiveX 控件存在的设计漏洞，黑客可以在惠普笔记本上远程执行代码或远程修改注册表进行恶意攻击，这些恶意攻击包括安装恶意软件、修改注册表信息，以便对受害电脑进行更加复杂的攻击，并从受害电脑中盗取敏感数据。

为了以最快速度控制这个漏洞的爆发，惠普公司先发布了一个补丁程序进行救急，但此补丁程序禁用了“信息中心”一键启动的快捷功能。因此只能说是一种临时的解决方案，是牺牲用户的一个功能来控制黑客的攻击，方案本身会给用户的日常使用带来不便。接下来要完成完整的解决方案即补丁程序的升级，惠普公司本身将消耗大笔的开支，最重要的是给用户带来的损失及声誉影响，是不可估量的。并且在漏洞被发现之前，许多用户已经遭受的损失及其后续影响也是惠普公司需要面对的。

1.2.2 奥运门票销售系统被迫关闭

备受国人关注的奥运门票第二阶段销售工作，虽然于 2007 年 10 月 30 日上午 9 时正

式启动，但是整个过程却并没有外界预期的那样顺利。系统启动工作大概一小时之后，由于访问流量大大超出售票系统的正常负荷，3个门票销售渠道开始无法正常处理票务信息。经过抢修，系统在当日下午5时恢复正常运行，但仍旧出现不稳定的情况。北京奥组委票务中心只好临时召开紧急会议，决定于当日下午6时关闭售票系统。

究竟是什么原因使得奥运门票销售系统瘫痪被迫停止使用？由于第二阶段的售票采用先到先得的原则，在10月30日上午9时刚过，也就是系统刚正式启动时，便有数以万计的抢票大军通过不同的渠道涌入奥运票务系统，抢购门票。据统计，在9时至10时之间，官方票务网站的访问量高达800万次，是系统预设每小时100万次访问量的8倍。这一时段，从3个渠道提交到票务系统的门票订单高达20万张，远远超出了系统预设的票务处理能力，从而造成了网络拥堵、售票速度慢和无法登录票务系统的情况。在众人的购票热情面前，仅有每小时15万张处理能力的票务系统正常运作了不到一小时，就彻底宣告瘫痪。据票务中心相关负责人介绍，从技术角度分析，整个奥运票务网络的带宽并没有出现问题，造成系统故障的主要原因是系统后台的数据库处理能力不足，无法承受大流量的访问。

为了解决此软件设计上的缺陷，奥运票务系统进行了大规模的扩容升级，并承诺系统于12月10日重新开放。与此同时，第二阶段的门票销售政策也只好进行调整，将“先到先得”的售票政策调整为“抽签得票制”。

1.2.3 美F-22机群系统瘫痪，软件质量威胁国家安全

2008年4月14日，有媒体报道，近日，美国空军声称，12架“猛禽”在执行从夏威夷飞往日本的任务中，当途经国际日期变更线时，飞机上的全球定位系统纷纷失灵，多个电脑系统发生崩溃，多次重启均失败。飞行员们再也无法正确辨识战机的位置、飞行高度和速度，随时面临着“折戟沉沙”的厄运。他们不得不掉头返航，幸运的是，当天天气非常好，能见度很高，给“猛禽”加油的KC-135型加油机也可以引导它们安全降落，最终顺利返回了夏威夷的希卡姆空军基地。

事情究竟是怎么回事呢？“猛禽”到达希卡姆机场几小时后，问题就真相大白了，有人在电脑系统编码中犯了一个错误，从而引发了一系列问题。美国空军退役少将史皮尔德称，对于那些“猛禽”战斗机飞行员来说，他们很幸运，因为如果在实战中发生这一问题，他们可能会被击落。并且这个小小的软件错误，将可能成为扭转整个战局的关键点，使美国陷入短时不利的战争局面。不到48小时，颜面大失的“猛禽”的承建商——洛·马公

司就寄来了新的系统软件，使之后的飞行任务得以顺利完成，但此次事件还是给各国敲响了软件质量控制的“警钟”。

以上3个软件质量事故，从软件测试的角度分析，是软件测试工作的疏漏。无论它们是属于安全性问题，还是压力负载，或者系统功能等不同类型的软件缺陷，由于测试人员没有及时发现它们，所以未能避免事故的发生。

1.3 把握测试岗位

读者朋友，通过前面的介绍，如果你是一个业外人士，是不是已经跃跃欲试了，很想知道进入测试领域有哪些要求，自己是否合适从事软件测试工作，那么请读“1.3.1 测试入门”。它将为你揭开“软件测试”大军基本装备的神秘面纱，告诉你合格的测试工程师是什么样的。如果你已经是一名软件测试工程师，首先，恭喜你！但是我们要对自己有更高的要求，才能取得更大的进步。成为一名优秀的测试工程师，是一个不错的选择，在测试的职业生涯中，无论日后是从事测试管理还是测试技术的工作，它绝对是一个含金量极高的砝码。优秀测试有哪些要求，如何才能成为一名优秀测试工程师，请看“1.3.2 优秀测试”，它将为你支招。如果你已是一名能独当一面的业务测试骨干，抑或是某一方面测试技术的大师，再或是在测试项目管理上的佼佼者，如何超越自己，往更高处发展，成为某一测试领域的专家或带头人，建议看“1.3.3 卓越测试”，相信会对你有所启发。

如果把对测试岗位能力的把握程度分为3个阶梯，根据其难易程度，其发展趋势可看成如图1-5所示的“金三角”。

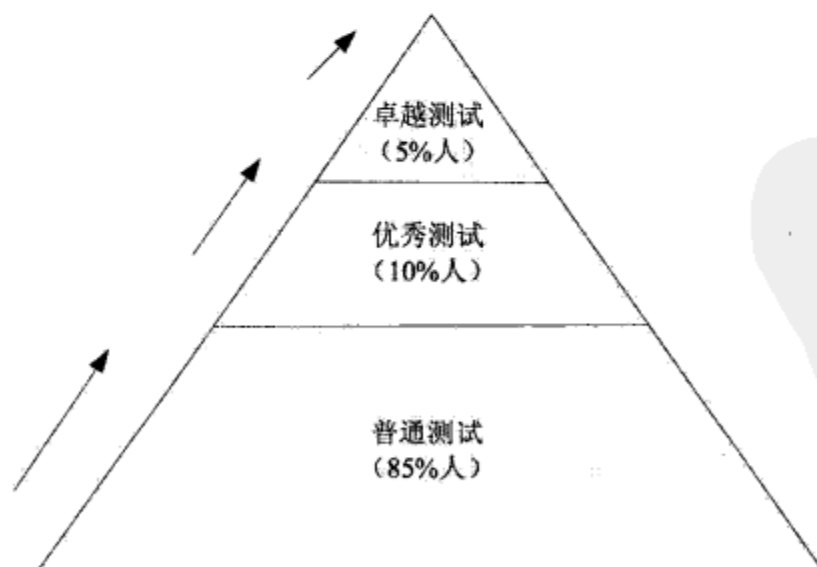


图 1-5 测试岗位能力把握程度“金三角”

图中，每一个台阶都是一个跨越，每一个跨越都需付出代价，特别是要做到卓越测试，

更需付出艰苦的努力，才能成就卓有成效的跨越。图中的人员比例是一个经验值，并不绝对。

事在人为，愿每一个测试朋友都心有目标，不断攀登高峰，成就测试的未来。

1.3.1 测试入门

首先，软件测试是一项技术性工作，在跨入测试这个大门之前，需对软测的基本知识有所掌握，1.4节是单独为这方面知识进行的简要介绍，已熟悉这方面知识的朋友可跳过。当然，如果你足够幸运，不知道软件测试是做什么的，却有机会在软件测试岗位工作，在软测人才如此缺乏的今天，也不是不可能的。比如加入测试执行行列，做一个测试执行人员（运行软件，按他人已设计好的测试用例一条条执行，给出测试结果），但是你很快会发现，由于自己基础知识的缺乏，工作起来很困难，更不用说往下发展。正如古人所说“强扭的瓜不甜”，如何把苦瓜变成甜瓜，关键要看你自己（这个瓜就是你自己）。

在步入测试岗位后，对于任何新人都一样，首先需考虑的是如何做一个合格的测试工程师，这是最基本的，也是最重要的。俗话说“万事起头难”，初来乍到刚做测试时，有这种感受是正常的。下面就从测试技术和测试素质两方面进行总结分享，希望对迷茫中的你有所指引。

1. 技术方面

- 测试设计能力：能够对项目的需求进行分析，提取测试需求、设计测试方案与用例。在不少公司里，测试执行与测试设计是分开的，做测试执行工作的人员只能算是测试员。测试工程师是一个通称，但戴上了“工程师”的帽子更显得专业，对于测试工程师的要求当然也更高。根据从事测试业务的领域不同，测试工程师也有不同的叫法，如性能测试工程师、自动化测试工程师、安全性测试工程师等。无论是哪方面的测试，对于工程师来说，“设计”两个字至关重要，测试工作的展开必须围绕这两个字打开思路，发挥潜能，向优秀测试迈进。测试设计能力，需要软件测试专业知识、行业产品业务知识及其他关联的系统知识做支撑。

小贴士：

“工程师”定义：工程师是把数学与自然科学知识用于实际目的，如设计、建造结构并加以操作的人（Someone who applies a knowledge of math and natural science to practical ends, such as the design, construction and operation of structures）。

- 代码分析：能理解设计文档，包括概要设计、详细设计，并读懂代码，适当的时候从设计原理出发，补充系统测试用例设计，或对测试程序设计进行必要的专项测试，如进行压力测试、负载测试、内存泄漏分析等。曾经有从事测试工作的朋友说：“目前测试界，采用黑盒功能测试方法的居多，两眼一摸黑，什么都不知道，有时被开发人员忽悠了还挺感激的。我们不能太黑了，要突破黑箱操作，就是要看代码”。
- 文档编写：当今是电子化时代，我们工作的输出最后都以电子文档的形式体现，常见的包括测试计划、测试方案、测试用例、测试代码、测试报告、测试总结。一次，有一位软件研发总监说：“很多软件开发人员都怕写设计文档，这是一个不争的事实，但现在一些刚毕业不久的学生写的文档，简直让你大跌眼镜，连语句都不通顺，更不知如何断句，句子没有标点符号、错别字多，有些同音字出现在逻辑性及严谨性都要求很高的技术设计类文档中，真让人哭笑不得，却又苦不堪言”。无论是开发还是测试，文档的编写是必修的基本功之一，现在网上的学习资料很多，此处推荐一本可以系统学习的书籍《软件文档编写》（高等教育出版社出版）。

2. 素质方面

关于软件测试工程师素质特点的文章，在测试专业网站，以及很多书籍上大家可能经常看到。作为一名合格的测试工程师，有哪些基本的关键要求，结合工作中常遇到的问题做如下总结。

- 热爱测试：俗话说“做一行爱一行，三百六十行，行行出状元”，这些道理大家都懂，可是在实际工作中，常听到的是“人在江湖，身不由己”，“身在曹营，心在汉”。如果是这样认为的话，即使你的能力再好，也发挥不出来，更谈不上主动去想，去做测试方法的创新、改进，最终只好南辕北辙，悻悻离开。
- 诚实：这是所有素质中最基本、最重要、最需坚持的原则。用例的执行是测试工作的必做任务，测试的最终结果也需通过用例的执行来体现。用例的执行，就是按照用例的描述在软件上执行操作，判断实际输出与用例中的预期输出是否一致；测试通过写 PASS，测试失败写 FAIL，没有执行的用例记录为 NT（No Test），好像简单得就像 $1+1=2$ 。但事与愿违，工作中常会出现一些投机取巧的“不测分子”，想当然地认为程序的输出如预期，不去实际操作软件，就置该条用例的执行结果为 PASS。后来有一天，其他测试人员发现这个用例其实是 FAIL，“不测分子”却还狡辩在自己当时测试的版本上是通过的，这种劣习是绝对要不得的。

- **耐心与毅力**: 也可理解为不懈的努力、坚持, 在对一轮又一轮成百上千条的测试用例进行版本回归测试时, 测试人尤其需要具备这一素质。在版本不稳定时, 都是人工手动执行, 决定了测试工作是一个重复性很高的工作。没有足够的耐心与毅力, 就会出现偷工减料, 未执行的测试用例置为 PASS 等情况, 敷衍了事, 应付工作, 当然这也与一个人的责任心密切相关。
- **细心**: 不单指细心地发现软件的一些细节上的问题, 如界面排版没对齐、字符串显示不全。这里更多是指在做测试计划的决策、测试设计等测试的核心设计工作时, 需特别考虑周密, 细致、深入地理解需求与设计, 考虑测试的充分性。很多从测试人员手中逃出去的 Bug, 是因为没有设计这样的用例。
- **善沟通**: 这是讲沟通能力, 主要指专职测试人员与开发人员、需求人员之间的交流。测试人员发现的问题提交到 Bug 库后, 常会出现开发人员不能重现或看不太明白, 这个看似描述不清的 Bug, 也是一种交流的方式。沟通是双方的, 当出现测试人员认为是问题, 但开发人员认为不是时, 测试人员需善于表达自己的观点, 且立场坚定, 最好说服开发人员修改, 或启动需求人员参与进行三方讨论等。
- **学习能力**: 知其然, 也要知其所以然, 测试执行人员与测试设计人员的一个重要区别就在这里, 测试执行人员只按用例执行, 通常不清楚为什么要这样设计用例; 而测试设计人员, 如果做不到这一点, 会只停留在软件表面上的应用, 造成测试遗漏。
- **怀疑精神**: 指对软件中存在问题的敏感性, 对于能否发现 Bug 显得很重要, 这个敏感性说的就是对软件的一种怀疑精神。往往, 软件测试人员看到一个界面或一个数据输出不正确, 但由于不能马上重现, 被开发人员说一通不可能后, 很多人就相信了开发人员, 认为自己看错了, 最后选择了放弃跟踪。很明显对自己没信心, 没有足够的怀疑精神, 一些 Bug 就这样在自己的眼皮底下悄悄地溜走了。

前面这些特点是对合格的测试工程师提出的要求, 那么在做好一名合格的测试工程师后, 很多人自然会想到如何做一名优秀的测试工程师, 在技术或管理上要有哪些突破。下节就为大家介绍优秀测试的要求及特点。

1.3.2 优秀测试

上学时, 对学习优秀的认识, 就是每学期结束时评上“三好学生”。参加工作后, 公司每年年终总结时都会进行评优评先进的表彰大会。三好学生也好, 优秀员工也罢, 站在耀眼光环笼罩下的颁奖台上领奖的确是件让他人羡慕, 也让自己备感自豪的事。那么, 优

秀测试是指什么呢？评优时每个公司的衡量标准会不同，但有一点可以肯定，他或她一定在某些方面取得了突出的成绩。下面是在优秀测试人员身上常见到的闪光点。

- **精通业务：**不仅能驾驭好测试流程中各环节的工作，还能超出岗位要求，发现隐含需求问题，并给需求提出有建设性的改进意见。在某个行业领域从事测试几年后，熟悉需求、精通业务，这样的测试人员常是需求设计部门求之不得的人才。
- **精通测试技术：**测试技术上的高手，发现内存泄漏、软件性能方面的严重 Bug，让开发人员折服。版本发布后指定必须由你来测试，只有经过你的确认，版本才能发布。
- **创造性：**有自己的思想，并能在工作中发挥出来；热衷于改进测试流程，主动在工作中不断尝试新方法，研究新技术。
- **富有探索精神：**有强烈的求知欲，不耻下问，爱较真，常会抓住软件的不良迹象，发现一堆 Bug 给开发人员。
- **分析定位问题：**发现问题后，还能分析问题，甚至告诉开发人员问题出在哪里，如何更改，有哪些影响。

优秀测试人员的这些素质，有些是可以在后天通过一段时间的自身努力达到的，如业务知识、测试技术、分析定位问题的能力。而探索精神，并不是一个人通过一天两天，甚至通过一年两年的学习就可养成的，它是一个人对各方面知识、认知、习惯培养等多方面综合素质的表现。这类测试人员常常喜欢对问题刨根问底，爱质疑（甚至有时被某些开发人员鄙视为异类、变态），而正是由于这个被视为“毛病”的品质，使得很多设计上的缺陷都逃不掉他们的火眼金睛。探索，敢于探索，使他们更有自己的想法。对问题的日积月累，慢慢地形成了自己的一套想法、看法。量变带来质变，自然而然就会带来创造性的思维，创造性的成果也就体现在工作中了。探索精神是创造性思维的前身，下面的小故事，正好说明了这个道理。

【牛顿与苹果的故事】

传说 1665 年秋季的一个下午，牛顿坐在自家院中的苹果树下正冥思苦想行星绕日运动的原因，这时，一个苹果落下，正巧落在牛顿的脚边。这是一个发现的瞬间，其实这次苹果下落与以往无数次苹果下落并没有什么不同。几千年来谁都觉得是理所当然的事，但却引发了牛顿的一些疑问，为什么这个苹果要垂直向下落到地面上？为什么它不斜着下落或飞到天上，而是始终朝着地心的方向？为了释疑，他进行了深入地研究，后来提出了令世人惊叹的“万有引力”定律。正是刨根问底、热爱思考、敢于探索的精神成就了牛顿的伟大！

1.3.3 卓越测试

卓越，从字面理解就是杰出、非常优秀，比优秀做得更好，就像学校里跳级的学生一样出类拔萃。如何从优秀到卓越，借用美国知名作家勒纳夫妇合著的《从优秀到卓越》一书中的一段话：对优秀的跳高运动员来说，尽管他们的成绩已经非常出色，但他们还是需要不断“提高自己面前的横杆”，以求每次都能跳得更高一点。这就是实现了“从优秀到卓越”。

那么，对于测试，如何从优秀测试到卓越测试呢？其实道理一样，就是“不断提高自己面前的横杆”。这样说好像很空，在实际操作时，这个横杆有时可能并不是有形的东西，比如当一个优秀的测试工程师，无论测试技术或业务知识都很突出，是不是已达到卓越，已没有发展的空间了？卓越测试确实那么难还有其他原因，为什么能达到卓越测试的人凤毛麟角？卓越测试人才，他们都有哪些特点？下面进行介绍。

- 测试事业：把测试当成自己的事业，对测试工作特别热情，富有激情，这种激情常能感染团队中的其他成员，一直朝着积极向上的方向前进。很多优秀的测试人才，在做了几年后，发觉前路已亮起了红灯，出现了发展的瓶颈。无论是从技术上还是管理上都难于进阶，于是很多人选择了退出，转行做开发、做需求、做销售、做生意等。甚至还有不少测试人员，可能从来都没有想过把测试工作当成自己的事业来做，也没想过其实自己有潜力可以做得比现在更好。
- 测试指挥官：培养及带领一批测试精兵强将，上刀山，下火海，在软件问题的救火场景中常见到他们的身影，高质量突出地完成一个个项目的测试。
- 分享与传递：卓越测试人员是可遇而不可求的，好像他们有测试DNA，除了他们自身有很好的问题敏感度，能找到真正的问题外，他们还乐于把资源、知识、经验分享并传递给团队中的其他成员，推动团队的共同成长。
- 专业技术的带头人：某一技术方面的专家、带头人，敢于尝试新方法、新技术，进行变革创新，突破瓶颈，取得认可。
- 引领未来：身处一角，眼观世界，把握市场，洞察未来，结合当下，前瞻思考，构建测试的未来，是测试团队中的指路明灯。

下面，我们来分享微软这个世界顶级软件开发公司的卓越测试团队的工作模式。

【微软卓越测试团队】

微软在2003年创造了卓越工程（EE）团队，团队的宗旨除了技术培训外，还有发现和分享整个公司的工程最佳做法。团队作为一个整体包含了所有的工程领域，卓越测试团队便是代表测试领域的其中一个团队。卓越测试团队认为微软的每一个测试人员（包括从

新员工到副总裁)都是他们的客户,团队的主要任务可以概括为共享、帮助、交流。共享包括共享做法、工具和经验,他们每年举行 20 多场测试讲座,交流有效的测试方法。帮助是指帮助测试相关人员解决问题,或许有些问题他们本身不能解决,但他们可以通过与其他团队的联系来解决问题,卓越测试团队担任着促进者和专家的角色。交流指向所有测试成员,沟通交流他们所知道的和所发现的信息。他们还预测微软测试工程师未来的需求,并积极主动地确定关于未来软件测试的长远规划,给测试领域指引方向和指导需要做的工作。更详细的介绍见《微软的软件测试之道》一书。

1.4 测试基础简要

如果你打算介入测试领域或者你是测试新手,那么本节的内容会很适合你,包括常接触到的基本概念、常用方法的介绍。

1.4.1 软件测试基本概念

通常对软件测试的定义有以下两种描述。

定义一:软件测试是为了发现程序中的错误而执行程序的过程。

定义二:软件测试是根据软件开发各阶段的规格说明和程序的内部结构而精心设计的一批测试用例(即输入数据及其预期输出结果),并利用这些测试用例运行程序,以及发现错误的过程,即执行测试步骤。

测试方案:阐述对于某一特定的测试点如何去测试的思路,也就是阐述用什么方法、如何去测试问题。

测试用例:所谓测试用例是为特定目的而设计的一组测试输入、执行条件和预期输出,测试用例是执行测试的最小实体。另外,设计的测试用例应具有步骤清晰、操作性强的特点。一个好的测试用例是发现了迄今为止尚未发现的错误的用例。

测试执行:根据事先设计好的测试用例而执行程序的过程。这个过程需要根据用例执行的输入数据,判断执行程序后的输出结果是否正确。

缺陷:我们常说到的 Bug(严格意义上来说,缺陷不等于 Bug,详见第 9 章小知识“Bug 与 Defect 的区别”),对软件缺陷的定义有以下 5 条描述:

- 软件未达到产品说明书中已经标明的功能。
- 软件出现产品说明书中指明了不会出现的错误。
- 软件未达到产品说明书中虽未指出但应当达到的目标。

- 软件功能超出了产品说明书中指明的范围。
- 软件测试人员认为软件难以理解、不易使用，或者最终用户认为该软件使用效果不佳。

1.4.2 软件测试目的

表面上看，软件测试的目的是要证明程序有故障存在，并且要尽可能多、尽可能早地发现程序中的错误。实际上，暴露问题不是软件测试的最终目的，发现问题是为了解决问题，只有解决了问题，软件的质量才有提高，才达到了测试的最终目标。

正如前面所说的“软件测试是为了发现程序中的错误而执行程序的过程”，正确认识这一点很重要。因为不同的测试目标，会设计不同的测试用例。如果目标是发现程序中的错误，则设计测试用例时就会考虑用各种方法设计出最能暴露错误的测试用例。反之，就会设计一些证明程序是正确的用例去执行。

1.4.3 软件测试策略

对一个项目进行测试，在做测试计划时，必须考虑清楚对本项目要采取哪些测试的策略，也就是说你将采用什么手段或方法进行测试。

在软件开发生命周期中，可划分不同的开发阶段，如常用的软件开发 V 模型，如图 1-6 所示。对应开发的不同阶段，软件测试也分为不同的单元测试、集成测试、系统测试及验收测试。

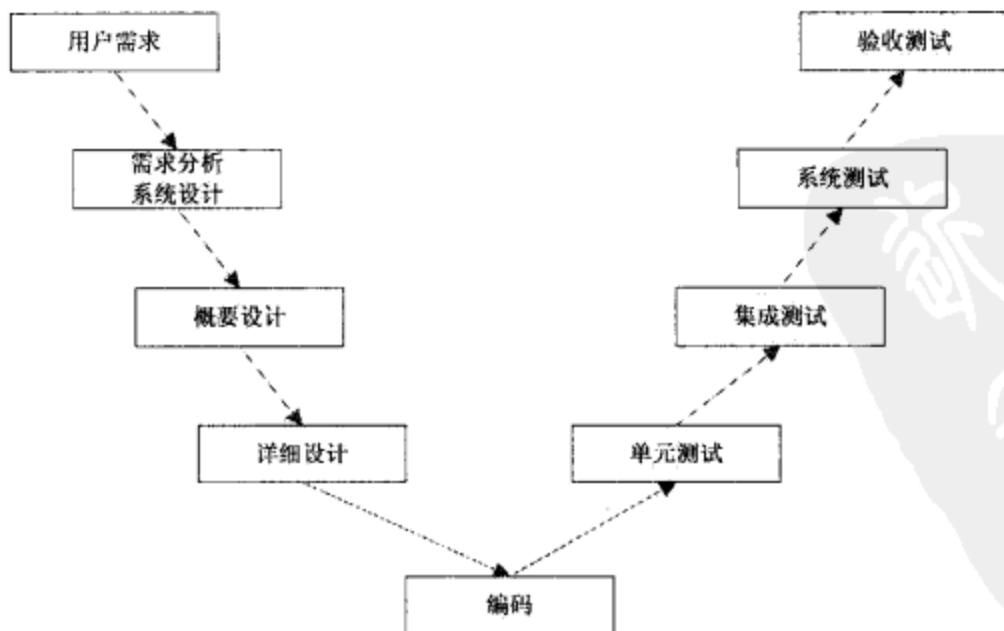


图 1-6 软件开发 V 模型

但是，通常需根据项目的进展及要求的具体情况，可采用迭代式的开发模式、敏捷开发模式等，各阶段也可并肩地进行。

单元测试：又叫模块测试，是软件开发各阶段中最低一级的测试活动。通常由程序编写人员自己完成，目的是检验程序的最小单位（模块）有无错误。针对编写的源代码，采用各种白盒测试技术进行，如语句覆盖、判定覆盖、条件覆盖、条件组合覆盖等方法。在面向过程的结构化程序中，如 C 程序，其测试的对象一般是函数。在面向对象的程序中，如 C++，单元测试的对象可以是类，也可以是类的成员函数。

单元测试主要针对以下 5 个方面进行测试：

- 模块接口。
- 模块局部数据结构。
- 重要的执行通路。
- 出错处理检测。
- 边界条件测试。

由于一个模块仅完成某一方面的功能，因此在软件系统中，一个模块不可能独立存在，模块与模块之间必然存在相互的联系。在进行单元测试时，需要根据具体情况来设计驱动模块和桩模块，与被测试模块一起构成测试环境。

集成测试：在单元测试之后，把通过单元测试的模块，按照设计的要求组装起来进行测试，主要目标是发现与接口有关的问题。

集成测试模块组装的方法分为非增量式和增量式两种。非增量式是指先分别测试每个模块，然后把所有模块按设计要求放在一起组装成所要测试的程序。增量式是把下一个要测试的模块同已经测试好的模块结合起来进行测试，测试完成后，再把下一个待测试模块结合进来测试，这样每次增加一个模块，直到把所有模块都测试完成为止。

系统测试：把软件、硬件及系统其他组成部分集成在一起，模拟最终用户的使用环境，站在用户的立场进行的综合性测试。目的是保证系统各组成部分能够协调工作，系统的性能达到产品规格的要求。

验收测试：根据软件需求说明书的规定，对软件产品进行评估，以确定其是否满足软件需求的过程。经过确认测试后，应对软件给出结论性评价，即合格或者不合格。如果软件的功能、性能及其他方面的要求都满足软件需求规格说明的规定，则视为合格的软件，反之则视为不合格，给出缺陷清单并提交到开发部门。

1.4.4 软件测试方法

1. 静态测试和动态测试

按照软件测试是否执行程序而言，软件测试通常可分为静态测试和动态测试两大类。

1) 静态测试

静态测试的主要特征是不实际运行被测试软件，主要目的是对软件的编程风格、结构等方面进行评估。

静态测试包括代码检查、静态结构分析、代码质量度量等。它可以由人工根据编码规范进行检查，通过分析代码结构来检测程序的正确性，也可借助软件工具自动进行，如PC-Lint便是一款不错的代码静态检查工具。进行静态测试的人员必须熟悉程序的结构、参数定义等，还需要测试人员对产品知识有一定程度的认识。这样不仅能发现程序正确性方面的问题，还能站在用户的角度分析发现程序的实现是否合理。

静态测试能发现逻辑设计和编码错误方面的大部分问题，但代码中仍会有隐藏的故障无法通过静态测试来发现。静态测试的结果不能作为最终结果，还必须通过动态测试进行详细地分析及验证。

2) 动态测试

动态测试与静态测试正好相反，必须真正地运行被测试程序，通过输入测试用例，对运行过程中的输入、输出数据进行分析，从而得出测试结论。

动态测试包括功能测试、覆盖率分析、性能分析、内存分析等。功能测试可采用手工测试，视具体情况也可采用自动化测试的方法。覆盖率、性能、内存的分析上常要借助其他软件测试工具，或者自行编码以实现测试的目的。

静态测试和动态测试是相辅相成的，通过静态测试可以比较细致地对代码进行检查，通过动态测试则可以对软件运行的结果得出明确的结论，具有较强的确定性。

2. 黑盒测试和白盒测试

如果按照测试时是否考虑程序的内部结构实现来分，软件测试可分为白盒测试和黑盒测试。

黑盒测试（Black-box Testing），又叫功能测试，是根据软件规格说明书的要求，运行并验证程序是否满足用户的需求，是一种从用户立场出发的测试。这种方法把被测试程序当做一个黑盒子，不考虑程序内部的逻辑结构和特性，通过输入测试数据，根据需求来判断输出是否正确。黑盒测试一般被用来确认软件功能的正确性和可操作性。

白盒测试（White-box Testing），又称为结构测试，是通过分析程序内部的逻辑结构，针对特定条件和循环设计测试用例，对程序的逻辑路径进行测试。要求测试者对软件的结构和代码相当熟悉，测试的目的是尽量覆盖程序的每一个分支，在程序的不同点检测“程序的状态”以判定其实际情况是否和预期一致。

这两种测试方法的出发点是完全不同的，反映了测试思路的两个方面。两种方法各有侧重，不能替代。黑盒测试的优点是效率高和实用性强，缺点在于测试往往是不完全和不充分的；白盒测试的优点在于能够对程序内部的特定部位进行覆盖测试，缺点是无法测出程序未实现的功能。

总之，黑盒测试和白盒测试各有自己的优、缺点，测试的效果是相辅相成的。工作人员在规划测试方案时，需要将黑盒测试与白盒测试结合起来，才能把软件测试得更充分、更有可靠性。

1.4.5 软件测试流程

软件测试贯穿于软件开发的整个生命周期中，在不同的阶段可采用不同的测试方法。但不管是单元测试、集成测试、系统测试都可以按下面的流程进行：

- （1）编写测试计划。
- （2）设计测试方案。
- （3）设计测试用例。
- （4）执行测试。
- （5）故障跟踪。
- （6）输出测试报告。
- （7）测试总结（分析）。

如图 1-7 所示，是一个没有考虑其他分支、迭代情况的软件测试环节流程图。

一般情况下，软件测试在工程实践中经常实施的有单元测试、集成测试、系统测试。系统测试，人们又常把它称为功能测试，这其实是不全面的，但也有一定的道理。因为系统测试主要的工作是功能测试，就是根据软件的规格说明书验证软件是否满足用户的需求。但除了测试功能之外，系统测试还必须验证系统的性能是否满足用户的需求，所以系统测试是不完全等同于功能测试的。

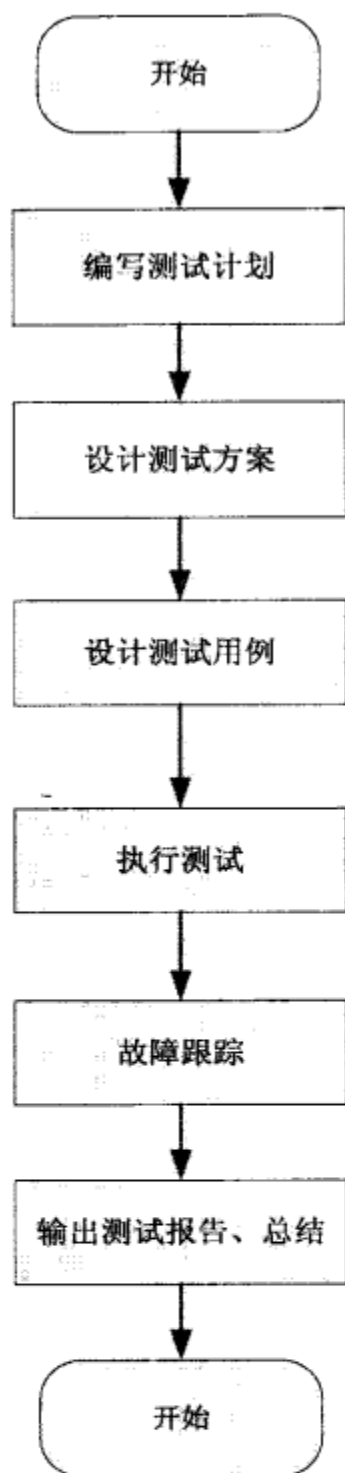


图 1-7 软件测试流程简图

第 2 章

找 Bug 的核心思维与境界

通过找 Bug 来推动软件质量的提高，是软件测试最基本的目的，围绕着 Bug，本章首先介绍了逆向思维与发散思维在找 Bug 中发挥的价值。接着介绍了测试的三重境界，对不同境界的理解，会有不同的行为结果。第一重境界：围着 Bug 转。通过发现 Bug，协助开发人员分析问题，定位问题，最后解决了 Bug，才能对质量有实质的贡献。第二重境界：站在 Bug 之上。我们完全可以跳出测试软件的小圈子，拓宽测试的视野。测试的价值不仅仅是发现 Bug，它服务于整个产品的开发链，项目的成功，可以带来测试的成功。第三重境界：挑战零缺陷。Bug 是设计出来的，它从来不会自生自灭。追求零缺陷，以预防为主，事后的测试验证为辅，主动推动设计尽量一次做好，这是测试的最高境界。通过测试三重境界的介绍，读者可以对测试的价值有更全面更合理的理解与认识。

2.1 情有独钟的思维模式

测试思维模式各种各样，如大家常用的逆向思维、组合思维、发散思维、比较思维等，其中“逆向思维”与“发散性思维”是重中之重，对测试的质量与效率有着重要的影响。

2.1.1 逆向思维

在软件研发办公室里，常会看到这样的场景如图 2-1 所示，测试工程师发现了一个严重的 Bug，笑呵呵地找负责此模块的开发工程师来看现场，以确认 Bug 的存在并且做现场分析。而开发工程师却是一副神情紧张、半信半疑的样子走过来，即便确认是一个 Bug 后，

还不相信别人也不相信自己，止不住地说“这怎么可能呢？我看看代码再说”。读者朋友们，这是为什么呢？

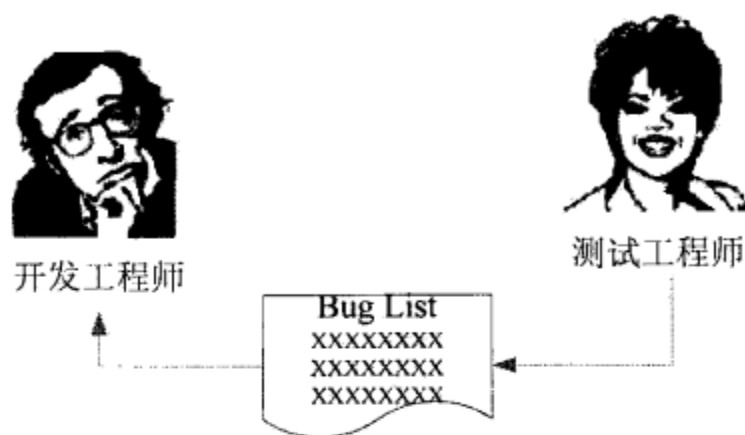


图 2-1 测试与开发之间的 Bug List 传递场景示意图

在笔者接触过的开发工程师中，曾经有不少人提到过同样一个问题：“他们辛辛苦苦、日夜奋战编写代码，设计出来的软件就像是他们的孩子，对它们有着一种特殊的感情。特别是在大型项目中，经过几十万，甚至上百万行代码编写的历练后，最后要离开这个项目，与这些曾经日夜为伴的代码说再见时，总有那么一种恋恋不舍的感觉”。从心理学的角度看，这也是符合人的正常心理的，谁愿意自己生产的是一个多病的“孩子”呢？但现实却很残酷，谁家的“孩子”会没有生过病呢？笔者曾经遇到过很多次这样的情况，测试提交 Bug 后，与负责的开发工程师交流，然而开发工程师却诡异地笑道“啊！此 Bug 终于被你们发现了，我还以为你们发现不了呢。”当时真不理解，怎么会有这种事情发生，看到代码中存在的 Bug，却不主动去改。现在回想起来，是现实与心理的种种原因的驱使吧，毕竟任何一个 Bug 的修改与否都存在风险。

在业界，也有人形象地比喻说，开发是盖房子，测试是拆房子，这种看法或许存在某些偏见。实际上开发也好，测试也罢，大家的目标是一样的，都是为设计出满足用户需求的高质量的软件。只是由于分工不同，开发与测试在工作的性质上天生成了对立面，开发的工作是要把需求实现，而测试是验证开发的实现是否满足需求，这种验证本身是带着一种怀疑的眼光和不信任的姿态，这也造就了测试工作本质上就需要有与开发不一样的逆向思维。事实证明，这种思维越强烈越能发现问题。测试的目的是发现 Bug，证明程序有错，要证明程序有错就需要有说服力的数据，而这些有说服力的数据就是 Bug。如果仅从正向思维出发，设计的测试用例自然也是正向的，这与开发人员进行设计实现时走的是同样的路，即验证程序是按需求实现了，能达到预期，但实现了的产品是否存在问题，不得而知。正向思维思考与逆向思维思考，仅是一字或一念之差，结果却大不一样。

逆向思维的方法，笔者通常把它理解为“不走寻常路”，这也是程序员常忽视的地方。

如图 2-2 所示是一个场景示意图，小红每天从家（S 点）出发到学校（C 点）上课，通常情况下，她的路线是 $S \rightarrow O \rightarrow C$ ，但是她也可以走到一棵树（O 点）时回头返回 S 点，然后再往回走或选择其他路线到学校，如 $S \rightarrow O \rightarrow S \rightarrow O \rightarrow C$ ，或者是 $S \rightarrow O \rightarrow S \rightarrow D \rightarrow O \rightarrow C$ 等。当然，实际生活中这种路线走法的人可能并不多（让人感觉有点不正常），但它确实也是一种数学上的拓扑路线。这种不寻常的路线可以看做是软件业务中实现某种功能的特殊路径，常常潜伏得较深的 Bug 就是需要这种迂回曲折的路径后才能暴露出来。这种路径如何有意识地执行，须对测试对象进行深入分析。

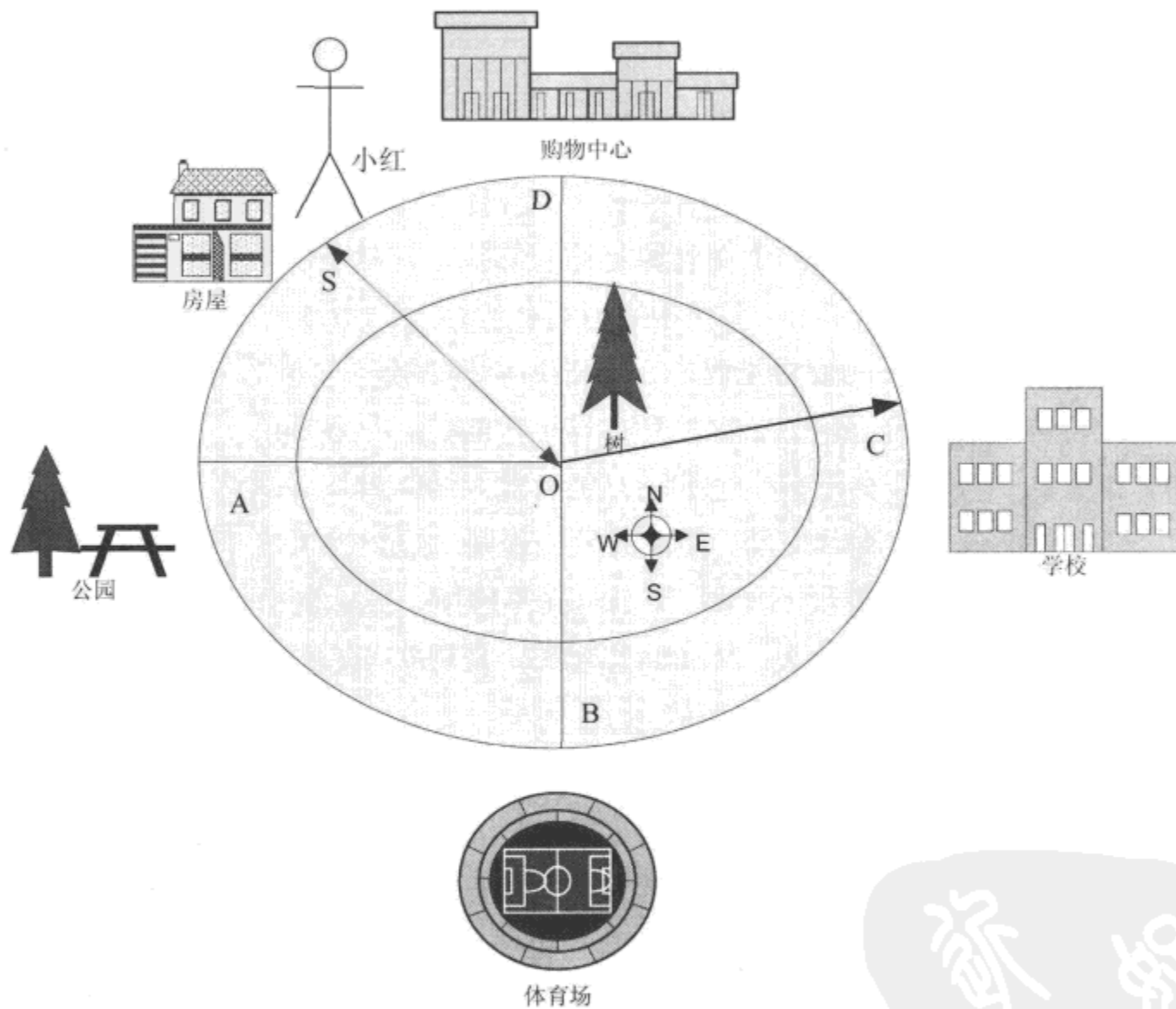


图 2-2 “不走寻常路”场景示意图

下面结合实际软件测试应用场景进行讲解。

【案例】

某手机通信录列表显示界面如图 2-3 所示。通信录列表用来显示用户的通信信息，包括姓名、电话号码等。界面图标工具栏上有“删除”按钮（图中红线标识），用于删除记

录。就“删除”记录的功能来说，正常情况下，选择一条记录后，单击“删除”，记录即可被删除。但是曾经遇到过这样的 Bug：列表中只有一条记录，删除此条记录后，再单击“删除”按钮，手机屏幕突然黑屏，程序 crash（崩溃）。后来查明是按钮的状态不对，当列表中没有记录的时候，删除按钮仍在激活状态。记录删除了（列表已为空），为什么还要去单击“删除”？难道测试人员真是没事干？而这恰恰是测试用例设计“不走寻常路”的逆向思维的体现。

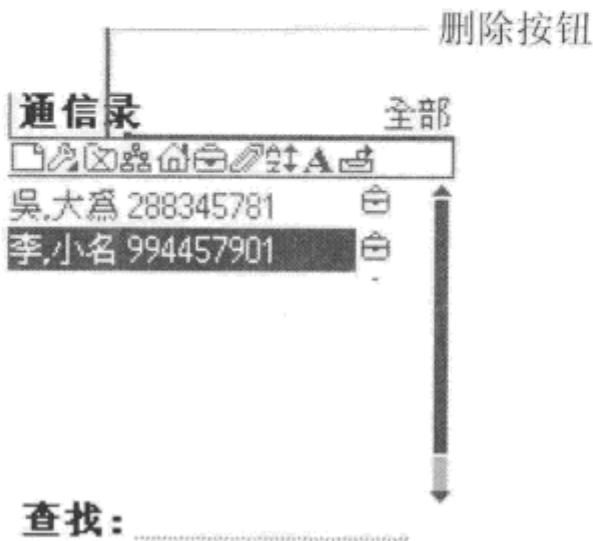


图 2-3 通信录界面示意图

不走寻常路，大家都是这样走，我却往反方向走，用与正向思维相反的思维方式设计用例，挖掘新 Bug。这是一种历练，也是历练中的摸索、创新。通过这种不同寻常的历练，常能让我们总结出自己的方法，并且在测试方法上不断得到扩展，让我们成长。

小贴士：

用不走寻常路的“逆向思维”设计用例，关键点在于找准测试对象的反面。

2.1.2 发散性思维

发散性思维是一种探求多种答案，最终使问题获得圆满解决的思维方法。由于其视野开阔，思维活跃，可以产生出大量独特的新思想。著名的心理学家吉尔福特指出：“人的创造力主要依靠发散思维，它是创造思维的主要部分”。正如其本身的特点所言，发散性思维在软件测试工作中的应用，从发现的问题来看，很有创意，是一种很好的新思维、新测试方法的体现。

概括来说，发散性思维在测试过程中可在以下两个阶段得到充分体现。

第一：测试设计阶段的发散。

也可理解为测试方案，即测试思路的形成阶段。下面以大家日常工作、生活中经常见到的场景为例，介绍发散性思维在软测中的应用。

【案例】

某公司招聘测试工程师时，有一道这样的笔试题：“某嵌入式软件有用 U 盘导出数据的功能，请写出测试此功能点的思路”（读者朋友，在往下看之前，建议先做做看）。

下面给出测试此功能点的测试思路，如表 2-1 所示。并非标准答案，旨在说明围绕一个测试对象，如何运用发散性思维进行思考。

表 2-1 发散性思维设计测试用例

考虑方向	检查点	测试思路简述	备注
正向功能	导出数据的正确性	用 U 盘导出某软件的数据（包括某软件产生的各种类型的数据），采用可行的方法，验证其导出数据的正确性	
正向功能	导出功能的有效性处理	某软件待导出数据不存在时，导出功能是否正确	
逆向功能	导出功能的配置	软件安装后，导出功能的配置是否正确	此点与软件的实现有关
边界容量	U 盘空间不足时的处理	当前 U 盘空间只能容纳待导出的部分数据时，执行数据导出	
边界容量	U 盘空间满（剩余 0byte）时的处理	U 盘空间满时，执行数据导出	
容错	U 盘写保护处理	U 盘写保护时，执行数据导出	
容错	坏 U 盘的处理	用一个坏 U 盘进行数据导出	坏 U 盘的准入条件可以 Windows 能否正常写入数据为准
容错	人为非法操作容错处理	导出数据过程中，拔出 U 盘，软件的后续处理是否有异常	
容错	软件工作时，遇特殊情况的容错处理	数据导出过程中，突然断电，再开机检查 U 盘中的数据及软件导出功能是否符合预期	
性能	压力测试：连续导出 N 次后，数据的正确性	脚本自动执行，连续导出 N 次（ $N > 1000$ 次）数据，检查数据的正确性	导出 N 次，重写 U 盘可以覆盖之前的数据，也可以追加的方式写入
性能	U 盘导出速度	同一 U 盘，从软件中导出批量数据的速度，与同样大小的数据在 Windows 下的写速度相比	
性能	不同 USB 驱动协议，对导出速度的影响	分别用 USB 1.0、USB 2.0 的 U 盘进行批量导出，比较其速度	

(续表)

考虑方向	检 查 点	测试思路简述	备 注
其他	不同分区格式 U 盘的识别	分别用 FAT, FAT16, FAT32, NTFS 等分区格式的 U 盘进行数据导出测试 (关注数据的正确性)	
其他	不同品牌 U 盘	用在多方面流行的不同牌子进行导出 (应与不同品牌无关)	可以 Windows 操作系统能识别为准入条件
其他	U 盘外接延长线使用	用户场景的考虑, 使用符合标准的 USB 接口延长线时进行数据导出, 验证导出数据的正确性	USB 规范的有效距离是 1.5M

第二：用例执行阶段的发散。

我们都知道，用例执行时须严格按事先已设计好的用例来测试，但由于在实际工作中，软件测试不能进行穷举测试，用例对代码的覆盖率做不到 100%，特别是对一些条件组合语句、模块接口相互影响之间的覆盖，更难覆盖全面。根据这一特点，在测试执行完某一测试点的用例后，可以根据已有用例的情况，进行发散性测试，这种发散性测试，也叫随机测试。经验越丰富，随机测试的效果越好。但这种随机测试并不是随便测试，它是有一定依据支持执行的。

或许，我们可以这样理解，对某一条已有的用例进行分解，即不完全按事先设计好的操作步骤进行，或走了一步或二步后，跳出既定的步骤提前结束，结果往往能发现问题。跳出既定的步骤，可以回头张望，可以疾步跳跃，可以反反复复做某个动作，这种思维发散的过程，本身就是一种用例设计的过程，如图 2-4 所示。这个过程就好像是我们的思维与已设计好的软件在玩一种游戏，是一种试探性的对抗。这种玩法，常能出乎程序员的意料，发现一些特殊情况下才能发生的问题。

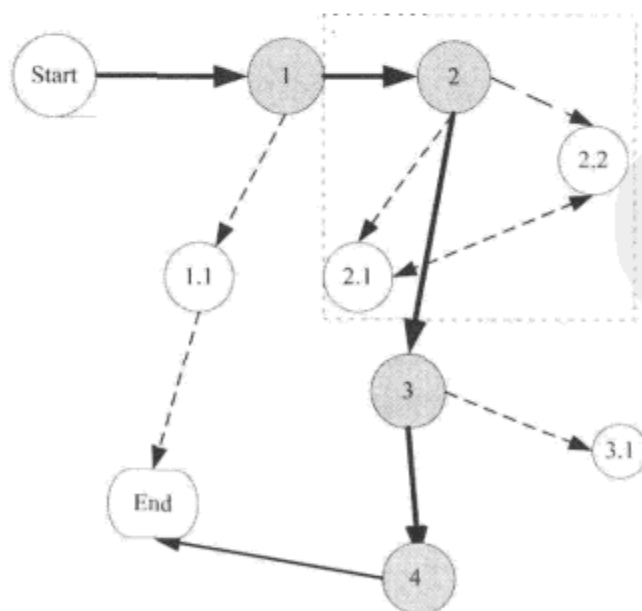


图 2-4 用例执行发散示意图

除了发散，我们还需严谨。这种发散思维，如果无边无垠，任其驰骋，就像一匹野马，如果控制不住，事情会弄巧成拙，可能做了很多工作，到最后却事倍功半。这也就要求我们既要有发散测试的创造性思维，又要有一个严谨科学工作的头脑。

下面对用例执行过程中的发散思维示意图进行解释。

图 2-4 中数字表示操作步骤，1.1 表示由步骤 1 扩展出来的分支步骤，依次类推，2.1、2.2、3.1 的意思类似。

正常主操作流程用例：Start-1-2-3-4-End。

发散用例 1：Start-1-1.1-End，从步骤 1 中分支出来，到结束，有走路时抄小路的味道。

发散用例 2：Start-1-2-3-3.1，此用例不能到流程结束，即不能完成某一有意义的功能，但测试时可以进行探索。正常情况下，它可以回到步骤 3，如果进行到 3.1，软件不能进退，说明设计上存在缺陷。

发散用例 3：Start-1-2<->2.1<->2.2，步骤 2 与扩展步骤 2.1、2.2 之间反反复复来回执行，如图 2-4 所示的虚框。

2.2 测试的第一重境界：围着 Bug 转

“意识决定行动，行动决定结果”是管理学中众所周知的名言。做测试的前几年，笔者并没有这个意识，也没有主动地去思考过这个问题，但随着一个个项目任务，一桩桩事件的历练，慢慢感悟到这句话也适合对测试工作境界的理解。“心态决定命运”，“态度决定一切”，有很多名家学者都写过这方面的书籍，基本上已成了我们不可否认的真理了，但是要真正应用在自己的工作生活中，恐怕就不那么简单了。诚然，测试工作，除了需要拥有过硬的测试技术外，还必须有正确的测试心态，也正是这些心态意识左右着你的日常工作。不同的心态反映了不同的测试境界高度，最终体现出不同的结果。

围着 Bug 转，是测试三重境界中的第一重。概括起来，它又可以分为三个阶段，第一：发现 Bug；第二：定位 Bug；第三：关闭 Bug。这三个阶段对测试人员的要求不仅在技术上需要逐层递进，在综合素质上也提出更高的要求。三个阶段之间环环相扣，直到 Bug 的生命周期结束。围着 Bug 转的三个阶段对测试人员的要求及 Bug 被发现到关闭的生命周期示意图，如图 2-5 所示。

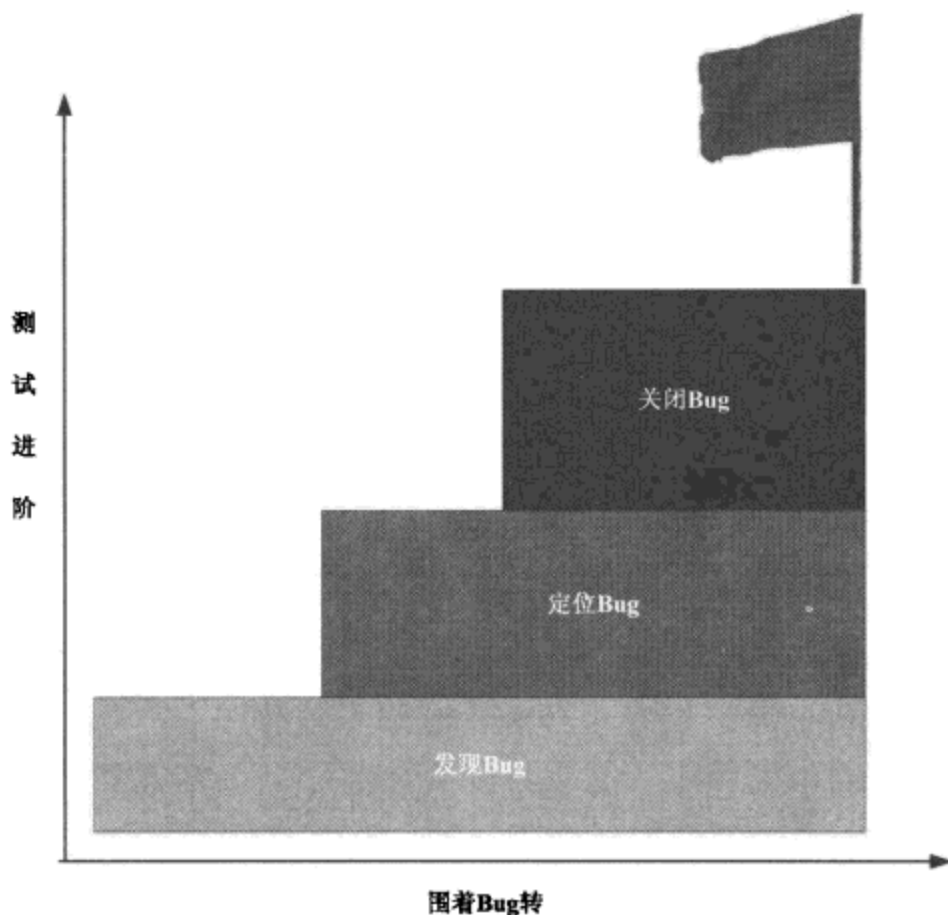


图 2-5 软件测试三重境界进阶图

谈到围着 Bug 转的三个阶段，不禁想起中国近代著名学者王国维在《人间词话》中提到的人生的三重境界：

“昨夜西风凋碧树，独上高楼，望尽天涯路”。

“衣带渐宽终不悔，为伊消得人憔悴”。

“众里寻他千百度，蓦然回首，那人却在灯火阑珊处”。

细细思量，感觉它们之间亦有异曲同工之处。

第一重“昨夜西风凋碧树，独上高楼，望尽天涯路”是说“古今之成大事业、大学问者，首先要树立明确的目标，即使长路漫漫，也下定决心将这条长路走下去。这是一个人在孤独之中寻找理想、寻找生命的落脚点的痛苦时刻”。围着 Bug 转的第一阶“发现 Bug”，同样首先必须有明确清晰的目标，找 Bug 的过程是漫长的，反反复复、枯燥无味是工作的特点，但是为了达到目标“长路再漫漫，也得坚持走下去”，直到找到一堆堆的 Bug。特别是对一些偶现的严重 Bug，重现 Bug 的过程真如大海捞针，但是坚持就是胜利。笔者曾经经历的一个项目中，花了近 1 个月的时间去重现与解决一个严重问题，最后在与开发人员的紧密合作下，终于找到问题的根源。

第二重“衣带渐宽终不悔，为伊消得人憔悴”是说“执着的追求、忘我的奋斗，直至憔悴消瘦，连衣服都变得宽大，这一切努力都是为了心中的梦想”。对应软测中围着 Bug

转的第二阶“定位 Bug”。这一阶段不仅在技术上提出了更高的要求，还要有刻苦钻研、穷追到底，不撞南墙不回头的铁人精神，直到把问题的原因搞清楚才罢休。在国内目前的测试领域，大部分公司这一步并没有要求测试人员来做，但是在国外，特别是一些知名的大公司，如在微软，几乎所有的测试人员都拥有深入调试程序的技能。它除了包含以最短路径重现问题，还要分析问题（例如分析 Bug 会影响到哪些模块），甚至给开发人员提出解决方案。显然，这一步要求测试人员要比开发人员具有更高的设计分析能力、代码调试能力、解决问题的能力。读者朋友，看到这里，对一些测试专业网上常看到的“测试人员是否要懂编程？”这一问题已释然于怀了吧。

第三重“众里寻他千百度，蓦然回首，那人却在灯火阑珊处”。这一阶段是指经过不断磨炼，多次的失败，某一时刻忽然灵犀一点，参透真谛，发现自己想要的东西原来就在自己的身边或领悟后的心里。在旁人看来，他的“蓦然回首”是如何偶然而幸运，但其背后的用功之勤、平时的积累之深，又岂是常人所能坚持，所能想象的呢？这时候，世俗目标是否已经达到已不再重要，重要的是灵魂的解放和心灵的归属。对应围着 Bug 转的第三阶“关闭 Bug”，如果仅从字面理解，很简单，不就是开发解决了 Bug，回归 Bug，然后把 Bug 关闭。如果是这样，笔者认为这种观念仍属于第一阶。第三阶的关闭 Bug，是指测试人员提交一个 Bug 后，要有主动意识推动开发人员解决问题，并协助他们解决，只有问题解决了，软件的质量才得以提高，测试人员的最终目的才能达到。提交的有些问题严格来说，它不属于 Bug，而是一种设计缺陷，此时测试人员该怎么办呢？需主动召集相关专家进行其影响面的风险分析，并跟进此问题的整个解决过程，如果风险点涉及其他专业的更改（如嵌入式软件涉及硬件，机械等方面的知识），可能需要专门成立一个专项问题解决团队，以全面解决此问题，直到各专业方向的问题解决到位，回归验证完成，此 Bug 方能关闭。站在 Bug 的生命周期角度分析，一个 Bug 由被发现的起点，走到被关闭的终点，才是一个合理的、完整的过程，如图 2-6 所示。但是要达到这一层，很可能有一大部分的工作已完全脱离了纯软件测试层面的工作，可是测试的最终目标不就是给用户一个高质量信得过的产品吗？我们需要有这样的大气胸怀，才能把产品的测试工作做得更深远、更宽阔。

接下来结合案例对围着 Bug 转的三个阶段分别进行介绍。

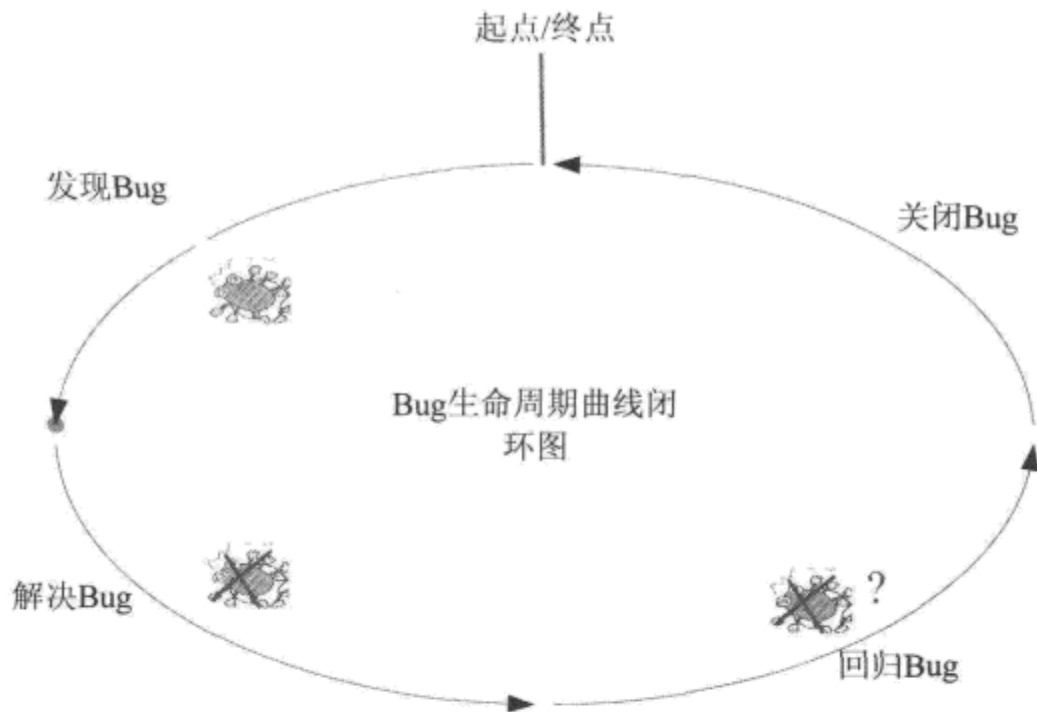


图 2-6 Bug 生命周期曲线闭环图

2.2.1 独上高楼——发现 Bug

【案例】

深圳九月天，仍是骄阳似火，一点都没有初冬的气息。我们的办公室，中央空调照常开着，比较瘦小怕冷的小祝，还穿上了冬装。昨天陈老板刚把我们测试过的版本提交给合作方，今天一早，测试组的 5 个同事都还没接到新任务，大家也就继续测试原来负责的模块。只听有同事在说“如果再发现 Bug，版本都已发了，怎么办呢？”“每个模块大家都已交叉测试过，近 1 周大家都基本上没提什么 Bug，应该没问题了吧！”几句话闲话之后，大家都又忙自己的事了。

整个上午，办公室异常安静，地板是用地毯铺着的，即使是偶尔有同事从身旁走过，轻轻的摩擦声也是若有若无。与往日不一样的是缺少了与开发人员的争论，或许太安静了，笔者反而觉得有点不习惯。忙了一阵之后，抬头看看坐在前面的同事，还发现有人给瞌睡虫咬住了呢。在已发布版本上继续测试已测试了近两个月的模块，真像咬老菜干，越嚼越没味，说不准过几分钟，自己也给瞌睡虫找上门来了。这样的场景，这样的氛围，不知不觉熬过了 2 天，一个 Bug 都没发现，笔者心中难免有几分失意。

第三天，陈老板来上班了，但并没有给我们安排新任务，我们只好继续咬老菜干。为了不让瞌睡虫找上门，也为了能更加集中精力测试那些看似没有 Bug 的软件，笔者在电脑

显示屏下方显眼位置，贴上了 3 个大红字“找 Bug”，如图 2-7 所示。并在心里不断提醒自己“Bug 是找不完的，并不是没有 Bug 了，而是你没有看到它而已”。甚至在假想着找 Bug，就好像挖地雷，每走一步，都得小心谨慎，一不小心，一个 Bug 就从你眼皮底下悄悄地溜走了。也就是带着这种心态，这种认真谨慎，第三天终于有了突破，提交了 2 个 Bug。一个是可能会误导用户的低级错误的错别字，另一个是属于第三方供给我们的手写识别系统的严重 Bug（当时测试的是随手写掌上产品）。由于这一次的“不小心”发现了其他同事模块的低级问题，按老板的苛刻规定，那个月笔者多拿了 100 元的奖金，而对方却扣了 100 元的奖金（那时我们的每月奖金是浮动的，人均 200 元）。在第三方软件给我们提交了补丁包后，老板重新发了版本。不知道后来被老板提升，会不会与这次重要的事件有关。这已是过去很多年的事了，但其中的感悟，在日后的测试工作中一直伴随着笔者，那就是“不管什么时候，测试的目的必须要清楚，Bug 是找不完的，并不是没有 Bug 了，而是你暂时没看到它”。

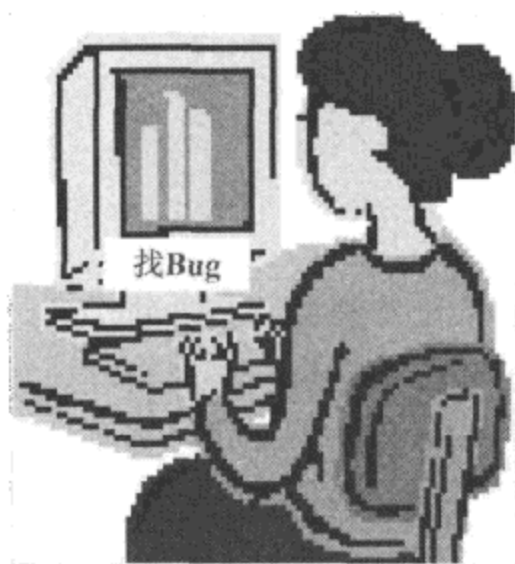


图 2-7 沉浸在找 Bug 中

小贴士：

Bug 是找不完的，并不是没有 Bug 了，而是你暂时没看到它。

或许有感于对测试正确认识带来的好处，后来在一家公司任职时，有幸与另外一些同事一起出题招聘公司的测试工程师。

以下是其中的一道选择题。

软件测试的目的是什么？

供选答案：

- ① 保证软件的质量，提高软件的质量。
- ② 验证程序的运行符合需求定义，是正确的。
- ③ 发现程序中的错误。

后来，我们总结了 86 位应聘者的答案，发现有 52 位的应聘者选择的是①，即近 60% 的应聘者认为软件测试的目的是“保证软件的质量，提高软件的质量”；选择②的有 10 人，约占 12%；选择③的有 22 人，约占 26%。出乎我们意料，有 2 人对此题做了多选，其中 1 人选择了①②③，另外 1 人选择了①③。

由于当时出题时考虑此题是放在最容易拿分的一类题（或叫送分题），但大大超出预期，笔者怀疑出的题是不是存在纰漏。于是特意又找出那本当年伴随着笔者成长，印象深刻的清华大学出版、张海藩编著的《软件工程导论》，打开“第七章测试”，对测试的基本概念再次细读了一遍，其中已很清楚地提到“测试是为了发现程序中的错误而执行程序的过程”。

锁定“软件测试的目的就是发现 Bug”，不管你用什么方法，黑盒，还是白盒，或者是灰盒，仅是方法不同，而其聚焦点是相同的，就好像不同的人拿着不同的探测器去搜寻不同层次的 Bug。有了这个清晰的目标后，所有的前期测试设计工作都需围绕如何高效发现 Bug 而着想，如测试方案的设计、测试用例的设计、测试执行过程的控制。尽管 Bug 数量不是衡量一个测试人员绩效的唯一考核点（因为 Bug 数量的多少与模块的复杂度、开发人员的设计水平、需求的变化等因素直接相关），但业界人士都清楚其所占的分量，可以说已成了大家心照不宣的能力判断标准之一。

2.2.2 为伊消得人憔悴——定位 Bug

站在测试的角度定位 Bug，可以从以下两个方面来理解。

第一，以最短路径重现必发 Bug。

以最短路径重现必发 Bug，也叫精益求精。很多测试朋友常问“什么样的测试工程师算是优秀的测试工程师？”笔者认为，能不能做到精益求精，正是一个优秀测试工程师的特征之一。

成就感，对个人的工作心理状态影响很大，也很重要。在面试测试工程师时，面试官常会问“测试工作中，最使你有成就感的是什么？”基本上都会回答“发现了 Bug”。发现了 Bug，当然是好事，值得高兴，但是发现了 Bug，只是意味着这个 Bug 被暴露出来了，它的生命周期却才刚刚开始。一旦一个 Bug 被发现，它就成了我们的敌人，接下来开发人

员要解决它，测试人员要回归它，直到最后关闭它才宣告结束。

要解决 Bug，首先开发人员要能重现此 Bug，在重现的基础上分析问题，进而解决问题。如果测试人员提交的 Bug 步骤描述中能给出最短的必然路径，可以明显缩短开发分析问题、定位问题的时间。表面上看来只是加快了开发解决问题的速度，但由于开发解决 Bug 的速度加快了，发布版本自然也会更加顺利，实际上对后期的测试也争取了时间。特别是对整个项目的进度能起到积极的推动作用。但是，在测试团队中我们常会看到这样的现象，测试人员一旦发现一个 Bug，便迫不及待地提到缺陷管理库上，有时描述都不清楚，步骤也是含糊地写了一堆。开发人员重现来重现去都没有出来，最后只得找测试人员来重现，发现原来是漏写了某关键步骤，有时甚至测试人员本人也重现不了，免不了被开发人员鄙视一番，只好以尴尬告终。

朋友，你是提交 Bug 的积极分子吗？这很好。但是在提交之前，请忍耐片刻（毕竟不会因为你这个 Bug 晚提交几分钟，成为影响开发解决问题的瓶颈，而最终影响版本的发布），对着自己填的 Bug 单操作一遍，然后再单击那个激动人心的“提交”按钮（因为一旦提交，意味着一个 Bug 的生命周期开始了），某个倒霉的开发工程师又多了一件差使。曾经在一个项目的结项总结会上，笔者亲自听一位资深的开发工程师说“一个偶发的严重 Bug，就好像一把利剑时刻悬挂在头顶上，每天都在揪着我的心，想拿下来却又谈何容易？”开发朋友的肺腑之言，其中更显测试提交必发 Bug，并以最短路径重现的重要性与必要性。

第二，重现偶发 Bug。

实践过的测试朋友，你是否也曾遭遇过不能重现的偶发 Bug？笔者曾经看到某公司对测试人员的绩效考核中有这么一条：“提交的偶发 Bug 不超过本人提交 Bug 总数的 3%”。当然，高要求是对测试人员工作的鞭策，从结果导向来说是件好事。

一次，与一位开发总监论道，他说：“在开发看来并没有偶发的 Bug 存在，之所以我们认为它偶发，是因为我们未找到必发的绝对路径，要么条件不符合，要么触发时机不对等。”其实，在任何一位测试人员心里，都是想把 Bug 重现出来，以节省更改时间。下面通过重现偶发 Bug 的案例，与大家分享通过强度压力测试来重现 Bug 的做法。

小贴士：

通过强度压力测试重现 Bug，是一种耐心与技术的挑战！

下面是一个关于重现 Bug 的精彩案例。

【案例】

某公司研发某精密测量仪器软件，此软件有提供用户用 U 盘导出测量数据的功能。一天，收到一客户的投诉，说在使用仪器的此功能时发现“U 盘导出功能有时会失效，重新拔插 U 盘后功能有效，且这种现象发生过多次”。

接到客户的投诉后，测试负责人马上组织相关测试工程师进行跟踪，经一番沟通、了解，详细的问题描述是这样的：在仪器的 USB 接口上插入 U 盘后，进入软件的数据导出界面，按“导出”按钮，软件提示“没有发现 U 盘！”，但是 U 盘却正安安稳稳地插着，此时拔下 U 盘再插入，再执行导出操作可以成功导出数据。同时，也记录了客户端仪器产品的序列号，由此为索引，在公司的产品追溯库中找到了此产品的软件版本号，硬件相关配置等系列配套信息。

在获悉所需信息后，开发工程师 A 与测试工程师 B 分头行动。开发工程师 A 从相关代码着手分析；测试工程师 B 则反复地重复操作试图重现这个偶发问题。重现这种偶发 Bug 而做一遍遍机械式的操作，绝对是一种耐力的磨炼，这种测试方法，在测试团队中被有人戏称为“魔鬼测试法”。测试工程师 B 这样魔鬼式测试了一天，盼星星盼月亮似的，盼着的“没有发现 U 盘！”提示并没有出现。还好，开发工程师 A 那边排除了一些非软件原因，如 U 盘没有插好、U 盘是否存在坏道等。

第一天没有重现此 Bug，对于测试来说，革命工作仍需继续。工程师 B 绞尽脑汁，用尽了不同的相关发散思维，但是黑暗的道路依然没看到一点光明。时间已是第二天下午的 17:30 分了（18:00 下班）。可怜的 B 心里想着，今天重现的可能不大了，需召集相关专家会谈，想想其他办法。正这样想着的时候，B 的动作放慢了，插了 U 盘之后，并没有着急马上按“导出”按钮，而是不经意之间按“导出”按钮，奇迹也就在这一瞬间出现了。只听，B 惊呼起来：“出来了，Bug 出来了！”

整个办公室的全体测试同事都站起身来，以为发生了什么事。有些同事，还马上凑过去看，好像不相信似的。B 马上说“大家不许动，我要叫 A 来分析现场”。开发工程师 A 查看特意准备的调试信息，以及内部的一些其他信息记录，跟踪对应的代码块，发现相关代码在识别 U 盘的地方存在漏洞。但是却不能下结论说，这个问题一定是由这块代码引起的。开发提出更改后，需测试人员再进行魔鬼测试。

通过两天的魔鬼测试磨炼，测试工程师也增长了经验，写了一个自动化脚本，自动挂载 U 盘，模拟按键“导出”动作，然后卸载 U 盘。晚上时间，让脚本自动触发，连续执行了 1 万次、2 万次、3 万次。后来，用自动脚本，在原有版本同样执行上万次，也没有出来此 Bug。相关人员开始怀疑自动化与人工操作的差别与问题出现有关。暂没想到其他更

好的办法，只好再由 B 在更改后的软件上进行第三轮的魔鬼测试。

苍天不负有心人，第三天 B 在更改后的版本上真再现了此 Bug，证实了开发人员的怀疑是正确的，也就意味着这个 Bug 的真正原因并没有找到。一场欢喜，一场忧，严峻的形势又使他们陷入了迷雾之中。犹如侦探小说中扣人心弦的情节，刚找到的一条线索，却发现是假象，而期望的目标又总是迟迟不能出现。

“专家会诊”共同切脉已启动，从已明确的种种迹象出发，有专家提出应该把重点放在“分析为什么手工连续进行 230~250 次正常的操作时会出现识别不到 U 盘，而自动化脚本执行上万次都正常，它们之间的本质差别是什么？”这一句话，对开发工程师 A 的工作起到了指点迷津的作用，于是把分析方向转向了负责 U 盘识别的硬件驱动上。历经 3 天的分析、调试后，A 终于得出结论“由于负责控制 U 盘驱动的芯片存在一个硬件设计缺陷，此缺陷只有在硬件寄存器复位后且复位前后的值正好相等才发生，一旦发生，则 U 盘识别不到”。

问题的真正原因找到了，解决 Bug 不费吹灰之力，更改后再提交版本给 B 验证，回归通过。悬在头顶上近半月的剑终于取了下来，Bug 销声匿迹，永退江湖了。“U 盘导出”事件，也成了某公司研发团队中的一个众所周知的插曲。

为伊消得人憔悴，定位 Bug 的过程对测试工程师来说，是毅力与技术的双重挑战，正所谓“道路是曲折的，前途是光明的”。

2.2.3 蓦然回首——关闭 Bug

“思想决定行动”，测试设计活动与测试的心理学密切相关，对软件测试的理解不同，对测试工作的要求亦会不同。

一直以来，笔者认为测试就是发现程序中的错误而执行程序的过程，并且把这句话写在工作记录本上的封面，让它每天都能提醒自己。即使换了笔记本后，依然把这句话写上面。刚开始时，这句“警示语”的确很有用，每天工作的目标很明确，就是发现 Bug。曾经，Bug 多的时候，一天提过 30 多个，有种 Bug 多得提不完的感觉（这种情况一般出现在新项目提交测试的前面几个版本）。正如开发同事常说的一句话“测试在笑，开发想哭”，这也从一方面说明，在工作性质上开发与测试本身就是对立的，但我们的目标是一致的，就是把产品的质量做好。

直到有一次，参加了 51testing 软件测试网在深圳举办的软件测试沙龙，一位代表发言提出“测试不仅仅是发现程序中的 Bug，更重要的是跟踪解决这些 Bug，因为只有 Bug 被

解决了，质量才有可能提高，我们的成绩才能得以真正的体现。”多么精辟且恰到好处的总结啊！事实的确是这样，测试提交了 Bug，如何说服开发人员解决这个 Bug，这才是关键。我们的测试工作应当不能仅仅停留在发现 Bug 上，Bug 只有被解决，测试人员对产品的贡献才能真正体现出来。有了这个思路，很自然地在对 Bug 的生命周期管理上、与开发的沟通交流上，我们会变得更加主动。

人的行为受着思维的控制，有什么样的思维就会有什么样的行动，不同的目标导致不同的结果。有了“关闭 Bug”的目标，可以使测试人员在提交 Bug 后主动跟踪 Bug 的各种状态。概括来说，一个测试人员在测试执行过程中与 Bug 的生命周期有着以下的关系。

第一步：发现 Bug，这一点是根本，测试人员在执行测试的过程中产生。

第二步：跟踪 Bug，例如每天定时查看缺陷库上 Bug 的状态（开发人员处理 Bug 后会置它于一个合适的状态）。如果几天后，此 Bug 的状态仍没有变化，主动与缺陷负责人交流解决情况，便是一种跟踪的方法。

第三步：回归 Bug，开发人员解决 Bug 后，测试人员需及时回归，验证是否如预期解决了此 Bug，有没有引出新问题？

第四步：关闭 Bug，如果回归 Bug 通过，即验证确实此 Bug 不存在了，则关闭 Bug，意味着它的生命周期到此结束。

除第二步外，其他都是测试人员自身要做的工作，对个人测试技术上的要求较多。测试人员提交 Bug 后，需要开发人员来解决 Bug。也就在第二步涉及测试人员与开发人员如何有效沟通的问题。一般情况下，当测试提交 Bug 给某模块的开发人员后，对方会在某时间内自动处理。但是，如果测试人员不主动跟踪所提交 Bug 的解决情况，结果常常会出乎意料，如某些 Bug 会被置为“无效”或置为“不解决”。此时很多测试人员可能在心里觉得很委屈，处于第三重境界的“关闭 Bug”的玄机也就在这里了。

对于测试人员来说，当然希望提交的每个 Bug 都能得到解决（缺陷被修正），但是为什么会遇到有一些 Bug，开发人员会认为“无效”或“不解决”呢？一些测试人员会找开发人员理论，但被开发人员说了一通后，就像泄了气的皮球，最后只好很不情愿地“取消关闭”此 Bug。就这样一个 Bug 来了又去了，这当然不是我们想要的（被开发人员置为无效的 Bug 通常可能是非常规操作或是用户难于遇到的场景，较好的做法是置为无效或不解决的 Bug 提交相关专家评估它的风险）。遇到这种情况，有些测试人员会找自己的主管出面，这里充满着沟通的艺术。下面的案例便是一个关于如何与开发人员沟通、解决测试人员问题的典型例子。

【案例】

一天下午，开完部门例会的测试主管陈刚，刚回到座位，测试工程师小楠便过来说她发现了一个关于“数据库备份的严重 Bug”，开发工程师正在解决，但听开发人员说，此 Bug 的更改影响较大。她觉得需要搞清楚设计的原理，以及更改的影响面，这样才能做全面的回归。只是与开发人员的交流中，他们显得很不耐烦，还说测试不必关心设计的细节。这让小楠很难受，同时她也当面指出他们这样做是错误的。

看小楠委屈得眼泪都要出来的样子，陈刚表示能理解她的心情，并说往深入、全面方面分析 Bug 是对的。而工作交流中关于开发人员很不耐烦的事，可能是在沟通过程中存在的一些误会。

第二天，陈刚拿着一张纸（上面写着几个关于数据库设计方面的问题）、一支笔去找开发工程师，首先说明是向他请教某数据库设计方面的问题，开发人员看陈刚一副真诚谦虚、好学的样子，就一口气地讲了起来。讲完后，陈刚微笑着赞美他把设计的東西讲得那么清楚，同时也提出自己对如此重要的设计只理解一点点，并不全面，并说自己及测试这边都很想学习全面的细节设计流程，能邀请到他来给测试的全体人员进行一次宣讲，对产品的质量来说真是太重要了。经怎么一说，开发人员最后笑着说：“没问题，我自己开发设计的，一定能讲得出来的。”

一场不愉快的沟通化解了，事情走向了美好、有效的结局。

小贴士：

卡耐基慧语：如果你想赢得人心，首先要让他相信你是最真诚的朋友。

2.3 测试的第二重境界：站在 Bug 之上

测试的价值仅仅是发现 Bug 吗？通过“站在 Bug 之上”测试第二重境界的介绍，希望能帮助读者正确理解测试的真正价值是什么，在实际工作中如何操作以体现这些价值。不同的理念，将会牵引着测试人员朝不同的方向迈进，“站在 Bug 之上”可以拓宽测试人员的视野，找到更多可以充分体现测试价值的测试链，让测试人员为项目的成功做出更大的贡献，从而带来更宽范围的测试成功。

2.3.1 测试的价值不仅仅是发现 Bug

一提到测试，大家马上会想到 Bug，测试仅仅就是为了发现 Bug 吗？这是值得我们思考的问题。

从软件测试最基本的定义出发，早在 1979 年 J.Myers 在《软件测试的艺术》一书中提到：

- 软件测试的目的就是尽早发现 Bug。
- 一个成功的测试就是发现了至今为止尚未发现的 Bug 的测试。

总之，测试就是为了发现 Bug，测试所做的工作无一不是围绕 Bug 而展开，如图 2-8 所示。测试发现 Bug 越多，越自豪，越有成就感，这些观点已几乎根深蒂固地扎在了我们的心里，测试除了发现 Bug 真没其他事情可做吗？

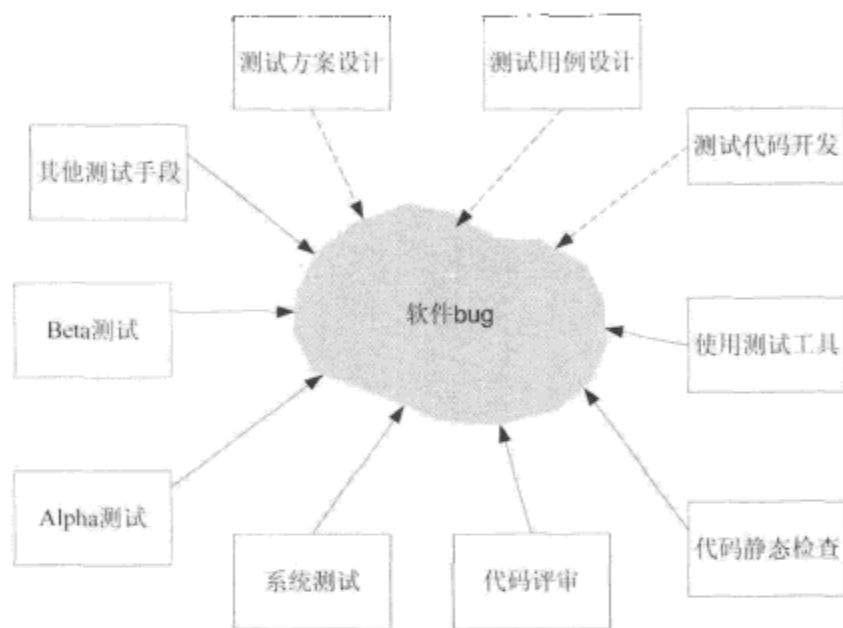


图 2-8 以发现 Bug 为核心的测试机制

发现了很多 Bug，测试人员高兴了，但老板肯定是不高兴的。很明显的道理，为了解决这些 Bug，他必须付出更多的成本，包括开发人员与测试人员的工资，更严重的还可能影响产品交付市场的时间。商场如战场，在商海中竞争，时间就是金钱，时间能给产品带来更多的市场空间，为企业赢得更多的利润。理解这些商业知识能帮助我们做正确的事，并且正确地做事。认识到这一点后，相信测试朋友就不会再为某个 Bug 还没有解决，版本却上市而耿耿于怀了。测试应该跳出仅发现 Bug 就沾沾自喜的圈子，看到项目整体，站在公司的角度想测试可以做什么，只有项目成功了，公司才能获得利润，最终达到员工与公司双赢的目标。

质量、成本、时间是项目管理的三要素，它们之间三足鼎立，稳如泰山，即质量好、

成本低、工期短，这样的项目当然是项目经理求之不得的。但一直以来它们都是矛盾地存在着，形象地看，它们犹如一个等边三角形，如图 2-9 所示。对其中的任何一个元素处理不当，三元素的三角关系就会变得不稳定，将给项目的成功带来风险。

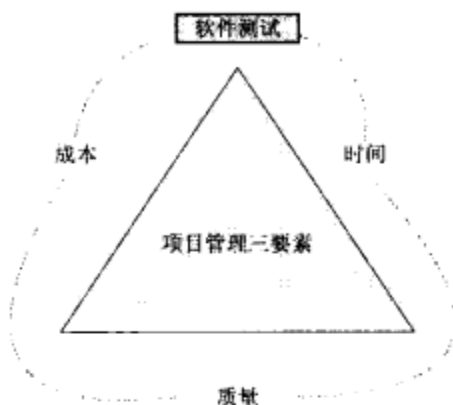


图 2-9 测试与项目管理要素关系图

软件测试团队是整个项目团队大家庭中的成员之一，在软件质量上把关，要尽可能早、尽可能多地发现 Bug。这也是软件测试成立的根本，是质量上能给项目做出贡献的地方。那么在成本与时间的控制上，测试可以做些什么，要如何做呢？也就是前面提到的测试如何配合项目的成功做正确的事，并且正确地做事。

小贴士：

做正确的事与正确地做事

做正确的事出发点是企业利益最大化，而不是站在个人和小团体的立场去做事，也不是怕承担责任，把事推给别人。要求我们在众多的可能性中选择，辨别出什么是正确的，什么是最直接、最可行的做事方式和方法，把企业利益最大化作为办事的标准。

正确地做事，是驱动具体做事的人员如何按照领导的意见去做事，而不去考虑是否符合企业利益最大化的原则。

对于测试，做正确的事就是站在用户的角度，进行常用功能（模块）重点测试，而避免非常用功能的过度测试，浪费成本，包括人力与时间的投入。正确地做事，就是采用合理、全面的测试方法验证软件是否符合用户需求，不想当然地通过用户根本不可能用到的非法操作或后门进行验证。下面讲述关于软件测试的 2-8 原则，通过此 2-8 原则，可以使软件测试在项目的成本与时间的应用上做到效益最大化。

举个大家在日常生活中常遇到的例子，如经常看到广告上说，现在的手机软件的功能

如何强大，软件的功能如何丰富，但每一功能用户使用的频度都一样吗？回答是否定的。这就有了在软件测试范围侧重点上存在的 2-8 原则，即要把 80% 的精力放在测试 20% 的重点功能上。从用户角度出发，这是值得的，也是需要这样做的。

首先，分析在我们的软件系统中，哪些功能对用户来说是核心且重要的功能，然后安排合适的测试工程师负责这些模块。设计出的测试方案、用例进行重点评审，测试执行过程重点跟踪。每一次软件版本发布时，即使没有更改此部分的代码，也对它们进行回归测试（这种回归需讲究策略与方法），因为它们太重要了，不允许有错误。

下面是软件测试 2-8 原则的详细内容。

1. 80% 的错误是由 20% 的模块引起的

简单、容易的模块或功能是很少引入过多 Bug 的，而对于存在复杂逻辑的一些关键模块往往会引起系统 80% 的错误。只有关键模块稳定了，整个系统才可能真正的健壮和稳定。

这个原则对于测试来说就是站在用户角度（而不是研发实现的角度），正确地选择重要功能模块作为测试的重点，不偏离方向。

2. 80% 的测试成本花在 20% 的软件模块中

设计测试用例时，常会用日产多少条用例来衡量工程师的工作，用例的多少与需求量有关，而往往影响软件架构设计的需求描述是比较少的。这种情况下，设计测试用例时特别需要结合软件的概要设计、详细设计一起考虑。如果用例设计人员为了达到用例的数量，通过大量复制用例，修改个别字眼，而没有真正去设计高效的测试用例，那么用如此低效甚至更多的用例数量来对待复杂的 20% 的核心模块，在测试执行过程中必将导致一部分关键 Bug 找不出来。

3. 80% 的测试时间花在 20% 的软件模块中

对于复杂的模块，前期的测试设计和思考可能会耗费大量时间，而产出的用例量可能并不大。对于复杂的系统，特别是对于全新系统，必须舍得投入充足的时间来优先考虑设计，前期方案、用例设计的时间越短，后期的风险越大。

在项目进展到一定阶段后，增加人力并不一定能解决缩短时间的问题。例如，如果复杂且核心模块在项目的后期才开始执行测试，由于 Bug 较多，而项目又需要短时间把版本稳定下来，通常的做法是加人。然而加入的新兵需要一段时间的熟悉期，必要时还需老兵来带，这本身已影响到老兵的工作。另外一些性能测试、自动化测试工作也只有等版本稳定后才会有更好的效果。

下面是一个关于测试时间与重点模块该如何分配的案例。

【案例】

背景：某软件系统，包含有 5 个功能模块，分别叫模块 A、模块 B、模块 C、模块 D、模块 E。从客户使用的业务功能出发，模块 B 是产品必须有的，且是用户使用频率最高的功能模块；模块 A，用户使用频率不高，但对数据的正确性要求特别高，存在些许错误，就可能带来法律法规风险；模块 C、D、E 主要是为核心业务或为后续产品的维护活动而使用的支撑模块。从需求的复杂度及描述的篇幅来看，模块 B 的需求虽篇幅不算长（需求点不算多），但比较难于理解，隐含需求较多。模块 A 的需求描述虽然显得特别简单，只有几句话的要求，但其对软件设计的需求依赖性较大；其他模块由于是业务支撑模块，有些功能是用来支持安装升级及产品的后期维护服务使用的，需求描述很细，内容篇幅较多。

项目的测试经理陈 A，并不是不清楚这些模块的用户使用需求，但他是一个完美主义者，在有时间与人力的情况下，总是想把所有模块的 Bug 尽可能多地找出来，始终坚守质量第一，认为这是测试能给项目带来贡献的核心所在。于是在项目的初期测试人力不紧张的情况下，根据用例须对需求达到 100%覆盖的原则，做了如表 2-2 所示用例设计工作的安排。

表 2-2 模块用例设计工作安排

任务（用例设计）	投入成本（人天）
模块 A	5
模块 B	10
模块 C	10
模块 D	10
模块 E	10

用例设计完成，各模块的用例按流程完成了评审。进入执行阶段后，各模块的测试工程师用自己设计的用例开始测试执行的工作。如表 2-3 所示是一周后各模块发现严重 Bug 的情况表。

表 2-3 各模块严重 Bug 比例

	用例总数	测试工程师	用例执行数	提交 Bug 数	严重 Bug 数	严重 Bug 比例(%)
模块 A	500	T1	500	5	1	20.00
模块 B	1000	T2	500	30	12	40.00
模块 C	1000	T3	800	10	2	20.00
模块 D	900	T4	750	8	2	25.00
模块 E	1200	T5	900	10	1	10.00

模块中严重等级的 Bug 越多，表明质量越差，意味着离版本稳定时间越远。据最佳实践表明，一个存在 10 个以上 Bug 的模块（只有几百行代码量的小模块除外），如果严重等级的 Bug 大于等于 30%，则表明该模块设计质量堪忧，在质量的关卡上已亮起了红灯。是设计架构的问题，还是编码本身的错误，或者是其他原因等，需要及时启动原因分析，以拿出合理的解决方案。

从表 2-3 的数据可看出，模块 B 的 Bug 严重性比率已超出警戒线，其他模块看似正常。于是测试经理陈 A 在接下来的一周时间里，对各模块的人力资源分配进行了局部调整，如表 2-4 所示。模块 A 的测试用例已执行完成，测试工程师 T1 被调整到与测试工程师 T2 一起测试模块 B。由于模块 C、D、E 用例还余下不少，所以负责相关模块的工程师留下来继续测试未完成的用例。而在第 2 周时，开发人员对上周提交的 Bug 有些已解决完成，测试人员在第 2 周需预留足够时间回归 Bug。另外，测试过程中也存在补充或修改用例的工作。

表 2-4 人力资源调整后任务分配与提交 Bug 情况表

	一周后余下用例总数	测试工程师	提交 Bug 数	备 注
模块 A	0	\	0	回归已解决 Bug 及补充或修改用例
模块 B	500	T1+T2	50	
模块 C	200	T3	2	
模块 D	150	T4	1	
模块 E	300	T5	1	

进入测试执行的第 3 周后，陈 A 做了一个小结，如表 2-5 所示。

表 2-5 测试任务完成成本预计

	用 例 总 数	提交 Bug 总数	开发人员解决 Bug 时间预计（人天）	测试总耗时预计（人天）
模块 A	510	5	1	6
模块 B	1200	80	15	22
模块 C	1000	12	3	9
模块 D	9200	9	2	7
模块 E	1150	11	2	8

表中数据表明，除模块 B 外，其他模块的测试任务看似都可以在第 2 周结束，甚至模块 C、D、E 的测试工程师的工作量并不很饱满。而模块 B 继续呈现警戒状态，Bug 居高不下，当用例执行完成后，就提交 Bug 开发人员解决，测试人员再次回归，直到所有 Bug 关闭，预计的时间远远超过其他模块的工作量。而在第 3 周，测试工程师 T1、T2、T3 已

属闲置状态，但如果在第 3 周后再调入支持模块 B 的测试，为时已晚，因为用例已执行完成，回归 Bug 最好由发现 Bug 的人来回归是最合适的。也就是说，本轮提交测试的 5 个模块，需要在 3 周后才能完成测试。

站在项目的角度，上面 5 个模块的测试在两周内完成也是可以的，一方面从第 2 周开始从测试模块 C、D、E 的资源中调入一个加入到测试模块 B，同时开发人员解决 Bug 的人力及速度需紧密配合。

而在实际的工程实践中，情况要复杂得多，比如遇到模块 B 需重构，测试完成的时间需重新评估；已解决的 Bug 并不一定彻底解决了，可能会重打开或产生新 Bug 出来，陷入一种拖泥带水的状况，拖了几个版本后，才会彻底改观，等等，都是常有的事。

几个月后，项目以延期 1 个月左右的时间上市了（尽管延期的原因可能不仅仅在软件上），而让陈 A 万万没想到的是，原来表现最少 Bug、最稳定的模块，用户端却反馈回一个关于数据正确性的错位问题（条目 A 的信息显示在条目 B 的地方了）。此问题很严重，存在法规风险。回过头来查看，用例确实没有考虑周全。明显对用户真正使用需求理解不足，重视不够。由于此产品的错误信息一旦被使用会涉及人身安全，所以如果用户真打起官司来，公司的损失可就大了。

上面的例子，一方面说明了测试对重点且复杂的模块认识不足，即使是后来出现严重警戒时，只是做了局部调整，致使后期有部分资源浪费，而某些资源却很紧张，最后拖延了项目的研发周期。另一方面是测试对用户的真实使用需求，以及出现问题后所造成的严重后果认识不足，在用例设计上出现遗漏，直接影响了软件的质量。但是，只要我们对测试工作进行科学的思考，不盲目地干活，把测试的 2-8 原则落到实处，那么测试执行工作完全可以在提高关键质量目标的同时，为项目的研发降低成本、减少时间，并全面真正支撑起公司商业成功所必需的关键要素——更快的进度、更低的成本、更高的质量。这时，你还会认为“测试的价值仅仅是找 Bug 吗？”

2.3.2 测试的服务链

2.3.1 节介绍了软件测试作为软件专业方向的一股力量在质量、成本、时间的合理控制上能促使项目成功。实际上，一个全新的项目或产品的研发，有一个长长的开发链，如图 2-10 所示。当跳出“测试仅仅是发现软件 Bug”这个圈子时，会看到“山外有山，楼外有楼”，软件只不过是产品的一个子系统，要获得项目的成功，整个体系的各环节需紧密合作，相互支持。软件测试的服务对象虽然重点是软件，但不仅仅是软件，还会涉及机械、

硬件，如软件在整机环境下的测试，需要这两个专业方向子系统的支持。反过来，它们有时也需软件给予支持，如协助开发某调试硬件部件的软件，方便相关组件的现场调试。除此之外，软件与工艺设计、生产、市场营销、客户支持也存在一些直接或间接的关系，如市场为了参展，需软件提供参展的特殊版本；用户支持需要软件帮他们实现简易的安装升级程序等。产品研发链上的各环节，与软件都存在或多或少的关联，测试自然也就需考虑服务于这些特殊用户的需求。

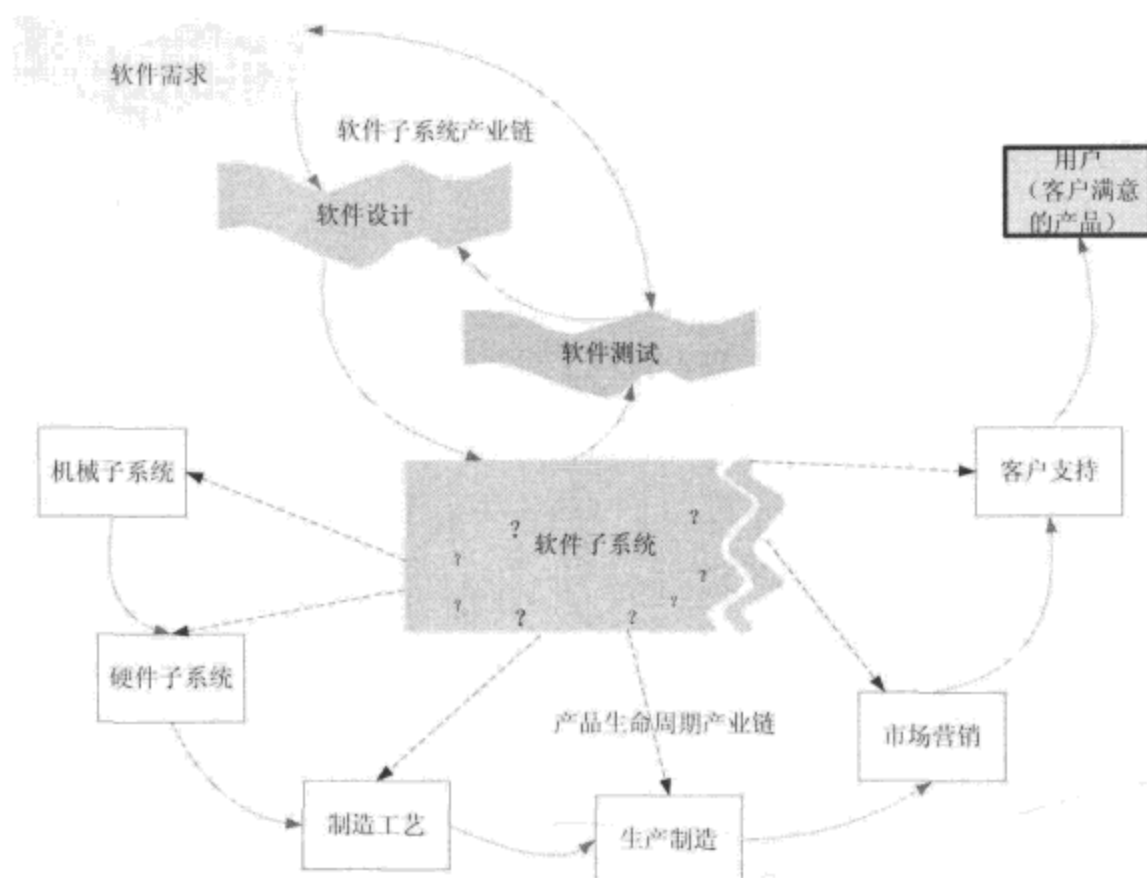


图 2-10 与软件直接或间接相关的产品开发链

“软件是产品的核心”，回顾我们身边林林总总的产品，像手机、PDA、数码相机等消费性电子小产品，大一点如手提电脑、桌面台式电脑，再大一点的如系统服务器，各式各样的医疗设备、军工产品等，如今哪里都是软件在主宰着。一个有趣的事，在整机上测试软件时，会遇到测试出来的 Bug 最后定位不是软件的问题，而是硬件或机械设计的问题。所以软件测试实际上并不仅仅是发现软件的问题，它对整个产品的质量都有不可估量的贡献。

2.4 测试的第三重境界：挑战零缺陷

孔子说“人无远虑，必有近忧”，用在软件测试上，是什么意思呢？可以这样理解，

如果我们不从发生问题的根源上解决问题，认为测试仅仅是找 Bug，千方百计找 Bug，觉得 Bug 总是找不完，意识中就会陷入“永无天日”的状态。然而，有小部分测试人员还真希望软件存在多一些问题（唯恐天下不乱），这样可以多提交 Bug，认为业绩比没有提交多少 Bug 肯定要好。无独有偶，有小部分开发人员也把自己犯下的程序错误，视为理所当然，甚至还有个别别人还会戏虐地说“软件如果没有 Bug 的话，测试人员不就失业了”，这好像在唱一出双簧戏。软件开发的整个过程中，Bug 是理所当然要存在的，是这样吗？软件工程中软件危机的根源问题只能通过找 Bug 的手段来控制吗？

实际上，我们都很清楚，任何一个 Bug 的产生，它都是有来源的，来源包括需求的设计、软件的设计（含代码的编写）等。相对于前端的设计，测试是事后的验证，是一种“堵”漏洞的措施。然而，在实际工作中，时间与成本并不允许我们去堵住所有的 Bug。日本质量大师田口玄一说得好“质量是设计出来的，而不是测试出来的”。如果我们能变被动为主动，在设计之前，就推动做好设计的防患措施，为设计高质量的软件打下坚实的基础。这便是本节打算向读者介绍的测试的第三重境界：挑战零缺陷。

2.4.1 缺陷的防与堵

几乎在每次面试测试工程师时，笔者都会问一个这样的问题：“你所负责测试过的模块，是否存在漏测的情况”，也几乎每个应聘者都回答说“有”。面对复杂的软件，纷繁复杂的运行环境，在有限时间内进行的测试活动，做到真正的零 Bug 是不可能的，也是不现实的。但这些都不是理由，所有的测试活动都是有目的的商业活动，每个公司都有自己测试通过的一套标准或原则。虽然漏测不可避免，但并不是说漏测是一种正常现象或应该的现象，出现的漏测问题如果超出公司所能接受的原则，就属于不正常的现象，很有必要进行漏测分析。进行漏测分析活动（需要特别注意的是它绝不是对漏测人员的批斗会），它的主要目的是通过分析过去的教训，找出问题的根源，分析测试中哪个环节工作存在缺失，以拿出规避的可操作的措施出来。

测试人员进行漏测分析时，免不了对问题进行追本溯源，软件是由开发人员设计出来的，所以漏测分析活动少不了开发人员在场，甚至有时还会涉及需求设计人员。关于漏测分析的追本溯源，这里有一个关于开发与测试之间的工作关系像修堤坝一样的有趣比喻，如图 2-11 所示。开发人员设计软件就像修筑一个堤坝，如果堤坝在结构上存在问题，当洪水冲击时，可能不只是局部的泄漏，而是直接的决堤，犹如软件的崩溃。高高的堤坝，难免会存在漏水的小洞，或渗水的小孔，就好像软件中存在的小 Bug，越是在堤坝基部的漏

水或渗水问题越难发现，解决的代价也越大。

在设计时就把结构搭牢，不存在漏洞当然更好，这是一种防范。而如果防范出界，把设计带出的大洞小孔遗留到测试手上，测试只好拿着各种放大镜（使用各种方法）来检测，以网罗各种深深浅浅、大大小小的问题，最后通过“打补丁”的方式，堵住堤坝上的“百孔千疮”。

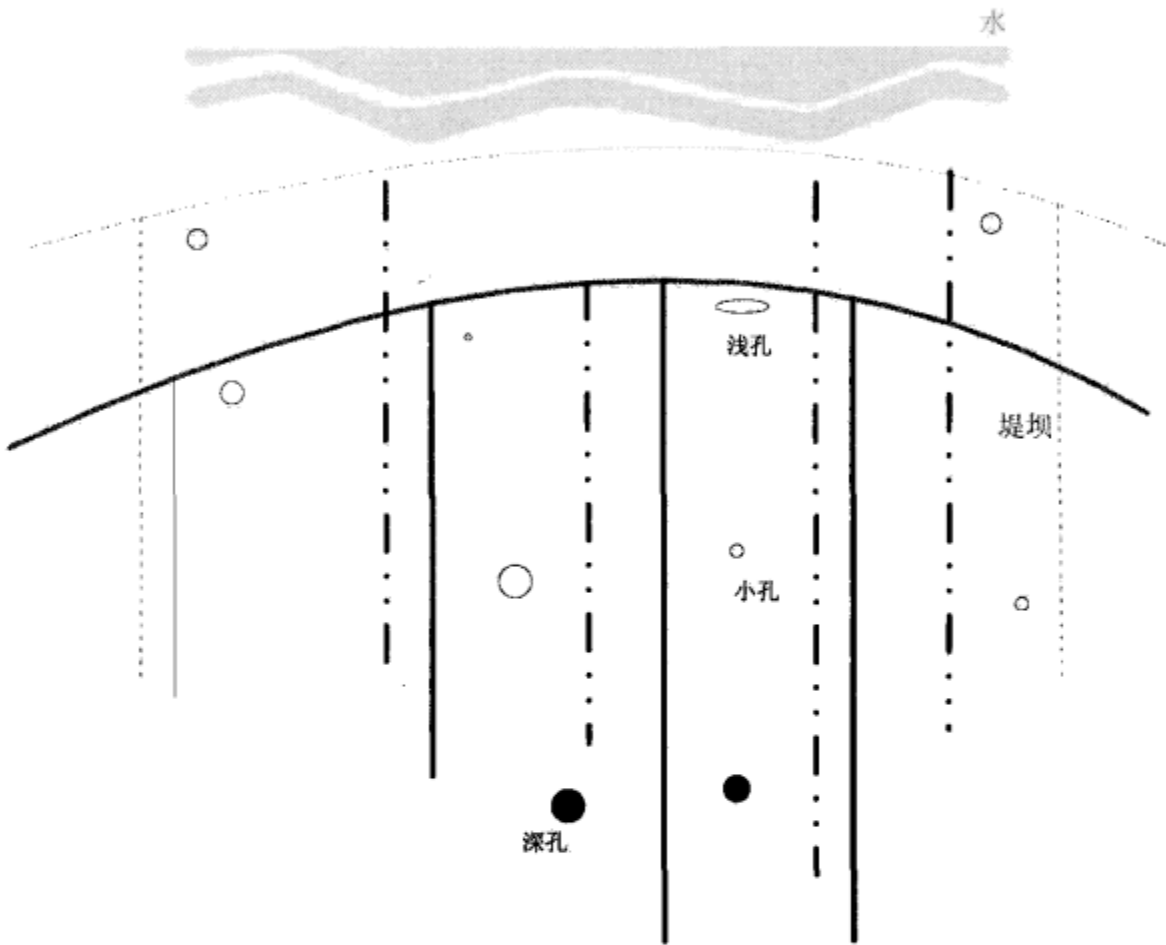


图 2-11 缺陷的防堵与堤坝的防堵形象理解示意图

在对缺陷的防与堵上面，测试是发现问题的中间角色，告诉开发人员哪里漏水或渗水了，防与堵的工作都是由建堤者来做的。当然，防是主动的，堵是被动的，主动变为被动后，中间经历了资源与时间的投入，诚然即使是同一个 Bug，它们的代价也是完全不一样的。这种堵越在后面，影响越大，代价也就越大，如表 2-6 所示（摘自《代码大全》）是一个根据缺陷出现的阶段来增加测试成本的例子。

表 2-6 根据缺陷的引入和检测时间，修正同一缺陷所需的平均成本

引入时间	需求	体系结构	建设	系统测试	发布之后
需求	1	3	5 ~ 10	10	10 ~ 100
体系结构	--	1	10	15	25 ~ 100
建设	--	--	1	1	10 ~ 25

如表 2-6 所示为在需求阶段引入的一个缺陷，如果立即发现了此问题，修改成本只需要 1 美元，但如果在系统测试阶段发现它，修改成本就增加了 10 倍。更为严重的是，如果在版本发布后用户端发现了此问题，则需付出 10 倍以上，甚至是 100 倍的代价。缺陷在系统中的时间越长，解决它的代价就越高，因为时间越长，开发与测试人员修改的成本就越高，还将影响大面积的用户端升级。

小贴士：

漏测：指软件在测试人员完成测试后，在后续时间被自己或其他人发现仍存在不该出现的缺陷。这里的他人包括其他内部测试人员、公司内部用户、终端客户。

2.4.2 “零缺陷”文化

在 2.4.1 节的例子中，我们已经很清楚地知道一个缺陷自它诞生后，如果发现得越晚则修改成本越大。那么有没有可能缺陷根本就不发生，或是控制在一个可接受的状态呢？

早在 20 世纪 60 年代初，被誉为“全球质量管理大师”的菲利浦·克劳士比 (Crosbyism) 提出了“零缺陷” (Zero Defect) 思想，并在美国推行零缺陷运动。后来，零缺陷的思想传至日本，在日本制造业中得到了全面推广，使日本制造业的产品质量得到迅速提高，并且领先于世界水平，继而进一步扩大到工商业所有领域。如今，“零缺陷”已经成为国际卓越企业的工作标准。

零缺陷，对软件来说，是神话还是另有其意呢？在大多数人的印象中软件常常是有一堆堆 Bug 存在的，而迫于市场竞争的压力，又不得不面市，然后通过不断发补丁包升级来解决不同的漏洞。一些业界朋友还常自嘲“国际软件巨头微软的软件尚且如此，更何况我们这些小兵开发的无名软件了”。实际上，克劳士比的“零缺陷”并不是说绝对没有缺点，或缺点绝对要等于零，而是指要以“缺点等于零为最终目标，每个人都要在自己工作职责范围内努力做到无缺点”。它要求生产工作者从一开始就本着严肃认真的态度把工作做得准确无误，在生产中对产品的质量、成本与消耗、交货期等方面进行合理安排，而不是依靠事后的检验来纠正，它特别强调预防系统的控制和过程的控制。2.4.1 节中关于建堤坝的防与堵所带来的成本消耗差异也正说明了这一点。

我们总是在默默地埋头苦干，想着找到更多的 Bug，认为找到更多的 Bug 就是胜利在望。然而，常言道“饮水思源”，必须找到 Bug 的源头，如果不从源头上控制，就像一个多病的孩子，即使把所有 Bug 找出来（实际上不可能），也是治标不治本。说不准哪一天

加入一个新需求点，在实现后又带出一拨新的 Bug。就这样，软件改了测，测了又改，周而复始，版本发布的时间拖了又拖。下面的小故事就是一个关于追本溯源的典型事例。

【小故事】

有一天，一个村民在村边的一条河边上走，看到有个人在河里快要淹死了，他跳进河里，把这个人救了上来。还没等休息，他看到河里还有一个人，他一边喊着，一边又回去接着救人。河里出现了更多的溺水者，更多的村民也都被召集来加入救人的行列。在一团混乱中，有个人走开了，沿着河边的一条小道朝上游走去。一个村民叫住他问：“你去哪儿啊？我们需要你的帮助。”他说道：“我要找出是谁在把这些人扔进河里。”

在软件开发的周期里，是谁把 Bug 扔了进来？同样地，我们需要追本溯源，找出问题的根源，分析后提出防范措施，并加以执行。你也许会说这已超出测试的工作范围，应由公司的专门质量管理团队来负责，但是质量管理团队也需我们的协助。从测试技术的角度出发，测试可以提出更多、更实际的软件设计质量防控的措施，所以，通常测试团队会承担着管理质量的某些方面工作。

“零缺陷”是一个体系，不是依赖于某个人某个团队就能做好，需要围绕产品的整个开发链的所有团队所有人都参加进来，同时需要有高层领导充当质量的倡导者。

2.4.3 “零缺陷”后的误区

有不少人都有这样的担忧，公司实行零缺陷运动后，是不是意味着测试人员该下岗了。从 2.4.2 节零缺陷的定义中，我们便可以否定这个问题。但是有一点是肯定的，那就是实行零缺陷运动后，软件中的 Bug 将变得少了，测试的难度将变强，主要的精力将会消耗在找深层次的 Bug 上面。

零缺陷后，测试中的有些人可能开始担任质量保证角色或其他角色，如转去做需求，转去做 QA，这也是可能的。这些人将会在分析和实施过程中改进缺陷预防技术，成为零缺陷运动质量文化的大使。

如今大多数软件都复杂、庞大，尽管“零缺陷”听起来很美妙，执行后的结果也好像很好，然而软件是由人做的，人不可能不犯错误，软件测试作为软件开发过程中质量把关的最后环节是少不了的。即使在微软，也同样是用测试来确保产品质量的。

第 3 章

测试设计景观

设计，大家都很熟悉的词儿，它是一种复杂的思维活动，类似地，测试设计是专为测试活动服务的脑力劳动。本章先对设计与测试设计的名称进行了咬文嚼字式的剖析，希望能更好地帮助读者理解测试设计的本质含意。接着分别以测试组织模式的管理设计、测试流程的设计、测试技术的应用设计为纲，结合案例讲述如何应用、体现测试设计是怎么回事、它对测试工作有哪些重要影响。通过本章的介绍，希望能帮助读者理解为什么说“测试设计是一个过程”的核心理念。

关于测试设计的话题，当今业界朋友们的讨论焦点主要在测试用例的设计上。毋庸置疑，用例的设计是测试设计在技术上的核心所在，在测试领域除了技术线路上的设计，还存在着围绕测试工作的测试流程与管理上的设计，使它们构成了一个整体，是事、人、技术、流程的统一体。

- 事：一件要完成的任务，可能是某个项目或某个模块的软件测试任务，它是测试工作的驱动源。
- 人：有了任务之后，需要由人来完成，即便是采用自动化测试，仍需有人的参与。
- 技术：如同软件设计一样，软件测试也是一项技术性很强的工作，如何能更好、更快地完成测试任务，与测试策略的制定、测试方法的选取、测试方案与用例的设计有着密不可分的关系。
- 流程：人是活动的个体，人与人之间也是千差万别的。如今，任何一个公司或组织都是一个团队在工作，人作为个体如果没有流程的制约，组成的团队将会变成一盘散沙，离有效、快捷协作相去甚远。
- 管理：实际上，上面的这些因素的考虑，都离不开两个字“管理”。管理分技术上

的管理和人与事的非技术上的管理，但所有这些都是围绕“事”来运行的。配合公司的发展，管理需有前瞻性，这里指的是测试领域的前瞻性思考，测试未来的洞察力。

3.1 放眼设计

在讲述测试设计之前，我们先来看一下“设计”的含义，理解了“设计”一词，再理解“测试设计”这个词将更容易。

设计，英文为 **Design**，它是一种跳跃性或者是逻辑性思维的某种冲动，是大脑对思维的一种具象化，也就是我们通常所说的创意。如图 3-1 所示是表征设计思维过程空间的示意图。从图中可以看出，设计是一个脑力劳动的过程，这个过程是由若干问题引发的，为了解决这些问题，会有一些零碎的思考结果片段，最后组合它们，从而形成一个完整的设计包。设计活动是有目的、有计划的创作行为，这种行为大部分为商业性质，少部分为艺术性质。也就是说，大部分的设计最终都是为解决某些商业问题而服务的，如广告设计，设计出的广告要为公司产品宣传服务。

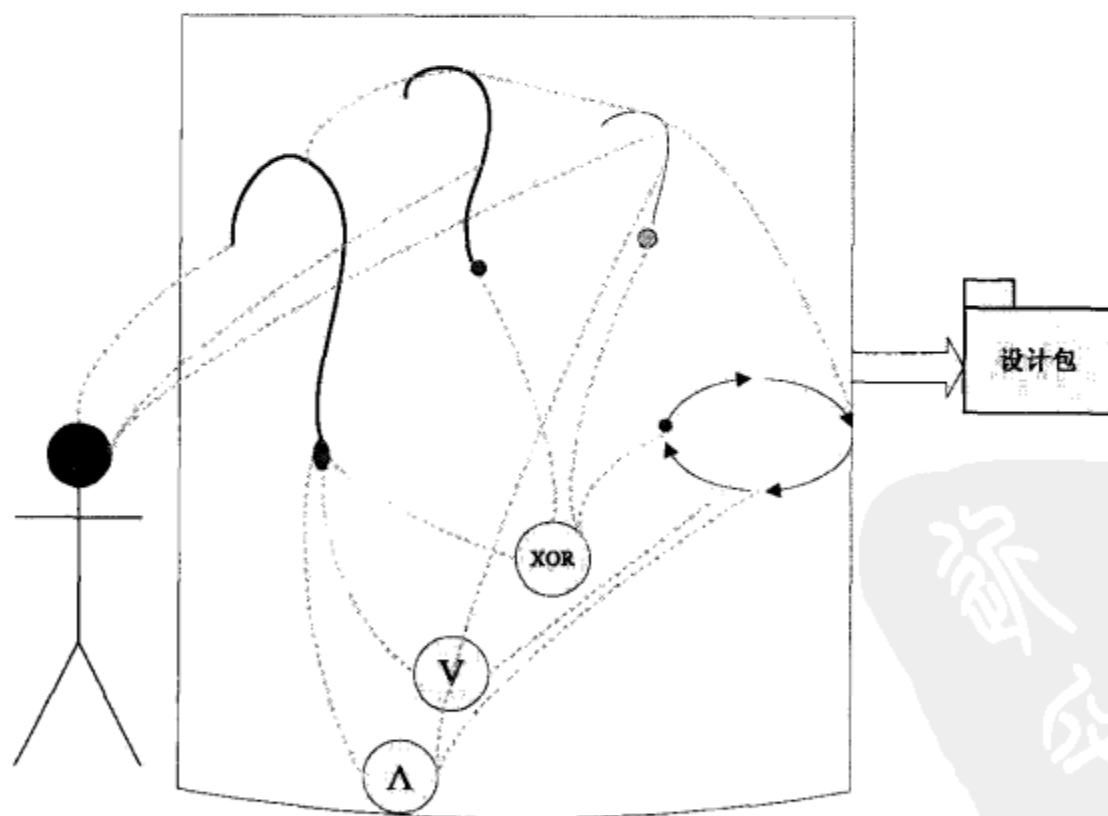


图 3-1 设计思维过程空间示意图

设计是一门职业，例如在电影业中有场景设计一职；在印刷业中，有包装设计一职；在工业设计中有平面设计师一职，等等。类似地，在软测领域，也有专门的测试设计工程

师一职。以设计为职业的社会环境通常就叫做设计界。设计界在欧美国家发展历史悠久，故设计史和相关理论常以欧美的工业设计、建筑设计为两大主流。在西方，大型的设计系统，往往以 Architecture（建筑）来称呼。这里指的建筑并非具体的建筑学，而是一种抽象的形容。

有很多从业人士认为设计就是技巧，其实不然。设计与技巧是有本质区别的，就好像一个画家与画师的不同境界。而不同的境界在表现手法、表现技巧及思想意识上的表达会不一样，以致最后画出的画在整体效果上会有不同的景观和气韵。在深圳龙岗有个闻名中外的油画村——“大芬村”，那里有一百多家画廊，每家画廊都有画师或画工，但不一定有画家。在大芬村，画家一般售卖自己的原创作品，他们一般不画行画，也很少临摹他人的作品。画师以制作行画为主，油画技巧娴熟，也称为师傅。他们没有原创作品，带着一批画工或画徒临摹名作，在画廊中工作。画师犹如企业中的技术骨干，从业的时间长了，正如“熟读唐诗三百首，不会作诗也会吟”，画师对画画的技巧运用，如笔法、色彩效果、原材料特性等都了如指掌，临摹出的作品足可达到以假乱真的程度。但是离开临摹的作品，如画写生搞创作，多是力不从心。而画家给自己的作品赋予思想，充满着哲学的韵味，它是画家所具有的方方面面素质的综合反映，是一种修养与品味，这一点是他人临摹不出来的。

有一位设计界的前辈讲设计即思想，是设计师专业知识、人生阅历、文化艺术涵养、道德品质等诸方面的综合体现。那么，测试设计与设计有什么区别与联系呢？接下来将介绍。

3.2 解读测试设计

测试设计，可以理解为它是对测试工作进行有目的、有计划的、创造性的商业活动，这种创造性活动与设计者本身所掌握的测试技术及拥有经验的丰富程度密切相关。测试界大师 Glenford J. Myers 在《软件测试的艺术》一书中说到“测试是极度富有想象力和高智商的、有挑战性的工作”。实际上，很多既设计又开发软件的人承认，他们通过创造力的软件测试获得了更多艺术上的满足感。

谈起测试设计，大家可能首先想起的就是测试用例的设计，因为无论采用何种方式来测试，测试用例都是离不开的。一条高效的测试用例，如同一面明亮的宝镜，很容易暴露隐藏的 Bug。特别当版本稳定，容易发现的 Bug 很少，余下的都是一些隐藏得很深的偶发严重 Bug 时，此时再往下挖掘如同大海捞针。由于软件本身的特殊性，以及在实际的项目

测试中还要考虑时间与代价，不允许我们做到 100% 的穷举测试，这也注定了我们不可能把所有的 Bug 都找出来。这就意味着总有一些 Bug 还在我们身边。在这个阶段，一个人的 Bug 敏感度尤显重要，而对 Bug 的敏感度依赖于过硬的技术积累与极富挑战性的创造性思维。而这种技术的积累与创造性的思维相结合迸发出来的一种思想，往往能带来一种高效地改进现有测试方法或流程的新设计。

测试设计是一个过程，不仅仅是解决某一个问题的方法，它主要包括测试管理的设计以及各种测试技术应用的设计，其中测试管理中的团队管理方法设计与测试流程设计是重中之重，犹如游戏中的游戏规则。

在软测领域，测试方法众多。从了解软件系统内部程序结构的程度不同来看，可分为白盒测试、灰盒测试、黑盒测试；从程序的运行状态来看，有静态测试、动态测试；从测试人员角色来看，可分为手工测试、自动化测试；从测试阶段来划分，又可分为单元测试、集成测试、系统测试、验收测试（包括 Alpha 测试和 Beta 测试），等等。而这些方法之间是存在交集的，单元测试属于白盒测试，同时可以是动态测试，也可以是静态测试，如代码走查；集成测试可以归并为灰盒测试，是有静有动的结合体；黑盒测试也属于动态测试。对于这些常见专业名称的解释见附录 A（专业名称解释）。在进行测试设计，也就是测试之初进行测试的分析时，在测试方法的选择上，面对林林总总的方法，该选取哪些方法或哪些方法的组合是对测试设计人员的考验与挑战。现假如模块 S 是整个系统的核心，处于系统架构的中间层，它的逻辑处理复杂，向下与底层的操作系统交互，向上给业务模块提供接口。由于这些特殊性，以及其输入与输出在业务层的表现并不那么透明等特点，最后测试设计人员决定采取如图 3-2 所示的测试策略，各阶段分别采用不同的测试方法来验证模块以保证它的质量。

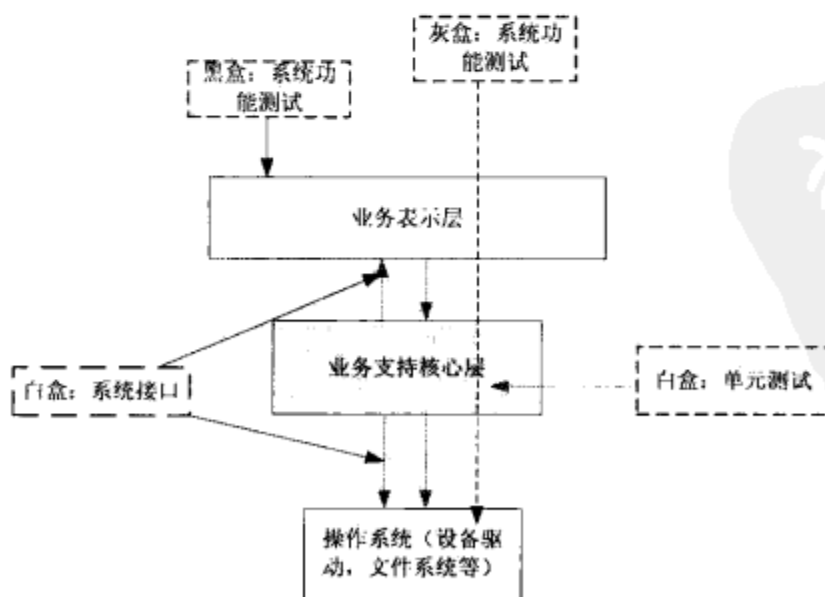


图 3-2 “业务支持核心层”的测试技术应用设计

通常，一个项目的完整测试过程，它是由多个环节组成的，每个节点的工作重点是什么，该做好哪些工作，哪些环节可以合并，哪些不可以跳过，根据项目的需求与约束，是需要设计（也可理解为策划）的。如图 3-3 所示是常见的测试阶段与输出流程图。

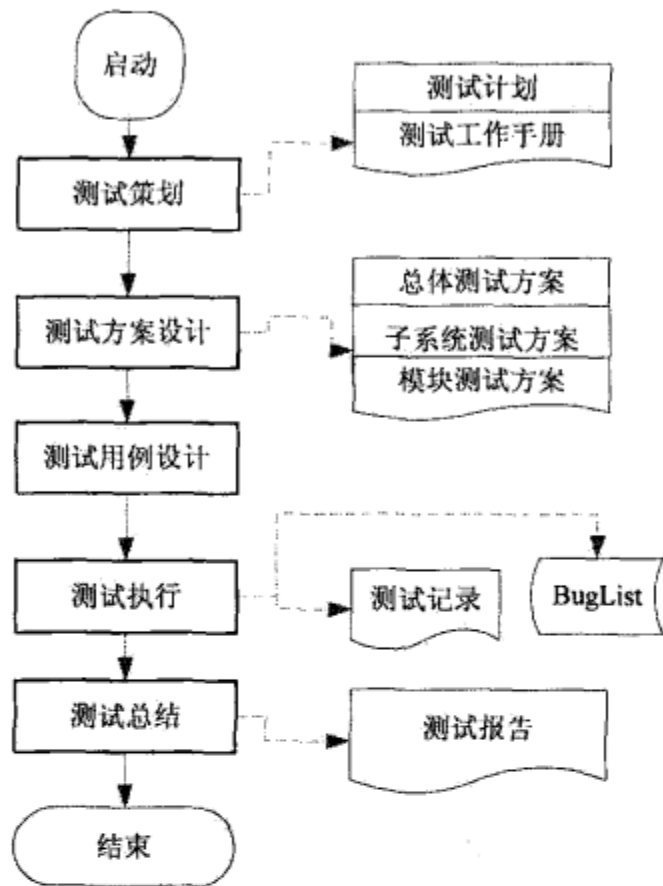


图 3-3 测试阶段与输出

注：图中的实线表示测试环节，虚线为各环节上的输出

测试流程管理的设计，还包括采用什么工具管理测试发现的缺陷，用什么工具来管理测试用例；当一个或多个测试团队一起工作时用例设计的规范如何定义；测试工具的代码、或自动化测试脚本、测试数据生成工具等输出工件如何进行配置管理等，如表 3-1 所示是一个需纳入流程管理的内容记录示例。

表 3-1 测试流程管理的内容

管 理 需 求	到 位 情 况	备 注
配置管理规范	已有，但需根据项目需求进行修订	
用例管理工具	已有（正在用的 TestLink）	
用例设计规范	已有	需重新制定
Bug 管理工具	已有，Bugzilla	
Bug 管理规范	已有，但需根据项目需求进行修订	
测试方案用例评审机制	需制定	

测试过程的设计犹如运筹学中提到的寻求最短路径问题，从起点 A 出发到达终点 Z，中间的路径很多，在每一个中间节点上可以选择分支，也可以在某个节点走直线，权衡利弊而做出最优的选择，这种选择的过程实质就是线路设计的过程。设计服务于过程，过程体现设计，推动着设计的改进，设计与过程是分不开的，测试设计就是一个不断改进的过程。如同马克思主义的实践发展观“实践是不断发展着的，科学的发展来自于不断的实践，实践推动着科学的不断发展”。

测试设计，给测试创新带来机遇，也带来挑战。通常测试设计是针对项目任务提出的解决策略、思路或方法，强调实用性。测试创新，可以是一种思路或一种方法，或一种流程规范，要结合项目任务来体现，以说明它的可行性与价值。测试创新也是一种设计，但这种设计是新的，是没有经过验证的，在得到证实之前，有种摸索的感觉。犹如摸着石头过河，有险情，但险情过后，是难得的喜悦，或者说是创新成功。这样，项目任务成了改革创新实验田。

小贴士：

测试设计是一个过程，它主要包括测试管理的设计，以及各种测试技术应用的设计，其中测试管理中的团队管理方法设计与测试流程设计是重中之重，犹如游戏中的游戏规则。

3.3 测试管理中的隐形指挥棒：测试组织模式的设计

很多公司都宣称，员工是最重要的资产。在测试管理中，关于测试团队的管理同样是一件重要方面。人是万物之灵，每个人都有自己独特的一面，如何才能让员工充份地发挥自己的潜能为公司服务，在个人的职业生涯发展道路上更顺畅，同时也给公司带来更多的效益，最终达到一个双赢的目的，这是职场人士及公司经营共同的目标。

测试团队的管理是测试管理的基础，不同的组织模式，决定着测试团队在公司及在项目中的位置。测试的组织模式就像一根隐形的指挥棒，而指挥官则是公司的管理层。下面就以测试组织的不同模式为切入点，与读者分享不同的测试组织模式设计将带来不同的境遇，实际操作中设计适合公司当前发展的组织模式即为上策。

3.3.1 以开发为核心的组织模式

不同公司由于规模或产品的复杂度等的不同，采取的组织管理模式也不同。不同的组织模式，决定着测试团队在项目中的不同地位，本节介绍“以开发为核心的组织模式”的设计。

1. 以开发为核心的基础模式

以开发为核心，测试是开发队伍中的一部分，如图 3-4 所示。测试负责人向开发经理汇报，多见于规模不大的一些小公司。在这种组织中，开发工程师除了负责设计的实现外，还需参与编写需求，或者需求直接由开发主管，甚至是开发经理编写，即某些开发人员承担着双重或多重的角色。例如，开发经理也是项目经理，带着大家一起做一个项目，在公司的发展初期常会有这种情况。

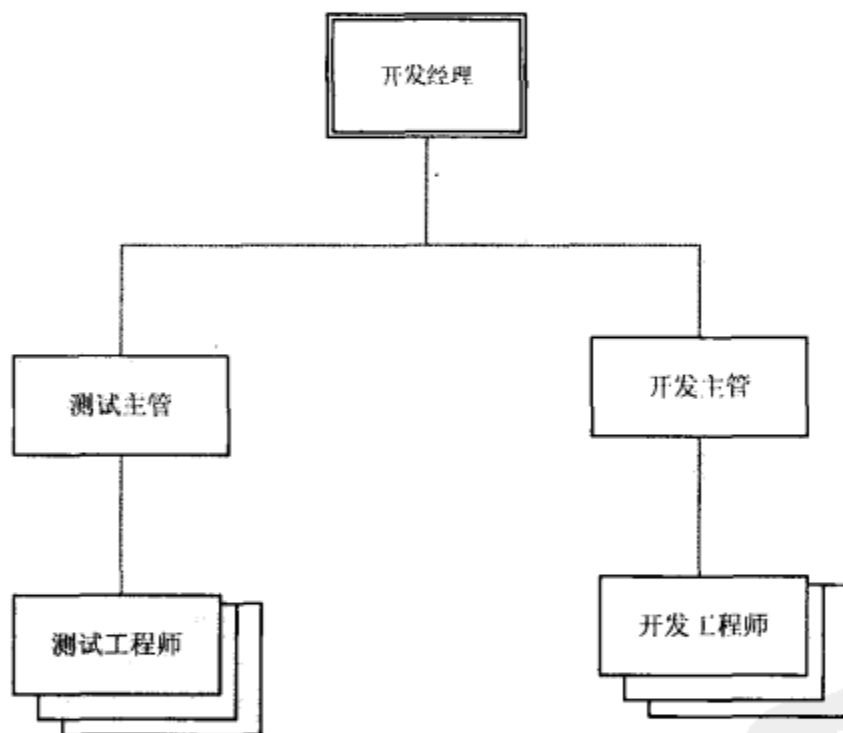


图 3-4 以开发为核心的组织模式

2. 以开发为核心的扩展模式

随着公司的发展壮大，项目的开发需求增加，上面的模式已不足以支持多项目并行开发的需求，需要把某些工作从某些身兼数职的人身上剥离出来，独立成立某专业组，使分工变细，更明确，更清晰。于是如图 3-4 所示的以开发为核心的组织扩展模式出现了。在此模式中，成立了项目管理组，软件需求组，软件需求组向开发经理汇报，项目管理组通常向更高一级，如总监级汇报。

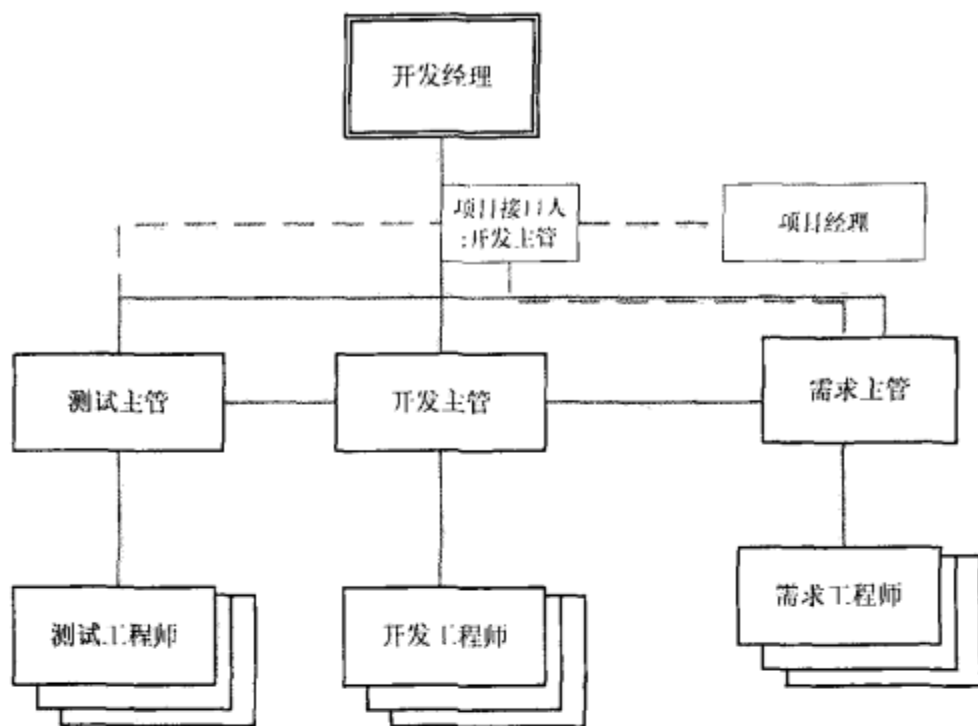


图 3-5 以开发为核心的组织扩展模式

注：图中虚线表示项目线，实线表示行政线。

这种模式很有趣的一点是同属一个软件部门的软件需求、开发、测试小组中，是由开发主管作为代表与项目接口，参加项目的各种会议。另一个突出的特点是此种模式分开了行政线与项目线。在行政线上，各小组在人力资源管理及专业技术上向开发经理汇报，但在项目线、工作进度与质量上，接口人向项目经理汇报，各技术小组向开发主管汇报。开发经理与项目经理并没有什么隶属关系，项目经理需要资源时需找开发经理，经理再安排主管进行传递与控制。

这种组织模式有哪些优点呢？在项目的接口上，只派出了开发人员为代表，可以减少接口沟通上的成本。但这其实是一把双刃剑，一方面从表面上看是可以减少各专业组直接与项目组其他专业组接口的开销，但开发代表是否可以代表需求与测试小组，参加项目会议回来信息传递是否能到位，是否了解需求与测试小组对项目的需求，都存在疑问。

再从反面分析，它存在哪些缺点？

- 测试主管在行政线上向开发经理汇报，测试主管会感到有一种特殊而微妙的感觉，在报告项目的测试结果时不带有色眼镜是不现实的。在项目线上，汇报进度与质量时，开发主管在某些方面会做一些过滤，不可能把最真实的项目质量情况反映给项目经理。当然，有色眼镜在原则上是有度数限制的，不能太离谱，否则产品到了用户手上再反馈回有严重问题，情况就不那么简单了。
- 在有紧急任务时，在开发经理的要求下，有一些测试资源常会去做协助开发的调

试工作（例如，开发改一个问题，测试帮忙验证是否改好，常是验证用于突发解决用户端某些问题的临时版本）。

- 测试人员需充当多个角色，如软件部门的配置管理员，软件帮助文档、手册的编写者等。
- 测试小组在整个部门，乃至在项目中的影响越来越小，在测试人员中，有的人渴望转为开发人员，有的人想转为需求设计人员，随着时间的流逝，测试组织非但壮大不起来，反而可能消失。

综上所述，此开发模式适合于公司发展的初期，属于一种过渡的组织模式。接下来，介绍以项目经理为核心的测试组织模式的设计。

3.3.2 以项目经理为核心的组织模式

首先介绍以项目经理为核心的基础模式，然后再介绍其扩展模式。如图 3-6 所示，它是一种典型的以项目经理为核心的团队管理模式，开发小组与测试小组并存，隶属于项目经理领导。这使笔者想起建筑工程队的包工头，或许不太合适，但二者有相似之处。这种模式，对于项目经理来说是有利的，项目经理不管你是什么专业方向，对他来说都是资源和工具，是完成公司交给他这项任务的必需工具，他只关注本项目的进度与质量，而各技术专业组的技术积累、人员培养等则超出他的管理范畴，但对公司来说，是需要考虑这些方面的，员工成长需要项目作为历练的平台。那么这种模式下的明显弊端需要如何弥补呢？这就需要如图 3-7 所示的以项目经理为核心的扩展模式。

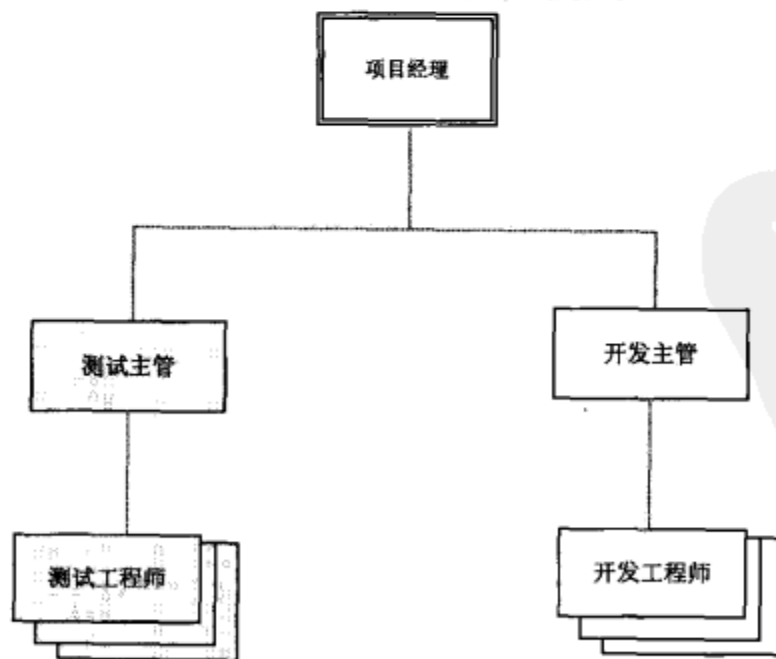


图 3-6 以项目经理为核心的基础模式

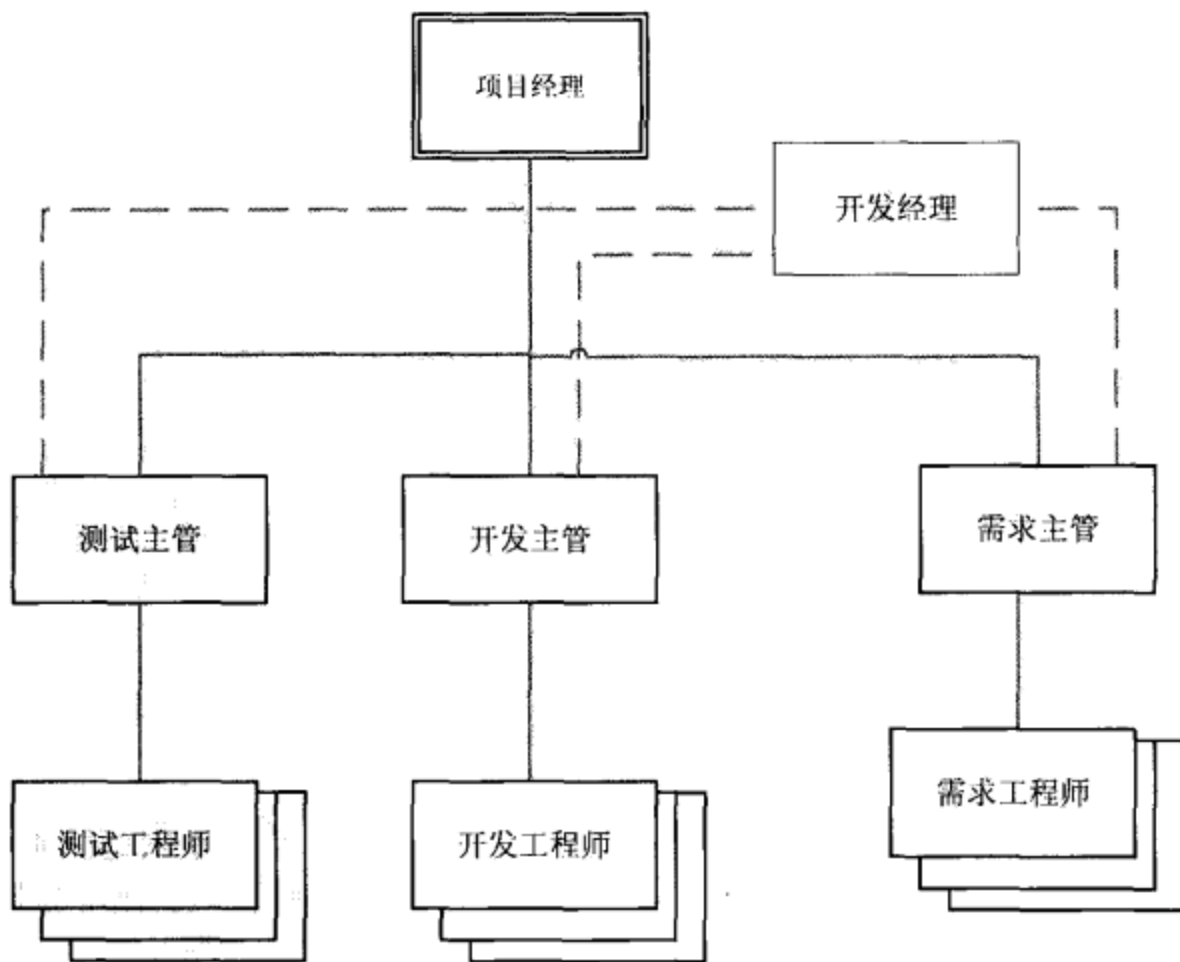


图 3-7 以项目经理为核心的扩展模式

注：图中虚线表示行政线，实线表示项目线。

在此扩展模式中，出现了开发经理，在行政线上他管理着软件需求、开发与测试小组这 3 个小组的人力资源使用情况。在项目的进度及质量上各技术小组单独向项目经理汇报。季度或年终绩效考核时，项目经理及开发经理分别在不同的管理方向对各技术小组进行评价。

此种以项目经理为核心的管理模式中，资源稳定，整个项目团队凝聚力高，大家都在为着同一个目标而努力，那就是把项目做好，按时按质地将研发出的产品交给公司。如果项目经理的号召力强，有魅力，能把项目的内外关系处理好，又能赏识团队，在到达不同阶段的里程碑后，及时表扬项目成员，如举行适当的庆功会，犒劳员工等，项目不成功都难。

对于测试来说，相对以开发为核心的组织模式，错误报告和其他测试信息直接向项目经理汇报，有机会说真话了，测试主管与开发主管是同级的，测试组织的声望更高了。但仍属同一个开发经理管辖，意味着仍需竞争同样的集中资金与预算。

在用这种管理模式完成项目测试任务的过程中，笔者也曾遇到过一些尴尬的事情，大

致有如下几点。

- 资源利用率不够高效。由于给项目的资源是固定的，即一旦某位工程师加入此项目，需要到本专业任务结束或项目结束后才能离开项目，对于某些特殊情况还需项目经理同意后才能释放资源。而实际上，在整个项目的开发过程中不同阶段的不同技术组的任务量是不同的，比如在项目后期，主要是解决 Bug 使版本稳定，需求及开发人员相对工作量较少，此时有些人员完全可以加入到其他项目中。但通常，项目经理是不会同意释放资源的，此时，即使是开发经理也很无奈。
- 技术平台积累不理想。技术平台积累的优先级永远低于项目任务的优先级。对于项目而言，目标很清楚，就是按时按质地做出产品，提交公司生产、销售去赚钱。而对于技术部门而言，需考虑人力、技术成本。如果做完一个项目，没有一定的积累，再做同一个类似的项目时，人力、技术投入的成本有增无减，这是技术部门管理的失败。
- 双重考核的尴尬。项目经理在拿到公司对项目的考核结果后，根据项目完成情况，把各种等级标准分到各技术组。各技术主管完成各技术组的一级考核后，发给项目经理进行二级考核，最后发给行政线经理，即开发经理进行最后的评定。由于开发经理没有实质性的项目结果包（或许可以理解为奖金包），评定更多的是留于形式，但也不排除出现残酷的一面，本来可以被项目经理定为“A”的工程师，被开发经理站在其他角度调整为“B”或“C”。

3.3.3 独立的测试组织模式

在独立的测试组织模式中，如图 3-8 所示。项目经理、开发经理和测试经理并行，测试组织具有真正意义上的独立，具有权威的地位。测试资源与预算再也不用与开发一起合计，被调去支持开发调试的救火工作不属于测试部门，而是开发部门内部的事。当项目增加时，可以根据需要自由增加人力、技术的预算。向上汇报时，除了可以如实汇报外，你还可以根据产品的质量状态提出有建设性的意见或见议，管理部门会用完全开放的心态听取来自测试状态的报告，测试在研发系统中的影响力较大，是测试组织管理的最佳模式。

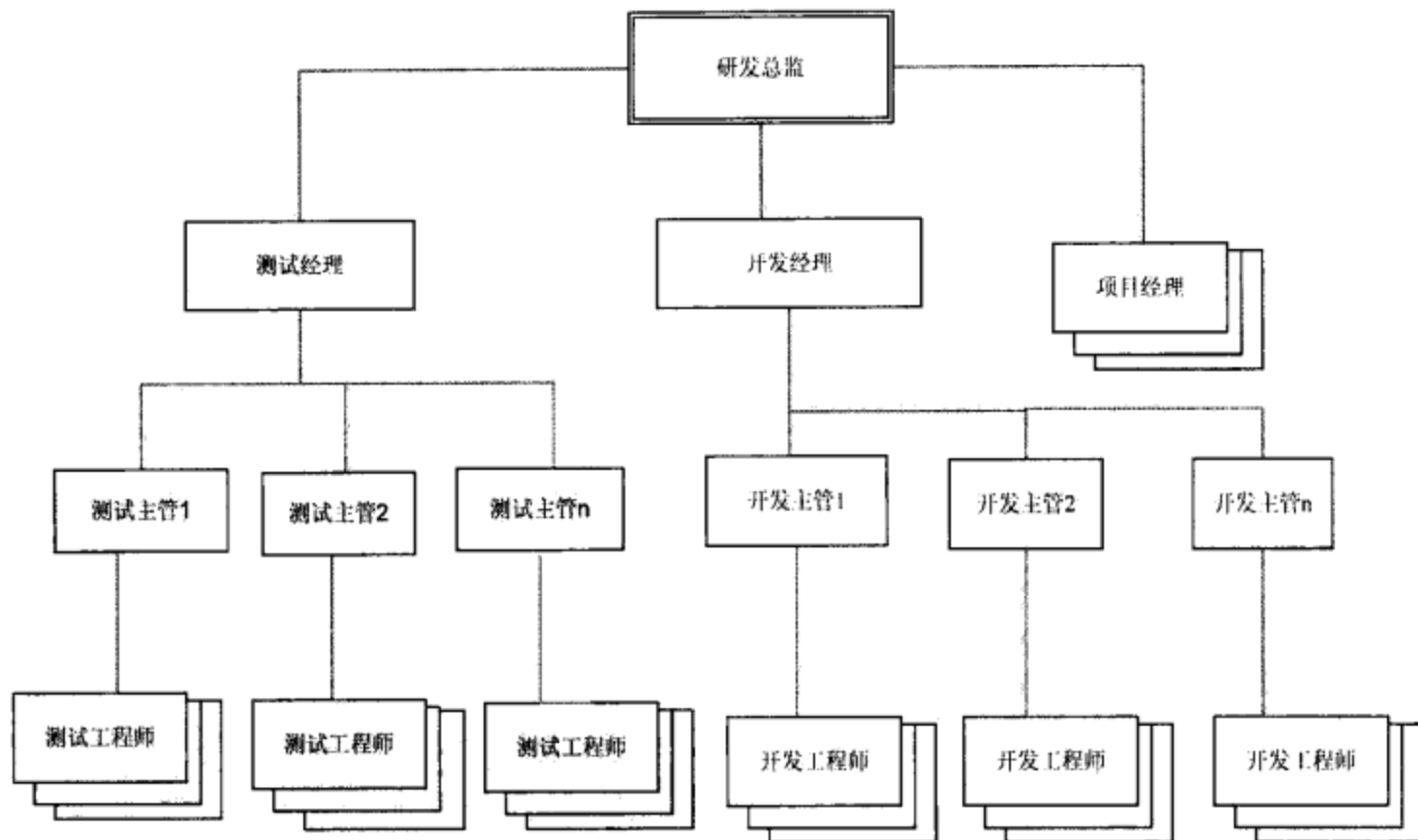


图 3-8 独立的测试组织模式

此种模式，对测试独立性、权威性来说是最好的，意味着在行政线上测试团队在独立的测试部门范围内工作，对应的开发团队及项目管理团队也在自己的部门中工作，为了一个项目，大家并肩协作。此种情况的模式有一个显著特点，那就是测试人员是对测试部门负责，一个人可以同时服务于多个项目。也就是说，对于项目来说，资源是不固定的，这对于技术平台化积累是很有效的。

在测试组织的管理模式上，还可以根据公司的不同情况进行调整。但是，无论采用哪一种模式，大家都知道“天上是不会掉馅饼的，地球上没有天堂”，公司是要通过销售产品来赚钱的。当项目进度紧，发布版本必须要在某一天拿出来时，质量屈服于进度的牺牲是一个永远存在的遗憾。在项目中常表现为开发在拼命地改 Bug，而测试没有完成充分验证，版本已顶不住市场的压力便匆匆上线了。

3.4 提高测试效率的有力武器：测试流程之设计

测试流程并非一些数字或文字的简单排列。合理可控的测试流程，犹如一条两边带有防护网的钢索桥，大家走在上面，既安全又可靠。即使是新加入测试团队的成员，也可以很安稳地度过一关又一关，把每一环节的测试工作顺利地贯穿起来，形成一个可见、可控、

有序的测试系统链。

下面通过两个案例“让大家一起忙起来”和“软件运行得犹如蜗牛在爬行”介绍如何把工作中出现的问题，通过改进测试流程的方式来解决。这种流程的改进正是一种适合公司业务发展的流程设计。

3.4.1 认识测试流程

当在工作中出现问题后，常会听到一些测试人员说“是不是测试流程有问题？”那么这个测试流程到底是指什么呢？

从狭义上讲，测试流程是指完成某项测试任务时，对如何完成任务做的先后安排，如先有测试计划，然后是设计测试方案及测试用例，接着是执行测试活动，最后输出测试报告，一件测试任务到此结束。然而，在实际工作中，测试的依据是需求，测试的对象主要是软件，不可能与其他团队成员没有联系，测试流程中需要考虑这些因素对测试各环节输入/输出的影响。

从广义上讲，测试流程是所有关于测试事务的抽象统一体，流程有静态性，同时也有动态发展的特点。从静态来看，主要指已有的测试流程规范，包括测试工作指南、测试计划、测试方案、测试用例设计模板等，这些属于测试流程体系中的文档输出指导范畴。还有属于测试管理流程范畴的内容，包括测试新员工入职指引、项目测试手册、测试配置管理手册、测试知识经验库、培训库等。还有如功能测试方法库、系统测试方法集、经典Bug分析集，灰盒测试、白盒测试指南等一系列技术性的流程、规范或指南。这些流程体系、知识库的积累，并不是一朝而就，是一个测试团队成长的重要标志。也正因为如此，流程体系有动态发展的一面，随着公司内外环境的变化，固有的流程体系有些可能过时，有些仍可继续使用，需要根据不断变化的需求做合适的调整。

测试并非空中来客，也非外星人那样另类、独立。需求、开发、测试在软件产品的整个开发链中，三者之间的关系特别紧密，犹如一个三足鼎立的主体，如图 3-9 所示。它们之间的工作方式可以是串行式，也可以是迭代式，当然也可以是并行式工作，这取决于研发团队采用什么样的开发模式。项目的研发是公司的生命线及核心竞争力，对公司的发展承担着重要角色。同样地，一个项目的开发模式，对测试流程体系的建设与发展变化有着重要意义。通常情况下，意味着对原流程体系的局部保留与改进，但也可能需另立一套全新的流程予以支持。总之，对待测试流程不能仅是看静的一面，静只是暂时的，它犹如行程中的流水，需适时适地顺势而变化，方能流得更长更远。

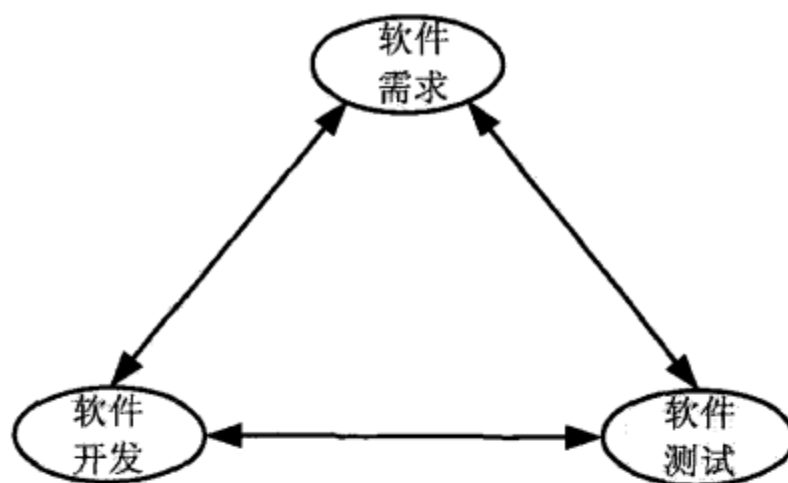


图 3-9 软件需求、开发、测试三足鼎立示意图

测试流程的实质是一种资源平台，且是核心资源，但与其他资源，如人力、物力资源相比，它具有一定的独特性，多数情况下流程看不见、摸不着，甚至说不清、道不明。然而测试的流程又确实存在于测试组织运作的方方面面，直接影响着所有测试人员的绩效，这就是流程隐含着的魅力。测试流程虽是一种资源，但不具有独立性，若是脱离了人这个主体，流程是无法创造价值的，几乎不名一文。

通过前面的流程内容介绍可看到，所有的流程规范都是设计出来的，是设计的结果。实际上正是透过各测试环节任务的安排，将测试的资源（流程规范）以适当的活动转化成了为测试所提供的服务。

3.4.2 让大家一起忙起来

本节的案例“让大家一起书忙起来”是测试流程改进设计的一个典型案例，通过案例的分享，希望能帮助读者理解测试流程对提高整体测试效率的重要性。

【案例】让大家一起忙起来

问题背景：在产品的研发阶段，出现开发在提交内部版本给测试后，由于暂时没有新需求要实现，也没有 Bug 需要解决，开发人员处于闲置状态。

问题分析：测试在接到新版本后，首先是回归已解决的 Bug。特别是，当前有比较多的 Bug 需要回归时，新提交的功能得不到及时的测试，而往往新提交的功能是存在新 Bug 的。但在这种工作流程模式下，新 Bug 不能在提交版本后的前期发现，造成了前期测试紧张而开发暂闲的状态。而在版本测试的后期，由于开发人员需解决不少的 Bug，且要进行版本联调，在后期的任务显得很紧张。而此时测试反过来是处于相对闲的状态，出现等候开发提交新版本来测试的现状。如图 3-10 所示为原开发—测试合作流程图。

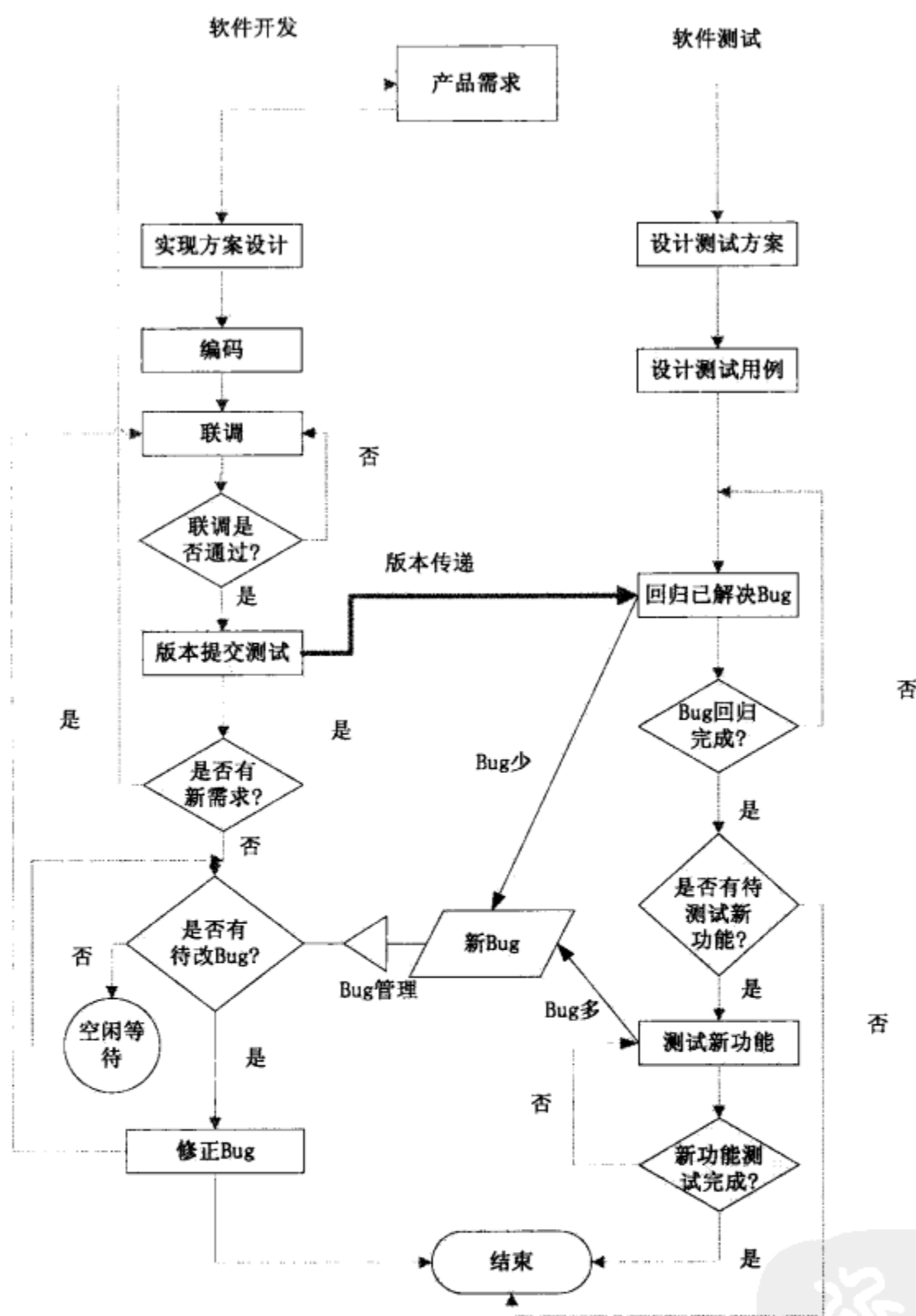


图 3-10 原开发-测试合作流程图

解决措施：针对此开发与测试的工作模式在流程上的问题，提出测试执行人员先不回归 Bug，拿到新版本后，先测试新的功能，提交 Bug 让开发人员改，改变开发人员普遍性的闲置状态。当开发在改 Bug 时，测试再回归已解决的 Bug。这样一来，开发与测试同步忙起来，流程上顺畅多了，项目进度得到了保证。这是一个典型的流程控制方面的测试设计。如图 3-11 所示为改进后的开发-测试合作流程图。

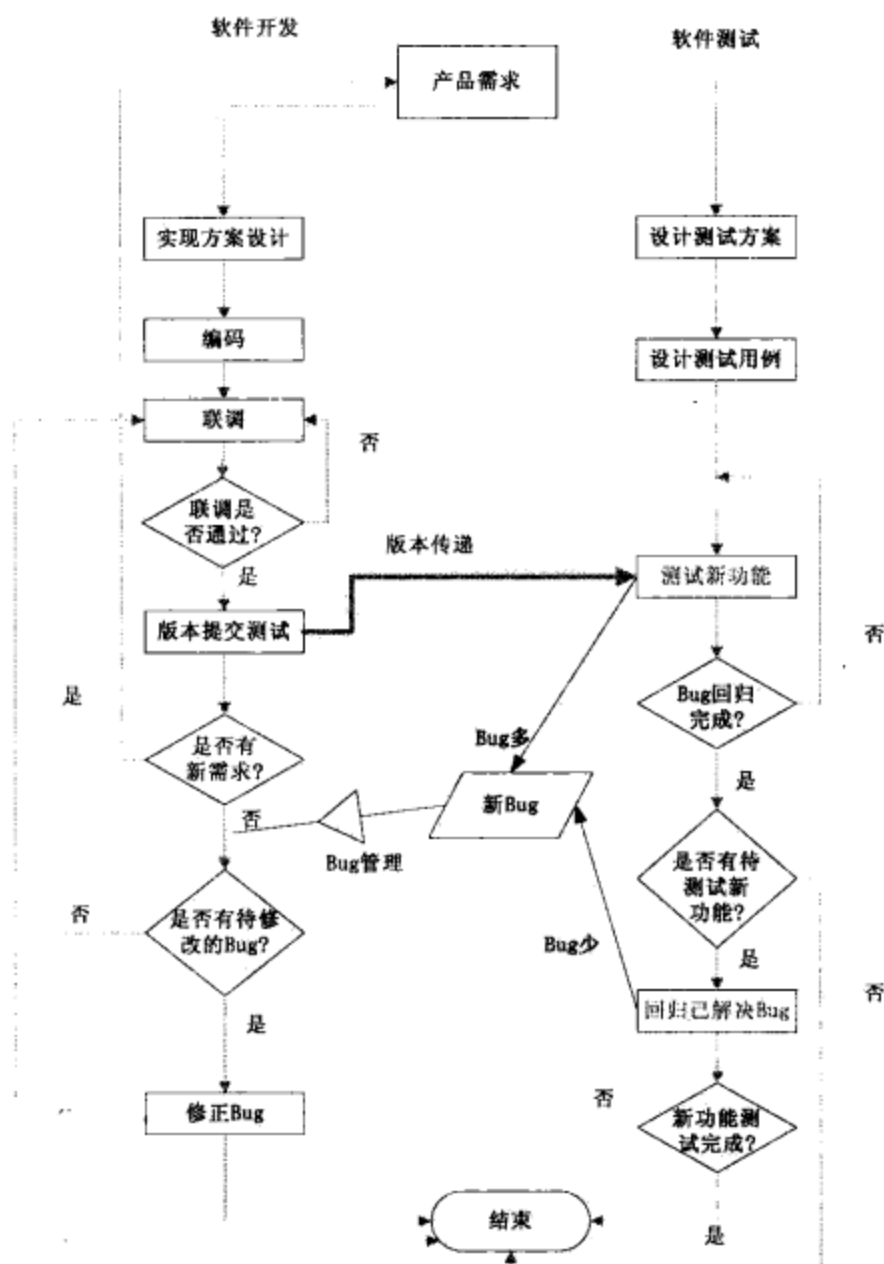


图 3-11 改进后的开发-测试合作流程图

3.4.3 软件运行得犹如蜗牛在爬行

测试设计不只是测试设计工程师的事，需要团队中每一个成员共同努力，哪怕是主动地反映遇到的问题，下面介绍一个真实的典型案例。

【案例】软件运行得犹如蜗牛在爬行

一个平常的工作日，将近下班时间了，刚参加完关于内存泄漏测试方案评审会的测试设计工程师赵函，习惯性地每天这个时候进入公司的缺陷跟踪系统，跟踪执行人员提交的Bug。其中一个Bug立刻吸引住了赵函的眼球，Bug描述是这样的：“仪器测量完成后，测量结果显示在界面的过程如蜗牛在爬行，响应缓慢（刷新完毕约1分钟）”。赵函马上找到

提交 Bug 的工程师了解详细情况,并察看这个蜗牛爬行的显示过程到底是怎样一种现象(蜗牛爬行的平均速度是每秒钟 0.5 厘米,如图 3-12 所示)。不看不知道,此问题不仅仅发生在测量结果显示的处理上,切换到其他任何一个界面同样异常缓慢,这不是明显影响测试的效率吗?于是马上找相关开发工程师一起分析、定位问题,并要求重发版本给测试。后来开发查明是因为某个开发工程师提交版本时忘了关后台的调试信息打印开关。

此 Bug 是下午 2 点钟提交的,也就是说测试执行人员在这样的版本上忍受了近 4 个小时的测试,竟然没人提出要终止这样的版本测试。问及执行的测试工程师时,回答各执一词:“我不用老是切换界面,没有多大影响!”“我已提了 Bug,软件慢我也没办法呀!”“开发已知道了,下版会改的。”

“我们的测试同胞都是逆来顺受的?还是哪个流程环节上的出了问题?”赵函这样想着,找来了开发与测试经理,商讨避免这种问题的对策。据调查,负责版本发布的开发人员在版本发布前是有按流程进行冒烟测试的,但遗憾的是,这是在测试环境上确认的功能,不是在真实环境下的用户场景中确认的。而测试工程师也没有流程来支持,遇到这种情况如何处理?

最后商讨的决策是改进开发与测试版本传递的流程定义,明确具体的细节。

要求负责版本发布的开发人员在版本正式提交给测试前,必须在真实的用户场景环境下运行软件,确认实现的新功能或更改的正确性,并做确认记录,此记录随同测试传递表,以及软件版本一起提交。

增加测试接收版本的约束: 增加版本接收确认环节, 时间不超过 2 小时, 如出现严重或致命 Bug, 影响测试工作的开展, 通知测试经理, 停止当前测试。



图 3-12 蜗牛在爬行

3.5 好钢用在刀刃上：测试技术应用之合适设计

不同行业不同业务,测试方向会不同,如测试手机产品等的嵌入式软件与测试 Web 互联网软件就有很大不同。面对林林总总、错综复杂的测试对象,选择什么样的测试技术,如何应用,正是测试技术的应用设计需考虑的。

下面通过“通信的心跳在狂蹦”及“解开用例失效之迷”两个案例的分享，希望能帮助读者理解测试技术应用设计的重要性。当出现漏测后，如何分析问题、解决问题，并做如何改进以防患未然。

3.5.1 通信的心跳在狂蹦

通信心跳是当今通信技术应用中常采用的一种方法，指服务端每隔一段时间发一指令包到客户端，客户端收到后，再反馈应答指令包，以表示自己还处于连接状态。否则，如果间隔一定时间未收到这样的指令包，则服务器认为客户端已经断开，而进行相应的客户端断开逻辑处理。

【案例】通信的心跳在狂蹦

背景描述：一天，某市三甲医院检验科的医生特别忙碌，等在门外边排队取微生物检测报告的用户明显比往日多。据当班的陈医生说是因为PC端的数据接收软件出了问题。原来与检测仪连接后，系统可一边在仪器端测量样本，一边在PC端进行数据处理，同时PC可自动打印报告单，但现在接收软件很容易报错，严重影响了检查速度。据陈医生说，前几天按仪器厂家维护人员要求，对仪器软件进行了升级，但这两天用过此仪器，也可以在PC通信软件上打印数据。今天人多，软件却出了问题。

说明：血球仪，即血液细胞分析仪，是用于医院检验科测量病人血细胞的精密仪器。背景中描述的检测仪与PC端数据接收软件通信的物理示意图如图3-13所示。

事故影响分析：从前面的背景描述中，可清楚看到对终端用户医院所造成的不良影响，更严重的是一些急诊病人不能及时得到检验报告单而延误了病情的诊断，更是间接造成了人的生命安危问题，影响非同小可，是一个急需解决的问题。

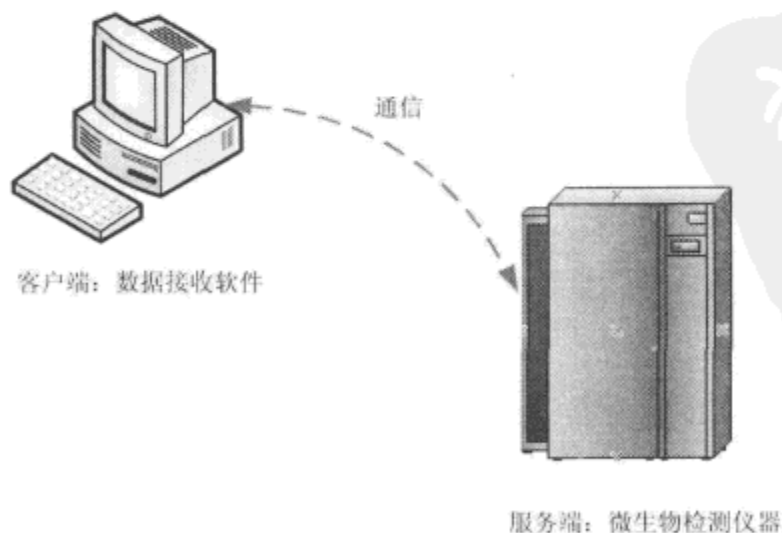


图 3-13 微生物检测与 PC 数据接收软件通信物理示意图

问题分析：该医院用的微生物检测仪与数据通信接收软件并不是同一家公司的，但它们都遵守行业的通信协议，所以双方的数据可以通信。医院先找到数据接收软件的公司，该公司分析说是由于仪器端软件快速发送心跳包，打破了原来的通信规则，把接收端的缓冲很快挤满了，而造成了内存溢出错误。于是问题很快被转到仪器软件的研发部门。

问题进一步分析：仪器软件的开发人员接到反馈后，马上组织相关人员进行分析，发现是由于开发人员在解决通信速度慢问题时，把通信心跳的 3 秒延迟去掉了，导致仪器（服务器端）一旦与 PC 数据接收软件（客户端）建立连接后，一直不停地发心跳包给客户端，使得通信协议工作紊乱。在客户的内存处理机制欠佳的情况下，很快造成溢出，软件工作失常。问题的原因找到了，开发人员快速地通过修改代码，对编译版本进行处理，测试人员复测后，第二天对院方的仪器软件做了升级。

漏测分析：分析当时的测试记录，通信的功能测试用例存在，用户常规使用场景测试用例也存在，采用的都是黑盒功能测试方法。而问题恰恰是出在与时间相关的通信连接上，且这种连接是在服务器端没有数据包（测量数据）传输的情况下才发生。测试为什么会遗漏这个测试点？要如何做才能发现这个问题，又要如何做才能在日后规避这个问题呢？

改进测试方法：对通信的测试，不能仅停留在业务功能。有效可行的方法之一是通过网络抓包工具截获通信端口直接出来的数据进行分析，可以透明地发现通信双方收发数据的正确性。一旦存在丢包严重，协议工作将紊乱，在通信端口截获的数据流中可以很快反映出来，并及时发现存在的问题。

一个人的成长过程中会遭遇很多挫折，只有从不断地跌倒中爬起来，再跌倒再爬起来，才能抵达一站又一站的成功彼岸。测试也一样，需从一次次的漏测分析中吸取教训，找到测试的盲点并进行突破。什么类型的测试，需应用哪些测试技术，并没有一个通用的标准。失败是一种财富，能否合理应用各种测试技术关键要在测试对象的深度分析上下工夫。

3.5.2 解开用例失效之谜

【案例】解开用例失效之谜

背景描述：某精密仪器有对自身部件进行自检功能，此自检的功能是通过软件发命令给硬件板卡上的控制处理器实现的。部件 A 有 3 种状态，“初始位”、“到上位”、“到下位”，用户单击“初始位”，不管当前部件正处于哪一个位置，都回到“初始位”；只有当前部件 A 在“初始位”时，单击“到上位”、“到下位”才有效。在“初始位”状态，表示部件准备就绪，当仪器工作时始终在这 3 种位置来回运动，完成一件工作后，重新回到初始位。

如图 3-14 所示是仪器自检工作的接口示意图。

从用户使用角度看，仪器部件 A 进行自检的人机交互界面如图 3-15 所示。

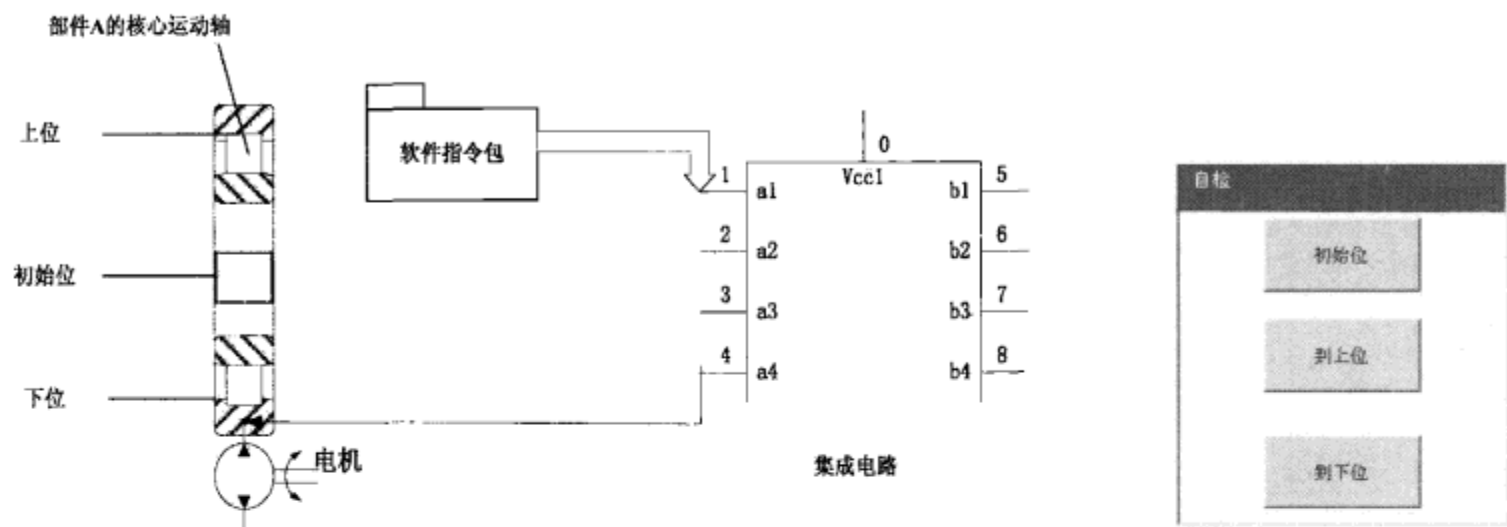


图 3-14 仪器自检工作接口示意图

图 3-15 部件 A 自检人机交互界面

在对测试对象的分析中，自检可以理解为软件的一个功能项，而此功能项又包括 3 个小点，包括“初始位”、“到上位”、“到下位”。设计测试用例时，需要针对这 3 个测试点分别进行设计。其中，“到上位”对应的测试用例及执行记录如表 3-2 所示。

表 3-2 仪器部件 A “到上位” 自检功能测试用例

用例 ID	测试标题	预设条件	操作步骤描述	预期输出	测试结果	测试人	测试时间
SelfTest-001	“到上位” 自检处理	部件 A 在初始位	进入部件 A 自检界面, 单击“到上位”按钮	部件 A 到达上位	PASS	工程师 A	2008-5-8
SelfTest-002	到上位后, 再命令部件 A 到上位	部件 A 已在上位	部件当前正处于上位时, 单击“到上位”按钮	部件 A 没反应	PASS	工程师 A	2008-5-8
SelfTest-003	部件 A 不能到上位时的异常处理	部件 A 在初始位	部件 A 与集成控制板之间的连接线没插上, 单击“到上位”按钮	部件 A 没反应	PASS	工程师 A	2008-5-8

就“到上位”自检本身的功能正确性来说，有上面 SelfTest-001 正向用例 1 条，SelfTest-002、SelfTest-003 逆向用例 2 条，完全达到了对需求点的覆盖要求。

但不好的事情发生了，此产品上市不久，终端用户反馈“自检功能使用失效”。测试部收到这个消息后，每个人都非常诧异。测试经理马上派人进行追查，测试工程师 A 找出当时的测试记录来看，如表 3-2 所示，很清楚写着是“PASS”的。用户就是公司的上帝，投诉电话已接到，投诉单很快会收到。于是测试经理亲自再在仪器上进行确认，结果是自检功能好好的。依往常经验，用户的投诉常有描述不准确，或误操作的可能，这已是常有

的事了。测试工程师 A 于是打电话和用户进行沟通，了解问题发生的详细情况。最后了解到“并不是自检功能失效，而是做了自检功能后，接着做一个用到此部件的测量，测量不成功，仪器报故障”。

当测试工程师 A 了解到此信息后，已明白了事情的一半原因，再进行测试，果真是这样的结果。这条路径测试确实没走到，是用户场景测试的重大遗漏，须马上出 ECR (Engineering Change Request, 工程变更申请) 补救。

事故影响：一旦出现这种使用场景，用户不能正常使用到此部件的测量功能，部分功能丧失。

问题分析：对于测试来说，表面上看是测试用例的缺失，没有像用户那样操作，对用户使用场景缺乏全面足够的认识。而往深层次思考，是缺乏对设计实现原理的理解，而恰好设计对这一点也没考虑周全。正如前面所说，仪器部件 A 的工作状态任何时候开始前都要在“初始位”。用户投诉的此事件，是因为部件 A 在自检中途退出时，没有复位，接着使用它，仪器状态不正确，所以上报了故障。理解了原理后，设计相关的测试用例也是自然而然的事了。开发最后解决此问题实际上也是增加了复位命令，即不管部件 A 当前在哪个位置，调试完成后，如果当前位不是“初始位”，都要回到“初始位”。

解决措施：紧急解决措施（临时解决）是用户关机，再开机，即重新启动仪器即可。但作为厂家，需考虑用户使用的满意度，彻底解决此问题才是上策。须及时出 ECR，以支持已售仪器的全线升级。

案例中的漏测是用例的缺失造成的，但什么原因导致做测试设计时没有考虑到这种情况，是值得重点思考的。也只有从教训中找到防范的控制措施，才能避免类似的问题重复出现。

小贴士：

ISO 9000 质量体系相关知识		
ECR: Engineering Change Request		工程变更申请
ECN: Engineering Change Notice		工程变更通知

第 4 章

测试架构的设计

本章从思索测试架构开始，探讨了测试架构是什么，包含哪些方面的内容，接着就测试人员最为关心的话题，测试职业发展线路的设计进行了剖析，引导不同的相关测试领域的读者找到自己的位置与努力的方向。本章在接下来提出了一种架构设计模型，然后围绕此模型，结合日常生活中大家熟悉的相框为例对处于模型中最底层的“测试框架”这个抽象概念进行了讲述。最后对框架应包括的内容及其设计方法进行介绍。

4.1 思索测试架构

测试架构，这个词不太好理解，本节从架构的解释开始，依托人们实际生活中熟悉的建筑架构设计，旁征博引，引出测试架构的设计是怎么一回事？它的发展与独特之处又是什么？测试架构都要解决哪些问题？测试架构之路在哪里？

4.1.1 认知测试架构

起初，听到测试架构设计时，如同听到软件架构设计一样，觉得很神圣，也很神秘。神圣，是因为这两个职位分别是软件测试，以及软件开发在技术线路上发展的顶尖职位，一般都是专家级职位。神秘，是因为不清楚它们具体是做什么的，不说专业外的人，即便是软件界的人士，也不一定都能说得清楚。下面就让我们一起来解读。

首先，弄明白什么是架构。架构（Architecture）在汉语词典中有以下 3 种解释：

①建造；构筑。

②框架；支架。

③比喻事物的组织、结构、格局。

在架构的解释中，首先提到的是建筑方面的架构，这也是最初的来源。后来人们对其他行业有着类似特性的工程设计也称为架构设计，如软件架构设计，测试架构设计等。架构设计凝集了设计师们的智慧，是一种高屋建瓴的高层设计。成功的设计，它是有灵魂的，是艺术与科学技术的完美结合体。根据设计的结果，架构设计又有层次之分，包括合格的架构设计、优秀的架构设计、卓越的架构设计，再有就是世上罕见的卓绝的架构设计。如图 4-1 所示，架构设计本身亦有它的梯度进阶发展规律。

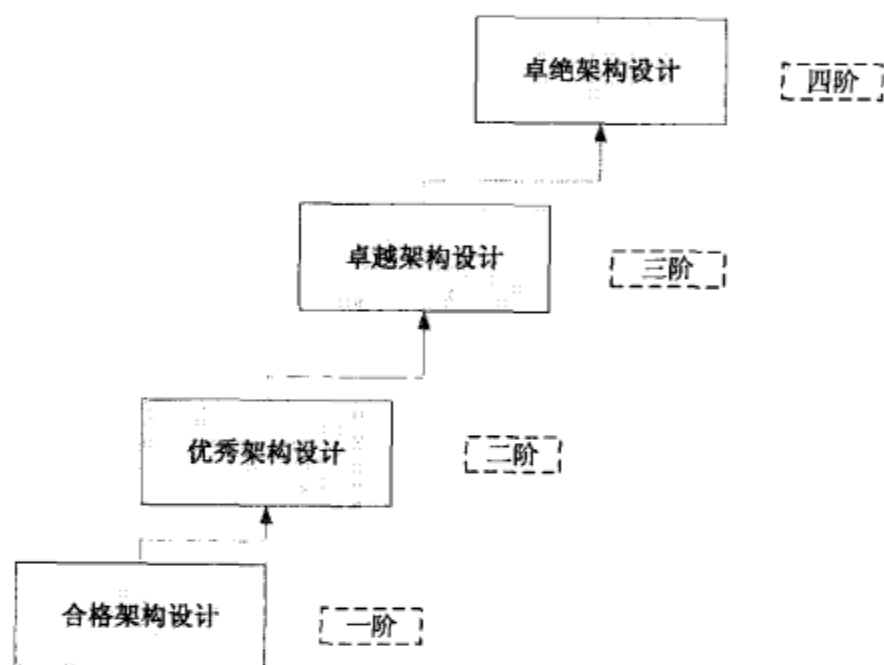


图 4-1 架构设计进阶图

建筑行业中的合格与优秀的架构设计，在我们的生活周围并不罕见，然而堪称卓越，特别是卓绝的产品并不多见，比如深圳的地王与北京的鸟巢。地王，它是深圳的标志性建筑，高 383.95 米，集建筑与人文景观于一身，堪称是一个卓越的创新型建筑设计；而 2008 年北京奥运会的巨型体育场——“鸟巢”，则是一个卓绝的架构设计，它不仅以美妙绝伦的外观给世人留下了深刻的印象，更重要的是它所蕴涵的意义和折射出来的思想，深邃而又遥远。“巢”象征着孕育生命，“鸟巢”就像一个摇篮，寄托着人类对未来的希望。

建筑行业的架构设计可以做到如此美妙绝伦，相比之于软件的架构设计、测试架构设计情况如何呢？建筑行业已有几千年的历史，而软件工程体系则是在近几十年随着信息产业的高速发展而逐步形成的。对于测试架构在业界的提及，更是近几年随着软件测试这个新兴行业的出现而出现的事。任何一个事物的发展，都有它的变化发展规律。在国外，软件测试行业的发展已有二三十年的历史了，测试已得到行业的普遍认识，对测试架构工作的认知度也较高。而在国内，它确实太新了，据智联招聘（www.zhaopin.com）2007 年关

于软测行业认知度的调查，广大职场人员对软测行业的认知度并不算高。选择非常了解的只有 8.8%，不到一成，选择“知道一些”的不足四成，其余超过半数的职场人都选择不了解甚至没听说过，如图 4-2 所示。这些数据也说明国内的软测行业还处于发展初期，但是国内的软件产业已处于高速发展期，一些大公司及对质量要求较高的公司已把软件测试的重要性提到了企业发展的战略日程上，对招聘测试架构师这个高端职位的需求也越来越多。

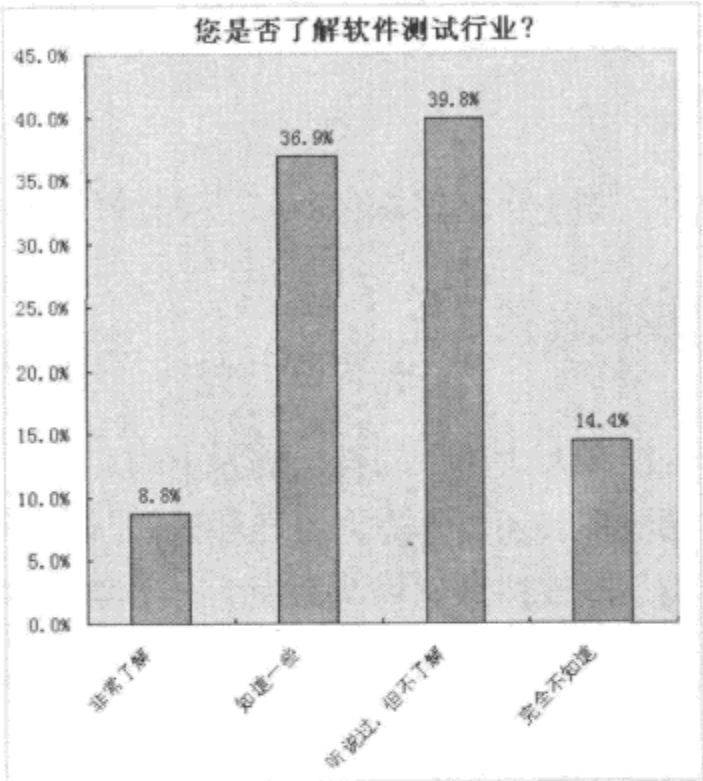


图 4-2 软测行业认知度调查

测试架构，尽管一些规模尚小的公司没有专门的这个岗位，但并不说明没有这个角色。实际上，测试架构自从有了测试岗位以来，这个角色就已存在。只是这个角色可能落在某个测试经理、测试主管，或者某个测试工程师，甚至还可能落在某个开发负责人的头上。所谓“麻雀虽小，五脏俱全”，只是对这个“五脏”的体现与发展程度会千差万别。测试架构的设计同其他方向的架构设计一样，需要纵横全局思考，不仅考虑测试技术的应用、研究，还需考虑测试人员的管理、测试流程的设计等。

测试相对于设计行业而言，如软件设计和建筑设计，是一种技术性服务行业，当然它也可以有自己的产品，即服务产品，如一批优秀的专业测试团队，一套合适的测试流程、方法等。如图 4-3 所示，以软件为圆心，所有测试服务产品，包括测试人力资源，都是围绕圆心而转动，推动软件的不断优化，最终输出优质软件产品。测试架构要考虑的事不止包括所有为软件服务的产品，同时还要考虑这些产品与接口部门的关系，甚至整个项目链各环节与软测之间的关系，是一个测试总体设计的工作。

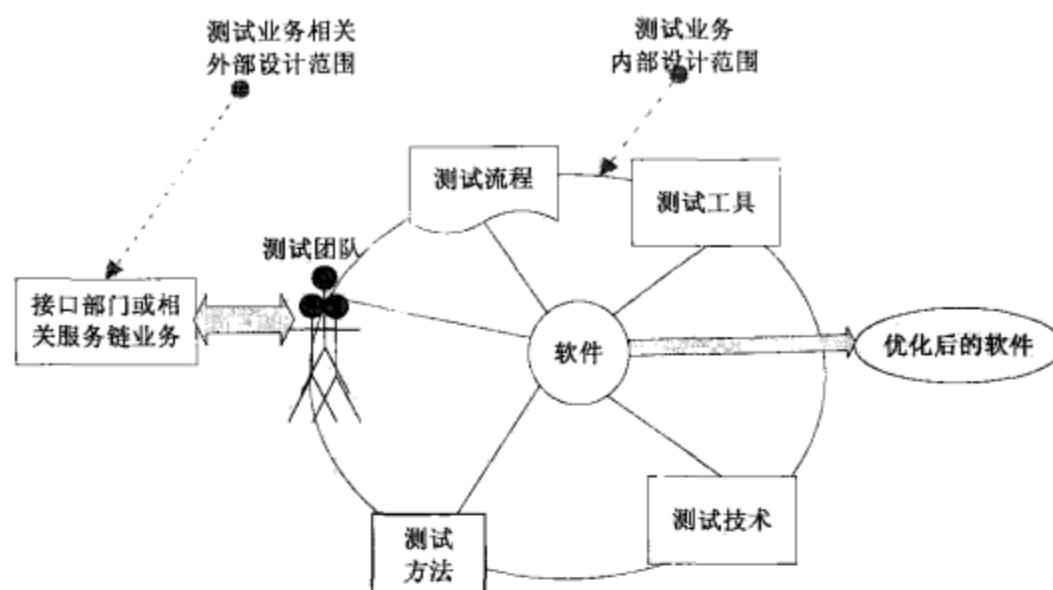


图 4-3 测试架构设计范围示意图

4.1.2 测试架构设计不仅仅在技术上

提到测试架构的设计，很多人认为指的就是技术上的高端设计，而在工程实践中，笔者觉得架构的设计不能仅考虑事，还要考虑人，即测试团队的管理。如图 4-4 所示为测试架构可以分为测试管理架构与测试技术架构两方面。对于测试团队的管理，既有行政上的管理，也有技术上的管理，两者之间相辅相成。比如在对公司测试人员的发展规划中，人员梯队如何构建属团队管理架构的范畴，而具体每一个梯队在技术上有哪些要求应由技术架构师定义更合适等。测试架构师应分为测试管理架构师和测试技术架构师。一般而言，目前管理上的架构师被别的头衔所覆盖，如测试经理或测试总监。

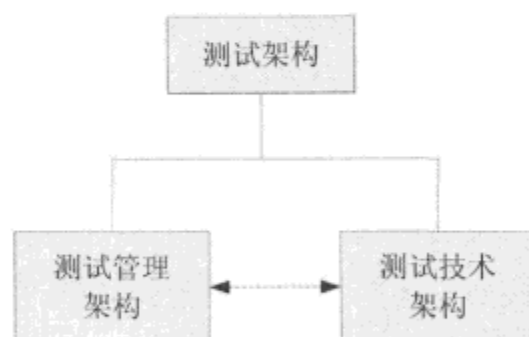


图 4-4 测试架构划分

4.2 让每个测试人员都看到希望

跳槽，对于职场人士来说是再熟悉不过的事了，当今，已成为可以改变自己的一种生活方式。然而，跳槽无论对个人还是对公司都是一个不容忽视的问题。对于个人而言，如

果跳得好那是职场人生新的起点，能走得更远更高，但如果没看清楚，也可能比原来更糟。对于公司而言，每年离职率在一个小范围内是很正常的，通过优胜劣汰的竞岗方式，每年主动淘汰一部分员工并补充公司的新鲜血液，也是必须的。但另外一方面，就是部分员工的主动离职，这一部分人中，有些可能是核心骨干，这就会给公司的业务造成一定的影响。那么如何留住更多的骨干员工、优秀员工，是管理团队必须考虑的事情，对测试团队的管理也不例外。

一次，笔者与一位在知名民企工作的资深优秀测试经理谈及测试团队的管理问题，这位经理提到“让每个测试人员都看到希望”的管理理念，让笔者深深震动。这是一种大度的、对员工负责、对社会有责任感的理念，可谓是管理架构设计的精髓、灵魂。它的功效是巨大的，是一股无形的力量。

测试团队管理的架构设计，在实际工作中要如何做才能真正让每位测试人员都能擦亮眼睛，看到这一束束的希望之光呢？更确切地说，这里的“每个测试人员”意指处于不同层次的测试人员，如图4-5所示。下面就测试职场人士最感兴趣的话题之一，即测试职业发展的路线设计与读者一起分享。

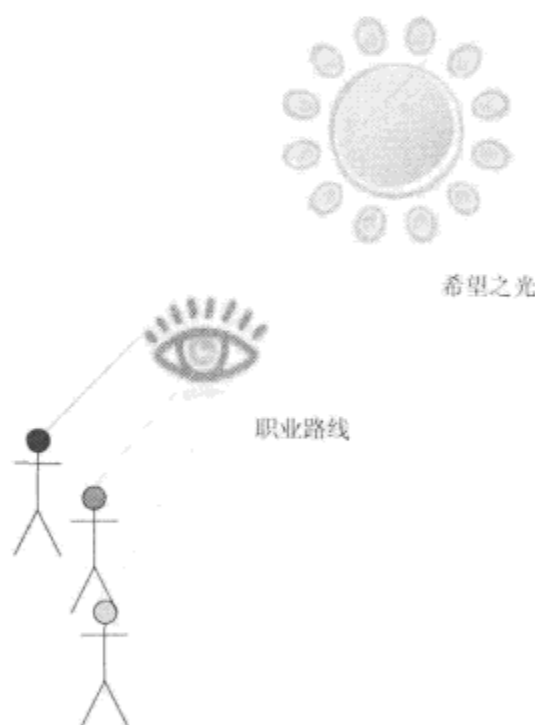


图 4-5 “让每一个测试人员都看到希望”示意图

4.2.1 回顾与思考微软的测试职业发展路线设计

按上节提到的，测试架构的设计可划分测试管理架构的设计与测试技术架构的设计两

方面，围绕这两方面，就测试职业的主要发展路线也可从管理线与技术线进行划分。伟大的物理学家牛顿说过“如果说我比别人看得更远些，那是因为我站在了巨人的肩上”，下面我们来看看微软这个软件巨头的测试工程师职业发展轨道。

【案例】微软的测试工程师发展路线

微软的软件测试工程师叫软件开发测试工程师（Software Development Test Engineer, SDET），与业界的普遍叫法不一样，其中发生一些有趣而又值得我们思考的故事，见本节的小故事“我们不能称他们为软件质量工程师（SQE）”。

在微软，测试人员有两种发展路线的选择，做独立贡献者（Individual Contributor, IC）或做主管经理。所有低级别的工程师都从独立贡献者做起，即从技术线的软件开发测试工程师做起。作为测试独立贡献者，从 SDET1（也叫 IC1）开始，一直到合伙人 SDET5（也叫 IC6），如表 4-1 所示。

表 4-1 从 SDET 独立贡献者开始的职业发展历程简介

职业阶段 名称	软件开发测试工程 师 1（SDET1）	软件开发测试工程 师 2（SDET2）	高级软件开发测试 工程师（SDET3）	首席软件开发测试 工程师（SDET4）	合伙人软件开发测 试工程师（SDET5）
对用户 的影响力	从产品支持服务 部门和其他的渠道 来探访用户的反 馈，从而阐明产品功 能并写出测试用例	直接与用户交 流，从而提供有建 设性的产品功能反 馈和开发测试用例	对用户关心的产 品整合和特定使用 场景，为角色的创 建提供方案，以达 到客户期望	具有直接建立用 户关系的技能，促 进用户和产品部门 之间的交流	领导覆盖整个产 品线的客户需求的 深度理解，从而促进 产品设计
对测试 的影响力	阐明产品功能如 何运行，以避免模 糊不清的要求	提供有建设性的 反馈意见，以提高产 品规范和技术设计	识别有可能引 入漏洞的高风险 设计模式	在主要产品上领 导测试方法和技术 的创新	在整个产品线上 领导测试方法和技术 的创新

从表中，我们可以看到作为测试领域的独立贡献者，他的级别晋升主要是根据技术深度、技术广度和影响力范围来区分的。一个 SDET1 通常处于学习测试、学习关于微软怎样开发软件、如何在一个定义明确的产品功能中找寻漏洞的阶段。而一个经验丰富的 SDET 也许更专攻一个领域，比如性能或者安全性的测试。当一个 SDET 达到合伙人级别的时候，他应该具有丰富的、不同方面的经验，也许还担任过一段时间的测试架构师。影响力的范围从一个狭窄定义的产品功能扩展到一个系列产品的功能、一个完整的产品，直到最后的一个产品线，影响力范围在横向、纵向上延伸。

测试管理是 SDET 的另外一条主要的职业发展轨道。一个测试管理者可以在测试工程师梯队里向上发展，管理更大的团队和领域。如表 4-2 所示是微软测试管理者的发展路线介绍。

表 4-2 从 SDET 管理开始的职业发展历程简介

职业阶段名称	软件开发测试主管	软件开发测试经理	软件开发测试总监
职业阶段	主管	经理	总监
产品范围	产品功能 一个 SDET 主管的工作范围通常是一组产品功能，或自成一个小子系统的非常复杂的产品功能，或者组件，或者一个简单的产品。产品功能的例子包括语音识别服务器、C#编译器、Microsoft Office Power Point 的图形引擎和 IP 栈	产品 一个 SDET 经理的工作范围通常是一个主要的产品，或构成一个产品的非常复杂的产品功能，亦或是一条简单的产品线。SDET 经理是产品线的主要贡献者。产品的例子包括 Word、Microsoft Money 和 Windows 内核	产品线 一个 SDET 总监的工作范围包括代表一个损益中心的产品线，或一条产品线下的非常复杂的系统或结构。产品线的例子包括 Windows、Office、MSN 和 Microsoft Exchange
招聘职责	领导一个组的招聘流程	积极地优化整个团队的招聘流程和实践	领导一个覆盖整个产品线的全局有效的招聘计划

像业界的很多公司一样，微软的 SDET 主管是也是测试管理的第一级，一个 SDET 主管通常管理一个由 2 ~ 10 人组成的小组。但有一个特别且很重要的点，就是在决定一个 SDET 主管能否在职业道路上继续前进和发展的因素中，技术复杂度和他的技术水平远比管理的测试组的大小重要。SDET 主管自己也需要做很多的测试、编程、分析和记录软件漏洞的工作。随着管理的团队的规模越来越大，主管需要承担的管理职责就更多。通常，软件开发测试经理管理着 15 ~ 50 人的团队，对于测试经理来说，很少需要亲自做具体的测试工作。但测试经理仍然需要懂得技术，但要求他们多注重建立测试的流程和工具，而不是在具体的功能测试上。另外，测试经理会花很多时间培养和提高其测试团队的素质和技能。

小故事：我们不能称他们为软件质量工程师（SQE）

Grant George 对会议室里的人说：“好了，让我们暂时把他们全都叫斑马好了。”我们正在辩论测试工程师的职称应当是什么，辩论已经进行了几个小时，但仍没有达成共识。

“我们都同意软件测试本质上是一个工程问题。我们都同意出色的软件工程师需要有很强的工程学基础，在理想的情况下最好有计算机科学方面的技能。我们也都同意一个优秀的测试工程师和开发工程师有所不同，他们都拥有内在的测试 DNA。”一边说着，Grant 离开座位走向白板，手里拿着记号笔。

我环视了一下会议室里的测试总监们，如果把出席这个会议的所有的微软测试管理经验加起来，要有 200 余年。有些人随着微软 Windows 一起成长，有些人在微软 Visual Studio 部门工作，还有些人来自 Web Services 部门。Grant 从他在微软 Office 工作过的背景来看待这个问题。他同时也是这里所有人中资历最高的，他是当时唯一的一个微软测试

副总裁。

Grant 开始在白板上写下一个列表，“测试DNA应当具有在系统范围内思考问题的本能、分解问题的技能、对提高产品质量充满热情、喜欢研究事物如何工作、又怎样能被搞坏。”他放下记号笔，注视着会议室里的人们，“这其实正是测试工程师不同于开发工程师的地方。通过测试软件，我们把这些 DNA 和工程学技能结合起来。我们选择的字应该反映这个事实，并且对我们想聘用的人具有吸引力。需要表明我们使用软件开发技能去驱动测试工作。”

“我们已经建立了所有的这些，Grant。”桌子边有人插话道，然后陷入了沉默。

“我想选 SDE/T。我们过去使用它，本质上描述了你们上面所讲述的，但我们不清楚使用 SDE/T 的过去。” Gregg 说道。

“我们使用不带斜杠的 SDET 怎么样？” David White 提了个建议。他是人力资源部门的职业模型项目经理，和这些公司级测试领导团队一起合作，统一制定所有测试工程师在微软的不同职业发展轨道，“它可以使你很快地向学校和社会上的待聘候选人表明，我们需要软件开发程序员，但他们的工作集中在测试方面。”

“这个名字比 SQE 好。”一个来迟了的只能坐在侧面桌上，桌下就是垃圾箱和废品回收箱的人喊道，“SQE 总是使我想起那些在马路交通信号灯边上拿着刷子要给你洗车窗换钱的家伙。”

Windows 测试部门的总经理 Darrin Muir 插话道：“我喜欢这个名字。很简单，只把斜杠去掉就好。”

我们又辩论了一个小时，最后决定微软所有测试工程师的新职称是 SDET，SDE 和 SDET 的不同处归为：激发工程师致力于测试问题而不是开发问题的核心 DNA。

公司在成立之初，一般在前几年，考虑更多的是在竞争激烈的商场中如何生存的问题。当有一定的业绩，产生效益之后，公司的规模扩大，向中型甚至大型企业发展时，人才的管理是关键。如何留住、吸引更多的优秀人才，让他们能把工作当成自己的事业，这一点很重要。一个好的企业凭什么吸引到优秀人才，除了高薪与福利待遇外，让处于不同阶层的人都能看到希望，可以称得上是管理架构设计中的上策。如果引导得当，对员工有着磁场般的吸引力。对软测而言，在国内仍处于发展初期，但随着市场需求的不断攀升，在这块职场领域打拼的人士也将越来越多。像其他专业领域一样，测试从业人员的技能、经验、素质也是参差不齐的。有不少测试人员是在不知不觉中踏入了这个行业，如一些应届毕业生，公司在校园招聘时以软件工程师的名义招入，实习后被分在测试岗位；也有些朋友原做开发，后由于公司成立专门的测试小组，自己便成了其中的一员，等等。当然，也有一部分人士求职时目标是很清楚的，就是想做一名测试工程师，但工作一段时间后，感到目

标模糊，不知该何去何从。此时，公司如能提供一个合理的职业发展路线平台，符合各阶层测试人员找到自己位置与目标，是实实在在地能让员工与公司达到双赢的举措。下面是借鉴国内外一些大公司的做法，以及最佳实践而设计的测试职业发展路线，具体实施时，可根据公司的性质及要求的不同，自行修改，补充完善。

从方向性的发展路线来看，测试职业本身主要有两条主要发展干线，即管理线与技术线，还有一种就是“转其他职业”，即转行。其中一个人在管理线与技术线上发展的过程中，可以不断调整，交叉进行，以找到适合自己的位置，如图4-6所示。

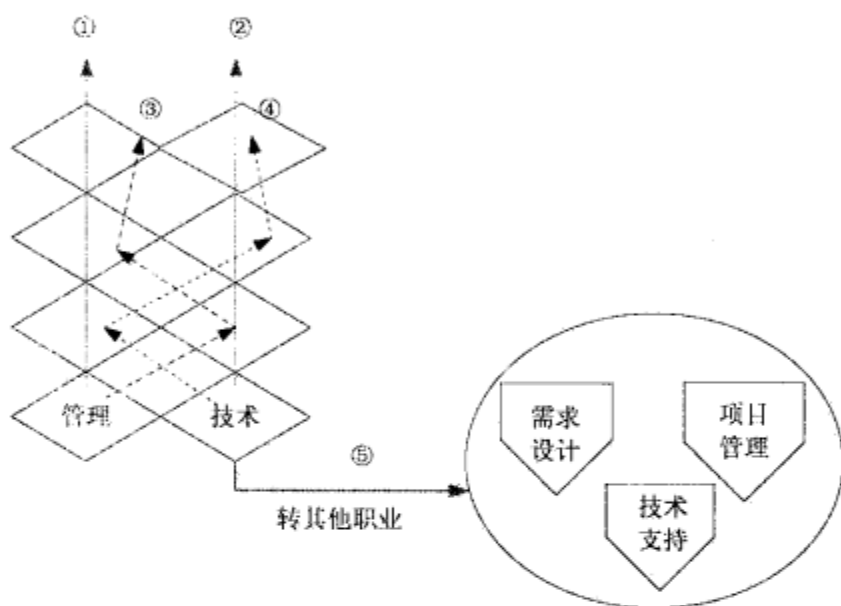


图 4-6 测试职业发展路线设计

关于职业道路上的交叉发展，笔者认为这是一种很重要也很人文的做法。笔者认识的一位某大型 IT 公司的软件开发人员，在公司的创立之初，他曾带着一个小团队，为公司立下了汗马功劳，他在技术上可说是让老板无可挑剔。当然，在技术职称上他很快得到了晋升。由于公司规模快速扩展，开发人员数量急增，在管理上很缺人，由于过去他的业绩好，得到了老板的器重，自然地他成为了部门经理。但在部门经理的位置工作了一年后，他觉得并不适合自己，不但没有做出一些可喜的成绩，反而由于过于追求完美，在与其他接口部门之间的合作关系上处理得并不那么如意。最后只好主动向老板提出，重新回到带一个小团队进行技术攻关的工作。这种做法，在微软公司是再普通不过的事了。但业界有很多人认为，只有做了管理者才是成功的，这种看法是错误的。管理线与技术线是一个并行发展的职业双通道，例如在一些大型 IT 公司里，一个资深的测试工程师与一个测试主管是同级的，尽管他们是处于不同的发展轨道。

还有一个更加灵活的发展路线，就是图 4-6 中的“转其他职业”。由于一些岗位的特殊要求，从校园或社会招聘，都不容易胜任，如需求设计人员，需要精通公司产品的业务，

外招人员培养成本较大。有些公司在进行岗位招聘时，就是特意把招聘名额给测试，一段时间后，选择合适人员转岗到需求，这种属于公司人才培养模式的主动转岗。还有一种是个人根据自身情况，在测试岗位工作一段时间后，转向技术支持或市场等岗位的。关于转岗的看法，也有不少人错误的观点，认为转岗就意味着失败。俗话说“三百六十行，行行出状元”，当发现自己确实不适合在测试行业继续发展时，又何必在一棵树上“吊死”。当一扇门为你关闭时，另一扇门也正为你敞开着，正如“条条大道通罗马”。

接下来分别就技术线与管理线的测试职业发展架构设计的细节进行讲解。

4.2.2 架构合适的测试技术发展梯队通道

首先，软件测试与软件开发一样，是一项技术性很强的工作。至于有不少朋友认为测试工作技术含量不高，笔者认为最主要是因为他们还没有意识到测试技术的深度、难度与重要性。

记得在 2009 年 51testing 软件测试网举办的一次测试沙龙上，一位演讲嘉宾（来自互联网行业名企的测试总监）在对测试职业发展规划中提到“技术决定未来”，即技术的重要性，对于打破测试发展的瓶颈问题，笔者在多年的测试历练中，非常有同感。其实，只要你留意，就在我们身边，“技术决定未来”、“技术为王”的事例，举不胜举。下面便是两个鲜活的例子。

【案例 1】车企：技术决定未来

以“畅想绿色未来”为主题的 2010（第十一届）北京国际汽车展览会（Auto China 2010）于 2010 年 4 月 23 日—5 月 2 日在北京中国国际展览中心隆重举行。从各大企业的参展阵容中，我们可以发现，各大自主车企的未来发展战略正在不约而同地转向以“技术为王”的时代，即使自主车企也不例外。

来自腾讯汽车网

【案例 2】索尼：技术为王

支撑品牌获得持续发展的，永远都是领先的技术。技术领先让索尼电视数十年立于不败之地，索尼无法割舍技术情结。索尼全球总裁中钵良治曾表示：“分析过去索尼业绩低迷的原因，首先就是有一段时间缺乏对技术开发的重视，索尼毕竟是‘技术的索尼’，这是必须被扭转的问题。”

来自 2007 年《世界经理人》杂志

软测作为软件工程体系的一个分支，在测试技术线路上的发展，亦有一个由低到高逐

级递进的梯度进阶规则，如图 4-7 所示。

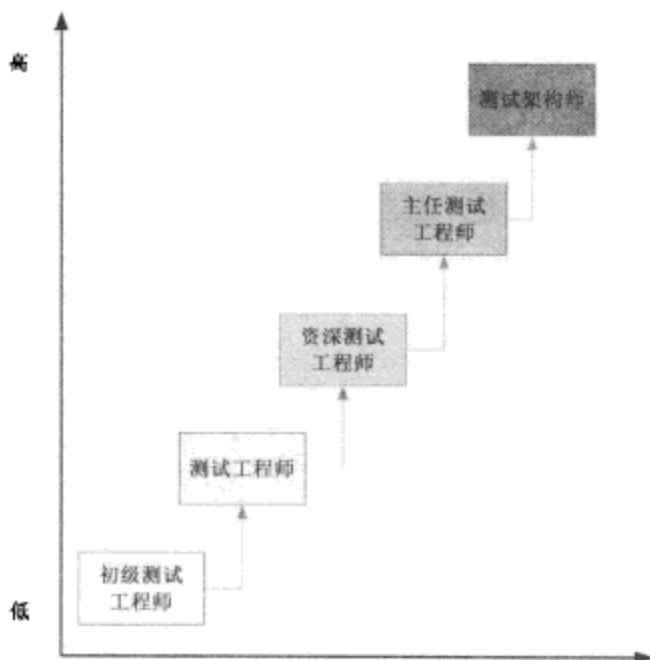


图 4-7 测试职业技术线发展阶梯

下面就技术线路发展各阶段对测试人员在技术技能上的要求，以及对象进行介绍。

- 职业发展一阶：初级测试工程师。
 - 技术技能要求：执行测试用例，记录测试发现的 Bug，跟踪 Bug 生命周期状态，回归 Bug；参与项目测试方案与用例设计等的评审。
 - 对象：一般为刚踏入测试行业的新手。
- 职业发展二阶：测试工程师。
 - 技术技能要求：以设计模块级测试方案、测试用例、测试代码为工作重点，执行测试用例为辅；组织模块测试设计评审；参与模块级开发设计方案评审，能独立完成规范的测试流程各节点的工作。
 - 对象：一般为有 1~3 年经验的测试人员。
- 职业发展三阶：资深测试工程师。
 - 技术技能要求：设计某项目总体测试方案，制订测试计划；对项目的某类或某特性进行测试，如自动化测试、内存泄漏、性能测试、安全性测试等；对有一定技术深度或难度方面的测试有独当一面的能力，且收到效果；培养测试新人成为合格的测试工程师。
 - 对象：有 3~5 年项目测试经验者。
- 职业发展四阶：主任测试工程师。
 - 技术技能要求：制定某类产品测试总体策略、测试流程，制定相关测试规范、

指南；负责某类产品测试平台建设；指导重点测试方案的设计，对测试设计有一定的创新能力，并收到效果；资深测试工程师的导师。

➤ 对象：一般为有 5~10 年的测试经验者。

● 职业发展五阶：测试架构师。

➤ 技术技能要求：负责某产品线测试策略、测试方法、流程规范的制定；规划、设计和开发测试平台；为了不断降低公司研发成本而进行新测试技术的研究、实践和推广；技术线上测试人才梯队的结构设计。

➤ 对象：一般为 10 年以上测试实战历练者。

对技术的追求，是永无止境的。上面设计的 5 个阶段，并没有绝对性，每一阶段还可以再细分，如资深测试工程师，可根据公司的具体情况再分为一级资深工程师，二级资深工程师、三级资深工程师等。

是否适合在技术线路上发展，主要取决于个人的能力特点。但有一点我们在进行个人职业发展规划时是不容忽视的，那就是与管理线路上的发展相比，技术线上的发展是有限制的，不存在管理线路方向发展时职位需求不断收缩的金字塔问题。

4.2.3 架构合适的测试管理方向发展轨道

一般情况下，先有一些测试人员，然后才有测试管理的概念，一个测试管理者，特别是基层的测试主管都是从测试工程师中提拔的。测试管理不仅仅是对人的管理，还包括对测试流程体系的建设。如图 4-8 所示是测试管理线路进阶图，如表 4-3 所示为测试管理线发展各阶段要求与特征。

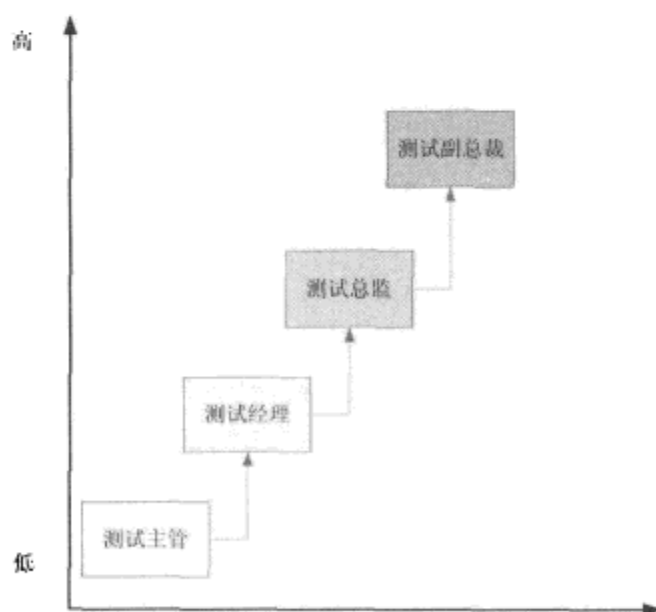


图 4-8 测试职业管理线发展阶梯

表 4-3 测试管理线发展各阶段要求与特征

职业阶段	测试主管	部门测试经理	测试总监	测试副总裁
产品测试与管理范围	1.负责系列产品中某类模块的功能或性能或自动化测试,例如手机的驱动测试、通信协议测试、游戏应用程序测试等 2.小组成员工作输出审核	1.负责某一系列产品的测试管理,包括测试规范、流程的建设 2.制定部门的管理目标和经济指标 3.制定测试人员绩效考核标准并推向实施 4.测试主管工作的指导与监督	1.监管整个产品线的产品测试 2.建立完善的测试组织体系,如各方面测试团队的组建 3.组织协调各测试部门,以实现公司的运营目标 4.监督绩效考核的实施	1.修订、执行公司战略规划及与日常测试工作相关的制度体系,以及业务流程 2.制定公司组织结构和管理体系、相关的管理、业务规范和制度 3.制订测试前瞻性发展计划 4.指导公司测试人才队伍的建设工作
团队大小	一般是 3~5 人	一般是 15~60 人	60~200 人	200 人以上
招聘职责	负责小组内的人员招聘	1.根据工作需要,规划部门的用人计划 2.负责与公司 HR 接口,积极优化整个团队的招聘流程和实践	1.规划整个产品线的用人计划 2.审核各部门的招聘计划	1.审批多个产品线的测试用人计划 2.协调公司各产线的用人计划
人才培养	培养测试技术骨干	领导、培养测试主管	领导、培养测试经理	领导、培养测试总监

测试管理线是测试职业发展的另外一条主要轨道,也是很多测试人员梦寐以求的路线。然而在管理线路上越往上走,职位需求就越少,也就是说机会就越少。就目前我们国内的公司而言,由于测试行业还处于发展的初期,设立测试总监、测试副总裁职位的公司还是寥若晨星。这也意味着我们要走的路还很长,长长的路上充满着机遇与挑战。

一名测试工程师在技术线路上发展时,主要是通过个人的努力给公司带来的贡献大小进行考核,而在管理线上,何谓一位成功的测试管理者呢?不同人可能会有不同的答案。让每个测试人员都能看到希望不是只挂在口头上,更重要的是在行动上。下面是一个管理成功的最佳实践案例。

【案例】他,在恋恋不舍中调离了

2010年春节前夕,某电子产品公司正在进行史无前例的组织结构重大调整。杨刚是公司软件测试部的一位测试工程师,由于刚加入公司还不到1年,对正在测试的产品业务并不很熟悉,即便一直加班加点学习、理解、分析需求,但所负责测试过的模块,还是漏了一个严重的Bug到客户端。也主要因为此事,杨刚的年中考核得了个C。由于种种原因,杨刚对此结果一直耿耿于怀。后来,主管陈A与他一起查找了原因,做了详细严谨的漏测

分析，提出防范的解决措施，并在日后的项目中，不断地鼓励他，相信他，给予他提升的机会。工夫不负有心人，通过下半年的心态调整、努力，杨刚的考核结果终于由 C 扭转到了 A。但“好景不长”，适逢公司的结构调整，在领导的要求下，杨刚被调到一个新的核心产品线担任测试工作。当主管与他谈及调动的事时，只见其默默地坐在自己的位置上一句话也不说，恋恋不舍的样子，快要掉出来的泪水在眼中打转。

此时无声胜有声，给每一位员工都能看到希望，相信这个小故事是一个很好的例子。

小贴士：

爱因斯坦说，一个人对社会的价值首先取决于他的感情、思想和行动对增进人类利益有多大作用。

4.3 万里航行总舵手——业务测试架构的设计

目前，国内的很多公司，包括一些知名大公司，都可能还没有这个职位，但应会有这样一个角色的存在，比如这个角色落在测试经理或是测试主管的肩上。笔者不敢称自己是一个专业的测试架构师，只是有一天发现业界有这个职位时，并对着职位描述的定义，发现自己很幸运地在不知不觉中做了一些这方面的事情。

对于架构，更具体一些指架构模式，如第 6 章介绍的关于测试对象分析的三层架构模式。一边是深不可测、充满挑战的技术与艺术的高度体现，一边是“又恐琼楼玉宇，高处不胜寒”的担忧。高深的东西如何平民化，即那些高调的架构，能不能具体应用到工程实践中，很好地达到预期，而不是成为束之高阁、脱离实际的一堆废话或模型。这里站在项目测试的实用角度，总结工作中的经验与教训，提出架构设计的操作模型，如图 4-9 所示。从图中我们可以看到一个完整的测试架构设计过程包括以下几个阶段。

- 业务测试框架设计：它包括业务测试技术与流程管理两个部分，基本框架的设计离不开业务需求与公司流程体系。其表现形式可以是一种测试方法、一块代码程序、一系列的流程规范等。
- 提取测试需求：广义上理解，包括与测试工作相关的业务及非业务需求，只有有了需求(工作中出现的问题也可认为是一种需改进的需求)，才可进一步完善框架。
- 决策/部署测试策略：为测试需求服务的一系列解决方案。
- 开发测试套件：具体解决测试需求的措施集，如测试用例集、脚本程序、测试工具等。

这4个阶段，它们之间是相互作用，相互影响的。细心的读者也许已注意到，位于图中内侧的“提取测试需求”，它与测试框架的设计并不是一种直接关系，没错，它们之间的关系要通过后续的工作体现在框架中。可以理解为一个新的测试项目开始了，以新的测试需求为起点，通过部署测试策略，开发新特性的测试套件，来完善测试框架。如此往复，依托一个个测试项目，不断改进、壮大测试框架。以使后续的项目测试能重用测试框架的内容或方法，并使整个测试过程始终在有序可控的状态下进行，最终能以高质量且减少项目的整体测试时间来完成测试工作，这也是架构设计的最主要目的。

对这4个阶段，可以理解为它是一个系统级的最顶层划分，对于每一个阶段，它又可划分为不同的节点。其包含的意思及操作的方法，将在接下来的章节中进行详细讲述。

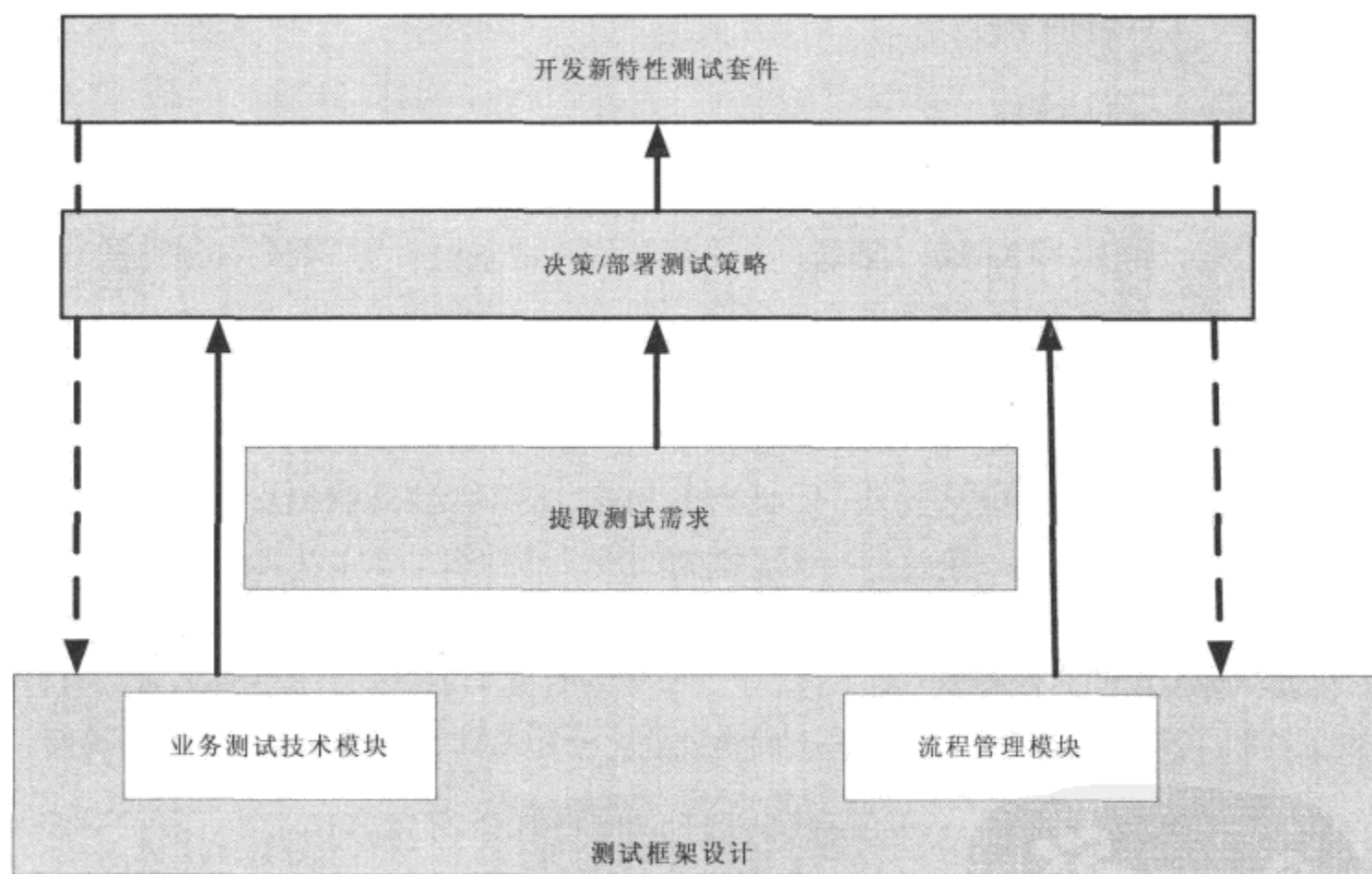


图 4-9 测试架构设计过程示意图

一个好的架构，只有在应用中收到实际的效果后，方显它的价值，比如节省了多少测试时间或提高了测试的全面性等。

主动向他人提需求，是一种架构能力的体现，从而影响开发、需求，甚至其他用户、市场部门为测试部门服务。测试架构设计，需重视过程，它是个不断发展的过程。架构必须由经验丰富的设计人员设计，很大程度上依赖于过去项目的成功与失败的经验。但是正因世界上万事万物都在不停地发展变化着，软件开发的方法、模式、具体项目的要求也不

同。随着过程中遇到问题的不同，需要做出快速响应，并进行合适的调整，从而提高架构的应用性，丰富它的内涵。提升它应用的高度与广度，为它画上更大的外延，这也是符合事物的发展规律的。

4.4 测试建设之基石——测试框架的设计

俗话说“万丈高楼从地起”，大家都知道，不管盖的房子是大还是小，都是有地基的。那么，这个测试框架，它就相当于一个地基。地基越牢固，抵抗外界干扰能力越强，如抗震、抗台风暴雨等。类似地，测试框架结构合理、重用率高、易移植、扩展性好、易维护，那么它就是一个好的框架。

4.4.1 相框与测试框架

说起相框，如图 4-10 所示，那是再平常不过的日常家居用品，我们都很熟悉它。记得小时候，家中的相框是专门请木匠师傅做的，然后由油漆师傅上漆，并画上花边以装饰，最后当然少不了压上玻璃。记忆中，那时的相框不管大小，清一色都是长方形的。后来，随着人们生活水平的提高，出现了各种各样的相框，如椭圆形的、心形的、月亮形的，不仅造型标新立异，材质也五花八门。此时的相框，已不仅仅是镶相片的相框了，更是一种点缀家居生活的工艺品。说这些与测试框架的设计有什么关系呢？真是风牛马不相及的废话吗？其实未必，反过来细细思考这些看似毫不相干，甚至相差万里的事物，它们之间存在着一种内在的必然联系。关于这一点，著名教育家丰子恺先生早在其艺术教育理论中就提出了“一通百通”的观点，指出不同类别的艺术和不同艺术形式之间具有相通性。



图 4-10 相框

初次看到“测试框架”一词，与大家的感觉一样，觉得很抽象，也很高深，有一种“高深莫测”的感觉。框架，在技术上是指针对某一特定领域解决方案的完整表达。框架设计，

就是设计一种完整的解决方案以解决某一特定领域的问题。框架并不是空中楼阁，提出来玩玩而已，是要为解决实际工作问题而服务的。框架有它的层次骨肋，这些层次是解决某类问题的方面；骨肋是一种支撑，是一种解决问题的方法。测试框架，是属于软件工程领域解决测试相关问题的方案，它里面放的是为测试活动服务的行为与方法，以及一些流程管理规则或规范，是看不见摸不着的知识库。为解决不同类别的测试问题，测试框架有类别之分，如功能测试框架、自动化测试框架、性能测试框架、流程规范框架等。而相框是用来展示照片，当然也可以陈列广告图片、宣传语等，里面放的内容是看得见、摸得着的，它为解决我们实际生活中的家居陈列或市场宣传问题而服务。后来，由于需要的服务对象变了，人们的生活品味也变了，也就出现了各种形状、材质的相框，带来了相框家族的必然发展。

以上观之，两者的表现形式是迥然不同的，相框是一种有形的实体，即使有着后期工艺美术的设计衬托，它仍是看得见，摸得着具体的有形实物；而测试框架是一种解决软件测试领域问题的技术方法与流程管理的抽象框架，是摸不着看不见的无形产物。但是，它们的本质是相同的，那就是都为解决对应领域的问题而出现，为解决对应领域问题的细化而发展。

4.4.2 化抽象为具体——测试框架内容

测试框架主要包括两方面的内容，如图 4-11 所示，一是业务测试技术，二是流程规范管理。前者是完成任务需用到的测试技术的集合，后者是技术应用过程中各环节需如何控制的策略。

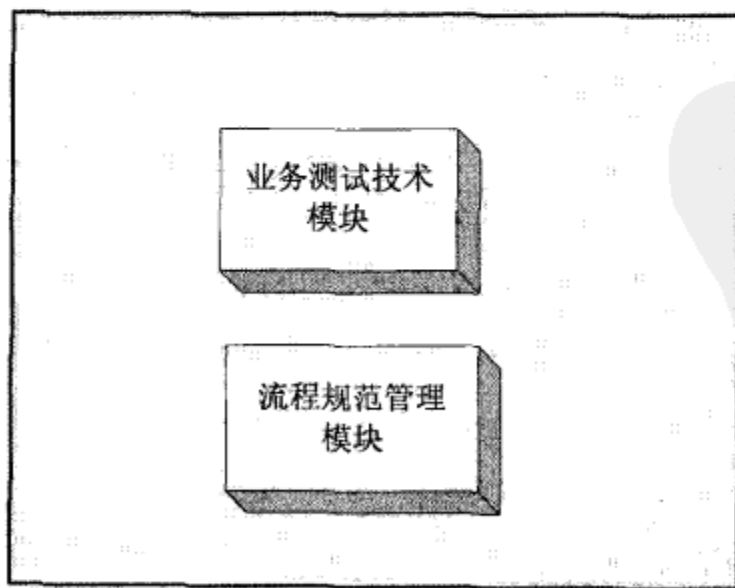


图 4-11 测试框架内容

业务测试技术模块与流程管理模块，可以理解为前者是测试技术线上的平台。后者是测试管理线上的平台。技术与管理对于一个完整的测试体系而言，它们之间并不是孤立存在的，而是相互依存相互制约的，如图 4-12 所示。

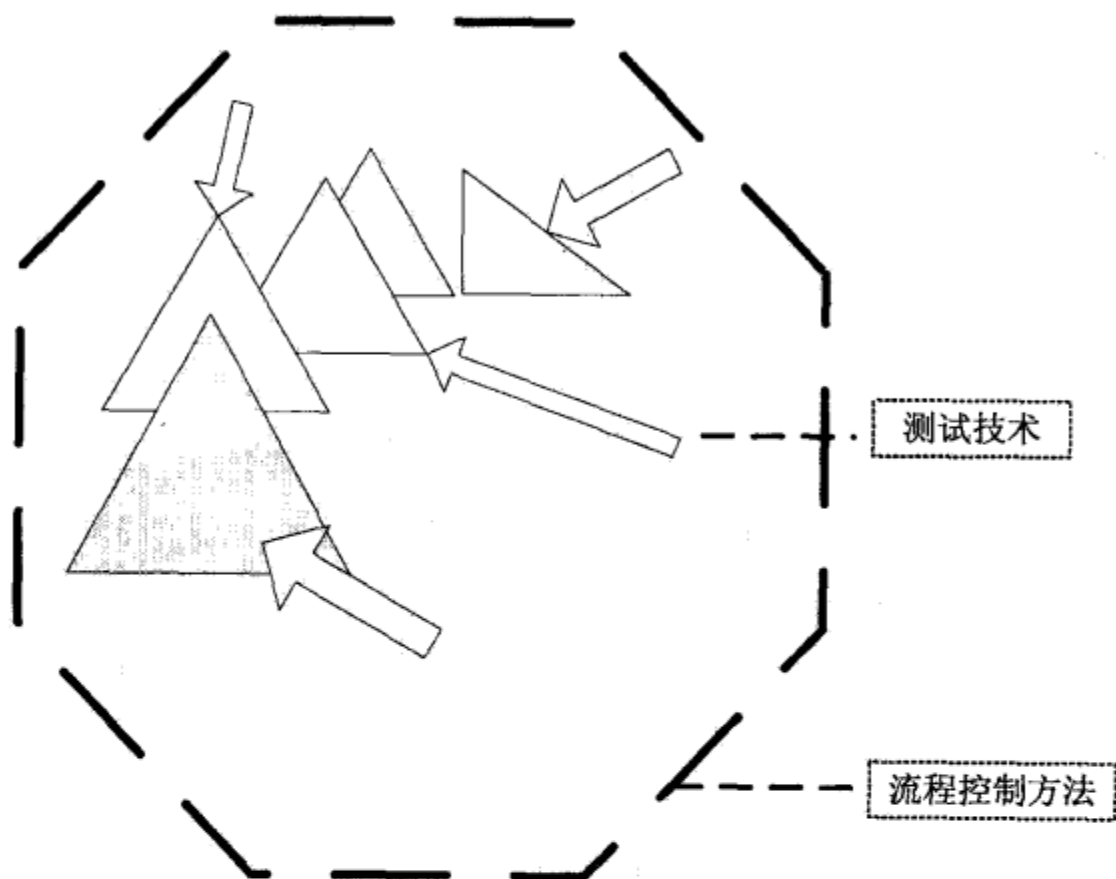


图 4-12 测试技术与流程管理在测试框架中的关系

业务模块就像一座座小山，测试技术如同图中的箭，多边形的边框则表示软件产品生命周期各阶段的流程规范管理手段。技术是攻破一道道难关的钥匙，不同的难关需要不同的技术。俗话说得好“没有规矩，不成方圆”，分布在技术力量外围的流程规范很重要，可以说都是通过一桩桩、一件件教训总结出来的，是管理的一种手段。技术再强，如果没有规则是行不通的，就如同我们玩游戏一样，必须遵循游戏规则，方可交锋，比个输赢。但反过来，如果流程规范约束太多，或不灵活，束缚了技术的发挥，会让事情弄巧成拙，流程规范成了绊脚石。比如用例编写规范和自动化脚本编写规范，这些规范是必要的，大家工作时遵循这些规范，设计出来的用例或代码可读性好、易维护。但如果规范过于约束，影响设计开发效率，在项目压力下，工程师做不到一直坚持遵循规范。最后往往导致“漂亮的外表后面却是一堆醒醒的代码”。换句话说，要求的功能是实现了，但是实现这些功能后面的代码却是一团糟糕，人见人畏，不可维护，好像有成堆的 Bug 危伏四起。看到这样的代码或用例，人人心里都在叫苦，重写的念头会异常强烈。所以只有技术与流程相结

合，才能组成一个强有力的测试框架。

测试技术的框架设计，犹如编织通用的不同类的网，以便捕到不同类的鱼。同样地，我们可以设计不同类的测试框架，以能更专注、更有效地发现不同类的 Bug。根据常用的测试类型，可把业务测试框架的类别分为功能测试框架、性能测试框架、自动化测试框架等，如图 4-13 所示为业务测试技术框架组成示意图。

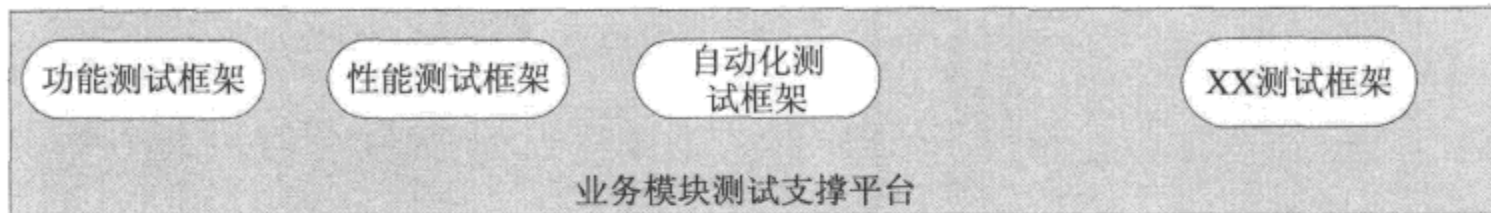


图 4-13 业务测试技术框架组成示意图

在测试流程管理框架的设计中，我们可以考虑测试规范系列、测试工作流程系列、测试设计模板系列，如测试方案设计模板、测试用例设计模板、测试报告模板等内容，如图 4-14 所示。

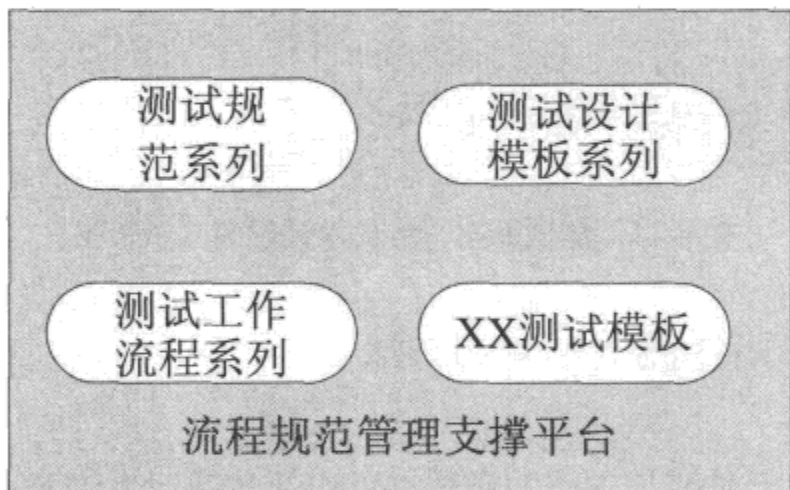


图 4-14 流程规范测试框架组成示意图

业务测试技术模块、流程管理模块都是框架的抽象。从图 4-12 中可看出，它们分散在不同的地方使用。流程控制方法的位置也是相对的，在不同项目的应用场景、不同的阶段，它需要做适度的改变。正如“3.4.2 让大家一起忙起来”提到的一样，它们都需要在过程中得到发展与变化。

业务测试技术框架与流程规范管理框架，不管它们之间的关系如何，能给新的测试项目带来多少重用部分，都需纳入到新项目的测试套件设计中，最终为测试项目的各业务模块服务，如图 4-15 所示。

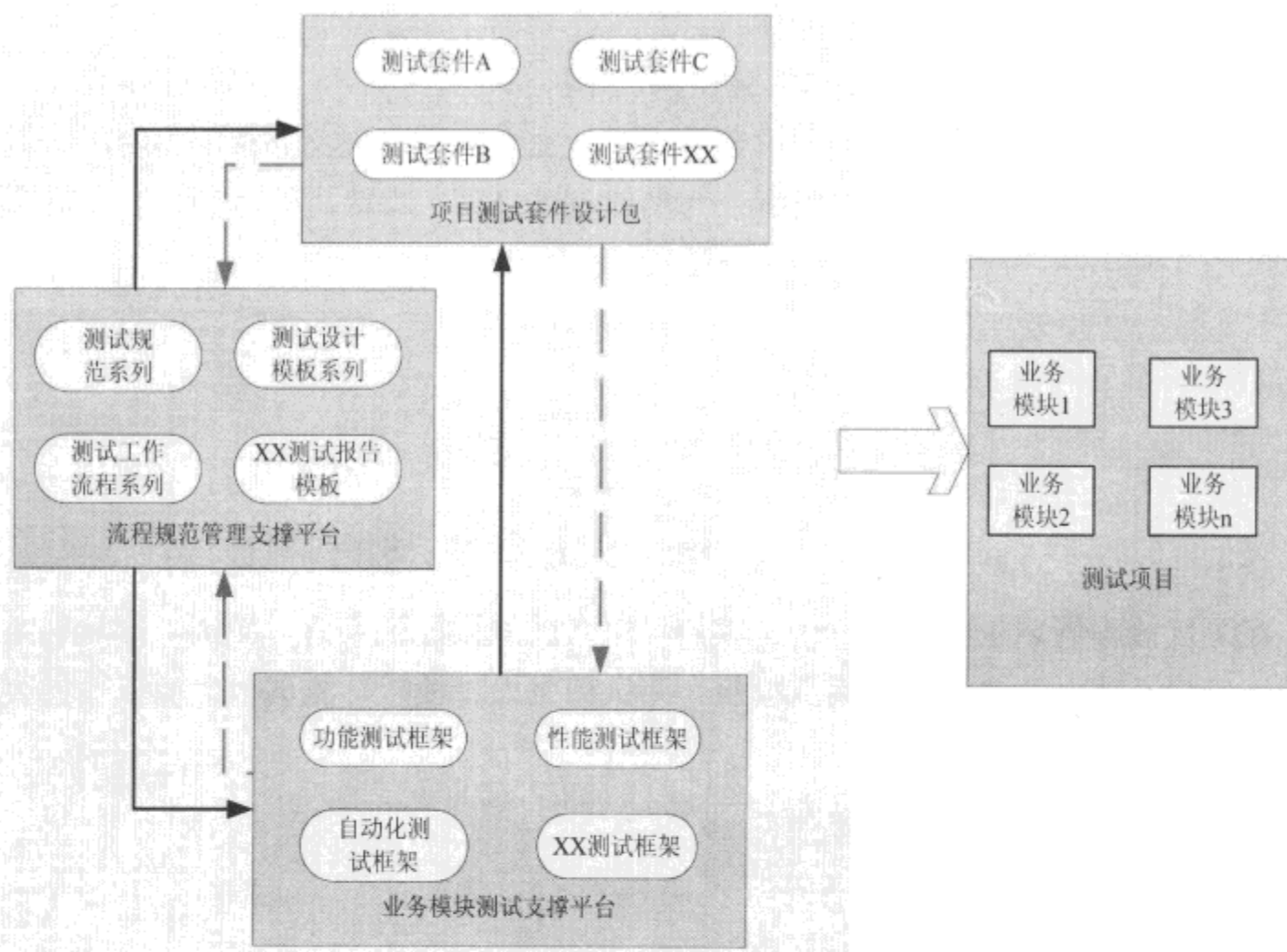


图 4-15 测试框架与测试项目的关系示意图

4.4.3 突破起点——搭建测试框架的方法

测试框架设计的主要目的是项目的测试都能以框架平台为基础，并在框架的控制内进行实施，使得在不断的项目测试历练中积累的经验能够沉淀、传承、复用，从而提高测试的整体效率，降低项目研发与维护成本。就框架设计的具体方法，比如我们该如何操作去突破从无到有，到进一步的改进，再不断地改进，以致日臻完善。笔者接下来结合自身的经历，谈谈具体的做法。

一个公司，在做第一个项目时，谈测试平台的建设，可能性不大。但如果测试负责人有过带项目的经验，在执行相关的测试设计时，即使没有这方面的安排，也能感受到这方面的必要。如果你是第一次做项目，比如刚毕业的学生，可能没有这方面的经验，但当做第二个类似项目时，对于相同或类似的产品功能需求及性能需求，自然而然你会知道如何去做，这是因为你有了一定的积累。这个积累不仅仅指解决问题的方式方法，是指有了一些可重用或部分可重用的测试套件，如某测试对象的测试方案、测试用例、自动化测试脚

本、一系列的测试数据、测试工具等。一般一个公司总是有自己的产品方向的，不可能什么产品都做。另外，在项目研发中如何节省人力与时间成本、提高项目开发效率、缩短开发周期，向来是公司老板的追求。这样，我们在做第二个类似项目的测试时十分有必要考虑平台化的测试需求，让后续的项目能在框架的基础上进行工作。这就要求框架是通用的、可重用的，当然重用的范围我们可按前面也提到过的测试类别来分，如业务功能测试、自动化测试、性能测试等。现就框架的搭建提出如下几点看法：

- 业务功能测试框架，可重用的是测试思路，框架中可列出各测试点及其测试方法。
- 测试数据，特别是一些影响性能测试的数据，包括一些测试数据生成程序，集中一起管理，并写好使用说明，以便各项目测试过程中需要时取用。
- 自动化测试脚本、接口函数等测试套件，需无条件纳入配置库进行版本管理。对各项目通用的接口函数考虑封装为独立的中间库，把此库作为自动化测试框架的元素之一进行扩展与维护。
- 回顾 Bug 库中历年发生的 Bug，对这些 Bug 进行分类，分析这些 Bug 的发生原因，拿出日后如何控制的可行方法，并对原有的测试框架做改进或补充。
- 测试文档设计模板，可结合公司内部开发流程要求，重点考虑测试文档本身应有的内容，模板中给出例子为宜。
- 制定测试设计评审机制、测试用例设计规范、Bug 录入规范、测试代码设计规范，并在项目的执行过程中不断完善。

小贴士：

测试套件包括测试方案、测试用例、测试报告、测试总结、测试数据、测试代码等一系列为某特定测试对象而服务的测试输出工件总和。

框架有它的优点，如与项目无关（跨项目）、可重用、易扩展。反过来，它也有缺点，如使测试设计在一定程度上受到约束，当不熟练规范的要求时反而会影响测试工作的进度。另外一个致命的问题就是“牵一发，动全身”，如果框架中漏了某个测试点，会导致用这个框架的所有项目都漏了这个测试点，直到发现了这个漏洞为止。

测试框架的设计是一个过程，正如前面所说，当反复测试相同或类似项目的业务时，自然而然萌芽了测试平台化的需求，从而有了首次的测试框架。随着后面一个个测试项目的开展，测试框架得到不断重用，同时随着不同项目需求的变化，失败教训和成功经验的积累，不断地推动着测试框架的更新、发展，如图 4-16 所示。

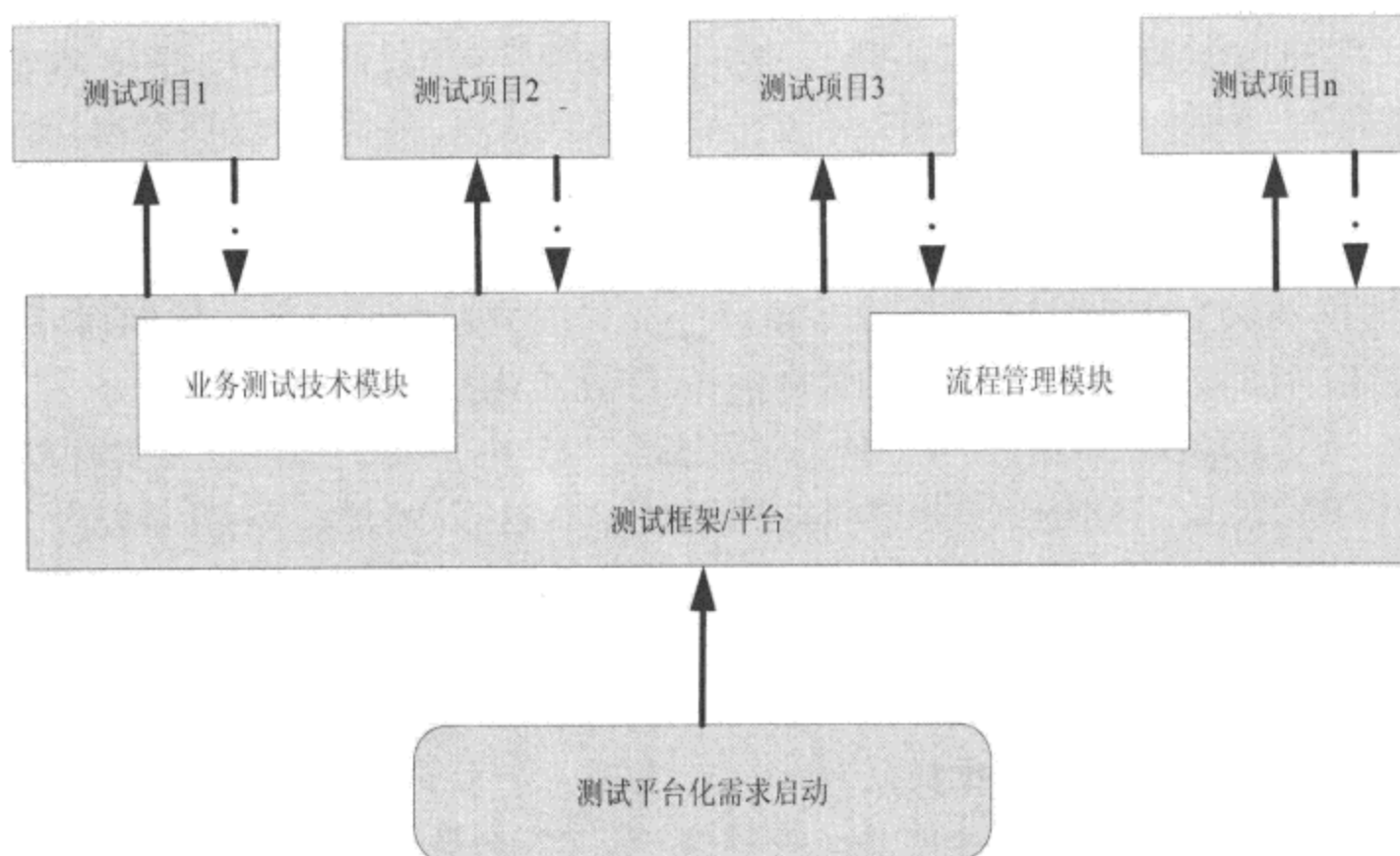


图 4-16 测试框架设计过程示意图

小贴士：

架构，没有最好的，只有更好的。没有完全适合您的，只有更适合您的。重要的是掌握方法，而不是生搬硬套。

第 5 章

测试需求分析与测试策略制定

需求，毋庸置疑，是软件设计与测试的来源，但是需求除了终端用户的功能需求外，还有设计性需求、可制造性需求、可测试性需求等。而这些需求，对于测试工作而言，最后都需转化为测试需求。本章从测试需求的介绍开始，与读者分享如何超越于需求文档之外，收集更多、更全面的需求，然后如何分析这些需求，特别是一些隐含需求的识别，从而提取方向性的顶层测试对象。接着为提取到的测试对象部署测试策略，重点介绍了各种测试技术的裁剪与合理应用的方法。主要体现在以黑盒功能测试为主，适当采用白盒测试，活用灰盒测试，部分功能或模块采用自动化测试的方法上。同时要着眼于专项测试，以突破某特性的测试模式，以使测试对项目软件质量在可靠性与稳定性方面做出更多的贡献。测试的计划与整个测试过程的跟踪、控制方法也是策略中需考虑的主要内容，在本章也做了介绍。最后就测试策略中需考虑但容易被忽略的其他因素进行了一番介绍，以飨读者。

5.1 从测试需求开始

需求一词，对于从事软件行业的测试朋友来说，已是再熟悉不过了。然而笔者对它的理解提升，却是在一次偶然的活动中。一次，在公司部门组织的一次春游活动中，而导游小姐的一句服务语让我久久难忘。当时车内每人都发了 1 瓶水，最后还余一些。在半途中时，导游小姐热情地询问：“尊敬的旅客们，我这边还余 5 瓶水，请问哪些贵宾尚有加水的需求。”话语一出，就听到一群人异口同声地欢呼：“太专业了！”需求，在软件开发过程中，有特别的含义。远离办公室，在大家心情放松的车上能听到如此熟悉的专业术语，

大家感到特别亲切，但又隐含着新意。服务源于需求，这是再正常不过的商业规则。测试是一种服务性的商业活动，测试需求的识别是后续测试工作的基础，也是起点。

测试需求主要来源于用户的业务需求，那么，该如何开始了解产品的业务，为测试任务迈开重要的一步呢？首先，要能识别测试需求，接着是分析此测试需求，最后确定并提取出测试对象，如图 5-1 所示。

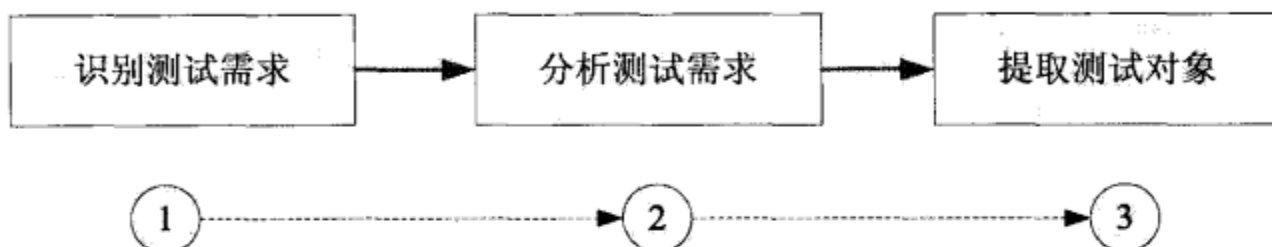


图 5-1 提取测试需求路线示意图

提取出了测试对象后，接下来需要确定对每一对象如何进行测试，拿出具体的方法及措施出来，这便是测试策略制定的问题。

5.1.1 多管齐下溯需求

料想测试朋友们，也曾遭遇过由于需求的频繁变化给后续开发、测试工作所带来的困扰，小则更改，大则原来的开发、测试工作被废弃。其中有需求管理的问题，也有市场压力问题，等等。总之，现实中需求就是有这种爱变的特性。话又说回来，爱变的需求，它是符合事物的发展规律的，如果不是这样，在我们工作生活的周围，又怎么会到处都充满着日新月异的新鲜事物？

需求，是开发工作的来源（输入），当然也是测试工作的来源。需求，要开发实现后才能变为产品，开发出来的产品是否符合用户的需求，需要测试来验证。无论是哪方面的需求，实现后都需要测试的验证。对测试需求的识别，也就是对需求的识别。针对爱变的需求，识别测试需求，对于测试工作来说至关重要的。对于这些变化了的需求（需求变更）或新增需求，如何及时获取正确且全面的信息，直接关系到测试的充分性与全面性。当然这与公司的需求管理有很大关系，在没有规范的开发流程支持或执行有偏离的情况下，测试人员的主观能动性发挥起着重要作用。

尽管有专门的需求文档，如产品需求或叫软件需求，对要做的产品功能有集中的需求定义。但是对于测试来说，不能只关注产品的功能需求，还要关注设计需求、非功能需求，如可靠性需求、可维护性需求等。如图 5-2 所示的需求类别，其中功能需求所占比例最大，约为 80%左右，接着分别是设计需求与可靠性需求。图中的比例关系在不同行业的软件中

要求会有不同，测试重点也就不同。

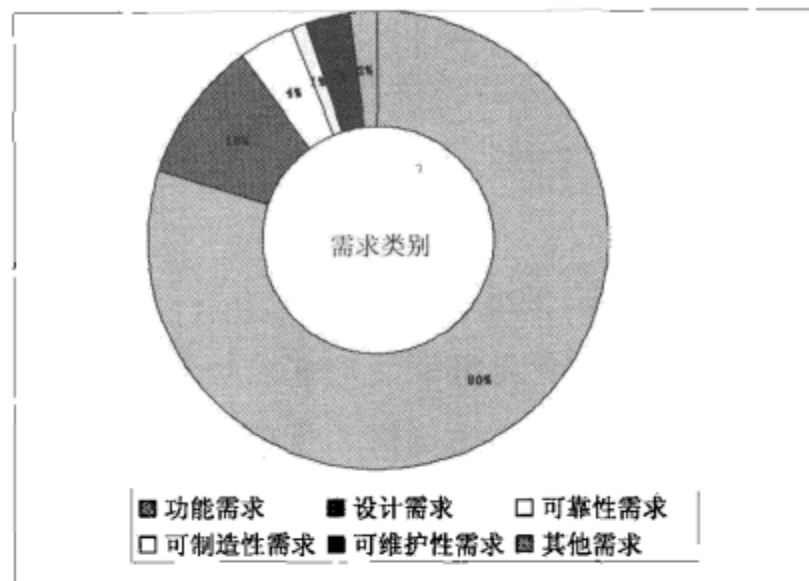


图 5-2 需求类别

无论是哪一类需求，在产品研发的过程中一直都会有变化，而这些变化常可能是体现在专家评审的会议纪要或邮件中，或 QQ 的即时讨论中等。按理来说，需求分析人员需转换需求到专门的文档中，但实际运作中，常会出现需求转换慢，或根本就漏掉了。但是测试不能漏，测试是软件质量的最后把关者。在这种情况下，我们需要把关注范围放大，以多管齐下的方式关注需求的入口，如图 5-3 所示。从各种可能的渠道及时获知需求信息，作为工作的来源，同时反推需求，要求把零散的需求文档化或纳入需求库，正式地给出测试的依据。从而使“测试尽早介入，开展测试相关工作成为事实”，并使测试影响着需求、开发成为可能（这与测试人员对系统及项目的熟悉程度，是否能给需求、开发一些合理或有建设性的建议有关）。

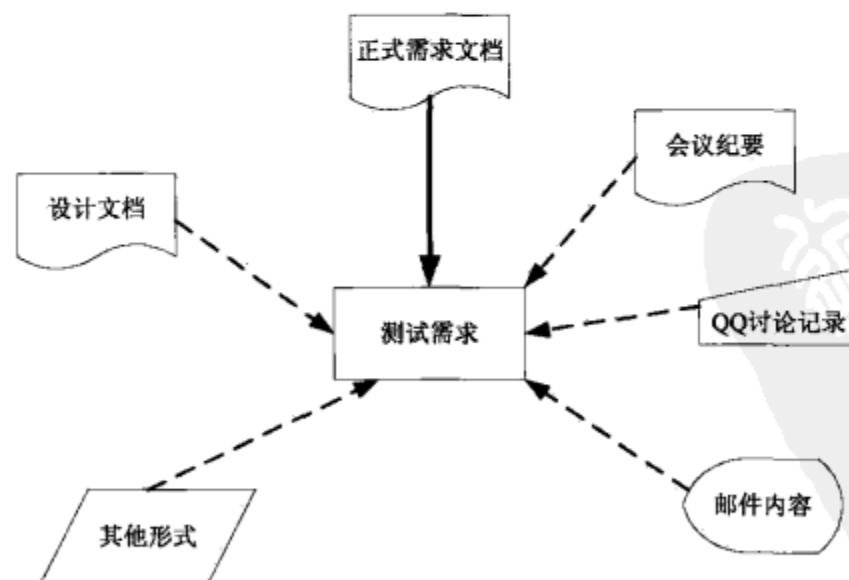


图 5-3 多管齐下的测试需求

注：图中正式需求的来源用粗线表示，非正式需求来源用虚线表示

质疑精神，是我们测试人员需要的气质，在提取测试需求、追溯需求的来源、实现背景过程中大可充分发挥，这对我们是否能真正理解用户场景需求很有用，见下面案例。有时需求分析人员未必能给你满意的回答，没关系，我们可以再往前溯，直接与市场，甚至是一线用服或用户直接交流，直到弄明白真相为止。这也是我们突破开发内部交流，把目光看得更远的机会。

【案例】MP4 的日历提醒事件丢失了

某公司生产 MP4 电子产品，其中的应用软件是自主研发的，日历行程便是其中一款应用程序。卓 A 是软件测试部负责此日历行程的测试工程师，在做日程提醒事件测试时，他发现如果 MP4 电力不足（不足以开机），而这段时间正好有提醒事件发生，则在下次开机后不会再提醒，即发生在没电池时段内的提醒事件会丢失。而对于这种特殊情况，需求并未有明确定义（需求只定义了到达约定的时间便进行响铃提醒，并弹出事件窗口）。于是卓 A 找到需求、开发讨论关于这种特殊情况的处理，开发认为，电力不足的情况下，本来就不可能开机，提醒事件也就不可能弹出来，目前的处理是合理的。需求设计人员认为，如果在不能开机的情况下，提醒事件需在下次开机后进行积累提醒，如果用户一个月没用此机，事务每天又多的话，再次开机后的消息太多了，光关闭这些事件窗口都不是一件易事，用户未必欢迎。最后这个问题就此搁浅（挂起）了。

无独有偶，当该产品上线约一年后，从市场返回一张更改申请单，上面反映“成都客户陈女士于某年某月某日购买的一款 M680 型号的高档 MP4，一天晚上，取出机子的电池在机外充电。后来由于忙于其他事情，3 天后才再次打开 MP4 来使用后发现，这 3 天内的提醒事件一个都没有，导致她把提前 3 天预订飞机票的事给忘了，误了她的一桩大事。后来打电话到某公司当地办事处进行投诉。

5.1.2 考虑可测试性需求

上节介绍的主要是用户角度业务性测试需求的提取，本节讨论站在测试角度本身可测试性需求的问题。可测试性需求需尽早发现，否则到了项目后期，就会陷入一种欲罢不能的状态。甚至对于一些需求，由于它很重要，但由于又不可测，犹如一把悬在空中的剑，想把它拿下来，但由于悬得太高，只能“望洋兴叹”。

可测试性（Testability），简单地说，是指软件可以被完全有效测试的程度。这一概念的解释最早在 1990 年的 IEEEstd.610.12 中给出。Freedman 在 1991 年提出把可测试性定义为可控制性（Domain-controllable）和可观察性（Domain-observable）的集合，随着时间的

推移这一概念不断丰富完善。目前软件可测试性要求主要包含可观察性（即可见性）、可控制性、可操作性、简单性和稳定性等。由于前二者更具代表性，下面主要介绍它们的特性与使用场景。

1. 可见性

指被测软件的状态、数据输出、资源利用和其他影响能被准确地测试到，以决定测试是否通过。

可见性，并不限制在 UI 界面中能看到数据或状态的输入与输出。一些软件由于工作本身服务于底层，如主机与客户端的通信、后台数据库的处理，这些数据的中间处理过程在 UI 层并不可见，此时就存在一种可测试性的问题。当然解决类似问题的方法很多，如增加日志记录就是一种常用且很好的方式。通过日志记录的内容，测试人员可以很清楚软件的中间处理过程。我们熟悉的 Windows 操作系统自带的日志记录便是一个例子（用鼠标右键单击“我的电脑”，在弹出的快捷菜单中选择“管理”命令，打开“系统工具”文件夹，选择“事件查看器”），如图 5-4 所示是日志记录的片段。

类型	日期	时间	来源	分类	事件	用户	计算机
信息	2010-5-8	8:13:07	Service Control M...	无	7035	SYSTEM	WWW-26663619634
警告	2010-5-8	8:13:01	Server	无	2506	N/A	WWW-26663619634
警告	2010-5-8	8:13:01	Server	无	2506	N/A	WWW-26663619634
信息	2010-5-8	8:12:42	eventlog	无	6005	N/A	WWW-26663619634
信息	2010-5-8	8:12:42	eventlog	无	6009	N/A	WWW-26663619634
信息	2010-5-7	23:47:59	eventlog	无	6006	N/A	WWW-26663619634
信息	2010-5-7	22:22:29	Service Control M...	无	7036	N/A	WWW-26663619634
信息	2010-5-7	22:22:29	Service Control M...	无	7035	SYSTEM	WWW-26663619634
...

2010-5-9 17:08:02 Service Control Manager信息360 杀毒实时防护服务 服务处于正在运行状态。
2010-5-9 17:08:02 Service Control Manager信息360 杀毒实时防护服务 服务成功发送一个开始控件。
2010-5-9 17:08:00 Service Control Manager信息360 杀毒实时防护服务 服务处于停止状态。
2010-5-9 14:58:19 Service Control Manager信息360 杀毒实时防护服务 服务处于正在运行状态。
2010-5-9 14:58:19 Service Control Manager信息360 杀毒实时防护服务 服务成功发送一个开始控件。
2010-5-9 14:58:16 Service Control Manager信息360 杀毒实时防护服务 服务处于停止状态。
2010-5-9 12:16:20 Service Control Manager 信息360 杀毒实时防护服务 服务处于正在运行状态。
2010-5-9 12:16:20 Service Control Manager 信息360 杀毒实时防护服务 服务成功发送一个开始控件。
2010-5-9 12:16:18 Service Control Manager 信息360 杀毒实时防护服务 服务处于停止状态。
2010-5-9 9:58:28 Service Control Manager 信息360 杀毒实时防护服务 服务处于正在运行状态。
2010-5-9 9:58:28 Service Control Manager 信息360 杀毒实时防护服务 服务成功发送一个开始控件。
2010-5-9 9:58:27 Service Control Manager 信息360 杀毒实时防护服务 服务处于停止状态。
2010-5-9 9:52:47 Service Control Manager 信息Application Layer Gateway Service 服务处于正在运行状态。

图 5-4 Windows 系统日志记录片段

图中的这些信息都是操作系统自动记录的，只要你设置好相关条件，系统就会按你的要求在后台默默无闻地记录下所有相关信息。通过它可以了解系统的“喜怒哀乐”和“一

言一行”，虽然都是一些流水账，但我们既可以从中品尝到成功的喜悦，也可以找到失败的原因，实在是一个忠实的系统助手。类似地，对我们所测试的软件，同样可以提出这方面的测试需求，把需要的数据、操作记录通过日志形式记录下来，需要时进行分析，这无论对测试人员还是开发人员无疑都是福音。

2. 可控性

指能向被测软件输入预期的数据，或修改它的状态，如一个应用程序有事件触发的阈值，能够设置和重新设置那些阈值可简化测试。

测试过程中，常会遇到一些测试环境难于模拟，但实际情况又有可能发生的场景，如运行在某特殊仪器上的监控软件，由于不同的环境温度对仪器的工作有影响，要求当仪器检测到正常工作范围外（超高温或超低温）的温度时，进行不同警戒的报警。而这种关于环境温度的用户场景，测试人员不好实际模拟，环境温度的变化也难于控制。存在一个可控性的测试需求，需要开发一个特殊测试工具与软件接口方可解决这个问题。

关于可控制性的测试需求，在我们实际工作中会经常遇到，如模拟成千上万用户同时访问某一网站，同时点击“登录”等。在相关的压力、性能的测试上更是如此。

下面是微软的一位测试架构工程师 David Catlett 讲述他们是如何测试上百个调制解调器的案例。

【案例】如何测试上百个调制解调器？

在测试 Microsoft Windows NT 远程访问服务器（RAS）时，我们需要利用有限的资源对调制解调器的拨号服务器进行可伸缩性测试。我们碰到了一个可测试性的问题，为了准确地模仿真实的用户部署，我们需要测试上百台调制解调器同时连接拨号服务器的情况，而现有的资金和实验室基础设施只能测试十几个调制解调器。测试团队想出了一个办法，用软件来模拟调制解调器，并将其与以太网相连，我们称之为 RASETHER。使用这个测试工具最终被证明是一个很好的办法。因为它是人们第一次在一个网络里创建另一个专有网络。如今，这个技术被称为虚拟专用网络或者 VPN。起初为了 Windows NT 调制解调器服务器的可伸缩性测试而设计的测试工具变成了一个巨大的商业成功，并且成了用户进入公司网络的重要工具。

3. 可操作性

指软件易操作、贴近用户。在软件上市前，进行用户体验测试是一个不错的做法。当然，软件可操作性高，被测试的效率也会更高。

4. 简单性

指提交测试的模块或组件和应用程序越简单，测试起来越容易（测试成本也更低）。

5. 稳定性

一般而言，测试的软件改动越小，质量就越稳定。但是，软件的稳定性，与需求变更的控制，开发周期，测试发现严重 Bug 的时间早与晚等都有关系。

在研发阶段，由于需求的变化、代码的变更，软件的可测试性在开发的整个过程都有可能发生，所以发现可测试性需求，时机是无限制的。但下面的几个早期阶段，显得特别重要，分别是产品需求设计、软件需求设计、设计需求评审阶段，即代码还没出来之前。测试人员要认识到可测试性，需要他们清楚了解软件设计，评审现有的设计文档和阅读代码。但常常，测试人员因为害怕他们的请求会被拒绝而不愿意提出可测试性需求。

软件的可测试性被提出以后，一方面它可以逐步成为软件度量的重要标准，成为衡量软件产品质量优劣的一个重要尺度；另一方面，软件的设计人员也可通过新的设计方法，逐步将这一标准应用于从软件分析开始的一系列软件过程，提高软件质量。不论是哪一方面，合理并有效的可测试性分析对软件的开发过程都起着重要的作用。而且这种分析在软件生产过程中，开始得越早，越能节省软件开发投入，并提高效能。

5.2 识别庐山真面目——分析需求

测试需求的获取，是测试工作迈开的第一步，这在上节做了介绍，接下来是在有了需求后，如何理解它、分析透它，成了问题的关键。也只有解决了这个问题，才能提取出具体的可操作的测试对象出来。

5.2.1 快速理解需求的捷径：需求宣讲

软件需求是软件项目开发的依据，代表着用户的需求，是软件设计及软件测试工作的入口，在整个软件项目开发过程中起着举足轻重的作用。对需求的理解是否到位，在很大程度上影响着开发过程的效率。曾经有个小项目（整个项目时间约两个月），需求不多，在进行需求评审时，包括开发及测试人员都认为理解了需求，但在后来版本测试中才发现，有一个重要需求点，开发人员与测试人员的理解完全不一样，到底谁的理解才是对的呢？双方找来需求讨论，恰恰需求设计人员对这一块的理解也讲不太清楚，写在文档中的描述就更不用说了。也就为此一点，开发整改设计及编码，测试整改测试方案

及用例，最后版本的发布推迟了两周。俗话说“吃一堑，长一智”，通过这样的项目事例，我们需反思整个开发过程做得不够的环节，就“开发过程中如何更好地理解软件需求”有以下几方面的最佳实践。

- 需求宣讲的组织：需求基线一旦形成后（即需求完成第一版后），启动内审，通知项目组相关人员事先预审，给出问题反馈的截止时间与内审时间。
- 介绍需求的背景：需求编写人宣讲需求，介绍需求的用户背景。这点很重要，让开发及测试人员能清楚知道用户的用途，实现后的软件是用来做什么的、能满足用户哪些要求、能给公司创造什么样的价值等，使开发及测试人员的工作目标很明确，处处从用户立场考虑。
- 需求宣讲内容：需求宣讲是否需把所有内容都讲一遍？回答是肯定的。正如前面所述，需求的至关重要性，如果需求内容多，可分开多次宣讲。当然讲解时，也要分开重难点，对于大家容易理解的需求（如之前项目做过或众所周知的需求）可以几句话带过。
- 辅助答疑式宣讲：在需求宣讲时，对项目组成员收集到的问题，一一进行解释，回答提问者疑问的同时，也分享给其他同事。

有一个事实我们不得不承认，需求不可能详细到面面俱到，总会存在着隐含需求，这种隐含需求的理解会体现出你对需求的掌握程度。对于这种情况怎么办呢？无论对于开发还是测试，如果自己意识到了，但不能确认，把问题记录下来，与需求确认，并要求需求补充明确。

需求的设计也不可能是完美无缺的，在测试活动的整个过程中，特别是在设计用例的过程中，测试人员会发现不少有需求定义的 Bug，此时把它也录入缺陷库，作为一个缺陷来跟踪管理是一个不错的方法。俗语说“好记性不如烂笔头”，事实证明，只在口头上说的事，需求人员容易忘改需求，使得最后测试用例与需求对应不上，且给后续加入项目人员的工作增添麻烦。后面常会发生“这个地方的需求与实现不同，是以需求还是以实现为准”的局面。

5.2.2 需求定义也会错并不是谎言

需求定义是否存在二义性，一直以来是衡量需求是否明确的一个标准。尽管需求设计人员采用了图、表等图文并茂的方式来表达，但面对有着不同背景的读者群，同样一段话，理解的结果就是不一样。特别是对于一些需要有一定背景知识的需求，不是理解不到位，

就是理解错误，这也就有了显式需求与隐式需求的叫法。显式需求指有明确定义的一系列约束软件实现的要求，可以是有给定具体值的数据字典，或有序的业务流程图，或一段介绍业务功能的需求描述等。隐式需求并不是需求设计人员特意隐藏，更多的是由理解人员对某方面专业知识，或对产品的业务了解程度有限导致的。例如：某软件需求中有一句这样的定义“对用户产生的数据，需每天定时备份（默认为每天晚上 12:00，用户可设置），并可按需生成书面报告，报告中的数据不存在法规风险”。那么，此句话中的“法规风险”具体是指什么呢？这正是需求背后的需求，也就是隐式需求。

隐式需求的识别要求较高，分析需求时，测试人员能否过滤到，直接影响着后续测试工作的有效性与全面性。另外，对显式需求中存在的错误是否能及时发现或意识到也同样重要。下面是一个需求定义中存在缺陷未及时发现发现的案例。

【案例】需求定义隐含缺陷

问题现象：软件版本国际化后，输入限制变小了？

问题背景：某软件有可输入用户信息功能，需求是这样定义的：姓名可输入 16 个中文，32 个英文，也即最多输入 32 字节字符。于是软件开发在实现此输入功能时，“姓名”字段的长度限制为 32 Bytes。但是有一天，软件需进行国际化实现，以支持法语、俄语、西班牙语等。开发人员于是在字符的编码格式上把原来的 GBK 换成了 UTF8。这样一来，对于每个字符的编码是不固定的（英文除外），测试人员发现原来中文可输入 16 个汉字，现在只能输入 10 个汉字了，于是提了一个 Bug 给开发。而开发人员认为是需求定义考虑不全面的问题，需求后来改为能输入 32 个字符，这个字符无论是什么符号（中、英、俄等语言）都一样。

问题分析：初看“姓名可输入 16 个中文，32 个英文，也即最多输入 32 字节字符”这句需求，是明确的，不存在二义性。当软件只支持中、英两种语言时，它也是没问题的。但是产品进入国际化市场是必然趋势，需求设计人员当初定义此条需求时，是否考虑了这一点呢？如果考虑到这一点，是否又能意识到目前定义的局限性？这让笔者领悟到“需求设计人员要有软件设计相关知识的重要性”。从开发人员角度分析，在编码格式转变后，表面上软件是被国际化了，但是否对原有功能造成了影响，并未无及时提出。从测试角度分析，在分析需求，同时也在对需求进行测试时（在本书的第 8 章需求测试部分有单独介绍），同样并关注到。

思考：类似这种需求定义不合理问题，测试人员该如何在早期发现，这正是需我们解决的问题。可能有不少测试朋友会觉得诧异，笔者也曾多次听一些开发或测试人员提到“怎么需求也会错？”需求定义错误的原因很多，如需求定义者并没有真正理解用户的需求，

文字表达上引起的歧义，需求写过头了，如把对设计的要求写进等。案例中需求定义“也即最多输入 32 字节字符”便是一个例子，“32 字节字符”是一种开发语言，而不是用户需求。另外，采用需求评审、需求宣讲、需求测试的方式对挖掘隐含需求都是有帮助的。

5.2.3 不可忽视：从设计需求中提取测试需求

软件需求是软件测试需求的主要来源，但不是全部来源，软件设计需求、软件概要设计、详细设计也都是测试需求的分析对象，是对测试需求的一种有力的补充。对于黑盒功能测试，几乎 98% 的需求都是来源于需求说明书，但有那么一小部分需求来自设计需求或概要设计、详细设计。也就那么小部分需求，如果我们没有意识到，就会给用户带来隐患。下面便是一个与设计需求相关的典型案例。

【案例】设计需求转换为测试需求失败

引言：软件开发环境，这一软件设计需求，对于产品需求来说，它只会说需要什么，不会关心这个东西用什么开发语言，什么开发环境去实现。但对于测试来说，需要掌握这些，因为测试的环境是测试必须考虑的要素之一，而测试环境与开发环境有关。

问题现象：待测软件在 Vista Home Bbasic 32 系统下安装失败。

问题背景：周五下午将近下班了，测试陈经理收到生产装机线上的信息，说昨天发布的软件，在 Vista Home Basic 32 下安装失败。

问题分析：测试人员在总结中提到“软件一直在各种宣称的支持平台上运行得很好，在不同的平台之间进行安装测试不下 100 次，仅是最后一个版本开发人员更改了安装程序，来不及在每个平台上铺开进行用户场景的测试，而问题却恰恰就出现在没测试的平台上”。或许，这种总结是我们日常工作中经常看到的。聪明的读者朋友，也许你已注意到了，这个仅是没有发现此 Bug 的表象，根本不是问题发生的原因。测试要及时发现此问题，难道每次对发布的版本都只能通过在各种操作系统环境之下重新再测试一遍的穷举方式来网罗未知的问题吗？

所测试的软件是用 C# (Dot NET) 开发的，而 Dot NET 开发出的软件运行与操作系统自身是否安装了 Dot NET Framework 有关，且还与版本有关。在进行安装测试时，测试人员并没有分析到这些影响因素。此 Bug 是因为软件在安装过程中发现当前操作系统自带了 Dot NET Framework 3.0 组件，且版本比发布软件包中自带的高。安装程序遇这种情况不会安装自带的 Dot NET Framework 2.0 SP1 补丁包，于是终止了安装过程，并提示安装失败。这里面包含着两方面的问题，一者，安装程序遇到这种情况，不应该失败

退出,应该继续往下安装;二者,操作系统 Vista Home Basic 自带的 Dot NET Framework 3.0 是否向下兼容。

为了搞清楚它们之间的兼容性,相关人员汇总了 Dot NET Framework 各版本之间兼容性以及与操作系统的集成关系,如表 5-1 所示。

表 5-1 Dot NET Framework 各版本之间兼容性以及与操作系统的集成关系汇总表

Dot NET Framework 版本	兼容性	与操作系统的集成
Dot NET Framework 1.0 Service Pack 1,2,3	支持向后兼容(注 ¹)	无
Dot NET Framework 1.1 Service Pack 1	支持向前兼容,也支持向后兼容(注 ²)	无
Dot NET Framework 2.0 Service Pack 1	支持向前兼容	无
Dot NET Framework 3.0 Service Pack 1	支持向前兼容 S	Vista Home Basic 带 Dot NET Framework 3.0
Dot NET Framework 3.5 Service Pack 1	支持向前兼容	

注¹: 向后兼容,指使用 Dot NET Framework 的较早版本创建的应用程序可以在更高版本上运行,例如使用 Dot NET Framework 1.0 版创建的应用程序可以在 1.1 版上运行。

注²: 向前兼容,指使用 Dot NET Framework 的更高版本创建的应用程序可以在较低版本上运行,例如使用 Dot NET Framework 1.1 版创建的应用程序可以在 1.0 版上运行。

上表这些测试环境相关信息,并不是一定出了问题之后才补起来,在分析设计需求、提取测试需求时就应该做,但测试人员在之前并没有意识到,成了测试的盲点。这也是笔者想讲述的核心,如何有效地把设计需求转换成测试需求。面对安装程序,测试人员是否理解了为什么要这样,而不那样?当不知道为什么要这样做,就意味着实现的背后仍可能隐藏着测试的盲区,就是那些没有触及的黑暗地方。例如,对于安装程序安装完成之后的预期输出,有不少测试朋友看到界面上提示“安装成功!”就以为大功告成。或许,这个提示骗骗用户还可以,对于专业的测试人员,除界面提示外,更重要的要把眼光放在软件安装过程中在操作系统上做了哪些事情,输出是否如预期。如果做到了这一点,就再也不会为“软件在未安装 Dot NET Framework 2.0 的计算机上安装 Dot NET Framework 3.0,同时自动安装 .NET Framework 2.0”而感到诧异。由于 Dot NET Framework 1.0、1.1、2.0 版,各版本之间是完全独立的,无论计算机上是否存在其他版本,这 3 个版本中的任何一个版本都可以存在于计算机上。基于这一点,当我们看到如图 5-5 所示的界面,也就不足为奇了。

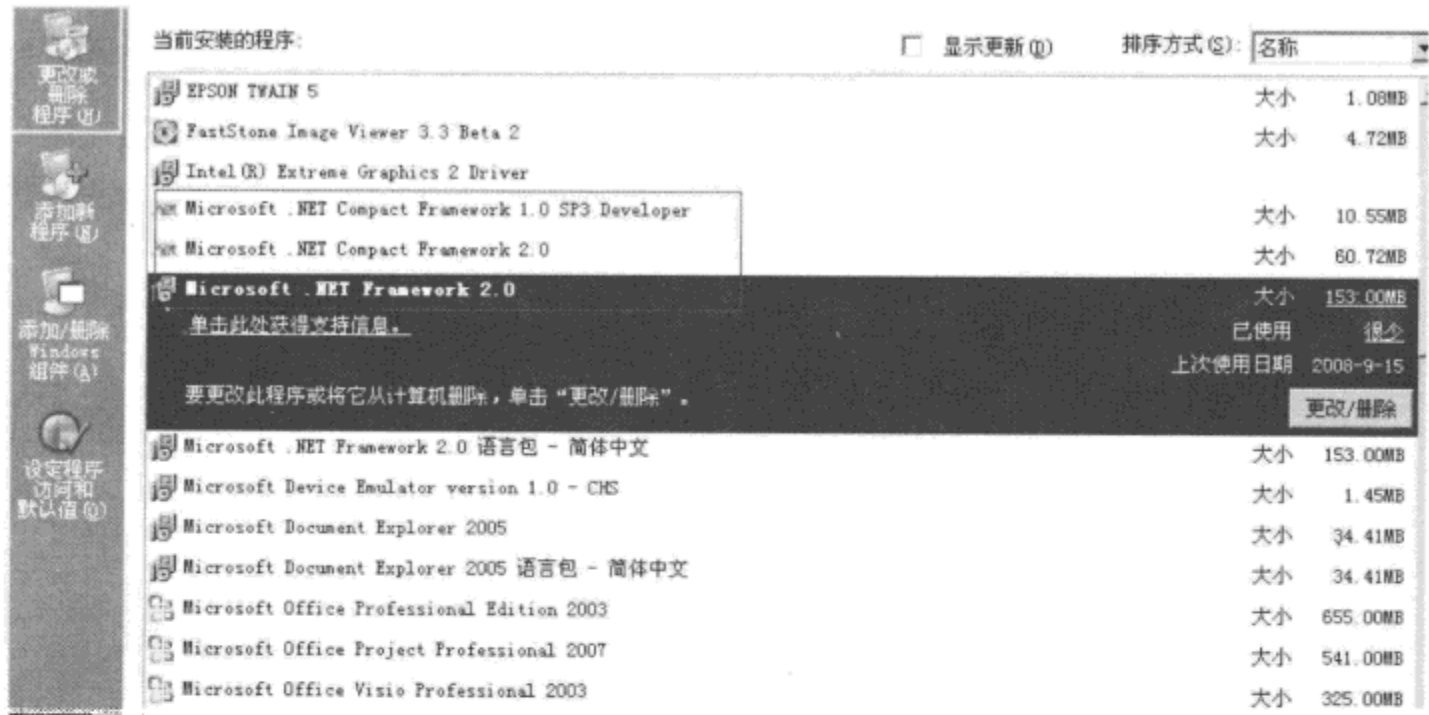


图 5-5 Dot NET Framework 多版本并存

对于基于 Dot Net Framework 开发的软件,其相关的专业知识,如工作原理(见图 5-6)、各版本之间的区别、兼容性等,对于测试人员来说,并不一定要精通,但要熟悉,方能占主动地位,把待测试软件验证充分。

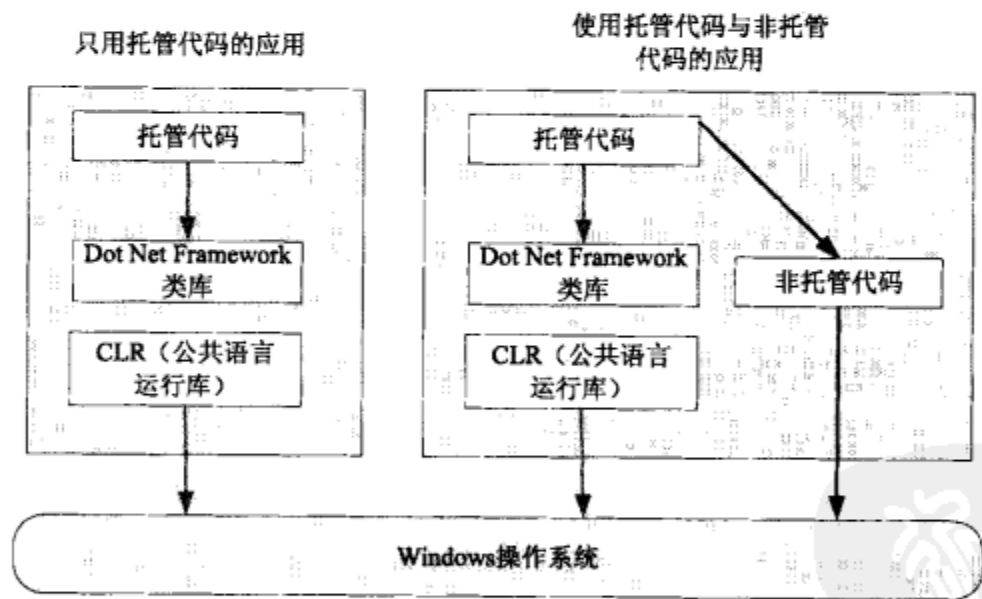


图 5-6 Dot Net Framework 应用软件工作原理图

在一些测试专业网上,总有不少测试新手在问“测试是否要懂编程?”,或许上面的案例就是一个很好的回答。

有一句话说得好,“不怕做不到,就怕想不到”,这其中的“想”,就是主动意识的体现。在工作中,类似需求背后的需求还很多,一些可能与产品业务知识相关,一些与技术背景有关。业务知识也好,技术背景也罢,要提高我们分析需求的能力,设计出高效的

测试用例，软件 Bug 的尽早发现，关键在于两个字——学习（放大一点理解，工作的过程也是学习的过程）。

5.3 确定顶层方向性测试类别

上节通过两个典型案例介绍了需求分析过程中遇到的问题，案例是笔者工作实践中曾发生过的实际场景，相信对读者会有启发。需求要如何分析，在网上可找到一些资料，如先准备好需求审核 Checklist（在 8.1 节需求测试中有介绍）是一个通用的方法，但是对需求的理解（或分析）是一个复杂的脑力思维过程，与分析人员的经验、技术等关系密切。需求分析的目的是为了提取测试对象，正如需求可分为不同的层级一样，测试对象也可从粗到细，逐步细分。

提取测试架构设计模型中的测试对象，需要确定顶层方向性的测试对象，也可理解为测试业务涉及的测试类别，以使测试人员能针对不同的测试类别考虑对应的测试策略。不同的测试类别，需要不同的测试方法，会涉及不同的测试资源，这在项目开始之初，需在测试计划中规划好。如图 5-7 所示是常见的一些测试类别，图中的比例仅作为示意，不同行业的软件可能会有不同的比例。

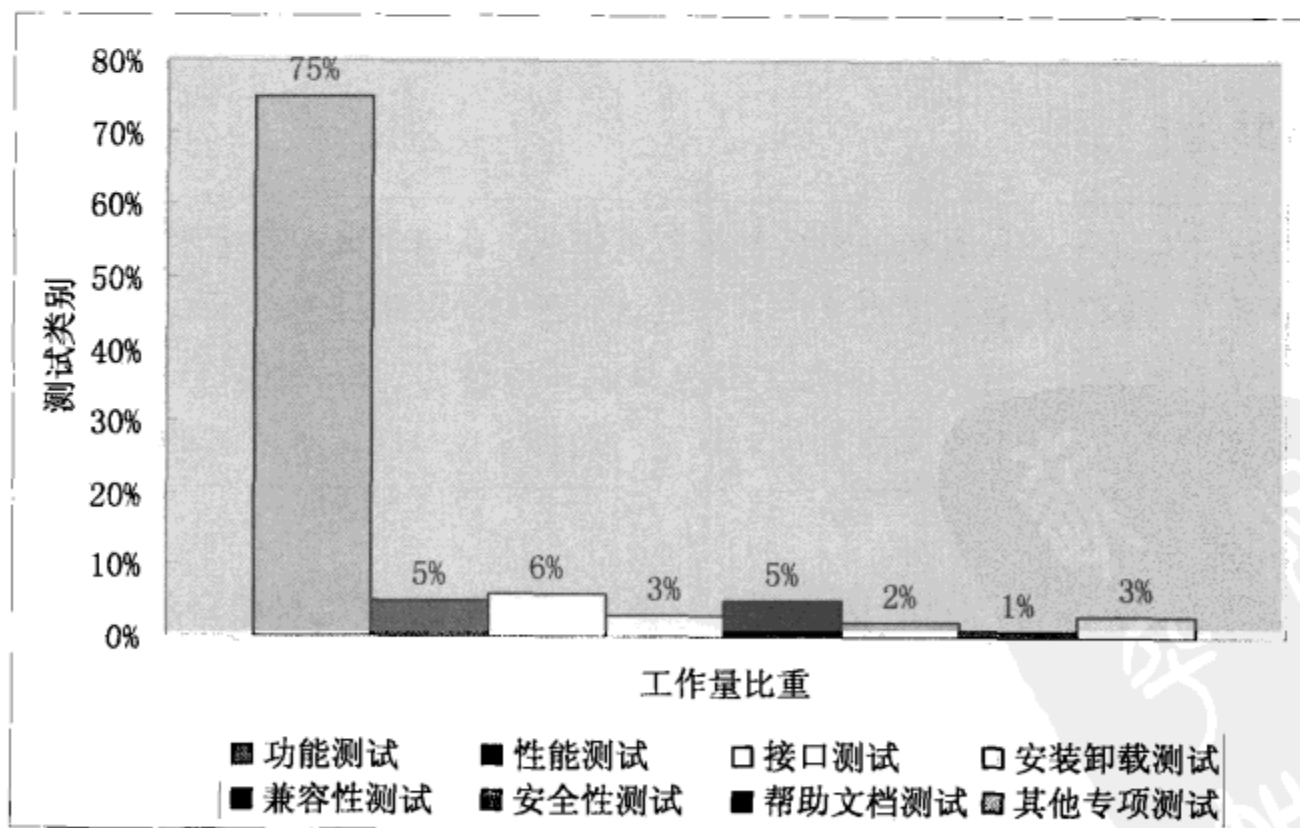


图 5-7 测试类别及其测试工作量比重示意图

1. 功能测试

业务功能是软件产品的基础，与用户最贴近的也是这些基本功能。功能测试恐怕也是测试任何产品的重点，没有功能的产品相当于不能用，也不叫产品。而功能是多种多样的，这一块的测试需求基本上会占 75% 以上。一个软件系统通常由若干个子系统组成，每个子系统又由若干模块组成。对于每一个子系统或功能模块，它们在系统中充当的角色及复杂度不同，在测试框架设计阶段，可只列出子系统及功能模块测试大项，接下来在进行测试方案的设计时，再对模块及下面的测试项（功能点）进行分析。

2. 性能测试

软件是否需考虑性能测试，测试什么？笔者认为这是任何软件产品都需要考虑的，只是被测软件不同，性能要求的关注点也会不同。如所测试软件是通过后台 SQL 2005 数据库来存储数据的，那么对数据库的访问速度，如查询速度是性能测试需考虑的，以及数据库的容量限置也需考虑。对于嵌入式软件，实时性要求较高，除了产品需求中明确定义的一些性能指标外，一些潜规则中与性能有关的特点，如软件界面的刷新速度、界面切换的速度等，都会影响用户的满意度，也可作为性能测试的关注点。这里的潜规则，指的是并未在需求中有明确定义，但它又是一些常识性的问题。例如，嵌入式产品中常有自带的 RTC 时钟，时钟的准确性就是一个潜规则的性能问题。笔者曾遇到过嵌入式产品上的时钟过了 3 天后，系统时间变慢了约 1 分钟，一周后，变慢了约 5 分钟的案例。

小贴士：

性能测试是一个较大的范畴，包括负载测试、压力测试和容量测试。其中负载测试是为了检验系统在给定负载下是否能达到预期的性能指标；压力测试是通过不断向被测系统施加“压力”，检验测试系统在压力情况下的性能表现；容量测试针对数据库而言，是在数据库中有较大数量的数据记录情况下对系统进行的测试。

3. 接口测试

这里的接口测试并不是软件内部模块与模块之间的接口（这种接口测试在功能测试中可考虑），这里指的是软件与硬件、机械等其他专业之间的接口测试。正由于接口测试跨专业，涉及的人与事多且杂，常需要我们主动收集或参与项目接口专业组的方案讨论，主动多请教他们，例如组织相关专业组对测试同事进行设计宣讲，针对某技术点进行培训也是不错的做法。多了解产品的实现原理，并记录相关信息，作为测试需求的提取源。甚至有时主动分析其他软件接口专业的需求变更，以了解其对测试的影响。例如设备的驱动便

是一种典型的测试对象，驱动程序是沟通软硬件之间协同工作的桥梁，驱动测试发现的问题有时不仅仅是软件的问题，还可能是硬件设计的问题。对于测试，在这方面的作用显得特别重要，如能发现这方面接口的问题，是对软件测试仅发现软件问题的常规要求的突破。发现的接口问题在项目经理眼里，你的贡献远远大于你发现一个纯软件的问题，因为它的影响远比纯软件的问题影响面广，对产品的质量意义更大。

小贴士：

驱动程序（Device Driver）全称为“设备驱动程序”，是一种可以使计算机和设备通信的特殊程序。它提供了硬件到操作系统的一个接口，协调了二者之间的关系，可以说相当于硬件的接口，操作系统只能通过这个接口，才能控制硬件设备的工作。人们都称“驱动程序是硬件的灵魂”、“硬件的主宰”，同时驱动程序也被形象的称为“硬件和系统之间的桥梁”。

4. 安全性测试

软件的安全性测试可从两方面考虑，一方面指待测试软件运行本身的安全，如待测软件受到病毒等一些恶意软件的攻击，文件被修改和数据被破坏等安全方面的测试。特别是一些运行于 Windows 平台的软件，感染机会与嵌入式软件相比，更加广泛。另外一方面，指待测软件由于行业性质的原因，如涉及生命、财产安全、个人隐私信息保护、商业机密等方面安全性的测试。这方面稍有不慎，很可能涉及法律、法规问题，是安全性测试的重中之重。

5. 安装升级卸载测试

对于应用软件的安装卸载测试，大家接触较多的可能是 Windows 平台上应用程序的安装与卸载，这是站在终端用户角度的简单使用场景。在单独为此项考虑测试对象时，需提取出涉及安装或升级的所有测试对象，如可能不仅仅是应用软件需安装或升级，例如嵌入式产品软件系统，还包括引导软件和操作系统，还常涉及其他硬件的写片软件，它们是如何安装、升级、卸载的呢？这些业务内容属于产品的服务支持类，但它们一样重要，如果在哪个环节出现问题，导致软件用不起来，招致客户抱怨或投诉，将严重影响用户满意度。安装升级卸载的测试，与其他类别不太一样的特点在于它的用户可能不仅仅是终端用户，如手机等消费性电子产品及医疗设备等产品软件。在生产制造、用服维护环节也涉及软件的安装、升级、卸载工作，其使用场景可能与终端用户不同，需做特别考虑。

6. 兼容性测试

同样，对兼容性测试对象的提取，抓住方向性的内容，也可从两个大的方向出发。一则是宣称兼容平台（操作系统）的测试，这主要是指运行于 Windows 平台的软件，如在网络上下载某软件时都可见到类似这样的宣称“运行环境：Windows XP/Vista”。对于嵌入式产品软件，由于操作系统一般情况下已绑定在软件中（厂家提供，已固定），与自己的应用软件是配套使用的，罕见可以兼容其他操作系统的宣称。另一个方向是软件升级前后，数据的兼容性，其中包括软件的向后兼容和向前兼容。向后兼容，即软件升级后，低版本软件或数据可在高版本中正常使用，这是常规要求，也是正向升级；向前兼容，即高版本软件或数据能在低版本软件中正常使用，相当于可以回退版本，是一种逆向升级的做法，实际使用场景也有可能存在，但相对较少。

7. 帮助文档测试

常见的有在线帮助测试与帮助文档测试，后者也叫帮助手册测试。这方面从技术难度来说会比较容易，工作量也相对较少。

8. 其他专项测试

根据项目的实际情况，除上面所列的这些大项外，还可考虑其他专项测试，如对于核心模块的内存泄漏测试、用户常用模块的用户体验测试等。

5.4 布道——部署测试策略

对于一个测试项目，在上节确定了方向性的测试类别后，这些测试对象该用到哪些测试方法，这些方法之间又该如何取舍，这便是测试策略部署的问题，本节接下来进行介绍。

提起布道，大脑中常会涌现出两种热闹且充满生机的场景。一种是有一对新人，新娘轻拽着穿在身上的一袭拖地婚纱长裙，新郎西装笔挺，轻挽新娘手臂，款款地走在鲜花簇拥下的鲜红地毯上，一直走到礼堂前，完成他们的婚礼宣言。另一种是一家新成立公司的剪彩仪式，企业家们站在摆满鲜花，同样也是鲜红的地毯上，从容、自信地拿起剪刀“咔嚓”一声，鲜艳的彩布被剪开，噼里啪啦爆竹声响起，公司宣布正式开业！前者是生活场景，后者是一种工作场景，它们都在宣布着一种全新道路的开始。软测中的布道，即测试策略的部署，想来亦有异曲同工之处，如图 5-8 所示为它们之间本质上是存在相通性的思想导图。

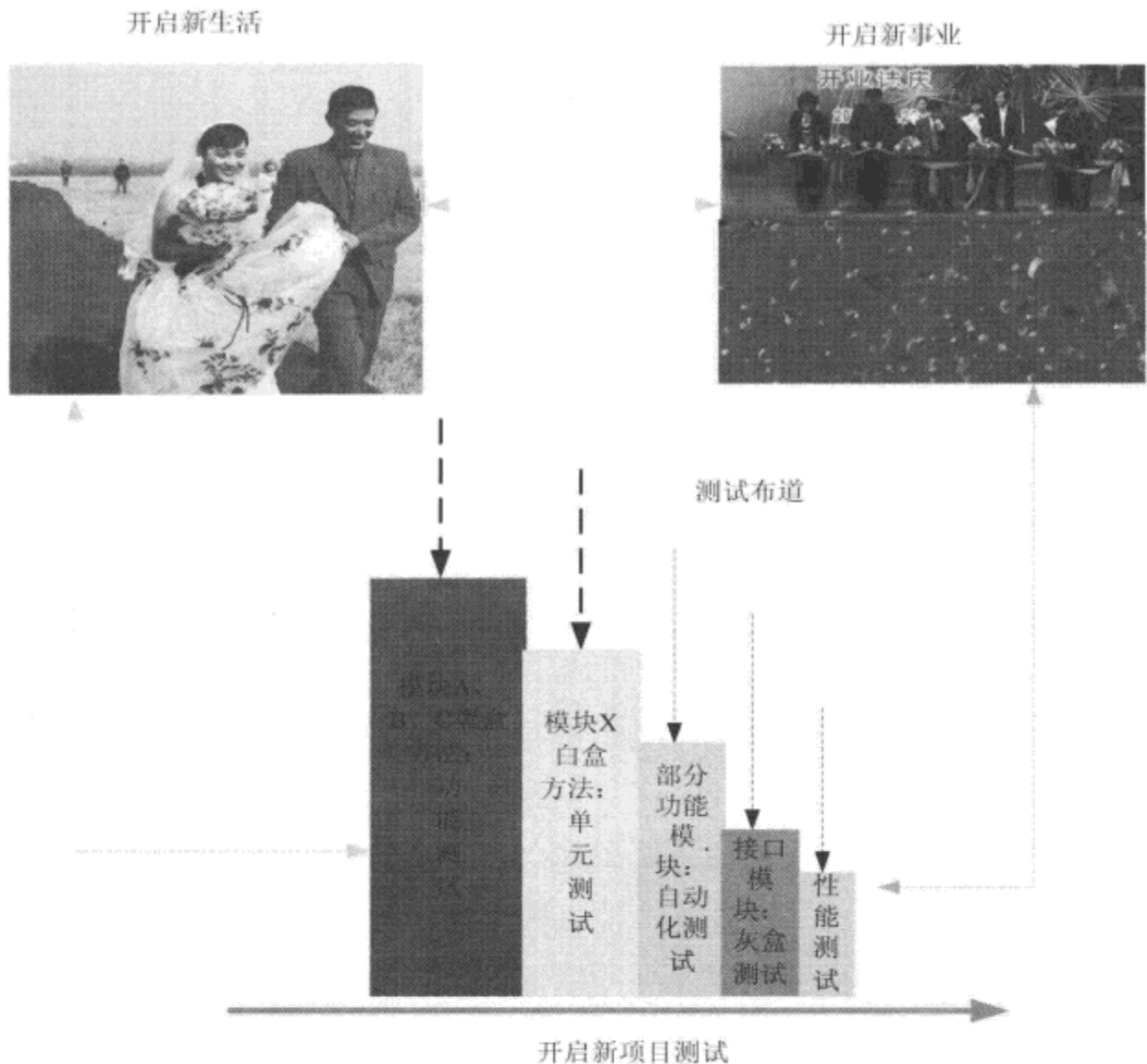


图 5-8 测试布道与工作生活场景本质相通性思想导图

测试策略的部署，可以这样理解，指完成一个测试项目需要的测试技术与方法、测试过程的管理与控制、一个测试团队的安排与培养等。测试架构师（有很多公司可能没有这个岗位，但会有这个角色，尽管这个角色是由某个测试经理或主管或工程师承担着）根据设计工程师的分析，决策项目使用的测试技术。设计工程师根据对测试需求的分析，考虑纯技术的应用。但架构师需结合项目的商业目标，如项目成本、测试人力成本、所需设备或工具的花费等，再决策采用哪些测试技术，以及开展哪些测试活动，这些测试活动如何跟踪控制等。这一点正说明了第 3 章所提出的“测试设计是一种有目的有计划商业活动”。

尽管在不同的行业不同的公司，采用的开发技术、管理模式千差万别，但就软件测试的工程思想，测试策略部署的方向性方面是存在共性的，在接下来的小节中分别进行介绍。

5.5 测试技术的裁剪与合理应用

关于测试技术，测试朋友们谈得最多的莫过于黑盒测试、灰盒测试、白盒测试了，再有就是自动化测试及一些专项测试，如性能测试、安全测试、内存泄漏测试等。在项目的初期，对测试需求的分解往往处于粗线条阶段，但就测试工作开展的技术应用上需尽早决策，以便安排资源尽早开展测试。接下来介绍常用到的测试技术及如何裁剪应用。

5.5.1 黑盒测试不等于手工测试

一般而言，任何一个测试项目，都不会不用黑盒测试，其中的原因有技术原因也有非技术原因，包括以下几个方面：

第一，黑盒测试是站在用户使用场景中进行的测试，在贴近用户使用的同时，能验证实现是否满足需求的定义。

第二，入门容易，对测试人员技术要求起点并不高，在研发的成本与质量及市场供求关系上能较快找到一个平衡点，这也是为什么很多创业初期的公司只要求进行黑盒功能测试的原因。但是黑盒测试虽然入门容易（门槛低），但并不意味着黑盒测试没有技术含量。目前业界很多测试朋友，特别是一些新人，存在这种认识误区。黑盒测试并不等同于手工测试，同样可用写测试代码，或脚本运行程序，或监控程序的运行，或获取程序的后台数据等方式来完成某项测试任务。笔者认为，要成为一个黑盒测试专家，技术上对广度的要求比白盒测试专家（主要在代码分析的深度上）要更高。

第三，在某些情况下，手工黑盒测试效率更高，可以很快地发现 Bug，如当软件不稳定时，以及测试一些特性方面时，如 UI 布局的效果、易用性测试等。还有一些情况只能用黑盒测试，如用户体验测试。美国软件测试大师 James Bach 在他的《软件测试经验与教训》一书中曾提到“手工黑盒测试可以发现 85% 的软件缺陷”。根据笔者多年的实践证明，事实确实如此，所以在部署测试策略时，黑盒测试是首先要考虑的测试方法，且投入比重方面比其他方面的投入要大，这里主要指人力的投入。这也就是为什么很多公司黑盒测试人员会占 70%~80%，或者更多。

以上观之，黑盒测试有其显而易见的优点，但在项目的实际测试过程中，如果只用单纯的黑盒测试方法经常会导致过度测试部分业务功能，而另一部分却测试不足，甚至于一直存在某部分测试盲区。因此适当采用白盒测试，作为补充，是制定测试策略中需考虑的事。如何适当采用，下节将进行介绍。

小贴士:

黑盒测试: 又叫功能测试, 把程序看成一个黑盒子, 完全不考虑程序的内部结构和处理过程, 根据规格说明书, 通过操作软件验证程序的功能是否与规格说明书规定的一致。

白盒测试: 也称结构性测试, 是基于代码的测试, 按照程序内部的逻辑结构, 检测程序是否能按预定要求进行正确的工作。

5.5.2 适当采用白盒测试

是否采用白盒测试, 从软件工程角度出发, 由于白盒测试能看到程序的内部逻辑结构, 可以在单元测试阶段就开始测试, 在修复 Bug 的成本方面占极大的优势。且对某一类的 Bug, 如果用黑盒测试方法, 需花很多时间或精力去准备测试数据或测试环境, 而用白盒测试方法却很容易做到。白盒测试不只限于单元测试, 代码走查、代码的静态分析、动态分析都属于白盒测试的范畴。下面是一个采用代码走查发现的对于黑盒测试来说很难测试到位的边界值问题。

【案例】一个边界值测试的问题

背景描述: 某精密仪器具有测量环境温度的功能, 当环境温度在 5℃ 到 35℃ 时它能正常工作, 当温度超过 30℃ 后, 测量结果的准确性将受影响, 仪器要求进入报警测量状态。

测试分析: 此功能黑盒测试工程师在执行测试时, 用一杯热水来加热温度传感器, 使其温度上升到边界值 35℃, 但由于精度的处理问题, 传感器感应到的温度通过硬件指令发送给软件后存在一定的误差, 软件在界面上显示的 30℃, 可能已超过真实的 30℃, 或实际上不足 30℃。还有像 30.1℃、29.9℃, 这样的边界数据, 用这种方法基本模拟不出来。在这种情况下, 必须考虑其他方法, 如进行软件插桩或硬件插桩 (指令仿真), 这两者的工程成本都不小。最后决定采用代码走查的方法审核代码。

测试结果: 结果很快发现在边界值定义范围的边界上, 开发人员把闭区间的表示, 写成开区间的表示了, 代码片段示例如下:

```
if( ( EnvrTemp > 5 ) && (EnvrTemp < 30 ) )
{
    //软件正常工作
}
Else
{
```

```
//软件进入报警测量状态
}
```

依据需求，第一行代码正确的表达应是 `if(EnvrTemp >= 5) && (EnvrTemp <= 30)`。

小结：对于黑盒测试难于验证，或重要的逻辑判断处理，核心模块的实现可采用白盒测试手段。方法很多，也可结合工具代替人工分析，如 `pc-lint` 代码静态分析工具就是一个不错的选择。

在项目测试中，决策白盒测试要不要开展很容易回答为“要”，因为它有显而易见的好处，上例便是一个很好的例证。但要如何开展，对代码的覆盖率做到多高，不是拍拍脑袋就能拍出来的。众多因素中，人力的投入与产出应该是最重要也最需要考虑的问题，如项目的代码量会有多少，全部进行单元测试需多少人力？单元测试由开发人员进行的话，对进度的影响可以接受吗？如果测试人员来做，代价会不会太大，目前的资源胜任度如何？如果只对核心模块或核心代码进行白盒测试，质量上可以满足项目需求吗？就白盒测试展开的“度”，包括广度与深度，也是策略决策者必须谨慎考虑的事情，有时甚至就是一个事倍功半或是事半功倍的决策。笔者曾见过的一个失败案例，见下面所述。

【案例】一个代码测试的失败案例

一个 20.4 万代码行的软件，安排 2 个测试工程师进行代码测试，介入时间在开发人员实现代码的中期。2 个测试人员共耗时 13 个月，对重要模块进行了单元测试、集成测试及代码走查。项目结束时，共提交 131 个缺陷，被开发人员取消的问题有 83 个，确认真正被开发人员解决的仅有 48 个。由于人力产出比与预期相差很远，对项目的实质性贡献不明显，最后只好取消这个代码测试组。导致这个结果的原因很多，其中最主要的是测试人员进行代码测试的条件不够成熟，包括代码的注释低（统计才 8.4%，业界公认代码注释率达到整个代码行的 30% 左右是比较理想的）、测试人员介入晚、人员少、自身能力有限等。

与黑盒测试技术相比，白盒测试要求门槛高，且弄不好就会吃力不讨好。但无可否认，白盒测试技术，是一把“好钢”，只是好钢要用在刀刃上，方能发挥它应有的价值。

5.5.3 活用灰盒测试

前面小节分别介绍了黑盒测试与白盒测试技术的应用，两者各有其优缺点。就好像矛与盾的关系，在不同的场合下，都有各自的用武之地。那么是否有一种介于黑盒测试与白盒测试之间的测试技术呢？回答是肯定的，那就是灰盒测试。

小贴士:

灰盒测试: 灰盒测试是介于白盒测试和黑盒测试之间的一种测试方法, 或者说两者的结合, 也有人称集成测试为灰盒测试。它关注输出对于输入的正确性, 同时也关注内部表现, 但这种关注不像白盒测试那样详细、完整, 只是通过一些表征性的现象、事件、标志来判断内部的运行状态, 或者说参考部分代码去做黑盒测试。

灰盒测试, 正如其定义中陈述的, 是介于黑盒测试与白盒测试之间的一种测试方法。由于这种测试方法在业界并没有像黑盒测试和白盒测试那样使用普遍, 在讲述这种方法在项目测试中如何决策之前, 我们先来看看这种方法的具体含义, 以及在项目中如何操作、能给我们带来哪些好处、不足之处又是什么。如图 5-9 所示是灰盒测试空间示意图。

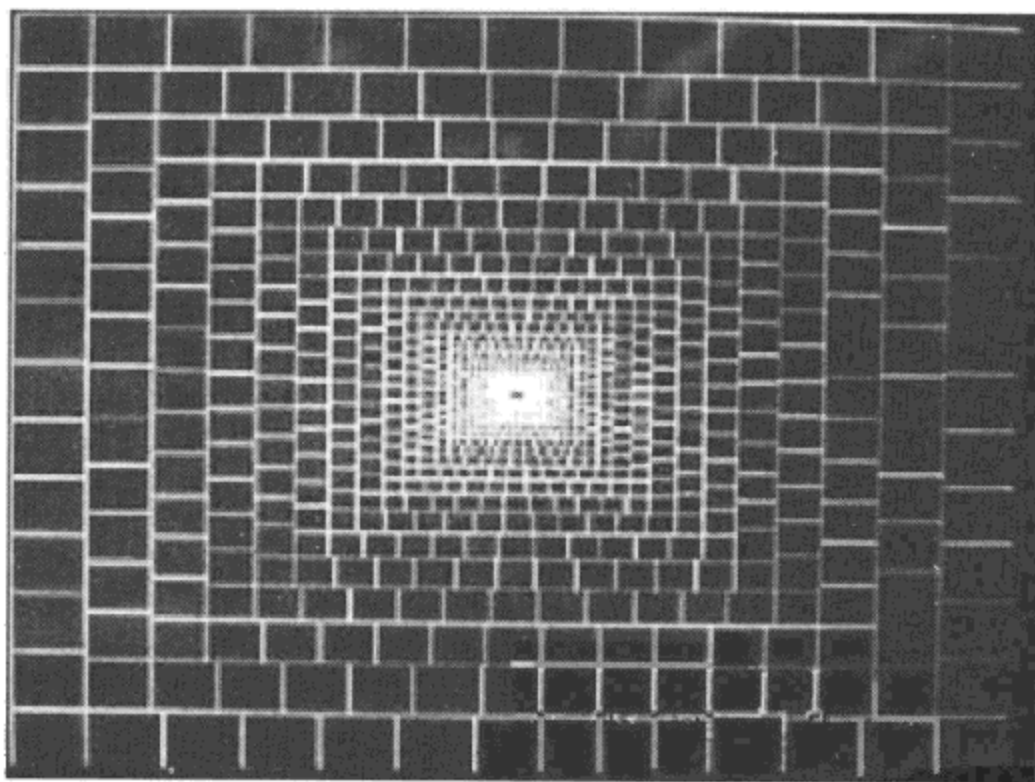


图 5-9 灰盒测试空间示意图

上图, 我们可以形象地理解为它就是一个方块盒子, 对于黑盒测试, 我们看不到盒子内部, 所以全都是黑色的。但对于灰盒测试, 它仍有黑色方块存在, 但已出现一层层白线围起来的方块, 表示看到了部分代码的实现, 白线越密意味着关注的代码越多, 最深处几乎为全白了。

灰盒测试结合了白盒测试与黑盒测试的要素, 它考虑了用户端、特定的系统知识和操作环境, 在系统组件的协同性环境中评价应用软件的设计。具体说来, 灰盒测试是在了解代码实现的基础上, 通过黑盒功能测试加以判断, 以验证软件实现的正确性。在理解概念

的基础上，开展实际工作，依据笔者的最佳实践，在以下几方面能较好地体现此方法的优点：

- 能有效地发现黑盒测试盲点。通过了解代码的内部实现，补充功能测试用例。这需要灰盒测试人员在查看代码之前，掌握需求，并清楚已有的功能测试用例。对某一功能点的实现进行代码分析（白盒测试），然后与黑盒测试（功能测试）充分结合起来，相互弥补，这就是灰盒测试的精髓。
- 可以避免过度测试，精简冗余用例。例如具有相同特点的某一功能，在进行功能测试时，是否每个界面或提示框、对话框都需进行该功能的测试。在没有采用灰盒测试之前，我们确实这样穷极测试过，虽说不上没有效果，但收效甚微。下面案例便是一个典型的例子。
- 能及时发现没有来源的更改。特别是在产品的维护阶段，每一行代码的更改都必须要有更改来源，但实际开发过程中，并不是每个开发人员都能做到，也并不是每个人都能清楚意识到更改后没有得到有效验证所带来的后果。像开发人员好心办坏事的例子已是屡见不鲜，所以测试人员采取事后代码审查的方法也是不得已而为之的事。

【案例】PDA 闹钟事件的测试

背景描述：闹钟是 PDA (Personal Digital Assistant) 上的一个应用程序，它的主要功能就是在用户设置的时间点到达时，进行响铃提醒。响铃时，无论用户当前在做什么操作，会弹出一个响铃提示界面。

提出问题：一天，负责测试此模块的工程师小叶提出一个问题“闹铃事件优先级高，在所有应用程序界面、对话框或提示框上面都会弹出，黑盒测试需要进入所有情况的用户界面，然后等待闹钟事件的发生，才能全面验证到各种情况的用户使用场景。”但这样穷举测试，需耗的时间太多，效果也并不一定好。

分析问题：关于闹钟的应用，实际上该模块只提供了一个接口。如能从代码角度分析到所有其他模块调用的是同一个闹铃接口函数（对于响铃及停止响铃后对原界面的恢复，都是统一一个接口处理的），那么再通过黑盒功能测试，在某一个用户界面进行此响铃功能的验证，即可证明代码的实现是符合需求的，这正是灰盒测试的方法。

解决问题：后来，对于闹铃功能点的系统验证，采用了灰盒测试的方法，做了代码正向、逆向分析，结合功能性用户场景进行验证，节省了近 2/3 的测试时间。

5.5.4 部分自动化测试

自动化测试一直以来都是业界讨论的热门话题，目前也有不少成熟的自动化测试工具，如 LoadRunnert, QTP 等。每谈起自动化测试，笔者与很多热衷于自动化测试的朋友一样，很兴奋，因为它确实能给我们带来很多好处，以及快乐的美好时光。例如无人值守的夜间测试，第二天一上班准能收到汇报测试结果的邮件；在办公室里你大可以优哉游哉地喝着咖啡，而它在听从你的指令拼命地跑，当你喝完一杯咖啡回到座位上后，测试结果已呈现在你眼前，可以尽享高效工作带来的愉悦。

自动化测试在以下几方面有突出的优势。

- 回归测试：在版本稳定的前提下，当每一次软件版本变更后，由于担忧更改影响原来功能，需要再执行原来成千上万条或更多用例，工作量可想而知。而如果此时能把这些用例自动执行，无疑可节省测试人员大量的执行时间。一旦测试脚本及测试环境搭建好，每次做很小的改动或不改动即可自动运行程序完成回归测试。
- 每日构建：在每日构建中加入自动化测试是一个很不错的做法。开发人员提交到版本库上的代码必须保证能顺利通过，而且要通过基本功能的自动化测试，以保证提交到测试的版本合格。
- 压力测试：可以代替手工执行困难或不可能做的测试。例如做一个反复点击某按键的压力测试，人工执行 100 次、200 次可以接受，但要连续点击 1000、2000 次可能就坚持不住了，如果是 10000 次或更多次则基本不可能完成。而这正是自动化测试的强项，再多它也不嫌累，速度上也一成不变。
- 一致性测试：执行相同的用例，不同的测试执行人员，常会有不同的结果，并不是用例有问题，预期结果也是正确的。主要原因是因为不同的人，对用例的理解不同，有些人执行过程中会发现其他的发散路径，做试探测试，而有些人可能没有。思维是不一样的，故同一条用例，有些人能发现 Bug，有些人则不能。而自动化测试则非常乖巧，对同一条用例，每次执行的结果绝对是一致的、可重复的，是主人最忠实的奴仆。

测试执行能自动化固然好，但不要盲目自动化，自动化测试并非万能。然而业界的很多测试朋友对自动化测试的认识存在一定的误区，认为自动化测试技术含量高，只有从事自动化测试才有发展前途。这些年，笔者在参与社会招聘时，收到的简历中几乎没有不提及自己熟悉某某自动化测试工具的。然而对所掌握的工具如何应用在工作中，给项目带来

了哪些实际的贡献等，却没几人能说得清楚，大部分是在学习、尝试中。在面试过程中还发现有不少人认为如果不提自动化测试，不掌握某项流行的工具，就好像差人一截，失去竞争优势。然而，自动化测试工具毕竟是由人设计出来的，工具本身并没有人所具有的逻辑思维能力，不具有人的智慧与创新能力。

在一个测试项目中，哪些可以进行自动化测试，哪些可暂不考虑，哪些根本不值得考虑等，需在项目规划初期，纳入测试策略的部署中。以下这些情况不适合进行自动化测试。

- 软件不稳定。如项目在研发的前期，版本频繁变更。如果此时就上自动化，那么随软件代码的修改，测试程序的修改也是相当巨大的。当然，这并不意味着在目前期自动化测试不用介入，相反可以做好测试方案、测试数据及用例的准备等。
- 用户很少用的功能。例如，某功能仅是为了产品的销售亮点而捆绑的一些锦上添花的功能，用户很少用或基本不用，这种情况不值得自动化。
- 用户体验测试。如易用性测试、UI 界面检查、与人的主观感觉直接相关的测试不适合自动化。
- 涉及物理交互的测试。例如，通信过程中、网线没插好、设备工作过程中开/关电源等。

以上观之，自动化测试所完成的功能是有限的，并非所有的手工测试都应该自动化。只有需要反反复复执行的用例，才需要将它测试自动化。

小贴士：

美国一位测试专家 James Bach 称：根据经验自动测试只能发现 15% 的缺陷，而手工测试可以发现 85% 的缺陷。

5.5.5 着眼专项测试

把测试方法分为黑盒、灰盒、白盒，主要是根据软件代码的能见度进行划分，而项目的测试过程是一个复杂的过程。除了会用到不同的方法外，还会涉及软件的一些特性测试，对于这些特性，需“专事专办”，走“绿色通道”，效率方可凸显。这些特性与软件的应用领域或市场需求有关，不同的软件会需要不同的特性测试，如内存泄漏测试、软件性能测试、数据兼容性测试等。下面以最让软件开发与测试人员头痛而又常被忽略的内存测试为例进行介绍。

内存泄漏（Memory Leak），特别是那种潜伏在代码深处的内存泄漏问题，无论对开发

人员还是测试人员都是最为痛心疾首，解决起来或重现起来最为棘手的问题。有一些软件对内存使用的要求特别高，例如嵌入式软件，由于设备本身的内存资源有限，对内存分配与释放的及时性需做出快速响应。否则即使是小块的内存碎片，也可能在“日积月累”后的某一天时间里，设备突然不工作了，或工作得犹如蜗牛爬行，甚至崩溃。内存问题都可归为严重问题（即使有些代码看上去只是使用了几个字节的内存块没有释放而已）。正如“内存问题无小事，一旦发作就要命”。下面是一个内存慢泄漏案例。

【案例】原来是内存慢泄漏

某市某三甲医院的检验科设备一天到晚总是那么忙，不仅医生忙，连检验设备也很忙。据某医生说，检验室的一台微生物精密检测仪已连续工作有一个多月了。好像有人性一样，这些天感觉测量速度有点慢，而今天甚至干脆就停下来了。软件界面点击无反应，好像是死机了，正等着维护人员来维修。

问题很快反映到了仪器的软件研发部门。尽管维护人员已收集了一些现场信息，包括软件停止界面图片、当前测量数据等，开发与测试人员在接到反馈后也马上组成解决小组，分析、调试软件，重现问题等，但历经近一周时间的努力，仍无结果。不过开发人员在分析过程中提到几个疑点，其中之一就是内存泄漏问题，但奇怪的是，在挂上了内存监测工具后，连续测量无发现有内存泄漏的丝毫痕迹。后来为证实是否存在内存慢泄漏的问题，解决小组采取了压力测试的方案。加快测量的时间，让在医院里工作一个多月的软件测量次数增加，最后发现当测量次数达到 1000 次后，发现存在 1KB 左右的内存消耗增长，相当于医院做两天的样本测量。最后证实，问题的确实是内存的慢泄漏引起的。由于该医院使用仪器长期没关过机，最终把此问题暴露了出来，而在每天都进行关机的医院中，此问题始终没有发生过。

上面案例中关于内存泄漏的问题，可见其影响的严重性及在后期解决的代价。有了前车之鉴，这样的问题是否可以在研发前期就发现呢，回答当然是肯定的。不过，据笔者调查，有不少公司常会出现这样一种现象，当测试负责人提出进行专人专项的内存测试时，常会遇到一大部分人的反对，理由是测试人员从来没有提过这方面的 Bug，产品卖了几年，也没收到用户在这方面有不良的反馈，软件在内存使用方面应该还是可信的。这是典型的侥幸心理，究其原因还是公司领导层对软件质量的重视不足，这也正是为什么国产软件的质量还远远落后于国外的主要原因。

内存测试比之于软件的其他特性测试，更显特殊一些。因为它与软件的类型或软件所用的开发语言无关，软件在运行期间都需用到内存。当然不同的开发语言对内存的申请、分配及释放的处理细节会有不同，但原理却是相同的。内存测试需考虑以下几方面：

- 测试软件是否存在内存越界访问、内存泄漏。内存越界主要包括数据越界读写、“野指针操作”和堆栈溢出等几种。内存泄漏则包含较多的种类，如分配的内存没有释放、打开的文件没有关闭、使用的窗口句柄没有关闭等。
- 验证软件在极端情况下的行为。如程序长时间运行后的情况，内存慢泄漏的案例便是一个典型。还有当内存资源被别的应用程序消耗很多时目标软件的运行情况如何。
- 待测软件对系统资源的占用率，防止软件使用的资源超出系统的限制。比如软件中使用的句柄数是否超出了系统的限制。

要对以上几方面的内存使用情况进行验证，我们可以使用对代码进行静态检查、动态检查及专门的内存监测工具等手段。静态检查代码可采用代码走查或借用代码静态工具，如 PC-Lint 就是一款不错的工具；也可以对代码进行动态检查，如借助内存动态检查工具，BoundChecker、Purify 等在软件运行时进行检查。还有就是通过自己编写测试代码，辅助内存监测工具来执行。

本小节介绍的专项测试，主要是基于软件的不同特性而言，把这些特性独立出来，以得到足够的重视，作为项目测试的一个子项。并且在产品的维护阶段一直纳入到每次版本更新的影响分析范畴，进行全程跟踪，为软件的质量可靠性验证与评估作出贡献。

5.6 测试计划与跟踪机制

一个项目的测试是一个持续性的过程，小的项目少则几天，大一些的通常会持续几个月，时间长的大项目也有几年的情况。项目开始之初，常看到不确定的需求太多，已确定的需求又觉得它太粗放，可测试性不好，让人无从下手，所有这些都是很正常的事情。本节介绍如何把项目的终极目标进行细分，定义过程测试里程碑，并进行跟踪与控制，以最终达到项目测试任务的胜利完成。

提到测试阶段，自然地想到软件开发的模型，因为在一般情况下，采用什么样的开发模型，会有对应的测试模型。例如，如果项目采用的是增量式（迭代式）的开发模型，测试也用增量式的测试模型划分不同的测试阶段才比较合适，如表 5-2 所示。一个完整的项目，站在项目经理的角度，需要输出一个项目的综合开发计划。其中，软件只是其中之一的子系统，可能还包括硬件、机械等产品组成部分的其他子系统（与开发的目标产品有关）。综合开发计划有它的阶段性里程碑目标任务，其中包括软件子系统，软件开发计划的里程碑目标应从项目计划中分解、细化出来。软件开发的详细计划出来后，符合项目要求的测试计划便可以跟随开发的计划进行测试阶段的划分与目标的制定。

表 5-2 迭代开发模式的测试阶段划分

软件需求	软件开发	软件测试
目标: 2009.1.1 ~ 2009.2.28, 完成模块 A, 模块 B 的一、二级功能设计, 以支持市场演示		
需求基线 1.0	设计方案 1.0	测试方案 1.0
需求设计 1.1	编码 1.0	测试用例 1.0
1.2		
1.3		
1. <i>n</i>		
迭代版本 2		
目标: 2009.3.1 ~ 2009.5.30, 完成模块 A, 模块 B 的三级功能, 模块 C, 模块 D 的一、二级功能可以使用, 以支持产品的系统级联调		
需求基线 2.0	设计方案 2.0	测试方案 2.0
需求设计 2.1	编码 2.0	测试方案 2.0
2.2		
2.3		
2. <i>n</i>		
...
迭代版本 <i>n</i>		
目标: 2009.9.30, 全功能版本, 版本正式发布		
需求基线 <i>n</i> .0	设计方案 <i>n</i> .0	测试方案 <i>n</i> .0
	编码 <i>n</i> .0	测试用例 <i>n</i> .0
需求基线 <i>n</i> 版归档	设计方案 <i>n</i> 版归档	测试方案 <i>n</i> 版归档

表 5-2 中反映的是典型的迭代开发模式, 它的最大优点就是需求、开发、测试密切相关的三方都在并行开展工作。需求是开发与测试工作的输入, 所以它也一直走在前面, 但有一点很关键, 就是无论哪个里程碑版本, 开发与测试的工作始终以某个需求基线为目标, 尽量降低频繁变化的需求的影响。

测试目标明确, 任务细化后, 团队成员的工作就有了一个共同努力的方向, 这是策略中很重要的一个环节。但有了目标, 如果没有跟踪过程, 会使得计划流于形式, 存在的问题不能及时发现与解决, 陷入一种测试过程失控状态, 最后的结果可想而知。如表 5-3 所示是一个测试任务过程跟踪矩阵表范例。读者可以根据所服务项目的不同特点加以修改便可使用。有了这个跟踪矩阵表, 项目的测试进展、质量完成情况、遇到的困难与解决措施等所有相关信息便可做到了然于心。通常这是项目测试负责人要做的事, 是对测试项目进行有序管理的策略。

表 5-3 测试任务过程跟踪矩阵表

测试阶段	测试任务	测试责任人	开始时间	结束时间	实际工作量(小时)	完成情况	困难与解决措施	备注
迭代版本 1	2009.1.15 前完成模块 A 功能测试方案设计, 第一轮评审完成	李然	2009.1.1	2009.1.15		按时完成		
	2009.1.31 前完成模块 A 功能测试用例设计	李然	2009.1.16	2009.1.31		正在进行		
	2009.1.10 前完成模块 A 性能测试方案设计	Carl Zhang	2009.1.1	2009.1.10				
	2009.1.20 前完成模块 A 性能测试代码编写	Carl Zhang	2009.1.11	2009.1.20				
	2009.2.1 开始第一次内部版本的功能测试, 可测试功能见提交清单, 完成时间待定	待定	2009.2.1					
迭代版本 2								
...								
...								
迭代版本 n								

过程跟踪机制, 可以及时发现问题, 但发现问题后还需解决问题, 方能推动各测试阶段的任务顺利完成, 更详细的过程控制方法见第 10 章关于此专题的介绍。

5.7 测试策略需考虑的其他要素

部署测试策略, 除了上节介绍的合理应用各种测试技术外, 还有一些不可忽略的要素需特别注意, 下面分别进行介绍。

1. 测试设计优先

避免没有用例进行的随机测试。无可否认, 随机测试能发现一些问题, 但它的特点是测试人员想到什么就测试什么, 导致有些功能点重复测试, 而有些业务根本就没测试到, 测试盲区无法控制, 整个测试工作陷入一团糟。测试是否达到结束的条件, 无法量化, 只

能靠拍脑袋说话。最后在用户端暴露出问题后，有些测试人员还在争辩，说这个地方他清楚记得测试过了，但又拿不出测试记录。朋友，告别这样的局面吧。

可以定义一个原则：测试用例没有设计好之前，不允许启动测试。

2. 保留清晰的测试记录

包括测试版本、测试人、测试时间、测试结果、发现的 Bug 等最基本的测试执行信息记录。如果用工具管理测试过程，这一点就不用人工参与，从工具中导出所需数据即可。

3. 模块独立化

能独立出来测试的模块尽量独立出来，这里包括一条龙的测试任务独立，包括测试方案设计、用例设计的独立。这样做的好处是质量容易评估，且专注于某一模块单元后，可测试得更加深入。

4. 子系统集成

在开发过程中，业务子系统不断迭代集成，测试需持续关注各模块接口的测试。需要时特别安排专门测试人员介入模块或子系统接口测试，如软硬件接口的驱动测试。

5. 系统级测试

任何时候，都不能忘记站在用户角度的系统级测试。系统级测试中发现的问题，有可能不是软件本身的问题，而是硬件或其他方面的问题，这种情况是很正常的。

另外，关于测试管理方面的相关要素，需依托在项目测试中进行考虑。

6. 人员培养

人员的培养，或许有人会说，这是管理线上测试经理们的事。这话没错，但是培养人才需要平台，这些平台来自于项目的支撑。结合领域的技术发展方向，用哪些技术来解决当前项目的测试需求，是测试设计师的任务。根据公司项目需求，需要哪些方面的人才，或需培养哪些方面的人才，测试设计专家更有权威。

7. 团队成长

俗话说“一方水土养一方人”，一个优秀的项目常能培育一支优秀的团队。项目测试中，团队成员除了完成工作任务外，还可以借助项目这个平台提高自己、完善自己、作为项目测试负责人，面对的团队成员能力可能参差不齐，有些是有经验的老员工，有些可能是刚毕业的学生，该如何凝聚他们的力量为项目服务，也是策略中需考虑的重要事情。结合项目的测试需求，推出一系列的培训计划，就是一种推动新人成长的有效措施。

第 6 章

聚焦测试方案的设计

本章首先回顾业界对测试方案设计的不同看法与做法，通过历经的事例，指出测试方案设计的重要性。使读者了解测试方案要解决的问题，以及方案设计的核心。接着给出最佳实践中提取出来的多种测试对象分析方法，包括三层架构模式分析法、多叉树节点分析法、业务状态变迁分析法、代码更改追溯分析法。对每一种方法，先进行原理上的讲解，然后结合案例与读者分享在具体的工作实践中的应用。

6.1 理解测试方案的设计

“测试计划”、“测试方案”、“测试用例”这三者之间的关系是什么样的，为什么业界有不少人会混淆这些基本概念呢？本节在回顾中将进行相关因素的客观分析，接着就测试方案设计的重要性及如何把握的方法与读者一起分享。

6.1.1 疑问与认识过程

说起测试方案的设计，有很大一部分业界人士会认为测试方案即测试用例，有小部分人认为测试方案是测试计划中的一部分内容。

记得在 1998 年，第一次读张海藩编著的《软件工程导论》第三版，其中提到“设计测试方案是测试阶段的关键技术问题，所谓测试方案包括预定要测试的功能，应该输入的测试数据和预期的结果”。感觉似懂非懂，说懂是因为当执行具体的测试工作时，是要输入测试数据的，结合操作步骤得出结果，再判断它的正确与否。说非懂是因为测试方案具

体是什么样子的并没有见过，也不清楚测试用例又是指什么（这与当时在国内专门的测试书籍非常稀缺有关）。

5年后，《软件工程导论》第四版与读者见面了，指定为“普通高校本科计算机专业特色教材精选”。笔者有幸捧读了此书，对原在第三版中似懂非懂的设计测试方案的内容又进行了阅读。书中提到“设计测试方案是测试阶段的关键技术问题，所谓测试方案包括具体的测试目的（例如，预定要测试的具体功能），应该输入的测试数据和预期的结果。通常又把测试数据和预期的输出结果称为测试用例”。比较两个版本中关于对“测试方案”的定义可知，第四版的主要变化就在后面的一句“通常又把测试数据和预期的输出结果称为测试用例”（一个微妙的变化，原第七章“测试”，第四版变为了“实现”，测试已演变为独立体系，仅放在一个章节中是说不清楚的，这与当时软件测试在国内正蓬勃地发展有关）。换句话说，测试方案包括测试的目的及测试用例。通过这样一番比较，笔者终于明白为什么“在业界，有很大一部分人士认为测试方案就是测试用例”了（目前，在网上搜索关于测试方案的资料，可能仍找不到一个比较完整的，并且让你满意的答案）。

在面试测试工程师时，笔者常会问起关于测试流程的问题，回答大都是测试计划→测试用例设计→测试执行→测试报告。当再问起测试方案的设计应包括哪些内容时，大部分人的回答提到了测试的目的、测试的范围、测试方法与测试用例；有少部分人回答是包括测试范围、测试资源、测试时间等，与测试计划的内容类似。笔者的第二个问题也正来源于此。其实，关于测试方案设计的这些回答，并不是谁是谁非的问题，其本身并没有标准的答案可言。但是，说到方案的设计，无论是什么方案的设计，原理应是相通的，就是要回答如何做的问题，要把测试某模块或某测试对象思路、方法说出来。如同我们在上学时解数学的分析题，要解决这个问题，把你的解题方法写出来，最好能拿出多个解题方法。

俗话说“实践出真知”，对于测试方案的设计，笔者有幸在一家公司任职时，把测试方案的设计正式纳入了测试流程来进行管理，有机会直接面对它。经过多年的历练，有了另一番的理解与认识，对测试方案要怎么样设计才有实际价值，也有了一些切身的体会。如图6-1所示是回答测试方案与测试用例及测试计划之间关系的简图。测试方案不等于测试用例，也不属于测试计划的内容。

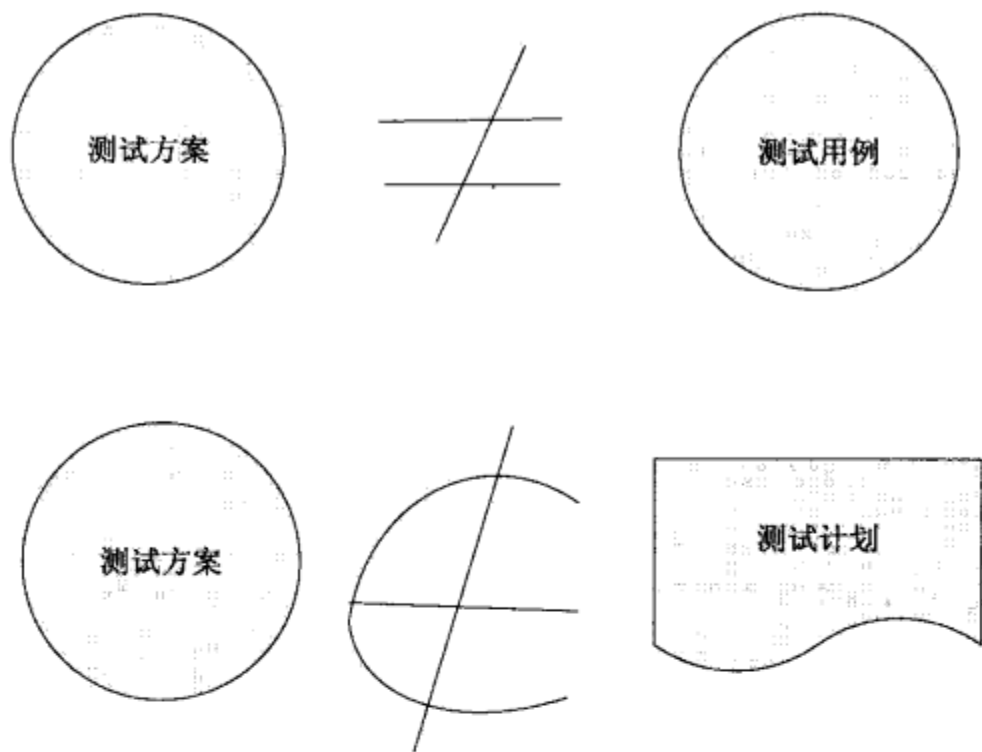


图 6-1 测试方案、测试用例与测试计划关系简图

6.1.2 测试方案设计的重要性

正如图 1-7 所示的软件测试流程简图，测试方案的设计是测试流程中的第一个设计环节。测试方案的设计是要解决从软件需求、设计需求，以及其他一些需求中提取出的测试需求该如何测试的问题。测试方案的设计关键是要体现测试对象的分析过程，最终得出合适的测试点与测试方法。再根据这些测试点与测试方法，设计测试用例，如图 6-2 所示为这个过程的体现。

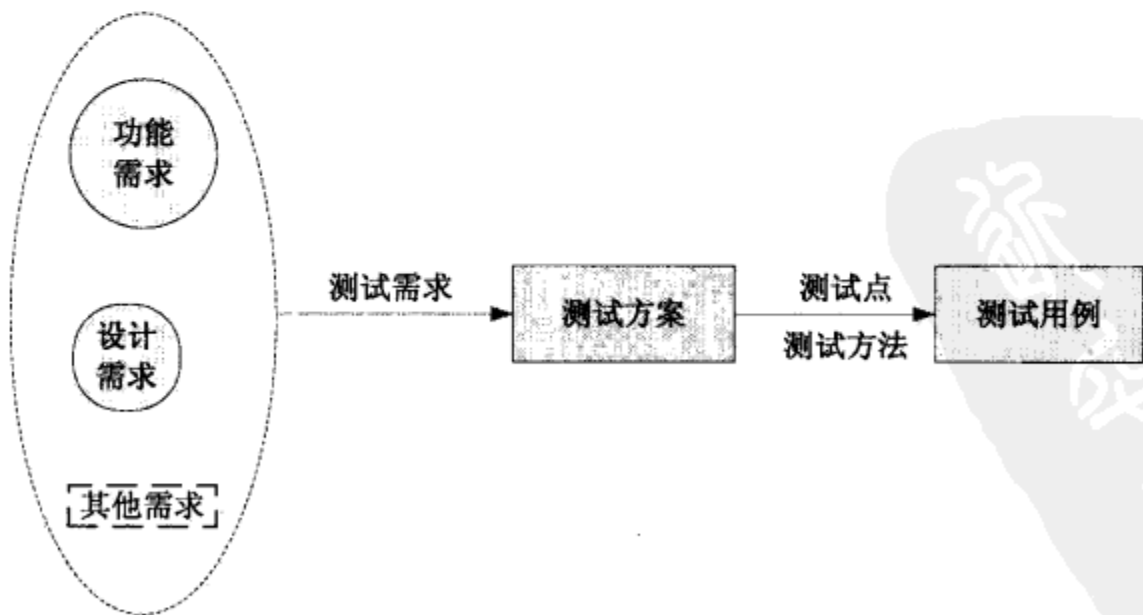


图 6-2 测试方案设计的输入与输出

测试设计中的测试方案与测试用例的关系，犹如软件开发的概要设计与代码设计的关系。在实际的项目开发实践中，此部分的开发与测试工作可以并行开展，彼此的输入都是需求，如图 6-3 所示。

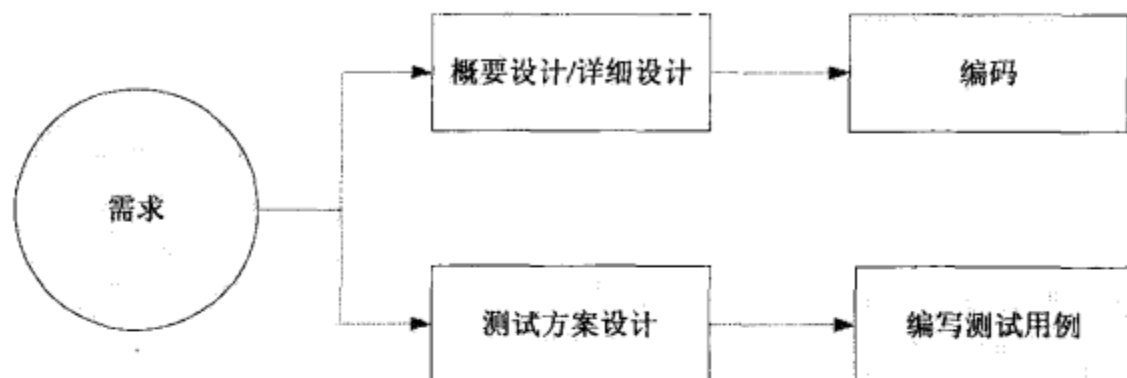


图 6-3 测试与开发并行设计工作示意图

关于测试方案的设计，笔者想起一些尴尬的现状。正如上节提到，有不少测试人员是分不清楚测试用例与测试方案的区别的，甚至认为测试设计仅是用例的设计。这就导致测试人员拿到需求后，就直接开始写用例，就像开发人员拿到需求直接写代码一样。这样做有什么坏处吗？相信读者看了下面的小故事后，会有所感触。

【故事】钓鱼——只知其然，不知其所以然

有个老人在河边钓鱼，一个小孩走过去看他钓鱼，老人技巧纯熟，所以没多久就钓上了满篓的鱼。老人见小孩很可爱，要把整篓的鱼送给他，小孩摇摇头，老人惊异地问道：“你为何不要？”小孩回答：“我想要你手中的钓竿。”老人问：“你要钓竿做什么？”小孩说：“这篓鱼没多久就吃完了，要是我有钓竿，我就可以自己钓，一辈子也吃不完。”我想你一定会说这是好聪明的小孩。

错了，他如果只要钓竿，那他一条鱼也吃不到。因为，他不懂钓鱼的技巧，钓鱼重要的不在钓竿，而在钓技。有太多人认为自己拥有了人生道路上的钓竿，再也无惧于路上的风雨，如此难免会跌倒于泥泞之中。就如小孩看老人，以为只要有钓竿就有吃不完的鱼，像职员看老板，以为只要坐在办公室里，就有滚进的财源。

下面是测试方案设计过程应注意的一些事项：

- 测试方案只有结论，没有分析过程。只说明某个测试对象（如某业务功能点）要测试的方法。为什么要这样测试，而不是那样测试？
- 方案的设计要能以理服人，测试经验固然重要，但没有以分析为背景的经验结果的直接表达，常会引导测试人员只看到对象的表面，而缺失深度分析。
- 测试方案的设计仅限于对软件需求的理解，是不够的，需要结合软件的概要设计、详细设计进行实现原理上的剖析，补充测试点与测试方法。

- 测试方案的分析结果，就是测试点与测试方法，可以通过建立索引表的方式，把测试点与测试方法转换为用例设计的思路。

6.1.3 把握核心——测试方案设计的三步曲

测试方案如何编制，是否有一个类似测试计划（GB8567-88）这样的模板呢？当第一次设计测试方案时，笔者也曾带着这个疑问在网上搜索，但没找到满意的答案。回到工作中，思考我们设计测试方案的主要目的是什么，为了达到这个目的，我们该如何做，至于其他内容认为是次要的。比如，有些是为了符合公司的开发文档模板要求而加入的内容，本身没有多大意义。以下是一个简例。

【测试方案模板简例】

第 1 章 概述

1.1 背景介绍

描述该测试方案设计的背景。

1.2 目的

编写本测试方案的目的，指出预期的读者范围。

1.3 用途

方案设计对象在软件系统中的位置、用户使用场景等。

1.4 定义

专业术语与缩写词的定義。

1.5 参考资料

列出此方案编写的参考资料。

第 2 章 测试范围

在顶层测试需求（通常项目测试负责人提供）的基础上，确认本测试方案的服务对象有哪些，如测试需求提出“模块 A 的测试”，在此处要明确包括其功能、性能、可靠性、与其他模块的接口等具体的测试方向。

第 3 章 测试分析

这是方案设计重点，根据测试对象在软件系统中所处的位置采取合适的分析方法，提取出需验证的测试点，以细化测试对象。测试分析的对象不同，要求立足点的角度不同，分析的范围也不同。如同软件开发中有总体设计方案、模块设计方案一样，测试也可分系统级的总体测试方案和模块级的模块测试方案，这个可以根据具体情况来操作。

第 4 章 测试方法

针对测试分析中提取出的测试点，提出具体的测试思路及方法，例如，哪些测试点采用手工功能测试，哪些采用自动化测试；哪些采用白盒测试，哪些采用灰盒测试等；是否有需要特别开发的测试数据、测试工具等。

第 5 章 测试用例设计

本处指出测试方案与测试用例的关系，例如，以测试方案中的测试点、测试思路为依据，设计测试用例，保证每一个测试需求都有测试用例与之对应。

具体的用例设计采用的工具，以及结合用例模板如何管理等也可在此处说明（庞大的用例集不用放在测试方案文档中，此处只说明用例设计的总体思路与方案的关系）。

第 6 章 其他（可选）

本部分内容可根据实际需求自行扩展，如写一些与方案有关，但想写没地方写的内容。在此测试方案模板简例中，第 1 章的内容是大众化的概述性内容；第 5 章是一个承上启下阐明测试方案与测试用例设计的一些关系，约束性声明；第 6 章是自由发挥的内容（可选）；其余 3 章，即测试范围→测试分析→测试点与测试方法，是一个测试方案的核心所在。这 3 个步骤被称为方案设计的三步曲，如图 6-4 所示。而此三步曲中，第二步的测试分析尤显重要，因为测试对象一旦分析清楚，分析到位后，测试点便也体现出来了，要采取什么测试方法，也已是水到渠成的事了。但测试分析的方法很多，具体要如何应用，才能着实体现出方案设计的核心呢？因此接下来结合实例，介绍 4 种在工作中常用且很有用的测试对象分析方法。

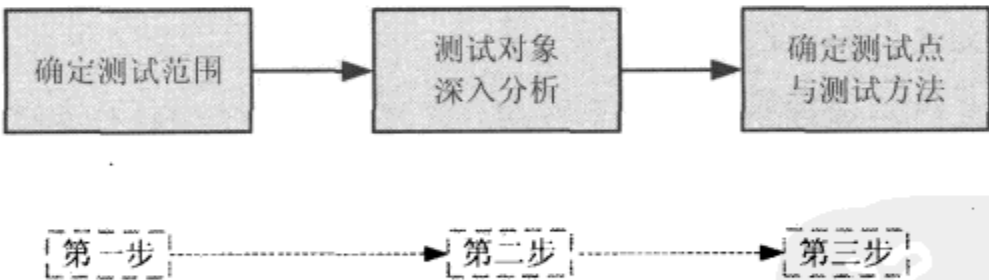


图 6-4 测试方案设计三步曲

6.2 创新乐园：多路测试分析方法

首先，要明确测试分析的目的，即提取与测试对象相关的所有测试点，并确定测试方法，为测试用例的设计提供有效的指导。我们最不想看到的漏测如何进行控制，关键就在这一步了。

测试对象分析是测试工作的枢纽地带，是测试需求（模块级需求或某方向性的需求）提取出来后，接着要开始的工作。一般由测试设计人员完成，是最体现测试人员设计能力的地方。

测试对象分析注重分析的充分性，也就是要尽可能把所有的测试点都挖掘出来，或许有人会指出，有些测试点如果考虑测试，付出的代价会很大，那么在进行测试对象分析时，可先不必考虑这些，不要因为成本的问题影响了测试分析的深入。事实上，当我们把一个对象分析得很透彻后，往往又找到不同的解决方案来节省成本。不同的测试对象，需采取不同的分析方法。如同安排合适的人，做合适的事，这样才会有更高的效率。分析方法众多，可谓是“道高一尺，魔高一丈”每一种分析方法都有它的独特之处，就像打仗用的兵器一样，各有千秋，在某些方面都很强悍（发现 Bug 的能力）。总之是各有各的优点，谁也不能代替谁。

小贴士：

测试项：在同一项目测试过程中，对具有相同背景的某一方面或某一类的测试对象的总称。在不同的应用场合，它代表的范围不同，是一个相对的概念，测试项可划分为功能测试、性能测试、安全性测试，此时测试项的划分原则是测试的类别；也可以以业务模块为单位进行划分，如 Windows 操作系统自带的记事本应用程序，可分为保存、另存为、打开、新增等功能测试项。

测试点：指某一测试项下一级的测试元素，如打开，可打开为文本文件，也可打开为非文本文件，如位图文件等。

检查点：或叫测试标题，是不能再细分的元素。在对测试对象进行分析时，需要分析到测试点；用例设计时，需对测试点进行展开，通过检查点（测试标题）来体现每条用例的测试目的。

分析之前，少不了熟悉需求，这个需求不能仅停留在产品需求或软件需求上，同时需关注系统总体设计需求、软件设计需求。还有需理解系统总体设计方案、模块设计方案等设计文档，以辅助对测试对象的深度影响分析。

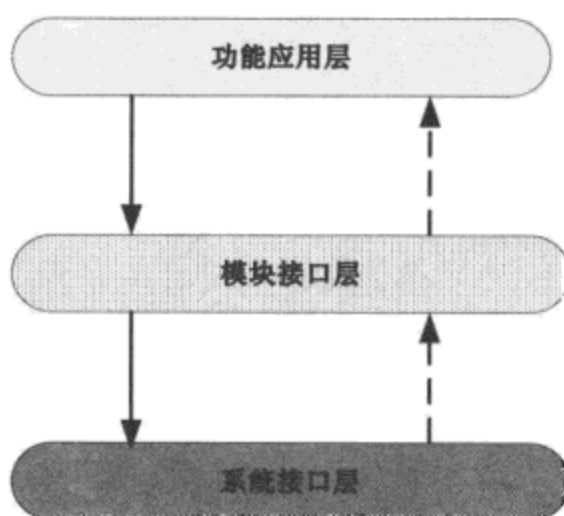
下面小节分别介绍最佳实践所得的新颖的测试对象分析方法，为方便读者理解，每一种分析方法结合案例的应用进行讲解。

6.3 三层架构模式分析法

不同的软件系统，三层架构中的内容可以不同，三层的软件架构模式很常见，此处的三层架构模式与软件设计的三层架构模式不同，但也有相似之处。本节先从理解其原理着手，然后结合案例应用，希望能帮助读者掌握如何用到实际工作中。

6.3.1 三层架构模式分析法的原理

首先要向大家介绍的测试对象分析方法是“三层架构模式分析法”，它是指把整个软件系统划分为3个层次，如图6-5所示，包括功能应用层、模块接口层、系统接口层；找出测试对象在整个系统中所处的位置，分析其本身业务功能，以及与其他模块之间，与系统接口之间的关系，然后由外向内，逐层深入分析。



实线：业务流，表示调用的依赖关系
虚线：数据流，表示数据的返回方向

图 6-5 三层架构模式

功能应用层，即用户接口层，是最外一层，直接面对用户，它的输入与输出用户能看到。模块接口层，即图中的第二层，在三层架构模式中处中间位置，它为功能应用层提供数据的输入，同时，也把来自于上层的数据传递到下一层的系统接口层，在三层架构模式中它是其上层与下层之间通信的桥梁。系统接口层，指与操作系统一起处理与硬件等其他子系统之间的接口模块，这一层是整个系统完成产品功能运行的支撑。对于上述3个层次的定义，是来自于实践中的总结，是从总结中形成的抽象理论。也许有些读者会问，我正在测试的软件系统比这个复杂，或不同类，也可以采用这种方法来划分吗？笔者认为万变

不离其宗，即便层次形式上可能各种各样，但其本质都是一样的。

下面以我们都熟悉的、与日常生活相关的手机闹钟功能的系统测试为例，讲解三层架构模式在实际工作中的应用。

6.3.2 应用案例

说到手机，顺便说一下，由于移动通信在中国最近 20 多年来的迅猛发展，从 20 世纪 80 年代末的大哥大、BP 机、GSM 手机，到现在的 3G 手机，无不影响着我们的生活，改变着我们的生活习惯。不说别的，单看拿手机当时钟来看的朋友们，料想已不计其数。在繁华、热闹的大街上，摩肩接踵的人群中，手腕上带着手表的人已是屈指可数，一些女性朋友即使带表也是为了装饰。就拿笔者自己来说，每天早晨进行 Morning Call 的时钟早就已是手机了（相信很多朋友的使用场景与笔者一样吧）。

言归正传，我们用三层架构模式分析法，来设计它的测试方案。

【案例】

首先，我们来看它的需求简述，因为需求是测试的主要依据。

【功能需求简述】：^①闹钟应用模块与用户的接口界面如图 6-6 所示。它提供用户设置响铃的时间，还可设计重复响铃的频率，包括日重复（每天的相同时刻响铃）、周重复（每周的某星期几响铃）、月重复（每月的同一天响铃）、年重复（每年的同一天响铃），如图 6-7 所示。如果同一天的相同时刻存在两种或以上的响铃提醒事件，如日重复与周重复事件时间相同，则先提醒重复频率低的事件，再接着重复频率高的事件，即先提醒周重复事件，接着提醒日重复事件，也即重复频率低的事件优先级高。无论手机当前处于何种状态（关机或开机、待机）及用户正在使用哪一种功能，只要响铃时间到，立即启动事件。响铃的过程中，按任意键停止响铃。



图 6-6 手机“闹钟”应用模块和用户接口界面

①手机上带的一款闹铃软件的功能需求，完全是一种假想的设计。假想在这种情况下，测试方案设计的三层架构模式如何应用，如果与某公司的设计吻合，纯属一种巧合。



图 6-7 闹钟用户设置界面

[测试范围]：闹钟应用模块的系统测试，包括功能测试、性能测试、易用性测试。功能测试主要指用户设置数据的保存、浏览、删除及各种事件提醒功能的正确应用；性能测试主要对提醒时间的准确性进行验证；易用性方面考虑用户常规使用场景操作的方便性。

[测试对象分析]：首先从功能应用层分析，从图 6-6 中可知，闹钟模块是手机产品软件系统中的一个应用模块，用户通过闹钟设置界面（见图 6-7），输入事件的提醒时间、事件内容及重复频率选择。其中，设置的每一项，以及它们之间的组合关系都需要作为测试点进行验证。数据的编辑、保存后的浏览、事件触发的正确性是属于模块内部处理的测试。易用性方面，包括闹钟设置时，数据存储操作的易用性，如给出人性化的、体贴的提示语。如果保存时遇卡没插好，应用界面宕机了（界面不动），是不可接受的。

事件设置完毕，用户可选择“确定”，把数据保存在内置的 SIM 卡中或外置的 SD 卡中，这就涉及文件管理模块的处理。需考虑存储时的各种情况，如存储卡满或坏了，或没有外置卡时的情况等。关于存储卡、存储空间的测试可作为其他独立项分析，此处只考虑闹钟模块与它的接口关系的测试。对于闹钟提醒事件，我们从需求“不管手机当前处于何种状态（关机或开机、待机）及用户正在使用哪一个应用程序，只要响铃时间到，立即启动事件”可知，闹铃事件实际上是一个具有高优先级的中断事件，这一点特别重要。由于整个手机软件系统可以有 N 个应用模块（有些手机可以自行在网上下载一些应用软件来使用，如游戏、电子书等），这些模块在应用过程中都有可能出现响铃事件，如正在编辑电话本时、正在播放歌曲时、正在拍照时、关机状态时、电池不足开机时等，用户常使用的场景都需考虑。对这方面的测试，需结合软件设计说明，把触发时机与手机当前的状态进行分

类总结，方能考虑全面。

小贴士：

SIM 卡是 Subscriber Identity Model（客户识别模块）的缩写，也称为智能卡、用户识别卡，是全球通数字手机的一张个人资料卡。

SD 卡（Secure Digital Memory Card）是一种基于半导体快闪记忆器的新一代记忆设备。

当事件的设置与保存成功后，最重要的当然是这个事件是否能如期按设置的时间及频率进行响铃提醒。响铃是由闹钟模块给系统的接口模块发硬件指令，由接口模块通过统一指定的通道传输给硬件子系统，由硬件子系统控制蜂鸣器的发声活动，如图 6-8 所示。关于铃声的大小、持续时长是需要验证的。触发时间的准确性也需注意。当然，如果条件允许，可以做一些非法操作，如取下蜂鸣器控制器的连接线，特意让其不能发声，验证响铃事件的触发情况（慎重，这种操作有破坏性）。做类似这些软硬件系统接口层的分析，需要我们对其设计原理有较好的理解。前面提到测试人员需要熟悉相关软硬件接口设计方案，也正基于此。

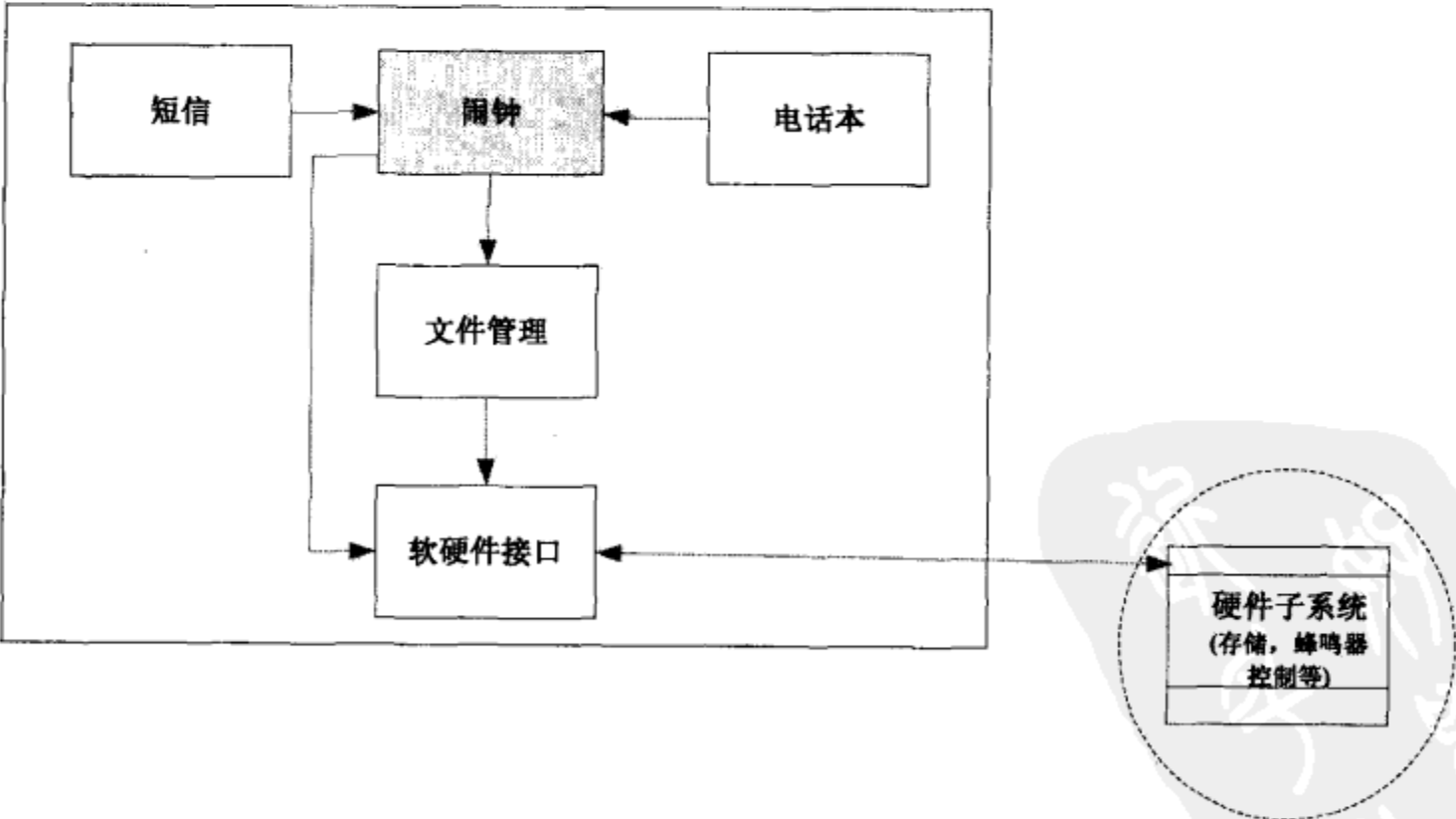


图 6-8 闹钟模块与系统中其他模块的接口关系示意图

[测试点与测试方法]：通过上节的测试对象分析，可提取出如表 6-1 所示测试项与测

试点，并给出对应的测试方法，以便设计测试用例。

表 6-1 “闹钟”模块测试点与测试方法列表

测试项	测试点	测试思路简述	测试方法	备注
闹钟设置	闹铃启动日期与时间的设置	检查日期与时间的可接受范围	边界值，等价类方法 选择输入数据	
	闹铃事件内容编辑	考虑编辑区为空，输满内容，输入特殊字符，输入数据保存后的修改，编辑过程中响铃	边界值，特殊值错误推测，非法操作设计用例	
	重复频率选择	不选，各个选项单选，两两组合，多个组合	边界值，正交法设计用例	设计用例时用正交表表达不同重复频率之间的组合
保存	保存在内置 SIM 卡中	卡满，保存后的信息浏览	边界值，正向用例设计	正向用例：此处指符合需求预期的正常情况下的用例
	保存在外置 SD 卡中	卡无、卡坏、卡满时的数据保存，保存后的信息浏览	容错，正向用例设计	
取消	取消设置操作	取消输入的设置值，退出	正向用例设计	
		输入数据后，非法操作取消保存	异常用例设计	
响铃触发的正确性	闹铃事件在各种设置下的触发正确性	不选、日重复、周重复、月重复、年重复、两两重复、多组重复	正交法，边界值	用正交表设计用例
响铃触发与手机状态的关系	手机开机、待机状态	手机开机后的不同状态下出现闹铃事件测试	用户使用场景测试	
	手机正被使用中	用户正在使用手机过程中，如正在某模块编辑信息、正在玩游戏等，突然出现闹铃事件，需分类综合考虑	用户使用场景测试	
	手机开机状态，电池不足	电池不足到了最后阶段，出现闹铃事件	用户使用扩展场景（特殊场景）测试	
	手机关机状态，有电	关机时，闹铃事件是否可自启动	用户使用场景测试	
	手机关机状态，没电	关机时，闹铃事件时间满足，但手机没电的情况	用户使用扩展场景测试	
	手机正在发声中（如播放音乐等）	正在占用着声道，闹铃事件是否能正常抢占声道	用户使用扩展场景测试	

此表中的测试项与测试点并不代表是全面充分的，此表在此仅说明三层架构模式分析法的最后一步是如何部署设计测试方案的。

6.4 多叉树节点分析法

多叉树节点分析法的提出来自于一次偶然的讨论会。本节先通过介绍小故事的方式帮助读者理解此分析法的精髓，然后通过一个应用案例介绍该如何应用此分析法。

6.4.1 多叉树节点分析法的原理

【智慧对对碰撞出的火花——多叉树节点分析法的故事】

故事发生在一次测试方案评审的激烈讨论会上，同一个测试项（新开发的小模块），测试经理 A 安排两个工程师分别用两种不同的方法设计测试方案。其中，工程师 A1 以传统的需求设计顺序为主线，罗列测试点，写出各测试点的验证方法（见图 6-9）形成测试方案。工程师 A2 采用三层架构模式的分析方法，通过系统性的分析需求，结合开发的设计方案，对测试对象进行了逻辑性的广度与深度方面的分析，结合图表体现出了分析的过程，最后提取出测试点与测试思路及方法。

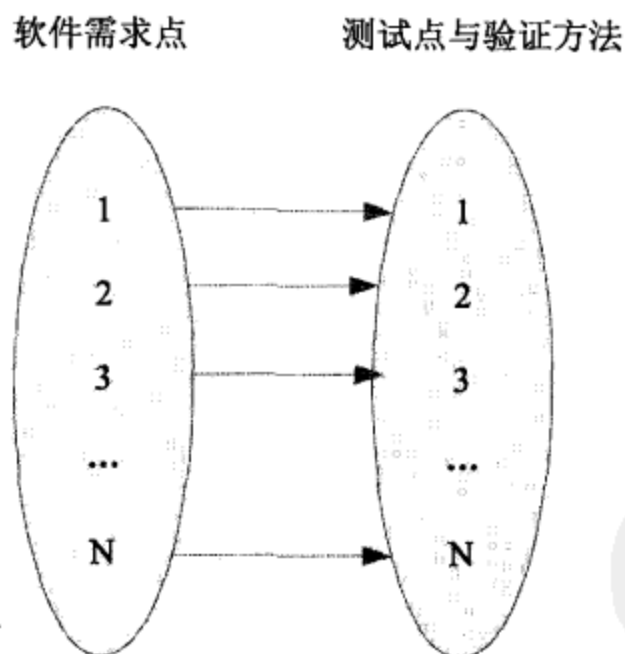


图 6-9 以需求设计顺序为主线的测试对象分析示意图

评审会上，首先亮相的是测试工程师 A2 的设计方案，由于采用的是新方法，方案中体现分析过程的图表较多，让大家耳目一新，很有测试开发设计的味道。把传统的测试方案没有体现出的分析过程表现得淋漓尽致，把很多经验丰富的测试人员想表达而从未表达出的思想一目了然地呈现了出来。但是在后来从分析过程中提取测试点时，大家发现有不少遗漏，特别是测试点与需求的对应关系上显得不足（这与方法本身无关）。借用当时一位评审专家

的说法“分析过程是很充分，很到位，但分析的结果提取有些粗糙。如何保证对需求点的覆盖，如何通过分析出来的测试点有效指导测试用例的设计，显得有点鸡肋”。而这方面正是传统的以需求设计顺序为主线设计方案的优点。工程师 A2 是一个新员工，工程师 A1 是一个经验丰富的老员工。到底哪一种方法更好呢，测试评审专家在议论着……

最后的讨论结果，老员工基本上从结果导向出发，不支持用新方法，而属新鲜血液系列的员工，喜欢用新方法，不愿看长篇大段的文字描述型的方案。此时，一位评审专家说：“测试方案设计是一种设计，所谓设计，当然不能没有分析，但是测试设计的主线放在用户的使用场景上，会不会更合适呢？”

那位评审专家的话音刚落，另一评审专家马上接着说：“对，我们可以以用户的使用流程为主线，把流程中的各节点作为起点，再派生出不同的使用场景，这些场景其实就是我们的测试用例。”

讨论会很激烈，大家你一言我一语，真是一次智慧的大碰撞，于是一个全新的 idea，全新的测试方案设计方法——多叉树节点分析法的雏形，就这样诞生了。

[多叉树节点分析法的分析思路]

多叉树节点分析法分析测试对象的总体思路：测试人员站在用户的角度，以用户使用场景为主线，结合设计原理分析目标测试对象与哪些因素有关，包括软件模块内部，模块与模块之间，软件与其他子系统如硬件、机械等。用户场景分基本场景和扩展场景，基本场景是指用户常用的功能操作流程，扩展场景是指某些比较特殊或不常用的功能操作流程。

由于这种测试分析方法很像一棵树，树的主干，就是测试对象在用户端的使用流程，树的枝节为使用流程中的节点，即为测试点。从某一节点出发（测试点出发），会有很多叶子，而这些叶子，就像我们对某一测试点进行测试时要关注的检查点，所以就叫多叉树节点分析法，如图 6-10 所示。

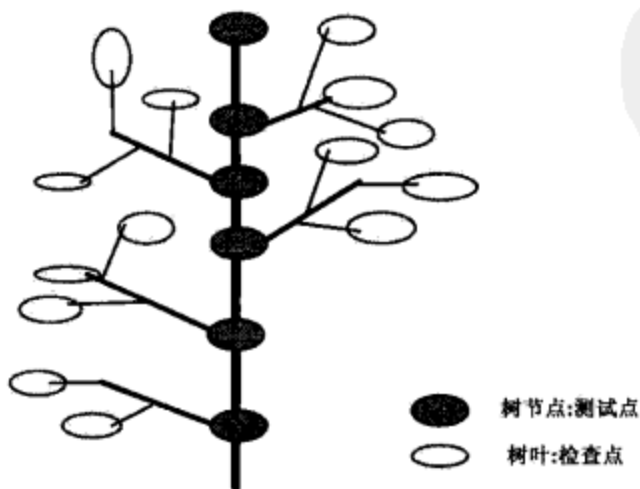


图 6-10 多叉树节点分析法示意图

6.4.2 应用案例

【多叉树节点分析法的应用案例】

此分析方法的关键是找出用户使用场景的主流程，然后以此为线索展开，找出沿途的各节点（测试点），接着再以测试点为起点，扩展用户的使用场景。

现仍以上节用过的闹钟应用模块为例，讲述此方法的应用。

首先，找出用户使用闹钟模块的主场景，假设用户使用场景：某用户白天 8:00 上班，设置每日早 7:00 进行 Morning Call，为了省电，晚上 10 点后处于关机状态。

分析真实使用场景，可把用户的使用场景分为基本场景和扩展场景。

1. 基本场景（成功的使用场景）

- A. 用户设置闹铃事件：设置闹铃启动时间、频度、提醒内容；
- B. 用户等待闹铃：对当前系统时间进行轮询，获取手机当前系统时间，与设置的时间相比，相同则触发响铃（手机软件内部处理）；
- C. 时间到，闹铃响起；
- D. 按任意键停止响铃。

如图 6-11 所示流程中的各节点即为测试点，它们构成了一个完整的使用流程场景。从各测试点出发，考虑完成测试点任务的周边影响因素，可扩展出其他分支使用场景。

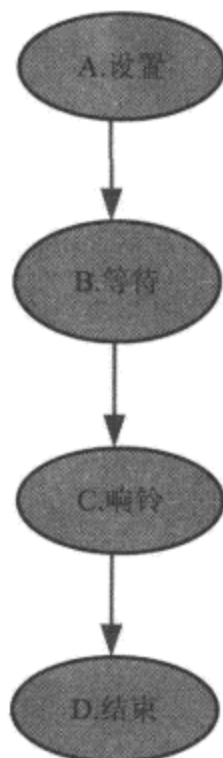


图 6-11 闹铃功能用户使用基本场景图

2. 扩展场景（特殊条件或失败的使用场景）

节点：A. 设置，相关扩展场景如图 6-12 所示。

A1: 事件设置失败，用户未发现，不响铃；

A2、A3、A4: 事件设置成功，但存在 2 个、3 个、4 个事件同时出现；

A5: 事件内容输入较多，需滚动条支持。

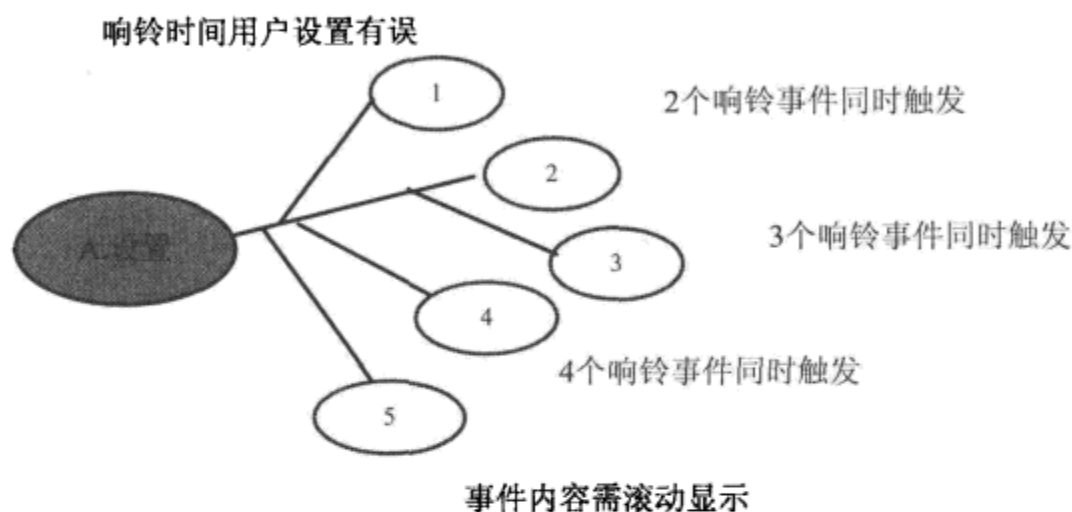


图 6-12 闹铃设置相关扩展场景

节点：B. 等待，相关扩展场景如图 6-13 所示。

B1: 待机状态等待；

B2: 关机状态等待；

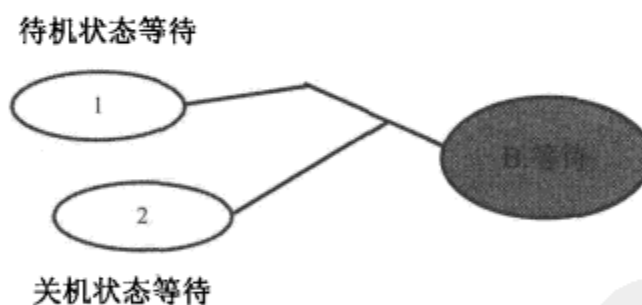


图 6-13 响铃等待相关扩展场景

节点：C. 响铃，相关扩展场景如图 6-14 所示。

C1: 手机通话中响铃；

C2: 手机播放音乐中响铃；

C3: 正在播放电影中响铃；

C4: 手机编辑信息中响铃。

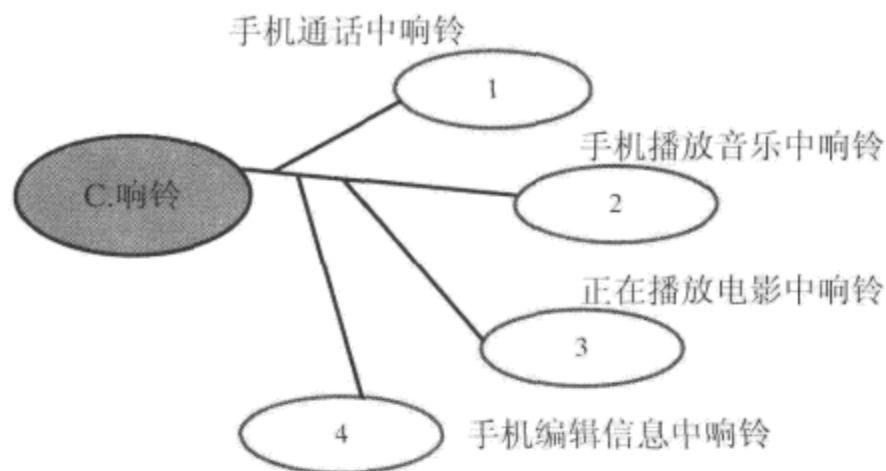


图 6-14 手机响铃相关扩展场景

节点:D.结束, 相关扩展场景如图 6-15 所示。

D1: 响铃一直叫, 不理睬;

D2: 人为断电关机。

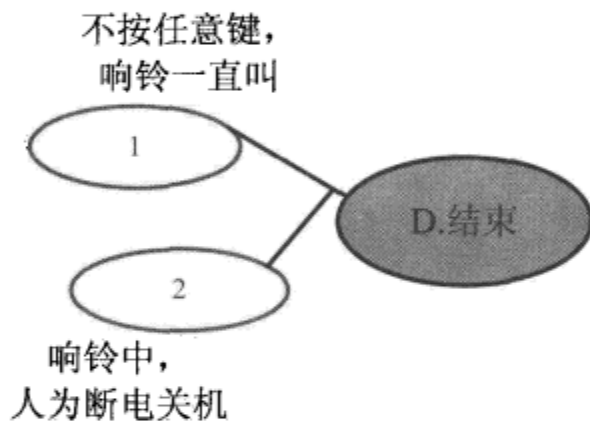


图 6-15 手机响铃结束相关扩展场景

注：场景分析图中的表达仅是例子，并不是全部场景的体现。

[多叉树节点分析法小结]

聪明的读者朋友，也许你已注意到，通过基本场景及扩展场景的分析，测试点与对应的用例已跃然纸上。如上例中的 A1、A2、B1、B2 等，已很容易转化为测试用例。但是，我们再回过头来思考需求与设计实现原理，发现也会有缺失。如下面场景并没有在场景分析中提到（与用户场景好像关系不大，或比较极端）。

- 时间准确性：连续一周、一月、一年后，测试手机时间的精确度。
 - 手机电量不足，关机状态时响铃时间到来，响铃事件是否可以正常触发？
 - 手机没装电池，响铃时间到来，时间已过后，再装上电池开机，结果如预期吗？
- 以上观之，多叉树节点分析法也有它的优缺点。

优点：主要考虑用户的实际使用场景，能把握测试的核心，保证了用户常规使用功能

的正确性与稳定性。测试点的影响因素，包括内外因素，能一目了然，能保证测试的全面性，不遗漏测试点。

缺点：作为一个模块或一个软件系统，除了软件的功能，还有与软件相关的特性，如性能；与其他子系统相关性的特点，如响铃与自动开机的关系、与手机电量的关系等。这些方面如果使用此方法，就很容易被忽略。

“道路是曲折的，前途是光明的”，走改革创新之路总是要付出代价的，这是事物发展的必然规律。评审会上形成的 idea，却是难得的智慧碰撞的火花。在项目测试工作中实际使用时，可结合其他方法一起使用，相互弥补，以达到效果最大化。

6.5 业务状态变迁分析法

业务状态变迁分析法与软件工程中提到的有限状态机很相似，只是此处的业务状态是指用户层面的功能业务状态。是一种有效提高用例设计的有效性 & 全面性的测试对象分析方法，本节在介绍其原理的基础上结合案例进行讲解。

6.5.1 业务状态变迁分析法原理

有限状态机这一软件工程方法，在软件设计中的应用已很普遍，此处向大家介绍的“业务状态变迁分析法”与它有相似之处。

业务状态变迁分析法，首先把软件完成业务功能的过程，看成是一序列状态变迁的过程。当条件满足时，软件就从一种状态切换到另一种状态。当软件处于某一种状态后，以此状态为测试平面，寻找各种触发点，使其到达另一种状态。用此法分析测试对象的核心是识别出合适的状态平面，然后捕捉每种状态平面上的触发点，直到完成一个完整业务功能点的分析。当把某一业务功能点的完整状态变迁图画出来后，测试点与测试用例的导出已水到渠成。如图 6-16 所示，假设它是某业务功能点的一个业务状态转换图，那么状态 A、状态 B、状态 C 是完成一个完整业务功能点的 3 个测试状态平面。事件 A、事件 B、事件 C，分别是 3 个测试状态平面的触发点。图中每一个状态平面的转换，只有一个事件，而实际工作中，对某一状态平面，常会有多个触发点，状态平面之间的变迁也是相互交错的。

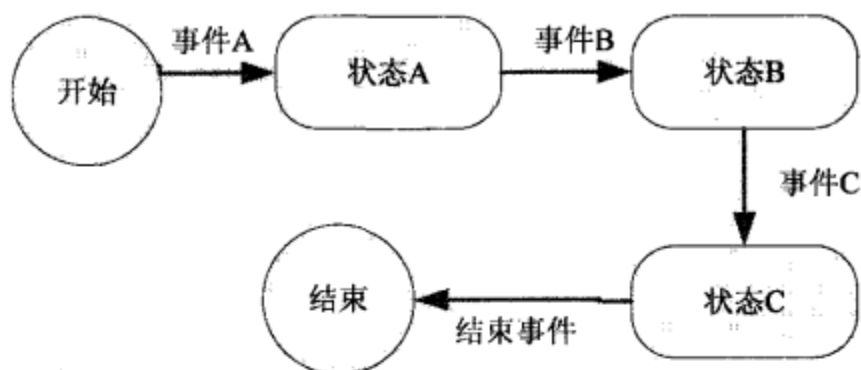


图 6-16 业务状态变迁图

6.5.2 应用案例

【业务状态变迁分析法的应用案例】

通常，手机在开机后便进入待机状态（等待用户的使用），如图 6-17 所示。如果不马上使用，为了节省用电，1 分钟后（不同厂家可能默认时间不同），手机将自动从待机状态进入省电模式。如果需继续使用，按任意键则可退出省电模式，回到原来状态。这些是我们经常遇到的使用手机时的场景。如图 6-18 所示是从系统层出发描述上述使用场景的状态变迁图。



图 6-17 手机实物图

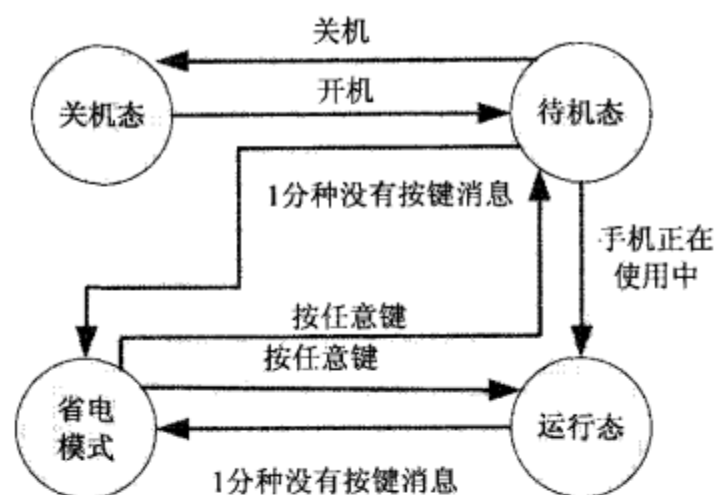


图 6-18 系统层的手机工作状态变迁图

图中，关机态指手机处于关机的状态，用户通过开机动作（如长按开机键，即图 6-17 中的挂机键），手机开机后，便进入待机态。此时，用户可使用手机本身提供的任何功能，如输入电话号码、播放音乐、发送短信、玩游戏等。在使用手机业务功能时，手机软件处于运行态（为加以区别，本文中的叫法）。同样，尽管手机实际上正处于忙的过程中（处理业务功能），但如果连续闲置超过 1 分钟，则会自动进入省电模式。同样，手机在待机态闲置超过 1 分钟，也会自动进入省电模式，按任意键则回到原来的状态。

假定系统层的下一层是模块业务层，根据其测试范围和业务的复杂程度，可以再把系统层的状态细分为多个层次。现在就以我们在日常生活中常用的编辑电话本信息为例，讲解在具体的模块功能测试时，状态分析法如何应用。

要保存朋友的电话，首先我们要进入电话录入界面，进入的方式可能有多种，可通过设置快捷键的方式（只按面板上的某一个键，与用户的具体设置有关）进入，也可从厂家提供的应用模块接口进入，如图 6-19 所示。

选中“电话本”后按确定键进入电话本录入界面，如图 6-20 所示。



图 6-19 电话本应用模块入口

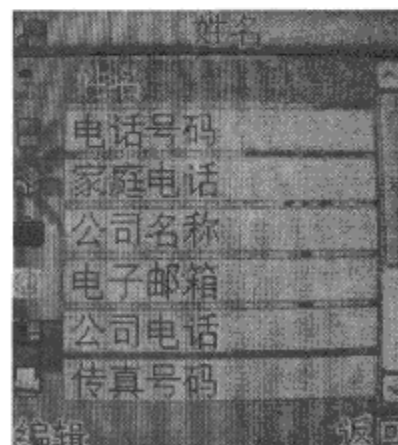


图 6-20 电话本录入界面

图中界面包括“姓名”、“电话号码”、“家庭电话”、“公司名称”等字段，“姓

名”字段正处于选中状态，表示用户可以开始输入朋友的姓名了。在此状态下，从界面下方的提示按钮可知，按左软键进入“姓名”的编辑界面，按“返回”回到上一界面。细心的读者也许已注意到，除了这两个功能按键响应外，还可以按向上、向下方向键以翻页滚动浏览信息。对于其他没有定义功能的按键，如数字键、左右方向键、拨号键、挂机键，这些按键按下后，软件是否做了正确的响应，是需要测试的。另外，当软件运行到此界面时，在某些时刻会出现突然闹铃事件（与机主是否设置有关）、收到短消息提示事件等。那么，界面突然被其他信息抢占了，当这些信息消失后，原来状态是否恢复正常，都是测试要考虑的范畴。显然，我们可以以此界面作为一个测试状态平面，围绕此平面捕捉此状态界面会发生的事件。如图 6-21 所示，是电话本录入界面状态与事件分析图（图中的事件不一定全面，如当手机电量低时，在此界面也可弹出提示，真实测试场景时需根据产品的需求及设计补充全面）。图中的圆心可以当做测试项，以圆心为端点的各条射线，如 OA、OB、OC 等，可看成是为圆心（测试项）服务的测试点集合。

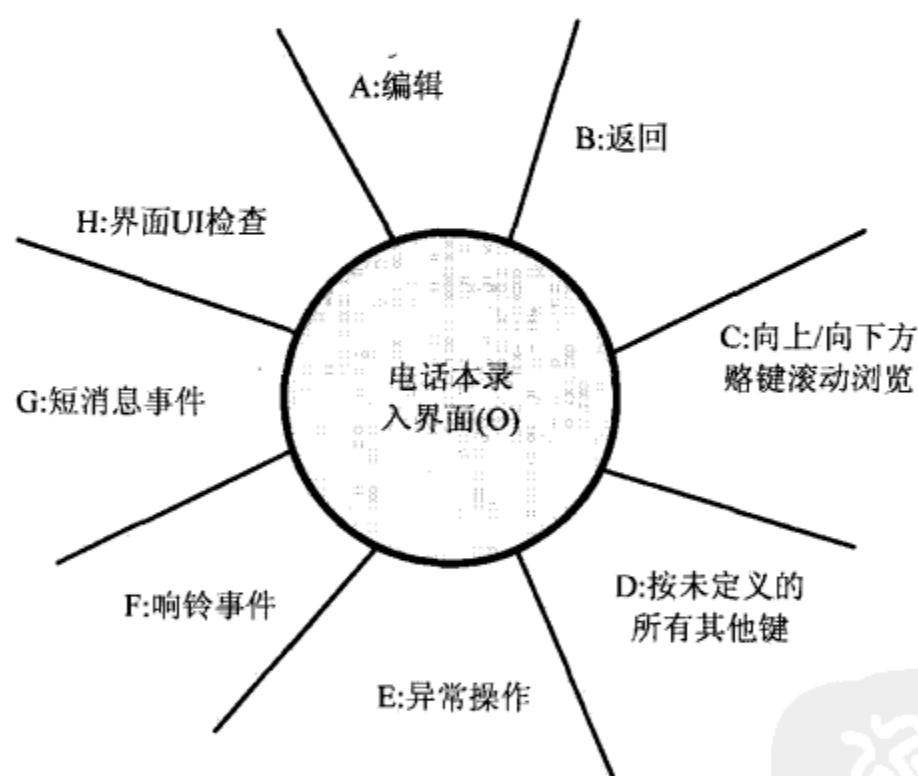


图 6-21 电话本录入界面状态与事件分析图

选中“姓名”，点击“编辑”，进入“姓名”编辑界面，如图 6-22 所示。同样，此界面下方的“选项”、“清除”显示的是功能按钮，“拼”表示当前为拼音输入法状态，单击此按钮，可切换不同输入法。同电话本录入界面一样，除了一些显式的功能响应需测试外，其他按键的响应及此界面会遇到的其他事件，如短信事件的触发，如图 6-23 所示的在姓名编辑过程中收到短消息通知；响铃提示界面突然弹出，如图 6-24 所示，同样需要测试。无疑“姓名”编辑界面又是一个测试状态平面，按系统层的测试状态来说，它属于“运行态”。

当姓名录入完成, 点击“选项”, 弹出如图 6-25 所示界面, 可以选择“完成”, 以结束“姓名”的录入过程。也可以返回原界面, 继续编辑姓名。同理, 此界面状态我们又可把它当做一个测试状态平面, 继而测试其他功能按键的响应是否符合预期。至此, 对于新增朋友姓名的业务流程及功能已分析完毕。对其他字段编辑的业务流程与功能可用此方法继续进行分析。

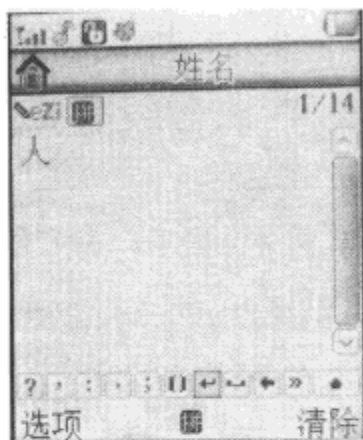


图 6-22 “姓名”编辑界面

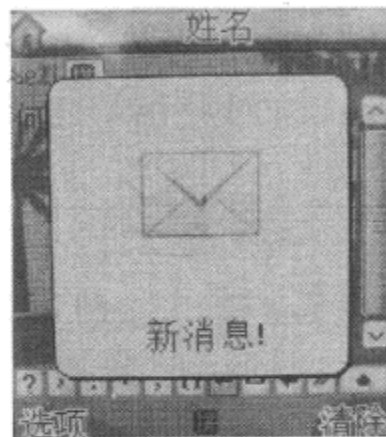


图 6-23 姓名编辑过程中收到短消息通知



图 6-24 姓名编辑过程中闹铃响起

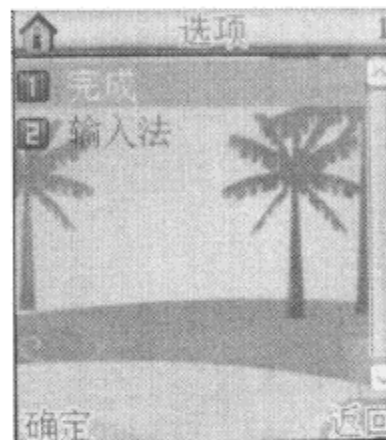


图 6-25 “姓名”输入完成确认界面

从在电话本中录入姓名到最后的保存, 其业务流程状态变迁可总结为如图 6-26 所示。

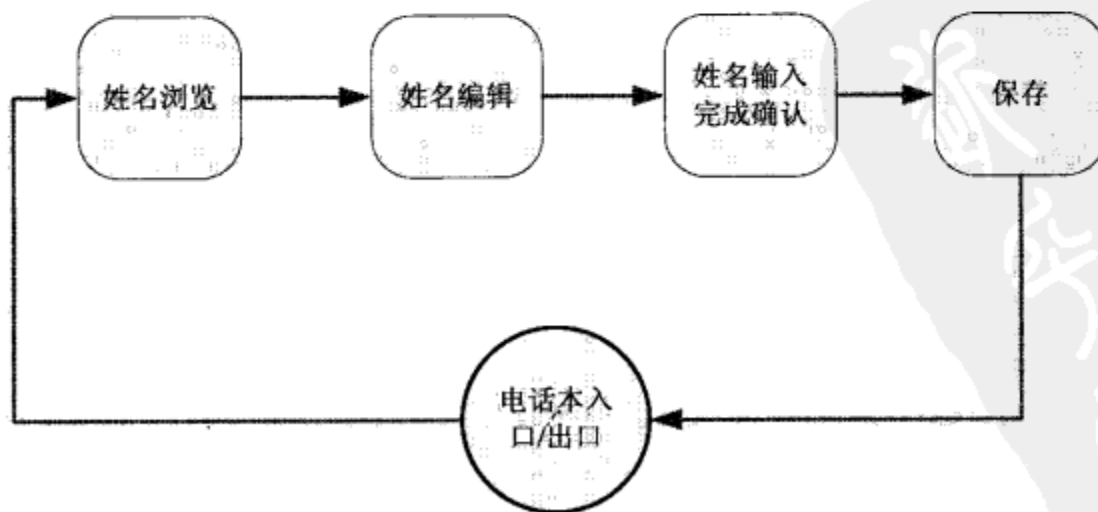


图 6-26 电话本姓名录入业务流程状态变迁图

读者朋友或许已注意到，前面讲述的只是姓名录入到完成录入的 3 个状态，保存状态并没有提到。保存状态，是对于电话本中所有字段而言一起保存的，录入的信息也只有保存到 SIM 卡或外置卡（如 SD 卡）上后，增加朋友的号码到电话本中的业务才算真正完成。

[业务状态变迁分析法小结]

业务状态变迁分析法作为测试对象的分析方法之一，也有其优缺点。

优点：从事件分析图中（见图 6-21）能清晰地看到测试项、测试点，再从测试点出发设计详细测试用例，能找出清楚的对应关系。测试项的业务流程连续，流程中各节点考虑周全。

缺点：对每一状态点同等对待，重点不突出，容易造成在用户不常用的功能或场景中，花费不少时间进行测试设计与执行，而在重要路径的使用上反而没有重点测试。因为这种方法，把触发业务路径中的所有状态条件都进行了测试。依赖经验强，如对状态点的提取、划分，状态点的外界触发，需要对产品知识或系统层的设计非常熟悉。某些事件的触发与界面并无关系，但在每一个状态界面都进行测试，如短信、闹铃事件。实际操作时，需结合设计实现原理加以分析，以减少重复用例。

6.6 代码更改追溯分析法

此方法从代码变更的角度出发进行分析变更记录，控制变更对测试的影响是测试对象分析方法的一种有效补充。

6.6.1 代码更改追溯分析法原理介绍

前面介绍的 3 种方法，基本上都是从需求出发，结合设计需求，站在用户角度层面对测试对象进行分析的。然而软件的生命周期中，要经历需求分析、概要设计、详细设计、编码、测试、维护各阶段，特别是软件上线后的维护阶段。例如在线的软件更改，比在研发阶段的测试需求分析，需更加全面、到位，稍有失误就会造成终端用户的投诉，甚至会导致产品的召回门事件。更何况在线软件更改时，质量要求高且时间急迫。下面就介绍这种情况下的一种测试对象分析方法。

代码更改追溯分析法，是指通过分析更改的代码，追溯更改的来源，分析它的影响，然后确定测试的对象与测试方法，是一种灰盒层次的测试方案设计方法。

追溯分析法，首先要确定代码的更改是否有来源，这是分析的基础，也是接着分析下

去的原则。来源通常包括新增需求、Bug 更改、代码的优化。更改的来源也是测试需求的来源，更改本身是否到位、是否对原有功能有影响等。测试分析时需要提出一系列的疑问（说到这一点，好像很不相信开发人员的工作，其实是职业要求使然），进行测试分析时，这种质疑精神是必须的，这些疑问将推动形成解决问题的思维空间，也正是这种思考的过程，最终才能得出合理的测试对象与高效的测试方法。其次，追溯分析法中需做更改的影响分析，这一点是难点也是重点。对于一个新项目，由于一直在实现新需求，代码的变化量大，代码的更改属于正常状态。而对于在线项目，每一个更改都必须进行严格的控制，更改的影响分析也必须周密严谨，如果出现增加了某一个功能，造成原有功能失效，是绝对不允许的。

如图 6-27 所示是代码更改追溯分析法的示意图。

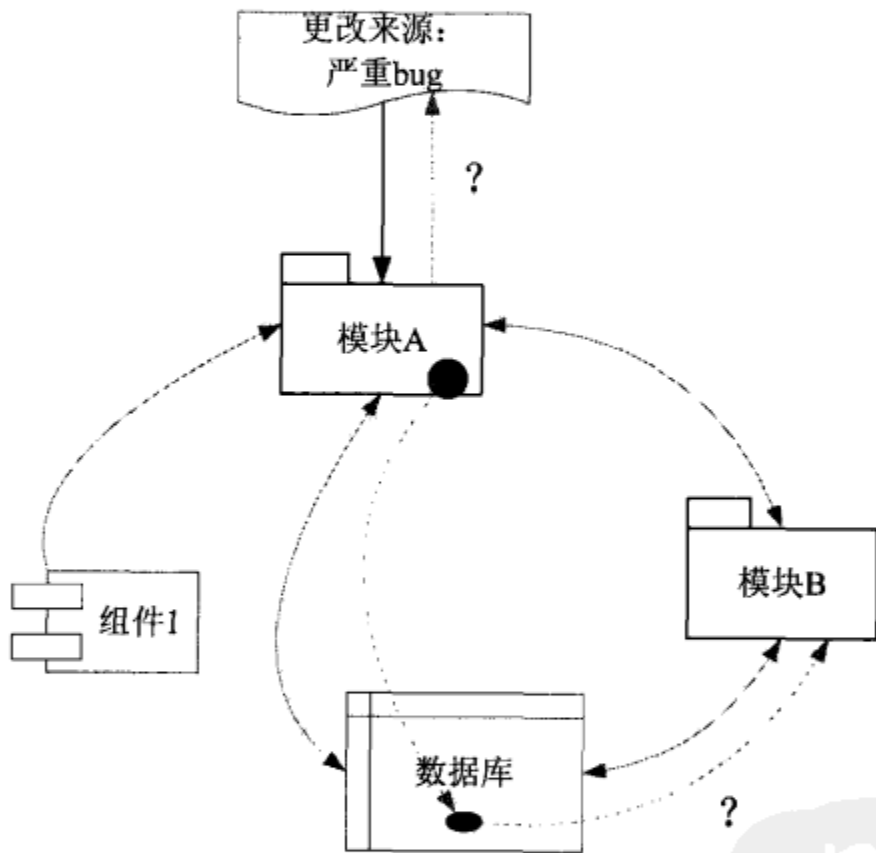


图 6-27 代码更改追溯分析法示意图

图中，模块 A 因为有一个严重的 Bug 存在，于是对它做了个补丁，如图中红色的圆圈所示，而这个更改涉及文件的存储，于是数据库管理模块也连带了做了修改。而此数据库管理模块同时也被模块 B 调用，那么模块 B 会不会受到影响呢？这正是需要进一步分析的地方。组件 1 模块由于是被调用者，没有受到影响。

在工作实践中，这种方法如何操作应用，下面结合案例进行讲解。

6.6.2 应用案例

【开发人员的分析也可能出错】

人员介绍：张 A，某公司 M380 手机项目的测试负责人，李 C 是负责此项目电话本与文件管理模块的开发工程师。

需求定义：用户可新增不同的群组来管理自己的电话本，“公司”、“家庭”是群组列表中默认的两个群组。当用户新增电话记录时，可选择此条电话是属于“公司”群的，还是属于“家庭”群的，如图 6-28 所示。

存在问题：现在，某测试工程师提交了一个电话所属群组标识错误的 Bug，Bug 的详细描述为：“用户新增了一条属于公司业务群组的电话记录，但保存后，在浏览列表中却显示为家庭群组的标识了”，如图 6-29 所示（见红框标识）。

查看电话本	
	0755-2895236
	0759-3358962
	020-55892350
	13658105892

图 6-28 预期的电话本记录

查看电话本	
	0755-2895236
	0759-3358962
	020-55892350
	13658105892

图 6-29 出现群组标识错误的电话本记录

一天，早上上班刚打开电脑，张 A 就看到了新版本已发布的消息，消息是昨天晚上发布的，消息中说除了解决电话所属群组错误的问题外，开发人员在更改代码中还看到了一个参数传递错误的问题，并说会导致电话所属群组完全错乱，且是偶发。于是开发人员很自信地更改了那句代码。由于需要更改来源，要求测试人员补录一个 Bug 作为更改来源。

看到这个消息时，张 A 很纳闷，如此严重的问题即使是偶发，过去近两年的时间怎么就从来没遇到呢？现在版本已发布，需马上组织相关人员进行“会诊”，评估更改的影响分析，测试该如何做才充分呢？于是邀请了开发人员李 C、开发负责人、测试工程师进行了讨论，李 C 讲解了问题的前后，测试负责人张 A 于是安排测试工程师对更改进行压力测试，对所有影响到的业务功能进行回归测试。

然而，张 A 总是觉得不太可能出现李 C 说的情况，于是从代码仓库中取出变更前后的代码，对仅更改到的一行代码进行了认真分析。电话本信息结构体定义如下：

```
Struct {  
    Telgroup TelGroupData;  
    Char Name[30];  
    Unsign Mobil;  
    Unsign HomeTel;  
    Unsign CompanyTel;  
    Unsign Fax;  
  
}TelRecord;  
  
TelRecord pstTelRecord;  
.  
pstTelRecord = TelRecord.TelGroupData;  
  
FormatGroup(TotalNum,pstTelRecord, acEdit[]);
```

原 FormatGroup () 函数原型定义如下：

```
FormatGroupFlag(int num, TelRecord *pstTeleRecord,*pBuf);
```

问题出在传递给第 2 个参数的指针固定为 TelGroupData 的地址了，而它是 TelRecord 结构的第一成员。当此结构体没有变更时，这样使用是正确的，但当结构发生变化，并且在 TelGroupData 这个成员的前面增加成员函数，就会导致传递给 FormatGroupFlag()函数中第二个参数的首地址不正确，从而使结构标识错乱。张 A 把分析结果说出来给开发人员听时，开发人员只好再次打开代码分析，并确认张 A 的分析是正确的。

看来，开发人员也会有理解错误的时候，不能轻易相信，而造成测试盲目过度。

小结：分析得出的测试对象与测试方法，可采用以黑盒业务测试为主，辅助以分析代码做影响分析，避免测试过度。



第 7 章

话说用例的设计

测试用例设计是测试人员的基本功，要打好测试之战，基本功要过硬。本章首先通过一个漏测的质量事故的例子，阐明用例设计的重要性与意义，然后对当开发流程不规范时，如没有需求与设计文档等依据的情况下，如何设计用例提出一些解决方法，希望对陷入此种困境的测试朋友有一些帮助。接下来提出了几个行之有效的用例设计方法，并结合案例讲述如何应用，想提高设计高效用例方法的朋友能从中找到想要的内容。有了用例设计的方法，还要具备对用例的正确认识，包括对有效、无效用例的正确认识，以及用例给我们带来的价值。用例是测试人员心中的世界，如同代码是开发人员的灵魂，用例的复用、重构是值得我们思考的事情，也将在本章与读者一起分享。最后，介绍了用例的设计规范，作为本章的结束。

7.1 漏测一个提示界面，不仅损失 158 万元

漏测，这个词对于测试界的朋友来说，是最不想听到的一个词。尽管不太可能把软件中所有的 Bug 都找出来，就像开发人员不太可能开发出零 Bug 的软件，但对测试人员提交给自己的 Bug 常耿耿于怀。另外一方面，漏测与否一直以来是很多公司衡量一个测试人员工作质量的重要考核点。

漏测指测试人员测试通过的软件在后来被他人或自行发现仍存在 Bug 的现象。由于软件的特殊性，漏测现象经常发生，每发现存在漏测 Bug，相关测试与开发人员需要进行漏测分析，分析此 Bug 的严重度、发生的概率、测试用例是否存在缺失、开发更改代码后的影响分析是否存在不足等。重要的是找出漏测的根源，提出改进的措施，避免同类问题反复出现。

【案例】

事情虽然过去 10 多年了，但依然记忆犹新。那时笔者与一些测试同事一起在一家研发与生产 PDA^①的私营台资企业工作。一天早晨，老板一反常态地在他的办公室里与电话中的对方大声争吵，最后非常生气地把电话挂掉了。接着，开发与测试主管被叫到他的办公室，办公室的门被关上了，他们在悄悄地商量着什么。后来才知道我们已发布出去的软件，在法语版本上，有一个提示界面仍用英文显示着。测试漏测了，而这个点在用户试用现场，被发现了。我们听后都不敢相信这个事实，因为这个提示信息在常规操作中就能出现。问题已出来了，怀疑无济于事，开发与测试相关同事立即动手检查分析，最后开发定位到是由于有一个地方的处理用了 HardCopy（硬拷贝），把英文字符串直接写在代码中了（据负责的工程师说，当时是为了调试的方便，后来在把字符串提取出来作为独立的资源文件时，漏了此字符串。而在对字符串资源进行本地化翻译时，也没有意识到此问题的存在）。

对于测试来说，是漏了这条用例，是用例设计的遗漏。

查出问题的原因后，软件很快进行了更改，测试复测，补丁版本当天即可以重新发布。然而，事情影响的严重性却超乎我们的意料。老板跟我们说，那家客户因为与其他代理商合同的原因，已不能接受我们的产品了，一个 158 万元的单就此泡汤。一个单是小事，但由它带来的负面影响是很难用短期的金钱来衡量的。这也许就是老板为什么会如此生气的主要原因吧。后来负责此模块的开发、测试人员受到通报批评，并扣除本季度全部奖金，但与公司的损失相比，却也算不了什么。

第二天，老板把我们所有测试人员叫到他的办公室，问及我们漏测的原因是否找出来了，对日后如何进行防范是否有什么措施。反思的结果是问题与我们现在的测试模式密切相关，我们没有用例，是在看着简单描述的需求来测试，时间长了，连需求都不看了（自认为需求记住了，看懂了）。平时测试完全是一种随机测试（目前很多人把它叫做 Ad hoc 测试），无系统可言，如图 7-1 所示。测试者的想法及操作没有留下任何记录，俗话说“好记性不如烂笔头”，一段时间后，哪些路径测试过，哪些没有覆盖到，无从说起。最后导致一些业务流程被测试 N 遍，而有些用户场景却一次都没有遍历到，也是顺理成章的事。

① PDA 是 Personal Digital Assistant 的缩写，个人数字助理的意思，是一种集中了计算、电话、传真和网络等多种功能的手持设备。

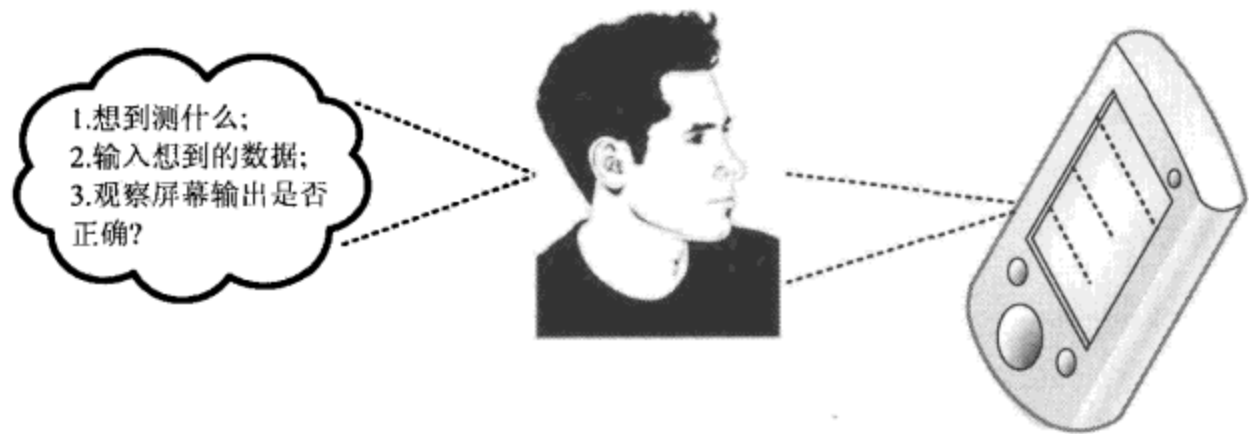


图 7-1 手工随机测试 PDA

真佩服当时的老板，讨论解决办法的当时，亲自拿出一张 A4 空白纸，在上面写着一个个测试功能点，要求我们按照这种方法把各模块的功能点一个个列出来，测试过的则在旁边的小框内打钩，如表 7-1 所示，后来我们把它叫做测试 checklist。回想起来，这虽然不是合格的测试用例，但比起原来拍脑袋式的随机测试，已经向前跨了重要的一步。

表 7-1 模块功能测试 checklist

功能测试 checklist			
模块名称:备忘录		测试人:	
		测试时间:	
序号	功能点描述	测试结果	备注
1	新增记录	✓	
2	保存记录	✓	
3	完全匹配查询	×	
4	模糊查询	✓	
5	删除一条记录	✓	
6	删除全部记录	×	
7	内存满时新增并保存	×	
8	删除记录再查询该记录	✓	
9	浏览记录	✓	

7.2 逆境中的用例设计

通过前面的案例事件，相信读者已充分理解在测试执行之前，事先设计好测试用例的重要性与必要性了。如果公司有着比较规范的开发流程，那么恭喜你，正常情况下，按照常规体系，在测试方案设计完成后，根据在测试对象分析中找出的测试点，结合详细需求、

详细设计等文档，用例的输入与输出明确、详细的用例设计可顺利完成。但是当我们在实际工作中着手用例设计时，常会遇到“愿望是美好的，现实却是残酷的”尴尬局面。如公司开发流程不规范，没有需求文档没有设计文档，或这些文档不全，但又要设计用例，怎么办呢？下面几节中将介绍在几种非正常情况下，如何设计用例的具体做法与大家一起分享（有些人可能觉得很正规，不可学之，但重要的是它有用，比起没有用例来它的作用更是立竿见影。特别是产品的后期，它可能是除了代码之外，最有价值的产物了）。

【案例1】操作软件+用户手册+代码静态分析，三合一情况下设计测试用例。

背景：我们曾遇到过这种场景，一个在线（正在生产与销售）项目，由于新需求不少，原来的系统也存在不少 Bug，公司决定重新立项来解决这些问题。为了尽量减少成本，相关负责人决定在原来的系统上进行更改，并优化某些模块的代码。然而原项目中基本没留下任何设计文档，包括软件需求说明书、概要设计、详细设计文档，测试用例也没有，软件设计的原班人马也都已离职。

项目要求：需要设计各模块全面的测试用例。

设计过程：当时我们几个测试人员对于此项目来说都是新手，任务下来时不知道该如何下手。后来我们商定以事实为依据（因为是在售软件，软件的功能及一些特性已被用户接受），即运行产品软件，根据软件的实际输入与输出，设计测试用例（实质是补充原系统用例）。写着写着，我们便遇到一些问题，如软件中某个配置参数默认值是什么？可设置的范围又是多少？某项功能的性能指标是多少等。还好，我们找到了产品的使用说明书，这些在手册上都有写明，对于我们来说，犹如雪中送炭。还有一点值得一提，因为看着手册来写用例，又同步在软件上进行验证，在补用例的同时，对手册也进行了一次全方位的确认，最重要的是让我们始终能以用户的使用为核心，把用户的日常操作及一些性能要求，进行了重点设计。在这种情况下的用例设计过程中，我们还碰到一些问题，如在一些非法操作情况下，软件应做出何种响应才算正确？手册中没提到，我们只好直接看代码，从代码中提取出已实现的输入输出。出乎意料的是，软件中的一些输入条件，我们根本就没考虑过，正好可以补充一些测试用例。

小结：俗话说“一分耕耘，一分收获”，通过我们几个测试人员的共同努力，把产品原有功能的测试用例终于补起来了，后来新需求实现，以及一些 Bug 更改的测试用例都在此基础上增加。这样，此项目完整的用例库被我们建立起来。由于测试设计的重要部分被控制好了，整个项目的测试最后顺利地按时完成，且质量可靠，特别让我们欣慰的是此项目在后续的几年内，用户端反馈回来的软件 Bug 几乎没有。想来，在项目结项聚餐时，项目经理颁发给我们测试小组的“质量保证奖”，确实名副其实。另外，我们当初没想到的

是，此用例库在日后居然成了项目人员经常查询实现结果的资料库（由于没有需求文档，时间长了后，不少同事问及项目的功能如何使用时就在用例库中找）。

小贴士：

什么叫测试用例？

在满足特定的环境下，由明确的输入与输出等一系列测试元素组成的集合，是为执行一个测试行为而准备的数据组合。测试用例元素包括但不限于这些元素：测试目的、测试标题、测试环境、输入数据、操作步骤、预期输出。

【案例 2】测试执行与用例设计同步

首先，一边执行测试一边设计用例，是不符合测试流程规范的。但软件开发过程中，又怎么能保证每一次测试都那么顺利呢？常常，这种事情是发生在“救火”的测试场合。例如有一天突然收到用户的投诉，说某某软件存在质量问题，测试需马上重现，或更改后需立即开展测试执行，由于时间紧急，几乎没有时间做用例设计的准备。遭遇这种情况时该如何应对呢？

就像消防队员一样，冷静、镇定是必须的。先理清更改思路，把“冒火”的原因，即把发生问题的原因搞清楚。一个有效的方法就是比较更改前后的代码，改了什么，对代码进行必要的走查，并随时写下测试点。运行软件，写下想到的测试点，然后马上展开测试，随时记下测试结果。执行完成后，再整理详细的测试记录，测试用例也就形成了。概括起来，这种边测试边设计用例的过程如图 7-2 所示。

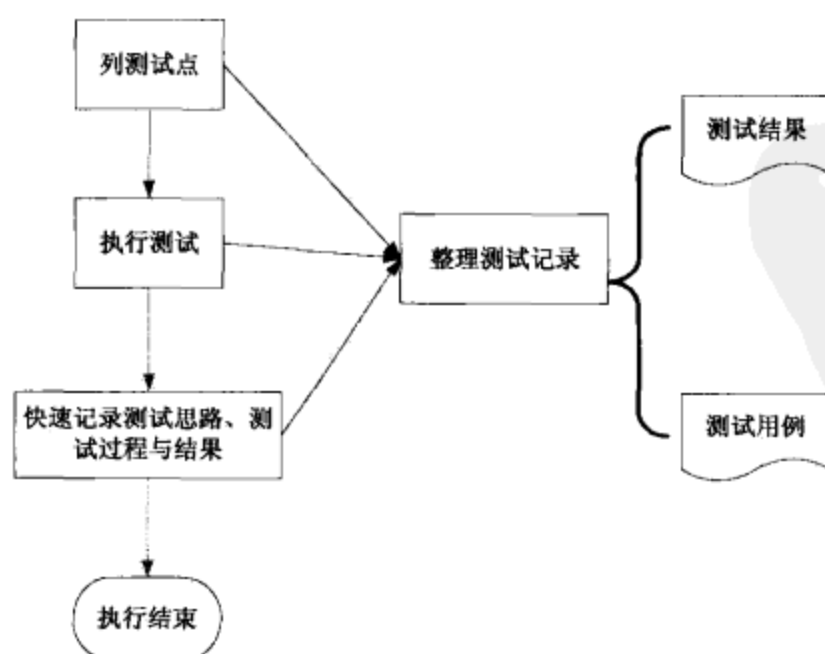


图 7-2 测试执行与用例设计同步示意图

图中的用例设计过程，相当于写下测试点（列出测试大纲）后，先插入了测试执行，然后记录测试思路与测试过程，为执行结束后的记录整理提供数据。测试记录整理出来后，测试用例与测试结果自然就出来了。

这种做法的好处是，用例设计与执行相互交错，缩短前期单纯的测试设计时间，能节省测试过程的时间。这种方法对于更改小的模块测试，还行得通，但对于更改复杂、业务功能多的情况不提倡。这种方法最大的不足也就是因为事先没有足够的准备，可能会遗漏一些测试点。依赖于测试人员对产品业务的熟悉程度及测试经验的丰富情况，存在不可控的因素。

7.3 技术攻关：高效用例设计方法

前面通过案例讲述了用例设计的必要性与重要性，及在实际工作中遇到恶劣情况如何解决用例设计的问题。实际上，无论在顺境或逆境之下，测试用例对于测试来说都是需要设计的，如同开发设计代码。对于用例的高效与否，方法很重要，用例的好坏，将直接影响到软件的质量。常见的功能测试用例设计方法，包括等价类、边界值分析、错误推测、因果图、正交组合法。在具体的项目测试中，这些方法都有用，但能否结合实践场景，灵活应用，即学以致用，才是关键。也只有在达到了学以致用的基础上，通过反反复复的实践、总结，才有可能对这些设计方法做进一步升华、创新，形成独具一格的新型用例设计方法。这正是接下来笔者要向读者朋友们介绍的设计方法，它们都是在长期工作实践过程中总结出来的实用新型用例设计方法。

小贴士：

高效用例：对需求的覆盖率高，不冗余，揭露 Bug 能力强。

7.3.1 隐式边界

在业界，边界值分析（boundary value analysis, BVA）、等价类划分（equivalence class partitioning, ECP）是大家最常用的测试方法（或叫测试工具），已成了测试人员必须掌握的测试基础知识。在一些公司招聘测试人员的笔试中常出现考查这方面知识的题（在网上搜索也可找到不少这方面的应用案例）。我们知道在实际工作中，应用这些方法确实可以设计出具有代表性的高效用例，执行这些用例很容易暴露程序的错误。而当应用这些方法

达到一种娴熟的状态后，你会发现这些方法完全可以扩展，比如边界值分析，测试的对象不一定非要是数值型，它可以是一个业务流程，也可以是一种业务状态，它们一样存在边界问题。这种边界分析，我们叫做隐式边界。下面是一个关于隐式边界的案例。

【案例】

现假定手机的电话本可保存在 SD 卡上，文件名为 telbook.dat。假定软件在设计时有如下定义：当使用 SD 卡保存电话信息时，手机将按系统定义的格式自动创建电话本文件，再次保存电话信息时，将在原文件后增加新内容。电话本信息保存业务实现流程，如图 7-3 所示。

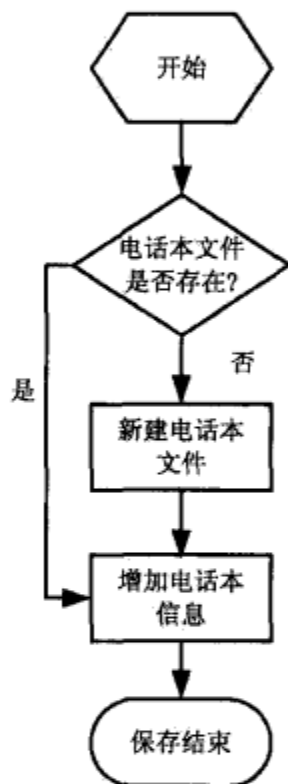


图 7-3 电话本保存设计流程

我们曾在第 5 章的“多管齐下溯需求”小节中提出，设计需求是测试需求的来源之一，在概要设计或详细设计文档中常能看到类似上面的流程图。上面的流程图看起来很简单，但每一个流程节点及节点与节点之间的关系，只要你细心思考就会提取到不少测试需求，正所谓流程图中暗藏玄机。

例如“新建电话本文件”的处理，创建文件有可能会成功也有可能会失败，我们可以创造环境，让软件创建文件失败；当文件存在，增加新录入的电话记录时，首先需打开文件，同样，我们可创造条件，让打开文件失败来验证软件的容错处理；电话信息无论保存成功与失败，从易用性角度出发，都需提示用户。依据这种思路，上面的流程图我们可以进行如图 7-4 所示的细化。

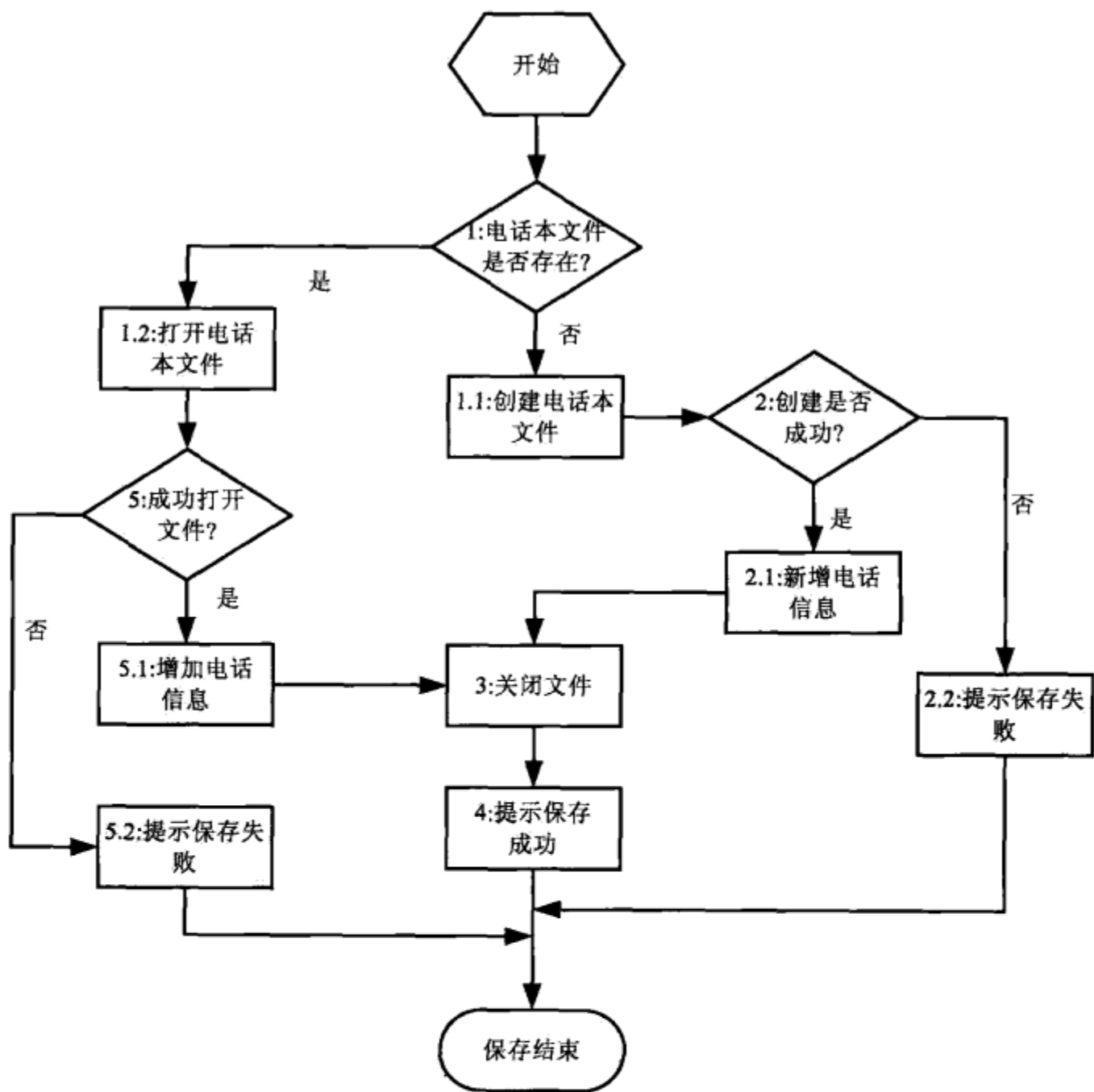


图 7-4 电话本保存详细流程图

从细化的流程图中，按标识的序号可以很清晰地导出如表 7-2 所示的 4 条流程用例。

表 7-2 电话本保存流程用例

用例编号	流程序号	测试条件	操作步骤	预期输出
TC-1	1->1.1->2->2.1->3->4	SD 卡良好，并已插入手机的卡槽中	新增电话记录，并选择保存在 SD 卡中	新增电话保存成功（提示保存成功，电话本中有该条电话记录）
TC-2	1->1.1->2->2.2	插入坏的 SD 卡（能识别）	同上	提示保存失败，电话本中没有该条电话记录
TC-3	1->1.2->5->5.1->3->4	执行 TC-1 后	同上	同 TC-1
TC-4	1->1.2->5->5.2	电话本文件被破坏	同上	同 TC-2

流程图中“电话本文件是否存在”是流程的起点，存在隐式边界，包括如下情况：

- 电话本文件存在，但名称被修改。
- 电话本文件被删除了。
- 电话本文件存在，但电话本内容被破坏。

从流程图上来看，TC-1 到 TC-4 用例已覆盖了所有的流程分支，上面的第 1 与第 2 种情况按等价类来划分，属于 TC-1 同类（认为文件不存在了），但实际结果是否一样呢？如果文件被改名后，浏览手机中的电话本，查询界面，查询情况等与本文件有关的功能都可能会受影响。第 3 种情况，可能会有两种结果，一种认为文件被破坏，还有一种情况可能认为文件仍正常（与程序设计对文件一致性的判断方法有关）。这些情况就像存在具体数字的边界值输入一样，往往在对一些边界情况的处理上程序会出问题。如表 7-3 所示为根据隐式边界设计的用例。

表 7-3 电话本文件隐式边界用例

用例编号	测试条件	操作步骤	预期输出
TC-5	电话本文件名被重命名	浏览电话本	操作的输出失败
TC-6		电话本中查找原已保存的电话	同上
TC-7		朋友打电话进来	同上
TC-8	电话本文件被删除了	重复 TC-5 到 TC-7 的操作步骤	同上
TC-9	电话本文件被破坏	浏览电话本	提示数据出错（但有可能为没有受到破坏的数据能正常显示,受到破坏的数据显示乱码）

隐式边界，一方面指这种边界比较隐蔽，需结合需求、系统设计及测试经验综合考虑，一旦发现问题，常是严重级别的 Bug。另一方面，用此法发现的 Bug 通常影响较大，或修改成本高，不会全部得到解决。下面关于软件容错性测试的故事便是一个例证。

【小故事】容错性测试的小故事

一天，测试文件管理模块的 JoJo 发了一个即时消息给她的主管 Jane，要 Jane 尽快去缺陷库上处理一个开发置为“不解决”的宕机 Bug。此 Bug 是 JoJo 把系统文件目录中的其中一个日志文件改了名，然后进入日志模块界面查看日志时，软件宕机了。负责此模块的开发人员评论此 Bug 不改的原因是“终端用户绝对不可能如此操作软件，建议测试人员不要用这种破坏软件运行环境的无意义的手段测试软件”。Jane 追加的评论是“对于终端用户来说，确实不可能出现这种改文件名的场景，但从软件容错能力的角度出发，会出现宕机，说明设计上确实存在不足，建议提交项目级评审”。最后，经项目相关专家评审，开发分析清楚原因后，由于更改代价大，此 Bug 置为延期解决。

关于这件事,后来听说因为 JoJo 提交了一个这样的 Bug,遭遇了一些开发人员的鄙视,曾郁闷了几天。在工作中,无论是开发人员还是测试人员,在 Bug 上面存在一些分歧,是很正常的,也是常出现的。

小贴士:

有意识地制造恶劣的软件运行环境,验证软件是否存在合适的处理,是判断软件健壮性的重要手段。

7.3.2 分类法

本节讲述的分类法与测试朋友都熟悉的等价类划分方法有相似之处,但又有不同点,分类法关注的是某一测试项的不同类域,抽象出类域后,再进行层层分解,直到设计出具体的用例。而等价类划分关注的是同一类域中的数据,对同一类域中具有同等特点的数据进行提取。

分类法的基本思想:首先将软件的输入输出数据空间进行抽象的划分,产生不同的类域,然后在不同的类域中再逐步求精,细分出属于同类域的不同子域。需要注意的是我们划分子域的原则是属于同一子域内的测试数据应是等价的,即它们对于同一程序揭示错误的的能力应是等效的。因此,最后我们只需从每一子域中选取少数的有代表性的数据作为测试数据,也只有这样,才不会造成人力、时间和金钱上的浪费。

现在设定某一程序的输入有效数据空间是一个圆,如图 7-5 所示。

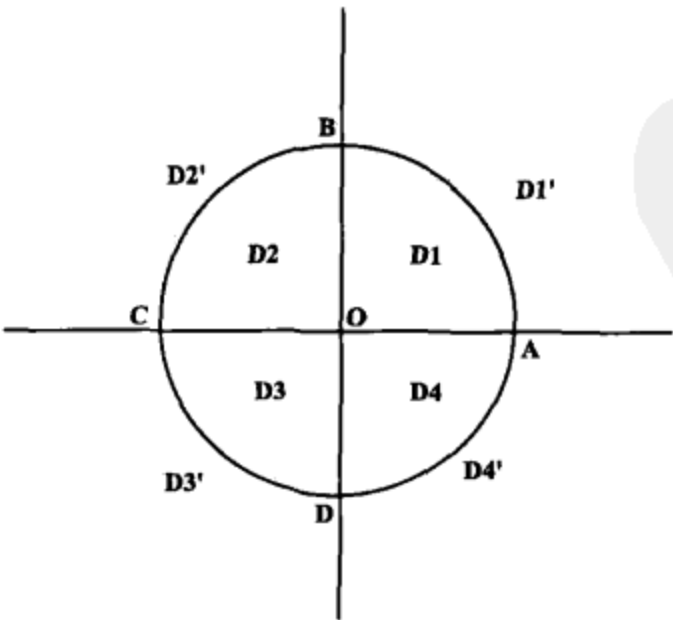


图 7-5 输入有效数据空间示意图

根据分类法的思想，可进行如下分析：

- 圆内的数据为有效值类域，圆外的数据属于无效值类域。其中 D1、D2、D3、D4 表示有效值类域的不同子域，D1'、D'、D3'、D4' 表示无效值类域的不同子域。
- 直线 OA、OB、OC、OD 上的值是属于不同子域的有效值，但它们是划分不同子域的分界点，在细分时应作为特殊值突出产生对应的测试数据。
- 弧边 AB、BC、CD、DA 是属于有效值与无效值的分界点，应作为边界值来考虑其测试数据。

分类法的设计原则：从上面对数据空间的分类分析中可知，一般情况下，在测试一个程序时，对其输入输出数据的有效无效值、特殊值、边界值的测试是我们必须考虑的。在实际工作中，因为不同的应用程序实现的功能各不相同，它们对应的这些值也各不相同。要对它们各自进行准确、合理的划分，找出最具代表性的测试数据，一方面要熟悉各应用程序的规格要求和功能运转情况；另一方面要注意积累经验。下面以手机中常见的应用程序为例，分别对这 3 类值的设计原则进行介绍。

1. 有效无效值

如果规定了输入值的范围，则可划分出一个有效值类域，两个无效值类域，如手机中的系统日期设置，若规定年的有效范围是[1901, 2099]，则有效值类域是 $1901 \leq X \leq 2099$ ，无效值类域是 $X < 1901$ 和 $X > 2099$ (X 表示年份)。

如果输入条件只允许输入数值型数据，如手机上的“计算器”，只涉及数值的运算，则其有效值类域为正整数、小数、负整数；无效值类域为无意义的数值，如-0、-0.00 等类似数据的集合。

如果规定了输入数据必须遵守的规则，则有效值类域是那些符合规则的输入数据。相反，从不同角度违反规则的则是无效值类域中的数据，如计算器处理某一运算时，要求输入一个操作数，接着输入一个操作符，然后再输入一个操作数，最后输入等号表示取值。据此，可设计出如下测试数据，如类似“ $2 + 3 =$ ”、“ $100 \div 9 =$ ”、“ $3 - 8 \times 2 =$ ”等的操作是符合规则的，属于有效值类域的数据。但类似“ $++ =$ ”、“ $3 + \% =$ ”、“ $MC + =$ ”的操作是不符合规则的，是无效值类域中的数据。

2. 特殊值

特殊值也属于有效值类域中的数据，但这些值比较典型，程序员在设计代码时，往往因为注重在正常情况如何实现程序功能的设计，而忽略了某些特殊情况的处理。下面是一些具体应用时的举例。

- 如果计算时有涉及除法的，则应考虑零不能作为被除数。
- 如果手机中的电话本有查找功能，则应考虑其特殊有效值，如输入“空白”、“回车”等特殊值，是否能把所有含“空白”或“回车”的记录都找到。

3. 边界值

实践证明，输入输出数据的边界值最能发现程序的错误。所谓边界值，是指处于有效无效值分界点的值，在选取测试数据时，我们应选那些刚刚大于、刚刚等于或刚刚小于有效值的数据。

手机中的应用程序，如常用的电话本、短消息等。用户输入所需的资料保存后再查看时，最后几个字符是否丢失，这也是一种基于边界值极限的测试。

假定电话本中姓名最多可以输入 20 个字符，可尝试输入 19 个字符、21 个字符，且输入的字符最好是汉字与英文的组合。

【分类法的测试实例】

在手机的系统设置中，系统日期及时间是可由用户自行设定的，但规格说明定义有效设置范围为：1971 年 1 月 1 日 0 时 0 分到 2035 年 12 月 31 日 23 时 59 分 59 秒，小时可用 12 小时制和 24 小时制两种方式表示。

按前述分类法的思想，首先我们依据题意及要求进行分析，系统日期和时间的设置涉及年月日时分秒，除注意规格的规定外，我们还需结合日常的实际情况。它们应只能接受规定范围的数值，其中年有平年和闰年之分，月有大月与小月之别。如图 7-6 所示为初步分析后得出的关于年月日时分秒各自对应的值域示意图。

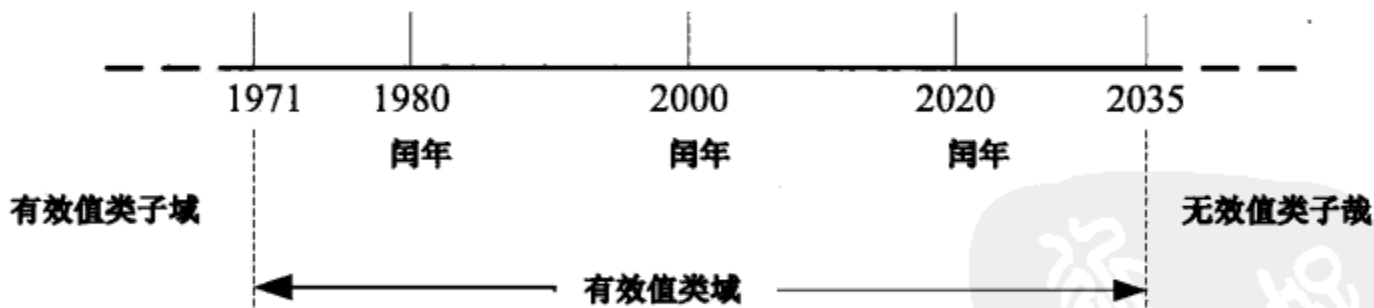


图 7-6 年份值域数轴图

我们可清晰地分清年月日各种情况下各自对应的有效无效值、特殊值和边界值，概括如下。

对于日期设置，从如图 7-6 所示的数轴上可看到其有效值域是 $1971 < X < 2035$ ，无效值域是 $X < 1971$ 和 $X > 2035$ (X 表示年)。因为年在实际中有平年与闰年之分（闰年的判定方法：能被 4 整除但不能被 100 整除，或者能被 400 整除的数，如 1988 年、2000 年、2020

年都是闰年)，在划分有效子域时，可把闰年作为一分界点。闰年对应的 2 月是 29 天，而平年 2 月是 28 天，所以在做年的特殊值闰年测试时，须与月日结合起来才会有实际意义。至于无效值子域的划分，如果程序按规格已做到了 1971 年到 2035 年的限制，则没必要再细分其子域，只要在类域范围内多测试几个数据，以便增强对自己在这方面行为的信心即可。当然，最后别忘了对边界值的测试。

因考虑到年月日各自的有效无效值、特殊值及边界值对应的数据组合情况很多，且必须把它们组合起来才能完成日期的设置。表 7-4 列出了有代表性的年月日测试数据。

表 7-4 年月日测试用例

编号	有效值类		无效值类	
	测试标题	测试数据	测试标题	测试数据
1	年边界值，月日有效值	1971-1-1	日无效	1972/01/32
2	年边界值，月日边界值	1971-12-31	日无效	1972/02/0
3	闰年-特殊年月日	1980-2-29	日无效	1985/07/a
4	年月有效值，日边界值	1994-3-31	日无效	1999/04/会
5	平年 2 月，有效日	1999-2-28	月无效	1999/00/20
6	有效年，月日边界值	2000-4-30	月无效	2000/13/18
7	有效年，月日边界值	2010-5-31	月无效	2012/B/31
8	有效年，月日边界值	2028-6-30	月无效	2030/+10
9	有效年，月日边界值	2030-7-31	年无效	1900-3-31
10	年日边界值，月有效值	2035-8-31	年无效	0000/12/18
11	年日边界值，月有效值	2035-9-30	年日无效	4038/11/38
12	特殊年，日边界值	2020-10-31	年日无效	20+)/13/20
13	特殊年，日边界值	2000-11-30	年月日无效	++++/-*/%

注：假定年月日编辑区可输入任何字符。

对于时间的设置，有 12 小时制和 24 小时制两种情况。在 12 小时制下，时间是 12:00:00am—11:59:59pm，如图 7-7 所示。数轴上表示换天或上下午的临界点，我们在测试时发现常出问题，如 12:00pm，是下午的开始，但程序实现为第二天的开始，第二天凌晨零点 12:00:00am 认为是下午的时间等。

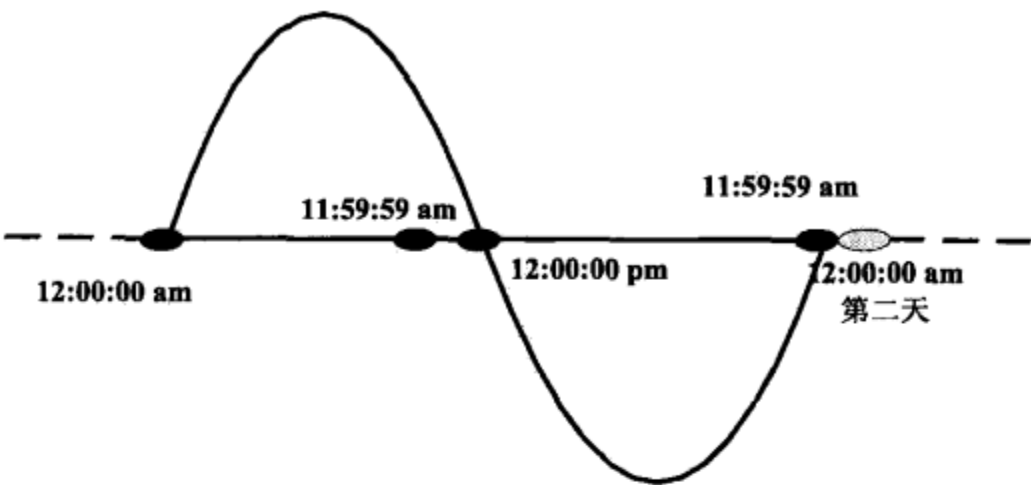


图 7-7 12 小时制时间值域图

根据以上值域图分析，可以总结以下有代表性的测试数据作为测试用例的输入，如表 7-5 所示。

表 7-5 12 小时制时间设置测试数据

编号	有效值类		无效值类	
	测试标题	测试数据	测试标题	测试数据
1	时分秒边界值（特殊值）	12:00:00pm	时无效	13:00:00am
2	时分秒边界值（特殊值）	12:59:59pm	时无效	+ -:12:50am
3	时分秒的跳转（特殊值）	01:00:00am	时无效	00:00:00am
4	有效值类（普通数据）	08:20:30am	分无效	12:60:00am
5	上下午时间跳转分界点（边界值）	11:59:59am	分无效	11:a*:50pm
6	上下午时间跳转分界点（边界值）	12:00:00pm	秒无效	10:12:60am
7	时分秒边界值（下午）	12:59:59pm	秒无效	08:20:/-pm
8	普通有效值类数据（下午）	03:15:50pm	分秒无效	01:60:0k
9	普通有效值类数据（下午）	07:30:30pm	时分秒无效	13:60:60pm
10	特殊值（一天最后一时间点）	11:59:59pm	时分秒无效	（全输空白）

注：假定时间编辑区允许输入数字、英文、符号。

24 小时制时，时间表示是 00:00:00—23:59:59，与 12 小时制相比，24 小时制的表示方法更直观。在我们测试 PDA 系统时间设置功能时，关于这方面的 Bug 也较少，但曾经出现过可设置超过 23:59:59 的时间，如 24:00、25:30 这样的时间。从数轴图上，我们可以清晰地知道，在 24 小时制下，有效时间范围是[00:00:00, 23:59:59]，如图 7-8 所示。

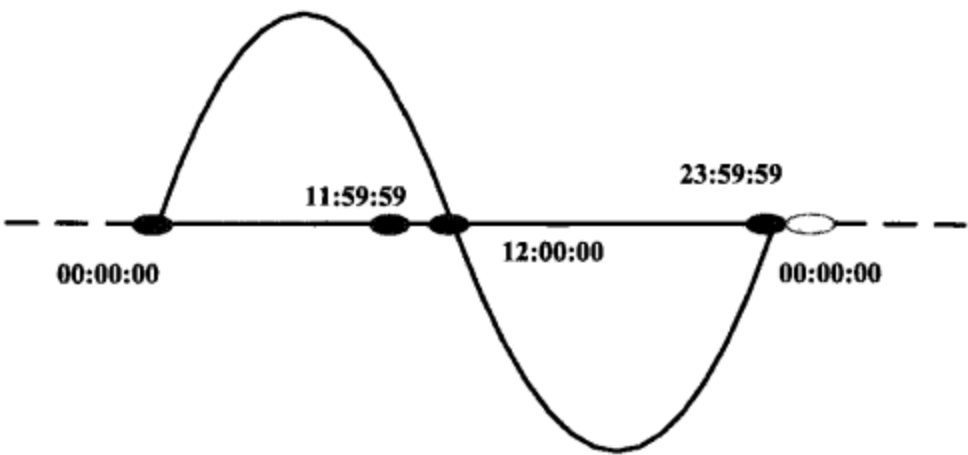


图 7-8 24 小时制时间值域图

根据以上值域图分析，可以总结以下有代表性的测试数据，来测试 24 小时制的时间设置，如表 7-6 所示。

表 7-6 24 小时制时间设置测试数据

编号	有效值类		无效值类	
	测试标题	测试数据	测试标题	测试数据
1	时分秒特殊边界值	0 00 00	小时无效，大于最大值	24 00 00
2	时分秒特殊边界值	0 00 01	小时无效，只允许数值	人 12 30
3	普通有效值类数据	8 30 30	分钟无效，大于最大值	0 375277778
4	上午最后一秒时间点	11 59 59	分钟无效，只允许数值	07 cd 36
5	下午开始时间点	12 00 00	分钟无效，不能空	10 -- 25
6	分秒边界跳转前	12 59 59	秒钟无效，大于最大值	0 844444444
7	分秒边界跳转后	13 00 00	分秒无效	21 #0 60
8	普通有效值数据	20 10 50	小时无效，不能为空	-- 18 20
9	普通有效值数据	22 20 25	时分秒全输空白	-- -- --
10	一天最后一秒时间点	23 59 59	时分秒大于最大值	24 60 60

7.3.3 反常规操作法

反常规操作，指在按规格正常执行某业务功能流程的过程中，突然改变方向（如半路折回、突然中断等情况），中途结束流程，或进入另一业务流程。正如人走路一样，正朝一个方向走的时候，突然去做一些别的事情。反常规操作法与上面所介绍的分类法不一样，分类法是以不同类的测试数据为出发点去找程序的错误，而反常规操作是以不同的操作动作作为出发点揭露程序的错误，但即使是这样，它还是要以某数据为驱动的。反常规操作有

点像动态的白盒测试，因为我们在执行程序的过程中，对应不同的操作步骤正好是白盒测试中的某条路径，所以无形之中我们也在进行着某条路径的测试。反常规操作，也是逆反思维测试方式的一种体现，用这种方法测试往往能出人意料地暴露程序的错误，甚至是系统设计方面的错误。

反常规操作法的测试原则：无论测试何种应用程序及其哪一方面的功能，都可采用基于逆反思维方式的反常规操作法，以验证该程序的功能正确性、易用性、健壮性。利用此法测试时，与分类法一样，首先要熟悉规格要求和程序功能运转情况（方法实际上是一种工具，应用哪一种工具，前提都是一样，需要熟悉需求与程序设计原理，这样才能把工具应用到最佳境地），然后按照上面讲述的思想，以某一数据为驱动，在执行某一功能过程中尽可能地执行不同的动作或不同的操作步骤，有时甚至可做些不合法的操作来验证程序本身的健壮性。如 PDA 与 PC 正在同步通信数据过程中，干脆把通讯线路拔掉以验证通信双方的容错性。

【反常规操作法的应用案例】

背景描述：PDA 中含有具有响铃功能的应用程序，要求用户设定闹铃提醒的时间，并可选择性地输入事件详细内容及提醒方式等，到达设置的时间时，会准时提醒用户做某件事情。此处，只谈论采用反常规操作法对响铃功能进行的影响测试。

根据反常规操作法的思想和原则，可设计如表 7-7 所示的有代表性的测试用例片段。

表 7-7 反常规操作测试用例片段

编 号	测试思路	操作动作描述
1	用户正在使用 PDA，已设定好的响铃事件是否能正常产生	正在编辑资料时，突然发生了响铃事件
2	触控笔触 Touch panel（触摸屏）产生的中断与闹铃产生的中断是否冲突	响铃时间已到，但笔点到屏幕上不离开，超过 1 分钟以上才离笔会怎么样？未超过 1 分钟离笔又是否响铃？（注：PDA 采用 LCD 屏幕，Touch panel 在物理结构上是放在 LCD 的上一层，用户通过笔触 Touch panel 读入软键盘某位置的坐标值进行输入，输出时通过 LCD 显示）
3	闹铃产生，会弹出事件的详细画面，但由于它有中断优先权，会暂时终止其他正在执行的动作，测试当另一画面正在刷新时，是否受响铃画面的影响	正在切换画面时，闹铃产生，离开闹铃，变换画面的动作是否仍持续
4	测试闹铃中断是否对正在写而又未写完的资料有影响	保存资料时，若闹铃产生，离开闹铃后是否仍继续正确的存档完成

(续表)

编 号	测试思路	操作动作描述
5	关机时，系统因提供有备份电池，即使是关机情况下，响铃都应不受影响	设置好一次响铃事件，关机，响铃事件触发时是否能自动开机
6	已设定好的响铃事件，时间上已固定，改变系统时间，只是响铃的快慢受影响。只要系统时间与它的时间一致，还是会产生响铃的	调整系统时间后，闹铃是否正常
7	已设定好的响铃事件，系统已作为一份资料保存起来。即使以后做修改，但还未保存它之前，即使修改它都不应受影响	正在修改下一分钟要闹铃的资料，响铃时间到后，闹铃是否如期响起
8	测试闹铃产生是否对 PDA 与外界通讯产生影响	正在与 PC 进行通讯或采用 IrDA 红外线两部 PDA 进行红外通讯时，闹铃产生，通信与闹铃事件是否正常

7.3.4 倒推法

首先，让我们一起来分享一位小学老师在课堂上讲述如何运用倒推法求解数学应用题的案例。

【案例】

在一堂小学三年级的数学课上，张老师给学生出了一道题：一次数学考试后，小林问小高数学考试得多少分。小高说：“用我得的分数减去 8 加上 10，再除以 7，最后乘以 4，得 56”。老师问：“小高得多少分呢？”

张老师看着一个个可爱的孩子都皱起了眉头，知道他们碰到困难了。于是提示说，这道题如果顺推思考，比较麻烦，很难理出头绪来。大家不妨用倒推法进行分析，就像剥卷心菜一样层层深入，直到解决问题。

几分钟后，老师给学生做了这样的转述：如果把小高的叙述过程编成一道文字题，一个数减去 8，加上 10，再除以 7，乘以 4，结果是 56，那么求这个数是多少？

学生通过分组活动，把未知结果用□来表示，根据题目已知条件可得到这样的等式：

$$[(\square - 8 + 10) \div 7] \times 4 = 56$$

如何求出□中的数呢？老师又引导学生从结果 56 出发倒推回去。因为 56 是乘以 4 后得到的，而乘以 4 之前是 $56 \div 4 = 14$ ，则 14 是除以 7 后得到的，除以 7 之前是 $14 \times 7 = 98$ ，而 98 是加 10 后得到的，加 10 以前是 $98 - 10 = 88$ ，而 88 是减 8 以后得到的，减 8 以前是 $88 + 8 = 96$ 。这样倒推使问题得到解决。

解: $[(\square - 8 + 10)] \div 7 \times 4 = 56$

$(\square - 8 + 10) \div 7 = 56 \div 4 = 14$

$\square - 8 + 10 = 14 \times 7 = 98$

$\square - 8 = 98 - 10 = 88$

$\square = 88 + 8 = 96$

最后,老师还提醒学生注意,用倒推法解题时,从结果出发,逐步向前一步一步推理,在向前推理的过程中,每一步运算都是原来运算的逆运算。这样,同学们便轻松地掌握了运用“倒推法”来解题的方法。

读者也许会问“这与测试用例的设计有什么关系吗?”问得很好。

是的,倒推法,我们并不陌生。或许你我也曾学过像上面案例中的方法,并用它来解题。而当我们长大后工作了,在工作中遇到的一些问题,是否也可运用这种方法呢?

倒推法是一种常用的解决问题的思考方法,正如其名,它是从事情发生的结果出发,利用已知的条件一步一步倒着分析、推理,直到解决问题。这种方法远不止可以解决学习上的一些问题,在工作中,就测试朋友关心的用例的设计上亦可大显身手。

有一个事实,业界的测试实践者都很清楚,Bug 是找不完的,或者说找出所有的 Bug 是不现实的(项目有允许的时间与成本控制)。也正基于此事实,无论你的用例设计得多严密,对需求的覆盖率有多高,也会存在某些 Bug,其不是事先设计了用例就能发现的。且有些问题,远远超出用已知需求来设计用例的思维空间范围,让你百思不得其解。这正是我们接下来要介绍的用倒推法来设计用例的来源。

据最佳实践,如图 7-9 所示,约有 10%~20%的 Bug 并不是事先设计好的用例发现的。遇到不是由用例发现的 Bug,需进行分析,并找出原因。主要原因有两个,一是对需求的理解不到位,设计的用例存在错误或不足,可用如图 7-9 所示的“倒推 2”方法进行整改用例。二是超出了已知的用例设计视野,Bug 的发现,正好突现了测试用例的空白区。如图 7-9 所示的“倒推 1”方法,需要完全重新设计新的用例。一旦发现某个出乎意料的 Bug,而事先也并没有这方面的用例时,通常不是补充几条用例就能把问题解决的。对测试人员来说确是一件既悲又喜的事,悲是因为用例片段性或某一区域代表性用例的缺失;喜是因为终于有幸发现了这些深藏不露的 Bug,突破了既有的测试空间,为填补测试的盲区提供了方向性的指引。

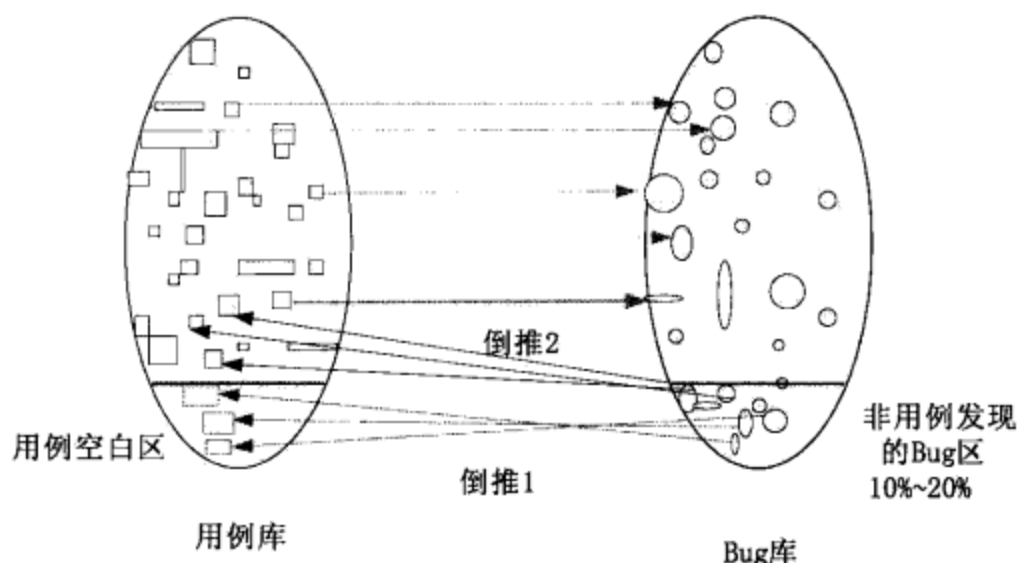


图 7-9 用例与 Bug 对应关系

下面就是一个典型的例子。

【案例】Bug 会自己出来吗？

问题：PC 上的仪器监控软件正在运行，中午时分，测试人员都离开自己的位置享受工作餐及午休了。当下午上班时，PC 的主人小 A 来后，发现 PC 上原正常运行的软件现在不工作了。软件会自己停止工作，出现宕机？

报告问题：小 A 把问题的现象如实向开发人员报告，开发人员于是开始着手分析这个问题，并要求小 A 重现此问题。

重现问题：小 A 重启软件后，按原来的路径重新操作，并放置不动，让 PC 休息。但问题并没有像想的那样如期出来。

分析结果：开发人员分析了两天后，发现原来是一个消息阻塞所带来的死循环问题。由于当时监控软件正在启动后台数据备份线程，而此时正好软件定时休眠的时间也到了，需做黑屏处理。自动备份的提示进度正在显示，且不允许被其他消息打断，而黑屏处理由于优先级高，打断了备份线程的工作，于是出现退出黑屏后，软件宕机的现象。实际上是备份线程一直在等备份结束的消息，却始终没等到的缘故。

知道了问题的原因后，小 A 很快重现了该问题。任何 Bug 都有它的必然路径，只是这个必然性的概率有高低之分。

案例中提到的正是用倒推法设计用例的精髓，即从已有的 Bug 分析着手，找出原因，补充设计用例。

此法应用时的注意事项：

- 用这种方法设计用例时，一定要搞清楚问题发生的根源，思路清楚。否则补充的用例会让他觉得空穴来风，摸不着头脑，为什么要这样操作，而不是那样？因

为这种方法补充的用例大部分情况并不是在需求中能直接找到依据的。

- 正常情况下，大部分 Bug 都是由事先设计好的用例发现的，对于少部分不是由用例发现的，用倒推法分析后，可补充设计用例。
- 在版本发布前，把缺陷库中曾经出现的 Bug 全部回归一遍（如果量非常大，需做分类考虑，如严重级别以上的 Bug 才回归），是一个不错的做法。据在多个项目中实践者言，全部回归每次都有收获，有重打开的 Bug（即曾经在某个内部版本上解决了，但后来在不断的版本更改中，此 Bug 又复生了，重打开率与项目整体质量稳定度有关），还有回归 Bug 时发现的新问题，主要是能解决周边不彻底的问题。

用例设计的方法很多，前面介绍的隐式边界、分类法、反常规操作法都是从正向思维的角度出发，即依据需求进行用例的设计。而倒推法刚好相反，是从发现的 Bug 出发，倒推用例的设计，是明显的以输出为起点的用例设计方法。它的依据是 Bug，通过倒推分析，最后确定是需求定义还是软件设计的缺失，再或者是其他什么原因，最后得出测试该如何进行防范的措施，从而设计出对应的具体用例。

倒推法分析的结果，不仅可以改进测试用例的设计思路，由于 Bug 的产生。与需求定义及软件的设计有直接的关系，所以对改进整个开发流程也有积极意义。

7.3.5 用例设计的综合策略

日常工作可用到的用例设计方法，除本章介绍的外，还包括一些基础性的设计方法，如边界值分析、等价类划分、错误推测法、因果图法、正交实验法，用户场景法等一系列的方法。林林总总的这些方法，需要相互结合起来应用，以设计出高效的用例。借鉴 Myers 提出的策略，结合笔者最佳实践的总结，有如下的综合策略：

- 客户就是上帝，永远可以用用户场景分析法来设计用例，并且把它放在第一位来考虑。
- 在任何情况下都必须使用边界值分析方法，经验表明用这种方法设计出的测试用例发现程序错误的能力最强。
- 用例并不是越多越好，用等价类划分的方法来减少冗余的用例。
- 任何测试阶段，都可以使用错误推测法追加一些测试用例。
- 对于存在多个条件组合输入的用例，最好采用正交矩阵表部署用例。
- 业务流程复杂的系统，采用业务流程图及状态变迁相结合的方法设计用例，以使

流程中的每个节点都能测试充分。在深度上可以以业务主线为线索，而流程节点之间的状态变迁，以及每一节点的状态，可以成就业务的广度测试。

- 对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度，如果没有达到要求的覆盖标准，应当再补充足够的测试用例（使用这种方法需要代码路径覆盖分析工具加以辅助）。
- 以 Bug 为出发点，用倒推法来补充测试用例，常常可以收到意想不到的结果。这种结果可能不仅仅是发现了更多的 Bug，或补充了更多的用例，还可能改进、创新了测试流程，甚至是影响到需求或开发的设计方法。

7.4 用例有效、无效的正确认识

常听一些测试人员在抱怨“今天执行完成多少条用例，但没有发现一个 Bug，软件真就那么好吗？真怀疑这些用例的有效性”。难道没有发现 Bug 的用例，就不是成功的用例，无效的用例吗？通常，大部分用例执行后与预期是一致的，即证明程序是符合需求的。如果说能发现 Bug 的用例才是有效的用例，那么有效用例未免也太少了。特别是软件开发后期，版本稳定的情况下，能发现 Bug 的用例更加少，这些用例都是无效的吗？显然不是。

接下来就如何评价用例的好坏，概括如下：

- 用例表达清楚，无二义性。当测试设计人员与测试人员不是同一个人时，可以减少理解错误的可能。
- 用例可操作性强。测试中常会遇到这种情况，执行一条用例，需准备大半天的测试环境，或用例太粗，操作描述模棱两可，测试执行人员很难顺利地往下执行。
- 用例的输入与输出明确。一条用例只有一个预期结果，如果一个用例有 n 个不同的预期结果，一方面容易使测试人员遗漏观察点，另一方面会造成测试统计不准确。
- 用例的可维护性好。用例的结构、表达，遵循用例的设计规范，犹如开发的编码规范，共同工作的团队有一个统一的风格，方便相互之间的交流。
- 用例对需求的覆盖率高。这一点很重要，可建立一个需求与用例的追溯表来保证每一需求都有对应的用例与之对应。
- 暴露程序 Bug 的能力强。G.Myers 曾提出“成功的测试是发现了至今为止尚未发现的错误的测试”，而一次成功的测试需要高效的能暴露 Bug 的用例来支撑。但是，是不是用例执行的失败率（发现 Bug 多）越高，用例就越好？反过来，是否

用例执行的通过率越高，软件就越稳定？这并不是仅回答是与否的问题，与软件本身的复杂度、测试环境等因素有关。

下面是好的用例和不好的用例的范例。

【案例】数码相机拍照测试用例

测试用例范例

用例元素：标题、测试思路、预设条件、步骤、预期输出。

1. 存在歧义的用例

- 标题：单拍。
- 预设条件：闪光强制关闭，电池充足。
- 步骤：
 - (1) 镜头对着拍摄对象，按下“快门”键。
 - (2) 按向下翻页键，浏览刚拍的照片。
- 预期输出：可见刚拍下的照片，照片正常。

2. 明确清楚的好用例

- 标题：单张拍照。
- 测试思路：检查从按下“快门”键到拍照结束整个过程的处理是否符合要求。包括拍照指示灯的状态，照片存储过程中屏幕的显示，拍完后照片的正确性检查。
- 预设条件：闪光强制关闭，电池充足。
- 步骤：
 - (1) 用相机镜头瞄准拍摄对象，准备好后，按下“快门”键。
 - (2) 注意听按下快门键后的咔嚓声。
 - (3) 检查拍照过程中，指示灯的状态变化。
 - (4) 照片保存过程中，观察相机屏幕显示的变化。
 - (5) 照片保存的正确性检查。
- 预期输出：

按下“快门”后，1秒内听到两次连续的“咔嚓”声；

拍照过程中，拍照指示灯连续以绿灯闪烁3次；

照片保存过程中，相机屏幕为黑色，并伴随出现一个沙漏型光标在屏幕中间运动，约3秒后屏幕恢复为照相模式；

照片为即见即所得。

注：像素大小、亮度、色彩还原度可导入到 PC 上，借助相关工具进一步分析研究。

7.5 用例的价值

在《微软的软件测试之道》中，阿伦·培智提到“缺陷与测试用例是测试人员的中心世界”，笔者当时看到后，颇有同感，相信很多测试朋友也一样。如果把测试工作的每一项输出当成一个工件，用例集无疑是砝码最重的工件。

对于测试人员来说，用例犹如开发人员设计的代码，从这一点就可知它在测试人员心中的位置。然而用例与代码不同，用例并不属于软件的一部分，而是为软件提供的一种服务。也正因为如此，有不少测试朋友认为用例一旦执行完成，便没有多大的价值。由于用例设计本身及日后的维护需花费很多时间，于是有些人提出没有必要设计详细的测试用例，用测试的 checklist 即可。在用例的使用价值方面，业界人士也存在不同的看法，但以下几个方面所体现的价值是显而易见的。

1. 测试执行者的食粮

如图 7-10 所示，依据测试方案中测试对象分析得到的测试点与测试方法，细化为一条条具体的可执行用例，从而形成规模或大或小的测试用例集。随着软件版本的不断迭代，测试执行的不断循环，用例集也面临着不断地被修改、补充，同时也推动着测试方案的不断更新。从图中可看到，用例集是测试执行过程的中心，是执行人员工作的依据，如果没有这些事先设计好的粒粒食粮，必将陷入巧妇也难为无米之炊的境地！

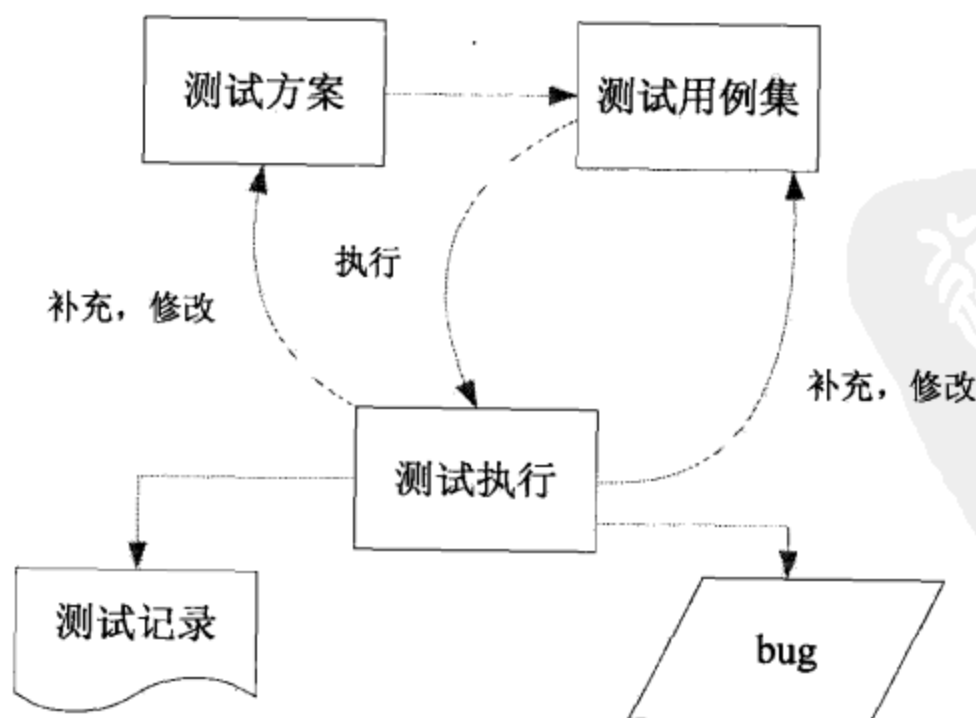


图 7-10 用例集在测试执行阶段的中心位置

2. 使得测试工作可重复，为自动化测试提供了基础

一条合格的用例有明确的输入及预期输出，可操作，不同的人执行结果是一样的。由于软件更改影响的复杂性，有可能在前面版本运行得很正常的软件，几个版本后却不正常了。使得前几个版本测试通过（PASS）的用例，现在为失败了（FAIL），所以用例常常被一遍又一遍地执行。这种情况下，很自然会想到用自动化来执行，这是很好的思路。需要人工反反复复执行的用例，应交给自动化测试来做，此时，业务功能用例为自动化测试提供了基础。另外，由于用例集的存在，可以使测试执行与测试设计工作相互独立，由不同的角色充当，并行前进，对项目的进度与质量都有帮助。

3. 评估需求覆盖率

每一个需求点是否已实现，实现是否符合需求，需要 V&V（验证与确认，Verification and Validation）。验证可以理解为我们研发阶段的测试过程，确认是质量审核的一种手段，可以通过建立一张需求追溯表来进行确认，以表明需求与验证的对应关系。为保护消费者的利益，软件作为产品进入市场销售前，不同行业会有不同的行业认证标准。如在食品药品医疗器械行业，产品在进入美国市场前，必须通过 FDA（FDA: U.S. Food and Drug Administration 美国食品药品监督管理局）的认证。而 FDA 认证要求必须出具需求与测试用例的追溯表，在官方网站（<http://www.fda.gov/>）可查到。需求与用例的索引关系可以用如表 7-8 所示的方式进行追溯，这点要求实际上也是对软件质量的最基本的要求。即使没有行业标准的要求，为确保对需求的覆盖率，测试本身也需要建立这样一张追溯矩阵表，以证明用例对需求覆盖的全面性。

表 7-8 需求与用例追溯表

需求索引 ID	用例索引 ID
SRS-Tel-001	TC-Tel-010, TC-Mem-100
SRS-Memo-001~SRS-Memo-005	TC-Memo-001~TC-Memo-200, TC-Mem-101~TC-Mem-500, TC-Com-1013~TC-Com-2000
SRS-Rec-1000	TC-Rec-001~TC-Rec-1024
...	...

4. 用例的复用

代码复用，在软件界是一个最常用的实用工程方法。测试用例作为软件开发过程的一种资产，同样可以复用，使测试人员从大量的重复设计工作中解放出来，大大提高测试的

效率。用例库是用例复用的一种很好形式，在新项目的测试中可以复用用例库中的用例，加快测试的过程。如何建设可复用的用例，将在 7.6 节与大家一起分享。

5. 提供测试数据，推动开发与测试过程的改进

测试用例描述了测试活动的具体进行过程；用例执行过程中记录了测试软件的软件版本号，记录了通过的、失败的、被阻塞的用例情况。通过这些数据，测试管理人员可及时掌握测试的进展及软件质量的情况。通过分析用例执行产生的这些数据，可以发现测试或开发过程中存在的问题，对解决存在的问题提供了有力的数据支持。总之，通过用例的执行，可以推动开发与测试过程的改进。

7.6 设计可复用的用例

对于不同项目类似功能的业务测试，用例能像代码一样复用吗？关于这个话题，在业界存在颇多争议。

测试用例的设计是测试工作的难点也是重点，用例的好坏直接影响着测试的质量。当做了多个项目后，如果各项目的功能有很大一部分相同或类似，那么可以设计可重用的用例，以减少后续项目设计用例的时间，但不是一件轻松的事。

不同的项目各有其特性，即使完全相同的功能，其操作步骤并不一定相同，有些可能还相差较大，从这点出发，重用用例不能包括详细的操作步骤。用例名称（用例标题）、操作步骤及预期输出可以说是用例的 3 个核心要素，不同的操作步骤与数据会有不同的输出，所以就具体业务功能用例，特别是与 UI 界面密切相关的用例，很难做到完全重用。但集合了一个或多个用例设计思想的测试思路是可以重用的，它就像解决问题的一种方法。有一种软件，它位于系统的中间或底层，如数学运算库、一些特殊算法库、底层硬件驱动程序等。当它们应用在一个新项目上，不做任何更改或只做很小的更改即可移植成功。对应地，我们的测试用例代码也可完全复用或大部分复用。如图 7-11 所示的就是一种公共模块或函数的用例重用场景示意图。

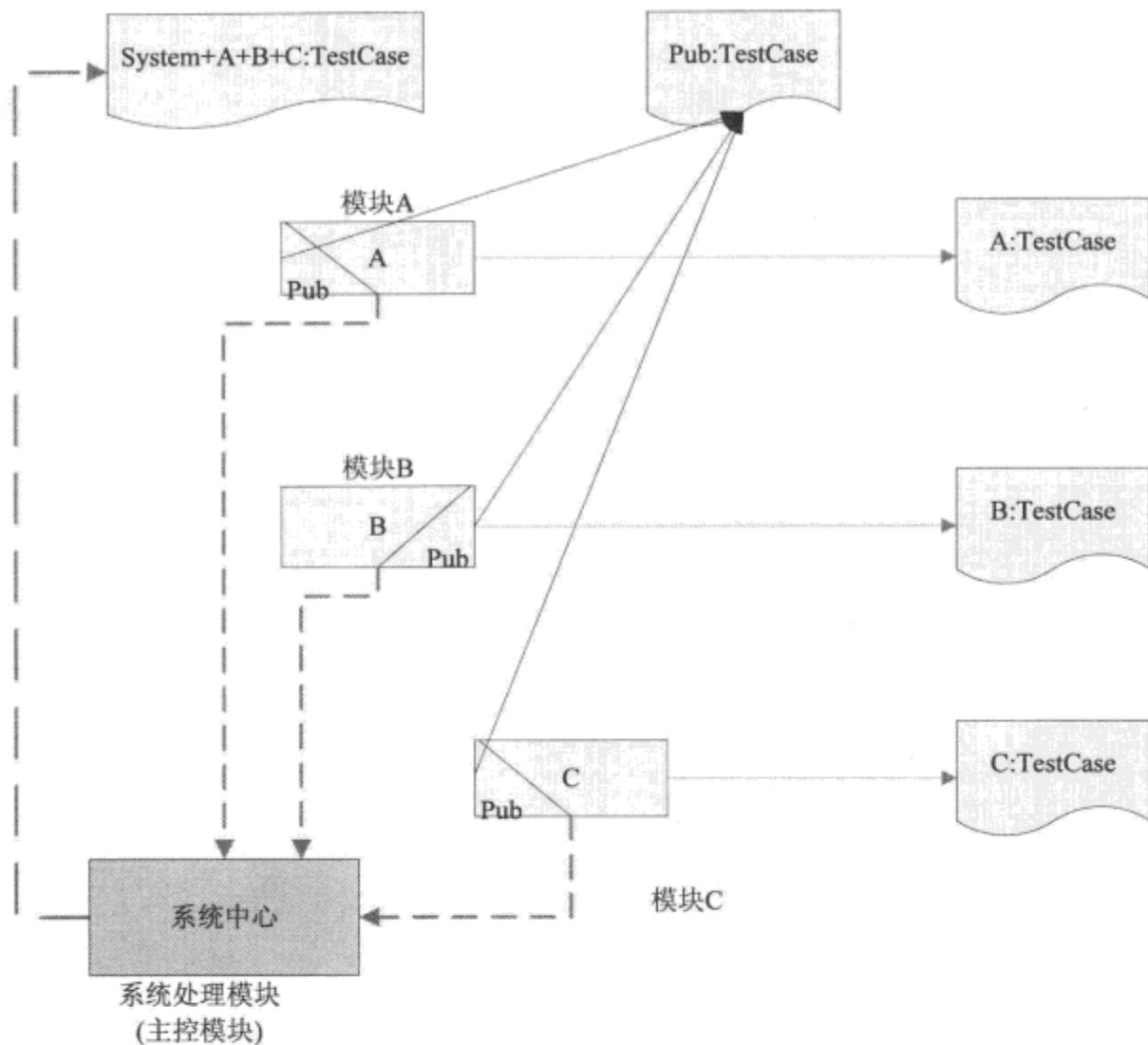


图 7-11 模块用例重用场景示意图

图中，Pub 表示一系列的共用函数，它可能是一个动态库或是一些文件，它们被同一系统中的多个模块调用，还可能被其他项目移植使用。对应地，它们也有可重用的公共测试用例集，如图中的“Pub:TestCase”。用例能复用，隐含着需求、代码都存在复用性，即在需求、代码实现不变的情况下，用例是完全可达到复用的。

接下来，向大家介绍一个用例重用设计思路的例子，希望能帮助读者更好地理解用例的重用性及在实际工作中开展。

【案例】用户登录可重用用例设计

需求背景：某软件有用户登录系统的功能，登录界面如图 7-12 所示。用户名编辑框支持英文、数字、汉字、特殊字符的输入，最大 20 个字符；密码编辑框支持英文、数字、特殊字符的输入，最大 20 个字符；如果输入无效，提示“用户名无效！”或“密码无效！”，两者正确则进入系统。

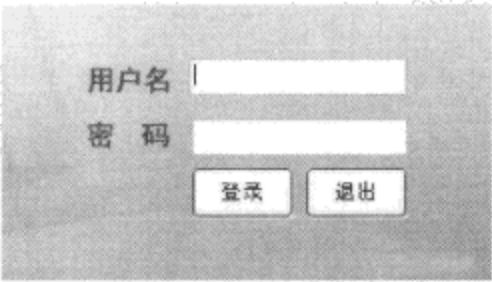


图 7-12 登录界面

根据上面的需求，一般情况，大部分人会采用如表 7-9 所示的方式设计用例。

表 7-9 用户登录测试用例

用例 ID	测试标题	操作步骤描述	预期输出	测试结果
DL-1	用户名正确	输入正确的用户名, 密码为空, 单击“登录”按钮	弹出“密码无效!”提示	
DL-2	密码正确	用户名为空, 输入正确的密码, 但用户名为空, 单击“登录”按钮	弹出“用户名无效!”提示	
DL-3	用户名和密码都正确	输入正确的用户名与密码, 单击“登录”按钮	进入系统	
DL-4	空登录	用户名、密码为空, 单击“登录”按钮	弹出“用户名及密码无效!”提示	
DL-5	用户名/密码编辑	用户名及密码输满由数字、英文、特殊字符组成的混合字符, 单击“登录”按钮	弹出“用户名及密码无效!”	

那么，上表中的用例是否可复用呢？

首先我们分析，根据软件的特点，用例不同点主要是输入数据的不同，而不是操作，上表把测试数据与步骤没有分开，描述性的文字较多，易读性不好。为了达到重用，用例的结构框架很重要，现提取公共的地方。

输入用户名或密码后，单击“登录”按钮，如表 7-10 所示是测试数据具体化后的用例。

表 7-10 具有代表性输入数据的测试用例

ID	测试标题	用户名	密码	预期输出	测试结果
DL-1	用户名与密码不匹配	John	空格	密码无效!	
DL-2		空格	0home	用户名无效!	
DL-3	用户名与密码匹配	John	0home	进入系统	
DL-4	无效用户名与密码	空	空	“用户或密码无效”	
DL-5	用户名/密码编辑	12345670ABDcf e#\$%&*dd	1*&!l!@#12@#DeE fer0121245	“用户或密码无效”	

同样的需求，用例设计采用的表现形式不同，易理解程度就不同，后者明显比前者简洁，突出了输入数据，要是遗漏很容易发现。用例的预期输出是明确且唯一的，可重用，也很容易转化为自动化测试用例。但也因为太具体，明确的数据定义限制了其他数据的输入，如果测试数据不具有代表性或不全面，会造成测试遗漏。

针对输入数据过于局限的问题，有一种改进的思路，就是扩展出专门的用例数据池，即使是同一等价类的数据，也可存储多组输入数据，由测试执行时选择。

7.7 用例重构

在测试工作中，用例设计者与用例执行者不是同一个人是常有的事，如在交叉测试阶段，彼此需交换用例集来执行测试。另外，有些公司测试设计与测试执行在岗位上就是分开的。在这些情况下，用例的可读性、可操作性、预期是否唯一，将直接影响着执行人员的工作效率。一个经常听到的声音就是“用例难看懂，测试执行过程中常要与设计人员反复确认，沟通成本太高”。更糟糕的是，遇到用例设计者已离职，连沟通清楚的机会都没有。于是，有的人凭着经验揣摩用例想表达的意思，有的人则“翻箱倒柜”，以找出相关需求进行确认，有时甚至需求找到了，但还是不清楚设计者的用意是什么。

有一句话说得好“这个世界唯一不变的就是变化”，当软件版本上线后，随着用户的深入使用，市场需求的变化，对软件进行在线升级是再正常不过的事了。当需求变化后，测试需要在原来用例集的基础上增加或更改用例。此时，问题又出来了，不同的设计人员，思维不一样，用例设计的组织框架也就不一样。有些用例还因为年代久远，当初可能合适的用例组织结构，现在已不合适了。如果硬要往里面增加，稍夸张一点说就是“明知下面有火坑，还要强迫自己往下跳”，所以重构的念头异常强烈。

谈到重构，在软件工程学里，是指代码的重构，关于用例的重构，还鲜为人所提及（可能与软件测试行业刚兴起有关）。代码重构通常是指在不改变代码的外部行为情况下而修改源代码，以增加可读性或者简化结构而不影响输出结果。较之于前面介绍的用例存在的问题，以及重构用例想解决的问题，笔者认为它们之间存在异曲同工之处，重构的本质是一样的，仅是表现形式不同而已。下面是一个功能测试用例重构的例子。

【案例】

原需求：排队自动叫号管理软件（银行、医院排队挂号时常用到），起始编号默认从1开始，用的过程中可以随时复位为1，也可以设置为从其他编号开始。前一编号的服务完成后，服务人员按“下一位”按钮后，软件将自动进行加1并叫号。编号最大可输入10

位数字，当抵达最大位数后，将自动回到 1，编号为 0 无效。

根据此需求，有测试工程师 A 设计了如表 7-11 所示的原需求对应用例。

表 7-11 原需求对应用例

用例编号	操作步骤描述	预期输出	备 注
...	
TC-No-101	新安装软件，检查当前编号	当前编号默认为 1	
TC-No-102	输入数字 9，单击“下一位”按钮	叫号 10	
TC-No-103	输入数字 99，单击“下一位”按钮	叫号 100	
TC-No-104	输入数字 999，单击“下一位”按钮	叫号 1000	
TC-No-105	输入数字 9999，单击“下一位”按钮	叫号 10000	
TC-No-106	输入数字 99999，单击“下一位”按钮	叫号 100000	
TC-No-107	输入数字 999999，单击“下一位”按钮	叫号 1000000	
TC-No-108	输入数字 9999999，单击“下一位”按钮	叫号 10000000	
TC-No-109	输入数字 99999999，单击“下一位”按钮	叫号 100000000	
TC-No-110	输入数字 999999999，单击“下一位”按钮	叫号 1000000000	
TC-No-111	输入数字 9999999999，单击“下一位”按钮	叫号 10000000000	
TC-No-112	输入数字 10000000000，单击“下一位”按钮	叫号 1	
TC-No-113	设置当前编号为非 1，单击“下一位”按钮	在原有编号基础上加 1	
TC-No-114	输入当前编号为 0，单击“保存”按钮	提示“无效编号”	
...	

此软件自销售一年多后，客户提出，他们需要编号能输入字母，以标识他们的不同客户群。于是用户需求转化成可实现需求为“除最后一位外，编号可允许输入任何字符，允许输入的位数扩展到 15 位”。需求变更后的测试任务分派给了测试工程师 B。

于是，B 找到如表 7-11 所示用例，读了用例后，列出以下几点疑问：

- 用例中的编号为什么只输入全是 9 的编号呢？是必须还是冗余？
- 数字的位数较多时，用相同的数字描述，看起来眼睛模糊，如 5 个 9 同时出现时，必须得一个个数，再往下就更不容易看了（表达上存在问题），预期输出的数字表达存在同样问题。
- 没有标明测试点，需要读者猜测。
- 就原需求而言，还可以补充一些必要的检查点，如编号的复位。

最让 B 头痛的是，面对新增的需求，已有的用例结构已不适合现在用例设计的要求，如果在其上面增加，只能是越加越乱。一番挣扎后，B 决定重构用例，新增需求对应的用

例也一起考虑。重构后的用例片段如表 7-12 所示。

表 7-12 重构后的用例

用例编号	测试标题	预设条件	操作步骤描述	预期输出	测试结果
测试思路：以用户使用场景为出发点，验证编号递增功能与自动叫号的正确性，还包括默认值、特殊值、最大最小边界值的处理是否符合需求定义。					
1.测试点：纯数字编号的变化与叫号管理					
TC-N0-1.1	相同位数的递增：1 位	当前编号为 1	单击“下一位”按钮	显示当前编号为“2”，并自动叫此号	
TC-N0-1.2	相同位数的递增：多位	当前编号为多位数，如为三位数的 109	单击“下一位”按钮	显示当前编号为“110”，并自动叫此号	
TC-N0-1.3	不同位数的递增：2 位跳到 3 位		设置当前编号为 99，单击“下一位”按钮	显示当前编号为“100”，并自动叫此号	
TC-N0-1.4	不同位数的设置与递增：3 位跳到 2 位	前一天最后的编号为 3 位数	设置当前编号为 9，并单击“下一位”按钮	显示当前编号为“10”，并自动叫此号	
TC-N0-1.5	刚好小于最大编号		设置当前编号为此最大编号小 1 的数，单击“下一步”按钮	显示最大编号（15 个 9），并自动叫此号（关注读法的正确性）	
TC-N0-1.6	等于最大编号	当前编号为系统允许的最大编号（15 个 9）	单击“下一位”按钮	当前编号显示为 1，并自动叫此号	
TC-N0-1.7	默认值	安装软件后，第一次登录	检查当前编号	当前编号为“1”	
TC-N0-1.8	重启软件后检查当前编号	假设前一天编号已到 568	退出软件，重新登录后检查当前编号	当前编号为昨天的最后编号“568”	
2.测试点：字符+数字编号的变化与叫号管理					
TC-N0-2.1

测试工程师 B 根据原用例中存在的问题，重构了用例，并留下了增加新需求后需设计新用例的接口。

下面介绍一下重构的用例特点：

- 调整了用例结构，增加了“测试标题”与“预设条件”。读者从“测试标题”中便可知每一条用例测试的目的是什么；“预设条件”也清楚地告知测试执行人员在执行用例的“操作步骤”之前需要准备什么，这无疑提高了用例的可操作性。
- 增加了“测试思路”的描述，清晰地说明了用例的设计思路，拉近了用例设计者

与执行人员在用例设计理解上的距离。

- 用例的内容之间有层次之分，分别用不同的“测试点”来标识，方便日后的维护与扩展。

功能测试用例可以重构，当这些用例转化为自动化测试用例后，由于自动化测试用例已参数化，相对而言，更容易重构，从而达到简化高效的目的。

从上面的案例中，我们可以看到，用例重构与代码的重构本质是一样的，就是指在不改变用例原有功能的情况下，为了改善用例的结构，提高清晰性、可读性、扩展性和可重用性而对用例进行的改造。简而言之，重构就是改进已经写好的用例设计。

7.8 用例设计规范诞生

从上一节我们知道，用例重构实际上是不得已而为之的事。重构，是要付出代价的。重构得好，当然物有所值。然而，一方面重构稍有不慎，可能会使某些重要路径的用例丢失，致使某些明显的 Bug 没有被及时发现，给软件的质量带来风险。另一方面，重构总是需要消耗不少人力的，如果对不是自己写的用例重构，很明显这种做法是有问题的，实际工作中也是不允许的。当今的软件测试工作，都是在团队协作的环境下完成的，为了统一大家的用例设计结构上的总体思路，减少让人费解的或含糊不清的词汇出现，需要有一个共同的规范来约束。比如开发的编码规范，它就是用例的设计规范。

用例的设计规范，可根据具体不同行业及公司内部的要求来编写。下面是几个共性的原则性要求。

- 用例编号命名规范化。用例的 ID 在整个产品的用例库是唯一的，如 TC-NO—001，“TC”表示测试用例（TestCase），“No”表示编号管理模块的用例，“001”即具体的编号。这样见到这个编号即可知是哪个模块哪方面的用例。当用例采用工具来管理时，如 TestLink（一种开源的测试用例管理工具），一样也可以实现编号的规范化管理。
- 定义用例的必填元素。包括“用例编号”、“用例标题”、“操作步骤描述”、“预期输出”，其中，“用例编号”可以自动生成，“用例标题”需简洁明了，不超过 20 个字（可根据实际情况而变化）。
- 可操作性强。操作步骤描述清楚，无歧义，不能出现“或”、“如果”、“多个”、“等等”等不确定的词，以避免不同的执行人员测试出不同的结果。
- 预期输出唯一、明确。更具体一些，可规定用例的预期输出不能出现模棱两可的

词汇，如“可能”、“如果”、“大约”等不确定的词语。

- 用例应该编写得少而精。建议越少越好，但是功能用例的覆盖面应该是全部的功能需求。精而少的用例节省执行时间，也便于维护。
- 尽量包括更多的测试内容。比如一些易用性测试、健壮性测试、界面测试，都可以包含在功能测试中，这样做不但可以减少测试次数，更能提高测试效率，同时把相关联的测试用例一起执行，会发现更多的缺陷。
- 采取合理的用例组织结构。如根据模块、子模块、测试项、测试点的思路进行分层，就是一个不错的思路，分层能使用例易读易懂、好扩展、好维护。如图 7-13 所示，就其扩展及维护而言，根据新增加需求所处模块或子模块的位置，可以方便地插入或扩展，而不影响原来用例的位置，如 1.2 子模块用例、1.1.2 测试项用例。

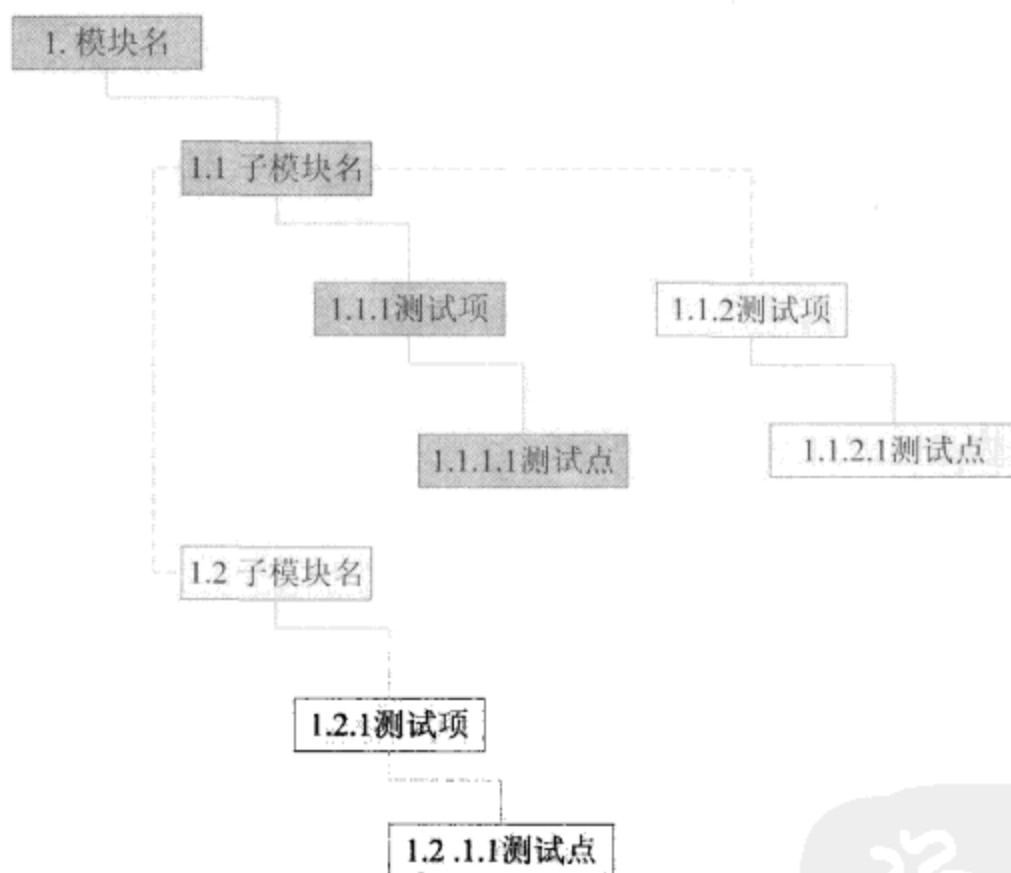


图 7-13 用例组织结构示意图

第 8 章

测试执行流程设计

测试方案与用例的设计，是属于纯测试技术上的设计，但对于整个项目的测试过程，光有技术还不够，需要配合合适的测试流程，策划什么时候做什么事，达到什么要求。好的策划可以对项目的测试起到事半功倍的作用。本章重点讲解测试流程的设计，结合案例，对每个阶段设计的测试环节及其具体应用进行介绍。

8.1 需求测试

第一次看到“需求测试”这个词，是在一本关于软件测试的技术书上，当时觉得很新颖，马上被吸引了。当读了里面的内容后，有一种似曾相识的感觉，原来过去在分析需求的过程中对需求提出的问题，在业界叫 RBT (Requirements-based testing)，是一种基于需求的测试。为方便记忆，以下简称“需求测试”。

需求测试，是一种“黑盒”测试流程策略，主要是通过分析检查系统的需求定义，尽早地修正和验证需求，然后使用各种黑盒测试设计方法来设计最小数目的测试用例，同时满足最大的功能覆盖。这种“基于需求的软件测试方法”由 Richard Bender 于 1977 年创建，并得到不断丰富，这一方法使测试更加有效，更专注于质量问题产生的根源。RBT 曾经成功地应用到几乎所有类型的软件，从 C/S 架构的软件系统到 B/S 架构的软件系统，再从嵌入式系统到超级计算机系统及两者之间的任意类型系统。同时，RBT 也能用来测试硬件的功能，所有的业务关键、任务关键，或者安全关键软件都采用到 RBT 测试流程。因此，RBT 从 20 世纪 90 年代起被中国软件测试工程师使用至今。

软件产品的研发模式无论采用的是迭代开发，还是敏捷开发，或者是其他模式，需求

一样具有变化的特性。所以，对于需求的测试，它是一个过程，而不是一个短期或临时行为。需求的测试与其他的测试阶段，如单元测试、集成测试、系统测试等不同，它主要是在编码设计之前对需求进行的一种检查、分析，涉及的需求可能包括静态文本的描述、图形界面、动态原型需求等。

能有机会对需求进行测试，应该是一件非常幸运的事。一方面，从项目的整体质量上考虑，需求的测试，能把需求的问题消灭在代码实现之前，可以高效节省成本的投入。一项调查（James Martin “An Information Systems Manifesto”, Prentice Hall, 1984）表明 56% 的缺陷其实是在软件需求阶段被引入的，如图 8-1 所示，而这其中的 50% 是由于需求文档编写有问题，不明确、不清晰、不正确导致的，剩下的 50% 是由需求的遗漏导致的。

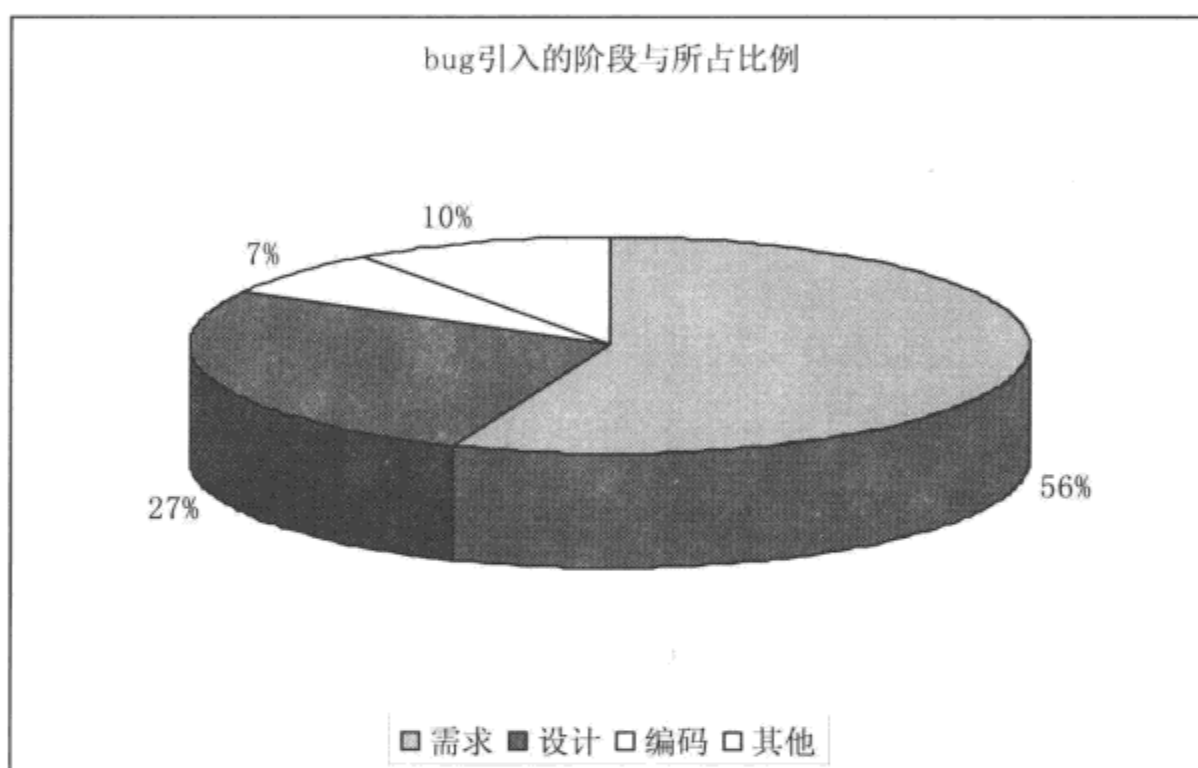


图 8-1 Bug 引入的阶段与所占比例

由此可见，需求测试的重要性与必要性。另一方面，需求是后续开发与测试工作的来源与依据，需求的质量，直接影响着后续的开发与测试工作。在网上常看到一些测试朋友提到“没有需求或需求不全时，如何开展测试工作”的问题，笔者也同样遭遇过类似问题。所以当有一天，能有比较全面的需求文档，让你有机会对需求文档进行测试时，感觉是一件很荣幸的事。

接下来，本章将从需求内审、外审、测试设计这几个方面介绍需求测试的实践方法。介绍方法时将会偏重于实际的应用，把可行的操作方法与读者分享。

8.1.1 需求内审中的测试需求

需求的内审，指同一专业部门内的相关人员参与的审核，如软件需求的内审由软件开发与软件测试，以及同行业中的软件需求工程师参加审核。审核过程，实际上是走读需求的过程，就像看书一样，看的过程中，有不清楚的地方，或需求没考虑到的地方，与需求设计人员进行交流。对需求的理解程度如何，直接影响着走读的效果，如对业务功能与逻辑非常熟悉的专家，很快能提出问题，而对于新手，可能看完了整篇文档，仍是似懂非懂，提不出一个问题来。更有一些有畏惧心理者，不但提不出问题，就连不能理解的需求也不敢提出来。我们曾遇到过走读后的输出形式各异的情况，如把遇到的问题仅与需求人员口头交流；把问题用邮件交流；问题在原文档直接批注者也有之，形式比较多样化。后期发现与需求人员口头交流的需求没有文档化，邮件也有丢失的情况，批注内容并没有得到一一回复，最后对需求的改进、完善方面的作用收效甚微。

针对上面存在的一系列问题，提出改进后的需求内审模式，如图 8-2 所示。

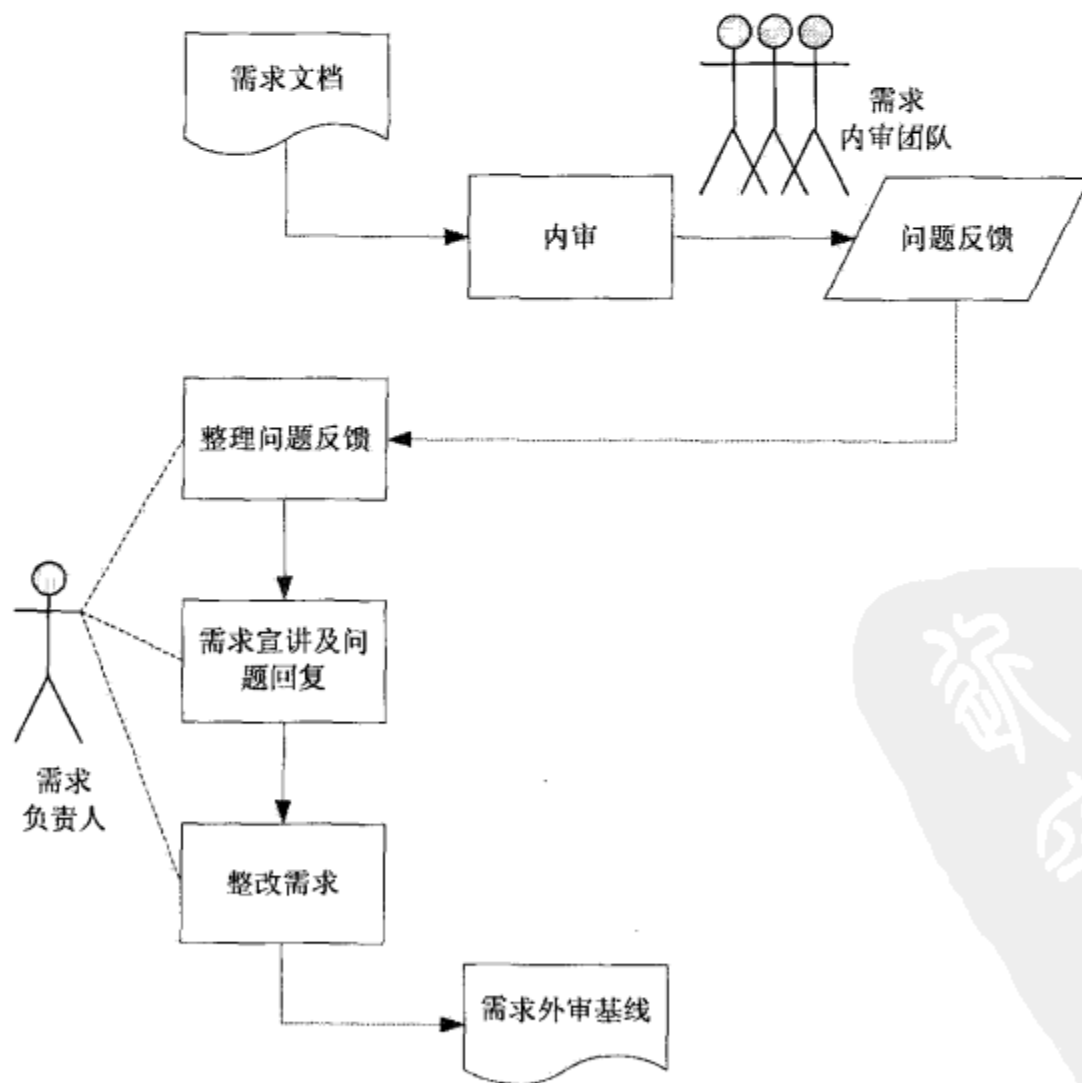


图 8-2 需求内审模式

这种模式下的操作步骤及一些注意事项如下：

- 提出评审开始与结束时间点。评审时间段就评审内容的多少及难度而定，一般不要超过两周，时间越长越没有紧迫感。有明确的时间计划，方便评审专家把评审纳入工作的一部分来做。
- 对评审反馈的问题可酌情提出要求。最好能有量化的数据，如每个参与内审人员必须提问题，根据需求的重要性与复杂度设定反馈的最少问题数。审核是需要时间的，而往往这个同行内审的时间，主管在安排任务时可能并没有列入工作任务，这将影响内审人员的积极性与内审的质量。
- 评审内容在突出重点的基础上求全。参与人员评审的内容进行模块化划分，可由各专业组主管负责安排，以使长文档各部分都有人看，重点模块可多人看。
- 记录需求问题。采用统一的方式，对需求问题进行记录，如直接在原文档进行批注，或者采用某种表格，如表 8-1 所示的记录表就是一种不错的方式，把问题记录下来，作为需求测试的问题单，反馈给需求负责人；或者把问题提交到需求问题管理库（采用工具），由工具进行流程化管理。

表 8-1 需求问题记录表

需求文档名称		版本号					
需求位置	问题描述	提出人	提出时间	解决方案	处理人	处理时间	状态
电话本模块 P10	内置记忆卡，最多可以保存多少条电话本记录？	周杉员	2010-6-1				提交
3.2.1 小节 P18	电话号码除了数字外，是否允许字符，未明确？	阮真	2010-6-8	补充需求：允许支持以下几个字符：+，-，P，W，其他字符不支持	贺子	2010-6-9	已解决
...	...						

- 需求评审宣讲。需求问题反馈后，在较短时间内（最好不要超过一周，时间长了，影响问题的记忆）由需求负责人组织需求宣讲会，并就内审提出的问题一一答复处理措施。
- 需求整改。通过内部评审、整改，推动需求的进一步完善，形成外部评审需求的基线。

简而言之，是借助需求内审的机会，对需求进行测试，重视需求内审测试的每一步骤，及早发现需求的问题，推动需求整改并完善，以形成需求外审的基线。

8.1.2 需求外审中的测试需求

需求外审，可以说是一种广义的需求测试，往往涉及各专业方向专家的评审，发出外审的需求，一般是已通过内审的需求。外审的评审会，常能听到专业外的专家提的一些软件人员较少去想或没想到问题，如软件的可制造性、可维护性、可靠性的需求等。曾经在一次评审数码相机产品需求（很多公司叫 PDRS, Product Requirement Specification）的评审会上，一位机械方面的专家提出，产品需求并没有明确产品是否需要丝印，而丝印会影响机械外壳的开模。还有外观设计的包装专家也提出一些关于外观颜色、开机 logo（商标）等方面的设计问题等。

需求外审的方式方法可参考需求内审的做法，此处不再详述。

需求评审机制，能发现需求存在的不少问题，但由于评审时间有限，考虑的问题会有其局限性。另外，评审后的需求仍可能变化。后来发现，当我们在设计测试方案及测试用例时，提出了更多的在需求评审阶段并未想到的问题，而这些问题的确定影响着测试方案、用例的设计。由于开发人员同期已在做编码的工作，也影响着编码工作的顺利进行。暴露出需求光有评审还不够，而越早发现需求的问题，对项目的质量与进度毋庸置疑都是好事，于是有了“测试设计过程中的测试需求”的必要性，见下一节的介绍。

8.1.3 测试设计过程中的测试需求

前面提到的需求内审与外审，关注的重点在一二级需求，如某些必备功能是否有遗漏，描述前后是否存在冲突、定义错误等。而测试设计（测试方案及用例的设计）过程中的需求测试是针对软件专业方向具体某模块的详细审核与分析，重点在三级需求（假如需求的层级分为三级），如图 8-3 所示。

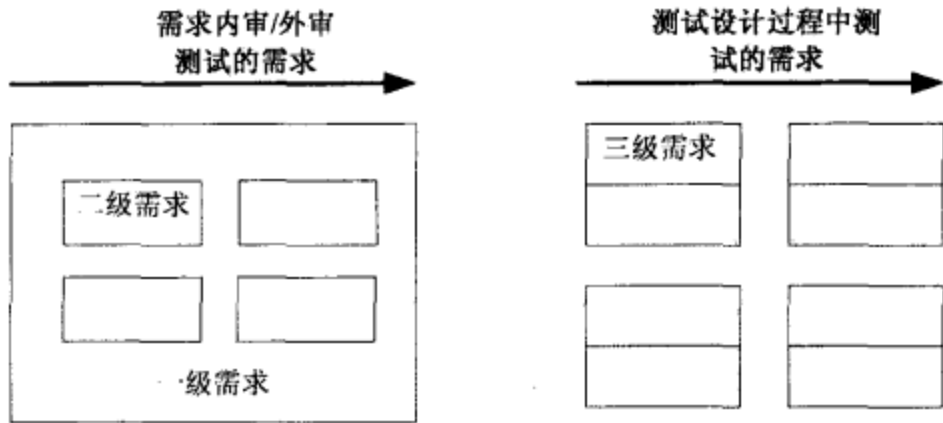


图 8-3 需求测试方法与各自关注的侧重点示意图

需求，是测试方案与测试用例设计的依据。由于测试方案的设计中需从需求中提取测试对象，需对每一点需求进行深入理解与分析，用例设计的预期输出需要需求中有明确的输入与输出定义。因此测试方案与用例的设计过程中常会遇到一些需求不明确或需求没考虑的问题，如需求常能考虑到用户的正常操作，对正常情况下输入输出有定义，而对于边界及异常操作情况，没有定义或不严格。

在测试设计中遇到的需求问题，必须记录下来，以方便查阅与跟踪，表 8-1 是一个示例。当需求文档来不及更新时，表中回复的处理措施将成为准需求，可用于指导后续开发、测试的工作。

8.1.4 需求测试检查点

上面介绍了几种需求测试的方法及其操作方式，接下来就需求测试中常遇到的一些问题，与读者一起分享心得。

1. 易读性

需求的描述，是否真正理解了？包括要制定这个需求的原因（需求的背景），需求的用户使用场景、测试对象的确认，按着这个思路去分析，不清楚的地方随时做记录。不清楚的地方，有两种可能，一是没有理解需求的含意，二是需求没有考虑到。有一个典型例子，需求文档升级了（可能改动了某章节的一小部分内容），但没有任何标识，包括修订记录。需求阅读者不得不从头开始看，特别是对于长文档，太浪费时间。这种情况下，需把问题列出来，要求整改。

2. 二义性

要避免出现二义性的需求。在笔者实践过的项目中，曾出现过同一段需求，开发实现结果是 A，而测试用例的设计输出是 B，最后需求工程师站出来，说原来他想表达的

是 C。

关于这一点，有一幅流行的关于需求信息传递过程失真的漫画，如图 8-4 所示。

图中反映了需求表达无二义性的重要，如果需求源头出现问题，则后续一系列环节的理解将出现偏差，最后做出来的产品与客户真正想要的相差甚远。

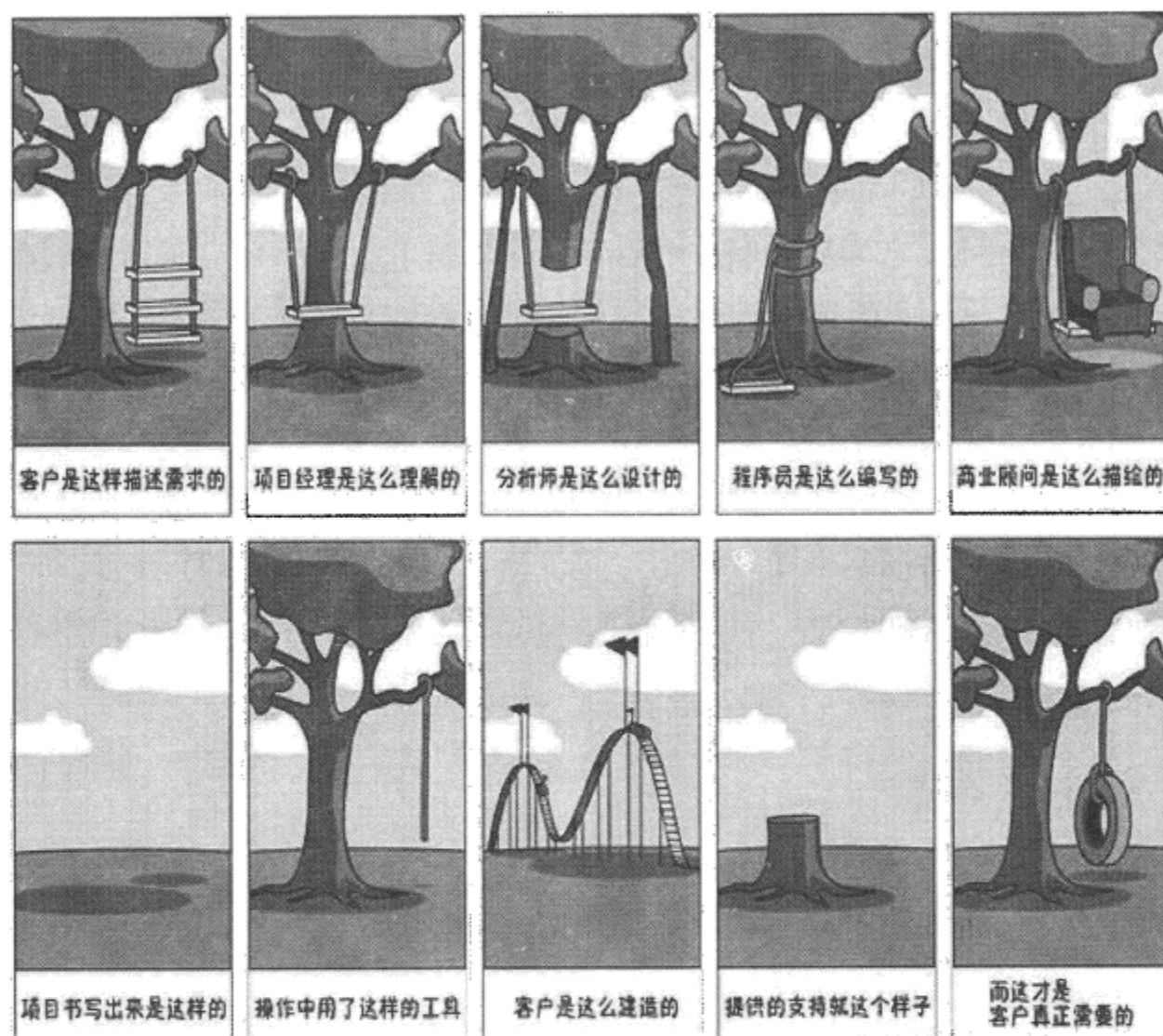


图 8-4 需求信息传递场景图

3. 一致性

指与其他软件需求或高层（产品级，系统级）需求相互矛盾。如某软件总需求中提到某软件需兼容 Windows 2000、Windows XP、Vista Home Basic 操作系统，而在具体的业务模块中却说只支持 XP、Vista Home Basic。这样总需求与模块需求的定义不一致。

4. 统一性

风格统一性，主要指 UI 需求。包括字体、字号、窗口、对话框、提示框的风格。例如，删除用户数据后的处理，不能出现有些没提示，有些有提示。提示框的风格也需统一。对于同样删除数据的提示，不能 A 模块用如图 8-5 所示的提示，而 B 模块用如图 8-6 所示

的提示。尽管它们之间的差别仅是按钮上的文字表达不同，但往往就是这些小问题处理得不够严谨，影响了公司的形象，所谓的“细节决定成败”不无道理。

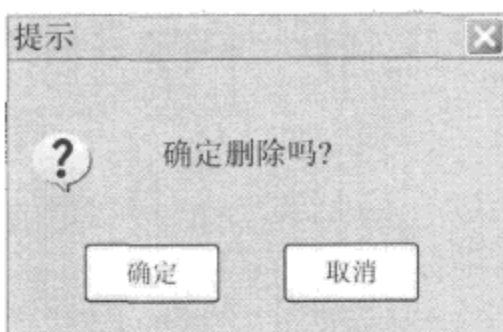


图 8-5 删除对话框 1

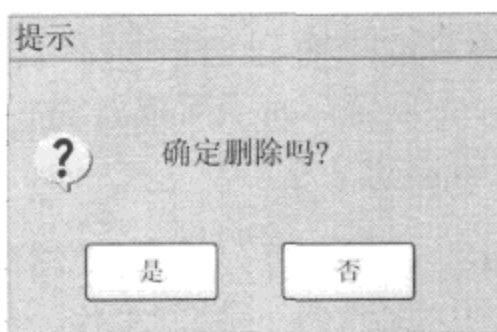


图 8-6 删除对话框 2

5. 是否存在需求过度

需求过度，这里指的是不该在需求中出现的内容却出现了，例如把实现的设计要求写入，下面是一个小案例。

【案例】需求过度

需求定义为“登录窗口，姓名长度最大可输入 32Bytes 的任何字符”。

由于开始时，软件只实现了中文与英文的操作界面，开发在实现时理解为汉字最多能输入 16 个，英文能输入 32 个，输入信息存储时采用的是 GBK 的编码方式。后来由于业务的发展市场要求需支持国际化多语言。于是开发在实现的技术上编码方式由原来 GBK 改为 UTF-8。但 UTF-8 的编码方式是变长的，对于每一种语言能输入的个数不同。在用户使用时，能输入 16 个中文汉字，英文 32 个字符，其他语言，如俄语、法语能输入的字符个数不一定是 16 个，也不一定是 32 个。后来发现是需求表达存在问题，需求实际上想要的是无论什么语言，能输入的姓名最大长度是 32 个字符。多少字节是属于设计范畴内考虑的事，需求的定义应该采用用户能理解的语言进行表达。这个需求点虽然是一句话中一个词的问题，但却误导了开发与测试人员，浪费了前面开发与测试的时间，明显延长了研发周期，提高了成本。

好的开始，是成功的一半。需求是开发与测试的源头，源头理顺了，后续环节必将越走越顺，离成功的软件项目开发也就越近。上面分享的几点实用方法可以应用在实际工作中。工作的过程，实质是一个学习与积累的过程，当对需求的测试积累到一定程度时，可形成一个高效的需求测试指引，如需求测试 checklist 就是一个很好的做法。

8.1.5 需求测试中的几个问题

对需求进行测试，一直到现在，在业界存在一些误区，下面分享我们在需求测试过程中遇到的一些问题，希望对读者朋友有所启迪。

1. 被动接受需求

工作中，很多人都会认为“需求说什么，开发做什么，测试测什么，这是合情合理的事”，需求是后续开发与测试工作的依据，从这一点来说是对的。但就像代码一样，需求也是由人工定义的，即使在前期有用户调研等数据的支持，依然会存在前面提到的二义性、完整性、统一性、正确性、不可测试性等问题。一个项目的成功，离不开团队成员的共同努力，产品的需求从来都不仅是需求工程师的事，需要全体项目团队成员各尽所能，群策群力，站在需求角度考虑需求的正确性、易用性。

2. 需求理解表面化

对需求的理解，是测试设计工作的前提，但就“理解”两个字，不同的人理解结果可能会大相径庭。常出现对需求的理解表面化，即需求说什么的情况就是什么，不知其为什么是这样？这个需求的背景是什么？用户的真正需求是这样吗？是一个竞争卖点，属于锦上添花的功能，还是核心业务？这些问题对需求的真正理解很重要。对于不同定位的功能，测试重点也不同。

3. 需求问题的管理

在整个软件开发过程，需求将会从一个基线迭代到下一个基线，对于每个基线的需求，都要进行需求测试。在需求测试的持续过程中，正常情况下，问题会从多到少、由浅到深、从显式需求到隐式需求，逐步深入发现需求的问题。对需求提出的问题，形成的需求问题列表，不仅帮助了测试人员对需求的深入理解，也支持了需求设计本身的改进，同时促使需求分析人员做更多的市场调研。需求列表，是需求问题管理的一种方法，但是时间长了，表格维护性不好（如 Excel 表格）。后来，我们尝试把需求问题当成 Bug 一样，提交到需求管理库，其流程与 Bug 库很类似。同样由测试人员提交，需求设计人员负责解决，测试人员确认更改后，关闭此需求问题。需求问题的管理，可采用专门的需求管理工具，如 Doors、Rational RequisitePro 等；可以给团队成员一个关于需求的公开视图，方便评论、问题状态跟踪等。

8.2 内部版本发布测试

无论公司的规模如何，只要有独立的软件测试组织，软件版本转测试后，版本发布的接收测试是首先要做的。本节先举个例子，分享在软件项目开发中常见的问题，接下来通过案例展示问题是通过改进开发流程来解决的。然后，通过另一个案例，介绍版本发布的相关重要信息传递给测试人员的方法。案例都是实践的结果，读者朋友，可以直接在实际工作中加以应用。

8.2.1 版本发布恶梦

测试朋友，下面的场景是否似曾相识？

- 某重要功能执行后死机，软件版本被迫重发。
- 开发提交的版本，漏了一个配置文件，某功能不能测试。
- 开发匆忙中更改版本，没有时间做验证，提交到测试手上后，连续多次不能通过版本验收测试。开发人员忙得团团转，而测试又在等版本测试。

下面分享一个具体的案例。

【案例】

某医疗设备公司，DG8800 软件项目最近进入最紧张模块集成阶段，无论是软件开发还是测试，基本天天加班到晚上 10 点或更晚。开发提交的迭代版本，常常是越紧张越出错。下面是一个笔者曾经遭遇过的场景。

背景：按项目计划，今天是软件提交版本给测试的时间。

软件测试：测试人员的准备工作已进入就绪状态，有些工程师沉不住气了，不时问测试项目经理“开发的版本发了吗？”“开发的版本什么时候发？”

测试项目经理来来回回地在开发与测试的办公区走动，探询版本发布的进展。时间很快就过去了，眼看再过 15 分钟就到下班时间了，几个开发工程师正扎在一起讨论着如何解决“界面刷新异常缓慢的问题”。只好通知测试人员“版本暂未提交，今晚不加班”。

软件开发：继续加班解决问题，联调，解决问题，再联调……终于，凌晨 0:30 版本发布，可以测试了。

软件测试：第二天，测试人员上班后，很高兴地看到版本提交了，马上开始测试。然而，半小时后，测试工程师小朱说，进入某模块界面，想保存一条新增记录，点“确定”

后，软件崩溃。这是一条不能跨越的拦路虎，严重影响测试，版本只好被打回。其他测试人员的工作也只好随之暂停。

软件开发：开发高手小李向来定位问题准确，解决问题也很快。于是，不到 1 个小时，第 2 个版本重新提交测试了。

软件测试：测试工作继续，两个小时后，一位测试工程师说软件的配置有问题，有些功能配置了，但实际上没法使用该功能。“真是屋漏偏遭连夜雨！”很遗憾，这样的版本是不合格的，版本只好第 2 次被打回。

问题原因分析：联调时间比预期长，且问题较多，主要原因是大部分开发人员是在单板上进行开发，直到版本即将发布时才进行联调，联调中产生的问题没有在预料之中。更严重的是版本发布前并没有在设备的真实环境下运行，导致运行某模块后程序崩溃没有及时发现，遗留在测试版本中。第 2 次的版本是在真实环境下运行，确认程序能跑起来，但并没有对每个变更点做最基本的功能正确性确认。

解决措施：要求每个开发工程师在提交代码后，必须在用户使用的真实环境下进行联调，并进行冒烟测试(smoking test)，以确认提交的功能能正常使用。冒烟测试的checklist由测试来提供，以提前发现进行版本接收测试时发现的问题。这样的措施，测试人员当然很高兴，因为可以减少测试的时间。事实证明，开发这样做是值得的，在后续的版本中再没出现前面恶梦般的版本，反反复复被打回的现象了。

如图 8-7 所示为冒烟测试（相当于版本发布测试）由测试进行的流程图，后改为由开发进行，如图 8-8 所示。从实施效果来看，是一种通过改进流程来解决问题、提高产品质量的好方法。

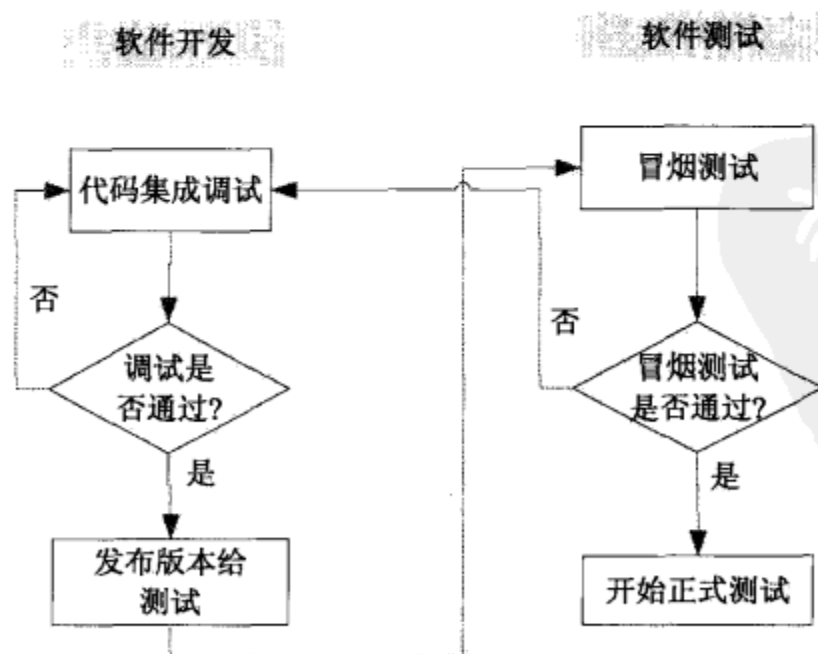


图 8-7 冒烟测试由测试进行的版本发布流程图

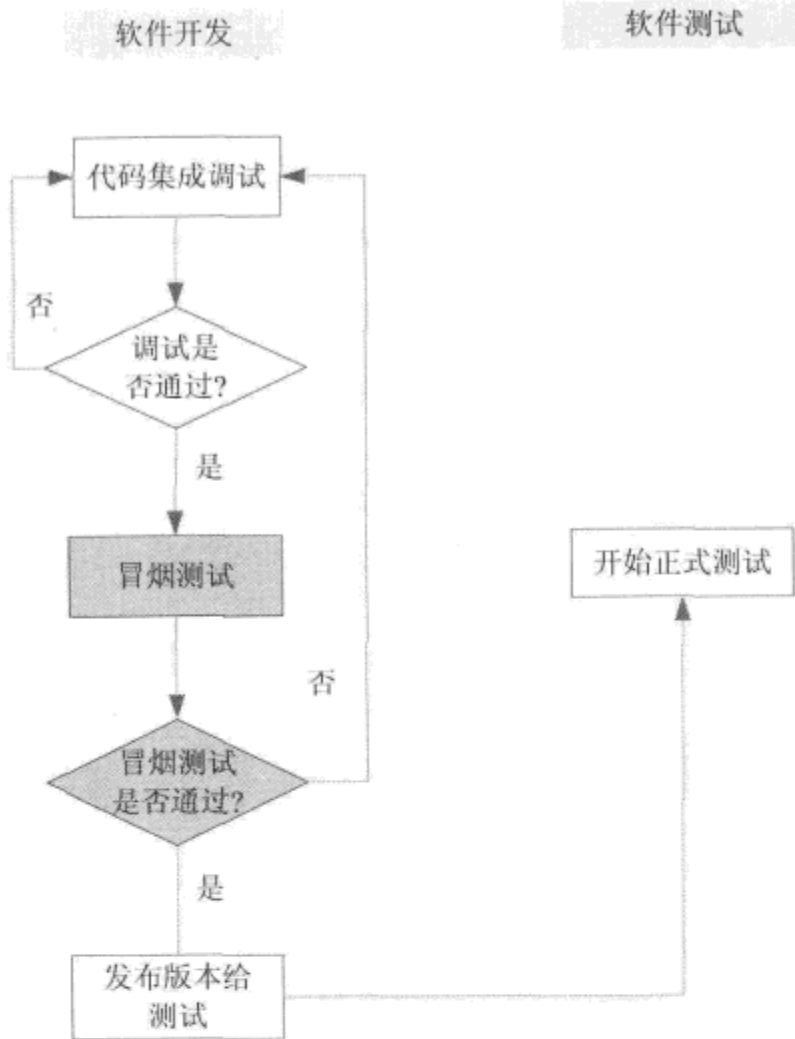


图 8-8 冒烟测试由开发进行的版本发布流程图

8.2.2 小议冒烟测试

冒烟测试 (smoking test)，这一术语应用于不同的行业中，如水管系统、木制乐器的修理、电子工程、软件工程、娱乐业等领域。至于它的最初来源，业界存在不同的争议。

根据工程实践中的应用发展史，笔者倾向于来源硬件行业的说法。在硬件中的“冒烟测试”是指对一个硬件或硬件组件进行更改或修复后，直接给设备加电，如图 8-9 所示为对硬件板卡进行冒烟测试的示意图。如果没有冒烟，则该组件就通过了测试。

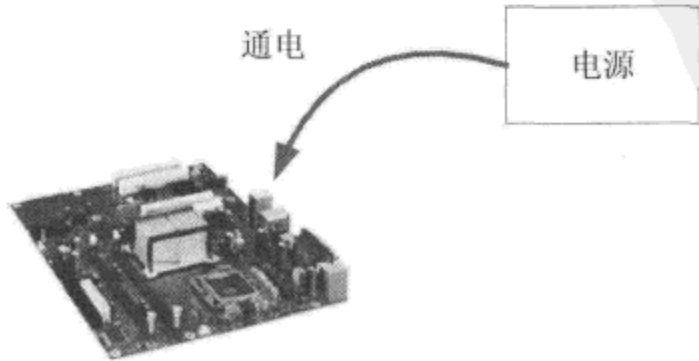


图 8-9 硬件板卡冒烟测试示意图

而在软件工程应用领域，冒烟测试是指在详细、深入的测试之前，对提交测试的软件进行的预测试。这种预测试的主要目的是暴露软件基本功能失效等严重问题，此类问题会导致软件需要重新发布。冒烟测试可以由开发人员执行，也可以由测试人员来执行，即在版本编译后正式提交测试之前由开发人员执行；或开发发布版本后，测试人员在接受这个版本作为正式版本进一步测试前执行。微软的研发团队较早就引入了冒烟测试，他们曾提出在审查了变更的代码后，冒烟测试是确认修复的缺陷及功能变更是否正确的最经济有效的方法。冒烟测试能手动执行，也可以在版本编译后自动化执行，如图 8-10 所示是一个基本原理示意图。

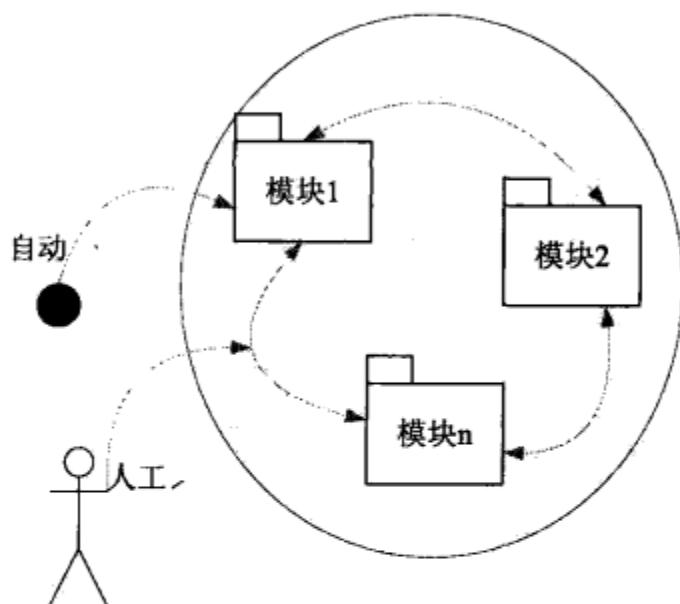


图 8-10 软件冒烟测试示意图

冒烟测试作为一种测试策略，它是对基本功能的确认，非深入测试，但要覆盖到面，即所有的更改点都要进行确认。采用自动化执行时，可以结合每日构建后进行自动化的每日冒烟测试，如果测试通过，则把更改后的代码自动合并到主干代码仓库中，作为正式提交测试的版本。

对于冒烟测试在软件开发过程中的应用，下面是一种可实施的步骤：

- 根据软件的变更，包括新需求的实现、Bug 修复，设计出更改满足预期的功能级 checklist。
- 准备好测试环境。如果软件的运行环境是嵌入式产品，如手机、数码相机、医疗仪器等，需准备好用户使用的真实运行环境。如果是 Windows 平台运行环境，需准备干净的操作系统。
- 执行 checklist，确认基本功能有效，足以支持更进一步的详细、全面测试。

8.2.3 版本发布信息传递

软件版本转测试时，除了软件本身外，另外很重要的内容就是关于版本的变更信息，如相关注意事项、对测试的建议等。对于测试人员来说，这些信息都相当的重要。就像“罗马不是一天建成的”一样，在软件迭代开发过程中，提交测试的版本也是某一个业务模块，或某一个功能点实现后提交的，但有些功能点可能还未来得及全面完成就需提交测试了。在迭代开发模式过程中，可以每日构建进行测试，也可以隔三差五式地发布版本给测试，这种形式可根据实际情况自定。但就每个内部版本变更了哪些内容，测试必须得清楚，以避免无效的测试、遗漏测试，并把握测试的重点。现在很多公司对代码的管理已采用了如 CVS、SVN、VSS 等专门的版本管理工具，程序员在提交代码时可以把变更内容写到版本库的评论中，作为版本的 log 信息工具一直保留，这是一种很好的信息传递方式。但有一些信息在上面得不到反映，如关于版本的匹配关系、测试建议等。这时，测试信息传递的载体“测试传递表”便应运而生了。下面是一个关于测试传递表应用的案例。

【案例】

欧也在面试测试工程师时，常会问应聘者一个同样的问题“你们公司开发提交版本给测试的流程是怎么样的？”

回答 1：直接把编译好的版本通过邮件发给测试相关人员，或放在服务器上的共享文件目录下，通知测试人员去取用。

回答 2：开发把版本提交到配置库上，由专门的配置管理员进行编译、发布。然后通知测试人员去取版本进行测试。

再细问下去，“提交的版本实现了哪些功能，或变更了什么，如更改的 Bug、注意事项等信息，你们是如何获知的？”

回答 3：提交版本要实现哪些功能，是在计划中预定好的，更改的 Bug，则在 Bug 管理系统中 Bug 的状态上能看到。

回答 4：有什么问题随时问开发工程师。

其实，欧也很清楚，他们的回答都是实话，是目前很多公司在开发流程上存在的规范的细节问题。

如“回答 3”所说的情景，流程上看先有计划，测试拿到版本后按计划清单进行测试本应没什么问题。但实际上，开发过程是存在很多变数的，如需求的变更，原计划某版本中要提交的功能，可能因为需求的变化，来不及提交；或某功能点的实现复杂度超出预期或某开发人员离职，而使计划有所偏离；开发修复某一个 Bug 后，但忘了在缺陷管理系统中处理此 Bug，使此 Bug 仍处于待解决状态，测试人员没有回归；如果做的是嵌入式产品

软件，常还涉及硬件写片软件（如单版 FPGA 程序）等配套版本，对于待测试的系统软件存在影响，等等。这些都是欧也在实践过程中常遇到的问题。

规范且有效的流程，是在实践过程中一点点建立起来的，不同的公司在测试不同性质的软件时可能会有所不同。但工作过程中遇到问题后，必须想办法解决才能改进我们的工作。在欧也刚加入 ABX 公司时，测试团队刚建立起来，与开发的接口方面，包括版本的传递都是一张白纸。

有熟悉欧也的同事说，“世上无难事，就怕认真两个字”是欧也对待工作的最合适写照。欧也加入公司后不久，在开发与测试接口的版本信息传递方面与开发达成一致，建立了“测试传递机制”。具体要求如下：

- 提交测试版本前，软件开发项目负责人指定专人负责冒烟测试，并由各代码提交人在软件的真实运行环境下，确认各自负责实现的基本功能在正常情况能否正确运行。
- 版本发布人负责填写测试传递表，并上传到项目配置库中。
- 版本发布人通过邮件或 QQ 正式通知项目组开发及测试相关人员，告知某某版本已发布的消息。

下面是此种测试传递机制下，开发人员填写的测试传递表范例，如表 8-2 所示。

表 8-2 测试传递表范例

第二次版本发布信息传递	
应用软件版本：1.0.0.12	
引导软件版本：1.0.1	
操作系统版本：1.0.2	
数据库版本：1.0.5	
源代码位置：（配置库中的访问路径）， http://192.15.10.1/code	
可执行文件位置：（配置库中的访问路径）， http://192.15.10.1/testver	
提交功能及注意事项	
1、提交通信录的新增、编辑、删除功能。注：超过 1000 条记录保存时有问题，请测试暂不测试此功能点	
2、需求变更实现：查询功能一次最多查找 1000 条记录	
3、修复的 Bug，见 Bug 库状态为“已解决”的 Bug	
测试建议	
1、为便于获取一些严重且偶发 Bug 的现场信息，代码中已打开串口输出宏开关，请测试人员连接好产品与 PC 的串口，用超级终端软件捕捉串口输出的信息	
2、遇死机，请保留现场，马上通知负责的开发工程师分析	
提交人：李 唯	提交时间：2009/12/5

8.3 回归测试

回归测试 (Regression testing) 是软件生命周期中的一个重要组成部分, 是测试过程中的一个必经环节。在业界, 众所周知, 一个软件系统在其开发及维护阶段都将面临不断地变更, 这些变更可以概括为 3 个方面, 一是新需求的驱动, 二是缺陷的修复, 三是代码优化、代码移植。软件一旦有变更, 测试必须进行验证。对于每一个变更都将涉及变更点本身实现的正确性验证, 以及是否影响到原有功能的正确使用。回归测试的最主要目的也就是保证这两点能得到充分验证。由于回归测试的工作涉及原来已测试通过的用例重复执行, 特别是在产品的维护阶段, 面对用例库中成千上万的用例, 是全面测试, 还是有选择性地挑选其中的部分用例来测试, 选择的原则又是什么。这些都需要在回归测试策略中决定。回归测试策略的设计直接影响着测试的成本与效率。本节讨论回归测试的内容, 结合案例讲述回归测试方法。

8.3.1 确定回归内容

首先, 我们来看一下, 软件变更的 3 个主要方面。

- 新增需求。
- 缺陷修复。
- 代码优化、代码移植。

不同的更改源, 实现的目的不同, 对原系统的影响面也不同, 需采取不同的回归策略, 以保证回归测试的效率与有效性。接下来从基于用例的回归测试方法及 Bug 的回归测试方法两方面介绍回归测试的策略。一般情况下, 在新增需求及遇系统代码优化或移植情况下, 需考虑用例型的回归策略。缺陷修复后的回归, 采用基于 Bug 的回归方法。

8.3.2 基于用例的回归测试方法

1. 影响分析回归

影响分析回归, 是指把变更点影响到的功能点对应测试用例挑选出来, 重新回归这些用例。这种回归方法要执行的用例较少, 从效率上来说是最高的, 但风险也是最大的。回归的好坏依赖于影响分析的准确性、充分程度。可结合以下两种操作方法进行控制, 以降低风险。

1) 从黑盒角度分析

从分析需求开始，站在黑盒角度的功能测试层面，找出更改点所处的业务位置，结合系统中与它关联的业务，画出业务功能的影响关系图。通过图中的影响分析收集（选择原用例，或新增用例，或更改原用例）测试用例，形成回归测试包进行测试，适合于新增需求的变更。下面是一个案例分析。

【案例】

现在假如手机通信录信息编辑中新增加了一个“电话号码”字段，以使用户可以多保存一个电话号码。从目前手机常用的业务功能关联分析，可得出如图 8-11 所示的影响关系图。

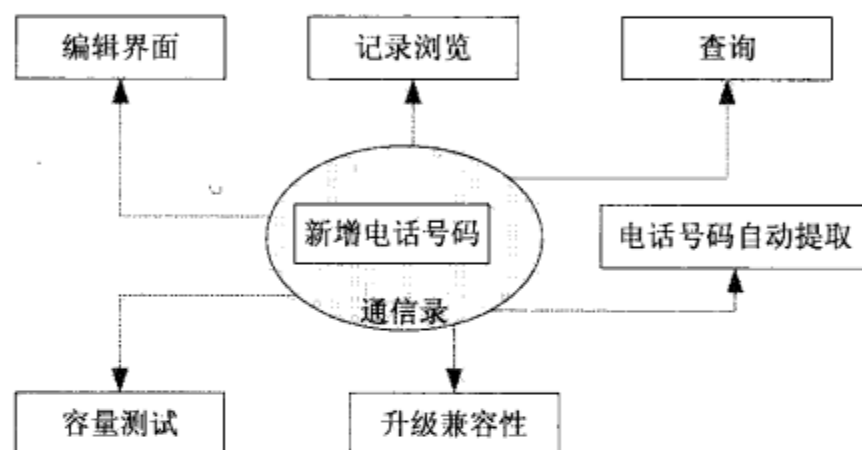


图 8-11 “新增电话号码”业务功能关联图

图中，表示“新增电话号码”的更改发生在通信录应用模块，增加字段影响通信录编辑界面的显示，以及编辑后记录的浏览；查询模块可以查到新加入的“新增电话号码”信息。同时，由于通信录数据结构的变化，软件升级后影响数据的兼容性。至于容量测试，新增一个字段，会使每条记录的大小发生变化，也需验证。上面的分析，实际上是提取测试点的过程，接下来可以根据这些测试点，挑选原用例或设计部分新用例来执行回归测试。

2) 从白盒角度代码层次分析

首先，找到代码变更的部位，然后找出发生变化的函数的调用与被调用关系。从代码实现的角度画出变更点的模块与模块之间、函数与函数之间的依赖图，从依赖关系上找出必测路径。但这种方法有一个弊端，就是容易陷入设计的思维中，把用户的实际使用场景忽略了。下面仍以上面的需求变更场景为案例，讲述从白盒角度的代码层次如何做回归测试的影响分析。

【案例】

同样以上面的案例为例，现在假如手机通信录信息编辑中新增加了一个“电话号码”字段，以使用户可以多保存一个电话号码。

从代码实现的思路分析，更改影响到 3 个表示层的模块，包括通信模块、查询模块、其他模块，业务逻辑层影响到文件管理模块。如图 8-12 所示，C1、C3、C4、C5 表示通信录模块中受影响的函数，而 C3 被其他模块中的函数 T1、T2 调用。由于新增字段，通信录的数据结构发生变化，在软件升级后，存在新旧数据转换的兼容性问题，此业务在数据管理模块处理。F1 是查询模块中的一个函数，需增加对新增字段的查询，直接与数据管理模块打交道。

通过上述分析，我们可以得出以下 8 条必测用例（图 8-12 中直线相连部分）。

用例名：路径

TC1: C1-D3

TC2: C3-D2

TC3: C4-D2

TC4: C5-D2

TC5: F1-D2

TC6: T1-C3-D2

TC7: T3-C3-D2

TC8:D1-D2

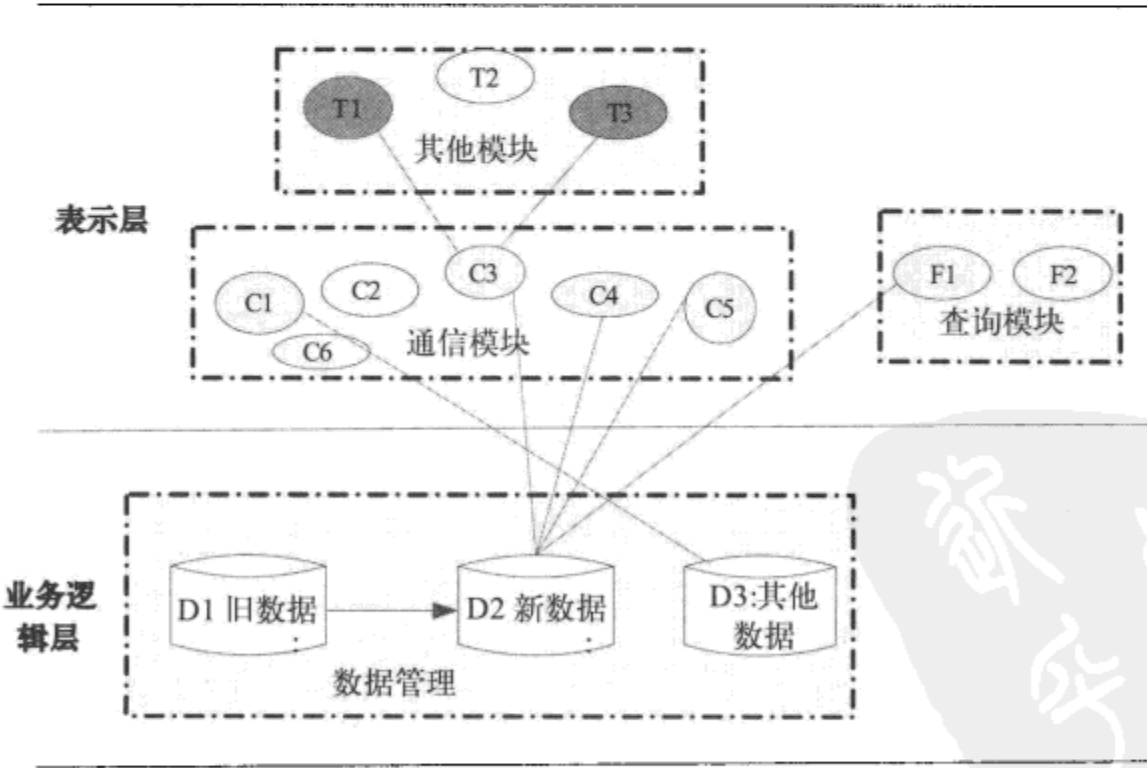


图 8-12 代码实现模块依赖影响分析图

上述 8 条用例，是转化成黑盒测试用例，还是通过代码走查或打桩的方式，再或者是构造数据等其他方式来验证，我们可以看具体情况，从测试的效率与有效性上衡量执行的方式。

2. 全部回归

全部回归是指把用例库中的所有用例重新再执行一遍，这种回归基本没有风险，但也可能不会发现新问题。它还有一个最大的问题就是测试成本高，特别是当用例集庞大，如有上万条或更多的用例时，手工回归的工作量往往不是以人/天来评估，而是需以人/月来衡量了。当然，此时读者朋友也许会说，我们可以用自动化测试来回归。没错，这的确是一个不错的选择，但在实际的工作中，特别是在研发中的项目，当版本不稳定时，自动化的策略往往不尽人意（关于自动化回归测试超出本节讨论的主题，此处只提及）。当然，用例全部回归后，测试人员心里对软件质量的整体表现将更加清楚，会更踏实。建议全部回归的策略要用，但少用，要用在关键点上。

3. 影响分析回归与全部回归相结合

在软件的开发阶段，由于版本一直在迭代，频繁提交的版本中会不断增加新功能。在新版本上除测试新功能外，理想情况下还需验证原来的功能，以确保新增加功能不影响原有功能。在实际的研发过程中，项目进度通常都非常紧张，在每一个内部测试迭代版本上执行全部回归是不现实的。但如果一直到最后才进行全部回归，工作量将非常庞大，同时，必将造成某些 Bug 发现得太晚。这种情况下，可以采取影响分析回归与全部回归相互结合的迭代回归策略，在某小阶段进行影响分析回归，大阶段进行全部回归，最后在版本发布时再进行全部回归。整个测试过程不断进行迭代回归测试，以降低 Bug 没有及时发现的风险，以及减少过度测试，如图 8-13 所示。

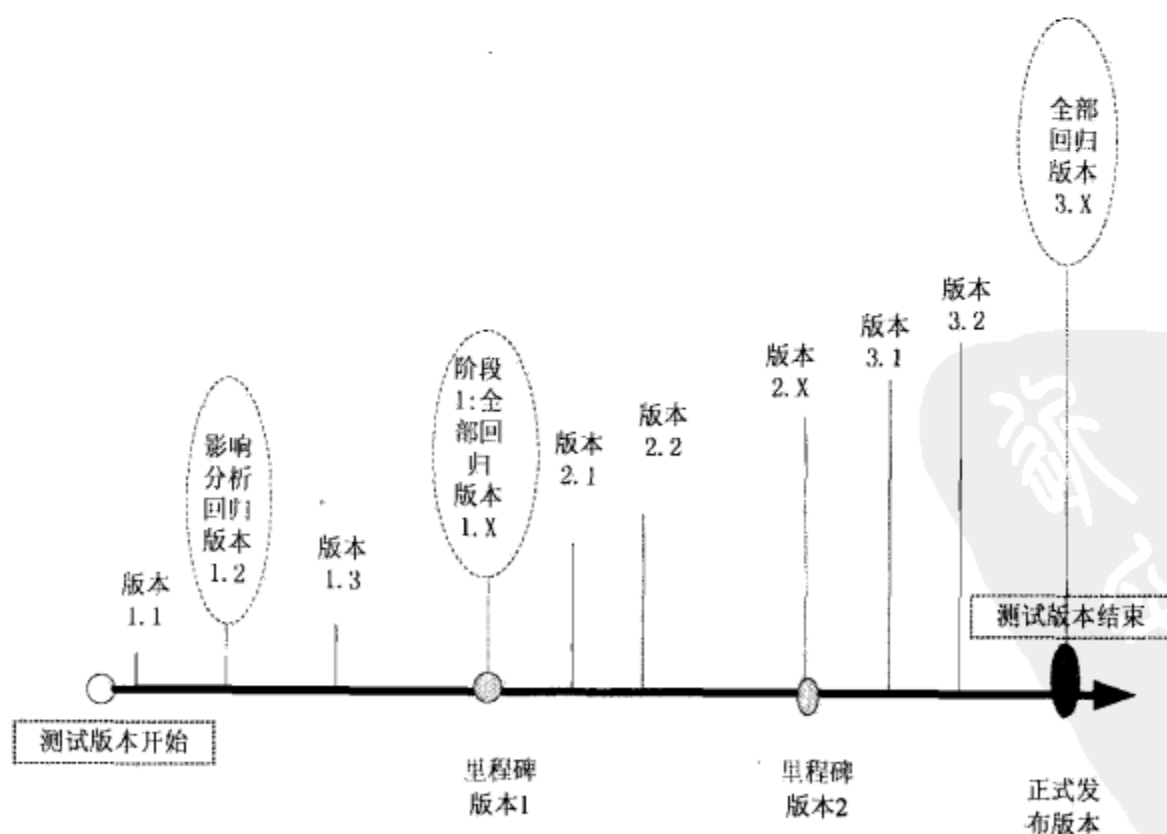


图 8-13 影响分析回归与全部回归相结合的迭代回归模式

4. 巩固性回归

适合于更改很小的补丁发布版本，无论正向分析，还是逆向分析，更改都不会影响原有功能的正确使用，对更改点的验证所有用例也都 PASS。这时，选择产品中最重要的功能用例，以及使用频度高的功能用例进行巩固测试，如影响生产或在线升级的功能用例，每次版本发布时必测（可事先设计好必测试项的 checklist），以提高软件质量可靠性的信心。

还有一种情况，也适合开展巩固性回归测试。开发过程中，特别是产品进入维护阶段，有这么一种现象，当某个版本发布后，如果没有新需求或用户投诉的 Bug，不会再主动发起测试的任务（认为任务没有来源），而此时可能有部分测试人员处于空闲状态或半闲状态。从软件工程本质上看，我们所采取的测试方法，是基于项目的进度与成本考虑的，这一点每个人都很清楚。换句话说，软件并不是没有 Bug 了，而是暂时符合了测试的停止条件，符合了项目的要求。但是当条件允许时，即使是版本发布了，根据质量比较薄弱的环节（重要模块但常发 Bug），有意识主动去进行巩固测试，是一件好事。比起接到投诉后再被动测试，结果完全不一样。

巩固性回归测试，就像剩饭一样缺乏新鲜感，所以在安排资源时可优先考虑没有测试过此模块或长时间没测试此模块的测试人员。常常，他们有不同的视角，能发现一些意想不到的遗留 Bug。

8.3.3 基于 Bug 的回归测试方法

1. 彻底回归版本的 Bug

开发人员解决了一个 Bug，同时又引入了新的 Bug，但测试人员在回归 Bug 时没有及时发现，这就叫回归不充分。这种现象无论对开发人员，还是测试人员都是相当痛心疾首的。该如何彻底解决此类问题呢？下面是几点最佳实践方法，供读者朋友借鉴。

1) 记录 Bug 是如何修改的

修改缺陷管理库，增加开发人员填写“Bug 解决方案”、“影响分析”、“测试建议”更改的影响分析，包括此更改对本模块、其他模块的影响，给测试提出建议。

开发人员注明修改了哪个模块的哪些函数（有些附上修改前后的代码），这些信息有助于测试人员站在测试的角度去分析判断该 Bug 修改后的影响。

2) 开发人员确认 Bug 被更改

开发人员修改好 Bug 后，先在开发环境下自行调试，然后安装版本到真实环境下，验

证 Bug 更改的有效性，以控制提交测试的 Bug 版本。

3) 记录 Bug 是如何回归的

测试人员设计 Bug 回归测试用例，并严格按用例执行，在缺陷库上做好回归路径的记录。Bug 回归的全面性很重要，如何设计更改点的用例，可采用上一节介绍的影响分析法设计回归 Bug 的用例。

解决 Bug 的彻底性、回归 Bug 的全面性，与解决 Bug 的开发人员、回归 Bug 的测试人员对系统业务、软件系统逻辑的熟悉程度有很大关系。

2. 回归缺陷库中的全部 Bug

据调查，几乎有测试实战经验的朋友都遇到过，并不是缺陷库上的每一个 Bug 都有用例对应的。或许有的测试人员会说，如果没有用例对应就补充用例。这样做是应该的，但是产品研发过程中的实际情况远比我们想象的要复杂，有些 Bug 可能直接原因不是软件本身，而是受相关环境的影响导致的，如存储设备、硬件驱动等。在最后的发布版本上可以对缺陷库上所有曾经发生过的 Bug 回归一遍，验证是否仍存在。以我们过去回归的经验，重新打开率在 0.5%~1% 左右。更重要的是，回归 Bug 时，可以同时做一些随机测试，常能发现一些新的 Bug。

8.4 交叉测试

测试人员都知道，我们想把软件中潜伏的所有 Bug 都找出来，但从软件工程体系分析我们是不可能做到穷尽测试的。在实际工作中，产品的研发有时间与成本的限制，于是测试的效率与有效性是我们工作的重点。提到测试的有效性，笔者很自然想到一种我们无法改变的事实，就是测试人员 A 测试通过的模块换测试人员 B 来验证，总会重新发现一些新问题，交叉测试也就因此应运而生了。开始笔者与很多人一样，对交叉测试的结果不太理解，好像充满了神秘色彩。后来随着对测试知识面的扩大，特别是读了测试大师 Glenford J. Myers 的《软件测试的艺术》后，对测试艺术有了深刻的理解，有一种恍然大悟的感觉。交叉测试能产生这样的结果，实际上是符合人的思维发展规律的，在社会心理学中，称之为定位效应。社会心理学家曾对此做过一个试验：在招集会议时先让人们自由选择位子，之后到室外休息片刻再进入室内入座，如此五至六次，发现大多数人都选择了他们第一次坐过的位子。交叉测试，正是利用了人的思维定势的弱点而有了突破性的收获。

交叉测试效果显著，是解决测试过程中出现迷茫时的一剂良方，但这剂良方如何用，何时用是一个需谨慎考虑的事情。接下来的章节，与读者一起分享关于交叉测试的职场故

事，可从中能找到交叉测试应该如何做、何时做的答案。

8.4.1 交叉测试的特点

谈到绩效考核，很多人心里就会有一种忐忑不安的感觉，有一种无形的压力。测试人员的业绩该如何考核，特别是要拿出一些量化的指标时，用什么呢？这是测试管理者的一道难题。或许交叉测试的方法效果特别，笔者了解到有不少公司是把测试人员交叉测试中发现的 Bug 作为考核测试人员的工作质量的。就笔者本身而言，也曾经历过。但是，交叉测试的目的绝不是为了考核，仅是控制质量的一种手段，通过绩效考核这支杠杆推动测试人员去反思，尽可能地避免漏测。

【案例】交叉测试的奖与罚

10 多年前，那时，笔者与一些测试朋友在一家台资企业测试 PDA 应用软件，比如电话本、备忘录、闹钟、约会行程，游戏等软件。我们的分工是采用模块负责制（这种方法现在仍是测试工作安排的主流方法），即一个人在一段时间内专门负责测试某个或某几个模块。当时基本上是只有测试执行的过程，在开发的中后期介入，平均 3~5 天开发发布一个测试版本，测试人员提交 Bug 单（纸张填写），开发解决 Bug 后再提交测试，就这样版本在迭代中周而复始地测试。当我们测试各自负责的模块约一个半月后，有些测试人员才提交一两个一般性 Bug（Bug 分级别进行管理，常常分为严重、一般、次要、建议 4 个等级，设立几个等级才合适，可根据公司的产品不同而异），甚至有些测试人员一个 Bug 都提交不出来。回答的原因都是“能想到的路径都已测试过，不觉得有 Bug 存在了”。于是老板问我们“如果这个软件卖给你，你是否有信心？”并要求我们多想想，再测试两天看看。老板的意思，我们当然明白，于是大家又在绞尽脑汁去测，第一天，依旧基本没有新发现。第二天，大家好像继续在埋头苦干，时光就这样在我们的默默工作中悄然逝去。

下午两点多时，测试人员小余突然叫了起来，“佟同学，约会行程显示乱屏了！你过来看看”（约会行程模块是由佟同学负责测试的），话一出，真成了爆炸性消息，办公室里 6 个测试人员几乎都围上来看。小余说“是因为测着测着，有点犯困了，于是到约会行程中逛了逛，无意中设置了约会闹铃。出于对新东西的好奇，就细细浏览刚设置的信息，突然，这条记录的响铃时间到了，结果原来正在浏览的内容给响铃界面冲破了，出现乱屏”。这个 Bug 的出现，可大大唤醒了我们的麻痹意识。于是我们私下商量相互交换着模块测试。效果果真不一样，就一个下午，一位测试“电话本”模块的同事与测试“备忘录”的同事

互换后，就发现了一个在画图的过程中机器突然黑屏的严重问题。但这种私下的交换测试，与个人的情绪有关，谈不上任务与责任。

后来，老板给我们定了一个制度“当所负责的模块已测试完成，测试人员之间进行轮流换模块测试，下一位测试人员发现前面测试人员遗留的 Bug 时，前面的测试者扣奖金，后面的发现者加奖金，加奖金不封顶，奖金扣完为止。具体扣多少与当月的浮动奖金挂钩”。这一政策出来，既让我们高兴也让我们忧愁，高兴的是奖与罚是透明的，能激发大家的工作热情。老板说到做到，在我们陷入测试的混沌状态时，拿出了这一“杀手锏”，让我们每个人精神大振。接下来的几天原说再也提不出 Bug 的测试人员还提交了几个 Bug。眼看一周又快过去了，老板看我们状态已转变，但效果仍是不太理想，于是决定下周开始，各测试人员就开始交换测试模块。时间为 1 周，1 周后再与下一个人交换，直到所有模块都被所有的测试人员测试通过。当交换模块后，后面的测试人员检查前面的测试人员的工作，谁都是检查者也是被检查者。

周一上班，个个测试人员都精神饱满（也与周末刚休息完有关吧）、干劲十足的样子，因为开始测试新模块了（交换后的模块，对他人来说就是新的），但也有几分担心，怕他人发现自己漏测的问题。即便这样，办公室的工作气氛还是不一样了，交流多了，讨论问题热烈了。一周下来，基本每个模块都存在漏测的 Bug，少的有一两个次要或一般性 Bug，多的有五六个 Bug。短短一周的交叉测试，让我们每个人都相互学习了，成长了。这正如英国杰出的戏剧作家萧伯纳所说：“倘若你有一个苹果，我也有一个苹果，而我们彼此交换这些苹果，那么你和我仍然是各有一个苹果。但是，倘若你有一种思想，我也有一种思想，而我们彼此交换这些思想，那么，我们每人将有两种思想”。

面对被其他同事发现而自己测试了近两个月都没有找到的 Bug，没有理由不好好反思、总结。这一次，笔者被扣了 100 元（两个遗漏的 Bug 被其他同事发现），但同时由于发现了其他同事遗漏的 4 个 Bug，也奖了 200 元。其中有一个比较典型的问题至今记忆犹新，是错别字的问题，需求要求的是“通讯”，而开发工程师把它写成了“通信”（在那时，通讯功能还没完全实现数字化，通信与通讯是有区别的，见下面的解释）。

小贴士：

通信与通讯

通讯：传统意义，通讯主要指电话、电报、电传。通讯的“讯”指消息（message），媒体讯息通过通讯网络从一端传递到另外一端。媒体讯息的内容主要是语音、文字、图片和视频图像。其网络的构成主要由电子设备系统和无线电系统构成，传输和处

理的信号是模拟的。所以，“通讯”一词应特指采用电报、电话等媒体传输系统实现上述媒体信息传输的过程。

通信：指数据通信，即通过计算机网络系统和数据通信系统实现数据的端到端传输。通信的“信”指的是信息（information），信息的载体是二进制的数字。数据则是可以用来表达传统媒体形式的信息，如声音、图像、动画等。

目前，由于旧的通讯系统早已实现了数字化、计算机网络化改造，因此可以认为目前的数据通信系统已涵盖了过去的“通讯”系统的功能。

当时，笔者也没有深入地想过老板为什么要采用此法来控制质量，对我们又有什么好处，因为涉及扣奖金的风险，更多人是抱怨老板要求的苛刻。但是，交叉测试立竿见影的功效，大家都有目共睹。后来，在不同公司任职时，笔者也常常采用此流程控制方法来加强测试，以尽可能地减少漏测。

8.4.2 交叉测试模式

通过前面的案例，读者朋友都已看到它给我们实际工作带来的好处。实践表明，交叉测试是一种符合人类思维发展规律中存在思维定势这一心理特点，同时运用他人的不同思维定势空间形成优势互补的一种科学的测试策略，如图 8-14 所示。

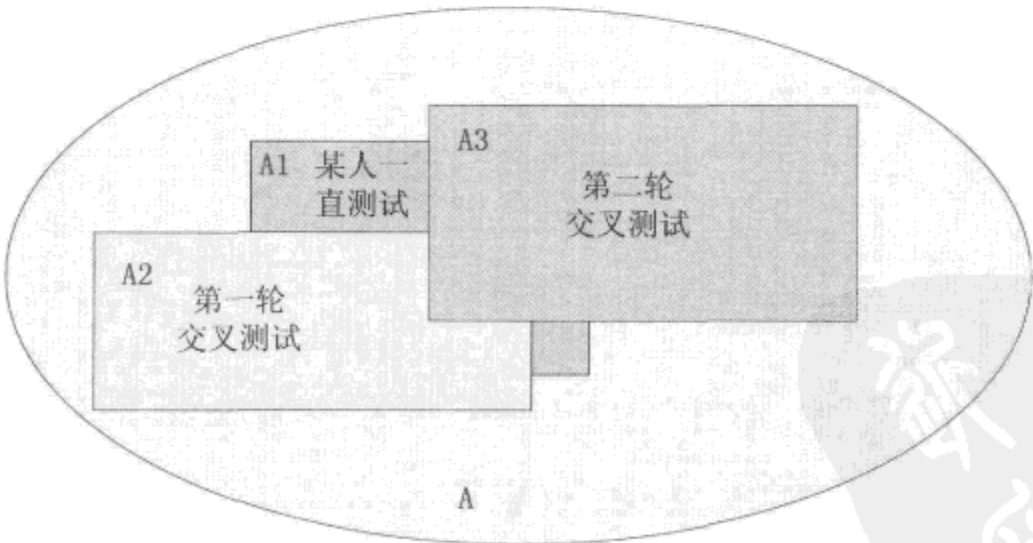


图 8-14 Bug 分布范围与交叉测试覆盖范围

图中，假如某模块的 Bug 分布范围为椭圆 A，那么 A1 表示第一个测试某模块测试人员所覆盖的范围，执行第一轮交叉测试后，Bug 的覆盖范围扩大到 A2 区间。同样，通过第二轮的交叉测试，Bug 的覆盖范围达到 A3 区间。由于每个人对同一事物的认知度不同，思维定势形成的区间也不同。也正因为有这些不同，通过交叉测试的互补，扩大了对

Bug 的覆盖范围。从理论上说（图中也可反映出），交叉测试执行次数越多，参与的人数越多，Bug 的覆盖率将越大，但在工程实践中，更关注的是适度测试。

交叉测试的模型大致可以理解为如图 8-15 所示。在实际工作中如何应用，什么时候用，常是我们实践派关心的问题，接下来介绍这方面的内容。

1. 保守式交叉测试

上面案例中的应用可以归纳为是一种保守型的交叉测试，测试的主体工作放在第一个测试此模块者身上，接下来的交叉测试作为一种互补措施来提高测试覆盖率。它的优点是突出模块负责人的核心作用，职责分明，能使测试人员不仅在深度，还在广度上深入测试。缺点是会造成某些遗留问题较晚才发现，因为思维定势的原因，某些问题一直存在，但测试者就是注意不到，直到后期交叉测试有新人加入发现为止。这种保守型的交叉测试开始时机是版本已基本稳定，模块测试负责人自身认为已很难再发现新问题时，交出模块进行交叉。并且如果时间条件允许，项目组内的所有人都交叉进行一遍，以此达到高覆盖率，如图 8-15 所示。

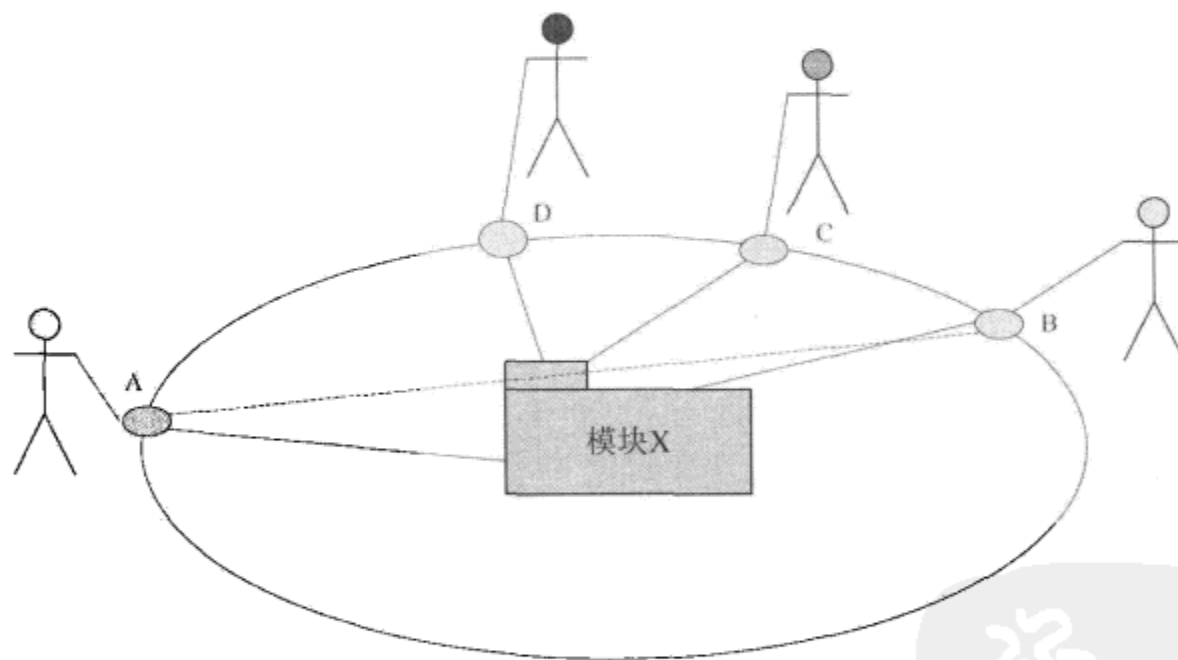


图 8-15 模块 X 的测试周期与交叉测试节点示意图

图中所示模块 X 从开始测试到最后版本发布，共经历了 A、B、C、D 4 个测试节点，经过 4 个不同的测试人员测试通过，其中 AB 大阶段由模块负责人测试，从 B 点开始交叉测试，共经历了 3 轮的交叉测试。

2. 渐进式交叉测试

针对保守式交叉测试的缺点，在后来项目测试的实施过程中，我们做了相关改进，称为渐进式交叉测试。就是在整个测试过程中，一个模块由项目组的所有人一起负责（通常

指3~5人的小团队，更大些的团队会不好操作)，一人负责测试一个小阶段（一般不超过3个内部版本，1个版本的周期不超过1周）。然后把模块交给下一个人测试，下一个人在测试上一个人测试过的功能时，还测试新增功能，如此往复，直到版本发布。这种模式的好处是交叉测试的轮次多。由于开发过程中版本一直在迭代，新功能一直在增加，对于不同阶段介入的每一个人来说，测试时都感觉是在测试新东西，能很好地集中注意力，对创造性思维的激活有帮助，常能发现彼此遗留的一些新问题。

这种方式的缺点是职责不够明确，有吃大锅饭的倾向。特别是越到后面发现一些新Bug时，要花不少时间才能搞清楚到底是哪个节点的测试遗漏。另外，由于模块没有专职的主力去测试，在业务的深度方面挖掘不够，更严重的是项目组内各成员都熟悉各模块的功能，但没有一人说得上精通某一模块。有了广度，却牺牲了深度，如图8-16所示。

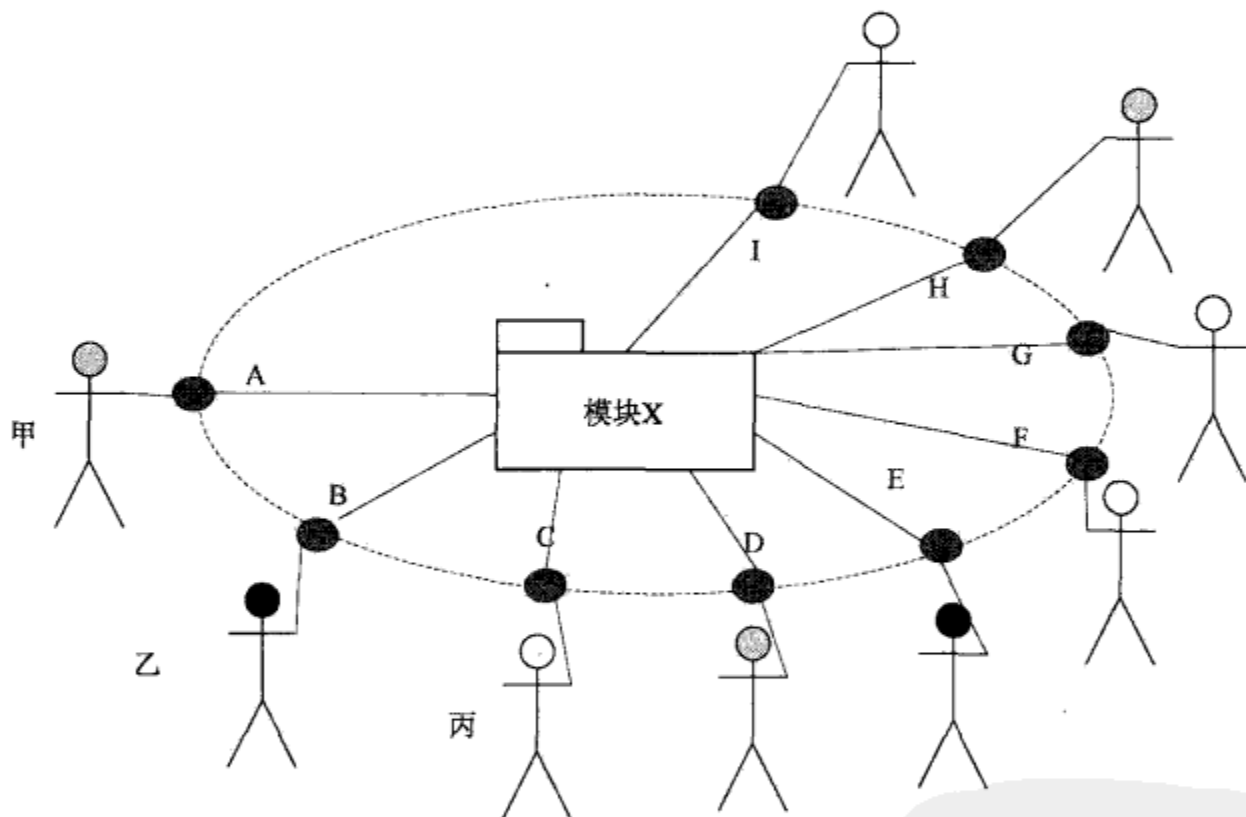


图 8-16 渐进式交叉测试示意图

图中，模块X的测试周期，共经过甲、乙、丙3人的测试，共测试9个小节点版本。在软件版本的迭代过程中，他们分别在测试若干内部版本后进行交换测试，但这种交换并不严格，如第三轮中，丙测试了两个小阶段版本（可能甲、乙请假或其他原因等，这是实际中经常出现的情况）。整个测试过程，甲测试了3个节点版本，乙测试了2个，丙为4个。

3. 零交叉

是否所有的测试都需进行交叉？笔者在实践中也经常问自己这个问题，回答是否定

的。例如以下类型的测试不必进行交叉测试：

- 由脚本控制的压力测试。
- 通过工具进行的专项测试，如内存泄漏分析。
- 自动化测试。

也就是说，如果输入是固定的，执行过程也是固定的，完全由程序进行控制的测试，没必要进行交叉测试。要改变测试的结果，需要更改的是测试代码，而不是换人就可以提高覆盖率的。

8.4.3 交叉测试后的进一步思考

从 8.4.1 节的案例中，我们能感知到，交叉测试是一种很好的控制软件质量的手段，也是推动测试人员之间相互学习相互提高的有效方法。但是，如果应用不当，会使团队成员走入形式上的“交叉测试”，而收不到实际上想要的效果。如在项目即将发布的后期才进行交叉，即使发现问题，常会由于太晚了，迫于产品上市的压力，而延期解决发现的 Bug；在项目的前期，当版本不稳定且大量 Bug 存在时，就进行交叉测试会失去方法本身的意义；目前的测试根本不适合做交叉测试，如自动化测试。所以在执行交叉测试前，需事先规划，考虑各方面的影响因素，选择合适的交叉测试模式，才能取得预期的效果。

当一个项目测试结束，总免不了一番总结，对于交叉测试过程中暴露出的一些问题，除了 Bug 外，还可能出现如下一些问题值得我们思考。

- 同样的一条用例，在代码没有发生任何变化的情况下，前面测试者 PASS 了这条用例，而后面测试者 FAIL。我们试图客观地寻找原因，是不是用例描述的二义性问题？用例的预期输出不唯一问题？得到的答案是否定的。
- 需求变更了，但用例没有及时更新，而前面测试者 PASS 了这条用例，后面的测试者发现此用例不可执行。
- 交叉测试后发现了一些新 Bug，但对应的用例却没有补充。

上述这些问题，测试朋友可能也遇到过，每当涉及与人有关的问题时，是最让测试管理者头痛的事。要拿出一个好的解决上述问题的方法，波及面较大，如用例的设计规范，以及有了规范后的如何执行问题；测试执行过程的环节如何规范化，如何用合适的工具来监控人的不可控的随意行为等。这是一套值得我们研究的测试工程问题。

第 9 章

测试输出管理设计

前面的章节主要是介绍测试方案、用例设计的方法，以及测试执行流程的设计，都是围绕如何更有效地发现 Bug 为核心的测试设计。本章反过来，以测试的输出为核心，介绍如何以有效的方式管理测试输出，充分利用测试输出的价值，改进测试流程，如图 9-1 所示，以期提高软件测试的质量。结合案例，介绍解决措施及改进测试过程的方法。

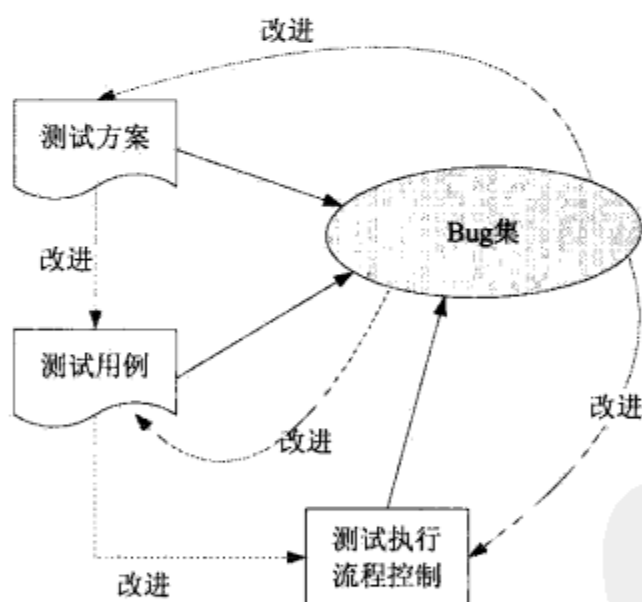


图 9-1 以测试输出为核心改进测试过程

9.1 Bug 管理

一位测试朋友向另一位外行朋友介绍测试工作时说：“测试工作，就是找 Bug 的活，即找出软件中的错误。”从测试工程学上来看，这种说法是不全面的，但却道出了测试中

人们最关心的问题：找 Bug。

笔者作为面试官，也曾问过不少测试朋友“做测试，最令你有成就感的是什么？”几乎有 85% 的人回答是“发现 Bug”。虽然测试的贡献不仅仅是发现 Bug，但是测试的核心目的就是尽早发现 Bug，通过 Bug 的解决提高测试质量。诚然，Bug 成了测试工作输出的重要指标。Bug，有很多人也称为缺陷，实际上它们是有区别的，详见下面的小知识。

【小知识】Bug 与 Defect 的区别

在不少 Bug 管理工具中，Bug 录入界面常看到“Bug 详细描述”字段被“缺陷详细描述”代替，认为两者是等同的。那么，这两者是不是有同样的意思，它们之间的区别又是什么？请看下面的解读。

Bug: n. 臭虫，在计算机软件中是指设计上的，或者是编码上的错误，也可能是需求上的错误等，它是一个错误。根据严重度的不同，可分成不同级别的 Bug。

Defect: n. 缺陷，是指设计不合理，或设计上存在漏洞有待改进。

下面是关于 Bug 和 Defect 由来的故事：

在我们还使用电子管技术制造计算机的那个年代，计算机的主机重达数吨，并常常占据整个房间。在某个实验室的某个平常的早晨，这个庞然大物突然停止了工作，我们的 IT 前辈们马上就开始寻找出现这种情况的原因。凭借设计图纸的引导，他们很快就圈定了可能发生问题的那一部分。在接下来的检查中，他们发现这次故障原来是一只虫子在经过两只继电器时造成了短路所致。在修复了计算机并重新开始工作之后，负责计算机维护的工程师把这次故障记录在了一份备忘录上，以便将来其他人遇到类似的情况可以迅速地找到答案。当然，他还写了一份文档给计算机的设计人员，希望以后可以在主机的散热孔那里加装一层更加细密的金属网。既不影响散热，又可以防止虫子爬到主机里。

发现上面的区别了吧，爬进主机引起短路事件的虫子，被我们称为 Bug，这个名词一直从计算机硬件故障沿用到了计算机软件故障。那么 Defect 又是什么呢？还是看上面的这个例子，真正的 Defect 是计算机维护工程师提出来的那个问题：在主机的散热孔那里加装一层更加细密的金属网，既不影响散热，又可以防止虫子爬到主机里。这是计算机设计人员疏忽的地方，是产品真正的 Defect。而虫子引发的那个故障只是这个 Defect 导致的其中一种故障的表现形式。也就是说，Bug 是 Defect 的一种表现形式，而一个 Defect 是可以引起多个 Bug 的。

接下来从“Bug 单的故事”谈谈测试人员与 Bug 的特殊情结，Bug 管理对于测试经验传递的重要意义；Bug 管理工具要如何选择；介绍如何管理 Bug，利用 Bug 资源为团队、为项目服务。

9.1.1 “Bug 单”的故事

在测试工作中谈及 Bug 的输出时，通常我们都说提交了多少个 Bug。但笔者的一个同事老 D 却很特别，每说 Bug，后面总要带着一个“单”字，如“提交了几个 Bug 单、回归了几个 Bug 单，等等”，好像与“Bug 单”有着特别的情结。一个“单”字，让人很容易联想到单据、票据之类的纸质东西。怀着这份好奇，笔者与老 D 聊了起来。他说这与他在第一家公司做测试时有关，当时他们公司规模小，电子化的办公流程也并没有像今天一样普及。每发现一个 Bug，他们就填一张 A4 纸大小的 Bug 单（见表 9-1），然后把此单交给开发人员处理。当把一张单交给开发时，他说那是他最有成就感的时刻了。后来，开发人员一见到测试人员手上拿着一张单，心情就特别紧张。开发人员填写完处理意见后，再交给测试人员回归 Bug，测试验证通过后，则此单测试人员将会把它放在一个专门的文件柜中归类保存。老 D 在那家公司一直工作了 5 年，已习惯了这种叫法。后来离职时，说由于提 Bug 单的感觉真的很美，就拿了一张空白的 Bug 单回家留做纪念。老 D 这样娓娓道来的故事，原来是对 Bug 单情有独钟。

表 9-1 Bug 单样式表

单号:				
程序名:		版本:		
现象描述:				
	测试人:		测试时间:	
原因及处理:				
	修改人:		修改日期:	
复测:				
	复测人:		复测日期:	

这是一个真实的故事，别看一张不起眼的 Bug 单，其中凝聚着我们的测试方法，测试的经验、技巧。一张张 Bug 单是测试新人学习的宝库，也是有经验的测试人员之间相互学习交流的平台。同时，也是测试对项目贡献的重要依据，对日后问题的追溯也是一个宝贵的资源。既然 Bug 对我们那么重要，如何把它管理起来促进我们的工作，在所有的测试输出中，是首先要考虑的任务。老 D 曾经用过的纸张填写 Bug 单的管理方式，对于普及电

子化的今天来说已过时，但聪明的读者朋友，也许你已看到里面隐含着一个很重要的 Bug 生命周期管理的迹象，如图 9-2 所示。这也是后来专用 Bug 管理工具的雏形。



图 9-2 简单的 Bug 处理流程图

当然，今天在我们实际工作中 Bug 的处理流程远非图中的那么简单。接下来就 Bug 工具的选择与管理的流程结合案例进行介绍。

9.1.2 Bug 管理工具的选择

正如“工欲善其事，必先利其器”，要管理好 Bug，我们必需选择一个好的管理工具，它是 Bug 赖以生存的载体。

这里回放一下，曾经见过的记录 Bug 的载体。

场景 1：一张 A4 纸面的 Bug 单，一个 Bug 记一张单。

场景 2：Word 文件，可以保存 N 个 Bug，但自动统计不方便。

场景 3：Excel 文件，可以保存 N 个 Bug，可以对 Excel 进行二次开发、实现数据自动统计、图表自动生成等，但需人工维护 Bug 生命周期状态，修改历史也不能保留。

场景 4：专业 Bug 管理工具，Bug 相关的信息保存在数据库中，状态自动更新，状态变迁数据可保留，可自动统计所需数据、图表，发 E-mail，不同用户有不同的处理权限等，功能强大。

目前基本上有一定规模的公司，都会使用专业的管理工具，即使是小公司，建议也引入专业的工具。专业工具主要有以下 3 种来源。

- 商业工具。
- 开源工具。
- 自主研发。

在工具的选择上，优秀的工具很多，目前比较流行的开源、商业工具总结如表 9-2 所示。从表上看，它们的特点大同小异。在决策时，主要看公司的管理投入与用户的使用要求。适合自己的就是最好的，也正因为如此，使用各种工具的都有（远不限于表中所列）。

表 9-2 常见 Bug 管理工具特性比较

工具名称/特性	Bugzilla	Mantis	ClearQuest	TestDirect
技术特性				
构架模式	B/S	B/S	C/S, B/S	B/S
支持平台	Windows linux	Windows linux	Windows linux	Windows
支持数据库	MySQL	MySQL	Oracle、Access、SQL Server 等	Oracle、Access、SQL Server 等
用户常用功能				
Bug 流程定制	能	能	能	能
功能表单字段自定义	能	能	能	能
附件支持	能	能	能	能
E-mail 支持	能	能	能	能
用户角色权限管理配置	能	能	能	能
报表、图表输出	能	能	能	能
导出数据	能	能	能	能
界面	不够友好	简洁	较好	较好
使用限制				
价格	开源免费	开源免费	收费(几万几十万不等)	收费(几万几十万不等)
访问限制	无限制, 适合大中型企业	无限制, 适合中小企业	访问人数受 license 限制, 适合大中型企业	访问人数受 license 限制, 适合大中型企业
安装配置	安装较复杂	安装简单	安装较复杂	安装较复杂
技术支持	版本更新较快	版本更新慢	有专门技术支持(收费)	有专门技术支持(收费)
可集成软件	多	少	受开发商限制	受开发商限制

从使用灵活性、成本方面考虑, 笔者建议使用开源的 Bugzilla 或 Mantis。如果公司使用工具团队的人数不多(几个或几十个), Mantis 已可满足, 安装也简单。如果公司较大, 使用的团队几百上千人, 建议使用 Bugzilla。虽然安装较复杂, 但用户群多, 一般技术问题在网上的开源社区中都能找到解决方案, 版本更新也较快。

9.1.3 Bug 生命周期设计

从表 9-2 中, 我们可以看到, 无论是开源还是商业的 Bug 管理工具, 它们的功能其实

都相似。如果是我们进行自主开发，大部分的功能也都会相似，这是由 Bug 管理的应用性质决定的。例如 Bug 的录入、查询、状态、数据统计、报表及图表生成等，这些都是测试日常工作中常用的功能。测试朋友都知道，其中的 Bug 状态变迁，即 Bug 的生命周期设计关系到开发、测试相关人员的工作，影响着研发流程的整体控制，是 Bug 管理中的核心。如图 9-3 所示是一个简单而常用的 Bug 生命周期定义。

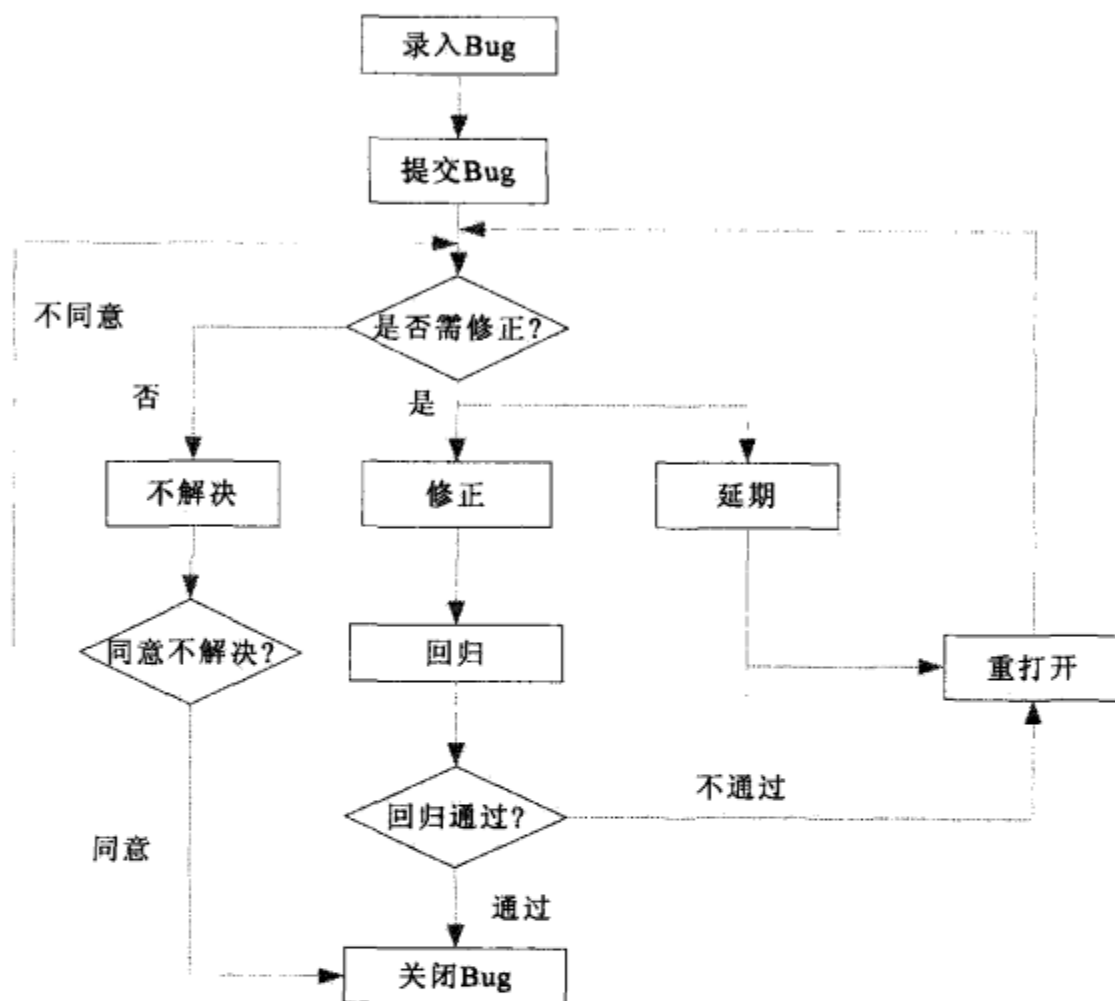


图 9-3 Bug 生命周期示意图

图中涉及的动作，表示 Bug 当前处于什么状态，如录入、提交、修正、回归、关闭。状态之间的变换通过开发人员或测试人员的处理进行触发，有些甚至需要项目经理、产品经理的参与，例如一些不解决的 Bug、延期解决的 Bug。流程中哪些动作由测试人员来完成，哪些动作由开发人员完成，涉及角色与任务的定义。这一点很重要，可根据公司的实际情况来定义。

下面是一个关于 Bug 在其生命周期中状态如何变换的实例。

【案例】那个 Bug，像一个皮球在开发与测试之间踢来踢去

背景：软件需求由开发人员编写，开发人员既是需求编写人员，同时也是代码实现人员。在这种情况下，处理 Bug 的流程上会出现一种“踢皮球”的现象，如图 9-4 所示，开

发与测试各执己见，互不相让。

这一天，测试主管欧也刚休完七天年假回来。上班后，按往日的惯例，打开电脑第一件事情就是登录到 ClearQuest 故障管理系统中，查看并分析库上 Bug 的变化。

当然，首先关注的 Bug 便是“严重”级别的，其中一个严重的 Bug 是一周前提交的，现在为“取消”（不解决）状态。按照目前 CQ 管理 Bug 的流程，它一定是开发取消的此 Bug，认为不是 Bug 或是这种问题不用改。这种状态的 Bug 欧也是必须关注的。

此 Bug 是关于 U 盘从设备中导出数据的过程中，用户拔了 U 盘，导致界面不动了（宕机）的问题。点击该 Bug 的变更历史记录，一条条往下看，记录意思大致如下：

2007-5-15 10:30 程慧（测试工程师）提交了此 Bug。

2007-5-17 9:00 元征（开发工程师）把此 Bug 重分派给开发主管陈成。

2007-5-19 16:00 陈成取消了此 Bug，并在解决说明中写道：“非法操作，修改代价大，无须修改”。

2007-5-20 10:05 程慧拒绝取消此 Bug，并说明“虽是非法操作，但软件不应该宕机，软件的容错性不足，建议开发处理之”。

2007-5-20 11:00 陈成又再次取消了此 Bug。

2007-5-21 9:00 程慧再次拒绝取消此 Bug。

2007-5-22 15:15 陈成第三次取消了此 Bug。

看得出来，他们意见不一，且双方都态度很坚决，开发认为不用改，测试坚决认为要改。

该如何解决这种典型的只会让开发与测试关系越来越僵的“踢皮球”事件呢？欧也想，就目前来说，测试是代表用户的角色来看的，站在测试的角度，不能因为是非法操作引起的严重问题，而不修改，程慧的拒绝取消此 Bug 的态度是可以理解的。于是，欧也找陈成交流，陈成说，这是软件的一个 Bug，不改的主要原因有两个：一方面它是用户的非法操作，用户需对自身非法操作的后果负责，就像 Windows 操作系统中写 U 盘过程中，如把 U 盘拔下由此而引起的系统异常，U 盘损坏，是要用户自行负责的。另一方面，这种场景不太可能发生（概率低），如果修改，时间及系统结构上将受较大影响，不值得。

欧也很清楚，并不是测试提交的所有 Bug 都能得到解决，要解决一些难免的 Bug 是需要付出代价的。但对于此问题，是否有其他更好的方法来解决，以达到开发不用改代码，又能提醒用户不要这样做，或把这样做的后果说清楚。同时让测试工程师也能接受这样的处理。于是说：“建议在用户手册中做特别声明，不允许在数据导出过程中拔 U 盘，以免引起意外！”经欧也这么一说，陈成欣然同意，认为的确是要这样处理才好。

欧也回到自己的座位，把程慧叫过来说“你对此 Bug 的看法是对的，但日后类似这种取消的问题，我们没必要反复拒绝取消，与开发之间把 Bug 当皮球踢，没意义。我们目前的需求由开发编写，对于这种问题开发是不会轻易改变他们的看法的，这是目前这种以开发为核心的管理模式存在的问题。在这种情况下，我们应多想想其他办法解决问题，在开发不改的基础上又能让用户规避风险的方法是上策。”当然对于此 Bug，站在测试角度，是不能关闭的，即不能同意取消而关闭此问题，因为关闭的问题，意味着此 Bug 的生命周期已结束，将不会再引起注意了。最后经项目组评审，此 Bug 转为延期状态。

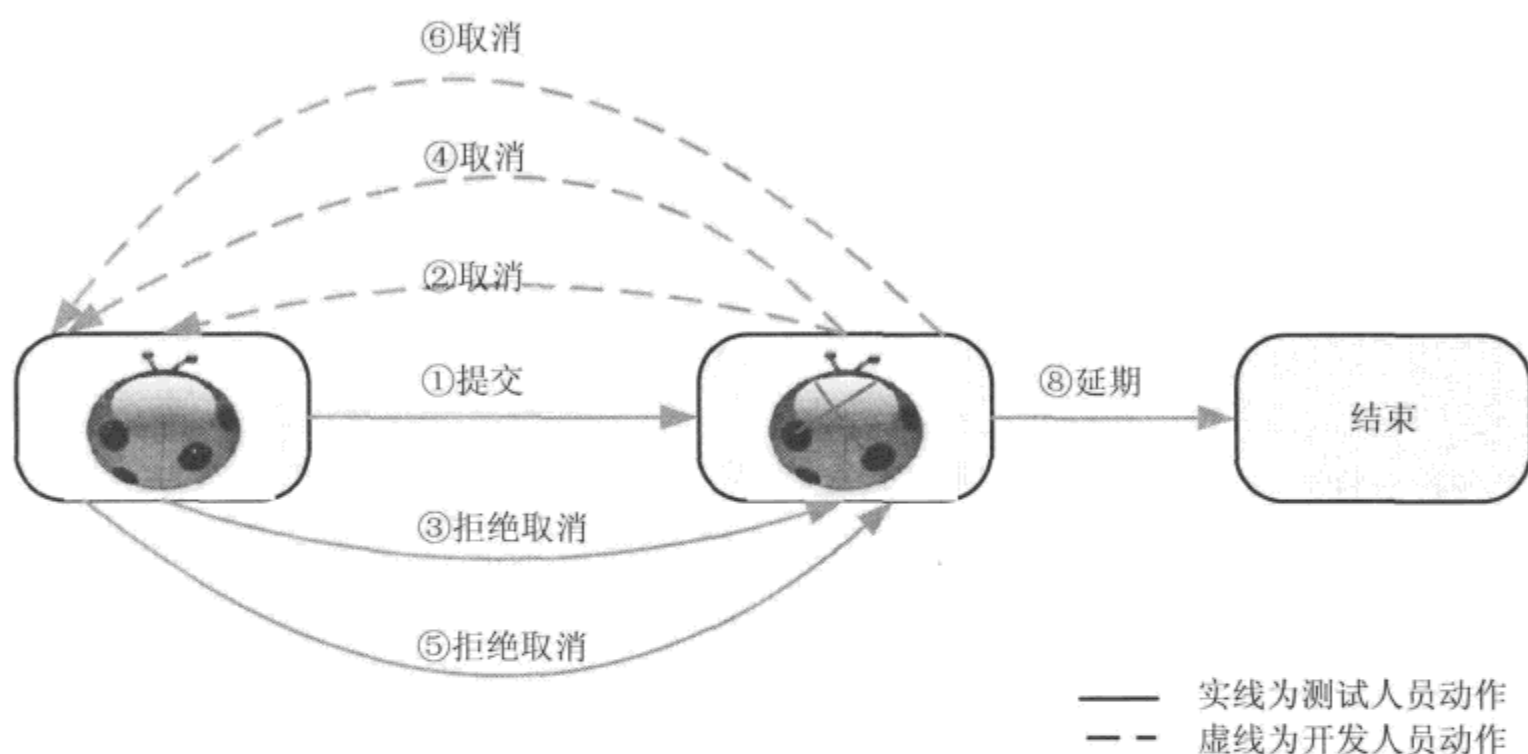


图 9-4 Bug 生命周期中的“踢皮球”现象示意图

案例中，相信读者已清晰地看到 Bug 状态与开发、测试人员的日常工作关系。在 Bug 的整个生命周期中，角色、权限、状态是相辅相成的，下面是一些在 Bug 生命周期定义中需注意的事项。

测试角色定义：如果公司的测试人员大部分都是新人，不熟悉产品业务，为了提高提交 Bug 的有效性，建议测试人员只录入 Bug，由测试负责人审核 Bug 后进行提交。但这种做法不足之处就是如果每天录入的 Bug 较多，对审核人员的处理可能会造成麻烦，导致 Bug 不能很快地传递到开发。当然好处也很明显，流程上使得测试负责人每天都能及时与测试人员交流，关注项目的 Bug 情况，对产品的质量状况了如指掌。

开发角色定义：一般情况下，Bug 可直接提交给负责具体模块的开发工程师，这样做的好处就是能使 Bug 的响应处理及时，同时可抄送给软件开发负责人。但有一点需特别注意，置 Bug 为“不解决”或“延期”状态的角色不能由开发工程师充当。处理此类 Bug

需谨慎，建议交给项目经理处理。如果公司流程清晰，经项目相关人员评审后，由软件开发负责人处理也是可以的。

软件测试目前在很多公司都是研发流程的最后一道关卡。常会遇到一些 Bug，软件开发负责人取消（不解决）了某个 Bug，但测试人员站在维护产品质量的神圣使命上，又不能接受不解决的处理。此时，需站稳立场，可以拒绝“不解决”，重新打开此 Bug。但这种情况下，最好是开发、测试、项目经理或有更高层管理人员一起参加，把风险分析搞清楚，然后再按讨论结论处理，以避免开发与测试之间不必要的冲突。上面案例中的“踢皮球”事件便是一个有代表性的例子。

对于延期的 Bug，什么时候再纳入更改计划，是我们需要注意的另一个点，否则延期的 Bug 将会永远石沉大海，对软件的质量没什么好处。笔者曾遇到过这种情况，测试人员曾提交了一个关于升级兼容性的严重 Bug，但由于当时版本急于发布，此 Bug 的更改无论对开发还是测试都影响较大，经相关人员讨论后，延期此 Bug，但延期到什么时候，并未确定。产品卖了两年后，突然有一天用户正好投诉了这个问题，要求马上解决，否则退货。此时，领导只好组织相关人员马上启动更改以解决此问题。有句话说“不怕一万，就怕万一”，群众的眼睛是雪亮的，在软件的质量上，来不得半点侥幸心理。

对于关闭的 Bug，意味着此 Bug 的生命周期已结束。开发人员解决一个 Bug，有没有彻底解决此 Bug，或解决了此 Bug 却引发了新的 Bug，这是回归 Bug 的策略中需重点考虑的。

9.1.4 约束的力量——Bug 管理规范

通过前面 Bug 管理工具的选择及 Bug 生命周期状态流转的设计，相信读者对 Bug 管理工具的应用已有了认识。但如何有效地与日常工作结合起来，使大家在使用工具上面有一个统一的认识，对 Bug 的处理进入一个有序状态，需要一套规范的约束。下面就规范中需要定义的内容结合案例进行介绍。

1. Bug 级别定义

严重度级别的定义，主要目的是统一全体对项目 Bug 严重度的认识，在工作中大家使用统一的标准来判断，以使不同级别的 Bug 能及时得到不同的处理。Bug 的严重属性也是查询测试结果中高频关键字之一。如表 9-3 所示是一种常见的 5 级定义方式，根据测试产品的不同，做局部改进后，便可订制出自己的 Bug 级别。

表 9-3 常见 Bug 级别定义

Bug 级别	特性描述
1 级致命错误	程序引起的死机，软件崩溃，死循环
	导致数据库发生死锁
	重要功能丧失
	输出数据错误，容易引起用户财产损失或法律纠纷
2 级严重错误	业务流程错误
	功能不符
	程序接口错误
	数值计算错误
3 级一般性错误	界面显示不清楚，难于理解
	打印内容、格式错误
	删除操作未给出提示
	拼写错误
4 级次要错误	UI 显示风格不统一
	操作复杂或提示不友好等
	帮助说明描述不清楚
	系统处理未优化，如界面刷新有较明显的闪烁
5 级测试建议	需求没明确定义的建议性处理意见
	符合用户常规使用习惯的建议

或许，如表 9-3 所示的 Bug 级别定义你已觉得够用了，也或许你面对的软件比较特殊，关注重点不仅仅列出的这些，再或者你已意识到此表中定义的局限性。无论如何，让我们一起读下面的案例，读完后相信你会有另一番想法。

【案例】与 Bug 级别定义不当有关的事件

一天，软件测试部经理召集全体测试人员说一位软件测试工程师提交的一个 Bug，由于级别定义低了，导致开发人员没有及时修订这个 Bug，而此功能正好又是生产必须要用的，即影响了生产的直通率。具体地说，这个 Bug 是这样的，产品的序列号在新版本上第一次不能保存成功，第二次输入后才能保存成功，但生产工艺有明确指示，只设置一次，工人在生产线操作是严格按工艺要求来做的，最后的结果可想而知。也因为这样，软件需马上更改此 Bug，验证后版本重发。

接下来在分析此 Bug 的防控措施时，我们发现测试人员两个月前就提交了此 Bug，严重性级别置为“一般”。由于当时的版本发布原则中有定义“严重以上 Bug 不能存在，一般性 Bug 可以遗留 2%~5%”，此 Bug 就这样不幸被遗留了。读者朋友，按照表 9-3 中的定

义，你会为它置什么样的级别呢？

一番反思后，我们认为当时所用的 Bug 严重性级别定义规则太单一，仅考虑了软件的功能，没有考虑用户的使用频率、用户体验（易用性等），是否影响公司品牌形象，是否对法律法规方面有影响等。也就是说 Bug 级别的定义不能仅仅从功能角度这条直线出发，要从多个角度出发，综合考虑。就像一个二维表，从不同角度出发，同一 Bug 可能得出的严重度不同，然后取其中最高的一个。这样可以保证此 Bug 解决的优先级会高。下面是根据案例中出现的疏漏，改进后的 Bug 级别定义，如表 9-4 所示。

表 9-4 Bug 级别二维定义

Bug 级别 影响面	1 级致命错误	2 级严重错误	3 级一般性错误	4 级次要错误	5 级测试建议
法律法规	不符合行业标准的法律、法规定义，可能导致人身伤害、财产损失等				
功能性能	程序引起的死机，软件崩溃，死循环	数据丢失或破坏、业务流程错误、程序接口错误等主要功能或性能丧失或受到影响	次要功能或性能丧失或受到影响，如界面刷新慢、打印报表格式错误等	界面个别字符串显示不全，但不影响整体意思的理解	需求没明确定义的建议处理意见
用户体验		严重影响用户使用。用户容易察觉到，可能引起用户严重不满，导致用户投诉	影响用户使用，用户容易察觉到，但不会引起用户严重不满，导致投诉。如 UI 风格不统一、提示不友好	不影响用户使用，用户很难察觉或无法发现的缺陷	符合用户常规使用习惯的建议
生产效率		严重影响生产效率和直通率			
品牌形象		明显影响公司品牌形象和公司声誉	可能对公司品牌形象和公司声誉有直接影响		
使用频率	用户高频使用的核心功能	用户常用功能	用户不常用，但给用户使用的功能	不影响用户正常使用，但存在缺陷	

2. Bug 单填写模板设计

对 Bug 的录入、解决说明、回归记录 3 个主要方面的填写进行介绍。如图 9-5 所示是能反映这 3 个方面内容的 Bug 信息界面。

图 9-5 Bug 信息界面

1) Bug 录入约束

无论用自动化的工具，还有是用 Word、Excel 来记录 Bug，一个 Bug 的核心要素是不能少的，不妨称它为 Bug 描述的三要素。

- **标题：**标题是对 Bug 言简意赅的描述，简述这个 Bug 是什么，告诉他人发生了什么事情。在一个团队中，可以在 Bug 的管理规范中定义一个标题的相对原则，如 Bug 的标题一般情况不超过 20 个汉字。
- **Bug 的详细描述：**对 Bug 发生的环境、条件、现象进行详细地描述，是标题的详细扩展。
- **产生 Bug 的步骤：**按步骤把产生 Bug 的路径写出来，各步骤中关键描述操作软件的动作，而不关心每一动作后发生了什么事情（发生了什么事情是详细描述中说的）。步骤要写清楚，简洁无误，让开发人员按你写的步骤能重现此问题，这是对

测试人员最基本的要求。更进一步,可以做到以最短路径重现问题,并且写在“步骤描述”中,相对于前面的基本要求,测试人员需要花多一些时间,特别是一些隐藏比较深的问题。Bug 的最短路径,对开发人员分析定位问题有很大的帮助,对提高测试人员自身分析问题的能力也是一种很好的做法。

如图 9-5 所示是某 Bug 管理工具的 Bug 录入界面, Bug 填写模板的设计可与要录入的字段结合起来。

小贴士:

提交 Bug,对于测试人员来说是再高兴不过的事了。录入 Bug 的过程心情是激动的,单击“提交”按钮的一刹那就更激动人心了,因为一单击“提交”按钮,就意味着宣布某某 Bug 开始诞生,无声地在说开发同事你们要注意(工具可以自动发邮件通知相关人员)。此时,亲爱的测试朋友,请按捺住激动的心情,再坚持几秒钟,把所填内容重新检查一遍, Bug 描述晦涩吗?是否还有什么条件或注意事项没有写出来?按所填 Bug 步骤重新执行一遍是否可以重现此 Bug?如果回答都为“是”,再请放心单击“提交”按钮吧(因为有些工具 Bug 提交后是不能修改的,一旦有错,将影响后续人员的工作)。

2) 故障解决说明填写

“故障解决说明”由开发人员解决问题后填写,记录 Bug 是如何解决的,更改是否对其他模块有影响,并给出测试建议。这样做的好处是能让每一个 Bug 的更改有详细的记录,相关审核人员在库上审核时,能清楚看到更改是否充分,特别是时间长了,能给问题的追溯带来方便。可通过配置或二次开发设计符合公司要求的管理工具。

下面对故障说明填写模板的内容进行介绍。

- 问题分析:给出故障原因。
- 更改方案:给出解决的方案描述,适当时候可直接给出代码修改片段,或涉及的文件名、代码行。
- 受影响分析:对更改涉及的模块或功能做影响说明。
- 测试建议:根据更改点,建议一些方法或路径来进行回归测试。

3) 回归测试记录

在项目研发过程中,开发解决一个 Bug,出现解决不彻底或改出新问题,是常见的现象,但是如果测试回归 Bug 时没有及时发现,遗留到后续版本或用户端,是一件让人心痛

的事。回归 Bug 时需写清楚回归策略，包括回归的路径、测试的数据（Bug 管理工具中可贴附件）等。定制回归测试记录的规范，一方面约束回归测试者写下测试思路及回归的路径；另一方面，记录的回归信息，能让审核人员判断回归是否足够充分，是否有必要采取补充测试（每个 Bug 都需写清楚回归策略，比较耗时间，但这种严谨的工作细节要求是需要的，也是值得的）。回归记录需体现以下内容。

- 验证的版本：回归此 Bug 的当前版本是什么。
- 回归策略：写清楚采用了哪些方法回归此 Bug，覆盖了哪些路径、用了哪些测试数据等。
- 验证结果：描述回归测试的结果，是通过还是不通过。通过可以关闭此 Bug，不通过则需重打开。

9.1.5 Bug 库的应用杂谈

Bug 库中积累下来的成千上万个已关闭的 Bug，或仍在沉睡的 Bug（延期状态），它们不仅仅是我们过去工作的成果，更是众多测试人员知识沉淀的结晶，为测试人员之间测试经验、技巧的传递提供了很好的平台。也为总结分析 Bug，推进开发、测试流程的改进提供了有力的素材，这在于你如何利用它、开发它。下面是一些常用的场景。

1. 为新人快速进入测试角色提供了一种捷径

记得刚开始做测试时，眼看上班两天过去了，还不知道什么样的问题是 Bug，看着其他老同事，也在测试同样的软件，但他们提交了一个又一个 Bug，心里挺着急的。于是问了旁边的一位老同事，他叫笔者先看看别人提交的 Bug。于是笔者一口气看了十几个 Bug 单，有些已改好了，有些是刚提交的还没改，便按 Bug 单上的描述来做。这样实践之后，大有“茅塞顿开”的感觉，结果在第三天的下午，开始提交了第一个 Bug，还是严重 Bug（是一个系统日期设置控件问题，日期控件响应了 Delete 键，居然把固定的日期格式标志也删除了，如 YY/MM/DD 中的“/”），也就在学习他人提交的 Bug 时，理解了什么叫“逆向思维”，以及如何结合具体工作来应用。

后来，当笔者成了老员工并有幸成为新员工的入职引导人时，在新员工的指导计划中，专门加入一条“回归 Bug 库中某项目的问题”的任务，以此为契机帮助新员工快速进入工作角色。

2. 测试经验与技巧沉淀的宝库

有一天，笔者与另一个测试同事小 A 一起在重现一个问题，在“备注”编辑区输满保

存后，再进入查看发现最后1个字符会显示乱码。由于当时需求并没有定义“备注”字段的最大长度，开发实现时给了128个字节，而小A说可以输入129个字符，但最后的字符显示乱码。在协助重现时，发现小A在输入字符过程中，能很快数出输入了多少个字符，觉得挺新奇的。后来看了她提的单，从她描述的操作步骤中发现一个很有用的测试小技巧，如图9-6所示。按顺序输入一串10个数字，然后再数其中的某一个数字共出现多少次，则很快可以得出总共输入的字符个数，这也便是数学中的分组计算法。关于这个Bug，更有趣的是，在最后一个字符时输入全角的字符，使得保存后再进入查看时显示为乱码，是一个逆向思维巧妙应用的例子。

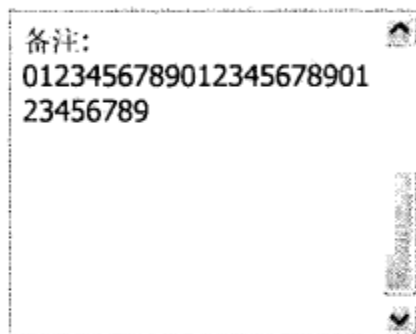


图 9-6 编辑区输入测试技巧示意图

这仅是一个典型的例子而已，还有很多Bug，沉淀着各方面的测试知识，或经验或技巧，值得我们学习、借鉴。

每到年终总结时，在很多人的总结上都会出现诸如经验不足、前面项目经验传承不够、测试之间的经验交流太少、培训不足等描述。下面就如何有效解决这些问题提出具体措施。

测试人员几乎每天接触的就是那些又恨又爱的Bug了（恨是因为它是软件的蛀虫，影响了软件产品的质量。爱是因为它是测试成绩的一个有效见证）。这些Bug中，只要用心去分析、总结，常会发现具有代表性的、开发常犯的错误。测试人员可以在日常工作中把这种Bug记录下来，并在小组例会或集中做不定期的测试经验交流。这样的总结对日后新人的快速成长很有帮助。收集问题后，可要求测试人员自己来宣讲，这样对测试人员也是一个总结的机会。要给大家宣讲，肯定会迫使自己去分析清楚问题发生的原因（设计原理），理清当时发现此问题时的测试思路，并提出日后类似问题该如何测试效果会更好。通过不断积累可总结出一系列的经验。

对于这些讲解过的经典Bug，把相关分析录入累积库中，可以用表9-5作为模板来记录，放入平台配置库，方便大家相互学习，促进团队的共同成长。

表 9-5 经典 Bug 分析记录

序号	Bug 编号	问题描述	问题类型	提交人	测试分析	分析人
1	11706	自定义报表名称与固定名“报表单”没有空格，英语、法语等多语言在翻译后不符合当地用户的使用习惯	本地化测试	Sunny	测试多语言的各打印标题是否符合本地化用户的使用习惯	Jack
2	10589	手机备忘录中新增一条记录并保存，提示“保存成功”后，马上断电关机，重开机后发现刚输入的记录丢失	数据保存	Carl	提示“保存成功”，但数据并未真正写入 Flash	Carl
...				

3. 测试人员相互学习的平台

无论是 B/S 结构，还是 C/S 结构的 Bug 管理工具，它们基本都是采用数据库来保存 Bug 信息的，从表 9-5 也可看出。这样，通过配置，项目组的全体人员通过网络都可以随时查看到 Bug 库中的所有 Bug。一旦测试人员提交了一个 Bug 后，大家便可同时看到此 Bug。不仅可以帮助你熟悉业务，同时也在学习他人的测试方法、思路，是无形中提高测试能力的一种高效方法。下面便是一个学习案例。

【案例】与操作系统及输入法都有关的一个 Bug

Bug 描述：某 Windows 应用程序正在 Windows 2000 中文操作系统上运行，当切换到紫光拼音输入法后，程序异常终止。

我们暂时先不探讨问题的原因，就此 Bug 的发现，用什么方法可以测试出来呢？是容易发现的问题吗？后来测试发现同样的测试组合（相同版本的测试软件与紫光拼音）在其他平台上运行，如 Windows XP 下并无此问题，最后分析出是一个测试平台（操作系统）与第三方应用程序之间运行兼容性的问题。这也告诉我们在测试 Windows 程序时，不仅需考虑测试软件与系统平台的兼容性，还要考虑测试软件与第三方软件之间运行的兼容性。

4. 测试人员与开发人员之间交流的桥梁

围绕 Bug 的生命周期，开发与测试就同一 Bug，将会来回地处理，直到 Bug 被关闭。开发与测试除了口头交流外，最常的交流也就在此了。与开发人员的交流也是一种学习的方法，可以快速提高对软件实现原理的分析能力，下面是案例之一。

【案例】某软件编辑区输入单引号保存时程序崩溃

背景描述：某 Windows 数据管理应用程序，采用 C# 开发，后台数据库采用 SQL2005。

Bug 描述：新增记录时，“姓名”字段中输入单引号（'），保存后程序崩溃。

看到这个 Bug 后,读者朋友,你会想到什么呢?如果熟悉 SQL 2005 原理的话,或许你已想到了单引号是 SQL 的分隔标识符,而笔者没有那么幸运,第一次遇到此问题时,是在执行功能测试用例时无意间发现的。后来与开发交流、分析原因后才搞清楚是因为开发人员进行数据注入库中对特殊字符忘了转义,程序抛出了异常。这些细节处理,有经验的开发人员有时还会犯错误,新手就更不用说了,通过与开发交流,了解了 Bug 的产生原因,增长了知识。

9.1.6 处理不可重现的 Bug

理论上来说,不可重现的 Bug 是没有的,但在我们实际的测试工作中,不时会遇到发现了 Bug 但不能重现它。

Bug 的出现与发生的条件密切相关,有些条件非常苛刻或严密。不能重现的 Bug,我们常会根据其出现的概率,判断它为“经常发生”或是“偶尔发生”等。殊不知,不能重现它,是因为我们还不清楚它的触发条件,后来与开发人员一起分析,并由开发人员解决后,才恍然大悟。下面案例中介绍的便是一个典型的“不可重现”的 Bug。

【案例】修改 IP 地址退出设置界面时,PPC (Pocket PC) 掌上电脑 (智能手机) 偶发宕机

背景:某手持 PPC 设备可作为一个小型服务器,与 PC 通过 USB 接口连接进行通信,把设备上的数据同步到 PC 接收端,如图 9-7 所示为物理场景。

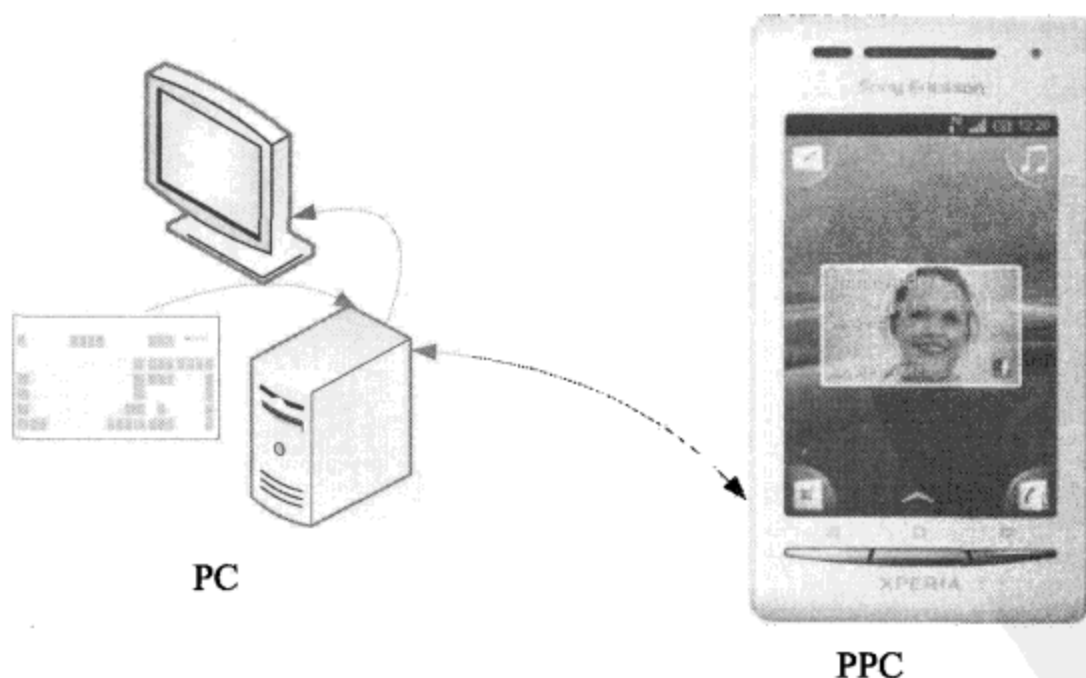


图 9-7 PPC 与 PC 通信物理场景图

需求：PPC 端可以设置通信对方的 IP 地址，与对方建立连接后，在状态栏将马上显示表示通信可正常工作的图标。当智能手机与 PC 数据接收软件正在工作中时，通信图标处于忙状态（界面可见其每隔两秒闪烁）。当两者未连接时，用一个黑白图标表示。

偶发 Bug：Bug 库上测试工程师 J 提交了一个与通信设置相关的 Bug。Bug 的描述是“在通信设置界面，修改了 IP 地址后，点触‘短信息’快捷图标切换界面时，界面宕机（死机）了”。

为了重现此 Bug，测试人员小 J 重复操作不下 100 次，只出现了两次，只好定义此 Bug 为“偶尔发生”。

分析定位：如此严重级别的问题，必须解决。于是，开发经理安排了一个排错高手 M 来分析此问题。首先 M 用他的专业工具调试（其实也说不上什么专业工具，仅是打开后台调试信息而已），然后要求测试人员重现，以期重现问题后捕获出现问题时的相关信息。小 J 真还神手，只重现了两次，第二次操作时即重现了此问题，与第一次发现后的重现容易多了。小 J 于是意识到，此问题的发生概率应该比较高，不属于偶发。

问题重现出来，开发工程师 M 当然高兴，就把设计信息放在自己的工作 PC 上，聚精会神地分析、咀嚼着那些大部分都是数字指令的信息宝贝。两个小时后，M 终于发现了代码中线程可能出现死锁的地方，于是在该处设置断点，并亲自按 Bug 的重现路径操作，看问题是否与定位点一致。然而，M 操作了 50 多次后，并未重现问题。于是又请测试人员小 J 前来重现，但奇怪的是小 J 同样是操作了 50 多次后，仍未出现。开发工程师 D 马上意识到是“测试环境”的问题，于是在小 J 重现过的测试环境（其实就是借用小 J 的 PC 环境）中再次挂上调试信息，果真不出 10 次又重现了该问题，且程序正运行于断点处，这也就证明开发工程师 M 的分析猜测是正确的。

最后的结论：在 PC 配置相对紧张的测试环境（如内存小于等于 1G）下，进行智能手机与 PC 数据接收软件的通信，容易出现多线程的死锁现象。PC 的配置越低，越容易出现，如在 256M 的内存配置环境下测试，此 Bug 将是必发。

小贴士：

PPC 是 Pocket PC 的缩写，是基于微软 Windows Mobile 操作系统的一种掌上电脑。

从上面的案例中，聪明的读者朋友，也许已感到这种场景似曾相识。下面是被我们视为“不可重现”Bug 的一些处理方法分享。

- 保留现场：如果问题出现，且是严重的（不能重现的 Bug 常都是严重以上的），一定保留现场，请开发人员过来察看。无论下次能不能重现此问题，对问题进一步分析都很有帮助。
- 追溯测试场景：细细回想，问题发生前你所做的操作，如点击按钮的速度，当时的软件状态，包括在什么界面或对话框、当前的数据、软件运行的环境配置等。
- 着手分析：一个比较好的做法就是要求软件中加入输出运行日志信息的功能，如日志中记录用户操作的路径，或操作过程中触发的消息等，这个日志对解决偶发问题将大有裨益。
- 专事专办：不要让“偶然”蒙蔽了我们的眼睛，冲昏了我们的头脑，认为 Bug 是偶发的而长期搁置不处理。世上没有那么侥幸的事，在公司内部出现过的 Bug，在用户端一定也会发生，只是时间与频率的问题。所以要视其影响度，考虑是否需由专人处理这类问题，以快速定位解决。

9.2 用例管理

测试用例，是测试人员重要的工作输出之一，是为测试这个没有硝烟的战场而准备的粮食、枪支弹药。没有充足的用例，测试执行过程将会像无头苍蝇，到处乱闯，毫无头绪，测试的有效性将严重受挫。就像打仗一样，准备好了粮食、枪支弹药后，还得把它们管理起来，才能急人所需，发挥它们的作用。对于我们的测试用例，它们是一些文字上的描述，或一段代码，或一些脚本文件，都不是有形的实体，对它们的管理需采取适合它们使用环境的策略。下面是一些用例管理方面的问题，读者朋友你曾遭遇过吗？

- 找某一个小功能点的用例花了半小时。
- 不能快速得到用例执行情况的统计。
- 已有的用例不知何时丢失了。
- 给不出某条用例曾在哪些软件版本上执行过。
- 用例与实现不一致。

本小节将介绍如何通过用例的管理，改进测试流程以提高工作效率。

9.2.1 用例管理工具选择

用例的设计，最终要输出一条条的测试用例，这些用例依赖什么载体来管理，在于用

例管理工具的选择。测试团队在成立之初，为操作简单方便，一般会采用 Excel、Word 这样的流行文档编辑工具来管理。但它们毕竟不是专业的用例管理工具，连用例管理中最基本的一些问题都不能有效地解决。看了下面的案例后，相信读者朋友会有所感触。

【案例】找某一个小功能点的用例花了半小时

背景描述：某通信公司的测试团队，测试用例采用 Excel 与 Word 编写，并放在 VSS (Visual SourceSafe) 配置库上进行管理。

一天，测试经理问测试人员小 C：“产品体验人员反映，CM-200 项目中原电话本记录删除时有确认提示，但改进后的 CM-200+却没有，你查一下我们的用例中是否有这个用例？”这可让小 C 犯愁了，由于项目中各模块的用例分散在多个 Excel 或 Word 文档中，CM-200+是原项目 CM-200 基础上的改进项目，对于变更需求部分的用例又单独放在另一个 Excel 文档中。小 C 为查这个小功能点对应的用例，先后打开了近 10 个文档后才找到了电话本模块的测试用例。但此表中有上千条用例，一条条往下找，翻看了几百条用例后，两眼便开始昏花（此路不通）。改用 Excel 本身自带的“查找”功能，按关键字来找，过滤了几十个用例后才终于找到。而此时，半小时已过去。

测试朋友，曾遇到过这种情况吗？这种用例管理模式（见图 9-8），实际上就是一些里面放着用例的 Word 或 Excel 文档集，严格来说还称不上用例库。

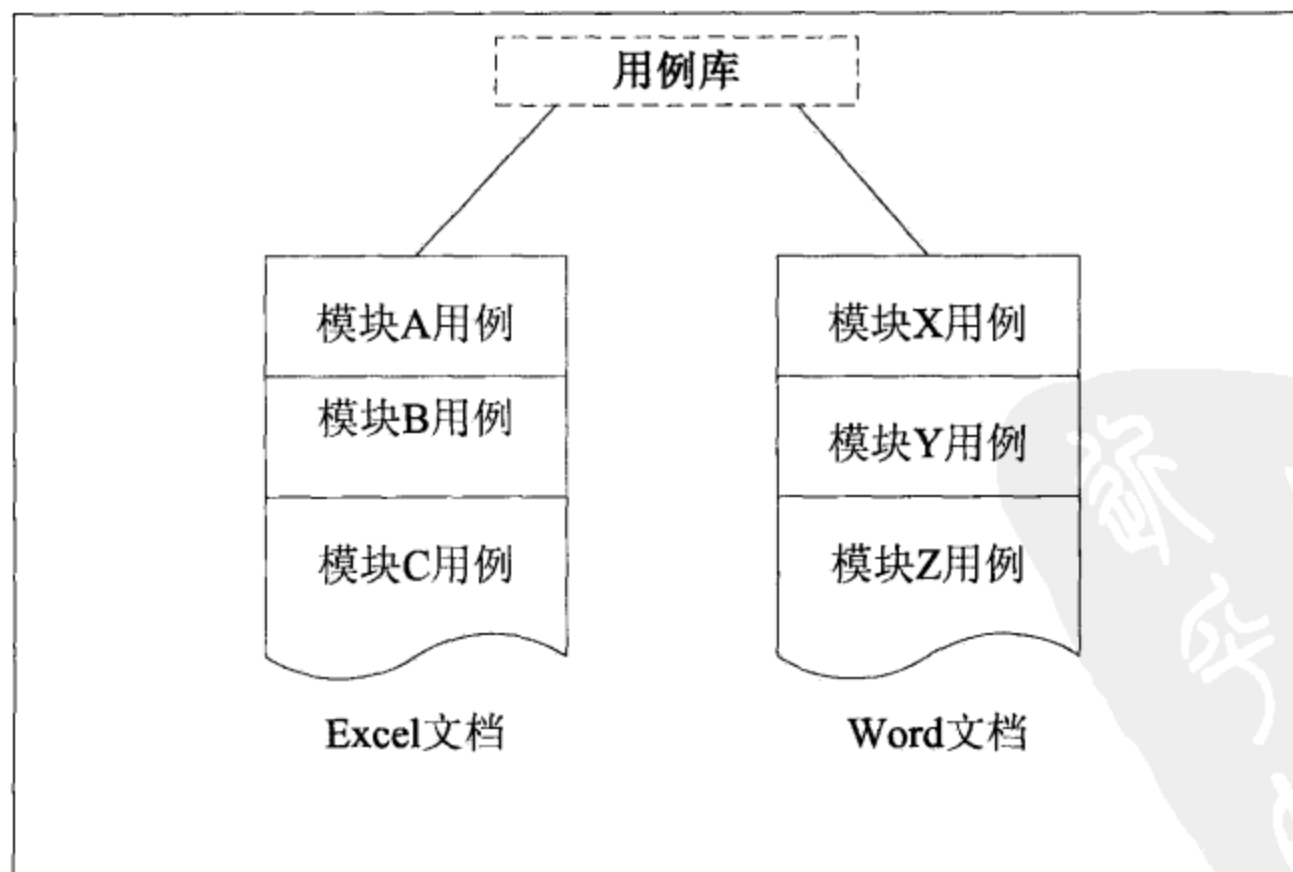


图 9-8 Excel 与 Word 文档组成的用例库

当用上面的模式管理用例时，还会遇到其他一些问题，例如统计已执行了多少条用例，已通过了多少条用例，在某个版本上执行了哪些用例，做起来会特别费劲。有一天突然有人问及你这些问题时，你可能会哑口无言，当然拍脑袋说出一个大约的数据也可以混过去。笔者甚至还遇到过有测试人员，居然对一个个用例人工数起来，真是到了“黔驴技穷”了，就连 Excel 中的筛选过滤功能都省略了。俗话说，“穷则思变”，当我们做一件事情需人工重复很多次才能完成且容易出错时，采用工具来代替是最合适不过的了。用 Excel 来管理用例，我们可对它做适度的二次开发，如用 Excel 中的宏功能来统计数据，能解决一些问题。但办公软件本身目标用途不同，一些重要的问题仍得不到解决，如用例修改后没有修订历史记录、没有版本的概念、Excel 始终保存的是最新的内容等。于是，对专业用例管理工具的考虑将成为必然。

用例管理工具如果按来源来分，与 Bug 管理工具类似，包括商业、开源、自主研发 3 个方面，如表 9-6 所示是常用的用例管理工具比较表，读者朋友可以根据所需选择适合自己的工具。

表 9-6 常见用例管理工具特性比较表

工具名称/特性	TestManager	TestLink	TestDirect
技术特性			
构架模式	C/S, B/S	B/S	B/S
支持平台	Windows linux	Windows linux	Windows
支持数据库	Oracle、Access、SQL Server 等	MySQL	Oracle、Access、SQL Server 等
功能特点			
附件支持	能	能	能
E-mail 支持	能	能	能
用户角色权限管理配置	能	能	能
报表、图表输出	能	能	能
导出数据	能	能	能
界面	友好	简洁	较好
使用约束			
来源	IBM-Rational 公司	开源	HP-Mercury 公司
安装配置	一般	简单	一般
可集成软件	可以和 Rational 的测试工具 robot、functional 等集成	可以与 Bugzilla 等开源缺陷管理工具集成	可以与 LoadRunner、Win-Runner 进行有效的集成

对于中小型公司，如果开源的用例管理工具能满足需求的话，那么从经济效益及灵活的可扩展性来说，TestLink 是一个不错的选择。如果是大公司，且在缺陷管理、配置管理、需求管理上已采用了配套的 IBM-Rational 或 HP-Mercury 集成产品，采用一条龙式配套软件中的用例管理的优点也很突出。就公司整体而言，容易形成规范的流程管理体系，技术支持也有保证。

9.2.2 用例结构与元素的设计

有了工具之后，需要考虑它如何与实际工作结合起来应用。其中，用例的描述要包括哪些元素，即用例的编写页面要包括哪些内容需首先确定下来。用例的内容通常保存在后台数据库中，是后续查询、数据统计度量的基础。

1. 用例结构

记得在小时候刚开始学习写文章时，老师就要求我们先把文章的大纲列好，然后再展开写，这样写出的文章条理清楚，主次分明。同样地，用例开始设计前，在内容组织上也需做好规划。用例库的组织可以以产品为单位，也可以以模块为单位，视需要而定。下面以产品为单位，结合开源 TestLink 用例管理工具提供的文件夹形式的管理，以及无限层级的用例组织形式进行介绍。

在默认情况下，TestLink 支持的测试用例管理包含 3 层，分别为 Component、Category、Test case。应用到实际工作中，我们可把 Component 对应到项目的功能模块，把 Category 与模块下的某功能点（function）对应。如果功能点下需再细化小功能，则再进行下一级的分层，以此类推，层层细化。最后一层就是具体的 Test case（测试用例），用不同的测试标题进行标识，如表 9-7 所示。

表 9-7 用例结构解释

TestLink 用例结构	改进后用例结构	实 例
Component	模块	电话本
Category	功能点	新增
Test case	测试用例标题	新增正常记录（有姓名及电话号码）
		空记录
		各字段内容输满
		编辑过程中响铃

如表 9-7 所示的用例结构在 TestLink 中的表现如图 9-9 所示。

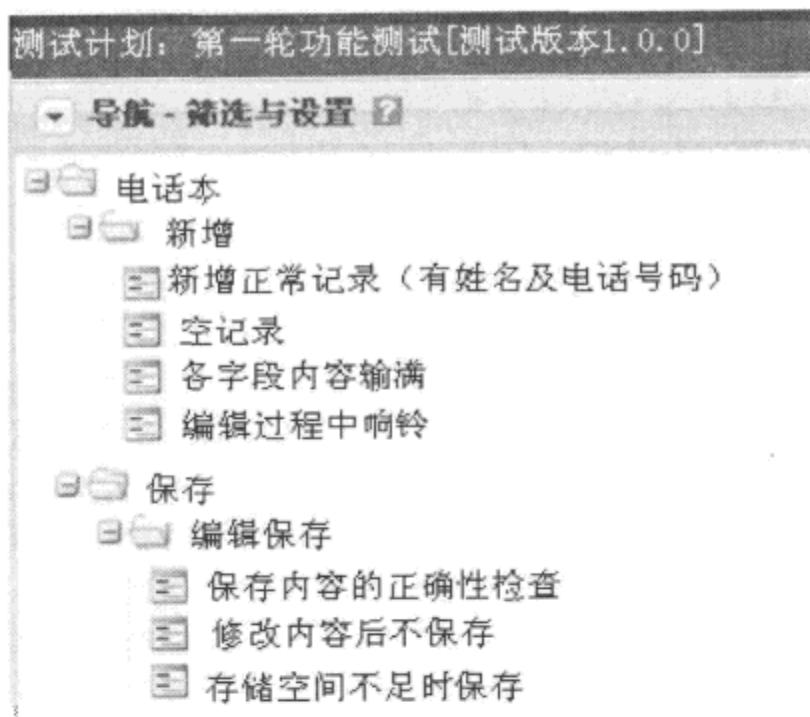


图 9-9 TestLink 中可定制的用例组织层次结构示意图

图中的用例组织结构，也叫用例树，根据测试对象不同，可以进行逐级分层，由大到小、由外到内，这是符合事物的解剖分析原理的。TestLink 工具本身支持无限深度的分层，但并不是分得越细就越好。据最佳实践总结，对于功能测试用例，分到 3~4 层比较合适，用例设置适中，可读性好，容易理解。

小贴士：

TestLink 由于其开源的优点，以及与其他开源产品，如缺陷库 Bugzilla、Mantis、配置库 SVN 等的集成兼容，已得到很多公司的青睐。感兴趣的读者可进入 TestLink 官方网站（<http://www.teamst.org/>）了解更多信息。

2. 用例元素

谈到用例元素，突然想起曾经做面试官时，有一次见过这样一道问答题，“设计一条用例，需要包括哪些内容？”笔者当时想，这对于有测试经验的测试人员来说，不就是送分题吗？由于是问答题，没有标准的答案，但必须要有包括的关键要点。当初以为自己已能答得很充分了，但与一些应聘者交流后，他们在不同行业、不同环境下工作所带来的一些思路让笔者觉得很新鲜，值得学习。下面结合 TestLink 中用例的编辑界面（见图 9-10），介绍各元素的特点，以及填写的要求。

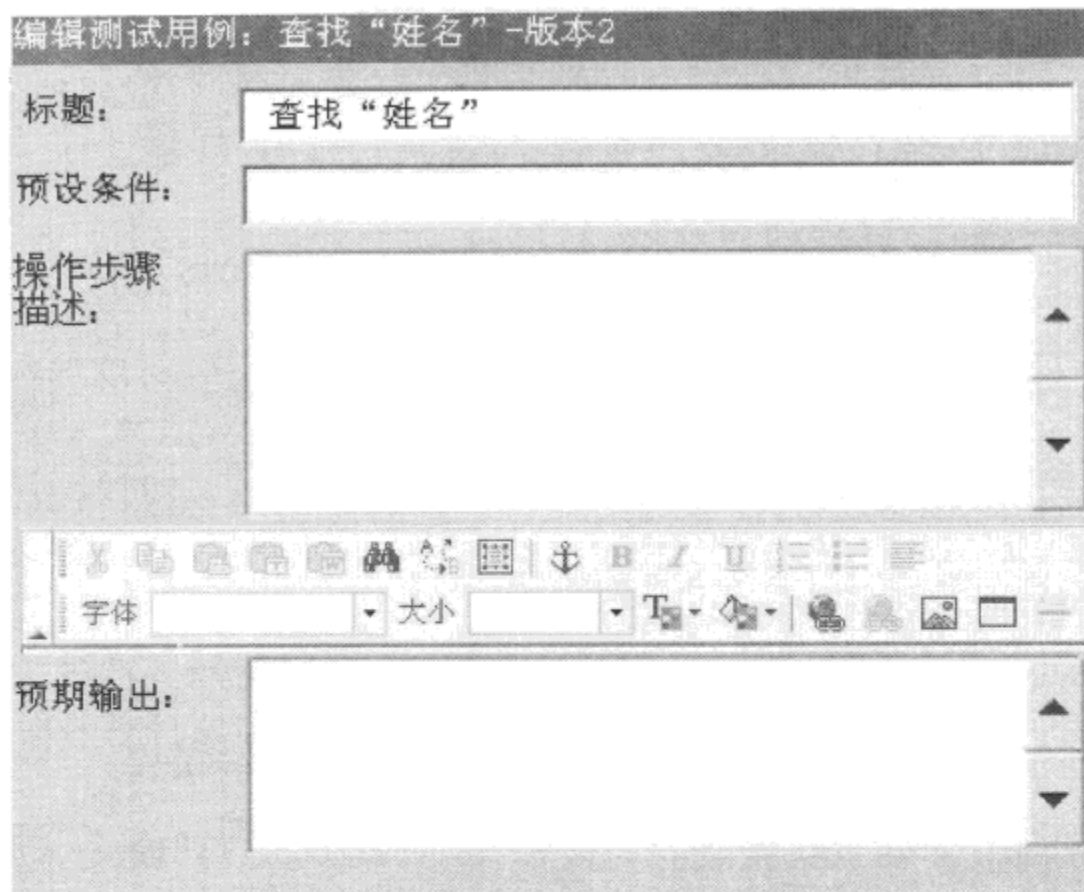


图 9-10 测试用例编辑界面

- 测试标题：简短描述用例，告诉审核及测试执行人员，此条用例的测试对象。这是最小的检查点。
- 预设条件：规定测试用例执行的前置条件如果用例的预期输出与测试的条件相关，则需要在此处明确，以避免预期输出不唯一。
- 操作步骤描述：用例执行时的操作顺序填在此处，可把动作与输入数据一起描述。
- 预期输出：用例执行后期望得到的结果。

除上面构成一个用例的 4 个关键元素外，下面几个元素，是测试管理工具自动记录的（如果采用 Word、Excel 管理用例需人工增加）。

- 用例 ID：工具自动增加。
- 测试时间：工具自动增加。
- 测试人：根据用例的所属分配，工具自动增加。
- 测试结果：每条用例执行后都有一个结果，由测试执行人员执行后置不同的结果，包括通过、不通过、阻塞、未测试 4 种情况。

3. 几个重要的用例属性

1) 用例的优先级

什么样的用例优先级高，优先级定义为几个级别较好，需针对不同的软件功能特点，

不同的用户场景，分别对待。优先级的表示方法，可用 A、B、C、D 来表示；也可用 0、1、2、3 来表示。一般情况下，定义为 3 个级别，分别为高、中、低，也可在此基础上再加上其他定义细分。

- 优先级高的用例：一定是用户常用的重要的基本功能，如核心业务功能。就拿大家都熟悉的 Word 文档编辑工具来说，“保存”功能就是它的核心功能之一。
- 一般级别用例：终端用户比较少用的基本功能，如我们都熟悉的手机中的“恢复出厂设置”功能（由于这个功能的特殊性，可能有些用户还不知晓自己的手机中有这个功能）。
- 低优先级用例：指一般不影响用户功能的正常使用、一些次要的用例，如界面 UI 检查、异常操作等的用例。

用例级别划分是否合适，将直接影响用例执行的先后次序及版本的稳定周期。“用户就是上帝”，对测试用例级别设置也一样，一切以用户的使用为中心，测试的重点（时间与精力）放在用户常用的基本功能上，也就是所谓的 2-8 黄金原则。把 80%的精力放在测试用户重要的常用功能上，用 20%的时间来测试那些次要的或是属于锦上添花的功能。

2) 用例的类型

每一条用例赋予一个“测试类型”值，一方面可方便数据的统计分析；另一方面也可以提醒设计人员用了哪些方法，评审人员也能清楚用例属于什么类型。有助于判断哪些用例考虑到了，哪些漏了。如表 9-8 所示是常见的系统测试类型，每一类型的详细解释可参考附录 A。

表 9-8 常见系统测试类型

序 号	类型名称	序 号	类型名称
1	功能测试	10	配置测试
2	界面测试	11	安装测试
3	易用性测试	12	卸载测试
4	压力测试	13	容错测试
5	性能测试	14	安全性测试
6	容量测试	15	随机测试
7	内存测试	16	文档测试
8	协议测试	17	在线帮助测试
9	标杆测试	18	回归测试

3) 用例设计方向

包括正向思维、逆向思维、相关影响思维，其中相关影响思维指考虑测试点在模块内部及模块之间有哪些相互影响的地方。这样划分的好处在于，一方面通过指派用例的设计方向，可提醒设计人员考虑的方向是否有遗漏，是否正向的用例过多，而逆向的用例根本没考虑到。另一方面方便数据统计，如正向分析、逆向分析及模块之间相互影响的用例各有多少，执行失败的用例是否与设计方向有关。简而言之，指明用例的设计方向，既为用例设计本身的全面性着想，也为用例执行后的结果分析提供有力的支持。

9.2.3 用例维护的设计

软件的维护阶段，是软件生命周期中跨度最长的一个阶段。当软件第一个版本正式发布后，随着市场的变化，用户需求的变化，经常需要对软件进行在线更改。这种更改由于是在用户正在使用的软件上面进行变更，因此在质量上不能有半点差错。这对软件测试也就提出了更高的要求，用例维护的正确性、全面性就是其中的一项。下面的案例讲述的就是关于用例维护的例子。

【案例】用例与实现不一致，是用例错了，还是 Bug？

背景描述：某公司对加入软件测试的新员工指导有一条特别要求，无论是谁，也无论学历高低、经验深浅，入职后的第一项任务就是执行测试用例，通过操作软件来熟悉产品业务。

Carl 是国内一所名牌大学毕业的计算机应用专业硕士研究生，加入某公司的软件测试部后，上班第一周安排执行 1000 条用户场景类的 A 级用例。用例是测试主管 Jenny 从用例库中筛选出来的，要求 Carl 执行每一条用例时，详细记录测试结果，执行完成后把结果反馈给 Jenny。

一周过后，Carl 按时完成了任务，Jenny 汇总了 Carl 提出的问题，总结如下：

- 确认是 Bug 的有两个。
- 用例与软件实现不一致的有 35 条。

第一点是之前测试遗留的 Bug（此处不展开讨论），第二点，Jenny 最后确认是用例描述不正确的问题。在目前用例库维护方面，因为存在以下几个问题才导致了现在的结果。

- 新增需求后的测试，是独立增加用例来测试的，新增的用例没有合入用例库，这是用例库中用例与实现不一致的主要原因。
- 需求变更后，用例更新不彻底，如更新了影响模块 A 的用例，但没有更新 B 模块

受影响的用例。

针对上面出现的问题，提出以下改进方法：

- 对于过时的用例，置于“无效”状态，在数据库仍保存着，但测试工具的界面上看不到（如果日后需求有变化，无效的用例可以再生效，只是这种情况应是很少的）。
- 用例库中增加“测试版本”字段，规范测试流程，在上面筛选用例、修改用例、新增用例。对于可独立的模块，直接在上面新增模块。
- 导入用例的功能仅开放给测试负责人。
- 用例更新后，导出用例进行评审，以加强分析用例设计的全面性。

用例的维护是一个持续性的工作，从版本的第一次发布，到最后此软件退出市场，软件的生命周期结束，才能宣布维护工作的结束。

小贴士：

用例如何在不同产品间复用？

用例的产品关联性：对于在线产品，常会遇到产品 A 需要新增此功能，产品 B 也需要，实现时也是进行相同代码的移植。对于用例设计来说，用例录入表单中可增加一个“关联产品”字段，所有关联产品都选上，以表明此条用例存在关联产品特性。当筛选产品 A 及产品 B 的用例时此条用例都能筛选到（采用用例管理工具可以做到）。

9.3 测试文档模板设计

通过前面章节的介绍，读者朋友已知道，测试工作的输出不止是 Bug，在一个全面完整的项目测试工作过程中，每一个环节都有测试输出的要求。如图 9-11 所示是一个从测试角度看的完整项目测试流程。其中测试执行阶段是一个不断反复（Bug 更改后需回归测试），不断迭代的测试过程，版本迭代的过程根据需要更新测试方案与测试用例。表 9-9 列出了每一环节需输出的基本测试工件。

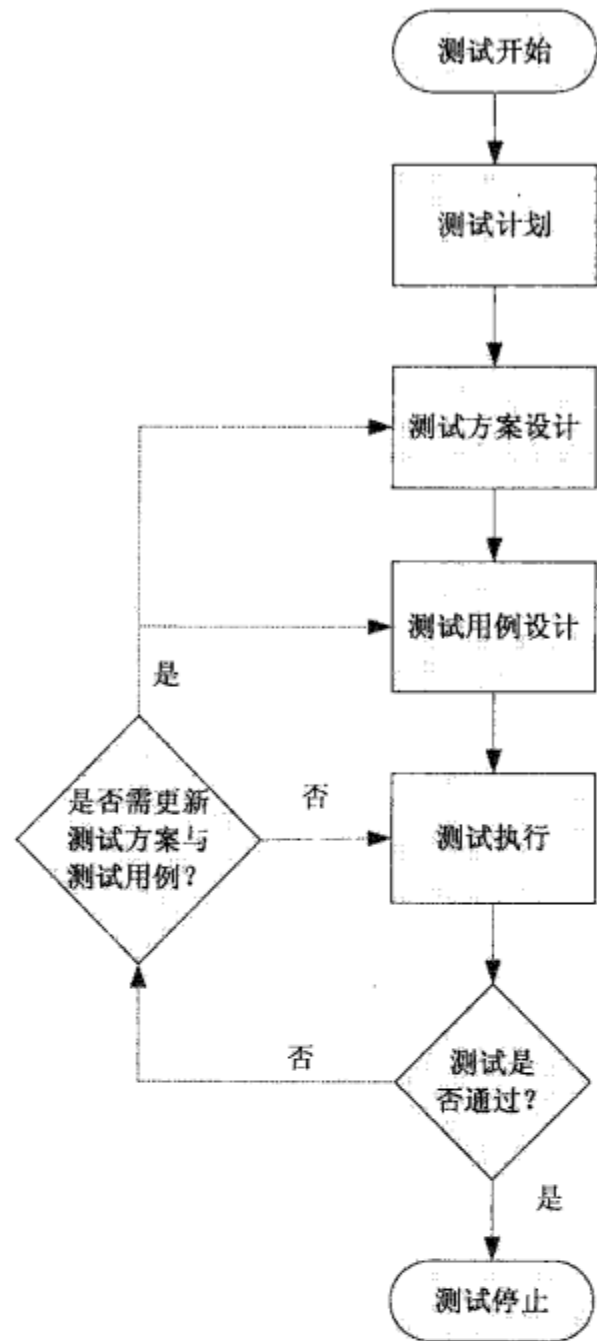


图 9-11 完整测试流程图

表 9-9 测试阶段与测试输出对照表

测试阶段	测试输出	备 注
测试计划	测试计划	根据项目需要，还可输出测试工作手册
方案设计	测试方案	
用例设计	功能测试用例、测试代码、测试脚本	不同种类的测试，用例设计的内容不同，如代码测试、性能测试，测试用例是代码或测试脚本
测试执行	测试记录、Bug、测试报告、测试总结、测试分析报告	

实战过的测试朋友都知道，说起测试文档的编写，该写哪些内容，结构如何组织，常让人不知所措。此时，很多人的做法是到网上搜搜，找一堆五花八门称之为模板的文件，然后东拼西凑，再加上一些自己的内容，形成一个测试文档（笔者也曾经历过）。然而，

只要你多加思考，有不少内容是可要可不要的（并不是文档越长越好），特别是一些自己都不清楚写那些内容有什么用的部分。

当公司有专门的测试团队时，需要有相关的流程规范来统一大家的工作输出，测试文档模板的设计便是其中的内容之一。根据表 9-9 列出的各测试阶段的基本输出，下面对测试计划、测试方案、测试报告模板的设计进行介绍（测试用例模板所含元素的设计可参考图 9-10）。

9.3.1 测试计划模板设计

一份好的符合项目需求的测试计划犹如量体裁衣，再好的裁缝师也做不到放之于四海皆准的标准尺码，同样地，测试计划也没有标准的答案。但同样是测试计划，存在着一些与项目无关的特性，即通常情况下它应包括的最基本的那些内容。通过回答下面的问题，可以找到答案。

- 事：要做哪些事？包括测试范围、测试对象有哪些？
- 工作量评估：评估完成项目测试任务需要的人力、时间。评估工作量时需初步确定方向性的测试策略，如需开展哪些方面的测试手段，包括代码测试、集成测试、系统测试、自动化测试等。
- 人：要完成项目的测试任务，需要哪些方面的人，需要多少？
- 测试里程碑定义：策划测试过程，测试分几个阶段？各里程碑阶段做什么事？输出什么？
- 设备资源：测试过程需要哪些设备、工具来支持？
- 其他计划：根据需要定义版本控制计划、质量控制计划等。
- 计划维护措施：定义计划变更的条件，如需求、软件设计的基线推迟提交给测试，测试里程碑受的影响等。

小贴士：

做人力计划时需考虑核心测试成员的业务能力、技术能力，新老员工的搭配比例。如果项目的新人多，需要考虑测试过程中的培训计划。

【样例】测试计划模板样例

测试范围：

列出待测试的方向性的子系统或模块。

任务与人员计划：

在项目的各阶段，根据测试的工作任务情况初步制订的人员需求计划（参考下表）。

项目阶段	测试里程碑	工作任务	投入人数	工作量	备 注

注：如果测试团队还兼做项目的配置管理、帮助文档的编写等工作（常见这种情况），则需在工作任务中列出。如果所需人员在时间点上不能到位，也需在备注中说明如何解决。

测试工具资源计划：

用表列出各阶段需要用到的测试工具及其他非人力资源，写明数量、来源。

培训计划：

各阶段需要安排的培训以帮助测试团队的成长，包括产品知识、业务知识、测试方法、测试经验交流等，或需要其他技术组支持的培训，都可列在培训计划中（包括培训与被培训的需求）。

版本控制计划：

定义与开发之间的版本发布原则，包括阶段性迭代版本的计划，发给项目组内外的临时版本必须由测试组发出，保证唯一的出口；开发提交版本时必须填写测试传递表（填写软件版本号及变更信息）；通过冒烟测试的注意事项等。

是一组与开发团队达成的关于版本控制的协议。

测试质量控制计划：

表明为了达到项目的质量要求需要用到的流程控制，如测试方案与用例的专家评审。测试执行阶段的持续测试。核心模块的代码测试等。

测试进度计划：

列出影响测试进度的可能因素及解决对策。

测试文档计划：

列出所有需要归档的测试文档及它们归档的时间点。

其他支持计划：

如果有需要就写出来，如项目配置管理的支持计划，软件帮助文档、手册编写的支持计划等。

9.3.2 测试方案模板设计

测试方案的设计，犹如开发对软件的概要设计。首先我们需清楚测试方案设计的目的，

要解决哪些问题。测试计划主要是测试管理上的问题。而测试方案设计的核心是为了达到计划中的目标，解决如何做的问题，是测试分析、测试策略制定、具体方法确定等一系列测试技术上的综合体现。

同样，测试方案的设计也不存在标准的模式，重要的是内容的表达是否解决了预期的问题。另外，通过测试方案的设计，最后提取出的测试点及由测试方法表现出来的思路，应能很容易导出测试用例，就像软件工程中提到伪代码。

方案设计的表达可分为3个步骤：第一步是测试对象的分析，第二步是测试点的提取，第三步是对各测试点对应测试思路的表达，如图9-12所示。一起实战的同事就曾这样说过：测试对象的分析犹如开发的概要设计，测试点与测试思路如同开发的详细设计，测试用例犹如软件代码。

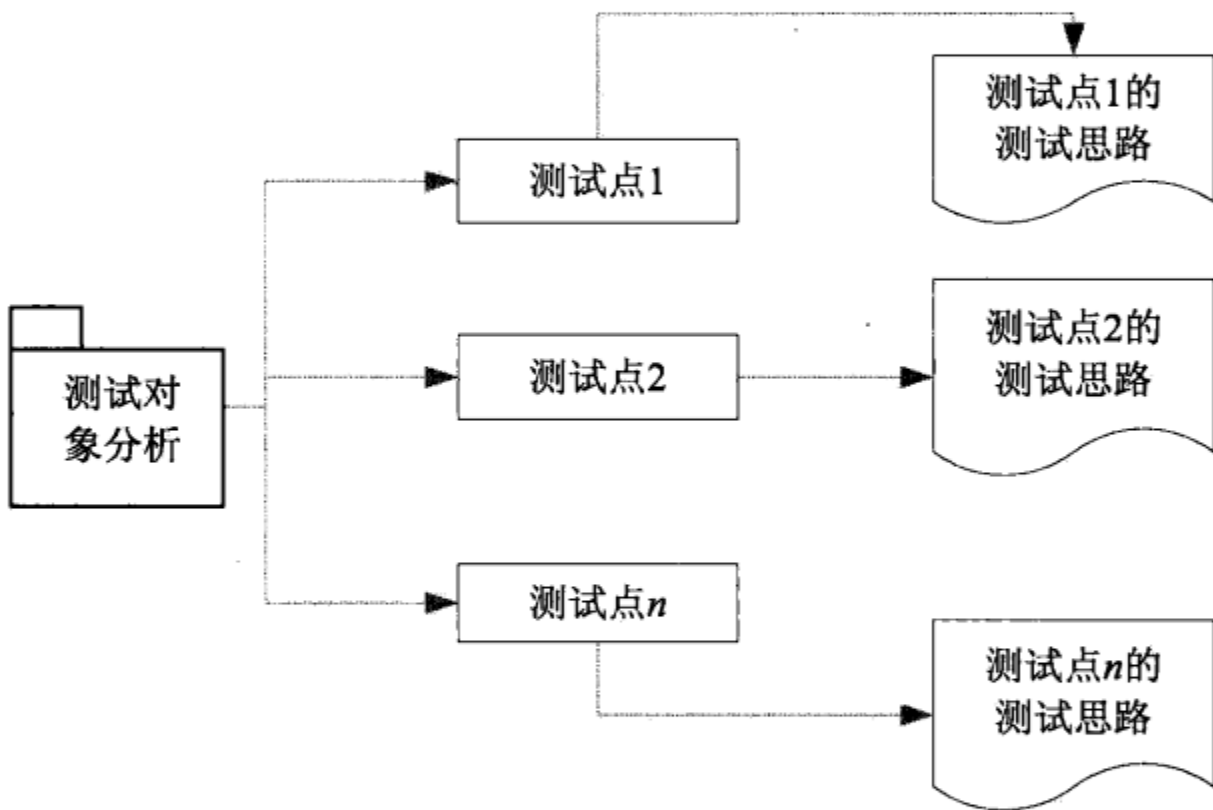


图 9-12 测试方案设计步骤示意图

测试方案设计中如何进行具体的测试对象分析，可参见第6章介绍的多个测试对象分析方法中的应用案例。

9.3.3 测试报告模板设计

测试报告，是测试工作的最后输出文档，记录着测试的结果，是一种结论性的文档。它对于软件项目有着重要的意义，是关注产品质量的领导层必看的文档。除了项目经理、部门经理等领导外，其他相关部门的接口人员也可能很关注，必要时还引用软件的测试报

告作为依据，来开展他们的工作。

关于测试报告的模板，在网上的资源很多，其囊括的内容也各种各样。笔者见过、用过不少测试报告模板，结合一些经验教训，总结了如下关于测试报告设计的关键点。

1. 严谨性

由于最后的测试报告需要有非常明确的结论，是通过，还是不通过（此处不讨论通过不通过的判断标准），因此报告中的结论词需特别谨慎。

2. 以数据说话

把判断通过或不通过的数据体现出来，以让他人信服。这些数据通常来自测试记录，如用例执行率、用例通过率、用例失败率、Bug 的解决率等。

3. 报告中体现的内容需简洁有力

做到不多说，也不少说。下面是一个关于测试报告中测试记录不严谨的事例，相信对读者会有所启发。

【案例】测试报告不严谨引起的投诉

背景描述：某软件运行在 Linux 操作系统的服务器上，由于开源软件中提供的打印机设备驱动受到限制，使其对不同品牌的打印机存在着兼容性的问题。软件所能支持的打印机是有限的，为了用户端购买打印机的方便性与正确性，每次新版本软件发布时，同时在测试报告中都会附加一份兼容打印机列表。

问题发生：新增打印机的测试完成后，在打印机兼容性的报告中，测试人员把打印机 HP Laser Jet 1050N，写成了 HP Laser Jet 1050。负责产品维护的同事拿到测试报告后，把兼容性列表中新支持的打印机型号很快告知用户。但用户购买后，说软件不能识别新打印机，不仅要求赔偿，还要退产品。

漏写一个 N，却造成如此大的影响！

就测试报告中要反映的内容，并不需要像计算公式那样一成不变，根据公司或测试的业务可以灵活变通。但就一般情况而言，包括的内容有：测试对象（具体到哪个版本）、测试内容、测试环境、测试工具、测试记录、测试人员、测试时间、测试结果、风险评估（特别需要有延期 Bug 的风险评估）等项目，其他可视具体情况增加。下面是一个测试报告模板的样例。

【样例】

测试对象：

指明测试的对象、版本号，例如测试对象是数据通信软件的 1.0.125 版。以便让读者

清楚本报告是对此版本软件进行测试的报告。

测试内容:

在报告中列出简要的测试清单,或者测试范围。测试内容实际上是测试方案中需首先明确的。在报告中只做简要说明,以便让读者清楚,是针对哪些项进行的测试报告。

测试结论:

一个测试报告,必须要有结论性的说明,如测试通过或不通过,并简要地说明其理由。这个结论很重要,是测试报告的精华部分,大部分领导只关心这个结论。

测试组织:

列举参与测试的团队成员名单,包括测试负责人、测试工程师。

测试时间:

说明测试阶段的具体开始与结束时间,所消耗的实际工作量(项目经理、测试经理非常关心的内容)。

测试环境:

包括测试软件运行的软件与硬件环境,软件环境包括操作系统及版本号、配套软件的版本信息等。硬件环境中需列出硬件的设备编号,如果测试结果与设备的有效期有关,还需列出它的有效期,以证明测试结果的有效性。测试环境的清楚说明主要是能再现当时的测试结果,达到可追溯的目的。

测试工具:

记录测试过程中所用的测试辅助工具,包括其名称、版本号,以便于再现当时的测试过程。

测试记录:

指用例的实际执行结果,测试记录中必须包含用例执行的执行结果(通常包括通过、失败、阻塞、未执行)、软件版本、执行时间、执行人。如果有必要的话(视项目的具体需要),测试执行过程中的中间数据也需作为测试记录的一部分。测试记录是软件在某一时刻运行状态的快照,是判断是否通过测试的有力证明。

风险评估:

不再解决的 Bug 或延期的 Bug,都有可能对产品在使用端的质量上造成影响。这些遗留的 Bug,必须通过项目相关专家的评估,并把评审结论体现在测试报告中,以预知可能存在的风险。

测试报告可以涵盖的内容还有其他,如可以包括测试对需求覆盖率的分析、缺陷分析、测试小结等。但最佳实践证明,这些内容仅是测试相关人员比较关心的,而测试报告汇报

的主要对象如项目经理、开发经理并不关心。并不是报告越长越好，一个内容少而精、数据科学且风格严谨的报告更受欢迎。

9.4 测试总结管理设计

如今，在职场中打拼的朋友们，谁要是说没有写过总结，真可谓是罕见了。提到写总结，对很多人来说，并不是一件轻松的事。特别是到了年终时，年度个人总结、小组总结、部门总结、绩效考核总结等一个接着一个。有一些公司，平时要求写的工作总结也不少，如日小结，周总结、月总结，一个都不能少。写总结，到底有哪些好处？将在下一节与读者一起分享。

9.4.1 写总结的好处

日常工作中有一个敏感而又现实的问题，同样是写总结，但主动写与被动写效果就不同。前者往往能收到意想不到的效果，为你打开成功的大门。下面是一则与主动写总结有关的职场故事。

【案例】主动写总结的职场故事

在加入某知名民营企业前，Jenny 已经是一家外资公司的测试主管，但是加入新公司后，仍然需从一个普通的测试工程师做起。Jenny 并不气馁，她相信“是金子，总会发光，会发光总会被发现”。入职一周后，便快速进入了工作状态，并开始负责一项具体的测试任务，两周后很漂亮地完成了经理安排的任务。由于初来乍到，Jenny 每天都在记录着遇到的问题，并向身边的同事请教，同时主动地写总结向经理汇报。经理自然地及时了解到 Jenny 的工作情况，在完成了第一件任务后，接着又安排了一个更大、更重要的任务给她。Jenny 保持着勤奋、积极、主动的工作状态，工作中遇到的问题都得到及时解决，一个多月后，超出经理的预期提前且高质量地完成了工作任务。从与经理的不断交流中，Jenny 已感到经理对自己工作的满意与信任。于是，Jenny 在第二个月的最后一周时，主动写了一个特别的总结并向经理提出“提前转正申请”的想法。成功绝非偶然，成功向来青睐有准备之人。一个星期后，Jenny 的申请得到批准。

Jenny 说，后来那位经理被提升为质量部总监，她自然也成了她手下的一名得力的中层干部。再后来那位总监自己开了家公司，Jenny 成了她的一位好朋友。她对 Jenny 的评价是，善于总结，善于思考，也善于表现。

读者朋友，从这个小故事中，相信你已体会到常写总结的好处了吧。概括地说，写总结一方面是给了自己在总结中思考、学习、成长的机会。工作总结是我们认识世界的重要手段，是由感性认识上升到理性认识的必经之路。通过工作总结，可使零星的、肤浅的、表面的感性认识上升到全面的、系统的、本质的理性认识上来，从而找出工作和事物发展的规律，进而掌握并运用。另一方面是建立了一条与他人分享知识的桥梁，达成了读者与作者之间无声的交流，沉淀下来的经验总结、失败教训可以不断传承，日积月累形成公司的宝贵财富。

测试总结，对于测试朋友来说也就是工作总结，在写测试总结时需注意如下事项。

- 实事求是：测试是一个技术性的工作，取得的成绩，经验与教训需要有数据或事例的支持。犹如科学来不得半点虚假，否则失去总结的意义。
- 表达清楚：测试工作总结，大部分都是技术方面的总结，与文学表达不一样，天生与比喻、夸张无缘。要求结构有条理，语句通顺，断句清楚，专业术语有解释，读者容易理解。
- 内容简洁：切忌形式，内容空洞，如流水账式的总结，只记录了什么时间做了什么事，这叫空有其壳。没有灵魂的总结，又何谈价值呢？

测试总结多种多样，如测试工作日志、专项测试总结、项目测试总结等。要开展哪些方面的总结，可根据工作的具体情况而定。但是，其中的测试工作日志是一种很重要也很基本的工作总结，或者叫测试日结。下面就测试工作日志的好处和写法与大家一起分享。

9.4.2 测试工作日志

有一句话是这样说的，“好习惯成就好人生”，话不无道理。很多朋友或许会说“话说容易，做起来却难呀！”几年前，笔者与大家一样，知道这句话，但从来也没认真去思考过。后来，有幸与一位香港同事合作一个项目，他的一件小事让我记忆深刻，并且从那以后也改变了我的工作习惯。也正因为这个写日志的工作习惯，帮我解决了不少工作上的麻烦。

一天早晨，香港同事 Simon 拿着一张表来找我（还以为是什么资料，原来是他的 WorkList），上面记着一件件他需要解决的问题，或已解决的问题。表的格式大致如图 9-13 所示。

Date	No	Problem description	Remark	Status
	10			
	9			
	8			
	7			
	6			
	5			
	4			
Jan.24, 2004	3	Bug#158, Take photo's mode is wrong? what is it's detail?	todo:ask Jane	
	2	For define item in global.h	todo:ask Carl	
Jan.23, 2004	1	1.3.1 what the unit have a beep noise during take a single photo?	need to discussion with Sunny	fixed

图 9-13 简洁的 WorkList

他认认真真地指着表上的一条记录（上面我看到了我的名字及他要解决的问题）问起了问题，并根据我的回答做了记录。虽然是一个小小的举动，但是此时“认真”、“细心”、“专业”这些词马上在我的脑中闪现。一开始我很好奇，还认真地盯着那张纸看了好一会儿。亲爱的读者，也许你已注意到，这张表很简单，但也很特别。如表中“倒序法”的记录就是一个，总是把发生在最近的事放在最前面，方便看到，符合人的习惯，值得学习。后来笔者也就有了属于自己的 WorkList。说来也真是好运，在一件小事上，平日里积累的测试工作日志，一次竟然还“救了我”。

事情是这样的：一次，公司考勤人员查到过去的某个月我有 3 天下班时间没打卡，要补办手续，并给经理确认。然后，无论如何向考勤人员解释说实际上班了，而且还加班到晚上 9:00 以后，但她就是说手续上过不去。最后灵机一动，就把当时的工作日志发给她看，证据摆在她面前，绕过一关。

工作日志的好处远不止这些，它给我们带来的价值，常体现在无形之中，下面是一些具体体现点。

- **提醒作用：**工作日志是记录任务来源及任务输出的过程，因此，对于我们来说，工作日志的提醒作用非常明显。在实际操作过程中，可能会同时进行多项工作，有时可能会因注意小的现象而忽略重要的事情。所以及时的查看工作日志，并进行标注，对每一位测试朋友都有重要作用。
- **跟踪作用：**跟踪，可以是任务完成情况的自我跟踪，但一般情况是管理人员跟踪

较多。不同的测试人员从事的业务不同，其工作内容就会有本质上的不同。团队越大，测试人员的工作情况就越难于控制，管理成本增大。此时，测试工作日志可以起到重要作用，测试主管或经理可以通过测试日志对重要工作进行跟踪，及时了解存在的问题，并进行解决，把风险降低到最低限度。

- 工作业绩证明作用：一方面，是给自己看的。作为一种数据的记录需长时间保留，必要时发挥它的价值，上例便是一个很好的例子。另一方面，给他人看的。公司内部员工之间的合作需要一个公平、公开的平台，在这个平台上做事，员工之间就不会有太多的疑义。员工工作量、工作效果等大家都一目了然，不会因为你讲，你的业绩就得不到体现，也不会因为你能讲，你的业绩就会扩大。通过工作日志这种方式进行证明业绩，大家都会心服口服。

9.5 测试知识库设计

刚开始从事测试工作时，会觉得太多东西要学要掌握，而当工作几年，在多个项目中历练后，自然就会积累一些经验、总结。而这些经验总结正是一个测试团队一步一步成长、壮大，走向成熟的基础。测试知识库的概念正是因为有了这些基础，而被管理者提上日程。下面就知识库的建立、管理、应用分别做介绍。

9.5.1 沉淀测试知识库

首先，让我们一起来关注一道数学题：某测试团队有 10 人，公司要求每人每周都要写总结，那么这个团队一年下来，能产生多少篇总结？

以一年共 53 周计，则 $53 \times 10 = 530$ （篇），即一年下来，这个团队将积累 530 篇总结。数目本身可能算不了什么，宝贵的是其中所包含的内容。那是前人一次次摔跤、一次次失败的经验教训，无疑它是后人少走弯路的指路明灯。

测试知识库，是一个与测试相关的知识体系集合，包括多方面的知识，测试总结只是其中的一项内容。测试知识库存在的价值，我们都毋庸置疑。就像书籍是知识的海洋，它是关于软件测试知识的海洋，专属于软件测试领域。从内容空间分析，为适应不同行业不同性质的业务领域发展，它有专业共性的一面，也有与业务特性关联的特性一面。如图 9-14 所示是一个表示内容空间的示意图。

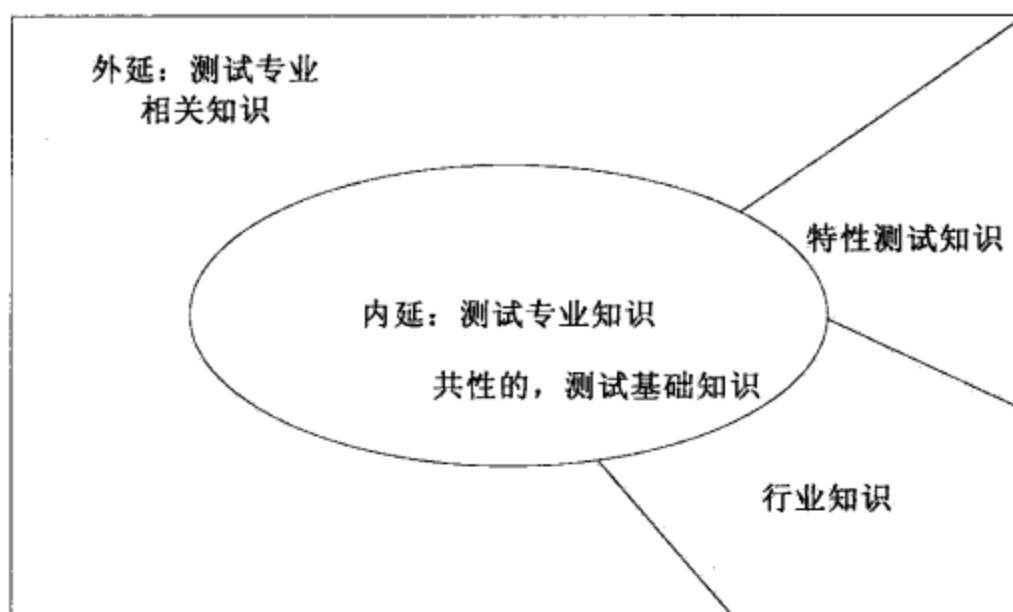


图 9-14 测试知识库体系示意图

图中，内延指测试专业领域内的共性测试原理、方法、策略等知识，与行业无关，是共性的，是基础性知识；外延指与软件测试密切相关或间接相关的其他专业知识，如软件需求、软件开发的相关知识，行业产品知识，硬件设计，工艺设计间接相关知识等。

知识库的建立，需要有一些基础，一些内容的沉淀。当没有独立的测试团队，或测试团队成立之初，还没有积累时谈不上知识库的建立。但有了测试团队后，作为团队的领导需考虑团队的成长、发展，鼓励团队成员之间常进行工作总结，分享工作心得，从而推动知识库的形成与建立。

小贴士：

知识库（Knowledge Base），是知识工程中结构化，易操作，易利用，全面、有组织知识集群，是针对某一（或某些）领域问题求解的需要，采用某种（或若干）知识表示方式在计算机存储器中存储、组织、管理和使用的互相联系的知识片集合。

9.5.2 测试知识库的管理

说到知识库的管理，例如那些沉睡在库中的“宝贝”，大家是否容易找到，当手头上的工作遇到某个问题时，是否能像 Google 或百度一样方便搜索，快速而又顺利找到想要的信息。个人的总结可以放在自己的 PC 上保存，需要时随时打开来读。但一个团队的总结，为了达到共享，最少也需放在一个大家都能访问到的地方，这是形成知识库的第

一步。但是共享文件的形式，绝不是一个知识库管理的最佳方案。如何让用户能快速搜索到想要的知识点，这是知识库管理需要考虑的问题，也是前人的经验总结得于流传、继承的关键。

目前，可应用于管理知识库系统的工具不少，如 Sharepoint（微软商业）、MediaWiki（开源免费），都是不错的知识库管理系统。选择工具时，需考虑我们的需求，如知识的分类管理、查找、上传、下载、讨论，最好有接口能自行配置或可做二次开发（开源的软件有这个好处），能真正达到知识的方便获取、分享、交流。

有了工具后，接下来就是考虑如何利用这些工具来为我们服务，如在部署测试知识库的目录时，可按分类进行，如图 9-15 所示是一个测试知识库的层次结构部署图，可供参考。

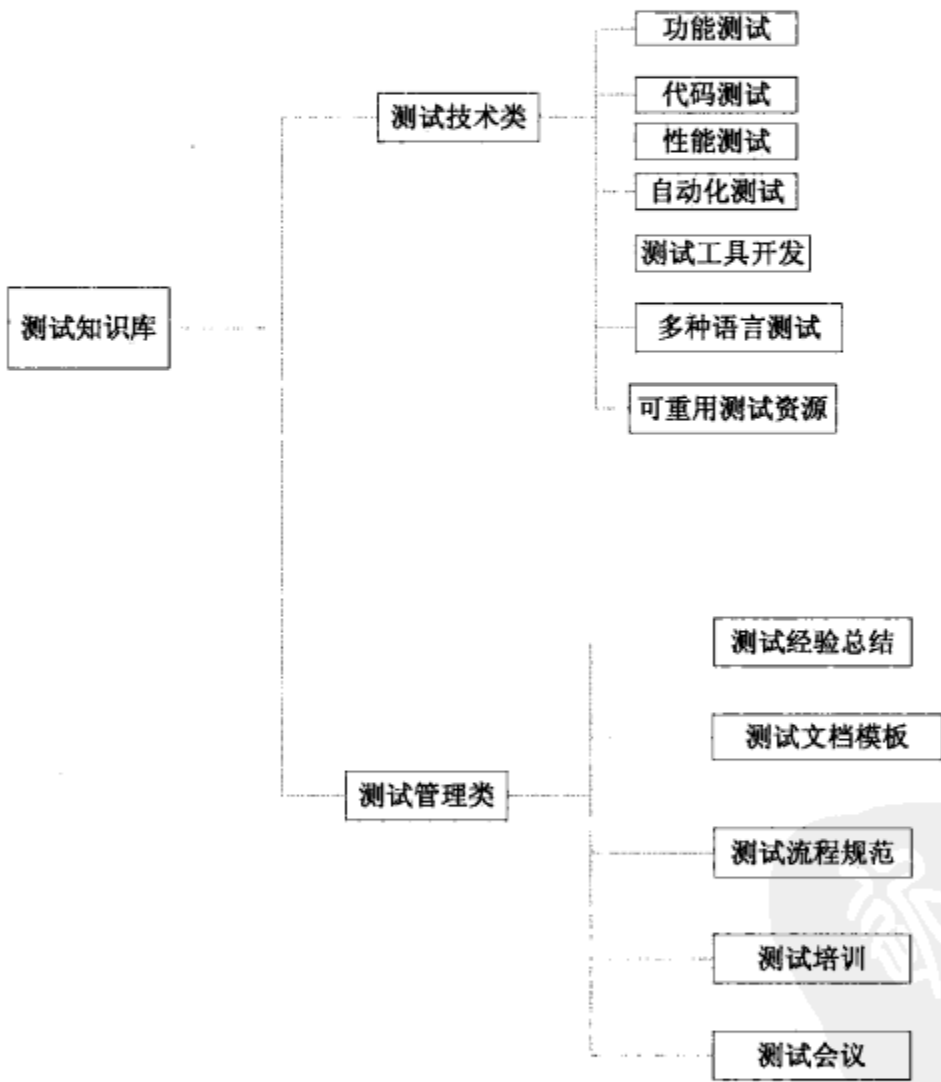


图 9-15 测试知识库结构部署图

测试知识库，为我们建立了知识学习、分享、交流的平台。在公司，特别是一些大公司，知识库中的数据每天都在增加，对这些数据的维护，包括数据的安全、权限管理、知识的更新等，需要专人来管理（当然如果规模小，找个人来兼职做也未尝不可）。

9.5.3 学以致用打折吗

俗话说“前人种树，后人乘凉！”能不能利用这些树，并在树下乘好凉，是体现知识库价值的关键。

有了前辈们留下来的一个个项目的测试经验总结，遭遇的失败教训案例，晚辈们在遇到同样的问题时，是否能顺利地借前车之鉴，少走一些弯路呢？下面是一个典型的应用案例，答案就在其中。

【案例】参数化法改进用例的设计

背景：陈希是刚加入测试团队的新测试工程师，杨也是他的入职导师。

入职第二周后，陈希接到一个项目更改点的测试任务。这个更改点需要设计新的用例来执行测试。杨也看了需求点之后，想到此更改点的测试特性很明显，与不同的数据关联，用“参数化”的用例设计方法比较合适。于是把这种方法向陈希做了简单的介绍，并指导他如何在知识库中找到这个培训教材，以及用例设计的实例。要求陈希看完后，能结合工作实例进行应用。

第三天后，杨也收到陈希用 Excel 表格设计的测试用例，打开一看，洋洋洒洒写了近千条用例。由于每条用例的操作步骤是一样的，仅是输入数据及预期结果中的数据不同，看得杨也眼花缭乱。出现在每条用例中的相同的操作步骤描述，使用例显得过于累赘、冗余。杨也不明白，为什么设计之前已清楚地指导过用什么方法，并进行了讲解，也把相关资料发给他学习了，但结果却如此不近人意。于是问陈希，“参数化”的用例设计方法是否搞清楚了。他回答说“当时看时，好像懂了，但想用时，不知从何下手，而用例设计表本身有一套模板，于是就按传统的方法，根据模板中的定义进行了此更改点的用例设计”。

对于案例中的事实，笔者一开始也很不理解，但后来想想，其实这样的事再正常不过了。试想想，知识库中的这些方法，是前人可能经过无数次实践、失败的教训总结出来的，他们当然相当清楚，而后人仅是从文字上看到了，学习了。是知其然而不知其所以然，此时，拿起来想用时，自然就不知如何下手了。

当然，知识库中的经验总结，并不是每一个都如案例中的一样，学起来容易，用起来难。有些资源是可以直接应用的，如一些已形成的测试脚本或代码、测试工具等。

“理论从实践中来，需再回到实践中去”，这是科学的真理。测试工作是一门技术性很强的工作，专业的特性要求我们多动手、多实践，甚至有时需没事情找事情做，从做中想（悟道），从学中用（应用），做与学相结合，才能体现出知识库的更多价值。

第 10 章

控制测试过程的实用方法

前面章节内容讲述了测试过程中各环节如何做，以及做出来的东西如何管理，涉及了测试设计的方法、测试执行流程的设计、不同类别测试输出工件的管理。方法、流程与管理形成了一条有序完美的测试链路，然而，链路上环环相扣的各测试节点是否如我们预期的扣紧了？毕竟只有过程中各节点的质量得到了保证，测试链路的末端才能画上圆满的句号。本章就如何控制测试过程各节点的方法结合案例进行介绍。

10.1 把握测试工作启动的起点

一个新项目立项后，测试何时介入？测试工作什么时候启动更合适？测试的启动时机对后续的测试工作又有哪些影响？本节将对这些问题向读者进行讲解。

10.1.1 测试人员何时投入项目合适

在测试界，“测试人员何时投入到项目中最合适”，这是个一直都存在颇多争议的话题。有不少朋友常会不假思索地回答“越早越好”。真是越早投入越好吗？早又是指早到什么时候呢？而认为测试人员不用太早投入，又是什么原因呢？

笔者认为，凡事都要视其具体情况。一方面，从对项目的认识上来说，在项目立项后，测试人员与开发人员、需求人员一道参与项目相关的讨论会，肯定比代码提交后才参与执行要强很多。另一方面，与测试实际要求做的工作有关。如果认为测试只要做黑盒功能测试，由于它是一种动态测试，需要运行软件才能完成，那么在编码完成前参与执行，与开

发只做一小步的并行，也是可以的。再者，如果测试方案设计、用例设计都省了，那么确实就是编码完成后才参与了。遗憾的是，据调查，现在仍有不少公司，特别是一些小公司，对测试的投入，仍处于这个阶段。测试人员的介入时机，与公司的研发模式密切相关，如传统开发模式，与敏捷的开发模式对测试人员介入的要求就不同。如图 10-1 所示，是一种业界用得较多，流程比较严谨、传统的软件开发模式。而如图 10-2 所示是一种当今比较流行的属于新型的敏捷开发模式。

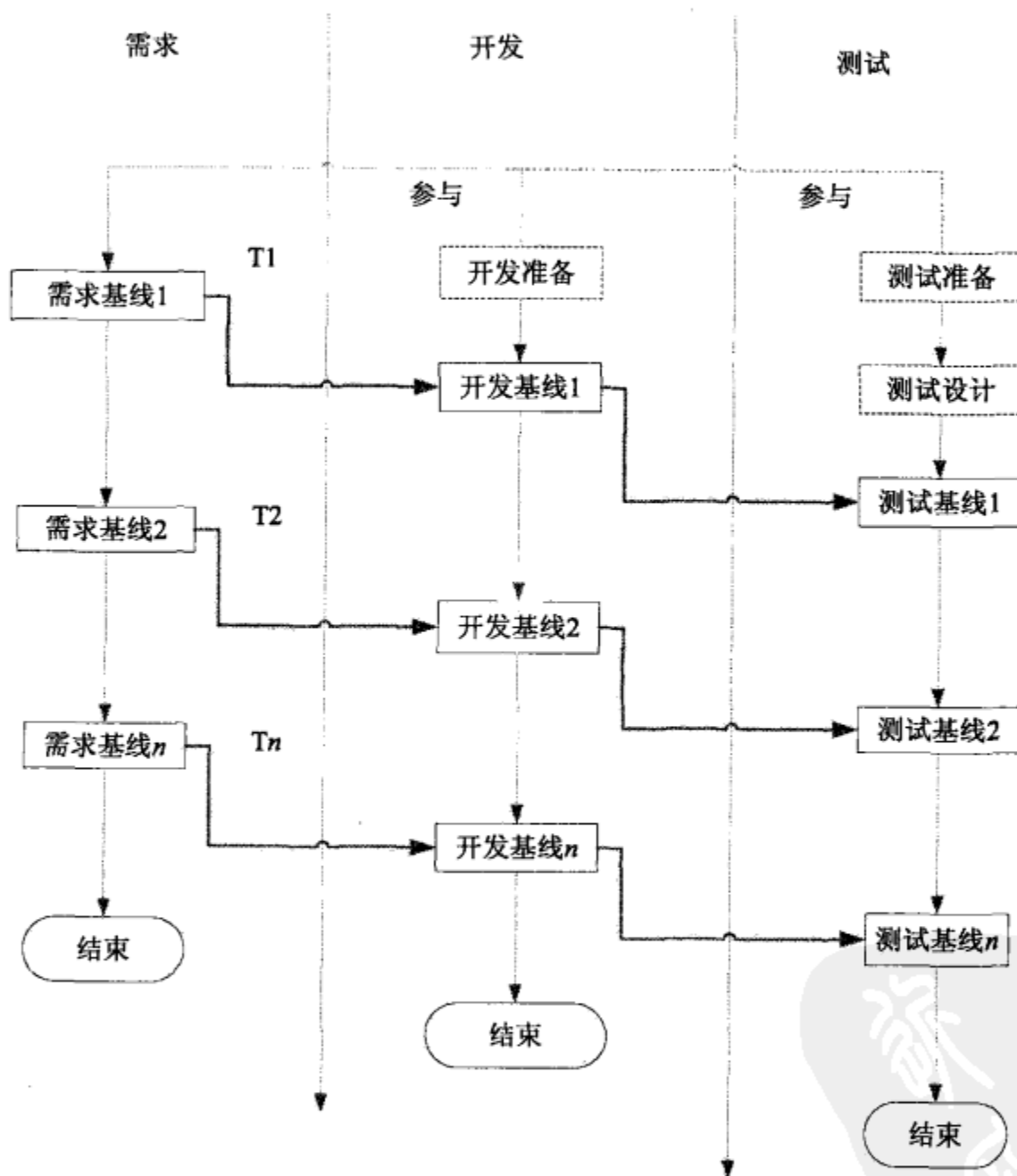


图 10-1 传统的软件开发模式

我们从图 10-1 可看到，在传统的软件开发模式中，需求、开发、测试的阶段呈一个“Z”字型。需求走在最前面，需求设计人员基本是独立作战，开发与测试人员只是参与，如参与需求的评审等。在大部分情况下，需求基线在还未形成之前，测试人员还忙于其他项目。

测试在“Z”型的下端，处于开发的下游。如果测试对自身要求不高，在测试准备、测试设计阶段没有规划，则在开发提交版本后才正式介入，的确时间太晚。也就是说，开发模式本身的机制上在前期对测试没有什么要求，但是测试仍完全可以通过主动规划，尽早展开需求测试、测试设计的工作，最起码与开发的设计工作做到并行或部分并行。这样当开发提交版本后，测试已是“万事俱备，只欠东风”，达到测试成功是可预期的。

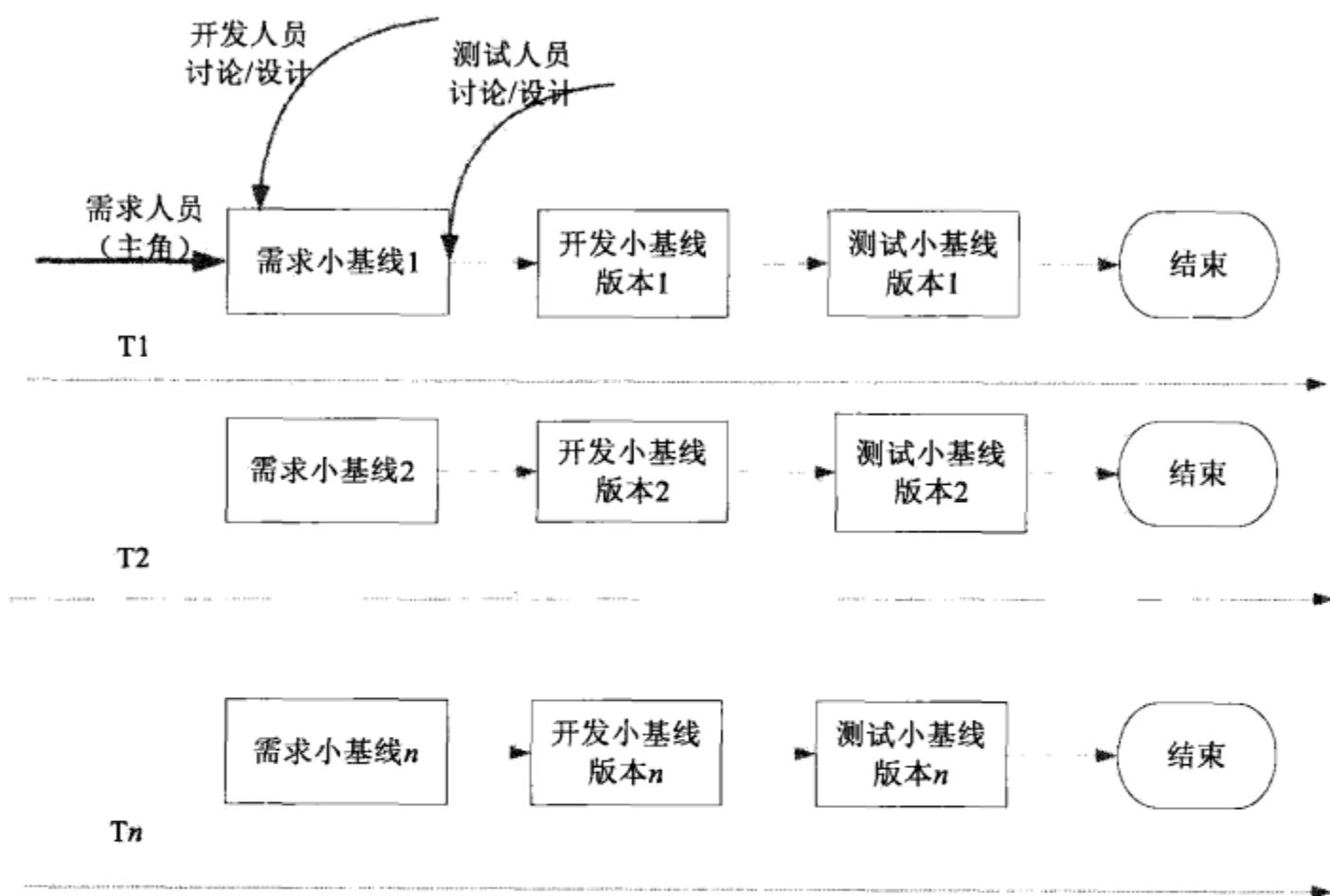


图 10-2 敏捷软件开发模式

较之于传统的“Z”型开发模式，敏捷开发模式的一个突出特点就是“并行”工作，即需求、开发、测试各方并行作战，测试、开发与需求人员一起讨论、设计需求，但需求设计人员为主角，讨论的结果由需求设计人员形成文档基线。此时的基线只需事后的记录或细节的完善，因为在三方讨论达成一致意见后，回到各自的位置，大家已开始分头做各自的工作了。这种模式下，测试对需求的理解是从讨论中开始的，有更多机会提出建议或意见及一些风险，并纳入需求中。或者可以理解为，测试在前期是在协助做一部分需求的设计，而不是像传统开发模式中的被动接受需求。

敏捷开发也好，敏捷测试也罢，敏捷方法从结果体现来看就是“高效率”发布软件版本，如图 10-2 所示的“T1”、“T2”、“Tn”，分别是一个个小里程碑，每个里程碑结束后对应都是一个预期的已具备一定业务功能的可用软件。始终围绕“可用软件”，集合需求、

开发、测试整个大团队的智慧，进行一次次迭代开发，使“可用软件”不断完善、成熟，如图 10-3 所示。这就是敏捷开发的精髓，测试成了其中一支不可或缺的力量。

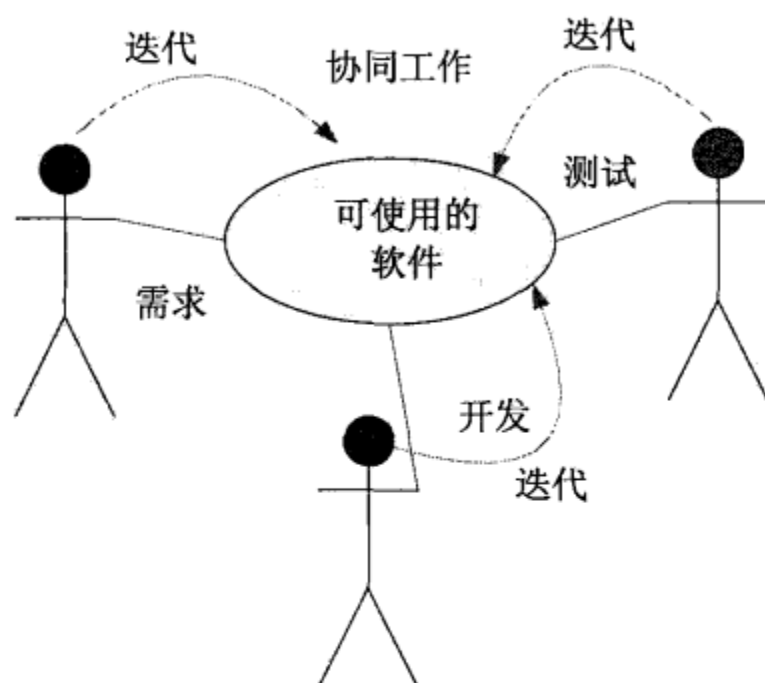


图 10-3 敏捷开发模式的核心

小贴士：

敏捷开发：

一种以人为核心，迭代、循序渐进的开发方法。在敏捷开发中，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。换言之，就是把一个大项目分为多个相互联系，但也可独立运行的小项目，并分别完成，在此过程中软件一直处于可使用状态。

敏捷宣言：

- 个体和交互比过程和工具更有价值。
- 能工作的软件比全面的文档更有价值。
- 顾客的协作比合同谈判更有价值。
- 及时响应变更比遵循计划更有价值。

10.1.2 项目测试启动会

首先，让我们先来看看下面的一个职场小故事。

【小故事】

一天上午十点左右，韩国爱敬集团旗下化妆品公司的业务部，新来了一名女员工。上班第一天，她意外地收到了一束美丽的鲜花。惊喜之余，她打开了夹在花束里的留言：欢迎你加入我们的团队，我深信你会是一名十分出色的业务员。但是，我最害怕的事情就是让员工应得到的奖励迟到了，所以我宁愿早点给你，我想你不会介意吧？落款是业务部经理的名字。

优秀领导者给员工的奖励，往往不是等到年终总结的时候。他们善于出其不意地让新人一进来就带着信心和动力投入工作，而这恰恰是一门管理艺术。

“好的开始是成功的一半”，故事中的业务部经理的做法，是一种很好的推动员工在日后有信心、积极努力工作的催化剂。

回顾我们身边的项目测试工作，通常，一个新项目经一番市场调研、可行性分析，最终决定立项时，项目经理也常会组织一次立项启动会，形式多种多样，在酒家或宾馆召开会议后共同进餐便是一种常见的模式。表面上看，张罗启动会需要花一些银子，增加了项目管理的成本，但由它带来的价值，对后续整个项目的良性影响，是无法用金钱来衡量的。比如它给大家提供彼此熟悉的机会，也会促进跨专业、跨部门团队成员间良好的沟通体系的建立。据说，1998年11月23日，联想集团ERP项目誓师大会在北京海淀工人俱乐部召开时，柳传志亲自到会讲话，把ERP项目摆到关乎企业生死存亡的高度，并亲手将一面大旗授予ERP项目的负责人。柳传志在会上说，上ERP有可能早死，不上ERP只能是等死，我们与其在这里等死，为什么不去拼搏一把呢？事实证明，这不仅极大地鼓舞了项目组成员的斗志，同时也使全体员工明白这不仅仅是信息部门的事，而是公司从上到下都要关心的事。

对于一个大项目，可能是一个包括成百上千人的团队，启动会也许不是所有参与人员都参加（常会只派各专业方向代表参加）。软件测试作为其中一个团队，要想在项目测试中获得成功，同样需要造势。下面便是一个经典案例。

【案例】

背景：PM-9800是某公司在电信设备方面一个重大项目，项目还涉及跨国团队，团队规模将近2000人。Shirly作为软件测试团队的代表之一，有幸参加了项目在某五星级酒店举行的隆重项目启动会。

Shirly参加启动会回到家，已是晚上11点多了。心里一直捉摸着，“明天该不该在测试团队内部作一个项目测试启动的宣讲会？今晚的誓师会毕竟不是全体测试人员都参加了。传达会议精神，传递相关信息等还是很有必要的”。于是就把明天的测试启动会内容

大致想了想，有些点子还是挺好的。怕过了这个村，没这个店了，时间一过，灵感消失，再也捕捉不住。于是打开电脑，新增一个 Word 文档，输入如下内容：

1. 人员：全体测试人员（包括暂无安排参加项目启动会的人，可能在项目的中后期会安排加入）。

2. 时间：1 小时内。

3. 地点：办公区的讨论室。

4. 内容：

（1）配合项目的启动，宣布测试启动。

（2）介绍项目的背景、目标及要求。

（3）转达昨晚项目经理、事业部总监的讲话精神。

（4）分析软件测试接下来要做什么？如何做？

列完清单，便发送到公司的内部邮箱。

第二天上班后，Shirly 看到项目经理及各技术组已忙起来了，特别是硬件、机械专业组的同事，可能与他们是要前期出样机有关吧。

“好在我已准备好，要不就显得有点落后了。” Shirly 看着他们，想着测试的事情。

“会议室订好了，10:30am，D-2F 小会议室，举行 PM-9800 测试启动宣讲会”，会议室订好后，Shirly 便高兴地把通知发送给了相关测试人员。

宣讲会上，工程师们高兴地听着，鼓舞着，有个别同事还不由自主地捋起了袖子，大有想大干一番之势。

软件测试是整个项目研发不可或缺的一环，如果说项目的启动会是给项目造外势，那么测试启动会就是给测试团队造内势。

10.2 测试设计的评审

这里说的测试设计包括测试方案的设计及测试用例的设计，评审的对象就是测试方案与测试用例。

首先，我们必须清楚评审的目的是通过发现设计中存在的问题，改进工作以提高设计输出的质量。由于人的一些个体差异，使得对项目的熟悉程度、积累的测试经验、所掌握的测试方法等都会存在不同，也将会影响测试设计的思维。直接地说，评审就是借别人的脑袋为自己服务，是一种“取人之长，补己之短”促进自我成长的有效手段。

10.2.1 三级评审机制

为了更好地达到评审的目的，有效控制设计输出的质量，经实践总结，我们可对评审内容按参与人员的范围采取由内到外的三级评审方式来执行。

第一级：自审，设计人员对设计内容进行自审。自审中发现了哪些问题，是否做了修改，以及如何修改的，需做记录。

第二级：内审，指把设计文档发给技术组内部人员进行评审，评审资料需至少提前 3 天发给评审人员，并给出评审反馈完成的截止时间。由于是同组的同事审核，大家对所测试的模块会比较熟悉，在测试技术方法的有效性，测试点是否有遗漏，特别是在一些细节上，常能提出很有效的意见或建议。

第三级：外审，指把设计文档发给部门外的专家评审，包括模块的软件开发人员、需求人员、项目经理、业务代表等。评审专家的角色众多，站的角度各不相同，常能提出测试设计方面方向性的缺失问题。如某需求的验证没有提到，特别是一些隐含需求；核心用户场景的测试考虑不够，或偏离了主题等。如图 10-4 所示为测试设计评审级别示意图。

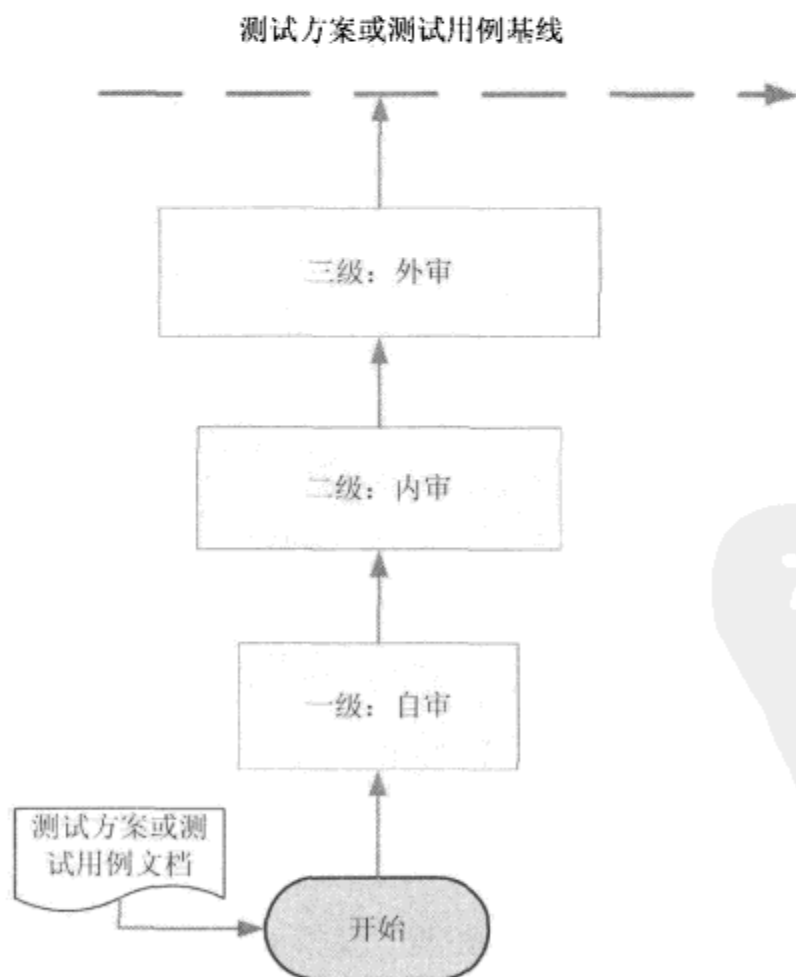


图 10-4 测试设计评审级别示意图

图中表明，依托设计人员已完成的测试方案或测试用例文档，评审级别由下至上，第一级自审在最下，然后逐级往上，最后形成基线，启动下一环节的工作。实际工作中，常常不能做到绝对的把各环节工作串联起来，在不影响下一个环节工作启动的前提下，各环节工作可并行开展。以用例的设计为例，它的前提条件虽然是测试方案的基线形成，但当测试方案某部分内容比较稳定时，便可开始对应的测试用例设计工作，不一定非等到方案的评审完成，形成严格的基线后才开始。

10.2.2 自审检查单的诞生

图 10-4 中的评审模式在流程上已很清楚测试设计评审要如何做，但是在做的过程中，每一评审级别要确定达到的目标。如果不提要求或需求，评审效果会大打折扣。下面，就让我们来看看一个关于测试方案“自审”的案例。

【案例】“自审”检查单的诞生

Sue 是某研发项目负责模块 A 测试的工程师，按照项目的测试计划，下周一前需完成模块 A 的内审。他做得很好，本周二 Sue 就把设计方案发给同组的测试工程师了，包括其测试主管 John。内审的资料提前 4 天发出来，符合内审的要求，主要目的是让参与评审的人员有足够的时间来审核。评审方案发出来后，John 为了活跃团队相互帮助的气氛，根据组内各成员的特点，把方案内容进行了划分，一个人只负责审核其中的某一小部分，并要求大家最少花半小时审核，把审核结果反馈给设计者时同时抄送给 John（由于平时大家的工作都有自己的任务计划，花太多时间来审核任务之外的工作不太现实，这种情况下采取减少审核内容及审核时间来减轻审核人员的额外工作是可取的）。

内审工作分工安排完成后，John 便马上开始审核起来（他是主管，工程师工作输出的审核对于 John 来说是责无旁贷的）。不审不知道，一个小时审下来，John 提出了近 20 个问题。有对需求分析不全面测试点遗漏的；有些测试点没有逆向的测试思路；有些分析所得的验证对需求的覆盖对应关系不清楚；文档的结构组织理解费劲，等等。John 只好通知组内人员暂停内审，把方案设计文档退回给 Sue。

John 找来 Sue，问他是什么原因遗漏如此多问题？自审是如何做的？

Sue 回答：“需求分析不全面有测试点遗漏是因为第一遍分析完后没有回头再多考虑。另一方面有点依赖思想，认为后面会有人评审，漏了的测试点会有人提出来，没提出来说明此测试点可能不容易发现，或并不重要。对于自审，由于觉得能分析到的都已分析了，最后只是对文档从头到尾扫描式地检查，并没有发现什么问题。”

通过与 Sue 的交流, John 意识到在自审的环节存在流程控制上的问题, 如何自审? 自审要达到什么样的要求? 人的自我意识占据着主导地位, 不同的人会有不同的结果。受代码走读检查单的启发, John 提出内审检查单的做法。并把这一想法与团队成员讨论, 把检查单的内容初步制订出来, 如表 10-1 所示。

Sue 是幸运的, 终于能清晰地按自审检查单来完成自审了, 自审记录也需同时备份到配置库上备案。在 John 的要求下, Sue 组织了一次会议, 对所存在的问题及如何整改向测试团队进行了宣讲, 从宣讲中进行反思, 同时也告诉其他人该如何进行自审 (是一个不错的管理方法, 有一箭双雕之功)。

表 10-1 测试方案检查单

序 号	检 查 项
1	测试对象的分析是否考虑了模块间的相互影响?
2	该测试对象是否存在性能上的要求?
3	设计需求是否有对应的测试方案?
4	对某一测试点的验证方法, 除了正向的思路, 是否也考虑了逆向的思路?
5	每一个测试点的验证思路都可以直接导出测试用例了吗?
6	用户场景的测试点是否足够?
7	业务流程的测试分析是否用了流程图来表达?
8	方案中的专业术语是否有专门部分进行了解释?
9	方案中的英文缩略词是否有全词的说明?
10	测试方案的组织结构是否符合公司的模板要求?
11	测试方案是否模块化了?
12	是否建立了测试方案与需求的索引表?

10.2.3 设计检查单——提高设计质量的有效工具

测试方案或用例的设计质量, 毋庸置疑, 与设计者本身所掌握的分析方法、测试经验、对业务的熟悉程度密切相关。除此之外, 站在流程管理控制的角度, 采用一些有效的约束规则, 可以使大家对设计目标有一个统一的认识。“自审”阶段, 是质量的主要把关阶段, 要求要高, 要细。“内审”阶段, 设计责任人可以根据自身的弱点或需专家重点审核的地方提出需求。“外审”阶段同样也可提出可能考虑不周的地方, 让专家们提出宝贵的见解等。

设计内审检查单, 包括测试方案检查单, 如表 10-1 所示。测试用例检查单, 如表

10-2 所示。

表 10-2 测试用例检查单

序 号	检 查 项
1	是否每一个需求都有其对应的测试用例来验证?
2	是否每一个设计需求都有其对应的测试用例来验证?
3	测试用例的设计思路是否合理?
4	每条用例的预期结果是否都唯一?
5	每条用例是否都可操作?
6	用例的测试条件是否清楚?
7	每一测试点是否都有逆向的用例?
8	每一测试点是否都有异常用例?
9	测试用例是否包含了已知的边界值,如特殊字符、最大值、最小值?
10	对流程业务,是否有对应的流程用例(从流程图中转化用例)?
11	用户场景用例是否足够?
12	是否考虑了与其他模块之间的接口用例?
13	用例的组织结构是否合理与清晰?
14	用例的编写是否符合规范的要求?
15	用例的格式是否符合模板的要求?
16	是否考虑了性能测试用例?
17	是否考虑了安装/卸载测试用例?
18	是否考虑了升级兼容性测试用例?
19	用例的元素是否齐全?
20	用例是否易读?
21	用例是否易维护?

测试方案及用例的检查单,主要可用于自审,当然作为设计评审的一种辅助工具,内审及外审专家也可以用。尽管内审与外审由他人进行审核,作为设计者,是非常希望得到他人的有用建议的,特别是一些有建设性的见解。下面是一些审核过程中的注意事项。

- 会前发出评审资料,让评审专家有充足时间事先阅读待评审的方案,并把软件需求也附上,方便他们需要时查阅(同行内审时,有可能评审专家当时在另一项目,功能需求及设计需求等资料并没有看过,作为一种贴心的服务,方便他人实际上是帮助自己)。
- 列出你需要专家们重点关注的点,这些点通常可能是你最没把握的,希望他们给

予这些方面的审核或指点。

- 评审专家团组成：内审，测试部同组测试人员，外审，方案设计者，设计者直接主管，开发工程师、需求工程师、项目经理、业务代表等。
- 评审记录：如果评审前有专家已反馈了意见，把专家意见在评审时进行宣讲，以及如何处理进行说明。评审会时，认真记录评审专家的意见，并在会后一一处理。

测试方案或用例评审后，根据专家意见进行整改，并形成整改记录。评审整改后的方案或用例作为阶段性里程碑版本进行归档，同时放入项目配置库的基线库中。

小贴士：

评审心理学

笔者曾问过一些测试工程师，“参加内审或外审中，在反馈一些问题给设计责任人后，你想得到对方的回复吗？”

测试人员 A：反馈了意见，但没有回信，最后也不知对方是改了还是没改，我们的意见或建议对他有没有帮助，不得而知，下次再内审时积极性会受影响。

测试人员 B：反馈了意见，没想那么多了，改不改是设计人员的事，但最好还是回复处理一下。

严谨的处理：无论是内审还是外审，对于评审专家提出的意见，都要进行书面回复，并进行封闭归档。

10.3 测试版本的控制

通常，开发人员提交给测试人员进行内部测试的软件版本，我们称之为测试版本。测试版本意味着此软件在某些方面是不成熟的、不稳定的，可能潜伏着不少 Bug，是一个待验证的版本。软件的代码由开发人员编写，代码的版本由开发人员控制。测试版本作为整个版本演变过程的某个节点，好像也是由他们控制的，但是事情是我们想象的那样吗？本节将就测试版本在传递过程中，如何做到有序的控制，介绍一些具体的方法。如图 10-5 所示是测试版本传递示意图。

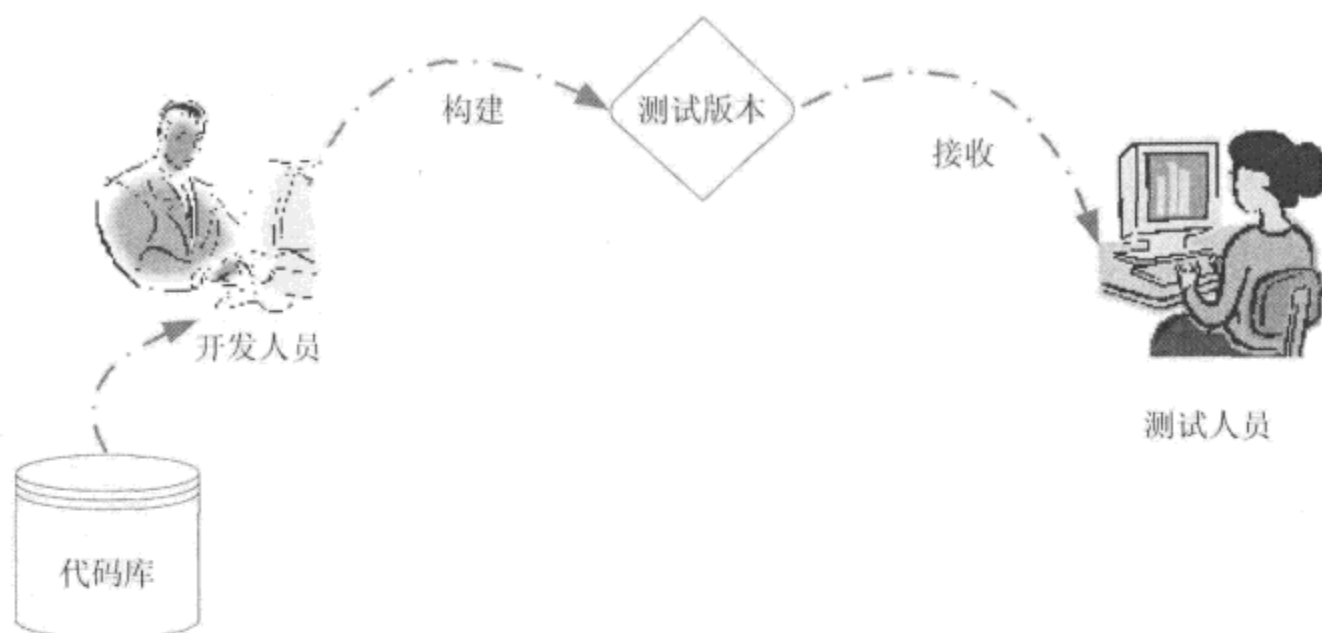


图 10-5 测试版本传递示意图

10.3.1 版本发布众生相

尽管目前在软件的项目管理上，版本的配置管理工具应用已很普遍，但是不是所有的软件项目开发人员都已采用了配置管理的理念，并在工作中实施了呢？下面的应用场景或许似曾相识。

场景 1：开发负责人构建。开发小组人不多，一个项目由 1~3 人组成，每个人负责一个或多个小模块的开发，在自己本机上完成代码的开发与管理。需要提交版本时，把代码发给开发负责人合并，编译版本，最后输出一个或多个可执行文件或安装包，再通过 E-mail、QQ 等方式发给测试人员。

场景 2：专门的配置管理员构建。某研发队伍有专门的配置管理员，开发人员提交代码后，通知管理员取版本编译，然后放在配置库上的某个目录，并通知测试人员去取版本测试。

场景 3：测试负责人构建。研发团队有专门的代码配置库，开发人员提交发布版本的代码，以及编译后的可执行文件，然后通知测试人员。测试人员重新取源代码进行编译，确认开发提交编译后的文件与源代码能正确对应，然后接收版本开始测试。

场景 4：自动构建。只要代码库有人提交版本（版本有变化），编译工具就会自动取下配置库中的代码进行编译，并把编译结果放在指定目录，同时发邮件通知测试人员；也可设置成符合某条件时才进行编译，以满足测试有需求时才编译并发布版本。

还有其他方式的版本编译与发布场景，但就上面的几种情况而言，据调查还是较常见的。场景 1 常见于公司成立之初或小作坊式的开发模式。这种方式，完全没有版本的概念，

软件界面上显示的版本号也是人工写在代码中的。除了容易产生版本混乱，意外的文件丢失外，好像还没找到它有什么好处，看了下面的案例相信你会更有感触。

【案例】找不到以前版本的源代码

某公司研发并生产数码相机，DC-2580 这个小项目的软件开发只有 Jack 一人，所有代码都放在个人的本机电脑中。一天，生产那边要求发一个临时版本给他们在工装上调试用。Jack 很快在自己的本机上编译成可执行的 bin 文件，用邮件发给工厂。第二天下午上班后不久，工厂那边来电话说相机拍的照片有明显的颗粒状毛刺。Jack 知道后马上做了分析，发现在数字插值算法处理上存在缺陷，于是他优化了相关代码，并在当天下午下班时把修改后的版本发给了生产。但第三天下午上班后，Jack 又接到投诉电话，说昨天的版本在拍照后保存图片时经常会死机，为了不影响他们的调试工作，他们想退回到第一次的版本。但在工厂那边只保留最新的版本，需要 Jack 重新编译一个旧版本发给他们。此时 Jack 才想到，自己并没有备份发出去版本的源码，第二次的版本是直接在第一次版本的源码上更改的，要退回版本的话，这两天修改过的代码都需注释掉。由于涉及的文件不少，代码块也不集中，Jack 自己也怀疑“还能完全退回吗？”

场景 2 与场景 1 刚好相反，常见于大公司，有专职的配置管理员，当然此管理员的角色不止是编译构建版本，还有其他配置管理工作，如项目文件的管理、用户权限的分配、配置库的日常维护等。这种版本发布方式的优点是开发与测试都不参与版本的制作，管理员编译与配置版本完全按规范流程做，版本错误几率小，且每个版本都有完整的备份。

场景 3 的版本构建模式，实际上已包含了版本确认的工作，是一种比较特殊但安全的构建模式。

场景 4 的自动构建，应是目前最流行的一种版本发布方式，既省时省力，还可进一步实现无人值守的自动化测试。要实现自动构建，首先代码需放在配置库上管理，如 SVN、CVS 都是很不错的开源版本管理工具。

10.3.2 版本接收/停止测试准则

上一节介绍了我们常见到的版本发布的模式，无论以何种形式发布的版本，最后测试人员收到版本后，是不是就必须接收呢？版本接受指测试人员开始以这个版本为基础开展测试执行工作。反过来，测试满足什么条件下可以停止测试呢？

1. 测试开始检查单

本书的 8.2 节介绍过“内部版本发布测试”，里面讲解了内部版本的质量控制方法，这里不再赘述。测试版本的质量是重要的，除此之外，还有一些事项也值得注意。在项目测试开始之时，测试负责人需与开发负责人达成一致的共识，形成准则加以执行。

结合公司的开发流程规范及实际存在的一些问题，我们可定制出合适的版本接收检查单作为一种辅助工具，在满足的项后面写“是”，不满足的项后写“否”，以作为检查依据。对不满足接受条件的“否”项，需分析对测试的影响面，再决定版本是接受还是打回。如表 10-3 所示是版本接收检查单样例。

表 10-3 版本接收检查单

序 号	检查项	检查结果	备 注
1	项目计划清单中指定此版本要实现的功能是否已提交?	是	
2	未修复的 Bug 是否影响测试工作的进一步开展? 有影响的情况下, 影响度是否可接受?	是	
3	Bug 修复率是否已达到故障管理规范的要求? (具体多少, 如 85%、90%视情况而定)	否	
4	该版本是否通过开发内部的冒烟测试?	是	
5	测试信息传递表是否已填写完成?	是	
6	是否存在没有来源的更改?	是	
7	提交功能对应的测试方案是否通过评审?	是	
8	提交功能对应的测试用例是否通过评审?	是	
9	测试环境是否已准备好? 如所需测试平台、服务器、仪器或其他辅助设备	否	
10	本版的测试目标是否已清楚?	是	
11	所需测试人员是否都已到位?	否	

2. 测试停止检查单

不少公司的测试报告中都会提到测试停止的条件，就此问题笔者曾与一些测试人员交流，综合起来有以下几种回答。

回答 1：项目要求时间到了，测试即停止。

回答 2：测试用例已执行完成，测试停止。

回答 3：要测试的任务已完成，包括测试用例及 Bug 的回归，测试停止。

在测试过程，常会有人问笔者这样的问题：“测试什么时候可以完成？”一开始，回答这样的问题，总觉得难以给出一个准确的时间。并不是对测试过的软件没有信心，而是像很多测试朋友一样，笔者也是一个追求完美主义者，总是想把所有的 Bug 都找出来。然而我们不是生活在真空中，现实中所有的项目都受时间与成本的限制，不可避免地需要在

时间、成本与质量上找到一个平衡点。这个平衡点就是我们所谓的测试停止条件。如表 10-4 所示是测试停止检查单样例。

表 10-4 测试停止检查单

序 号	检查项	检查结果	备 注
1	所有功能需求都有 1 条或多条功能用例与之对应	是	
2	建立了需求与用例追溯表, 需求覆盖率已达到 100%	否	
3	所有设计需求都已实现并测试通过	是	
4	所有用例都测试通过, 测试记录已保留	是	
5	所有要解决的 Bug 都已解决, 并回归完成	否	
6	延期的 Bug 已通过评审专家的影响评估, 并在可接受区	否	
7	缺陷库上的所有 Bug 都已处理完成, 如已关闭、已延期等	是	
8	若有个别用例没通过, 经相关专定评审, 在可接受区	是	
9	同一模块至少有两人以上测试过	是	
10	最后版本上已通过全部用例的回归测试	否	

上表仅作为示意, 操作时可根据公司内部的具体情况进行合适的补充。检查单上的全部条件满足或哪些部分条件可不满足, 即可停止测试, 需在测试策略的制定中定义。

10.3.3 测试与版本号

1. 测试版本的一致性

测试版本的一致性, 这里指同一个项目的测试团队成员, 在同一时间段测试同一个版本。这样, 发现的 Bug 都是基于同一个对象的, 这对版本间的 Bug 趋势分析与 Bug 的管理都是有好处的。另外, 也可避免一些不必要的时间开销。例如测试朋友常会遇到提交了一个 Bug 后, 开发人员开始会不相信有这个 Bug, 此时如果你测试的不是最新版本, 会收到要求在新版本上重新确认的要求。

2. 版本号检查

这里, 有一个简单而又重要的测试问答题。

问: 对于新安装或升级的软件, 第一次打开软件后, 你首先做的是什?

回答 1: 操作软件。

回答 2: 回归 Bug。

回答 3: 测试新功能。

读者朋友，如果是你来回答，你的答案是什么呢？

下面是一些应用场景。

场景 1：测试人员 Jerffy 正在测试一个某数码相机的辅助管理软件，发现一个 Bug 后录入 Bug 库，当选择版本号时，才突然发现此软件版本号是什么还不知道，在软件的界面上也没有显示。

场景 2：测试人员 John 有个好习惯，每次测试新版本时，第一时间会去检查软件的版本号。所以，今天刚接收的版本发现与上一次的一样，还以为自己取错版本了。原来开发人员把版本号写在了代码中，然后在某个软件界面上进行显示，类似如图 10-6 所示。但是，本次提交版本时，忘了修改软件的版本号。

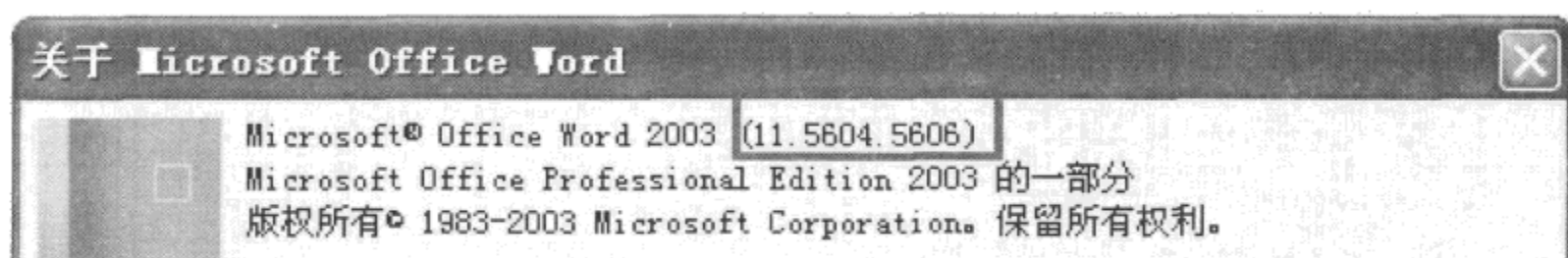


图 10-6 版本号的显示

10.4 测试配置管理

软件配置管理（SCM，Software Configuration Management），如今在软件工程领域的应用已很普遍，测试的配置管理属于软件配置管理的一个子集，是属于配置管理的范畴。主要用于管理我们的测试输出，包括测试计划、测试方案、测试用例、测试代码、测试工具、测试脚本、测试经验总结等。基本上所有的测试输出都可以纳入配置库中，运用配置库的版本管理与控制功能为测试过程服务。配置库的管理不会直接改变测试输出的质量，但可以避免测试输出工件混乱，提供资源共享，使整个测试团队的工作井然有序。接下来通过案例，分析测试配置管理对测试工作的重要性与意义，以及介绍测试配置库的有效管理方法，包括创建与维护管理等。

10.4.1 测试也需“电子眼”

深视三套财经生活频道有一个“法眼看天下”的栏目，印象深刻的是几乎每个案例的侦破都有“电子眼”这个现代化电子警察立下的汗马功劳。这个功劳的核心就是它所提供的在其监控范围内场景变化的实时快照。测试也需“电子眼”，是不是也需要在测试人员

工作的场所安装“电子眼”，对测试人员的工作举动进行监控呢？显然不是。这里的测试“电子眼”指的是测试配置管理（TCM, Test Configuration Management），而且，这个测试“电子眼”，它不会主动触发快照，需要测试人员提交变更，或有自动脚本等外界条件来触发。在接下来进一步介绍这个测试“电子眼”之前，与读者分享一些也许正发生在你身边或你也曾遇到过的工作场景。

- 测试团队中每位成员的日常工作输出，如测试用例、测试代码、测试报告等放在各自的个人电脑中。
- 测试文档要提交（归档）时，或发出来评审时，到处找找不到，又说不清楚是何时丢的，怎么丢的。有些急于提交，只好加班加点补一份（但已不是原汁原味了，质量上大打折扣）。
- 突然有一天上班后，电脑开机黑屏，机子上所有测试数据一去不复返，所有的工作成果好像是顷刻间化为乌有！
- 工作过程中，团队成员之间免不了信息的交流，常用邮件或 QQ 传递着测试信息，如测试过程中的软件版本、测试文档，常出现一个文档一天之内，发送者发送多个版本，但又没有版本的标识，接受人员每次接收都要进行保存，最后保存多了就乱了（保存的路径不同或文件名不同等）。

如何解决上面这些问题，让我们的工作有条理、可控、可追溯，这正是测试“电子眼”的专长。下一节将介绍如何构建与应用这个测试利器。

10.4.2 测试配置的构建与应用

可以单独建立一个测试库，也可跟随项目一起，作为项目的一个子项进行管理。但就测试类公共的资源方面，笔者建议用一个独立的测试资源库进行管理，可参考如图 9-15 所示的测试知识库的结构部署。采用的工具可以与常用的软件配置管理工具一样，如 SVN、CVS 都是免费开源的优秀配置管理工具。测试的配置，关键就是对日常工作中常用的测试输入输出资源进行归类存放，先划分大类，然后再细分小类。让测试团队的每一成员都清楚如何快速、方便地找到资料，明白工作中的输出的位置，从而形成一个统一的配置管理规定。如图 10-7 所示是测试配置库的一个目录构建样例。

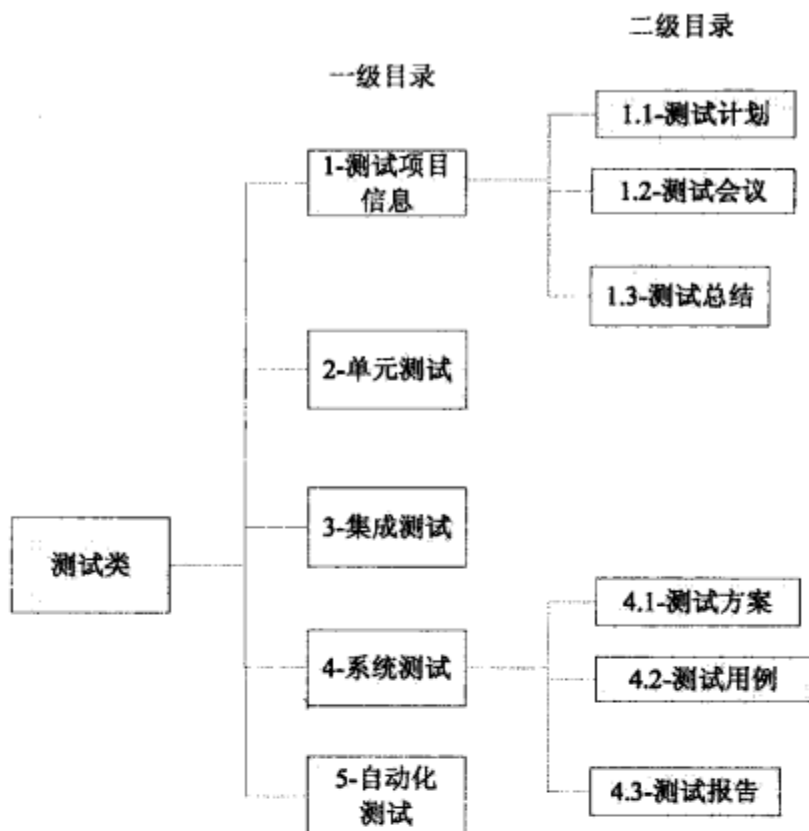


图 10-7 测试配置项目目录

测试人员日常的输出是否每天提交到配置库，还是等到一件事情完成后才把测试输出更新到配置库，在我们实际工作过程中存在着争议。赞成者认为，配置库上需要单独开辟个人的工作空间，用来记录个人的过程测试输出，就像开发更新代码一样，每一次提交代码后的版本并不是马上转入测试，仅是说明版本演变的过程。个人工作目录只有其个人有权限写，是否向他人公开，视情况而定。而反对方认为，个人的平时工作数据是临时的、变化的，没有版本的概念，不具有配置的属性，放在个人电脑中自行保管更合适。而笔者认为，从配置管理的严谨意义上来说，反对方说的有道理。但是如果空间许可，把配置管理本身的用途扩大，充分利用工具本身的优点，就可以记录测试过程中产生的数据，弥补人的记忆思维空间、时间有限等问题。更最重要的是工具还能自动增加版本标识，支持回滚，是一个忠诚的时光机器，这是人工难于做到的。用它又何乐而不为。

测试配置库搭建起来后，就转向应用了。要推动测试团队所有成员都用它，向相关使用人员做宣讲是一个很好的开端（事先需准备好使用说明、宣讲 PPT 等）。中国有句古话，“没有规矩不成方圆”，测试配置管理的使用，同样存在着一些规则，只有用户遵循它，才能更好地发挥工具的价值，使得团队的工作有条不紊，无形中提高大家的工作效率。配置管理需要工具的支撑，同样也需要人的日常管理与维护，下面就介绍配置管理规范中需考虑的一些内容。

- 配置管理的目的：让用户知晓配置管理的目的是什么。

- 配置管理工具：说明采用什么工具进行管理。
- 配置管理员：指定专门的管理员，方便日后用户的联系。
- 角色与职责：
 - 管理员：定义管理员主要做哪些事，如用户的权限分配、配置库数据的备份，以及日常维护等。
 - 测试项目经理、测试经理、测试工程师等，他们的职责又是什么。
- 配置库目录与权限使用说明：可用一个表来罗列，包括目录的内容、使用范围、使用权限等，以统一大家在使用时的数据存放规则。
- 数据安全：定义配置库的数据如何进行数据安全的考虑，采用什么措施等。
- 权限申请流程：当有用户加入或离开时，如何申请加入或申请退出配置库。

小贴士：

SVN (Subversion)：是一个开源的版本控制系统，它管理着随时间改变的数据。这些数据放置在一个中央资料档案库 (repository) 中。这个档案库很像一个普通的文件服务器，不过它会记住每一次文件的变动。这样你就可以把档案恢复到旧的版本，或是浏览文件的变动历史。许多人会把版本控制系统想象成某种“时光机器”。

10.5 漏测分析：测试流程改进的助推器

漏测，对于测试人员来说，是一个非常敏感的问题。相信每一位测试人员，都不希望自己负责测试过的模块存在漏测的 Bug，但是在现实中，我们测试过的模块常会存在这样那样的漏测问题。人们常说“希望总是美好的，现实却总是残酷的”，对于漏测问题，我们该如何客观对待，并从中吸取经验教训，改进测试流程或测试方法，本节接下来将向读者进行介绍。

10.5.1 漏测的定义与漏测分析的意义

漏测，广义上是指软件中的缺陷没有在软件开发过程中被发现，而是遗漏到客户端被客户发现，这个客户包括内部客户（本公司内其他接口部门）及外部客户。狭义的漏测是指在一轮完整的测试后，测试本该发现的 Bug 却没有被及时发现，而遗漏到下一轮测试中。一个模块自己测试完成，转交给另一个测试人员进行交叉测试发现的问题，严格地说也属于漏测。

漏测的影响是很大的，一旦缺陷被最终用户发现，可能会造成用户投诉，常需发布紧急的补丁版本来更改这些缺陷，有的甚至会造成产品的直接召回。但是，如果我们能够将漏测的问题进行分类、汇总、分析，就能在开发过程中发现更多的软件缺陷，使遗漏到客户端的缺陷尽可能少，这将大大降低软件的风险和项目的投入。因此，对于漏测问题，特别是那些有代表性的、严重的 Bug，进行分析总结，改进软件测试的方法或流程，对预防后续同类问题的漏测，是非常有意义的。

进行漏测分析的目的是为了提高软件的质量与用户满意度，使开发、测试过程得到持续改进。具体来讲，就是通过分析开发和测试过程中漏测的缺陷，制定相应的预防措施以避免今后再发生类似的漏测。测试过程的持续改进将提高测试环境和测试执行的效率，降低遗留到用户端的缺陷数和缺陷解决成本，从而提升软件的质量、声誉和销售量。在软件产品开发过程中重视漏测分析，并且参与到漏测分析工作中的团队越多，漏测分析的效果就越好。如果开发和测试团队都重视漏测分析，有时甚至需加入需求设计团队，因为有些缺陷与需求直接相关，并密切配合进行漏测分析工作的话，漏测分析将取得非常好的效果。

形象地说，漏测的 Bug 就像撒网捕鱼时的漏网之鱼，如图 10-8 所示，不同密度的网注定是捕不同的鱼。漏测分析犹如研究鱼网的密度、材质。这个密度就好像我们的用例覆盖度，我们在实践总结中，发现大部分的漏测 Bug 都属这一类。鱼网的材质，犹如我们的测试方法，如果方法上存在不足，即使很小的 Bug 也会漏过去。漏测分析犹如补网与织网，补网即为补充用例，防止再出现同类问题，而织网则是采用新的测试方法或流程来捕获更多的鱼。

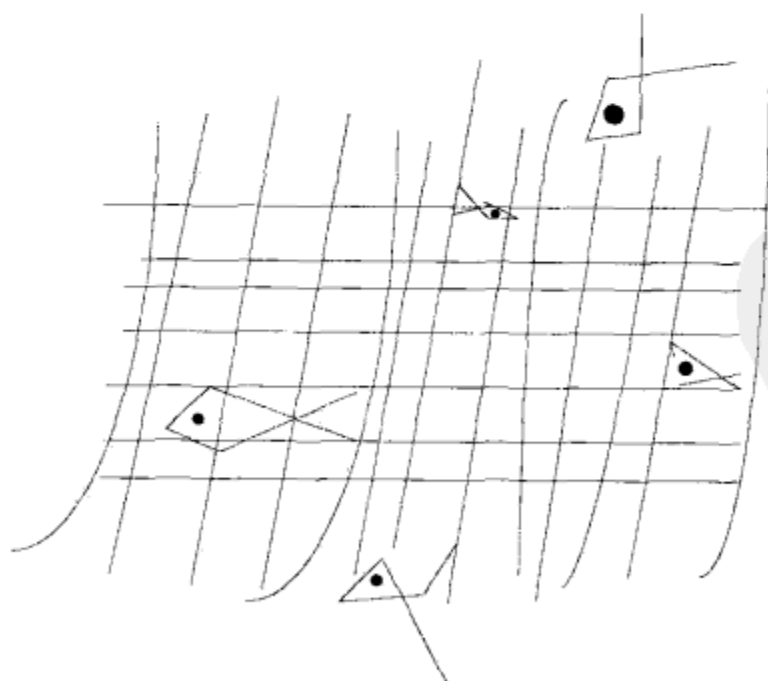


图 10-8 漏网之鱼

10.5.2 漏测问题收集

在进行漏测分析之前，需收集属于漏测的问题。一般情况下都来自于缺陷库，即使缺陷由用户反馈，也要录入缺陷库，作为记录以方便跟踪。为了突出漏测的问题，可以从项目的缺陷库中筛选出来放在一个单独建立的记录漏测问题的库中。如表 10-5 所示是这种表格包括哪些项的说明，主要用于分类统计，有助于分析。

表 10-5 漏测问题表元素

表 格 项	说 明
缺陷编号	首先把缺陷录入到日常使用的项目缺陷库，然后再加入到漏测库中。漏测库中的问题编号，就是项目缺陷库中的编号，是唯一的
所属模块	问题发生的软件部位，可以根据系统实际情况划分模块，比如文件管理、数据库、本地化模块等
简要描述	问题现象的简要描述，一般不要超过 50 字
详细描述	问题现象的详细描述，包括问题的重现步骤等
问题级别	指致命、严重、一般、次要级别，用来确定是否要进行详细漏测分析
漏测产生活动	漏测产生活动是指在软件开发测试过程中，某类被漏测的缺陷最应该在该活动中被发现。设置该项分类的目的是为了便于更进一步地细化分析漏测的活动。开发阶段主要包括原始需求评审、概念评审、设计评审、代码检视、单元测试、模块联调等。测试阶段主要包括功能测试、系统测试、本地语言测试、设备驱动测试、安装测试、性能测试、异常测试等
发现 Bug 的版本	填写发现 Bug 的软件版本号，这个版本号要与代码库中的版本号对应，以便于追溯
问题发现时间	问题录入的时间、格式（XXXX 年 XX 月 XX 日）
问题封闭版本	录入到漏测库的缺陷必须在预期的时间内得到封闭，封闭时需在漏测库填写软件封闭时间、格式（XXXX 年 XX 月 XX 日）
关联产品	漏测缺陷对哪些产品有影响，必须一一列出
缺陷影响	指缺陷对用户造成的影响，例如系统崩溃、业务中止、数据完整性、命令失效、安装失败、可用性、易用性等，对用户将造成什么样的影响，是一种风险分析
问题解决措施	对于等级为致命和严重的问题，在此处进行简要说明解决措施。具体的解决措施，需要在漏测问题解决追踪表中进行跟踪
开发引入原因分析	开发人员进行漏测分析，除了分析产生 Bug 的原因之外，还必须进行相关的影响分析
开发解决措施	开发提出规避遗漏 Bug 的方法
测试漏测分析	测试人员进行漏测分析，除了分析漏测原因之外，还必须进行相关的影响分析
测试解决措施	测试提出规避漏测的方法
需求引入分析	需求人员进行漏测分析，站在需求角度分析漏测原因，以及进行相关的影响分析
需求解决措施	需求人员提出规避漏测的方法，如果漏测的直接原因，与需求相关，如需求变更了，没有告知给测试，则需求需改进变更的控制方法

10.5.3 漏测分析计划

漏测问题的收集是一种积累，当积累到一定的量时，需实施漏测分析的活动，这就需要计划。漏测分析是控制测试过程的一种有效方法，做什么、何时做、需哪些人参加等，需要在计划中体现，实施起来才会有更好的效果。下面是测试计划中需考虑的内容。

1. 分析范围

首先要明确问题的严重等级，并且根据等级确定哪些问题需要进行漏测分析，哪些不需要。例如，按公司定义的问题严重性等级来划分，导致系统崩溃、业务终止或失效的严重问题需要进行典型问题分析，并纳入到漏测故障追踪库里，对漏测故障的封闭方法和措施进行封闭。对于影响面大的严重或有代表性的典型 Bug，可优先从漏测分析库中选出进行分析，这样可以避免在那些对用户无关紧要的漏测缺陷上花太多的时间。也可以只分析那些被关闭和修复了的漏测缺陷，因为如果分析那些没有被关闭和修复的缺陷，可能会漏掉一些至关重要的信息。

2. 活动频度

漏测分析时间不能太频繁，这样不容易抓住漏测问题的重点，可能会导致花很多精力在一些不重要的问题上；也不能间隔周期太长，如果太长，可能造成大量的严重漏测问题没有得到及时分析、解决和跟踪。最理想的频率是 1 个月进行 1 次。

3. 参与人员

一般来讲，漏测分析参与的技术团队越多，分析问题就越全面，解决问题的办法会更加具体可行，漏测分析的效果也会更好。最起码要有软件开发和软件测试团队的相关成员参加。

10.5.4 漏测分析实施

有了漏测分析的计划，接下来是根据计划执行漏测分析活动了。活动最终的目的就是要拿出有效防控漏测的缺陷的方法或措施出来。为了达到这一目的，有如下需要注意的事项。

1. 活动前的准备

漏测分析负责人从漏测库中筛选出若干有代表性问题（复杂且更改难度高的问题，一次最好不要多于 3 个），并把此问题列表发给相关参加人员，如开发人员、需求设计人员，

让他们在会前有所准备。

漏测分析负责人提前准备好开会讨论的场景、设备等。

2. 会前分析

会前开发需搞清楚问题发生的根源；测试需清楚测试为什么漏测，是用例遗漏，还是其他什么原因等；需求设计人员亦可考虑前端需求是否有影响。各专业代表都要进行反思。

3. 达成解决方案

会议中各专业代表分别进行发言、讨论防控方法，以及流程改进的措施等，形成统一的解决方案。

10.5.5 漏测措施执行跟踪

漏测分析活动结束后，对每一个漏测问题都有解决方案后，接下来就是对这些解决方案进行实施了。为方便跟踪，可列如表 10-6 所示的跟踪表。关键点是责任人与完成时间。

表 10-6 漏测措施执行跟踪

表 格 项	说 明
需改进点	分析活动产生的全流程和局部流程改进点（概要描述）
改进措施	详细描述改进点的改进措施，包括改进的详细措施、计划完成时间、验收标准等
完成情况	完成情况包含完成进度和实施过程中遇到的问题。对于遇到的困难需要在漏测分析活动中寻求解决的办法
责任人	对于某个改进点，具体的负责人可以是开发、测试、需求等

跟踪表的目的是为了能更加有效地封闭我们发现的软件开发过程中的问题，跟踪表的维护可以跟漏测分析周期一样。

下面举例如何分析 Bug，发现出问题的地方，开发过程中如何避免再犯同样的错误。

【案例】

某校学生管理系统中有学生成绩管理模块，对其功能有如下定义，语文、数学、英语 3 门主科，任意一科不输入数值时，将不计算平均分，并给出“请输入 XX 科分数！”的提示，录入界面如图 10-9 所示。

图 10-9 学生成绩录入界面

漏测 Bug1: 只输入语文、数学，英语为空，单击“计算”按钮，平均分有值，为语文与数学的平均分。

分析: 此 Bug 是设计没有实现最基本的功能需求，需找设计人员分析，是看错需求，还是编码问题调试不够等。分析原因后，必须找出控制措施，避免同样的问题重复发生。对于测试来说，如果是用例缺失，需补充用例。

漏测 Bug2: 输入 3 门功课的成绩，计算出的平均分小数点为 4 位。保留几位小数点，需求并无定义。而保留 4 位小数点的成绩在日常生活中，基本上没有，建议为 2 位，最后一位四舍五入。

分析: 此问题是需求类问题，需分析设计之初是什么原因没有提出这个很容易想到的问题，如需求是否得到评审。以后如何防控，也需制定可行的解决措施。可以在漏测分析会议上召集各专家讨论，群策群力得出可行的解决方法。

注: 案例中的问题是属比较简单的问题，实际工作中进行漏测分析时会远比例子中的问题复杂，此处只是铺垫思路，进行抛砖引玉。

第11章

追逐软测之理念

前面 10 章以测试技术为线索全面介绍了测试设计工作的相关内容，按原计划笔者该收笔了。但却又感到言犹未尽，特别是前段时间写的一篇博文《浮躁的测试界，为你惋惜!》，首日、首周访问量给笔者带来了惊喜，从回帖信息来看，深得读者共鸣。这让我仿佛看到一个热闹而又鱼龙混杂的集市，市场上人头攒动，小摊小档琳琅满目，到处充斥着叫卖声、吆喝声，卖家、买家翘首以待寻找着自己的目标对象，而更多的是闲逛者，漫无目的地游荡着、迷茫着……作为软测界的前辈，一种分享，抑或能指点方向的愿望越发强烈，于是有了这一章。下面就循着测试朋友们工作或发展方向的一些重要理念，与读者一起分享心得。



苍穹之下，一朵渐渐直起、盛开的奇葩，犹如软测行业的发展、兴起。

11.1 开拓测试管理新思维：测试环境创新

“自主知识产权”、“技术创新”、“管理创新”这些词汇，无论在工作还是生活中，到处都能听到或看到。毫不夸张地说，如今商业的竞争，就是自主知识产权的竞争，创新的竞争。创新来源于不断的实践，离不开源源不断的设计。当今，全国上下都在谈论如何调整发展方向，走自主知识产权的道路。在很多公司，为了鼓励研发人员多创新，多研发出具有独立知识产权的产品，不惜重金奖励创新者、专利发明者。由于软件的特殊性，只能申请著作权，但这并不意味着软件不能创新，软件测试也一样。

国内软件行业普遍规模较小，缺乏开发大型软件产品经验，开发过程不够规范，这决定了国内软件测试行业必须根据国内行业现状，确定软件质量目标和测试策略，而不是照搬照抄国外成熟软件企业的测试模式、方法。正是由于存在这样那样的问题，谈到测试创新，测试行业的很多朋友显得不够自信。测试的创新，并不是只有某个人才有的特殊行为，在我们日常工作中，实际上每个人都在自觉不自觉地实施着创新行为或享受着创新成果。测试的创新与测试的环境有关，这个环境包括内部与外部环境，社会上的一些外部环境之于个人或公司来说，可能不太容易改变，但是在公司内部的环境，是可以考虑变革的。

曾见过一道这样的测试笔试选择题：

软件测试的目的是（ ）。

- a) 发现软件中的错误
- b) 提高产品的质量
- c) 保证产品的质量

站在软件质量层次上，能正确回答这道题，就可以搞清楚测试可以做什么，测试不可以做什么。然而据抽样统计，有很大一部分人给的答案是全选。这让笔者想到这样一个话题测试与质量保证。

现在有不少公司，测试属于 QA（Quality Assurance，质量保证）部门，从公司的组织结构来看，测试本身属质量体系的一个分支，从属于质量部，也很正常。但是有些公司把测试工程师谓之为 QA 工程师，就有点搞不清楚对测试工程师岗位职责的定义了。这与每个公司的质量文化有关，然而，测试与质量保证是两种不同的工程活动。测试是一种被动的方法，它通过多种检测技术来找到潜藏在产品中的缺陷，是通过发现 Bug，解决 Bug 来改进软件质量的方法，是一种事后的补救手段。而质量保证是一种主动方法，它建造一个把预防和质量文化固化在开发过程中的环境，通过开发流程各节点的预防来控制软件的质量，是一种事前进行的预防手段。从图 11-1 中可以看到 SQA（Software Quality Assurance，

软件质量保证) 贯穿于整个开发过程中, 测试是产品研发过程中的一个节点, 它们的角色及职责是不同的。

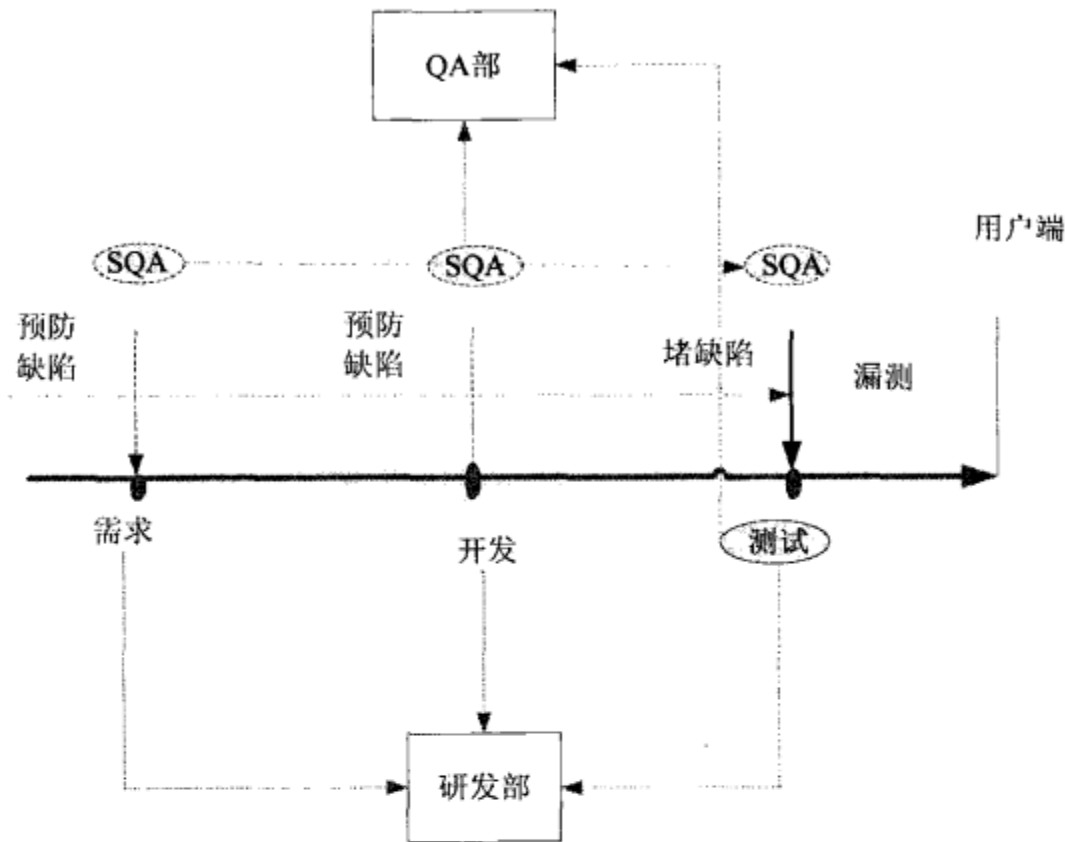


图 11-1 测试与质量保证区别示意图

本书第 2 章介绍的“缺陷的防与堵”，反映的也就是 QA 与 Test 的关系，贯穿整个开发过程的质量保证工作，注定会对最后环节的测试产生影响。质量保证团队努力培养的质量文化是公司内部整体测试的软环境。测试环境的创新，实质也就是能突出缺陷预防的重要性，并可操作的行之有效的一系列解决方案。这些解决方案的构想，虽不是测试团队的主要任务，但从长远来看，从工作效率来看，测试团队作为维护产品质量的一个核心团队，非常需要主动推动缺陷预防流程的改进。质量保证团队的工作往往过多关注流程本身，而对各环节的细节并不那么清楚，测试中发现的问题正好是一个很好的数据，从分析数据出发，反推流程的改进。笔者认为，这种想法本身就是一种缺陷预防分析技术的体现。

在理想情况下，质量在设计阶段已得到充分考虑并被集成进去，并且开发人员只产生了很多的缺陷。那么，此时测试人员该做什么呢？注意那只是理想的情况，事实上，现在我们发现一个 Bug 太容易了。如果真有一天达到理想情况，那么测试人员的工作将更有挑战性，将有更多时间专门发现潜伏在代码深层路径上的问题，也将有更多时间去调查、分析复杂的用户场景，找到很多原本只会被用户在某些场景下才触发的 Bug，从而免除紧急补丁、更新维护所带来的巨大开销。

测试环境的创新，测试质量意识的前置，将使得很大一部分人转向质量保证的角色，

专门分析和实施开发过程的改进，研究缺陷预防技术等。测试与质量保证的关系，下面小贴士中介绍的是微软的做法，从这里我们可以看到及学习到质量保证技术的重要性。提高软件质量的决定因素不是软件测试技术，而是看重软件质量和测试的思想观念。只有把提高软件质量上升到企业战略发展的高度，才能从根本上解决问题。

随着测试环境的改变，测试的下一步该如何走，这应该是很多测试界朋友关心的话题，《微软的软件测试之道》中提到的“软件质量的最大幅提高可能还是要靠预防技术才能实现”，值得我们思考。

小贴士：

微软的卓越工程部设计了一门课程，每隔一段时间就提供给微软的高级测试员，其中大部分材料是关于质量保证技术，而不是测试技术。在很多情况下，高级测试员的职能更像是质量保证而不是测试，主要的职能是提高影响到一组或部门内的所有工程领域的质量实践。

11.2 畅想：测试团队的发展之路

要构建一个什么样的测试团队，首先需要搞清楚你的需求是什么？需要解决什么问题？很多公司在刚开始时，只是为了应对项目的任务，才招入测试人员把事情做完，但随着公司规模扩大，业务的增长，对测试人员的需求也不断递增。由于测试人员每天都在用着不同的测试方法、测试技术来完成工作，对它们的优缺点，以及如何做改进方面都能及时作出反应。然而，牵动着测试团队中每一成员神经的测试组织结构对测试工作的影响，并不是每一个人都能体会到，更何况提出有建设性的意见或建议。而往往决定着一个团队“生死存亡”的恰恰就是这支隐形的指挥棒。

一个组织，首先是因为需要而存在，而不因为存在而需要。测试组织也一样，它有一个发展变化的过程，并不是你想如何部署，就可以如何部署，正所谓“时势造英雄”。测试组织结构的部署，须结合公司的具体需求，以及测试组织的发展状况进行调整、变革。一个一个台阶地往上发展。从宏观上看，测试组织的发展可分为4个阶段，如图11-2所示为测试组织发展阶段示意图。只有经历了上一步的历练，才能走到下一步的起点，测试组织的发展变化亦是一个逐步走向成熟的过程。

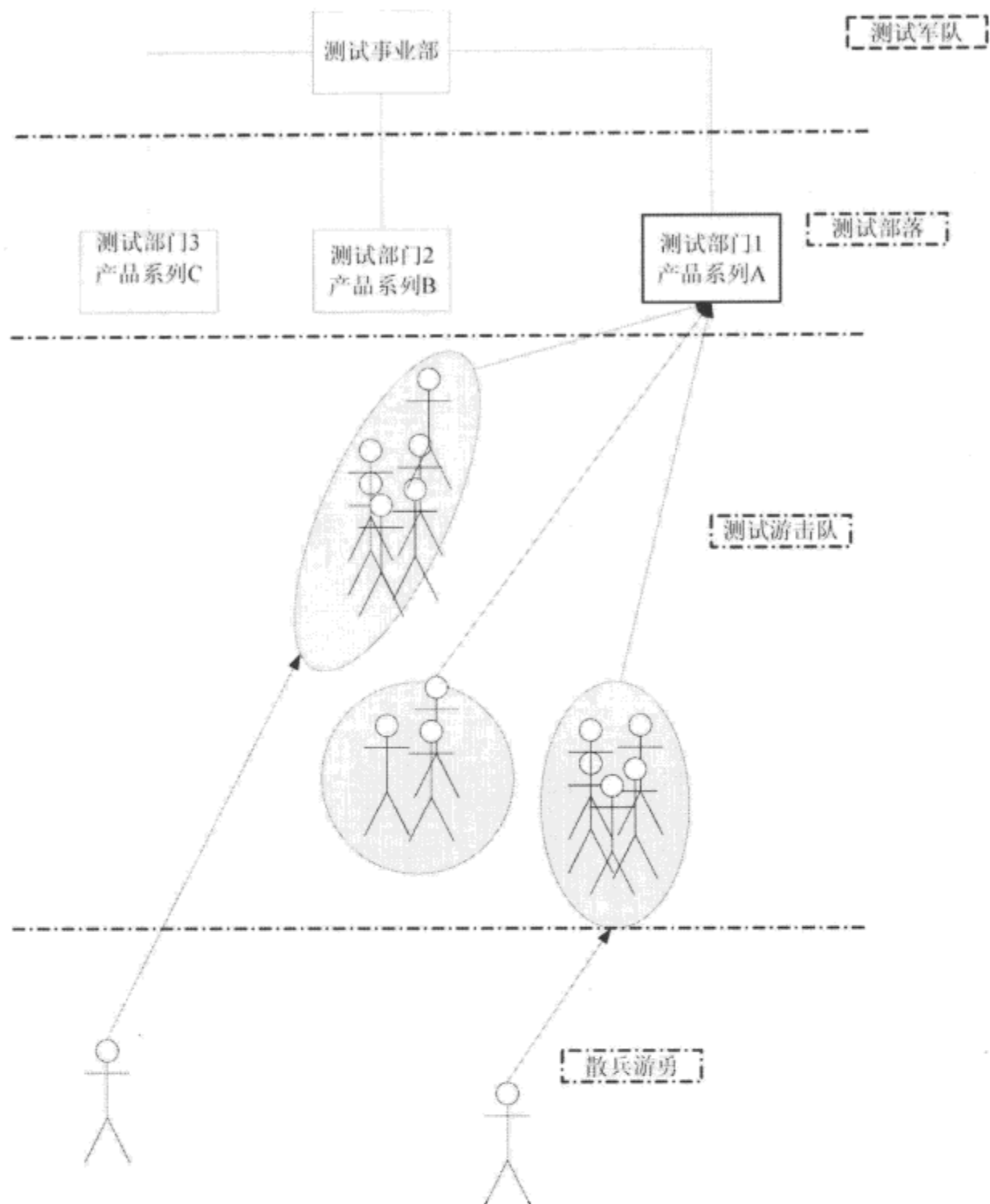


图 11-2 测试组织发展阶段示意图

从图 11-2 中可以看到，一个测试组织从零开始到最后成立软件测试事业部，经历了“散兵游勇”→“测试游击队”→“测试部落”→“测试军队”4 个里程碑阶段，每一个阶段可以说都是在前一阶段的基础之上的跨越。下面对这些阶段的特征进行分析，以便需要时组建合适的测试团队。

11.2.1 散兵游勇年代

公司成立之初，一般只有几个员工，软件开发的模式也多属于作坊式，通常是开发人

员自己编码自己测试，最后把产品卖出去。后来，业务多了起来，对产品的质量要求高了，老板便开始雇用专门的测试人员，此时测试人员进来后还可能身兼数职，如兼软件的配置管理员、用户手册编写员等，如图 11-3 所示。此时，测试人员可能就你一个人，也可能还有另外半个人，在测试任务紧张时，行政秘书也加入进来，这些情况都不足为奇。毕竟作为企业老板，首先是要把员工养活，在公司有赢利的前提下，才能谈更进一步的发展。这种环境下，测试人员并非专职的测试人员，他们犹如散兵游勇，有可能连岗位名称都是怪怪的，如测试员、软件工程师，有些甚至与开发人员一样，被称为软件开发工程师。组织上属于软件部，与软件开发人员一起，由软件开发负责人管理。身陷其中的朋友，可能正感到困惑、迷茫。但是正因为公司测试方面的不成熟，才给了你很多的发展空间，如果公司发展前景可观，这种状态坚持一到三年后，你便是公司测试方面的领军人物了。

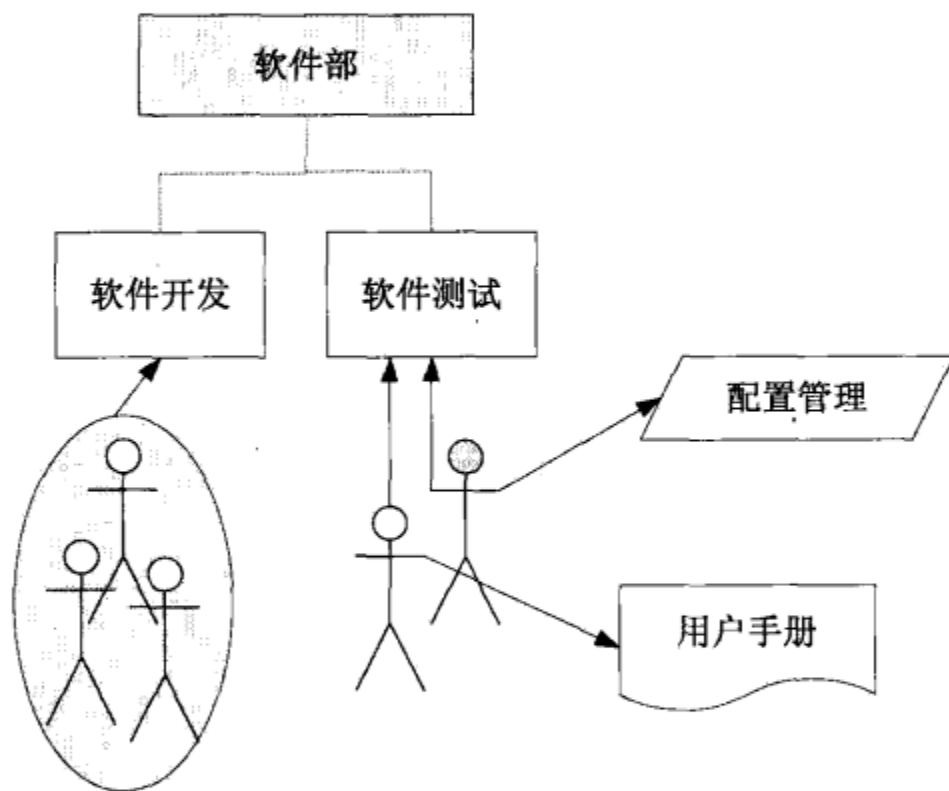


图 11-3 散兵游勇测试年代

11.2.2 测试游击队

为什么说是测试游击队？它与抗战时期的游击队有哪些类似之处？

测试游击队常出现在公司业务繁忙的非常时期，不同的几拨人，同时忙于不同的项目。每支游击队各司其职，为某类或某个项目服务，资源隶属于项目，是项目团队的一股力量，如图 11-4 所示。据调查这是目前很大一部分公司的组织管理模式。

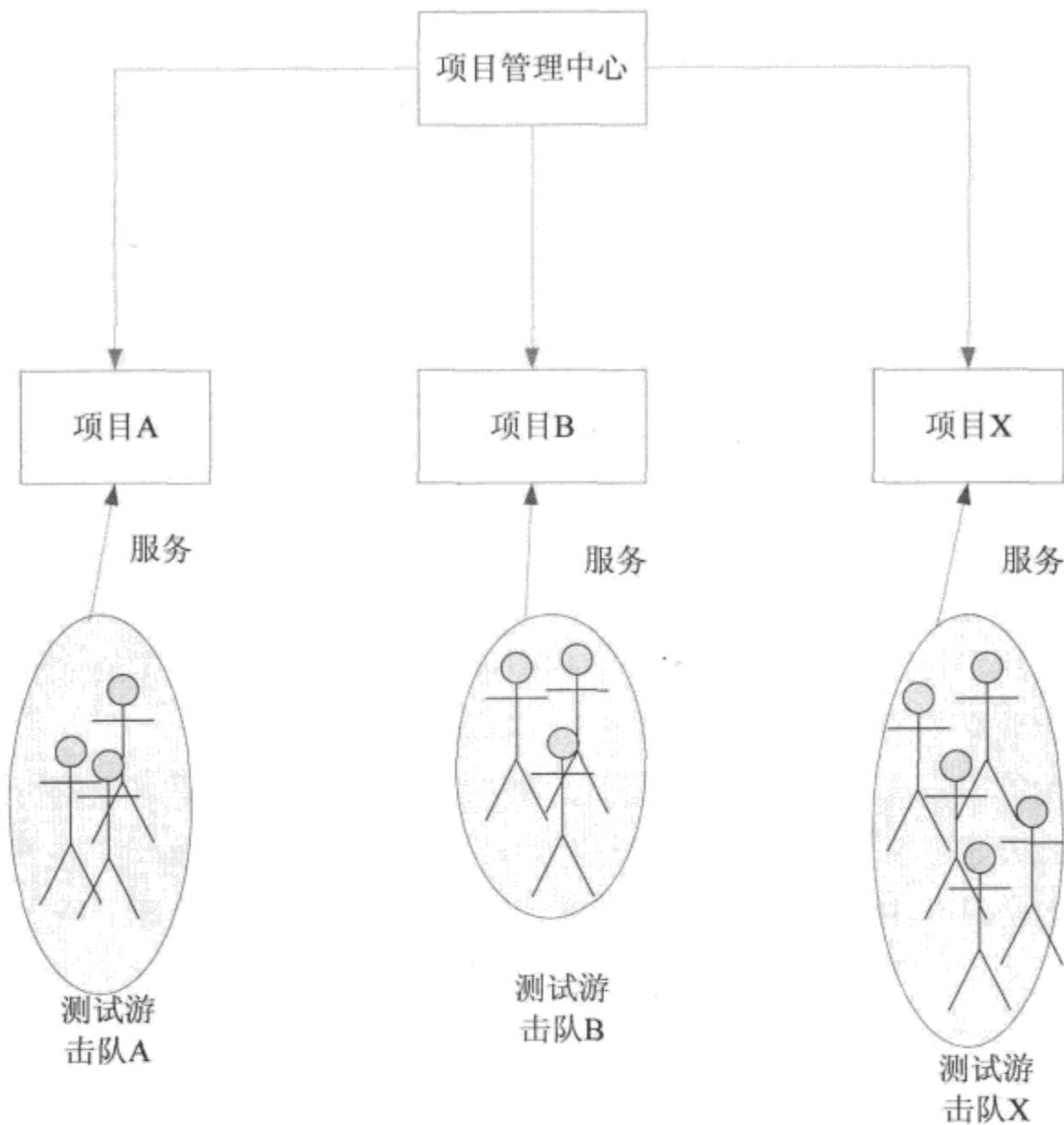


图 11-4 以项目为中心的测试游击队管理模式

各测试游击队之间，他们虽然都是在做测试，但很少有技术上的交流或几乎没有。目光中只看项目，项目成功，则测试成功。一个项目测试完成，马上转向另一项目的测试，犹如抗战游击队的打一枪，换一个地方，如此往复。几年后，公司产品系列已初成规模，市场竞争亦日见激烈，需要削减项目成本来降低投入已迫在眉睫。于是领导们开始绞尽脑汁思考，哪些技术可以继承、哪些模块代码可以复用、哪些测试流程可以简化等。但此时才发现，经历过几年多个项目历练的测试游击队，除了人多了点外，在技术上还谈不上有积累。一切以项目为中心，项目做完，测试即结束。每测试一个新项目，测试用例需重新设计，某个项目中曾经犯过的错误，在另一项目中继续犯。测试涉及的业务知识面倒是越来越广，但在测试技术深度上的掌握却停滞不前。这也正是这种以项目为中心构建的测试团队的特点。

游击队之间严重缺乏交流，测试大团队的概念仍是个空壳。如何改变这种现状，把几

支游击队的力量合并，转变游击队为兄弟连，即凝聚所有测试力量形成一股合力作战的测试部落，这对于管理人员来说是一个组织结构需如何变革的挑战。

小贴士：

游击队：执行游击作战任务的武装组织。通常组织简单、装备轻便、行动灵活，同当地群众有紧密的联系。

11.2.3 测试部落

在面试测试工程师时，有很多应聘者问过笔者同样一个问题：“你们公司有独立的测试部门吗？”诚然，大家都关心这个问题，从中也可知这个问题在众多测试朋友心中的重要位置，是否有独立的测试部门对测试人员来说真的就那么重要吗？

在软件开发生命周期中，先是有设计，然后再有测试，测试是开发的下一道工序，这个原理测试朋友也都知道。微软是众所周知的世界顶级软件开发公司，它创办于 1975 年，最初同样也是没有测试人员。据肯·约翰斯顿（微软资深测试经理）所言，微软最早的测试工程师是一位 1979 年加入微软的高中实习生，叫罗伊德·福克林（Lloyd Frink）（见小故事一则）。然后在 1983 年，微软 Archive 产品开发组雇佣了第一个全职的软件测试工程师，接着在 1985 年又招聘了一批测试工程师。测试作为一个正式的职称，并有自己的职业发展轨道却是 20 世纪 80 年代后期的事情了。

【小故事】也许我们需要在软件发布之前测试它的功能

之前我见过比尔几次，其实我就是这么得到实习生的职位的。那年，我正要进西雅图私立湖畔学校（Lakeside School）高中部学习。我妈妈认识比尔的妈妈玛丽（Mary），在一次学校组织的拍卖募捐活动中，两个妈妈闲聊讲起自己的儿子都喜欢计算机。又正巧我和比尔都在会场，她们就介绍我们认识。那时我 14 岁，比尔 24 岁，我们决定一起吃中午饭。几个星期之后，我和妈妈来到微软，和比尔及他的姨妈丽碧（Libby）一起吃饭。丽碧在学校比我高一个年级。我把我编写并卖出的计算机游戏演示给比尔看，他就给了我暑期实习生的职位。这就是整个事情的开始。

第一个实习期，我主要给格雷克（Greg Whitten）工作，帮助他测试 BASIC 的编译器功能。我们把很多的 BASIC 程序在编译器下运行，看是否能够得到正确的期待值。

——罗伊德·福林克（Lloyd Frink），前微软员工及 zillow.com 的创始人

目前，国内的软件公司大都是一些规模不大的中小公司，软件测试行业兴起，也是近

几年的事，从个别的“散兵游勇”到发展为具有独立职能的“测试部落”，需要有一个过程。完成这个过程的快慢与公司的发展步伐有密切关系。只要公司的发展势头强劲，又何必太在意暂时性的开发领导管理你或你与测试团队。目前有一些测试朋友认为开发领导不懂测试，不理解测试，特别是开发经理在评估 Bug 遗留与否时，既充当裁判，也充当队员，风险控制上会引发问题。即便是事实，抱怨也解决不了问题，组织结构的存在总有它的道理。反过来，有一点测试朋友都很清楚，由于工作性质的原因，测试人员的编程技术往往处于弱势地位。今日属于散兵游勇的你，只要抱定坚定的信念，努力提高自己的技能，为公司作出更多的贡献，尴尬的行政关系终将过去，成立独立的测试部门是迟早的事情。

独立的测试部门是指与开发部门在行政上并列的一个部门。有了专门的测试部门后，一般情况下，测试人员少则会有几十人，同时拥有多个测试技术组，测试分工更明确。如功能测试组、性能测试组、自动化测试组、测试工具开发组、前瞻性测试技术研究组等，如图 11-5 所示。

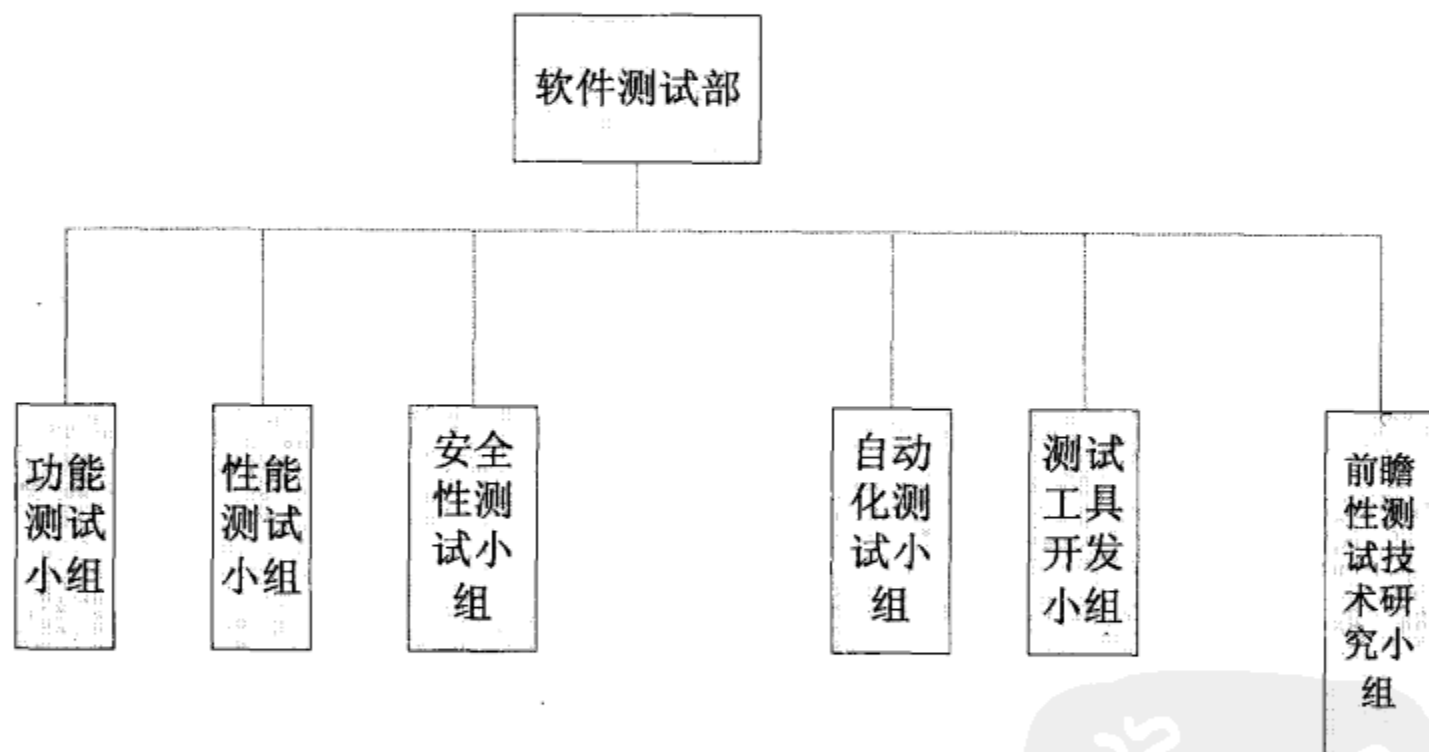


图 11-5 独立测试部门测试小组分工示意图

当一个公司的业务发展到一定的程度，多个项目并行开发时，多支测试游击队便应运而生。就每支测试游击队而言，实际上他们也是一个小团队，通常由一个测试小组长或项目测试负责人带领着。然而，以什么为核心组织测试团队，需要领导者审时度势，深思熟虑。如图 11-6 所示是一个以技术为核心的测试团队组织模式。

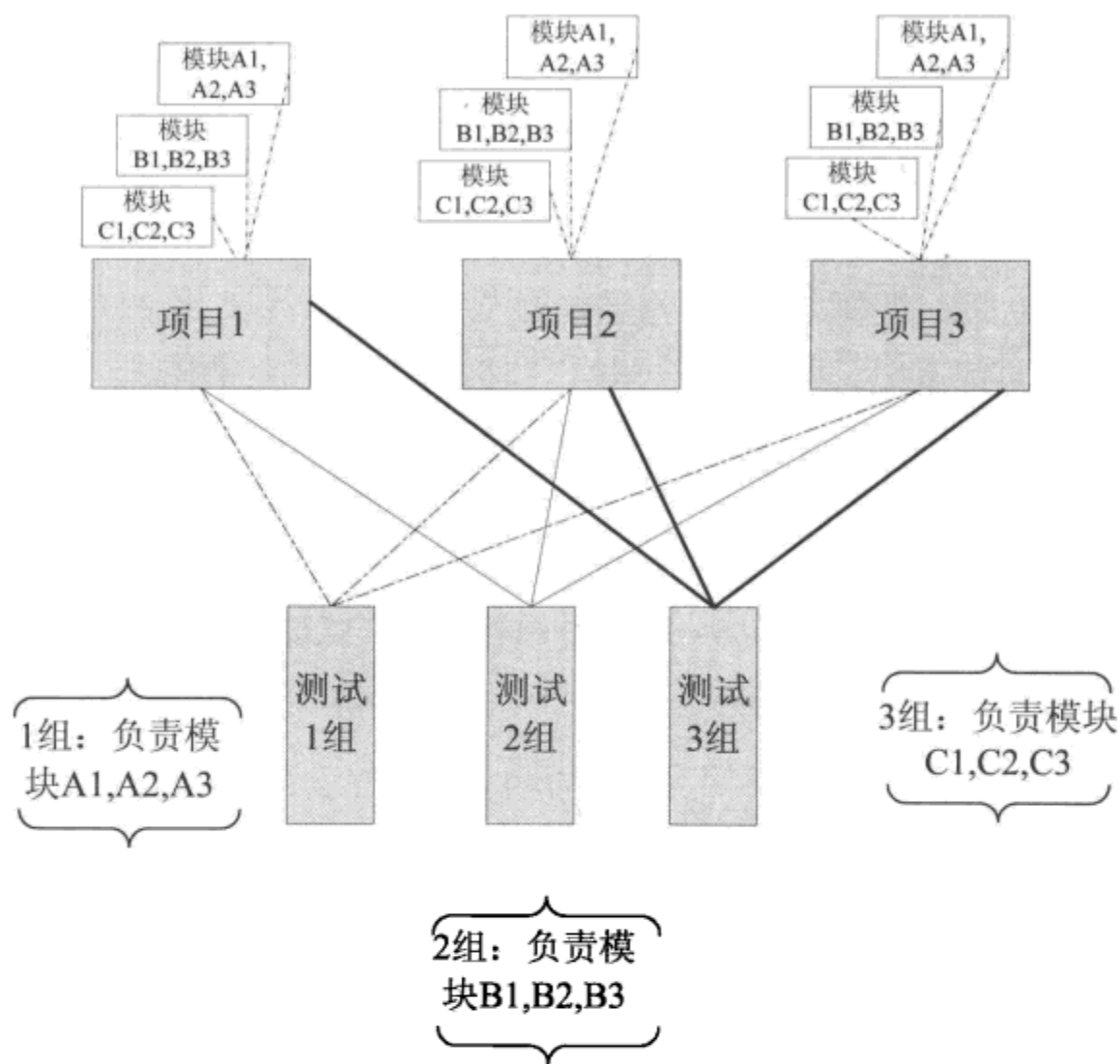


图 11-6 以测试技术为核心的测试团队组织模式

在以技术为核心的测试团队组织模式中，一个小组负责某类业务功能或某特性方面的测试，在不同项目中，他们仍负责同类业务或相同特性的测试。与以项目为中心的测试组织模式相比，具有以下优点。

- 模块化：测试分工细化，一部分人始终测试某几个同类或相同特性的模块，有助于测试在深度上的挖掘，如通信协议、数据安全性等朝某类技术领域的纵向延伸。
- 专业化：正因为分工的细化，使得大家可以集中精力专注于某业务或某技术方向的研究。测试目标聚集在测试的深度，而不是广度。
- 跨项目：测试人员眼中不再视项目为唯一目标，看到的是要负责测试的模块，而且是各个项目中的同类模块。冲破了项目墙的壁垒，透过项目墙，在 A 项目中测试模块 A 系列的测试人员，在项目 B 中的同类模块仍是由他们测试，随着经验的积累，犯错误的概率必将越来越少。
- 平台化：由于在不同的项目中都是测试同类或相同特性的模块，俗话说“一回生，二回熟，三生巧”，自然会考虑到测试代码、测试用例的重用及测试方案的继承。

容易沉淀平台化的测试资源，包括可用于各项目的测试代码、公共用例等。在此基础上，项目测试所花人力、物力成本将降低，更重要的是将会使测试周期有明显的缩短，使得研发的产品能尽快上市。

除了上述优点外，这种组织模式也存在一些不足：

- 应用软件与系统之间的接口，应用软件内部模块与模块之间的接口关系，有些并不好区分。规划在哪个技术组会更好，或者本身就分不开，这样容易造成软件内外的接口测试不足或过度。测试不足，是由于各技术组一般情况下重点都在关注自己所负责的范围，从而很容易漏测各模块接口地方的测试或全局性的系统测试。反之，相关接口的技术组都对接口进行测试，这样就造成过度测试。
- 业务理解不全面，因为只测试本技术组的几个模块，眼中没有系统的观念或观念淡薄，目光很有限，使得用户场景的测试会考虑不足。

小贴士：

什么是团队？

A team is a group of people who agree on a goal and agree that the only way to achieve the goal is to work together. (Park, G.M. 1990) 翻译过来就是“团队是为了一个共同的目标而一起努力的一群人”。这里面涉及 3 个要素：共同的目标，一起努力，一群人，这三者缺一不可。

11.2.4 以项目经理为核心的组织模式

任何一个团队发展的历程都是从无到有、从弱到强的一个过程，测试团队的发展也一样。测试部作为一个独立的职能部门时，有自己的职责范围，与开发部门并行隶属于上一层的管理机构。公司的目的是赢利，公司在发展的过程中，成立一个新的团队，是希望这支新团队能为公司带来更多的效益。当公司的产品发展到多个系列时，会出现多个产品线，而当产品线发展到一定规模时，很多公司会按产品线来划分，形成一个个独立的事业部。测试部门可按原来的模式与开发部门并行纳入各事业部，也可以独立出来成立测试事业部，专门为各产品线的产品测试提供服务。此时，对测试来说，所发生的变化是前所未有的，是颠覆性的。它直接隶属于公司总部，服务于公司内部，相当于做内包业务，如图 11-7 所示。

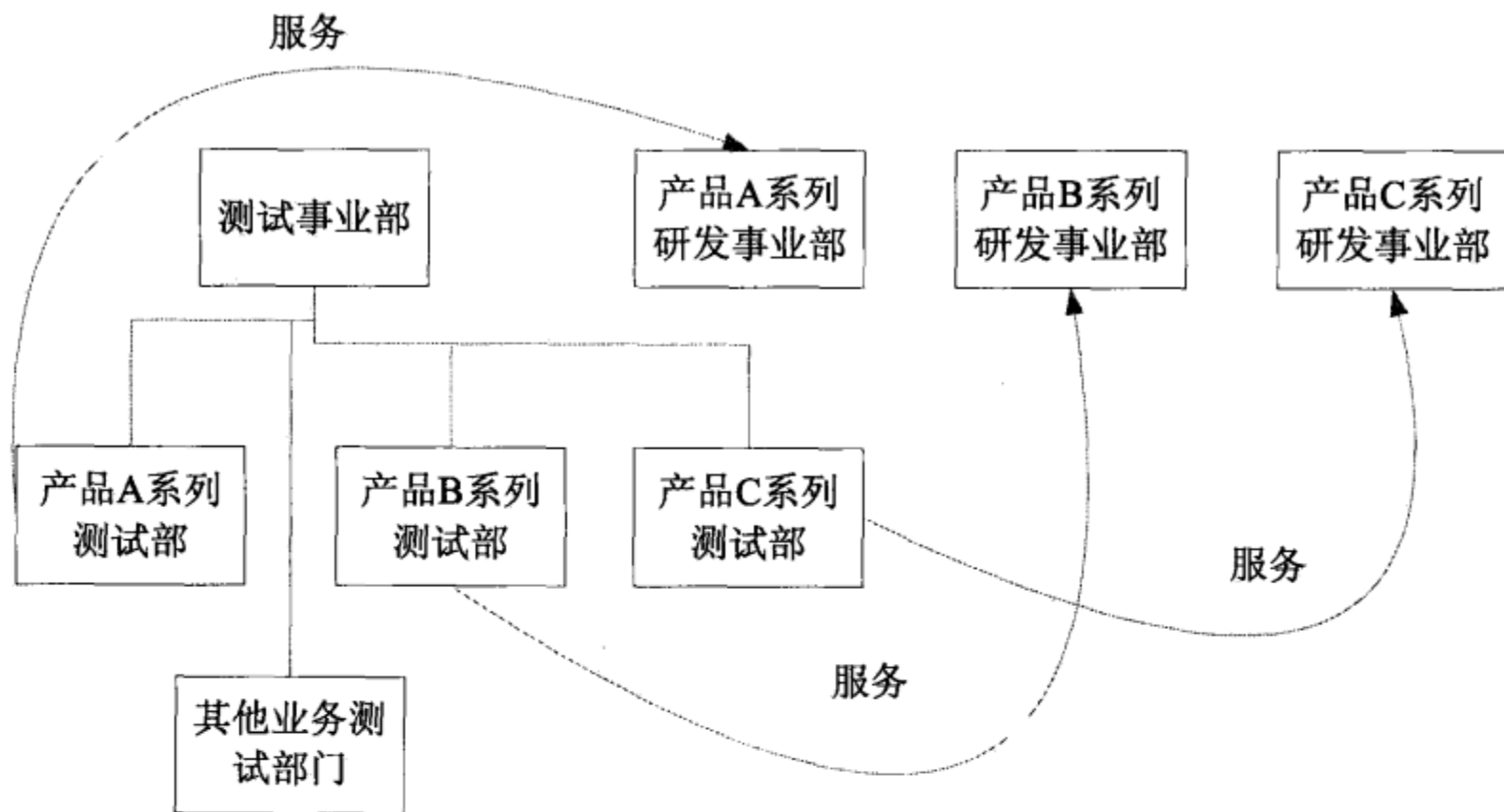


图 11-7 测试事业部组织结构示意图

公司下面成立事业部，是公司管理体制的重大改革，这让笔者想起 20 世纪 70 年代末 80 年代初，中国农村开始实行家庭联产承包责任制的改革。改革是具有划时代的历史意义的，彻底解决了吃大锅饭在生产队里累死还吃不饱的问题，农民的生活发生了翻天覆地的变化。当一个公司成立了独立的测试事业部后，由于需自主运营，自负盈亏，服务的方式可以变得灵活，在满足公司内部测试服务的基础上，还可以外接业务，带来新的利润增长点。或许这是事业部独立后的机会，类似于在不违背“国家政策”（公司制度）的前提下，要“过”（经营）得更好，前方的路就在自己脚下，充满着机遇与挑战。

成立测试事业部后，测试相当于成了开发的外包单位，只是这个外包很特殊，严格地说，是内包。也正因为这一关系，会引入一些新的问题。当开发与测试同属一个产品线（事业部）时，即便不同的部门，开发与测试也会经常一起探讨某业务或某模块的细节和实现，这样有利于测试人员的前期介入。这种前期介入包含很多方面，例如测试人员对设计文档的评审检视、测试分析与设计的分工合作、测试人员前期参与代码走读、集成测试等。但反过来，测试和开发人员过于“亲密”也会造成测试无法扮演好“黑脸”的角色。甚至有些开发人员与测试人员商量好，有些 Bug 直接告诉开发人员，不提到 Bug 库中，在私下私了，因为有不少公司开发人员的绩效是与产生的 Bug 多少及 Bug 的严重性挂钩的。

小贴士:

事业部: 一种公司的组织结构, 在公司总部下, 设立若干个自主营运的业务单位, 即事业部。这些事业部, 或者是按产品来划分, 或者是按地区来划分。每一个事业部都是要对成本、利润负责的利润中心。

选择什么样的组织结构才会实现最大化测试效益呢? 答案没有定论。需要结合开发的组织结构、开发模式、产品复杂度、需求稳定度、测试人员构成、组织的测试经验积累、当前产品测试的软肋是模块还是系统等因素综合考虑。

11.3 测试设计理念至上

理念, 就是我们对某种事物的观点、看法和信念, 是一个人的思想反映。好的理念不是一天两天就可以形成的, 它是一种潜移默化的认知能力。测试理念也一样, 具体指哪些方面的认知, 没有统一的标准答案, 它永远在适应环境、社会的变化而在不断地发展变化着。测试的价值是什么? 测试有前途吗? 需要有测试设计吗? 测试能创新吗? 诸如上述这问题的答案就是一直在变化着的理念。

每一个好的公司都有一个好的属于自己的理念 (愿景), 测试作为一个新兴的行业, 同样亦有它的发展理念, 与公司不同的是, 它不属于某人或某公司, 它是一个行业发展所需的, 属于这个行业中的所有人。下面是一个关于测试用例设计理念的案例, 希望对读者有所启发。

【案例】需不需要设计测试用例

记得在 2002 年, 笔者在一家跨国外资企业工作, 当时软件测试从业者并不多, 自己所在的部门, 研发人员 100 多人, 质量保证工程师 6 人, 其中有两人负责整机测试 (硬件、机械方面), 有 4 人负责软件测试方面的工作, 笔者便是其中一位。工作一年多后, 公司管理层变更, 新调来一位研发总监直接负责我们, 幸运的笔者被他提拔为 QA 部主管。由于是新来的领导, 对他的领导方向并不是很清楚, 但能感受到他对笔者的信任与重视, 很快笔者也成了他的得力助手。随着公司业务的增长, 规模的扩大, 人员的变化, 原来 4 位测试人员已明显不足。于是与总监商量如何壮大、培养测试队伍, 如何改进现有的流程提高质量。然而, 不知他是不了解测试还是由于其他原因, 竟然要求笔者不用设计测试用例, 直接写简单的 Checklist 清单就好了, 这样可以节省很大一部分测试时间。

古人云“欲速则不达”，笔者问自己：亲身经历的不写测试用例的惨痛教训（详见第7章“漏测一个提示界面，不仅损失158万”中的介绍），还想来第二次吗？尽管总监不赞成这样做，但为产品的质量着想，我们在下面还是设计了详细的测试用例来执行测试。

现实中，有很多事情你可以不做，但也有很多事情你不可以不做。时隔多年，回首思索何谓测试的理念，案例中的坚持用例设计的原则便是核心理念之一。测试的底线是什么？从测试设计的角度看，笔者想这也是作为一名合格的测试工程师的底线了。

正确的理念，是一束束希望之光，能为你照亮测试道路上的坎坎坷坷，如图11-8所示是一个示意图。

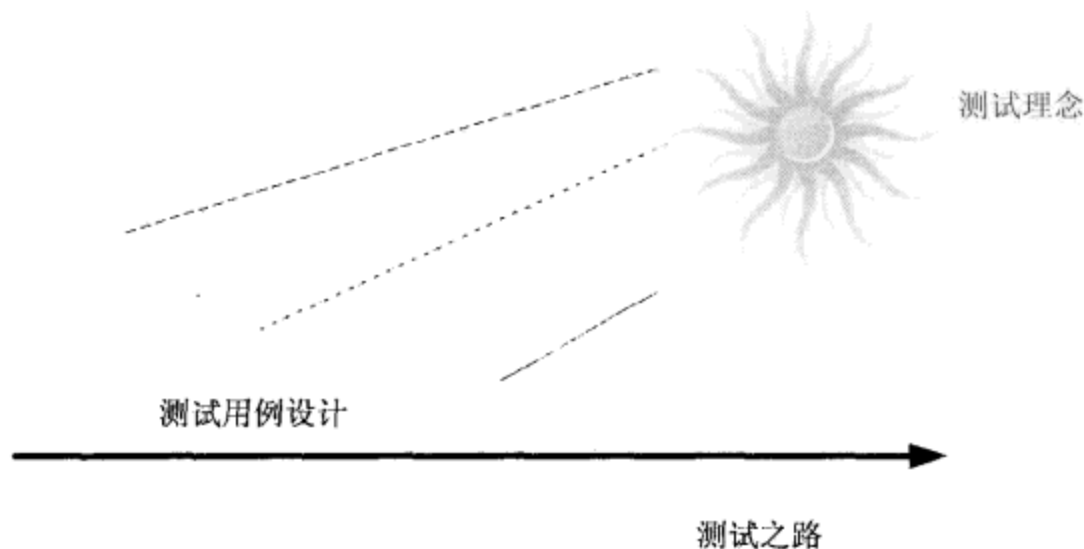


图 11-8 测试理念的作用示意图

11.4 挑战测试新技术

谈起测试技术，测试人员都能如数家珍，例如：等价类、边界值、错误推测、因果图等测试用例的设计方法；黑盒测试与白盒测试方法；功能测试、性能测试、压力测试等测试类型；手工测试与自动测试的策略。工作过程中，需根据实际情况，在不同阶段采取不同的测试策略与方法。

毋庸置疑，测试的关键目标是如何在有限的时间内提高或改进软件的质量，因此测试技术的创新需围绕测试的效率与测试质量来发挥，也就是如何使得测试做得又快又好。如图11-9所示，一个中心四个基本点，中心是软件质量，位于X轴方向的测试成本与测试时间属于测试效率的元素，位于Y轴方向的测试覆盖率与测试有效性属于测试质量范畴的元素。测试的创新可以从这四个基本点出发进行发挥，创新的内容不仅局限于测试方法，可以是测试流程、测试模型等，只要对这四个基本点有改进作用，都是对核心目标软件质

量有贡献的。下面就围绕这四个基本点介绍几个可创新或改进的点。

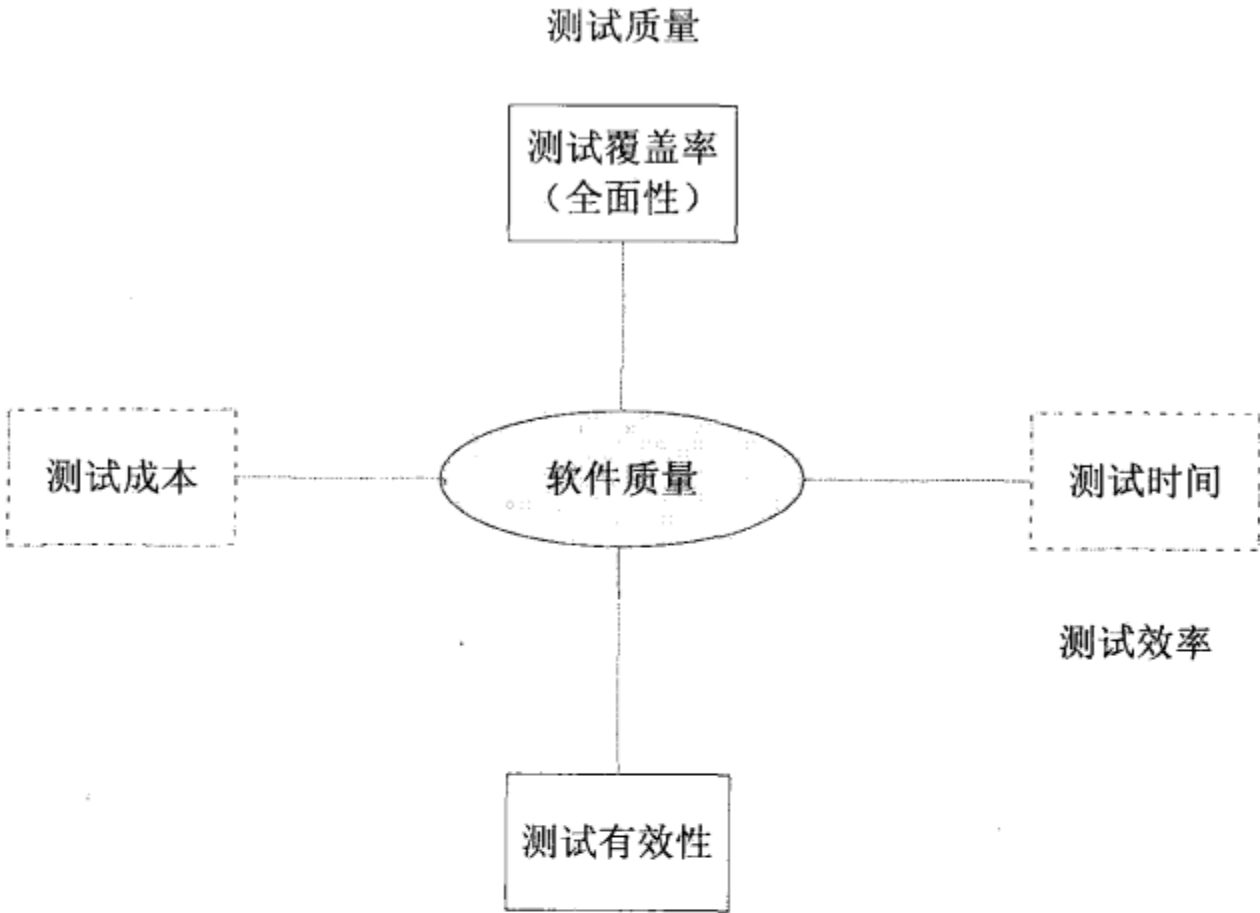


图 11-9 测试技术创新方向示意图

1. 杂交测试方法

有时候，一些公司招聘测试岗位的人员时会说招白盒测试工程师或黑盒测试工程师，就是把测试方法放在岗位的前面修饰岗位，以突出该岗位的特点。测试岗位叫什么，本质上只是一个名称而已，无可非议。但让笔者出乎意料的是，恰好这个岗位的定义限制了一些测试工程师的手脚。曾经，笔者与一位测试工程师讨论测试方案时，谈到某测试对象的测试方法采用白盒的方法会更有效率，没想到该工程师这样回答：“我是功能黑盒测试工程师，白盒测试方面的考虑不应该由我考虑吧。”这是一个小插曲，不再详细讨论。

我们都很清楚，每一种测试方法都有它的优劣之处，为了提高测试的有效性，笔者比较赞同于使用杂交的测试方法，多管齐下，但又有的放矢，取它们之最优组合。比如针对某一个测试对象，如果仅采用黑盒测试方法覆盖率难于保证，则结合代码测试等白盒手段，仅是部分结合，不是全部代码重新采用白盒的方法来测试，这种杂交式的测试方法，可称之为灰盒测试，如图 11-10 所示是一个示意图。往往这种方法在测试质量与效率上都可得到满意的结果。灰盒测试只是杂交测试方法的一种，与此类似的交叉使用多种其他测试方法的手段，也都属于杂交测试，如系统测试与单元测试相结合、手动测试与自动化测试相结合等。

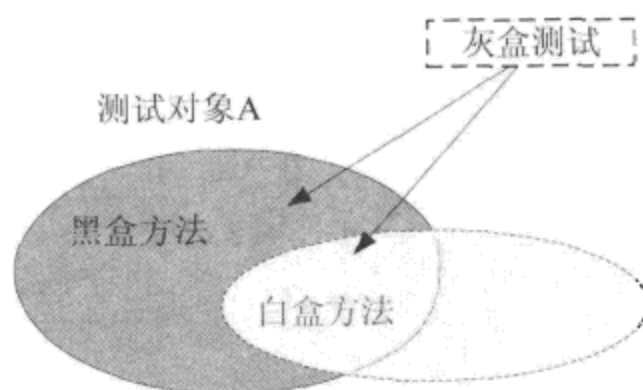


图 11-10 杂交测试-灰盒测试示意图

关于灰盒测试的更详细介绍，请见本书第 5 章的“活用灰盒测试”部分。

2. 自主开发测试工具

测试工具种类很多，有大有小，作用也各不相同，有商业的，也有开源的。测试中常用的缺陷管理工具与用例管理工具，主要是在改进测试流程的规范管理上为测试做出贡献。还有如自动化测试工具，主要能节约测试时间及人力成本，从而提高测试效率。这里要介绍的“自主开发测试工具”，是针对工作中的具体测试需求而言的专项测试工具。如测试数据生成工具，一般情况下在市场或开源社区中是找不到完全满足你需求的工具，为了测试的覆盖率与测试效率，很有必要自主开发。下面是一个例子。

【案例】测试数据生成工具

某公司自主研发数码相机，相机内置存储空间为 1G，保存的图片文件格式是公司内部设计的特有格式，如果不符合这种格式的文件，相机是读不出来的。测试工程师 Jane 被安排测试数码相机的软件，其中内置存储空间的边界测试需模拟用户场景进行测试。但是 1G 的空间，按默认一张照片约 1M 的空间来算，需要拍 1000 张照片。依靠手工方式拍照片积累数据的方法显然不是一个好的方法（1 分钟拍一张的速度，也要消耗人工近 17 小时）。

Jane 这样分析后，找来开发的设计文档，并对相关代码进行理解、分析，找出文件保存的具体结构信息。然后自行用 C++ 开发了一个所需文件自动生成的工具，且工具可以做到你要多少个文件，即可生成多少。测试时只要把所需文件放入相机内的指定位置，测试路径很容易得到满足，使难于模拟的用户场景测试得到了有效解决。

3. 构建公共用例库

构建公共测试用例库，就像代码的模块化。用例库中的用例如果也能达到模块化，当新项目上来后，相同或类似的功能可从中移植，便可节省大量的用例设计时间。

当产品间的需求相同时，用例移植过来是很自然的事。然而不同产品间的需求细节常会不一样，对于系统功能测试而言，由于用例采用的是文字性的描述，描述的粒度要如何

确定，一直是困扰我们的问题。如果描述得细，则复用得少，描述过粗，用例的输入与输出不确定。

关于用例的复用，请读本书第 7 章中“设计可复用的用例”部分的介绍（对此话题如有兴趣，欢迎发送电子邮件至 ch_testinfo@126.com，笔者会及时回复）。

创新的思维来源于不断的实践，在工作过程中不要怕遇到问题，有问题是好事，可以为自己的成长创造条件。甚至有些时候，可以去特意找问题。当然，遇到问题后，不总结、不思考就解决不了问题，更谈不上创新。古人云“实践出真知”，创新固然重要，但创新的结果依然要回到实践中，也就是应用在测试工作中，检验对测试的效率与测试的质量所起到的效果。

11.5 测试是不可或缺的“一条腿”

无论是网上的虚拟软测社区，还是在实际的测试工作中，常能听到这样的声音：“测试人员与开发人员的比例悬殊大，公司重开发轻测试”、“测试没有技术含量”、“测试有前途吗？”等等。这与对测试这个专业的认识深度有密不可分的关系。就拿测试人员与开发人员的比例这一点来说，在国内可能鲜有测试人员会多于开发人员的企业。

如表 11-1 所示是来自国内知名软件测试网 51testing 在 2009 年的调查结果。

表 11-1 2009 年公司测试人员与开发人员比例

	1:1	1:2	1:3	1:4	1:5	1:6	1:7	1:7 以上
2008 年	11%	14%	20%	11%	12%	5%	5%	22%
2009 年	9%	14%	20%	14%	13%	5%	5%	20%

如果只凭测试人员与开发人员在数量上有一定差距，就下结论说公司不重视测试，笔者认为过于草率了。我们都很清楚，追求产品的利润是一个公司的商业目标，无论是测试还是开发，对于公司来说，就是为其实现商业目标的资源。在公司发展的每一阶段，商业目标是不一样的，对产品的质量要求也是不同的，这就影响着对测试投入的成本。例如：在公司成立初期，也许公司是为了快速上市而占领重要的市场，急需有上市的产品，并不追求一个完美的产品，此时开发人员是公司事业发展的关键，需要大量的开发人员。而对测试一开始要求不需特别高，只需要对已承诺给用户的需求进行确认测试。而当公司的业务市场打开，需要在行业中以品牌取胜时，质量无疑需要上一个台阶，这时就需要更多的测试人员，且对测试人员的要求更高。当我们这样想，相信你会发现测试的处境其实并不

像上面说的那样糟。

在软件业，需求、开发、测试三者之间的关系，就像一个“人”字结构，如图 11-11 所示。需求是左边的一撇，犹如人的左腿，开发处于中间的核心位置，测试是右边的一捺，犹如人的右腿。一个人走路时需要用两条腿走路，左腿迈一步，右腿迈一步，一个人才能轻松地往前走。需求是开发设计的依据，也是测试验证设计是否满足需求的依据。开发是实现需求的关键，是建造软件“帝国”的英雄，这个无可置疑。测试是软件质量的最后把关者，也是软件赛场上的裁判，是整个软件开发过程中不可或缺的强有力的另一条腿。



图 11-11 测试在“人”字结构中的位置

需求、开发、测试三者紧密联系在一起，才能构成一个完整的“人”字，三者各有其位，缺一不可。

11.6 通向“罗马”的测试之路

俗话说，百门通不如一门精，一个人能否成功，关键在于能否持之以恒地坚持做一件事。做一行爱一行，做测试也一样，爱它，才能把它做得更好。中国有句名言“三百六十行，行行出状元”，每一位测试朋友都有机会成为测试状元。下面是成为测试状元之前的重要认识，与读者一起分享。

11.6.1 识别自己——英雄不问出处

在软测领域，正从事测试的朋友，有很多是转行过来的，如原来是做技术支持的，或者做软件开发的、做行政助理的、做老师的。也有不少毕业生踏入社会第一份工作就是做软测的；而还有一些人是所谓“阴差阳错”加入进来的。在过去的面试经历中，曾经有一些人谈起自己是在稀里糊涂中走上测试岗位的。例如有一位应聘者谈到，自己是某公司到学校进行校园招聘时被招进来的，当时说是招聘软件工程师岗位，具体做什么工作并不清

楚（当时是想，能拿到 Offer 已是很不错了，对日后具体做什么工作并没想太多）。进入公司培训一段时间后，就被分配到了测试岗位。

无论之前做什么职业，也无论你是从学生直接过来，还是有丰富的工作经验者，既然踏上了测试这条路，就要有“既来之，则安之”的心态。软件测试是一个前景一片光明的朝阳行业，消极、怀疑、抱怨只会给自己在前进的路上拖后腿。古人云“英雄不问出处”，只要你能在测试工作中找到自己的位置，实现自我价值，又何必太在意自己是从哪里来的呢？图 11-12 是 51testing 调查的 2009 年测试从业者的专业学习情况。

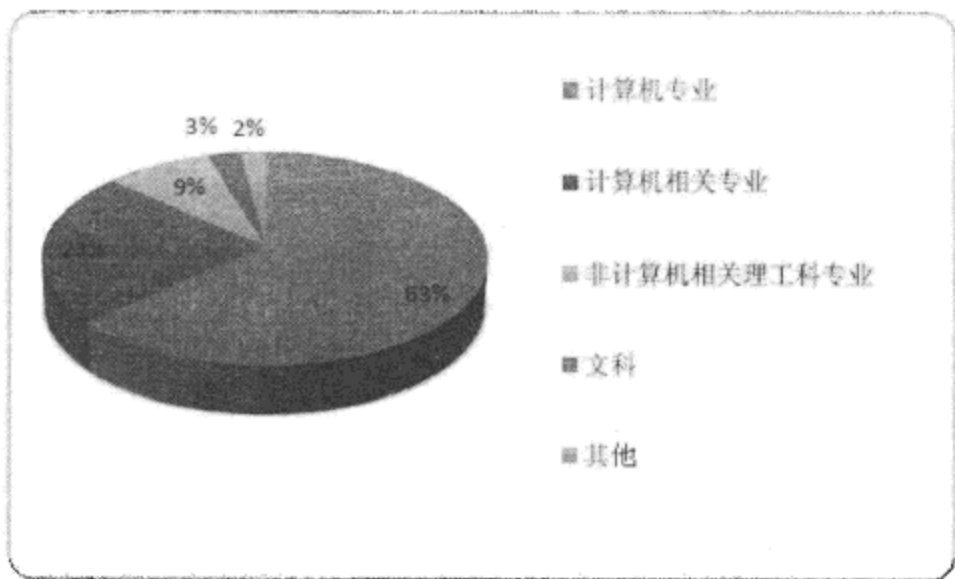


图 11-12 2009 年测试从业者所学专业情况

一个人在学校所学的专业对口与不对口，对工作多少是会有影响，但并不是决定性的，工作中的持续学习才是最重要的。信息技术发展日新月异的今天，不学习就没有进步，俗话说“学到老，活到老”，说的也就是这个道理。同样，一个人在测试岗位上是否能做好，与其过去所学什么专业，或过去是做什么的，并没有本质上的必然联系。下面就是一个出现在笔者身边的案例。

【案例】

朋友 Lin 毕业于国内一所林业学校，就读农林专业，但对计算机相关知识非常感兴趣，不停地自学着相关知识，后来有幸加入了 IT“武林”行列（每每调侃时，常与笔者如是说），也就在那时，笔者认识了他。

刚开始认识 Lin 时，听同事说他是林业学校毕业的，我们都不敢相信，“隔行如隔山”这句古话，怎么在他身上给颠覆了？上班一段时间后发现，在我们测试组的几个新人中，他的确是每做一件事都要慢一拍。比如提交 Bug，一般情况下，在模块还不太稳定的研发阶段，每人每天至少都会提交 3~5 个 Bug，而他一般是提交 1~2 个，有时一天都提不出

一个。但他非常谦虚，也很好学和勤奋。后来听他说，其实他自己心里很清楚自己的短处是什么，为了弥补薄弱的专业基础，特意报读了中山大学自考计算机应用专业的本科，通过三年的坚持努力，终于顺利毕业。更重要的是，边学习边工作的过程，让他受益颇丰收。学而用，用而学，在他身上得到了很好的体现。也就是3年后，他被公司提拔为测试部的主管，成了公司的核心测试骨干。

11.6.2 选择一条适合自己的测试康庄大道

正走在或正打算走上测试大道上的朋友，相信有不少仍很迷茫。测试作为一个专业领域，从工作类型来看，可以再细分，且每一个分支都是一个发展方向，每一个方向都需要人才。自己适合哪一个方向，需结合自身已具备的知识与能力，以及打算学习的领域，还有社会的需求等全方位结合起来分析、定位。下面是常见的一些细分工作发展方向。

1. 黑盒手工测试专家

很大一部分测试朋友，很不愿意向别人说自己是黑盒手工测试人员，甚至说了好像低人一等，实际上只是自己在瞧不起自己，主要是认为其没有技术含量，这是一个典型的认识上有问题的例子。图 11-13 是 51testing 对 2009 年国内测试人员所从事测试工作类型分布的调查结果，这正是目前国内市场需求的一个缩影，社会对测试的需求仍处在普及化的过程中，也表明了测试领域正处在起步阶段。

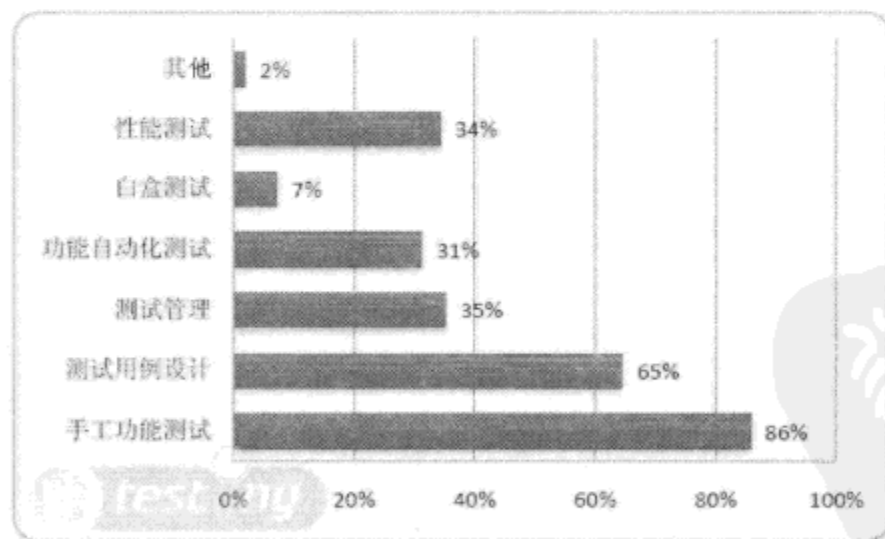


图 11-13 软测从业人员从事的测试工作类型分布

测试是一门技术性很强的工作，黑盒测试是属于其中一种最常见的类型。相对于其他类型的测试，其是入门的第一阶段，是基础。无论你怎么认识手工功能测试，它始终都会存在，因为它有不可替代性。另外一个事实就是手工测试能发现大部分的 Bug，而用其他

方法发现的却是小部分（据测试大师 James Bach 的经验报道，手工测试与自动化测试发现的 Bug 之比是 85%:15%）。依笔者之见，正由于大部分的人从事黑盒测试，黑盒测试领域将会涌现出更多的测试专家，这些专家将会有一部分人转向测试设计、测试管理，也可能转向性能、自动化或白盒等细分领域。

2. 灰盒测试专家

由于灰盒测试行为的一些特殊性，如既分析部分代码，又执行部分功能，它常用来解决一些单纯用黑盒测试方法来执行用例效率低且效果不好的问题。灰盒测试要求测试人员不仅能看懂代码、分析代码，还要非常熟悉业务。

关于灰盒测试的更详细介绍可见本书第 5 章中关于“活用灰盒测试”的部分。

实际工作中，此类工作岗位目前比较少见有独立出来的。有些公司是合并和功能测试人员进行，有些则合并在白盒测试人员进行。需不需要有专门的测试人员来做此项工作，关键是要看测试业务的需求。

3. 白盒测试专家

白盒测试，因为工作目标是要能直接发现他人写的代码中存在的问题，这个问题包罗万象，如变量定义、数组访问越界、内存泄漏等问题，更难的是要发现一些逻辑设计，系统设计存在的隐性问题。要求对测试人员的代码理解分析能力着实要很强。对于复杂的软件系统，动辄几十万，甚至上百万行的代码量，白盒测试的难度可想而知。也正因为白盒测试的难度，对白盒测试人员的要求比一些开发人员还要高是很正常的事。然而对公司而言工作是要考虑成本的。从事过白盒测试的人都清楚，他们的输出与黑盒测试发现的缺陷是不同类的，有些问题黑盒测试很难发现，或根本发现不了，如内存的慢泄漏。但有些反过来，黑盒测试容易发现，但白盒测试难以发现，如界面显示的问题。从问题的数量上来说，即使代码再烂，白盒测试人员提交的问题与黑盒测试人员提交的问题都肯定不是同一个数量级的。

白盒测试要求高，而投入与产出在短期内又不划算。这不仅困扰着测试人员，也让企业雇主们对白盒测试，敬而远之。尽管从长远来看，白盒测试的成本是会得到回报的。但由于市场上的产品更新换代相当快，测试需求也需适应市场的需求，做出相应的快速响应。如果白盒测试中引入工具来做辅助分析，不仅可以加快问题暴露的速度，还可以就某些方面的问题加快代码的覆盖率，达到事半功倍的效果。如 PC-Lint 就是一款不错的静态代码检查工具。

尽管白盒测试面对的是代码，但并不是说不需要熟悉业务，相反，在熟悉业务需求的情况下，更能分析出代码设计的正确性、合理性。

4. 性能测试专家

由于不同行业产品的应用领域不同，对性能测试指标的定义会有所不同。要成为性能测试专家，要从性能测试工程师做起，而在做性能测试工程师之前，同样需要有 2~3 年的功能测试功底。

性能测试是否需要设立一个岗位招聘一些专门的测试人员来做，同样要看公司产品的测试需求。如多用户访问的网络软件，一些对性能要求较高的实时控制软件，需特别关注其性能。性能方面一旦出问题，小则招致用户的投诉，大则可能会导致用户的重要数据丢失，有的还可能涉及经济纠纷，如电子商务交易平台在交易过程中出现性能问题，后果就很严重。性能测试人员需具备良好的代码分析能力，Bug 的定位能力，能协助开发人员一起解决问题。另外，在性能调优方面，如果没有定位问题的能力，会觉得寸步难行。

同样，掌握一些性能测试的工具，是相当必要的，如目前业界流行的性能测试工具 Loader Runner，开源系列的性能测试工具 OpenSTA 等。

5. 自动化测试专家

说起自动化测试，业界也存在一种类似最前面提到的某些测试人员对黑盒测试的认识误区。但与轻视黑盒测试恰好相反，有不少测试人员认为只有自动化测试才有发展前途，出于这样的认识，无论实际上有无做过自动化测试，他们在应聘的简历上都写“熟悉自动化测试”，更有甚者，只是从网上下载一些自动化测试软件，录制、回放了几回，就敢说自己是精通自动化测试，此种对自动化测试的过度推崇同样是一个认识的误区。下面是纠正此误区的一些原则性问题和答案。

- 哪些测试问题需自动化？优先自动化那些需反反复复进行同样操作的手工测试用例，然后是多次执行一致的回归测试。
- 自动化测试不能做什么？自动化并非万能，它是有限制的，它并不能代替手工测试，不能代替人的随机创新思维。
- 自动化测试不仅限于功能。目前比较常见的是功能自动化测试专家，但自动化测试的领域不仅仅包含功能，还可以包括压力测试、性能测试，甚至是每日构建中的冒烟测试，都可以自动化起来。

想向自动化测试专家方向发展，以下是一些基本的要求：

- 精通一门常用编程语言，如 C/C++/C#/Java，并有 3 年以上编程经验。
- 精通某类测试脚本开发语言，如 VBScript、JavaScript、Perl、Python。
- 有开发自动化测试工具及搭建自动化测试平台的实战经验。
- 能组建专业化的软件自动化测试团队。

- 有非常强的技术背景，精通软件测试理论、方法和过程，并能够不断学习、总结和提高。

6. 测试工具开发专家

这一角色与开发人员的角色很类似，都是要进行设计，然后编码实现。但二者的服务对象是不同的，测试工具开发人员服务的是测试人员，而开发人员服务的是公司的产品。

测试工具包括支持测试管理、测试流程改进或测试方法改进的工具，如缺陷管理工具 Bugzilla 的二次开发、用例管理工具 Testlink 的二次开发等。主要的目标是通过工具的使用，提升测试团队的工作效率。

对测试工具开发者有如下一些基本要求：

- 精通一门开发语言，有丰富的开发经验。
- 熟悉所从事测试行业的业务知识。
- 有软件系统设计与架构的经验。
- 熟悉软件测试流程、方法。
- 熟练掌握常用的脚本语言。
- 学习新知识能力强。

7. 测试管理者

最后才谈成为测试管理者，主要是因为这条线上的发展所需人才有限，特别是越往上走，需求将越小。而在国内，不仅仅是软测领域，各行各业都有仕途情结，好像没有“升官”，就是落后，就意味着不成功。可见，拥有成为测试管理者梦想的人有很多。事实上，一个管理者与一个独立贡献者的价值是没有可比性的。这就好比很难判断一个成功的企业家，与一个科学家或一个作家，谁更成功，谁的价值更大。

归根到底，还是人的价值观的问题。一个企业能给社会解决很多就业问题，给政府带来源源不断的税收，给社会的和谐、健康发展带来贡献。而一个科学家呢，他的技术发明足可以改变整个世界，给世界带来天翻地覆的变化，世世代代造福人类。如电灯的发明者爱迪生，他的发明具有划时代的意义，从此改变了人们的生活，使电灯走入了寻常百姓家。被誉为“杂交水稻之父”的袁隆平，他的发明圆了世界粮食安全之梦，等等。

由于测试是技术性的工作，要想成为测试管理者，笔者建议还是从基层做起，有了一定的测试技术积累后，再转向做测试管理，工作起来会更加轻松。测试管理路线发展的详细介绍见本书第 4 章中“测试管理发展线路”的部分。

最后，希望仍在迷茫中的朋友，能走出测试发展的误区，找到适合自己的发展方向，开始向测试专家方向迈进。

附录 A

专业名词解释

在软件测试领域，关于测试方法、测试类型的专业名词繁多，它们之间存在一定关系，但又有各自的特色，很容易搞混。下面从不同的角度出发，把常见到的测试方法或类型进行分类，同时进行简要的解释。

A.1 根据对代码的可视程度进行划分

从了解软件系统内部程序结构的程度不同，可分为白盒测试、灰盒测试、黑盒测试。

1. 白盒测试

白盒测试（White Box Testing）也称为结构测试或逻辑驱动测试，它是按照程序内部的结构测试程序，通过测试来检测产品内部动作是否按照设计规格说明书的规定正常进行，检验程序中的每条通路是否都能按预定要求正确工作。这一方法是把测试对象看做一个打开的盒子，测试人员依据程序内部逻辑结构相关信息，设计或选择测试用例，对程序所有逻辑路径进行测试，通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。

2. 灰盒测试

灰盒测试（Gray Box Testing），灰盒测试是介于白盒测试和黑盒测试之间的一种测试方法，或者说是两者的结合，也有人称集成测试为灰盒测试。它关注输出对于输入的正确性，同时也关注内部表现，但这种关注不像白盒那样详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态，或者说参考部分代码设计用例去做黑盒测试。

灰盒测试结合了白盒测试与黑盒测试的要素，它考虑了用户端、特定的系统知识和操作环境，它在系统组件的协同性环境中评价应用软件的设计。

3. 黑盒测试

黑盒测试 (Black Box Testing) 也称为功能测试或数据驱动测试，它是针对已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用。在测试时，把程序看做一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，对程序进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。

A.2 根据程序的运行状态进行划分

从程序的运行状态来看，可划分为静态测试和动态测试。

1. 静态测试

静态方法是指不运行被测程序本身，仅通过分析或检查源程序的语法、结构、过程、接口等来检查程序的正确性，需要对需求规格说明书、软件设计说明书、源程序流程图进行分析来找错。静态方法通过程序静态特性的分析，找出欠缺和可疑之处，例如不匹配的参数、不适当的循环嵌套和分支嵌套、不允许的递归、未使用过的变量、空指针的引用和可疑的计算等。静态测试结果可用于进一步的查错，并为测试用例选取提供指导。静态测试包括代码检查、静态结构分析、代码质量度量等。

2. 动态测试

所谓软件的动态测试，就是通过运行软件来检验软件的动态行为和运行结果的正确性。它由连续的 3 个步骤组成：设计测试用例、执行程序、分析程序的输出结果。目前，动态测试也是测试工作的主要方式。

A.3 根据所处的开发阶段和作用不同进行划分

根据动态测试在软件开发过程中所处的阶段和作用的不同，动态测试可由如下测试环节组成，在逻辑上前一个阶段是后一个阶段的基础。

1. 单元测试

单元测试 (Unit Testing) 是对软件中的基本组成单位进行测试，其目的是检验软件基本组成单位的正确性。这里的基本单元不一定是指一个具体的函数或一个类的方法。“单元”具有一些基本属性，如明确的功能、规格定义，与其他部分明确的接口定义等，可清晰地与同一程序的其他单元划分开来。

2. 集成测试

集成测试 (Integration Testing) 是在单元测试的基础上，将所有模块按照概要设计要求（如根据结构图）组装成为子系统或系统进行测试，也叫组装测试，或叫子系统测试或部件测试。其主要目的是检查软件模块之间的接口问题，在实际工作中，我们把集成测试分为若干次的组装测试和确认测试。

3. 系统测试

系统测试 (System Testing) 是将已集成好的软件系统作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起，在实际使用环境下，对已经集成好的软件系统进行彻底的测试，以验证软件系统的正确性和性能等是否满足其规约所指定的要求。软件系统测试方法很多，主要有功能测试、性能测试、兼容性测试等，见 A.5 中的详细介绍。

4. 验收测试

验收测试 (Acceptance Testing) 是一项确定产品是否能够满足合同或用户所规定需求的测试，让相关的用户或独立测试人员根据测试计划和结果对系统进行测试。它是软件版本正式发布之前的最后一个测试操作，其目的是确保软件准备就绪，让系统用户决定是否接收系统，是管理性和防御性的一种控制方法。常见的方法包括 Alpha 测试、Beta 测试。

- Alpha 测试：是由一个用户在开发环境下进行的测试，也可以是开发机构内部的用户在模拟实际环境下进行的测试，有时也叫室内测试 (In-house Testing)。
- Beta 测试：是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试，与 Alpha 测试不同的是开发者通常不在测试现场。

A.4 根据测试的执行方式进行划分

根据测试的执行方式是由人工进行还是由程序自动执行，测试可分为手工测试与自动

化测试。

1. 手工测试

由测试人员根据测试用例中描述的规程一步步执行测试，将实际结果与期望结果进行比较，最后得出测试的结果。整个过程由人工完成，是一种必须且最为基础的测试方法。

2. 自动化测试

是指软件测试的自动化，是把以人为驱动测试行为转化为机器执行的一种过程。

A.5 系统测试类型

1. 功能测试

功能测试 (Functional Testing) 主要根据产品的需求说明书对产品的各功能进行验证，根据功能测试用例逐项测试，检查产品是否达到用户要求的功能，是系统测试中最基本的测试。

2. 界面测试

界面测试 (User Interface Testing) 是通过运行程序，检查界面的实现是否与界面需求吻合，界面的元素主要包括窗体、对话框、提示框、菜单、按钮、列表框，以及界面的内容、大小、颜色、位置等。

3. 性能测试

性能测试 (Performance Testing) 是通过自动化的测试工具模拟多种正常、峰值及异常负载条件来对系统的各项性能指标进行测试。负载测试和压力测试都属于性能测试，两者可以结合进行。通过负载测试，确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统各项性能指标的变化情况。压力测试是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。

4. 容量测试

容量测试 (Volume Testing) 的目的是使系统承受超额的数据容量来发现它是否能够正确处理。这种测试通常容易与压力测试混淆，压力测试主要是使系统承受速度方面的超额负载，例如一个短时间之内的吞吐量。容量测试是面向数据的，并且它的目的是显示系统可以处理目标内确定的数据容量。

5. 兼容性测试

兼容性测试是针对待测试项目在特定的硬件平台上，不同的应用软件之间，不同的操作系统平台上，在不同的网络等环境中能否正常运行所进行的测试。兼容性测试的目的是确保待测试项目在不同的操作系统平台上能正常运行，包括待测试项目能在同一操作系统平台的不同版本上正常运行；待测试项目能与相关的其他软件或系统“和平共处”；待测试项目能在指定的硬件环境中正常运行；待测试项目能在不同的网络环境中正常运行。

6. 协议一致性测试

协议一致性测试（Protocol Conformance Testing）是检测实现的软件系统与标准协议的符合程度。

7. 安全性测试

安全性测试（Security Testing）用来验证集成在系统内的保护机制是否能够在实际运行中保护系统不受到非法的侵入。

8. 安装测试

安装测试（Installing Testing）用来验证在正常情况和异常情况的不同条件下，软件是否都能正常地在目标环境下进行安装，以确保其在用户端的正常使用。安装测试包括测试安装代码和安装手册，安装手册提供如何进行安装的说明，安装代码提供保证一些程序在安装后能够运行的基础数据。

9. 卸载测试

卸载测试（Uninstalling Testing）与安装测试相反，验证安装后的软件能通过常规的方式正常卸载。常规的方式包括应用软件本身自带的卸载工具，以及在操作系统控制面板中“添加/删除应用程序”中卸载。

10. 容错测试

又称为容错性测试（Fault Tolerance Testing）或健壮性测试（Robustness Testing），用于测试系统在出现故障时，是否能够自动恢复或者忽略故障继续运行。

附录 *B*

参考书目和资源

- [1] (美) 梅尔斯 (G. J. Myers) 等著. 软件测试的艺术[M]. 第2版 王峰, 陈杰 译. 北京: 机械工业出版社, 2006.1。
- [2] Alan Page, Ken Johnston, Bj Rollison. 微软的软件测试之道[M]. 张爽, 高博, 欧琼, 赵勇等译. 北京: 机械工业出版社, 2009.9。
- [3] 朱少民. 全程软件测试[M]. 北京: 电子工业出版社, 2007.9。
- [4] 古乐, 史九林. 软件测试技术概论[M]. 北京: 清华大学出版社, 2004.4。
- [5] 周伟明. 软件测试实践[M]. 北京: 电子工业出版社, 2008.5。
- [6] 蔡为东. 赢在测试—中国软件测试先行者之道[M]. 北京: 电子工业出版社, 2010.1。
- [7] 51Testing 软件测试网, 董杰. 软件测试精要[M]. 北京: 电子工业出版社, 2009.3。
- [8] 赵斌. 软件测试技术经典教程[M]. 北京: 科学出版社, 2007.5。
- [9] 张海藩. 软件工程师[M]. 第3版. 北京: 清华大学出版社, 1998.1。
- [10] (美) Ron Patton. 软件测试[M]. 周予滨, 姚静等译. 北京: 机械工业出版社, 2002.3。
- [11] [美] Mark Fewster & Dorothy Graham. 软件测试自动化技术与实例详解[M]. 舒智勇, 包晓露, 焦跃等译. 北京: 电子工业出版社, 2000.1。
- [12] [美] Steve McConnell. 代码大全[M]. 金戈, 汤凌, 陈硕, 张菲译. 北京: 电子工业出版社, 2007.2。
- [13] [美] 弗雷德里克·布鲁克斯(Frederick P. Brooks Jr.). 人月神话[M]. UML China 翻译组 汪颖译. 北京: 电子工业出版社, 2006.11。
- [14] 微软亚洲研究院. 微软的梦工场[M]. 北京: 清华大学出版社, 2008.10。
- [15] <http://51testing.com.cn>

[16]<http://www.51testing.com/?26026>

[17]<http://baike.baidu.com/>

[18]<http://www.teamst.org/>

[19]<http://www.Bugzilla.org>

[20]<http://sourceforge.net/>

[21]<http://www.svn8.com/>



技术凝聚实力 专业创新出版

博文视点 (www.broadview.com.cn) 资讯有限公司是电子工业出版社、CSDN.NET、《程序员》杂志联合打造的专业出版平台, 博文视点致力于——IT专业图书出版, 为IT专业人士提供真正专业、经典的好书。

请访问 www.dearbook.com.cn (第二书店) 购买优惠价格的博文视点经典图书。

请访问 www.broadview.com.cn (博文视点的服务平台) 了解更多更全面的出版信息; 您的投稿信息在这里将会得到迅速的反馈。

博文本版精品汇聚



加密与解密 (第三版)

段钢 编著

ISBN 978-7-121-06644-3

定价: 69.00元

畅销书升级版, 出版一月销售10000册。
看雪软件安全学院众多高手, 合力历时4年精心打造。



疯狂Java讲义

新东方IT培训广州中心

软件教学总监 李刚 编著

ISBN 978-7-121-06646-7

定价: 99.00元 (含光盘1张)

用案例驱动, 将知识点融入实际项目的开发。
代码注释非常详细, 几乎每两行代码就有一行注释。



Windows驱动开发技术详解

张帆 等编著

ISBN 978-7-121-06846-1

定价: 65.00元 (含光盘1张)

原创经典, 威盛一线工程师倾力打造。
深入驱动核心, 剖析操作系统底层运行机制。



Struts 2权威指南

李刚 编著

ISBN 978-7-121-04853-1

定价: 79.00元 (含光盘1张)

可以作为Struts 2框架的权威手册。
通过实例演示Struts 2框架的用法。



你必须知道的.NET

王涛 著

ISBN 978-7-121-05891-2

定价: 69.80元

来自于微软MVP的最新技术心得和感悟。
将技术问题以生动易懂的语言展开, 层层深入, 以例说理。



Oracle数据库精讲与疑难解析

赵振平 编著

ISBN 978-7-121-06189-9

定价: 128.00元

754个故障重现, 件件源自工作的经验教训。
为专业人士提供的速查手册, 遇到故障不求人。



SOA原理·方法·实践

IBM资深架构师毛新生 主编

ISBN 978-7-121-04264-5

定价: 49.8元

SOA技术巅峰之作!
IBM中国开发中心技术经典呈现!



VC++深入详解

孙鑫 编著

ISBN 7-121-02530-2

定价: 89.00元 (含光盘1张)

IT培训专家孙鑫经典畅销力作!

博文视点资讯有限公司

电话: (010) 51260888 传真: (010) 51260888-802

E-mail: market@broadview.com.cn (市场)

editor@broadview.com.cn jsj@phei.com.cn (投稿)

通信地址: 北京市万寿路173信箱 北京博文视点资讯有限公司

邮编: 100036

电子工业出版社发行部

发行部: (010) 88254055

门市部: (010) 68279077 68211478

传真: (010) 88254050 88254060

通信地址: 北京市万寿路173信箱

邮编: 100036

博文视点 · IT出版旗舰品牌

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：(010) 88254396；(010) 88258888

传 真：(010) 88254397

E - mail: dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

电子工业出版社总编办公室

邮 编：100036



[G e n e r a l I n f o r m a t i o n]

书名 = 软测之魂：核心测试设计精解

作者 = 肖利琼著

页数 = 3 0 0

出版社 = 北京市：电子工业出版社

出版日期 = 2 0 1 1 . 0 2

SS号 = 1 2 7 5 9 2 2 4

DX号 = 0 0 0 0 0 8 0 2 5 2 6 1

URL = <http://book.szdnnet.org.cn/bookDetail.jsp?dxNumber=000008025261&d=0D1BF737C023BCE9D8DE4CD58422428A>