

高等学校教材·软件工程

可下载教学资料

<http://www.tup.tsinghua.edu.cn>

# 软件测试方法 和技术

朱少民 主 编



清华大学出版社

## 本书特色

主要包括以下内容:

- ◆ 系统的软件品质保证体系和软件测试原理,从软件工程学、质量保证、风险管理等角度阐述了软件测试以及软件测试策略的设计原则
- ◆ 软件测试的流程和技术/方法,深入剖析和探讨了各种类型测试类型和不同阶段比较成熟的技术以及方法,包括从单元测试到验收、安装测试、从应用服务器测试到本地化、自动化测试等
- ◆ 软件测试的实际应用参考。从怎样组建测试队伍,搭建测试环境到测试用例组织、设计,化繁为简,将抽象理论知识变为可触摸到的实际操作,更好地理解 and 消化前面的理论基础

> > > > > > > >

## 作者简介

**朱少民** 曾任合肥工业大学副研究员、硕士生指导老师,从事软件开发、管理十四年,先后获得青岛市、合肥市、安徽省、机械工业部等科技进步奖,最近五年一直担任WebEx中国QA经理、QA总监(WebEx是在美国NASDAQ上市的通讯软件公司)。全国软件测试和质量保证高级培训班首席讲师,中国科技大学软件学院客座副教授,负责《软件工程》、《软件测试和质量保证》等课程的教学和实习指导。

ISBN 7-302-11133-2



9 787302 111337 >

定价: 36.00 元

高等学校教材·软件工程

# 软件测试方法和技术

朱少民 主编

清华大学出版社

北 京

## 内 容 简 介

本书系统介绍了软件品质保证体系和软件测试原理。从软件工程学、质量保证、风险管理等角度阐述了软件测试以及软件测试策略的设计原则。

本书重点讲解软件测试的流程和技术/方法,深入剖析和探讨了各种测试类型和不同阶段比较成熟的技术以及方法,包括从单元测试到验收、安装测试,从应用服务器测试到本地化、自动化测试等。

本书作为软件测试的实际应用参考。从怎样组建测试队伍,搭建测试环境到测试用例组织、设计、化繁为简,将抽象理论知识变为可触摸到的实际操作,更好地理解 and 消化理论基础。

本书适用于高校计算机及软件工程专业作为教材使用,也可作为软件测试人员的技术参考书。

版权所有·翻印必究。举报电话:010-62782989 13501256678 13801310933

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

本书防伪标签采用特殊防伪技术,用户可通过在图案表面涂抹清水,图案消失,水干后图案复现;或将表面膜揭下,放在白纸上用彩笔涂抹,图案在白纸上再现的方法识别真伪。

### 图书在版编目(CIP)数据

软件测试方法和技术/朱少民主编. —北京:清华大学出版社,2005.7

(高等学校教材·软件工程)

ISBN 7-302-11133-2

I. 软… II. 朱… III. 软件—测试 IV. TP311.5

中国版本图书馆CIP数据核字(2005)第054798号

出 版 者:清华大学出版社

<http://www.tup.com.cn>

社 总 机:010-62770175

地 址:北京清华大学学研大厦

邮 编:100084

客户服务:010-62776969

组稿编辑:丁 岭

文稿编辑:许振伍 丁 岭

印 刷 者:北京市世界知识印刷厂

装 订 者:北京鑫海金澳胶印有限公司

发 行 者:新华书店总店北京发行所

开 本:185×260 印张:28.25 字数:680千字

版 次:2005年7月第1版 2005年7月第1次印刷

书 号:ISBN 7-302-11133-2/TP·7357

印 数:1~3000

定 价:36.00元



# 出版说明

改革开放以来，特别是党的十五大以来，我国教育事业取得了举世瞩目的辉煌成就，高等教育实现了历史性的跨越，已由精英教育阶段进入国际公认的大众化教育阶段。在质量不断提高的基础上，高等教育规模取得如此快速的发展，创造了世界教育发展史上的奇迹。当前，教育工作既面临着千载难逢的良好机遇，同时也面临着前所未有的严峻挑战。社会不断增长的高等教育需求同教育供给特别是优质教育供给不足的矛盾，是现阶段教育发展面临的基本矛盾。

教育部一直十分重视高等教育质量工作。2001年8月，教育部下发了《关于加强高等学校本科教学工作，提高教学质量的若干意见》，提出了十二条加强本科教学工作提高教学质量的措施和意见。2003年6月和2004年2月，教育部分别下发了《关于启动高等学校教学质量与教学改革工程精品课程建设工作的通知》和《教育部实施精品课程建设提高高校教学质量和人才培养质量》文件，指出“高等学校教学质量和教学改革工程”，是教育部正在制订的《2003—2007年教育振兴行动计划》的重要组成部分，精品课程建设是“质量工程”的重要内容之一，教育部计划用五年时间（2003—2007年）建设1500门国家级精品课程，利用现代化的教育信息技术手段将精品课程的相关内容上网并免费开放，以实现优质教学资源共享，提高高等学校教学质量和人才培养质量。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作，提高教学质量的若干意见》精神，紧密配合教育部已经启动的“高等学校教学质量与教学改革工程精品课程建设工作”，在有关专家、教授的倡议和有关部门的大力支持下，我们组织并成立了“清华大学出版社教材编审委员会”（以下简称“编委会”），旨在配合教育部制定精品课程教材的出版规划，讨论并实施精品课程教材的编写与出版工作。“编委会”成员皆来自全国各类高等学校教学与科研第一线的骨干教师，其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求，“编委会”一致认为，精品课程的建设工作从开始就要坚持高标准、严要求，处于一个比较高的起点上：精品课程教材应该能够反映各高校教学改革与课程建设的需要，要有特色风格、有创新性（新体系、新内容、新手段、新思路，教材的内容体系有较高的科学创新、技术创新和理念创新的含量）、先进性（对原有的学科体系有实质性的改革和发展、顺应并符合新世纪教学发展的规律、代表并引领课程发展的趋势和方向）、示范性（教材所体现的课程体系具有较广泛的辐射性和示范性）和一定的前瞻性。教材由个人申报或各校推荐（通过所在高校的“编委会”成员推荐），经“编委会”认真评审，最后由清华大学出版社审定出版。

目前，针对计算机类和电子信息类相关专业成立了两个“编委会”，即“清华大学出版社计算机教材编审委员会”和“清华大学出版社电子信息教材编审委员会”。首批推出的特色精品教材包括：

(1) 高等学校教材·计算机应用——高等学校各类专业，特别是非计算机专业的计算

机应用类教材。

- (2) 高等学校教材·计算机科学与技术——高等学校计算机相关专业的教材。
- (3) 高等学校教材·电子信息——高等学校电子信息相关专业的教材。
- (4) 高等学校教材·软件工程——高等学校软件工程相关专业的教材。
- (5) 高等学校教材·信息管理与信息系统。

清华大学出版社经过近二十年的努力，在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌，为我国的高等教育事业做出了重要贡献。清华版教材经过二十多年的精雕细刻，形成了技术准确、内容严谨的独特风格，这种风格将延续并反映在特色精品教材的建设中。

**清华大学出版社教材编审委员会**  
**E-mail: [dingl@tup.tsinghua.edu.cn](mailto:dingl@tup.tsinghua.edu.cn)**

# 前 言

2002 年, 国家信息产业部在软件产业发展公报中列举了我国软件业发展的三大问题, 其中一个问题就是国内软件企业出口能力很弱。公报是这样描述的: “随着国内软件企业的发展壮大, 国内软件企业也在开始不断开拓海外市场。但由于缺乏有自主知识产权的拳头产品, 同时又缺乏较强的项目分析和设计经验, 对国际市场信息、先进软件的设计、开发方式缺乏了解, 大多没有完善的质量保障体系, 对软件开发过程缺乏有效的管理体系, 缺乏严格的质量认证和规范化管理, 不能与国际标准接轨, 这些都构成了软件出口的重要障碍”。由此可见, 完善的质量保障体系、严格的质量认证是提高软件企业生产能力和竞争能力的重要因素。

软件测试是软件质量保证的关键步骤。软件测试研究的结果表明: 软件中存在的问题发现越早, 其软件开发费用就越低; 在编码后修改软件缺陷的成本是编码前的 10 倍, 在产品交付后修改软件缺陷的成本是交付前的 10 倍; 软件质量越高, 软件发布后的维护费用越低。另据对国际著名 IT 企业的统计, 它们的软件测试费用占整个软件工程所有研发费用的 50% 以上。

相比之下, 中国软件企业在软件测试方面与国际水准相比仍存在较大差距。首先, 在认识上重开发、轻测试, 忽略了如何通过流程改进和软件测试来保证产品或系统的质量, 也没有认识到软件项目的如期完成不仅取决于系统设计水平和代码实现能力, 而且取决于设计、代码、文档等各方面的质量。其次, 在管理上表现为随意、简单, 没有建立规范、有效的软件测试管理体系。另外, 缺少自动化工具的支持, 大多数企业在软件测试时并没有采用软件测试管理系统。所以对软件企业来说, 不仅要提高对软件测试的认识, 同时要建立起独立的软件测试组织, 采用先进的测试技术, 充分运用测试工具, 不断改善软件开发流程, 建立完善的软件质量保证的管理体系。只有这样, 才有可能达到软件开发的预期目标, 降低软件开发的成本和风险, 提高软件开发的效率 and 生产力, 确保及时地发布高质量的软件产品。

为了缩小国内软件测试水平和国际水平的差距, 我们将多年来所积累的软件测试经验与技术实践, 依理论、方法和实践 3 部分整理成书, 与大家共享。同时, 也将作者在大学软件学院的软件测试专业课、在全国性软件测试和质量保证高级培训班以及其他培训班等的授课经验与体会, 融入本书之中。

全书共 3 部分, 分 17 章, 涵盖了软件测试技术和方法所涉及的各方面内容, 包括软件测试团队的建立、测试环境的设置和维护、软件测试的组织和管理等, 既有理论方法, 又有实践经验。

第 1 部分 软件测试的原理。共分 4 章来阐述软件测试的重要性、基本概念和方法等。

第 1 章介绍软件开发过程和软件开发过程中所采用的过程模型, 结合过程模型来阐述软件测试的地位, 并力图从一些经典的软件质量事故中给读者一些启发。

第2章一开头就介绍“软件质量”这个重要概念,然后以此为出发点引出软件测试的基本概念和方法、软件缺陷(bug)的含义,以及软件测试的分类、阶段和过程。

第3章主要介绍软件测试策略和测试计划的内涵、制定方法,并讨论了质量保证与测试的区别,以及如何进行质量可靠性、测试风险性的评估。

第4章从软件质量标准,逐步深入到软件测试的依据和规范,介绍了什么是规范的软件测试和质量管理的评判体系,简单地讨论了CMM和ISO9001思想和结构体系。

第2部分 软件测试的技术。共分7章来介绍软件测试在各个阶段(单元测试、集成测试、系统测试、验收测试和安装测试)的技术和方法,并通过对典型应用软件领域测试特点的讨论,帮助读者深入理解本章的核心内容——软件测试的技术。

第5章主要介绍单元测试的概念和各种方法,包括等价划分、边界条件确定、程序路径与逻辑验证、程序状态变化等测试方法,简单讨论了编码标准和规范、代码的审查等。

第6章介绍集成测试和系统测试,重点在系统测试上,包括压力测试、容量测试、性能测试、安全性测试、可靠性和容错性测试等方法及比较。

第7章内容集中在验收测试阶段,其中包括安装测试,涉及到产品说明书的验证,可用性、兼容性、可安装性、可恢复性和文档等各个方面的测试。

第8章介绍目前比较流行的面向对象软件这一领域的各种特定的测试方法,包括数据流测试、面向对象的单元和集成测试以及基于UML的系统测试等。

第9章介绍面向应用服务器的测试,具有内容新、技术深的特点,包括Web服务器、数据库应用服务器、J2EE平台等应用系统的测试技术。

第10章介绍软件国际化和本地化的测试方法和注意事项、国际化和本地化的应用。

第11章介绍软件测试自动化的概念、流行测试工具的分类和应用,最后给出了基于IBM-Rational、MI、Compuware这3家著名公司产品的整体解决方案。

第3部分 软件测试的实践。共分6章来介绍软件测试团队和环境的建立,以及如何设计测试用例,如何报告软件缺陷,如何写测试报告等,最后介绍软件测试项目管理的方法和经验。

第12章介绍软件测试团队的任务、构成、规模和组织模型,并详细介绍测试团队的招聘、面试、激励、发展等实践经验。

第13章介绍一个标准的、规范的测试环境是如何建立和配置起来的,以及如何做好维护,满足测试对环境的严格要求。

第14章介绍软件测试用例的设计方法和经验,不仅包括白盒测试和黑盒测试的用例设计方法,还包括用户使用情景的测试用例设计方法,以及用例的组织、维护和改善。

第15章介绍一般测试人员所需要掌握的、最基本的实践能力——如何描述、处理和跟踪所发现的软件缺陷。

第16章介绍如何写测试和软件质量分析报告,特别提供了评估系统测试的覆盖程度,产品质量的量化分析等方法,以及测试报告的模板和实例。

第17章介绍了国际化的、先进的软件测试项目的组织与管理方法和经验,包括测试资源分配和进度控制、软件版本和分支的控制等。

每章后面都附有小结和思考题。

本书最后附有测试常用的中英文术语对照、常用的各种测试文档模板、参考文献和测试信息资源。

全书由朱少民主编、审稿和定稿。第1、第2、第10、第11、第12、第16、第17章由朱少民编写，第3、第4章由张家银编写，第5、第7、第13章由吴培宏编写，第6、第8章由王顺编写，第9、第14章由高飞编写，第15章由朱晓婧编写。张静参与了第10章的部分编写工作，汪学娟参与了附录的编写工作。感谢张勤、钟声、胡晓明、范雅琛、张建华等同志对本书提出的修改意见以及其他工作，同时要感谢作者的家人、作者所在的网迅（WebEx）公司的大力支持，感谢清华大学出版社所提供的合作机会，使这本书可以早日和读者见面。

由于水平和时间的限制，书中不可避免会出现一些错误，请各界同仁不吝赐教。

作者



6.2.2 功能测试的方法.....	112
6.3 系统测试.....	115
6.3.1 系统测试的内容.....	116
6.3.2 回归测试.....	117
6.4 压力测试、容量测试和性能测试.....	119
6.4.1 压力测试.....	119
6.4.2 容量测试.....	120
6.4.3 性能测试.....	121
6.5 安全性、可靠性和容错性测试.....	122
6.5.1 安全性测试.....	123
6.5.2 可靠性测试.....	124
6.5.3 容错性测试.....	125
小结.....	127
思考题.....	128
<b>第7章 验收测试.....</b>	<b>129</b>
7.1 验收测试的过程和主要内容.....	129
7.2 产品规格说明书的验证.....	130
7.2.1 产品规格说明书的审核.....	130
7.2.2 产品说明书的验证.....	131
7.3 用户界面和可用性测试.....	131
7.4 兼容性测试.....	134
7.5 可安装性和可恢复性测试.....	135
7.6 文档测试.....	137
7.6.1 文档的种类.....	138
7.6.2 文档测试的重要性.....	139
7.6.3 怎样进行文档测试.....	139
7.7 验收测试报告 and 用户验收测试.....	140
小结.....	140
思考题.....	141
<b>第8章 面向对象软件的测试.....</b>	<b>142</b>
8.1 面向对象软件的特点.....	142
8.2 面向对象测试的层次与数据流.....	146
8.2.1 类与子类的测试.....	146
8.2.2 分层与增量.....	147
8.2.3 面向对象层次结构测试重点.....	147
8.3 面向对象的单元测试.....	149

思考题 .....	275
<b>第 14 章 软件测试用例的设计 .....</b>	<b>276</b>
14.1 测试用例设计概述 .....	276
14.1.1 测试用例的重要性 .....	276
14.1.2 测试用例设计书写标准 .....	277
14.1.3 测试用例设计考虑因素 .....	279
14.1.4 测试用例设计的基本原则 .....	282
14.2 白盒测试用例设计方法 .....	283
14.2.1 逻辑覆盖法 .....	283
14.2.2 基本路径测试法 .....	288
14.3 黑盒测试用例设计方法 .....	289
14.3.1 等价类划分法 .....	289
14.3.2 边界值分析法 .....	291
14.3.3 因果图法 .....	293
14.3.4 错误推测法 .....	293
14.3.5 功能图法 .....	294
14.4 测试用例的组织 and 跟踪 .....	296
14.4.1 组织测试用例 .....	296
14.4.2 跟踪测试用例 .....	299
14.4.3 维护测试用例 .....	301
14.4.4 测试用例的覆盖率 .....	303
小结 .....	303
思考题 .....	303
<b>第 15 章 报告所发现的软件缺陷 .....</b>	<b>304</b>
15.1 软件缺陷的描述 .....	304
15.1.1 软件缺陷的基本描述 .....	304
15.1.2 软件缺陷属性 .....	305
15.2 软件缺陷相关的信息 .....	308
15.2.1 软件缺陷的图片、记录信息 .....	308
15.2.2 分离和再现软件缺陷 .....	309
15.3 软件缺陷的处理和跟踪 .....	311
15.3.1 软件缺陷生命周期 .....	312
15.3.2 软件缺陷处理技巧 .....	313
15.3.3 软件缺陷跟踪系统 .....	313
15.3.4 缺陷跟踪的方法和图表 .....	317
小结 .....	319

思考题 .....	319
<b>第 16 章 软件测试和质量分析报告 .....</b>	<b>321</b>
16.1 软件产品的质量度量 .....	321
16.1.1 软件度量的内容和分类 .....	322
16.1.2 软件度量的分工和过程 .....	324
16.1.3 软件质量模型 .....	325
16.1.4 软件质量的度量 .....	327
16.1.5 质量度量的统计方法 .....	327
16.2 评估系统测试的覆盖程度 .....	329
16.2.1 对软件需求的估算 .....	330
16.2.2 基于需求的测试覆盖评估 .....	330
16.2.3 基于代码的测试覆盖评估 .....	331
16.3 软件缺陷分析方法 .....	331
16.3.1 缺陷分布报告 .....	332
16.3.2 缺陷趋势报告 .....	333
16.4 基于缺陷分析的产品质量评估 .....	334
16.4.1 经典的种子公式 .....	335
16.4.2 基于缺陷清除率的估算方法 .....	336
16.4.3 软件产品性能评估 .....	337
16.4.4 借助工具的方法 .....	337
16.5 测试报告的模板、实例 .....	337
小结 .....	340
思考题 .....	341
<b>第 17 章 软件测试项目管理 .....</b>	<b>342</b>
17.1 软件测试项目管理的概述 .....	342
17.1.1 软件项目管理的共性 .....	343
17.1.2 软件测试项目管理的特点 .....	345
17.2 软件测试项目的组织 .....	346
17.3 软件测试项目的过程管理 .....	349
17.3.1 测试计划阶段 .....	350
17.3.2 软件测试设计和开发 .....	353
17.3.3 测试执行阶段 .....	355
17.4 软件测试项目的资源管理 .....	358
17.5 测试项目的进度管理 .....	360
17.5.1 测试项目的里程碑和关键路径 .....	360
17.5.2 测试项目进度的特性及外在关系 .....	362

17.5.3 测试项目进度的管理方法和工具 .....	364
17.6 测试项目的风险管理 .....	368
17.7 测试项目的质量和配置管理 .....	370
17.8 软件测试文档的管理 .....	371
小结 .....	373
思考题 .....	374
附录 A 软件测试的英文术语及中文解释 .....	375
附录 B 质量管理体系——要求（国家标准 GB/T 19001-2000,Idt ISO9001:2000） .....	391
附录 C 信息技术——软件包质量要求和测试（国家标准 GB/T 17544—1998, Idt ISO/IEC 12119:1994） .....	402
附录 D 测试计划模板 .....	413
附录 E C++ Inspection Checklist.....	425
附录 F Java Code Inspection Checklist.....	428
参考文献 .....	431

# **第 1 部分**

## **软件测试的原理**



# 第 1 章 软件及其开发过程

做任何事情，首先要从概念入手，只有概念清楚后，才可能少走弯路或不走弯路，才能对与此概念相关的问题有一个正确的理解和分析，最终解决问题。软件测试的对象就是软件，为了进行软件测试，我们就会问什么是软件？软件有什么特点？是怎样开发出来的？经历哪几个阶段？

作为本书的第 1 章，就先来回答这些问题，通过回答这些问题，可以把软件测试中的“软件”、“软件开发过程”、“过程模型”等主要概念介绍清楚，为本书的后续内容打下良好的基础。

## 1.1 软件的含义

我们每天都说软件测试、软件开发、……，但是否真正全面理解什么是软件，大多数人不敢肯定。那软件真正的含义是什么？

先看看一般教科书所给出的规范、科学的定义，即软件是：

- 能够完成预定功能和性能的、可执行的指令(计算机程序)。
- 使得程序能够适当地操作信息的数据结构。
- 描述程序的操作和使用的文档。

即“软件=程序 + 数据(库) + 文档”，在这里给出了软件的最基本的组成成分。实际上，还少了一项内容：服务。我们可以用一个简单的公式给出软件的定义：

$$\text{软件} = \text{程序} + \text{数据(库)} + \text{文档} + \text{服务}$$

为了帮助读者更好地理解软件的含义，我们一起来看看软件有哪些特征。软件是相对硬件而相对存在的。硬件是可以直观感觉到、触摸得到的物理产品。生产硬件时，人的创造性的过程(设计、制作、测试)能够完全转换成物理的形式。例如，生产一个新的计算机，初始的草图、正式的设计图纸和面板的原型一步步演化成为一个物理的产品，如模具、集成芯片、集成电路、电源和塑料机箱等。

正如我们政府官员经常提到的，我们不仅要搞好投资的硬件环境，更要搞好软件环境。这里的硬件环境，包括交通、水电、办公楼和厂房等，而软件环境指的就是优惠政策、政府职能转变和服务等。

软件则是逻辑的、知识性的产品集合，是对物理世界的一种抽象，或者是某种物理形态的虚拟化。因此，软件具有与硬件完全不同的特征。其主要表现在以下 3 个方面。

### 1. 软件是硬件的灵魂，硬件是软件的基础

计算机硬件必须靠软件实现其功能，如果没有软件，硬件就好比一堆废铁，所以说软件是硬件的灵魂。同时，软件必须依赖于硬件，只有在特定的硬件环境上才能运行。

虽然“软件工厂”的概念也已被引入，这并不是说硬件生产和软件开发是一回事，而是引用软件工厂这个概念促进软件开发中模块化设计、组件复用等意识的全面提升。

### 2. 软件是智慧和知识的结晶

软件是完全的智力产品，是通过技术员的大脑活动创造的结果。软件现在被认为属于高科技产品。软件产业是一种知识密集型产业。

一个价值很高的软件，可能就装在几张软盘上，包括程序和文档。少数不了解软件价值的领导，不愿意为此付出几十万人民币。他可能会说，几十万元钱可以买一大堆计算机，可以买一辆桑塔纳或奥迪小轿车，几张软盘哪会值那么多钱？在这里，这位领导只看到了软件的载体，也就是只看到其物理的表现形式，而没有看到其实质的内容以及开发这个产品过程中所投入的、高技术的大量人力。软件的主要成本在于先期的开发人力。软件成为产品之后，其后期维护、服务成本也很高。而软件载体的制作成本很低，如磁盘、光盘的复制是比较简单的，所以软件也就容易成为盗版的主要目标。

### 3. 软件不会“磨损”，而是逐步完善

随着时间的推移，硬件构件会由于各种原因受到不同程度的磨损，但软件不会。新的硬件故障率很低，随着长时间的改变，硬件会老化，故障率会越来越高。相反，隐藏的错误会引起程序在其生命初期具有较高的故障率，随着使用的不断深入，所发现的问题会慢慢地被改正，其结果是程序越来越完善，故障率会越来越低。

从另一个侧面看，硬件和软件的维护差别很大。当一个硬件构件磨损时，可以用另外一个备用零件替换它，但对于软件，不存在替换，而是通过开发补丁程序不断地解决适用性问题，或扩充其功能。一般来说，软件维护要比硬件维护复杂得多，而且软件的维护周期要长得多。软件正是通过不断的维护，改善功能，增加新功能，来提高软件系统的稳定性和可靠性的。

## 1.2 软件开发过程的特性

在了解软件含义之后，我们就来了解软件是通过一个什么样的过程开发出来的，也就是了解软件的开发过程。软件开发过程是软件工程中的重要内容，也是进行软件测试的基础。

## 1.2.1 软件开发的基本过程

软件开发的基本过程，可以被简单地分为需求分析、设计（概要设计、详细设计）、编程、测试和维护等几个阶段，即通常所说的“传统生命周期”，也就是著名的软件开发过程的“瀑布模型”，如图 1-1 所示。通过这个模型，能比较直观地理解软件开发的全过程。

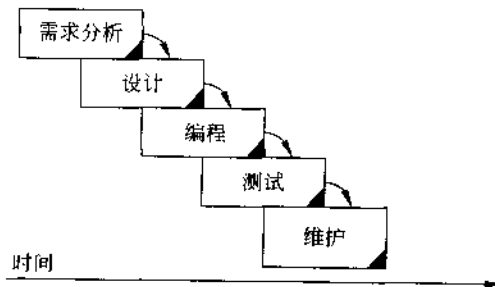


图 1-1 软件开发过程的瀑布模型

### 1. 需求分析

需求分析是根据客户的要求，清楚地了解客户需求中的产品功能、特性、性能、界面和具体规格等，然后进行分析，确定软件产品所能达到的目标。软件产品需求分析是软件开发过程的第一个环节，也是最重要的一个环节。如果需求分析做不好，下面的设计、编程做得再好，客户（用户）也不可能对开发出来的软件产品感到满意。软件产品需求分析的结果要文档化，如 MRD(marketing requirement document)，而且这类文档的描述尽量不要用专业术语，从而使用户能够完全理解需求分析的结果，参与对其复审的过程。

### 2. 设计

软件设计是根据需求分析的结果，考虑如何在逻辑、程序上去实现所定义的产品功能、特性等。可以分为概要设计和详细设计，也可以分为数据结构设计、软件体系结构设计、应用接口设计、模块设计、算法设计、界面设计等。设计过程将需求转换成软件表示，设计的结果将作为编码的框架和依据，以提高编码的效率和质量。设计的文档化体现在产品规格说明书（functional specification）、技术设计文档（development design document）和软件配置文档（software configuration document）。

### 3. 编程

经过需求分析、设计之后，接下来就是用一种或多种具体的程序语言（如 C/C++、Java、PHP/ASP/JSP 等）进行编码，即将设计转换成计算机可读的形式。如果设计做得好、做得仔细，编程就容易了。

### 4. 测试

任何编程，免不了存在这样或那样的错误，所以有必要进行软件测试。测试过程集中

于软件的内部逻辑——保证所有语句都测试到，以及外部功能——即引导测试去发现错误，并保证定义好的输入能够产生与预期结果相同的输出。测试按不同的过程阶段分为单元测试、集成测试、功能测试、系统测试、验证测试等。

## 5. 维护

从理论上，软件测试的覆盖率不可能做到百分之百，所以软件在交付给用户之后有可能存在某些问题，而且用户的需求会发生变化，特别是开始使用产品之后，对计算机系统有了真正的认识 and 了解，会提出适用性更好的、功能增强的要求。所以，软件交付之后不可避免地要进行修改、升级等。正如前面所说，软件维护复杂、周期长，其成本必然很高。通过提高软件的需求分析、设计和编程的质量，强化软件测试，可以大幅度降低软件的维护成本。

## 1.2.2 软件开发过程模型

随着计算机应用的飞速发展，软件的复杂程度不断提高，源代码的规模越来越大，软件开发过程越来越不容易被控制。在长期的研究与实践中，人们越来越深刻地认识到，建立简明、准确的表示模型是把握复杂系统的关键。为了更好地理解软件开发过程的特性，以及跟踪、控制和改进软件产品的开发过程，就必须对这一开发过程模型化。

模型是对事物的一种抽象，人们常常在正式建造实物之前，首先建立一个简化的模型，以便更透彻地了解它的本质，抓住问题的要害。在模型中，先要剔除那些与问题无关的、非本质的东西，从而使模型与真实的实体相比更加简单明了、易于把握。总的来说，使用模型可以使人们从全局上把握系统的全貌及其相关部件之间的关系，可以防止人们过早地陷入各个模块的细节。

经过软件领域的专家和学者不断努力，科学、切合实际的各种软件过程模型不断被推出，它们来源于实践，是用户的需求和软件开发技术共同促进的结果。目前主要有以下模型：

- 瀑布模型
- 原型模型
- 快速应用开发(RAD)模型
- 螺旋模型
- 增量模型和迭代模型
- 构件组装模型
- 并发模型

瀑布模型在 1.2.1 小节简单介绍过，它反映了软件工程的基本思想。当今的软件业，竞争非常激烈，节奏明显加快，瀑布模型的应用越来越少。相反，由于认识到软件开发过程中和用户沟通困难性，那些提高开发效率、速度以及产品后期不断改进等完善的模型越来越受到重视，如原型模型、RAD 模型和螺旋模型等。

## 1. 原型模型

需求分析是在软件开发的最前端,也就说明它对后期的影响最大,所以我们说,软件需求分析很重要,要想成功开发一个软件产品,首先要做好需求分析。但另一方面,在需求分析时,往往很难做到彻底弄清楚用户对产品的各项具体的要求。由于大多数使用或将要使用计算机产品的用户,不是计算机方面的专业人员,甚至对计算机一点都不了解,所以对计算机能做哪些事情、不能做哪些事情、善于做哪些事情、不善于做哪些事情等都不清楚,只能给出软件的一般性功能或目标要求,不能提出具体的要求,也不能给出规范的、科学的、详细的输入和输出需求。有时候,一套软件系统的应用,需要相应的管理体系相适应,而在新的管理体系建立之前,许多东西是不确定的,但软件系统要先行,开始其试验阶段。在这些情况下,原型模型可能是最好的选择。

原型模型的指导思想就是,在进行了基本需求分析之后,快速开发出产品的原型,然后基于这个原型,就比较容易同客户沟通、交流,更好地了解客户需求,不断修改这个原型,到了双方认可的程度,再做详细地分析、设计和编程,最终开发出令客户满意的产品。一般步骤如下:

(1) 先定义软件的总体目标,根据已知的需求来规划出可实现的区域。

(2) 然后是“快速设计”,集中于系统的总体框架、基本功能和直观的输入方式和输出格式等。

(3) 有了原型,使客户对系统实现哪些具体功能、功能实现到什么程度有更好的理解。开发者可以边开发边评估,不断细化软件的需求,逐步调整原型使其满足客户的要求。这形成一个迭代的过程。

即使开始建立的原型过于简单或性能很差,难以使用,但为下一次建立适用的模型积累了经验,而浪费的成本、时间有限。原型模型的优点是使用户能够感受到实际的系统,使开发者能够快速构造出系统的框架。其缺点是产品的先天性不足,因为开发者常常需要做实现上的折中,可能采用不合适的操作系统或程序设计语言,以使原型能够尽快工作。

## 2. RAD 模型

RAD (rap application development) 模型,即快速应用开发模型。由于其模型构图形似字母“V”,故也称V模型,是属于线性顺序一类的软件开发模型。它通过使用基于构件的开发方法来缩短产品开发的周期,提高开发的速度。RAD模型实现的前提是能做好需求分析,并且项目范围明确,这一点正好和原型模型相反。RAD模型包含如下几个开发阶段。

(1) 业务建模:业务活动中的信息流被模型化。通过回答以下问题来实现:什么信息驱动业务流程?生成什么信息?谁生成该信息?该信息流往何处?谁处理它?

(2) 数据建模:业务建模阶段定义的一部分信息流被细化,形成一系列支持该业务所需的数据对象。标识出每个对象的属性,并定义这些对象间的关系。

(3) 处理建模:数据建模阶段定义的数据对象变换成要完成一个业务功能所需的信息流。创建处理描述以便增加、修改、删除或获取某个数据对象。

(4) 应用生成:RAD过程不是采用传统的第三代程序设计语言来创建软件,而是使用



4GL 技术或软件自动化生成辅助工具, 复用已有的程序构件(如果可能的话)或是创建可复用的构件(如果需要的话)。

(5) 测试及反复: 因为 RAD 过程强调复用, 许多程序构件已经是测试过的, 这减少了测试时间。但新构件必须测试, 所有接口也必须测到。

RAD 过程模型如图 1-2 所示。很显然, 加在一个 RAD 项目上的时间约束需要有“一个可伸缩的范围”。如果一个商业应用能够被模块化, 使得其中每一个主要功能均可以在不到 3 个月时间内完成(使用上述的方法), 它就是 RAD 的一个候选件。每一个主要功能可由一个单独的 RAD 组来实现, 最后集成起来形成一个整体。

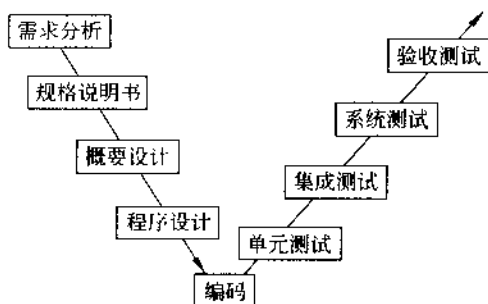


图 1-2 RAD (V) 模型示意图

RAD 模型还有一种改进型, 将“编码”从 V 字型的顶点移到左侧, 和单元测试对应, 从而构成水平的对应关系。下面通过水平和垂直对应关系的比较, 使用户能更清楚、全面地了解软件开发过程的特性。

► 从水平对应关系看

左边是设计和分析, 右边是验证和测试。右边是对左边结果的检验, 即对设计和分析的结果进行测试, 以确认是否满足用户的需求。如:

- 需求分析和功能设计对应验收测试, 说明在做需求分析、产品功能设计的同时, 测试人员就可以阅读、审查需求分析的结果, 从而了解产品的设计特性、用户的真正需求, 可以准备用例 (use case)。
- 当系统设计人员在做系统设计时, 测试人员可以了解系统是如何实现的, 基于什么样的平台, 这样可以事先准备系统的测试环境, 包括硬件和第三方软件的采购。因为这些准备工作, 实际上要很长时间才能完成。
- 在做详细设计时, 测试人员就可以准备测试用例 (test case, 以有效地发现软件缺陷的最小测试执行单元。详见 2.5.2 小节和第 14 章)。
- 一面编程, 一面进行单元测试, 是一种很有效的办法, 使我们可以尽快找出程序中的错误。充分的单元测试可以大幅度提高程序质量、减少成本。

从图 1-3 可以看出, RAD 模型避免了瀑布模型所带来的误区——软件测试是在代码完成之后进行。RAD 模型说明软件测试的工作很早就可以开始了, 项目一启动, 软件测试的工作也就启动了。

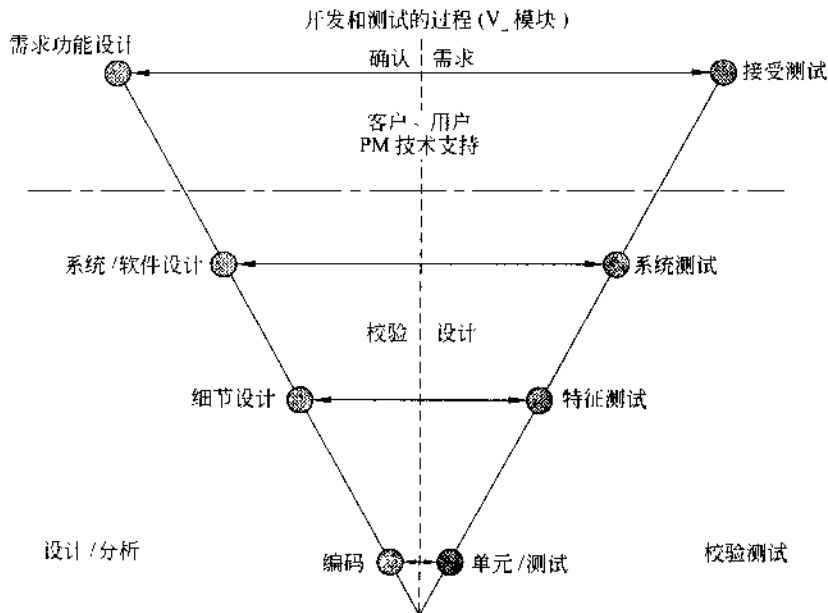


图 1-3 改进的 RAD 模型及其解释

#### ► 从垂直方向看

水平虚线上部表明，其需求分析、功能设计和验收测试等主要工作是面向用户，要 and 用户进行充分的沟通和交流，或者是和用户一起完成。水平虚线下部的大部分工作，相对来说，都是技术工作，在开发组织内部进行，由工程师完成。

所以 RAD 模型一般适合信息系统应用软件的开发，而不适合高性能、技术风险高或不易模块化的系统开发。如果一个系统难以被适当地模块化，那么就很难建立 RAD 所需的构件；如果系统具有高性能的指标，且该指标必须通过调整接口使其适应系统构件才能达到，使用 RAD 方法可能会导致整个项目失败。

### 3. 螺旋模型

螺旋模型，最早是由 Boehm 提出来的，是一个演化软件过程模型，它将原型的迭代特征与线性顺序模型中控制和系统化方面结合起来，使得软件增量版本的快速开发成为可能。在螺旋模型中，软件开发是一系列的增量发布。在早期的迭代中，发布的增量可能是一个纸上的模型或原型；在以后的迭代中，更加完善的被开发系统版本逐步产生。

螺旋模型被划分为若干框架活动，也称任务区域。一般情况下，有 3~6 个任务区域。图 1-4 形象地描述了包含 4 个任务区域的螺旋模型。

- **目标、选择和限制：**系统要达到的目标，同时要受预算、时间等条件的限制，而且必须作出一定的选择和取舍。
- **风险评估：**基于上述目标，评估技术及管理的风险，以决定如何实施项目。
- **开发和测试：**包括系统需求分析、概要设计、详细设计、编程、单元测试、系统测试和验证测试等项目具体实施的各种任务。
- **计划：**定义资源、进度及其他相关项目信息所需要的任务，以调整项目的目标和

改善系统实施的效率。

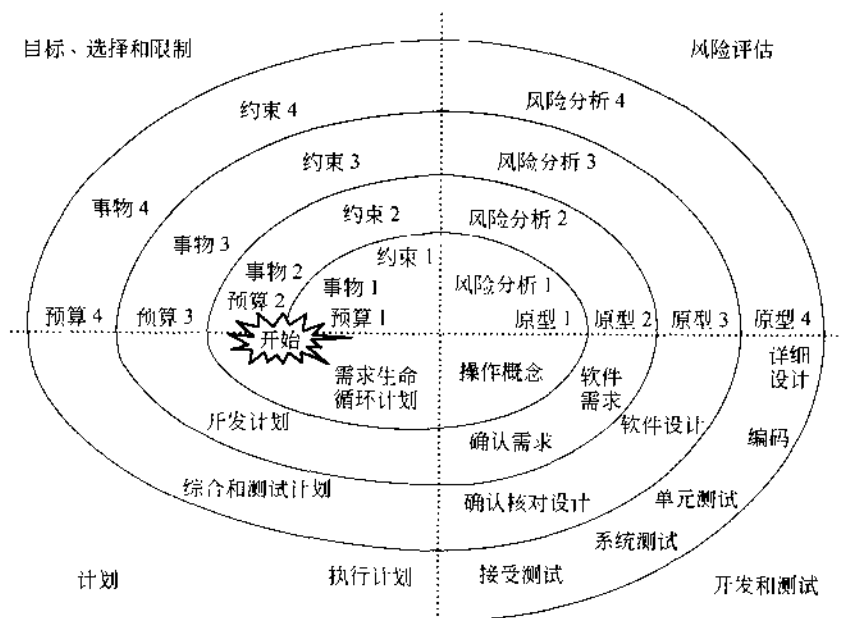


图 1-4 螺旋模型及其解释

随着演化过程的开始，软件工程项目组按顺时针方向沿螺旋移动，从核心开始。螺旋的第1圈可能产生产品的规格说明；接续的螺旋可能用于开发一个原型；随后可能是更加完善的下一个版本软件。经过计划区域的每一圈是基于从用户评估得到的反馈，调整费用和进度。此外，项目管理者可以调整完成软件所需计划的迭代次数。

与传统的过程模型不同，在螺旋模型中，软件交付了并不意味着结束，其适用于计算机软件的整个生命周期。一个“概念开发项目”从螺旋的核心（水平轴）开始一直持续到概念开发结束。如果概念被开发成真正的产品，过程从水平轴一个新的起点开始，意味着一个新的开发项目启动了。

本质上，具有上述特征的螺旋是一直运转的，直到软件退役。有时这个过程处于睡眠状态，但任何时候出现了改变，过程都会从合适的入口点（如产品增强）开始。

对于大型系统及软件的开发来说，螺旋模型是一个很现实的方法。因为软件随着过程的进展演化，开发者和用户能够更好地理解和对待每一个演化级别上的风险。螺旋模型使用原型作为降低风险的机制，更重要的是，它使开发者在产品演化的任一阶段均可应用原型方法。它保持了传统生命周期模型中系统的、阶段性的方法，但将其并进了迭代框架，更加真实地反映了现实世界。螺旋模型要求在项目的所有阶段直接考虑技术风险，如果应用得当，能够在风险变成问题之前降低它的危害。

不过，和其他模型一样，螺旋模型也不是十全十美的。它可能难以使用户（尤其在有合同约束的情况下）相信演化方法是可控的；它需要相当的风险评估的专门技术，且其成功依赖于这种专门技术，如果一个大的风险未被发现和管理，毫无疑问会出现问题；最后，

该模型本身相对比较新，不像线性顺序模型或原型模型那样已经被广泛应用。

#### 4. 增量模型和迭代模型

软件开发不是一蹴而就，其过程犹如雕琢一件工艺品，由无形到有形、由粗到细，很难一次就能开发出功能完善、强大的一个版本，而往往是分阶段进行，一个版本接一个版本的发布。其主要原因是：

- 市场的压力和竞争策略的需要。较早地、不失时机地占领市场对软件企业是很重要的，如果软件企业用太长时间把一个完美的、功能强大的软件产品开发出来，可能市场已经被对手占领了。作为一个优秀的软件公司，会处理好市场策略和产品功能的平衡。
- 产品的开发预算是有限的，产品的开发周期和资源会受到预算的限制。
- 软件的复杂程度不断提高，增加了项目失败的可能性，将一个产品进行分阶段处理，可以尽早发现产品的市场问题或方向错误，降低风险。
- 软件的复杂程度不断提高，也使系统的分析和设计也变得非常困难。对于越来越复杂、庞大的系统，多数情况下，不容易一次性整体实现，而是通过分解逐步实现。

所以软件在实际开发过程中是按阶段进行，逐步完善或深化系统的功能，如图 1-5 所示。

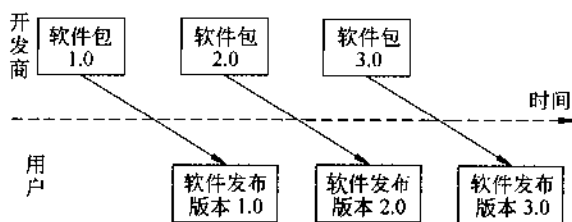


图 1-5 软件分阶段开发示意图

软件开发分阶段可以通过两种模型来描述，即增量模型和迭代模型。

- **增量模型** 描述软件产品的不同阶段是按产品所具有的功能进行划分，先开发主要功能或用户最需要的功能，然后，随着时间推进，不断增加新的辅助功能或次要功能，最终开发出一个强大的、功能完善的、高质量的、稳定的产品，如图 1-6 所示。

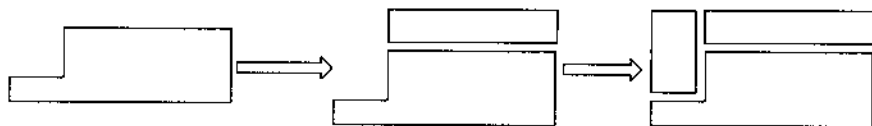


图 1-6 软件分阶段增量模型示意图

- **迭代模型** 描述软件产品的不同阶段是按产品深度或细化的程度来划分。先将产品的整个框架都建立起来，在系统的初期，已经具有用户所需求的全部功能。然后，随着时间推进，不断细化已有的功能或完善已有功能，这个过程好像是一个

迭代的过程,如图 1-7 所示。最终的目标是一致的,也是为了实现一个强大的、功能完善的、高质量的、稳定的产品。



图 1-7 软件分阶段迭代模型示意图

### 1.2.3 UML 代表着软件建模的发展趋势

软件开发技术和模型的表现手法层出不穷,但在目前的软件开发方法中,面向对象的方法占据着主导地位。面向对象方法的主导地位也决定着软件开发过程模型化技术的发展,面向对象的建模技术(OMT)方法也就成为主导的方法。根据对目前软件业的研究和估计,UML(unified modeling language,统一建模语言)可以说代表今后 5~10 年软件建模的发展方向。UML 将成为面向对象技术领域内占主导地位的标准建模语言。UML 融入了软件工程领域的新思想、新方法和新技术,不仅可以支持面向对象的分析与设计,更重要的是能够有力地支持从需求分析开始的软件开发全过程。总的来说,UML 是一种定义良好、易于表示、功能强大且普遍实用的建模语言。

公认的面向对象建模语言出现于 20 世纪 70 年代中期。从 1989 年到 1994 年,其数量从不到十种增加到了五十多种。20 世纪 90 年代中期,一批新方法出现了,其中最引人注目的是 Booch 1993、OOSE 和 OMT-2 等。但是在早期这些众多的建模语言中,存在一些致命的问题,阻止了其进一步的应用,概括起来有两点。

- 面对众多的建模语言,用户由于没有能力区别不同语言之间的差别,因此很难找到一种比较适合其应用特点的语言。
- 众多的建模语言各有千秋,存在一些差别,极大地妨碍了用户之间的交流。

上述原因在客观上促进了 UML 的诞生,UML 克服上述缺点,吸收了早期不同建模语言的优点,在总结面向对象技术应用实践的基础上,根据应用需求,求同存异形成的,统一建模语言。1994 年 10 月,Grady Booch 和 Jim Rumbaugh 首先将 Booch93 和 OMT-2 统一起来,并于 1995 年 10 月发布了第一个公开版本——UM 0.8(Unified Method),称之为统一方法。1995 年秋,OOSE 的创始人 Ivar Jacobson 加入到这一工作中,经过 Booch、Rumbaugh 和 Jacobson 三人的共同努力,于 1996 年 6 月和 10 月分别发布了两个新的版本,即 UML 0.9 和 UML 0.91,并将 UM 重新命名为 UML(Unified Modeling Language)。1996 年,一些机构将 UML 作为其商业策略已日趋明显。UML 的开发者得到了来自公众的正面反应,并倡议成立了 UML 成员协会,以完善、加强和促进 UML 的定义工作。当时的成员有 DEC、HP、I-Logix、Itellicorp、IBM、ICON Computing、MCI Systemhouse、Microsoft、Oracle、Rational Software、TI 以及 Unisys。这一机构对 UML 1.0(1997 年 1 月)及 UML 1.1(1997 年 11 月 17 日)的定义和发布起了重要的促进作用。

面向对象技术和 UML 的发展过程可用图 1-8 来表示。标准建模语言的出现是面向对象技术和 UML 发展的重要成果。在美国,截止到 1996 年 10 月,UML 获得了工业界、科技



界和应用界的广泛支持, 已有 700 多个公司表示支持采用 UML 作为建模语言。1996 年底, UML 已稳占面向对象技术市场的 85%, 成为可视化建模语言事实上的工业标准。1997 年 11 月 17 日, OMG 采纳 UML 1.1 作为基于面向对象技术的标准建模语言。UML 经历了 1.2、1.3、1.4, 目前 UML 2.0 版本已经制定。

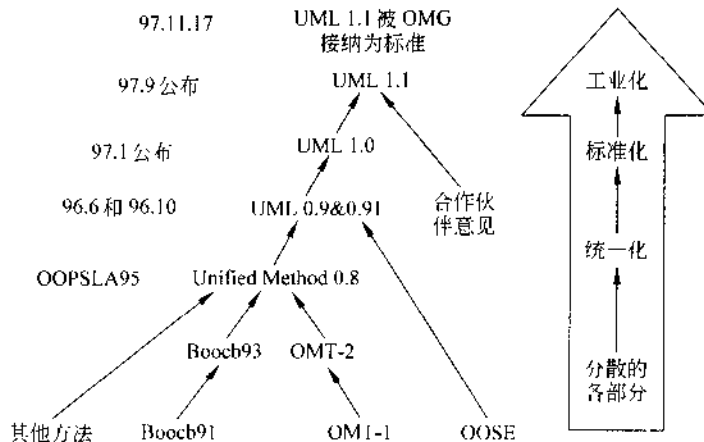


图 1-8 UML 的发展历程

UML 共定义了三大类, 共 12 种模型图。

**结构类模型图 (structural diagrams):** 用 4 种模型图描述系统应用的静态结构, 包括类图、对象图、组件图和配置图。

**行为类模型图 (behavior diagrams):** 用 5 种模型图描述系统动态行为的各个方面, 包括用例图、序列图、行为图、协作图和状态图。

**模型管理类模型图 (model management diagrams):** 用 3 种模型图来组织和管理各种应用模型, 包括软件包、子系统和模型。

## 1. 结构类

- **类图 (class diagram)** 描述系统中类的静态结构, 展示了一组类、接口和协作及它们间的关系。类图描述的是一种静态关系, 在系统的整个生命周期都是有效的。它不仅定义系统中的类, 表示类之间的联系, 如关联、依赖、聚合等, 也包括类的内部结构 (类的属性和操作)。系统可有多个类图, 单个类图仅表达了系统的一个方面。要在高层给出类的主要职责, 在低层给出类的属性和操作。
- **对象图 (object diagram)** 展示了一组对象及它们间的关系, 用对象图说明类图中所反应事物实例的数据结构和静态快照。对象图是类图的实例, 几乎使用与类图完全相同的标识。其不同点在于对象图是类图的一个实例, 对象图显示类的多个对象实例, 而不是实际的类。由于对象存在生命周期, 因此对象图只能在系统某一段时间段存在。
- **组件图 (component diagram)** 描述代码组件的物理结构及各组件之间的依赖关系, 用于对源代码、可执行的发布、物理数据库和可调整的系统建模。

- **部署图 (deployment diagram)** 展现了运行时处理节点以及其中构件的部署。它描述系统硬件的物理拓扑结构 (包括网络布局和构件在网络上的位置), 以及在此结构上执行的软件 (即运行时软构件在节点中的分布情况)。用部署图说明系统结构的静态部署视图, 即说明分布、交付和安装的物理系统。

## 2. 行为类

- **用例图 (use case diagram)** 展现了一组用例、用户以及它们间的关系, 即从用户角度描述系统功能, 并指出各功能的操作者。用于系统需求分析时收集用户实际需求所采用的一些方法中, 在对系统行为组织和建模方面, 用例图是相当重要的。
- **顺序图 (sequence diagram)** 展现了一组对象和由这组对象收发的消息, 用于按时间顺序对控制流建模, 用顺序图说明系统的动态视图。如果强调时间和顺序, 则使用顺序图; 如果强调上下层次关系, 则选择协作图。
- **活动图 (activity diagram)** 活动图是一种特殊的状态图, 描述需要做的活动、执行这些活动的顺序 (多为并行的) 以及工作流 (完成工作所需要的步骤)。它对于系统的功能建模特别重要, 强调对象间的控制流程。
- **协作图 (collaboration diagram)** 展现了一组对象, 这组对象间的连接以及这组对象收发的消息。它强调收发消息对象的结构组织, 按组织结构对控制流建模。
- **状态图 (state chart diagram)** 展示了一个特定对象的所有可能状态以及由于各种事件的发生而引起的状态间的转移。一个状态图描述了一个状态机, 用状态图说明系统的动态视图。它对于接口、类或协作的行为建模尤为重要, 可用它描述使用用例的生命周期。

从应用的角度看, 当采用面向对象技术设计系统时, 首先是描述需求; 其次是根据需求建立系统的静态模型, 以构造系统的结构; 第三步是描述系统的行为。第一步与第二步中所建立的模型都是静态的, 包括用例图、类图 (包含包)、对象图、组件图和配置图这 5 种图形, 是标准建模语言 UML 的静态建模机制。第三步中所建立的模型或者可以执行, 或者表示执行时的时序状态或交互关系, 它包括状态图、活动图、顺序图和协作图这 4 种图形, 是标准建模语言 UML 的动态建模机制。因此, 标准建模语言 UML 的主要内容也可以归纳为静态建模机制和动态建模机制两大类。

## 1.3 软件测试的重要性

软件无处不在, 人们在不同的场合都有可能会不知不觉地使用软件, 如日常生活中的手机、智能冰箱、新一代的数字彩电、计算机等。在日常使用软件中, 也或多或少会碰到一些不愉快的事情, 如信号显示不对、数据不完整、操作不灵活等, 但软件问题有时引起的麻烦还不止这些, 造成的危害可能会非常严重。在这一节, 通过一系列典型的软件质量

问题实例，阐述一个简单又重要的道理——软件测试的重要性。

### 1.3.1 软件所带来的悲剧

由于软件本身特有的性质决定了只要存在一个很小的错误，就可能带来灾难性的后果。虽然这种情况不是很多，但一旦发生后果是很严重的。这里，我们介绍几个典型的例子，如千年虫、“冲击波”计算机病毒、火星登陆事故、爱国者导弹防御系统和放射性机器系统等。

#### 1. 千年虫

在 20 世纪 70 年代，程序员为了节约非常宝贵的内存资源和硬盘空间，在存储日期时，只保留年份的后两位，如“1980”被存为“80”。但是，这些程序员万万没有想到他们的程序会一直被用到 2000 年，当 2000 年到来的时候，问题就会出现。比如银行存款程序在计算利息时，应该用现在的日期“2000 年 1 月 1 日”减去当时存款的日期，比如“1989 年 1 月 1 日”，结果应该是 21 年，如果利息是 3%，每 100 元银行要付给顾客大约 86 元利息。如果程序没有纠正年份只存储两位的问题，其存款年数就变为-89 年，变成顾客反要付给银行 1288 元的巨额利息。所以，当 2000 年快要来到的时候，为了这样一个简单的设计缺陷，全世界付出几十亿美元的代价。

#### 2. “冲击波”计算机病毒

新浪科技引用《商业周刊》网站在“网络安全”专题中的文章，对“冲击波”计算机病毒进行了分析。2003 年 8 月 11 日，“冲击波”计算机病毒首先在美国发作，使美国的政府机关、企业及个人用户的成千上万的计算机受到攻击。随后，冲击波蠕虫很快在因特网上广泛传播，中国、日本和欧洲等国家也相继受到不断的攻击，结果使十几万台邮件服务器瘫痪，给整个世界范围内的 Internet 通信带来惨重损失。

制造冲击波蠕虫的黑客仅仅用了 3 周时间就制造了这个恶毒的程序，“冲击波”计算机病毒仅仅是利用微软 Messenger Service 中的一个缺陷，攻破计算机安全屏障，可使基于 Windows 操作系统的计算机崩溃。该缺陷几乎影响当前所有微软 Windows 系统，它甚至使安全专家产生更大的忧虑：独立的黑客们将很快找到利用该缺陷控制大部分计算机的方法。

随后，微软公司不得不紧急发布补丁包，修正这个缺陷。

#### 3. 火星登陆事故

仅仅由于两个测试小组单独进行测试，没有进行很好沟通，缺少一个集成测试的阶段，结果导致 1999 年美国宇航局的火星基地登陆飞船在试图登陆火星表面时突然坠毁失踪。质量管理小组观测到故障，并认定出现误动作的原因极可能是某一个数据位被意外更改。什么情况下这个数据位被修改了？又为什么没有在内部测试时发现呢？

从理论上讲，登陆计划是这样的：在飞船降落到火星的过程中，降落伞将被打开，减缓飞船的下落速度。降落伞打开后的几秒钟内，飞船的 3 条腿将迅速撑开，并在预定地点着陆。当飞船离地面 1800 米时，它将丢弃降落伞，点燃登陆推进器，在余下的高度缓缓降

落地面。

美国宇航局为了省钱，简化了确定何时关闭推进器的装置。为了替代其他太空船上使用的贵重雷达，在飞船的脚上装了一个廉价的触点开关，在计算机中设置一个数据位来关掉燃料。很简单，飞船的脚不“着地”，引擎就会点火。不幸的是，质量管理小组在事后的测试中发现，当飞船的脚迅速摆开准备着陆时，机械震动在大多数情况下也会触发着地开关，设置错误的数据位。设想飞船开始着陆时，计算机极有可能关闭推进器，而火星登陆飞船下坠 1800 米之后冲向地面，必然会撞成碎片。

为什么会出现这样的结果？原因很简单。登陆飞船经过了多个小组测试。其中一个小组测试飞船的脚落地过程（leg fold-down procedure），但从没有检查那个关键的数据位，因为那不是这个小组负责的范围；另一个小组测试着陆过程的其他部分，但这个小组总是在开始测试之前重置计算机、清除数据位。双方本身的工作都没什么问题，就是没有合在一起测试，其接口没有被测，而问题就在这里，后一个小组没有注意到数据位已经被错误设定。

#### 4. 爱国者导弹防御系统

美国爱国者导弹防御系统是主动战略防御（即星球大战）系统的简化版本，它首次被用在第一次海湾战争对抗伊拉克飞毛腿导弹的防御作战中，总体上看效果不错，赢得各界的赞誉。但它还是有几次失利，没有成功拦截伊拉克飞毛腿导弹，其中一枚在沙特阿拉伯的多哈爆炸的飞毛腿导弹造成 28 名美国士兵死亡。分析专家发现，拦截失败的症结在于一个软件缺陷，当爱国者导弹防御系统的时钟累计运行超过 14 小时后，系统的跟踪系统就不准确。在多哈袭击战中，爱国者导弹防御系统运行时间已经累计超过 100 多个小时，显然那时系统的跟踪系统已经很不准确，从而造成这种结果。

#### 5. 放射性设备治死 4 个人

由于放射性治疗仪 Therac-25 中的软件存在缺陷，导致几个癌症病人受到非常严重的过量放射性治疗，其中 4 个人因此死亡。一个独立的科学调查报告显示：即使在加拿大原子能公司（AECL, Atomic Energy of Canada Limited）已经处理了几个特定的软件缺陷，这种事故还是发生了。造成这种低级但致命错误的原因是缺乏软件工程实践，一个错误的想法是软件的可靠性依赖于用户的安全操作。

### 1.3.2 其他一些例子

除了上述一些实例外，还有一些相对影响较小的由于软件缺陷而造成的事例，如因为软件缺陷给企业带来经济上或商业名誉上巨大的损失。下面介绍几个曾经给美国迪斯尼、微软、英特尔等公司造成或多或少的损失的例子。

#### 1. 迪斯尼的圣诞节礼物

1994 年圣诞节前夕，迪斯尼公司发布了第一个面向儿童的多媒体光盘游戏“狮子王童

话”。尽管在此之前，已经有不少公司在儿童计算机游戏市场上运作多年，但对迪斯尼公司而言，还是第一次进军这个市场。由于迪斯尼公司的著名品牌和事先的大力宣传及良好的促销活动，结果，市场销售情况非常不错，该游戏成为父母为自己孩子过圣诞节的必买礼物。

但结果却出人意料，12月26日，圣诞节后的第一天，迪斯尼公司的客户支持部电话开始响个不停，不断有人咨询、抱怨为什么游戏总是安装不成功，或没法正常使用。很快，电话支持部门就淹没在愤怒家长的责问声和玩不成游戏孩子们的哭诉之中，报纸和电视开始不断报道此事。

后来证实，迪斯尼公司没有对当时市场上的各种PC机型进行完整的系统兼容性测试，只是在几种PC机型上进行了相关测试。所以，这个游戏软件只能在少数系统中正常运行，但在大众使用的其他常见系统中却不能正常安装和运行。

## 2. 丹佛新机场启用推迟16个月

丹佛新国际机场希望被建成现代的（state-of-the-art）机场，它将拥有复杂的、计算机控制的、自动化的包裹处理系统，而且还有5300英里长的光纤网络。不幸的是，在这包裹处理系统中存在一个严重的程序缺陷，导致行李箱被绞碎，居然还开着自动包裹车往墙里面钻。

结果，机场启用推迟16个月，使得预算超过32亿美元，并且废弃这个自动化的包裹处理系统，使用手工处理包裹系统。

## 3. Windows 2000 安全漏洞

微软曾经承认，Windows 2000 操作系统远程服务中存在7个漏洞，并发布了相应的补丁软件来进行修补。微软远程服务是一种用于远程登录到大学、政府机关以及其他机关网站的系统或邮件服务器上的协议。Windows 2000 内运行的远程服务软件所出现的安全漏洞可能导致3种截然不同的安全隐患——拒绝服务、权限滥用、信息泄露。安全漏洞可能会导致DOS攻击，使得系统无法向合法用户提供远程登录服务。而另外两种安全缺陷更严重些，都涉及到系统管理权限，有可能帮助攻击者通过键盘输入的一个系统功能在无须登录的情况下完全控制Windows 2000 系统。这样攻击者便可以在计算机上执行任意操作，包括在计算机上添加用户，安装或删除系统组件，添加或删除软件，破坏数据，或执行其他操作。

据美国军方在2002年3月18日证实，微软网络软件中一个原来未知的缺陷让一名联机攻击者控制了美国国防部服务器的公开接口。美国陆军网络技术事业司令部（NTEC）主任 Dmuchowski 上校说，受到攻击的军事网站不属于军方。上校强调，陆军很认真地对待这种威胁。微软 IIS 5.0 和 Windows 2000 中的这个缺陷使微软公司的安全团队人吃一惊，因为没有一名安全研究人员曾经发现这个问题。在通常情况下，发现缺陷的安全研究人员或黑客会公布缺陷详情，或者将问题报告提交给软件制作者。

#### 4. 英特尔奔腾芯片缺陷

在计算机的“计算器”程序中输入以下算式：

$$(4195835 / 3145727) \times 3145727 - 4195835$$

如果答案是 0，就说明该计算机浮点运算没问题。如果答案不是 0，就表示计算机的浮点除法存在缺陷。1994 年，英特尔奔腾 CPU 芯片就曾经存在这样一个软件缺陷，而且被大批生产出来卖到用户那里，最后，英特尔为自己处理软件缺陷的行为道歉并拿出 4 亿多美元来支付更换坏芯片的费用，可见，这个软件缺陷造成的损失有多大！

这个缺陷是美国弗吉尼亚州 Lguchbny 大学的 Thomas R. Nicely 博士发现的。他在奔腾 PC 机上做除法实验时记录了一个没想到的结果。他把发现的问题放到因特网上，随后引发了一场风暴，成千上万的人发现了同样的问题，以及其他得出错误结果的情形。万幸的是，这种情况很少见，仅仅在进行精度要求很高的数学、科学和工程计算中才导致错误。大多数进行财会管理和商务应用的用户根本不会遇到此类问题。

这个故事不仅说明软件缺陷所带来的问题，更重要地是说明对待软件缺陷的态度。

- 英特尔的软件测试工程师在芯片发布之前进行内部测试时已经发现了这个问题，但管理层认为这没有严重到一定要修正，甚至需要公布这个问题。
- 当软件缺陷被发现时，英特尔通过新闻发布和公开声明试图掩饰这个问题的严重性。
- 受到压力时，英特尔承诺更换有问题的芯片，但要求用户必须证明自己受到软件缺陷的影响。

结果舆论大哗。因特网新闻组充斥着愤怒的客户要求英特尔解决问题的呼声。得到这个教训之后，英特尔在网站上报告已发现的问题，并认真对待客户在因特网新闻组上的反馈意见。

#### 5. 赛门铁克安全软件的安全缺陷

安全软件制造商赛门铁克公司曾通知客户，使用该公司联机安全检测（security check）服务的用户有可能下载了一个带有缺陷的 ActiveX 控件，该控件有可能被入侵者利用并侵入受害者的计算机。

安全检测（security check）服务的目的是帮助人们锁定系统并安装一个 ActiveX 控件以对计算机进行扫描。但具有讽刺意味的是，在扫描后仍存留在计算机中的这个 ActiveX 控件具有一个内存缺陷，该缺陷会被攻击者利用并进入计算机。

不过，该问题已经得到了较好的解决赛门铁克已经用一款新的软件替换和覆盖了原来的软件。

### 1.3.3 测试是软件开发重要环节之一

在 G.J.Myers 的经典著作《软件测试技巧》中给出了测试的定义：“程序测试是为了发现错误而执行程序的过程。”测试的目的是发现程序中的错误，是为了证明程序有错，而不

是证明程序无错。在软件开发过程中,分析、设计与编码等工作都是建设性的,惟独测试似乎带有“破坏性”。测试可视为分析、设计和编码3个阶段的“最终复审”,在软件质量保证中具有重要地位。为了确保软件的质量,较理想的做法应该是对软件的开发过程,按软件工程各阶段形成的结果,分别进行严格的审查。

软件测试在软件生命周期中占据重要的地位,在传统的瀑布模型中,软件测试学仅处于运行维护阶段之前,是软件产品交付用户使用之前保证软件质量的重要手段。近来,软件工程界趋向于一种新的观点,即认为软件生命周期每一阶段中都应包含测试,从而检验本阶段的成果是否接近预期的目标,尽可能早地发现错误并加以修正,如果不在早期阶段进行测试,错误的延时扩散常常会导致最后成品测试的巨大困难。

事实上,对于软件来讲,不论采用什么技术和什么方法,软件中仍然会有错。采用新的语言、先进的开发方式、完善的开发过程,可以减少错误的引入,但是不可能完全杜绝软件中的错误,这些引入的错误需要测试来找出,软件中的错误密度也需要测试来进行估计。测试是所有工程学科的基本组成单元,是软件开发的重要组成部分。自有程序设计的那天起测试就一直伴随着。统计表明,在典型的软件开发项目中,软件测试工作量往往占软件开发总工作量的40%以上。而在软件开发的总成本中,用在测试上的开销要占30%~50%。如果把维护阶段也考虑在内,讨论整个软件生存期时,测试的成本比例也许会有所降低,但实际上维护工作相当于二次开发,乃至多次开发,其中必定还包含有许多测试工作。

软件测试在产品开发中占据相当重要的一部分,是软件行业二十几年的实践所证明的一个道理,或者说是从不断的失败中总结出来的经验。以微软公司为例,大家可以感觉到,微软以前的产品时时会发生崩溃、死机等现象,而今天的产品则比5年前的产品功能要强得多,稳定性却好得多。为什么呢?这是因为微软公司重视测试工作,测试人员越来越多,如今微软的软件测试人员是开发人员的1.5~2.5倍。其次,测试人员越来越有经验,测试工作也就越做越好。正是由于清晰地认识到了软件测试的重要性,微软的产品质量才有了明显的提高。

最初,微软公司与大家一样,认为测试不重要,重要的是开发人员。通常,一个团队中有几百个开发人员,但只有几个测试人员,并且开发人员的待遇要比测试人员高很多。经过多年的实践后,微软公司发现,去修正那些出现问题的产品所花的钱,比多聘用几个测试人员的费用要多得多,所以,开始不断地聘用测试人员。同时,现在测试人员的待遇和开发人员的待遇非常接近,测试人员水平越高,找到bug的时间就越早,软件就越容易更正,产品发布之后越稳定,公司赚的钱也越多。这也是多数软件公司慢慢悟出来的道理,软件测试是软件产品开发中最重要的几个环节之一。

## 小结

本章主要围绕软件及其开发过程来介绍一些基本而又重要的概念。基于这些概念,我们讨论了软件开发的特性,对这些特性的理解有助于我们对软件开发生命周期、软件开发流程模型的认识。瀑布模型是一个经典的开发流程模型,几乎被所有的软件工程书介绍,

但目前更常用的开发流程模型不是瀑布模型，而是快速原型方法、RAD 模型和并发模型等。

正确选择软件开发模型有利于提高软件开发的效率和可视性，而且有利于分析软件开发中出现的問題，从而不断的改进开发流程。但是，不论开发的模型多先进，开发的软件中肯定还会出现问题，有些问题如果不能及时发现、控制和处理，可能会给软件开发公司带来不可估量的损失，甚至造成灾难性的后果。所以，当软件业不断成熟，走入工业化阶段，软件产品的质量是软件产品及其开发公司的生命，这也说明，在软件开发领域，软件测试越来越重要。

## 思考题

1. 为什么相对瀑布模型，V 模型可以缩短开发周期？
2. 请谈谈自己对软件测试重要性的看法。
3. 比较原型模型和螺旋模型的优缺点。



# 第2章 软件测试的基本 概念和方法

在上一章中，着重介绍了软件及其开发过程，这些内容有助于理解软件测试的基本思想和方法。从本章开始讨论软件测试的基本概念和方法。

软件测试是软件质量保证的一个手段，软件测试的概念相对软件质量而存在，所以让读者先了解软件质量的概念，然后使读者更好理解软件缺陷（bug）是什么，以及软件测试的白盒子和黑盒子方法、静态的和动态的方法等内容，最后，使读者建立一个完整的软件测试概念，包括软件测试的分类和阶段、软件测试的工作内容等。

## 2.1 软件质量就是客户的满意度

软件质量建立在一般产品质量概念及理论的基础之上，既具有一般产品质量特性，又具有软件自身的特性。要对“软件质量”这个概念有一个全面的理解，首先必须搞清楚什么是普通意义上的质量概念，再分析软件质量所蕴含的特性或特征。

### 2.1.1 质量的概念

“质量（Quality）”这个词，从汉语文字来看，是由“质”和“量”构成的，就是在质和量上的程度。量的含义比较容易理解，而质的含义相对比较复杂。“质”作为形容词，具有“朴实、朴素”、“诚实、诚信”等含义。在这里，我们可以理解“质”为事物的素质、本质或禀性。

从哲学角度说，量的积累能产生质的飞跃。量是过程（过程品）的累积，不断增加并完善过程品，最终实现质的飞跃。当满足一定需求时，即达到基本的质量要求，而满足需求的程度即是我们所说的质量优劣。

在权威的韦氏大词典（*Webster's Revised Unabridged Dictionary*, © 1996, 1998 MICRA, Inc）中，对 Quality 有详尽的解释：

- Quality is the condition of being of such and such a sort as distinguished from others; nature or character relatively considered, as of goods; character; sort; rank.
- Quality is the special or temporary character; profession; occupation; assumed or asserted rank, part, or position. ...
- That which makes, or helps to make, anything such as it is; anything belonging to a subject, or predicable of it; distinguishing property, characteristic, or attribute; peculiar power, capacity, or virtue; distinctive trait; as, the tones of a flute differ from those of

a violin in quality; the great quality of a statesman.

- An acquired trait; accomplishment; acquisition.
- Superior birth or station; high rank; elevated character.

但是对“质量”的解释和说明依旧困难，或者说，我们使用“质量”这个词本身就具有风险。传统的理性观点把世界分为主观和客观两部分，但质量似乎被排除在这种区分之外，既不是客观的，也不是主观的。质量不是客观的，因为没有什么科学仪器可以直接测出质量来；质量也不是主观的，它不仅存在于人们的脑海中。

世界著名的质量管理专家朱兰对“质量”给出了一个确切的含义，即满足使用要求的基础是质量特征，产品的任何特性（性质、属性等）、材料或满足使用要求的过程都是质量特征。从而，演变为国际标准化的定义，即1986年ISO8492中所给出的质量定义：质量是产品或服务所满足明示或暗示需求能力的特性和特征的集合。

IEEE在“Standard Glossary of Software Engineering Terminology”中给出的质量定义是被普遍接受的概念，即质量是系统、部件或过程满足明确需求。

在Rational Unified Process（Rational标准过程理论）中，质量被定义为：

- 满足或超出认定的一组需求，并使用经过认可的评测方法和标准来评估，还使用认定的流程来生产。

因此，质量不是简单地满足用户的需求，还得包含证明质量达标所使用的评测方法和标准，以及如何实施可管理、可重复使用的流程，以确保由此流程生产的产品已达到预期的质量水平。

质量还是一个复杂的多层面概念，从不同的层面或角度对质量有不同的理解：

- 先验论：质量是产品的一种可以认识但不可定义的性质。
- 用户角度：质量是产品满足使用目的的程度。
- 制造者的观点：质量是产品性能符合规格要求的程度。
- 产品观点：质量是联结产品固有性能的纽带。
- 基于价值观点：质量依赖于顾客愿意付给产品报酬的数量。

质量所关联的另外一个概念就是“客户”，质量和客户息息相关，两者相对而存在。对客户定义至少存在两种范畴——内部的和外部的：

- 外部的客户是产品的实际使用者或服务的对象，是传统意义上大家所认可的客户。
- 内部的客户是更为广泛意义上的客户，客户可以被理解为下一道工序的接受者。这样，在软件生产的环节中有关的人员都可被定义为这一类型的客户，软件的设计者是需求分析人员的客户，编程人员是设计者的客户，软件测试者是编程人员的客户。

## 2.1.2 软件质量的内涵

软件质量是一个软件企业成功的必要条件，其重要性无论怎样强调都不过分。软件质量与传统意义上的质量概念并无本质差别，只是针对软件的某些特性进行了调整。

软件质量由三部分构成：

- 软件产品的质量，即满足使用要求的程度。

- 软件开发过程的质量，即能否满足开发所带来的成本、时间和风险等要求。
- 应用领域或业务上质量。

总结起来，高品质软件应该是相对的无产品缺陷（bug free）或只有极少量的缺陷，它能够准时递交给客户，所花费用都在预算内，并且满足客户需求，是可维护的。但是，有关质量好坏的最终评价依赖于用户的反馈。

软件质量具有 3A 特性：accountability（可说明性）、availability（有效性）和 accessibility（易用性）。

- 可说明性：用户可以基于产品或服务的描述和定义（例如：市场需求说明书、功能设计说明书）加以使用。
- 有效性：产品或服务对于客户的需求是否能保持有效，如具有 99.99% 有效性，可以说达到质量要求。
- 易用性：对于用户，产品或服务非常容易使用并且一定是非常有用的功能（例如：确认测试和用户可用性测试）。

在 Rational Unified Process 中，质量被定义为具有以下三个维度，它们和上面所述的 3A 特性有一定的对应性。

- 功能（对应可说明性，但概念不同）：按照既定意图和要求，执行指定用例的能力。
- 可靠性（有效性）：软件坚固性和可靠性（防故障能力，如防止崩溃、内存丢失等能力）、资源利用率、代码完整性以及技术兼容性等。
- 性能（易用性）：测试对象的计时配置文件和操作特征。计时配置文件包括代码的执行流、数据访问、函数调用和系统调用。性能的操作特征包括与作业负载相关的特征，如响应时间、操作可靠性（MTTF），以及与操作限制相关的特征，如负载容量或强度。

对于广义上的软件质量，又是由产品质量、过程质量和商业环境质量这三者决定的。下面分别做进一步介绍。

### 1. 产品质量

产品质量是人们实践产物的属性和行为，是可以辨识的，并能进行科学的描述。可以通过一些方法和人类活动，来改进产品的质量。软件产品质量一般体现在以下几个方面。

- 功能性（functionality）：软件所实现的功能达到它的设计规范和满足用户需求的程度。
- 可用性（usability）：对于一个软件，用户学习、操作、准备输入和理解输出所作努力的程度，如安装简单方便，容易使用；界面友好，并能适用于不同特点的用户，包括对残疾人、有缺陷的人能提供产品使用的有效途径或手段。
- 可靠性（reliability）：是用户使用的根本。在规定的条件和条件下，软件所能维持其正常的功能操作、性能水平的程度。
- 性能（performance）：在指定条件下，用软件实现某种功能所需的计算机资源（包括内存大小、CPU 占用时间等）的有效程度。
- 容量（capacity）：系统的接受力、容纳或吸收的能力，或某项功能的最大量或最

大限度,有时需要确定系统特定的需求所能容纳的最大量、所能表现的最大值。如 Web 系统能承受多少并发用户访问,会议系统可以承受的与会人数等。

- **可测量性 (scalability)**: 系统某些特性可以通过一些量化的数据指标描述其当前状态或理想状态。
- **可维护性 (service manageability)**: 在一个运行软件中,当环境改变或软件发生错误时,进行相应修改所做努力的程度。
- **兼容性 (compatibility)**: 软件从一个计算机系统或环境移植到另一个系统或环境的容易程度,或者是一个系统和外部条件共同工作的容易程度。兼容性表现在多个方面,如系统的软件和硬件的兼容性、不同版本的软件系统、数据的兼容性。
- **可扩展性 (extensibility)**: 指将来功能增加,系统扩充的难易程度或能力。

计算机界对软件产品质量进行了较多的研究,得到了一些有效的质量模型,包括 McCall 模型、Boehm 模型、ISO9126 模型。图 2-1 就是 McCall 模型的示意图。

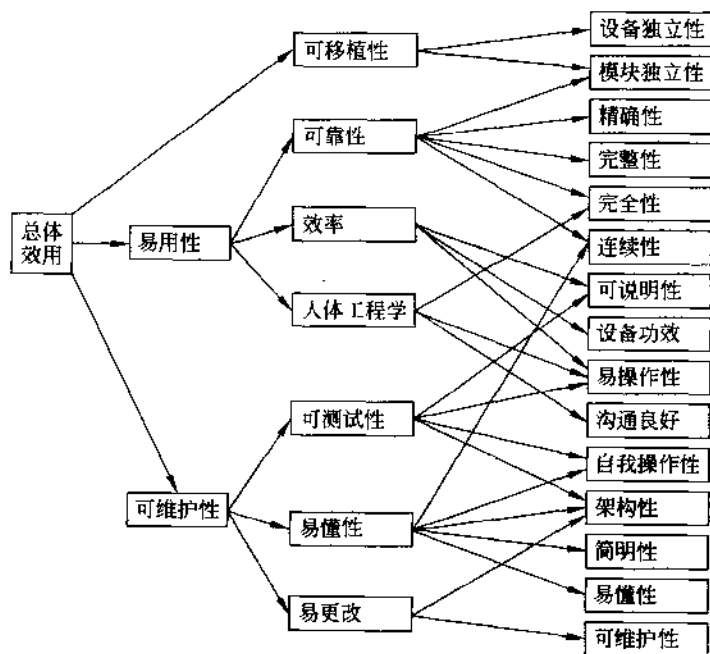


图 2-1 McCall 模型的示意图

## 2. 过程质量

探索复杂系统开发过程的秩序,按一定规程工作,可以较合理地达到目标。规程由一系列活动组成,形成方法体系,建立严格的工程控制方法,要求每一个人都要遵守工程规范。目前主要流行的过程改进模型或工程规范有以下几种。

- 软件能力成熟度模型 (CMM, Capability Maturity Model)。
- 国际标准过程模型 ISO9000。
- 软件过程改进和能力决断 (SPICE, Software Process Improvement and Capability dEtermination)。

这些内容会在第4章做较详细的介绍。

### 3. 软件在商业环境中所体现的质量

开发软件的目的是要投放市场，其质量的表现最终还要在其生存的商业环境中体现出来。软件在商业环境中的表现好坏，不一定与产品质量和软件开发过程质量保持同步，一个好的软件产品不一定获得好的市场。原因很多，因为软件产品会涉及与其商业/应用环境相关的一些因素，包括产品的客户培训、向市场发布的日程安排、商业风险评估、产品的客户、维护和服务成本等。

软件产品投放到市场时，要考虑培训的周期和用户的习惯意识。比如，一个新版本的软件系统在界面上做了彻底的改变，界面变得非常友好，从产品本身看，是好质量的一种体现，但在商业环境中，可能会给产品的推广带来一些阻力，因为原来的用户不一定能适应这种太大的变化。比如微软的 Windows 操作系统，从 Windows 3.x、Windows 95、Windows 98、Windows 2000 到 Windows XP，在界面和操作变化上，就遵守循序渐进的原则。

软件发布的时间会受到市场的影响，或者说，制定一个合适的软件发布时间，对打开市场有较大的影响。控制或降低软件的风险和成本，提高软件整体的生产能力，都是软件开发企业或团体所追求的。

## 2.2 软件缺陷 (bug) 是什么

由于软件开发人员思维上的主观局限性，且目前开发的软件系统都具有相当的复杂性，决定了在开发过程中出现软件错误是不可避免的，软件过多的或严重的错误会导致程序或系统的失效。软件错误产生的主要原因有：

- 需求规格说明书 (requirement specification 或 functional specification) 包含错误的需求、或漏掉一些需求，或没有准确表达客户所需要的内容。
- 需求规格说明书中有些功能不可能或无法实现。
- 系统设计 (system design) 中的不合理性。
- 程序设计中的错误。
- 程序代码中的问题，包括错误的算法、复杂的逻辑等。
- 若能及早排除软件开发中的错误，有效地减少后期工作可能遇到的问题，就可以尽可能地避免付出高昂的代价，从而大大提高系统开发过程的效率。

根据 G.J. Myers 观点，对软件测试的目的可以简单地概括为：

- 软件测试是为了发现错误而执行程序的过程。
- 一个好的测试能够在第一时间发现程序中存在的错误。
- 一个好的测试是发现了至今尚未发现的错误的测试。

他指出“软件测试是为了发现错误而执行程序的过程”，而更多专家认为软件测试的范围应当更为广泛，除了要考虑测试结果的正确性以外，还应关心程序的效率、可适用性、维护性、可扩充性、安全性、可靠性、系统性能、系统容量、可伸缩性、服务可管理性、

兼容性等等因素。随着人们对软件测试更广泛、深刻的认识,可以说对软件质量的判断决不只限于程序本身,而是整个软件研制过程。

不管怎么定义软件测试,基本的结论是一致的,即软件测试是为了发现软件产品所存在的任何意义上的软件缺陷(bug),从而纠正(fix)这些软件缺陷,使软件系统更好地满足用户的需求。那么,什么是软件缺陷呢?

## 2.2.1 软件缺陷的定义和种类

对于软件存在的各种问题,我们都用“软件缺陷”这个词,在英文中人们喜欢用一个不贴切但已经专用的词“bug(臭虫)”,实际和“缺陷(bug)”词相近的词还有很多,如

缺点(defect)	偏差(variance)
谬误(fault)	失败(failure)
问题(problem)	矛盾(inconsistency)
错误(error)	毛病(incident)
异常(anomy)	

但人家都习惯使用“bug——软件缺陷”这个词,它包含了一些偏差、谬误或错误,更多地表现在功能上的失败(failure)和实际需求的不一致,即矛盾(inconsistency)。

软件缺陷(bug),即计算机系统或者程序中存在的任何一种破坏正常运行能力的问题、错误,或者隐藏的功能缺陷、瑕疵。缺陷会导致软件产品在某种程度上不能满足用户的需求。在 IEEE 1983 of IEEE Standard 729 中对软件缺陷下了一个标准的定义。

- 从产品内部看,软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题。
- 从外部看,软件缺陷是系统所需要实现的某种功能的失效或违背。

软件缺陷就是软件产品中所存在的问题,最终表现为用户所需要的功能没有完全实现,没有满足用户的需求。

软件缺陷表现的形式有多种,不仅仅体现在功能的失效方面,还体现在其他方面。软件缺陷的主要类型有:

- 功能、特性没有实现或部分实现。
- 设计不合理,存在缺陷。
- 实际结果和预期结果不一致。
- 运行出错,包括运行中断、系统崩溃、界面混乱。
- 数据结果不正确、精度不够。
- 用户不能接受的其他问题,如存取时间过长、界面不美观。

美国商务部国家标准和技术研究所(NIST)进行的一项研究表明,软件中的 bug 每年给美国经济造成的损失高达 595 亿美元。说明软件中存在的缺陷所造成的损失是巨大的,从反面又一次证明软件测试的重要性。如何尽早彻底地发现软件中存在的缺陷是一项非常复杂,需要创造性和高度智慧的工作。同时,软件的缺陷是软件开发过程中的重要属性,反映软件开发过程中需求分析、功能设计、用户界面设计、编程等环节所隐含的问题,也

为项目管理、过程改进提供了许多信息。

软件缺陷一旦被发现,就要设法找出引起这个缺陷的原因,分析对产品质量的影响,然后确定软件缺陷的严重性和处理这个缺陷的优先级。各种软件缺陷所造成的后果是不同的,有的仅仅是不方便,比如计算机游戏只能用键盘玩而不能用鼠标玩;也可能是灾难性的,比如在第1章介绍的几个例子。在这些事件中,显然软件未按预期目标运转。作为软件测试员,可能所发现的大多数问题不是那么明显、严重,而是难以觉察的简单而细微的错误,有些是真正的错误,也有些不是。一般来说,问题越严重的,其优先级越高,越要得到及时的纠正。软件公司对缺陷严重性级别的定义不尽相同,但一般可以概括为4种级别:

- 致命的(fatal):致命的错误,造成系统或应用程序崩溃(crash)、死机、系统悬挂,或造成数据丢失、主要功能组完全丧失等。
- 严重的(critical):严重错误,指功能或特性(feature)没有实现,主要功能丧失,导致严重的问题,或致命的错误声明。
- 一般的(major):不太严重的错误,这样的软件缺陷虽然不影响系统的基本使用,但没有很好地实现功能,没有达到预期效果。如次要功能丧失,提示信息不太准确,或用户界面差,操作时间长等。
- 微小的(minor):一些小问题,对功能几乎没有影响,产品及属性仍可使用,如有个别错别字、文字排列不整齐等。

除了这4种之外,有时需要“建议(suggestion)”级别来处理测试人员所提出的建议或质疑,如建议程序做适当的修改,来改善程序运行状态,或对设计不合理、不明白的地方提出质疑。

软件缺陷除了严重性之外,还存在反映软件缺陷处于一种什么样的状态,便于跟踪和管理某个产品的缺陷,可以定义不同的bug状态。

- 激活状态(Active 或 Open):问题还没有解决,测试人员新报的bug,或验证后bug仍然存在。
- 已修正状态(Fixed 或 Resolved):开发人员针对所存在的缺陷,修改程序,认为已解决问题,或通过单元测试。
- 关闭或非激活状态(Close 或 Inactive):测试人员验证fixed bug后,确认bug不存在之后的状态。

以上是缺陷的3种基本状态,除此之外,还有一些情况需要相应的状态描述,如果所报的bug目前无法解决或是第三方产品引起的,可以置为Hold状态;如果所报的bug暂时不需要解决或在下一版本解决更彻底一些,可以置为Differed状态。这里仅作一个简单的介绍,详细讨论在第15章。

## 2.2.2 软件缺陷的产生

如前所说,由于软件系统越来越复杂,不管是需求分析、程序设计等都面临越来越大的挑战。软件缺陷的产生,首先是不可避免的。其次,造成软件缺陷的主要原因有哪些?我

们可以从软件本身、团队工作和技术问题等多个方面分析，比较容易确定造成软件缺陷的主要因素，归纳如下。

### 1. 技术问题

- 算法错误：在给定条件下没能给出正确或准确的结果。
- 语法错误：一般情况下，对应的编程语言编译器可以发现这类问题；对于解释性语言，只能在测试运行的时候发现。
- 计算和精度问题：计算的结果没有满足所需要的精度。
- 系统结构不合理、算法不科学，造成系统性能低下。
- 接口参数传递不匹配，导致模块集成出现问题。

### 2. 团队工作

- 系统分析时对客户的需求不是十分清楚，或者和用户的沟通存在一些困难。
- 不同阶段的开发人员相互理解不一致，软件设计对需求分析结果的理解偏差，编程人员对系统设计规格说明书中某些内容重视不够，或存在着误解。
- 设计或编程上的一些假定或依赖性，没有得到充分的沟通。

### 3. 软件本身

- 文档错误、内容不正确或拼写错误。
- 没有考虑大量数据使用场合，从而可能会引起强度或负载问题。
- 对程序逻辑路径或数据范围的边界考虑不够周全，漏掉某几个边界条件造成的容量或边界错误。
- 对一些实时应用系统，要进行精心设计和处理，保证精确的时间同步，否则容易引起时间上不协调、不一致性带来的问题。
- 没有考虑系统崩溃后的自我恢复或数据的异地备份、灾难性恢复等问题，从而存在系统安全性、可靠性的隐患。
- 硬件或系统软件上存在的错误。
- 软件开发标准或过程上的错误。

## 2.2.3 软件缺陷的构成

根据上面讨论，我们知道软件缺陷是由很多原因造成的，如果把它们按需求分析结果——规格说明书、系统设计结果、编程的代码等归类起来，我们发现：结果规格说明书是软件缺陷出现最多的地方，见图 2-2。

软件产品规格说明书为什么是软件缺陷存在最多的地方，主要原因有以下几种。

- 用户一般是非计算机专业人员，软件开发人员和用户的沟通存在较大困难，对要开发的产品功能理解不一致。
- 由于软件产品还没有设计、开发，完全靠想象去描述系统的实现结果，所以有些



特性还不够清晰。

- 需求变化的不一致性。用户的需求总是在不断变化的，这些变化如果没有在产品规格说明书中得到正确的描述，容易引起前后文、上下文的矛盾。
- 对规格说明书不够重视，在规格说明书的设计和写作上投入的人力、时间不足。
- 没有在整个开发队伍中进行充分沟通，有时只有设计师或项目经理得到比较多的信息。

排在产品规格说明书之后的是设计，编程排在第三位。许多人印象中，软件测试主要是找程序代码中的错误，这是一个认识的误区。

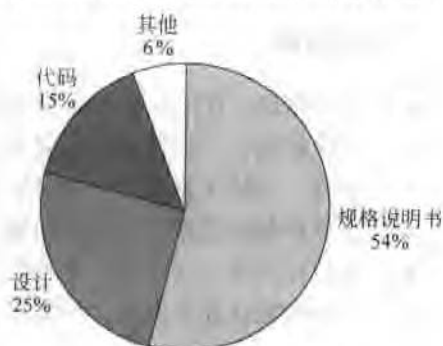


图 2-2 软件缺陷构成示意图

如果从软件开发各个阶段来看软件缺陷分布，也主要集中在需求分析、系统设计中，代码的错误要比前两个阶段少，如图 2-3 所示。

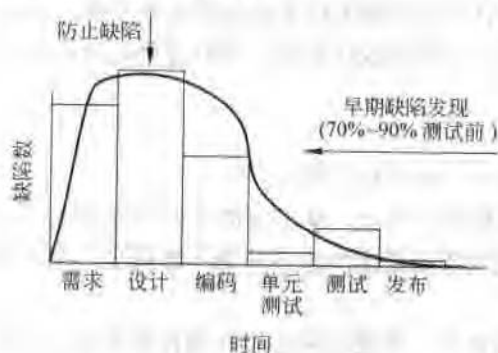


图 2-3 软件缺陷在不同阶段的分布图

## 2.2.4 修复软件缺陷的代价

在一开始讨论软件测试时，我们曾说测试人员要从需求分析时就介入进去，问题发现得越早越好。缺陷被发现之后，要尽快修复这些被发现的缺陷。为什么要这样做呢？原因很简单，错误并不只是在编程阶段产生，需求和设计阶段同样会产生错误。也许一开始，只是一个很小范围内的潜在错误，但随着产品开发工作的进行，小错误会扩散成大错误，为了修改后期的错误所做的工作要大得多，即越到后来往前返工也越远。如果错误不能及早发现，那可能造成越来越严重的后果。缺陷发现或解决的越迟，成本就越高。

Boehm 在 *Software Engineering Economics* (1981 年) 一书中写到：平均而言，如果在需求阶段修正一个错误的代价是 1，那么，在设计阶段就是它的 3~6 倍，在编程阶段是它的 10 倍，在内部测试阶段是它的 20~40 倍，在外部测试阶段是它的 30~70 倍，而到了产品发布出去时，这个数字就是 40~1000 倍。修正错误的代价不是随时间线性增长，而几乎是呈指数增长的。图 2-4 就是说明这样一个道理。

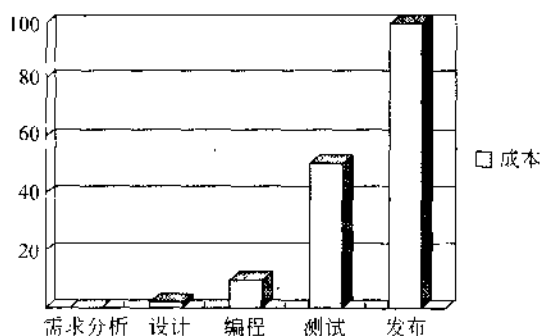


图 2-4 软件缺陷随着时间的推移带来的成本越来越大

## 2.3 软件测试的基本方法

通过对软件缺陷的产生、分类、构成所做的讨论，使我们更容易理解软件测试的目的，软件测试是为了更快、更早期地将软件产品或软件系统中所存在的各种问题找出来，并促使程序员尽快地解决这些问题，最终及时地向客户提供高质量的软件产品。要做到这一点，确保找出系统中所有或绝大部分的软件缺陷，必须建立在软件测试非常有用的基本方法之上。软件测试方法可根据测试对象在测试过程中是否发生状态变化分为两大类：动态测试和静态测试方法；又可根据对测试对象了解的程度，或者按哲学的观点，分为黑盒测试和白盒测试两类。

### 2.3.1 软件测试的原则

在介绍软件测试的基本方法之前，我们必须先介绍软件测试的基本原则。原则是重要的，方法应该在这个原则指导下进行。软件测试的基本原则是站在用户的角度，对产品进行全面测试，尽早、尽可能多地发现缺陷（bug），并负责跟踪和分析产品中的问题，对不足之处提出质疑和改进意见。零缺陷（zero-bug）是一种理想，足够好（good-enough）是测试的原则。

如果进一步去研究测试的原则，我们发现，在软件测试过程中，应注意和遵循的原则可以概括为 10 项。

- 所有测试的标准都是建立在用户需求之上。正如我们所知，软件测试的目标在于揭示错误。测试人员要始终站在用户的角度去看问题，系统中最严重的错误是那些导致程序无法满足用户需求的错误。
- 软件测试必须基于“质量第一”的思想去开展各项工作。当时间和质量冲突时，时间要服从质量。
- 事先定义好产品的质量标准。只有建立了质量标准，才能根据测试的结果，对产品的质量进行分析和评估。同样，测试用例应确定预期输出结果。如果无法确定

测试结果,则无法进行校验。必须用事先精确对应的输入数据和输出结果来对照检查当前的输出结果是否正确,做到“有的放矢”。

- 软件项目一启动,软件测试也就开始,而不是等程序写完,才开始进行测试。在代码完成之前,测试人员要参与需求分析、系统或程序设计的审查工作,而且还要准备测试计划、测试用例、测试脚本和测试环境。测试计划可以在需求模型一完成就开始,详细的测试用例定义可以在设计模型被确定后开始。
- 穷举测试是不可能的。即使一个大小适度的程序,其路径排列的数量也非常大,因此,在测试中不可能运行路径的每一种组合。然而,充分覆盖程序逻辑,并确保程序设计中使用的条件是有条件的。
- 第三方进行测试会更客观,更有效。程序员应避免测试自己的程序,为达到最佳的效果,应由第三方来进行测试。测试是带有“挑剔性”的行为,心理状态是测试自己程序的障碍。同时对于需求规格说明的理解产生的错误也很难在程序员本人测试时被发现。
- 软件测试计划是做好软件测试工作的前提。所以在进行实际测试之前,应制定良好的、切实可行的测试计划并严格执行,特别要确定测试策略和测试目标。
- 测试用例是设计出来的,不是写出来的,所以要根据测试的目的,采用相应的方法去设计测试用例,从而提高测试的效率,更多地发现错误,提高程序的可靠性。除了检查程序是否做了它应该做的事,还要看程序是否做了它不该做的事。不仅应选用合理的输入数据,对于非法的输入也要设计测试用例进行测试。
- 对发现错误较多的程序段,应进行更深入的测试。一般来说,一段程序中已发现的错误数越多,其中存在的错误概率也就越大。
- 重视文档,妥善保存一切测试过程文档。测试计划、测试用例、测试报告都是检查整个开发过程的主要依据,有利于今后流程改进,同时也是测试人员的智慧结晶和经验积累,对新人或今后的工作都有指导意义。

除了这10项原则之外,在测试当中,还有许多注意事项或经验。

- 应当把“尽早和不断地测试”作为测试人员的座右铭。
- 回归测试的关联性一定要引起充分的注意,修改一个错误而引起更多错误出现的现象并不少见。
- 测试应从“小规模”开始,逐步转向“大规模”。最初的测试通常把焦点放在单个程序模块上,进一步测试的焦点则转向在集成的模块簇中寻找错误,最后在整个系统中寻找错误。
- 不可将测试用例置之度外,排除随意性。特别是对做了修改之后的程序进行重新测试时,如不严格执行测试用例,将有可能忽略由修改错误而引起的新错误。
- 必须彻底检查每一个测试结果。事实上有相当一部分最终发现的错误是在早期测试结果中遗漏的。
- 一定要注意测试中的错误集中发生现象,这和程序员的编程水平和习惯有很大的关系。
- 对测试错误结果一定要有一个确认的过程。一般由A测试出来的错误,一定要有

一个B来确认。严重的错误可以召开评审会进行讨论和分析。

## 2.3.2 白盒测试和黑盒测试

从哲学观点看,分析问题和解决问题的方法有两种:白盒子方法和黑盒子方法。所谓白盒子方法就是能够看清楚事物的内部,即了解事物的内部结构和运行机制,通过剖析事物的内部结构和运行机制,来处理问题。所谓黑盒子方法是没办法或不去了解事物的内部结构和运行机制,而把整个事物看成一个整体——黑盒子,通过分析事物的输入、输出以及周边条件来分析和处理问题。

软件测试具有相类似的哲学思想。根据测试是针对系统的内部结构还是针对具体实现算法的角度来进行,分别称为白盒测试和黑盒测试。

### 1. 黑盒测试法 (black-box testing)

黑盒测试,也称功能测试或数据驱动测试,如图2-5所示。它不管程序内部结构是什么样的,只是从用户出发,根据产品应该实现的实际功能和已经定义好的产品规格,来验证产品所应该具有的功能是否实现,每个功能是否都能正常使用,是否满足用户的要求。

在测试时,把程序看作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,测试人员针对程序接口和用户界面进行测试,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。

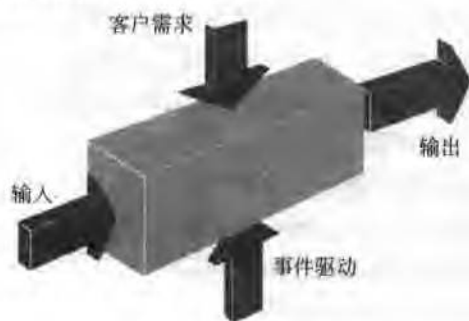


图 2-5 黑盒测试方法示意图

黑盒测试主要用于发现以下情况:

- 是否有不正确或遗漏了的功能;
- 在接口上,能否正确地接受输入数据,能否产生正确的输出信息;
- 访问外部信息是否有错;
- 性能上是否满足要求;
- 界面是否错误,是否不美观;
- 初始化和终止错误。

黑盒测试方法主要用于软件确认测试。其具体方法有等价类划分、边界值分析、错误

推测法、因果图等。

黑盒测试方法着眼于程序外部结构,不考虑内部逻辑结构,针对软件界面和软件功能进行测试。在用黑盒法测试时,必须在所有可能的输入条件和输出条件中确定测试数据。黑盒测试中不可能做到穷举测试,因此局限于功能测试是远远不够的,还要结合白盒测试方法,进行逻辑和路径测试。

## 2. 白盒测试法 (white-box testing)

白盒测试,也称结构测试或逻辑驱动测试,也就是已知产品的内部工作过程,清楚最终生成软件产品的计算机程序的结构和语句,按照程序内部的结构测试程序,测试程序内部的变量状态、逻辑结构、运行路径等,检验程序中的每条通路是否都能按预定要求正确工作,检查程序内部动作或运行是否符合设计规格要求,所有内部成分是否按规定正常进行。主要用于软件验证。白盒测试的主要方法有逻辑覆盖、基本路径测试等。

白盒测试要求全面了解程序内部逻辑结构和处理过程,以检查处理过程的细节为基础,要求对程序的结构特性做到一定程度的覆盖,对所有逻辑路径进行测试,并检验内部控制结构和数据结构是否有错,实际的运行状态与预期的状态是否一致。白盒测试法是穷举路径测试,但贯穿程序的独立路径数可能是一个天文数字,所以也不可能进行穷举测试。企图遍历所有的路径是很难做到的,即使每条路径都测试了,覆盖率达到 100%,程序仍可能出错。

- 穷举路径测试绝不能查出程序违反了设计规范,即程序在实现一个不是用户需要的功能。
- 穷举路径测试不可能查出程序中因遗漏路径而出错。
- 穷举路径测试可能发现不了一些与数据相关的错误。

白盒法是“基于覆盖的测试”,应朝着提高覆盖率的方向努力,尽可能多地进行测试,找出那些被忽视的错误。一般来说,白盒测试的原则是:

- 保证每个模块中所有独立路径至少被使用一次。
- 对所有逻辑值均测试为真值 (true) 和假值 (false)。
- 在上下边界及可操作范围内运行所有循环。
- 检查内部数据结构以确保其有效性。

综上所述,白盒测试用例的常见设计方法有逻辑覆盖、循环覆盖和基本路径测试。逻辑覆盖又可分为语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。这些内容会在第 5 章、第 14 章详细介绍。

### 2.3.3 静态的和动态的方法

根据程序是否运行,测试可以分为静态测试和动态测试。静态测试就是静态分析,对模块的源代码进行研读,查找错误或收集一些度量数据,并不需要对代码进行编译和仿真运行。静态测试采用人工检测和计算机辅助静态分析手段进行检测,只进行特性分析。

- 人工检测:人工检测是指不依靠计算机而完全靠人工审查或评审软件。人工检测

偏重于编码风格、质量的检验，除了审查编码还要对各阶段的软件产品进行检验。这种方法可以有效地发现逻辑设计和编码错误，发现计算机不易发现的问题。

- 计算机辅助静态分析：利用静态分析工具对被测程序进行特性分析，从程序中提取一些信息，以便检查程序逻辑的各种缺陷和可疑的程序构造。如用错的局部变量和全局变量，不匹配的参数，潜在的死循环等。静态分析中还可以用符号代替数值求得程序结果，以便对程序进行运算规律的检验。

静态测试包括对软件产品的设计规格说明书的审查，对程序代码的阅读、审查等。静态分析的查错和分析功能是目前方法所不能替代的，已被当做一种自动化的代码校验方法。

动态测试是通过观察代码运行时的动作，来提供执行跟踪、时间分析，以及测试覆盖度方面的信息。动态测试通过真正运行程序发现错误。通过有效的测试用例，对应的输入/输出关系来分析被测程序的运行情况。

不同的测试方法各自的目标和侧重点不一样，在实际工作中，应将这两种方法结合起来运用，以达到更完美的效果。

以上的测试方法各有所长，每种方法都可设计出一组有用的例子，用这组测试用例可以比较容易地发现某种类型的错误，却不易发现另一种类型的错误。因此在实际测试中，应结合各种测试方法，形成综合策略。在单元测试主要用白盒测试；在系统测试时主要用黑盒测试，或者以黑盒测试为主要测试方法，白盒测试为辅助方法等。

## 2.3.4 验证和确认 (verification & validation)

软件测试不仅要检查程序是否出错，程序是否和软件产品的设计规格说明书一致，而且还要检验所实现的正确功能是否就是客户或用户所需要的功能，这就引出软件测试中有名的 V&V。V&V，即英文两个单词 verification 和 validation 的第一个字母组合。

### 1. 验证

Verification，一般书上将它翻译为“验证”，但也可以翻译为“检验”，即检验软件是否已正确地实现了产品规格说明书所定义的系统功能和特性。验证过程提供证据表明软件相关产品与所有生命周期活动的要求（如正确性、完整性、一致性和准确性等）相一致。

验证是否满足生命周期过程中的标准、实践和约定，验证为判断每一个生命周期活动是否已经完成，以及是否可以启动其他生命周期活动建立一个基准。

### 2. 有效性确认

Validation，一般书上将它翻译为“确认”，但更准确地翻译应该是“有效性确认”。这种有效性确认要求更高，要能保证所生产的软件可追溯到用户需求的一系列活动。确认过程提供证据表明软件是否满足系统需求（指分配给软件的系统需求），并解决了相应问题。

### 3. 两者的区别

为了更好地理解这两个单词的区别，可以概括地说，验证是检验开发出来的软件产品

和设计规范说明书的一致性，即是否满足软件厂商的生产要求。但设计规范说明书本身就可能存在错误，所以即使软件产品中某个功能实现的结果和设计规范说明书完全一致，但可能并不是用户所需要的，因为一开始，设计规范说明书已经理解错了用户在这一功能上的需求，所以仅仅进行验证测试还是不充分的，还要进行确认测试。确认就是检验产品功能的有效性，即是否满足用户的真正需求。

所以，BOEHM 是对 V&V 最著名又最简单的解释。

- **Verification:** Are we building the product right? 是否正确地构造了软件？即是否正确地做事，验证开发过程是否遵守已定义好的内容。
- **Validation:** Are we building the right product? 是否构造了正是用户所需要的软件？即是否正在做正确的事。

### 2.3.5 ALAC 测试

ALAC，是 act-like-a-customer（像客户那样做）的简写。ALAC 测试方法是一种基于客户使用产品的知识开发出来的测试方法。它的出发点是著名的 Pareto 80/20 规律。用到软件测试中，可以概括为两条：

- 一个软件产品或系统中全部功能的 20% 是常用的功能，用户的 80% 时间都在使用这 20% 功能，而软件产品或系统中剩下的 80% 不是常用的功能，用户使用得比较少，只有 20% 时间在使用剩下的 80% 功能。
- 测试发现的所有错误中的 80% 很可能集中在 20% 的程序模块中，另外 20% 的错误很可能集中在 80% 的程序模块中。

ALAC 测试就是基于这样一个思想，并考虑复杂的软件产品肯定存在许多错误，而测试的时间有限，然后像客户那样做，对常用的功能进行测试。ALAC 测试方法适合一些特别的场合，如产品只是一个演示版，开发预算很低，没有足够时间进行测试，整个开发计划日程表很紧。其最大的益处是降低测试成本，缩短测试时间，缺陷查找和改正将针对那些客户最容易遇到的错误，如图 2-6 所示。

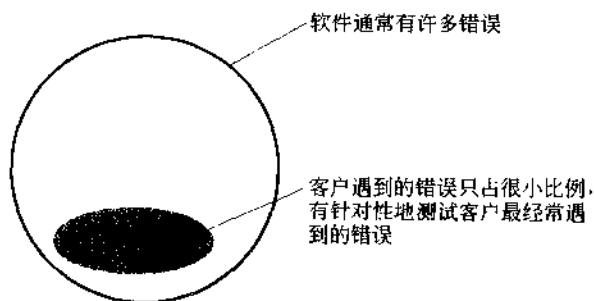


图 2-6 ALAC 测试方法的原理示意图

### 2.3.6 自动化测试和随机测试

前面已经介绍了软件测试的一些基本方法,如白盒测试、黑盒测试、静态测试和动态测试等方法,这些方法既可以通过手工执行,也可以通过一些软件工具进行。通过工具自动执行软件的测试,一般称为软件自动化测试。

软件测试的大部分工作可以由测试人员开发出来的系统或软件工具来进行,尤其是有些无法通过手工测试来完成的测试任务,比如一些负载测试、性能测试、可靠性测试就适合使用自动化测试方法。如模拟1万个客户访问某个网站,我们不可能安排1万个测试人员在1万台计算机上进行操作,只有借助能在一台计算机上模拟几百个客户的软件工具去实现,这时只要在几十台机器上同时运行这个工具,并通过另外一个监控管理工具来协调、控制这些机器的运行,并将结果记录下来,用于测试结果的分析,就可以获取所需要的数据。

除了通过测试工具来执行实际的测试,也可以通过一些软件系统来管理测试过程,这也可以看作测试自动化的一部分。

在自动化测试中,一种是按测试用例一步一步来执行测试,另外一种是完全模拟客户进行随意的操作。后一种测试称为随机测试(random test),或称为猴子测试法(monkey test)。

随机测试来源于一个概率统计的思想,或者可以看作是一个有趣想法的延伸:如果让100万只猴子在100万只键盘上敲上100万年,它们最终可能会写成一部莎士比亚的巨著。因为当一个新软件发布时,成千上万的用户会使用它,其中许多客户会故意考验它——乱敲乱试。但通过有限的测试用例很难覆盖各种各样的情况,所以有必要设计测试工具,模拟客户操作的随意性,进行大量的、自动化的随机测试,来发现今后用户可能会碰到的问题。

### 2.3.7 软件测试的误区

我国软件业主要集中在最终客户软件项目的开发上,通用软件商品的开发比较少,所以我国的软件开发水平和软件业发达国家(美国、印度、爱尔兰等)相比,有一定的距离,主要体现在软件测试和质量保证的水平。

随着我国加入WTO之后,软件测试越来越受到人们的注意。人们认识到,要提高软件质量,软件测试是极其重要的一个环节。但是,相对于软件开发而言,软件测试还不为众人所了解。很多软件开发人员,包括多数软件企业的高层管理人员,由于缺乏软件测试的知识和实践经验,对软件测试的认识还有很多误区,这对软件测试工作极为不利,必须加以澄清,才可能提高软件产品的质量。

**误区一:**如果发布出去的软件有质量问题,那是软件测试人员的错

软件测试是一种有效提高软件质量的手段,但即使在投入上有所保证,测试也并不能百分之百地发现所有质量隐患。况且,软件的高质量不是靠测试测出来的,而是软件开发过程中每一个环节都要有质量意识,做好检查、审查等工作,才能保证质量。任何一个环节出问题,都有可能带来质量的隐患。



**误区二：**软件测试技术要求不高，至少比编程容易多了

很多人认为软件测试就是运行一下软件，然后看看结果对不对。实际上，软件测试不仅仅只是运行或操作软件，还要涉及测试环境的建立、测试用例的设计等技术问题。当采用白盒测试方法时，需要良好的编程经验；在进行自动化测试时，需要写测试脚本，也需要良好的编程经验。如何在有限的资金投入下，提高软件测试的效率和保证产品的高质量是一件较困难的事情。所以，一位优秀的软件测试人员不仅要掌握各种测试技术，还要具备丰富的编程经验和对 bug 的敏感，从一定意义上说，对测试人员的要求比编程人员的要求还要高。

**误区三：**有时间就多测试一些，来不及就少测试一些

软件测试并不是可有可无的，测多少、怎样测也不是随心所欲的。规范化的软件开发过程需要对软件测试早做计划，分配必要的时间、人力和财力等资源，并将其作为项目管理的一个重要部分，进行跟踪、控制和协调。

**误区四：**软件测试是测试人员的事，与开发人员无关

为了减小相互的影响，一般要求开发和测试相对独立，但这只是分工上的不同。编程和测试是软件项目相辅相成的两个过程，人员之间的交流、协作和配合是提高整体开发效率的重要因素。而且在实际操作中也会有一些测试，比如单元测试，会由测试人员提供测试用例，由开发人员运行，或全部由编程人员完成。

**误区五：**根据软件开发瀑布模型，软件测试是开发后期的一个阶段

不少人根据软件开发瀑布模型，容易得出这样一种结论：“程序代码写完之后再进行测试”，这是错误的，这种错误的观念会给测试工作带来很多问题。

生命周期中的“测试阶段”表明在该阶段主要工作是测试，即到了“测试阶段”，测试的主要任务是执行测试、运行测试脚本、测试结果分析和递交测试报告，而测试的大量准备工作，诸如测试计划、测试用例设计以及测试脚本编写等，实际是在还没有开始写程序之前就开始了，可以说项目什么时候开始，测试工作也就什么时候开始。如果到“测试阶段”才开始启动测试工作，那为时已晚，没有计划，没有测试用例，测试也就很容易是“走过场”。

即使程序测试，也不是要等到软件程序全部完成后才开始运行。针对每个程序单元、模块，可以进行单元测试；把程序单元或模块进行集成时，需要进行相应的集成测试；在进行集成测试和系统测试之前，就要准备测试环境。所以，编程与测试几乎是同步进行的。

如 2.2.4 小节所述，软件产品中的质量问题发现越晚，修正错误的代价就越大。就这一点而言，也要求软件测试尽早开始，包括需求分析和设计文档的审查。

在实践中，除对软件测试不够重视之外，还可能存在一些对软件测试本身不正确的看法：

- 认为测试工作不如设计和编码那样容易取得进展，难以给测试人员带来某种成就感。
- 以发现软件错误为目标的测试是非建设性的，甚至是破坏性的，测试中发现错误是对责任人工作的一种否定。
- 测试工作枯燥无味，不能引起人们的兴趣。
- 测试工作是艰苦而细致的工作。
- 对自己编写的程序盲目自信，在发现错误后，顾虑别人对自己开发能力的看法。

## 2.4 软件测试的分类和阶段

为了帮助读者学习本书第2部分、第3部分的内容，在本节和2.5节，将介绍软件测试的分类、阶段划分和其各项工作内容，从而呈现一个涉及软件测试所有内容的整体架构，使读者能抓住其内涵和外延。

软件测试的方法和技术多种多样。对于软件测试技术，可以从不同的角度加以分类：

- 从是否需要执行被测软件的角度，可分为静态测试和动态测试。
- 从测试是否针对系统的内部结构和具体实现算法的角度可分为白盒测试和黑盒测试。
- 按测试的对象（Web、Client、Server、Database 等）进行分类，涉及面向开发的单元测试、GUI 和捕获/回放测试、基于 Web 应用的测试、C/C++/Java 应用测试、负载和性能测试、数据库测试、软件测试和 QA 管理等各类工具测试。
- 其他测试方法，如回归测试、压力测试、恢复测试、安全测试和兼容性测试等。

也可以从另外三个角度来划分测试阶段：面向测试操作类型的阶段划分、面向测试操作对象的阶段划分、面向测试实施者的阶段划分。

- 测试操作类型包括调试、集成、确认、验证、组装、验收、操作等。
- 测试操作对象可以是单元、部件、配置项、子系统、系统等。
- 测试实施者可以是开发者、测试者、使用者、验收者等。

各类标准从不同角度定义测试评审阶段，而测试组织者可以在符合所选标准的同时，结合多个划分因素规定本系统的测试阶段。

### 2.4.1 测试的分类

软件测试可以分别按测试范围、测试目的、测试对象、测试过程分类。

#### 1. 按测试范围分类

- 单元测试（unit testing）
- 组件测试（component testing）
- 集成测试（integration testing, string testing）
- 系统测试（system testing）
- 验收测试（acceptance testing, beta test）
- 安装测试（installation testing）

#### 2. 按测试目的分类

- ▶ 正确性测试（correctness testing）
  - 白盒测试（white-box）

- 黑盒测试 (black-box)
- ▶ 性能测试 (performance testing)
- ▶ 可靠性测试 (reliability testing)
  - 强壮性测试 (robustness, strong testing)
  - 异常处理测试 (exception handling testing)
  - 负载测试 (stress, load testing)
- ▶ 安全性测试 (security testing)

### 3. 按测试对象分类

- 单元测试 (unit testing)
- 组件测试 (component testing)
- 模块测试 (module testing)
- 程序测试 (program testing)
- 系统测试 (system testing)
- 文档测试 (documentation testing)

### 4. 按测试过程分类

- 需求阶段的测试 (requirements phase testing)
- 设计阶段的测试 (design phase testing)
- 程序阶段的测试 (program phase testing)
- 测试结果的评估 (evaluating test results)
- 安装测试 (installation phase testing)
- 验收测试 (acceptance testing)
- 测试变化: 维护 (testing changes: maintenance)

### 5. 其他测试技术和方法

- 回归测试 (regression testing): 为保证软件中新的变化 (修改) 不会对原有功能的正常使用有影响而进行的测试。也就是说, 已经满足用户需求的功能不应该出现任何问题。每当软件缺陷被修改之后、或者对原有功能进行一些调整和加强, 或者是在原有版本基础上增加新功能的时候, 都会采用回归测试方法, 特别是到了开发周期的最后阶段。回归测试, 更适于使用软件自动化测试工具。
- 压力测试 (stress testing): 用来检查系统在大负荷条件下的运行情况。在非正常的巨大负荷下, 某些动作和输入大量重复, 输入大数, 对数据库进行非常复杂的查询等 (如测试一个网站在不同负荷情况下的状况), 以确定在什么情况下系统响应速度下降或是出现故障。也称为性能测试 (performance testing)。
- 恢复测试 (recovery testing): 在系统崩溃、硬件故障, 或者其他灾难发生之后, 重新恢复系统和数据的能力测试。
- 安全测试 (security testing): 测试系统在应付非授权的内部/外部访问、故意损坏

时的系统防护能力。

- 兼容性测试 (compatibility testing): 测试在特殊的硬件/软件/操作系统/网络环境下的软件表现。

## 2.4.2 测试的阶段

软件测试是软件开发过程中的重要内容之一,是软件质量保证的关键。软件测试贯穿软件产品开发的整个生命周期——软件项目一开始,软件测试也就开始了,从产品的需求分析审查到最后的验收测试、安装测试结束。整个过程如图 2-7 所示。

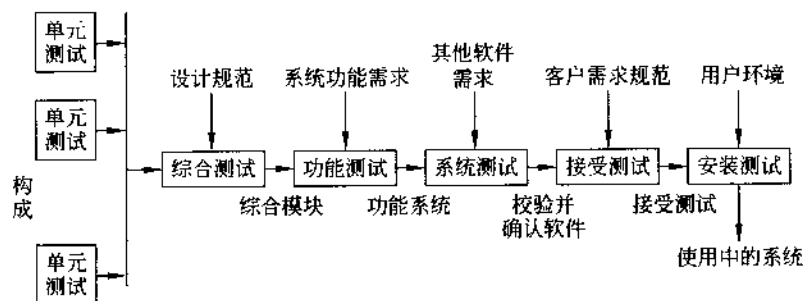


图 2-7 软件测试阶段示意图

从过程来看,软件测试是由一系列的不同测试阶段所组成的,这些阶段分为:规格说明书审查、系统和程序设计审查、单元测试、集成测试(组装测试)、功能测试、确认测试、系统测试、验收测试和安装测试。软件开发的过程是自顶向下的,测试则正好相反,以上这些过程就是自底向上,逐步集成的。当然,这里阐述的测试过程是一个完整的过程,对于不同的软件系统或产品可以进行适当的裁减或合并,如功能测试和确认测试可以合并为确认测试,验收测试和安装测试可以合并为验收测试。图 2-8 是测试流程的另一种表示法。

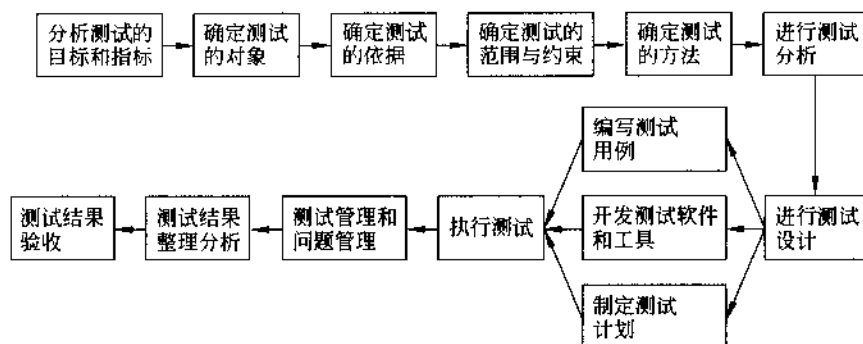


图 2-8 测试流程的另一种表示法

### 1. 规格说明书审查

需求分析规格说明书是否完整、正确、清晰是软件开发成败的关键。为了保证需求定义的质量,应对其进行严格的审查。

测试人员要参与系统或产品需求分析,认真阅读有关用户需求分析文档,真正理解客户的需求,检查规格说明书对产品描述的准确性、一致性等,为今后熟悉应用系统、编写测试计划、设计测试用例等做好准备工作。

### 2. 系统和程序设计审查

软件设计是基于对用户需求的理解基础上,借助计算机技术,将客户的需求转换成计算机软件表示的过程,其设计的结果能描述出系统结构和逻辑、数据输入、详细处理过程、数据存储模式、数据输出等。例如,可以按照需求规格说明书对系统结构的合理性、处理过程的正确性进行评价,同时利用关系数据库的规范化理论对数据库模式进行审查。

代码会审是一种静态的白盒子测试方法,是由一组人通过阅读、讨论来审查程序结构、代码风格、算法等的过程。会审小组由组长、3~5名程序设计人员、编程人员和测试人员组成。会审小组在充分阅读待审程序文本、控制流程图及有关要求、规范等文件基础上,召开代码会审会,程序员逐句讲解程序的逻辑并回答其他人员提出的问题,对有争议的问题进行讨论,以达成一致意见或得到解决方案。实践表明,代码会审做得好的话可以发现大部分程序缺陷,甚至程序员在自己讲解过程中就能发现不少代码错误,而讨论可能进一步促使问题暴露。例如,对某个全局变量的默认值改变或某个参数变量选项改变的讨论,可能发现与之有关的,甚至能涉及到模块间接口和系统结构参数的大问题,从而进行程序结构的调整、参数的优化,最终改善软件的质量。

### 3. 单元测试

单元测试的对象是程序系统中的最小单元——模块或组件,在编码阶段进行,针对每个模块进行测试,主要使用白盒测试方法,从程序的内部结构出发设计测试用例,检查程序模块或组件已实现的功能与定义的功能是否一致,以及编码中是否存在错误。多个模块可以平行地、对立地测试,通常要编写驱动模块和桩模块。

由于模块规模小,功能单一,逻辑简单,测试人员有可能通过模块说明书和源程序清楚地了解该模块的 I/O 条件和模块的逻辑结构。采用结构测试(白盒法)的用例,尽可能达到彻底测试,然后辅以功能测试(黑盒法)的用例,使之对任何合理和不合理的输入都能鉴别和响应。高可靠性的模块是组成可靠系统的坚实基础。

单元测试是测试执行的开始阶段,而且与程序设计和实现有非常紧密的关系,所以单元测试一般由编程人员和测试人员共同完成,编程人员有时起主要作用。在单元测试中,除了上述的 I/O 条件、程序逻辑结构、程序路径等实际测试手段之外,还会采取其他辅助手段,如代码走读(code review)、静态分析(static analysis)和动态分析(dynamic analysis)等。

#### 4. 集成测试

集成测试,也称组装测试、联合测试、子系统测试,在单元测试的基础上,将模块按照设计要求组装起来同时进行测试,主要目的是发现与接口有关的模块之间的问题。如数据穿过模块接口时可能丢失;一个模块与另一个模块可能有由于疏忽的问题而造成有害影响;把子功能组合起来可能不产生预期的主功能;单个模块的误差累积起来,是否会放大,从而达不到能接受的程度;全程数据结构是否有错误等。

选择什么样的方式把模块组装起来形成一个可运行的系统,直接影响到测试成本、测试计划、测试用例的设计、测试工具的选择等。通常有两种集成方式:一次性集成方式和增殖式集成方式。

- 一次性集成方式:也称整体组装。首先对每个模块分别进行模块测试,然后再把所有模块组装在一起进行测试,最终得到要求的软件系统。
- 增殖式集成方式:也称渐增式组装。首先对一个个模块进行单元测试,然后将这些模块逐步组装成较大的系统。在组装的过程中,一边连接一边测试,以发现连接过程中产生的问题,最后通过增殖逐步组装成为要求的软件系统。

集成测试应提交集成测试计划、集成测试规格说明和集成测试分析报告。

#### 5. 功能测试

功能测试一般须在完成集成测试后进行,而且是针对应用系统进行测试。功能测试是基于产品功能说明书,是在已知产品所应具有的功能,从用户角度来进行功能验证,以确认每个功能是否都能正常使用。在测试时,不考虑程序内部结构和实现方式,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。功能测试包括用户界面测试、各种操作测试(菜单、按钮等)、不同的数据输入、不同的逻辑思路、数据输出和存储等。

#### 6. 确认测试

确认测试的目的是向未来的用户表明系统能够按照预定要求那样工作。经集成测试后,已经按照设计把所有的模块组装成一个完整的软件系统,接口错误也已经基本排除了,接着就应该进一步验证软件的有效性,这就是确认测试的任务,即软件的功能和性能达到用户合理的期待。

确认测试也称有效性测试,验证软件的功能和性能及其他特性是否与用户的要求一致。如果加入用户信息,也称验收测试。基于需求规格说明书和用户信息,验证软件的功能和性能及其他特性。确认测试步骤如下:

(1) 进行有效性测试。有效性测试是在模拟的环境(可能就是开发环境)下,应用黑盒测试的方法,验证所测试软件是否满足需求规格说明书列出的需求。因此,需要制定测试计划、测试步骤、测试种类、测试用例。

(2) 软件配置复查。

## 7. 系统测试

软件开发完成以后，最终还要与系统中其他部分配套运行，进行系统测试。系统测试是将软件放在整个计算机环境下，包括软硬件平台、某些支持软件、数据和人员等，在实际运行环境下进行一系列的测试，包括恢复测试、安全测试、强度测试和性能测试等。

经过上述的测试过程对软件进行测试后，软件基本满足开发的要求，测试宣告结束，经验收后，将软件提交用户。

## 8. 验收测试

验收测试的目的是向未来的用户表明系统能够像预定要求那样工作。经集成测试后，已经按照设计把所有的模块组装成一个完整的软件系统，接口错误也已经基本排除了，接着就应该进一步验证软件的有效性，这就是验收测试的任务，即软件的功能和性能如同用户所合理期待的那样。

## 9. 安装测试

安装测试是指按照软件产品安装手册或相应的文档，在一个和用户使用该产品完全一样的环境中或相当于用户使用的环境中，进行一步一步安装的同时所做的测试。安装测试主要进行以下三个方面的测试：

- 环境的不同设置或配置。强调用户的使用环境，考虑各种环境因素的影响，如在一个完全崭新、非常干净的操作系统或应用系统上进行某个产品的安装，或者是考虑各种硬件接口的要求。
- 安装文档的准确性。进行安装测试时，必须一步一步地完全按照文档去做，不能下意识地使用自己已有的经验或知识。
- 安装的媒体制作是否有问题，包括最后制作时可能会丢了一个文件，或感染上计算机病毒等。

安装测试有时容易被忽略，如果没做好，可能造成的损失依然很大，如必须换回全部安装盘或重印安装手册，技术支持负担很大，所以安装测试也是重要的一个测试阶段。

表 2-1 总结了各测试阶段的输入和输出标准。

表 2-1 各测试阶段输入和输出标准

阶 段	输 入	输 出
需求分析	需求定义、市场分析文档、相关技术文档	市场需求分析会议纪要、功能设计、技术设计
设计审查	市场需求文档、技术设计文档	测试计划、测试用例
功能验证	代码完成文件包、功能详细设计说明书、最终技术文档	完整测试用例、完备的测试计划、缺陷报告、功能验证测试报告
系统测试	代码修改后的文件包、完整测试用例、完备的测试计划	缺陷报告、缺陷状态报告、项目阶段报告
确认测试	代码冻结文件包、确认测试用例	缺陷状态报告、缺陷报告审查、版本审查
版本发布	代码发布文件包、测试计划检查清单	当前版本已知问题的清单、版本发布报告

## 2.5 软件测试的工作范畴

了解软件测试各个阶段任务之后,就要开始通过一系列的测试活动来完成各个测试阶段所规定的任务。一般来讲,由一位对整个系统设计熟悉的设计人员编写测试大纲,明确测试的内容和测试通过的准则,设计完整合理的测试用例,以便系统实现后进行全面测试。虽然每个阶段具体的任务和要求是不一样的,但软件测试工作的基本范畴是一样的,软件测试工作范畴,可以分为两个层次。

- 软件测试工作的组织与管理:制定测试策略、测试计划,确认所采用的测试方法与规范,控制测试进度,管理测试资源。
- 测试工作的实施:编制符合标准的测试文档,研制测试环境,与开发组织协作实现各阶段的测试活动。

软件测试工作可以分为六个方面。

- 测试组织和管理:建立测试队伍,设立不同功能或完成不同任务的测试小组,对测试用例、软件缺陷、测试执行、测试文档等进行管理,当然,也可以把测试管理工作看成是软件质量管理工作的一部分。
- 测试计划:独立的测试组织负责定义软件测试的方法与规范。开发组织负责编制单元测试的计划和说明。测试组织主要负责编制其他各测试阶段的测试计划和说明。
- 设计测试用例:为了更有效地进行测试,需要设计测试用例。
- 测试实施:按测试计划与测试说明的定义对测试对象进行相应的测试,填写测试报告中相应的表格。
- 测试结果分析:对测试结果进行定量和定性的分析,以检查测试工作执行的状态。
- 测试评审与报告:依据软件测试评审准则在各测试阶段评审时提交类型完整的测试文档。

对于测试组织和管理,专门有一章进行详细的论述。在此,对另外五项工作内容进行简单介绍,有利于阅读第2部分的内容。

### 2.5.1 测试计划制定

无论做什么工作,都是计划先行,然后按照所制定的有效计划去执行、跟踪和控制。这里有必要介绍一个著名的PDCA模型。

PDCA代表计划(plan)、实施(do)、检查(check)和措施(action),其模型如图2-9所示。

- 计划就是设定可以达到的目标,决定哪些事需要完成,哪些事可以做,哪些资源可以利用。
- 实施(do):有了计划后,接着就是要去做,以执行计划中已定义的各种任务。



- **检查 (check):** 对所做的结果进行检查, 以确认是否达到预期的结果。做了但不检查结果, 就不知道是否按事先制定的计划去执行了。通过检查发现计划或执行中的问题。
- **措施 (action):** 如果检查发现了计划或执行中的问题, 就需要采取措施纠正错误, 从而在制定下一步计划中有所改善, 制定的计划会更切合实际、更合理。

PDCA 循环

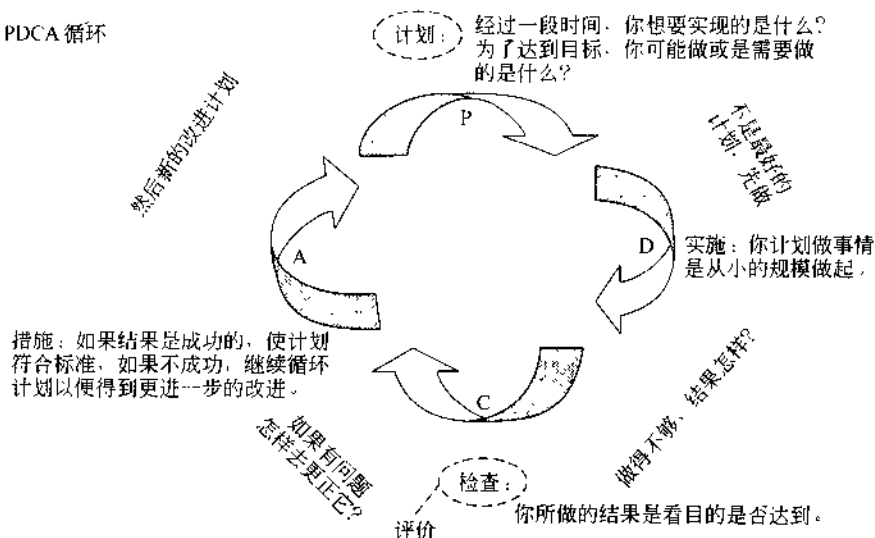


图 2-9 软件开发流程改进模型 PDCA 示意图

软件测试也一样，先要制定测试计划。软件测试计划是做好软件测试工作的前提。所以在进行实际测试之前，应制定良好的、切实可行的测试计划并严格执行，特别要确定测试策略和测试目标。

测试规划与软件开发活动同步进行。在需求分析阶段，要完成验收测试计划，并与需求规格说明一起提交评审。同样，在概要设计阶段，要完成和评审系统测试计划。在详细设计阶段，要完成和评审集成测试计划。在编码实现阶段，要完成和评审单元测试计划。对于测试计划的修订部分，需要进行重新评审。

在测试计划中，明确要完成的测试活动，评估完成活动所需要的时间和资源，设计测试组织和岗位职权，进行活动安排和资源分配，安排跟踪和控制测试过程的活动。在测试计划中，主要内容包括制定测试策略、确定测试范围、测试用例的设计方法和要点、所需资源和日程安排。

### 1. 制定测试策略

制定测试策略主要分析测试的目标和指标、确定测试的对象和依据，明确测试的重点和所采用的方法。

- 全面细致地了解产品的项目信息：应用领域、测试范围、市场需求、产品的特点和主要功能、技术架构。

- 基于模块、功能、整体、系统、版本、压力、性能、配置和安装等各个因素对产品的影响，公正客观地开展测试计划。
- 根据程序的重要性和一旦发生故障将造成的损失，来确定它的测试等级和测试重点。
- 认真研究测试策略，以便能使用尽可能少的有效测试用例，发现尽可能多的程序错误，因为一次完整的软件测试过后，如果程序中遗漏的错误过多并且很严重，则表明本次测试是失败的；而测试失败意味着让用户承担隐藏错误带来的危险。同时反过来说，如果过度测试，则又会浪费许多宝贵的资源。需要在这两点上进行权衡，找到一个最佳平衡点。

## 2. 确定测试范围

测试主要依据“产品规格说明书”，根据产品的新特性或功能修改需求所做的代码变化，以及这种变化可能引起的问题或给其他模块造成的影响。一般在确定测试范围时，主要考虑的因素有：

- 优先级最高的需求功能。
- 新功能和编码改动较大（提高性能表现）的旧功能。
- 运用有效的测试技术提高测试效果。
- 经常出现问题部分的功能。
- 经常被用户使用的功能和配置。

## 3. 所需资源和日程安排

犹如一般工程计划那样，进度和资源安排是测试计划的重要组成部分。

资源包括计算机硬件、软件和人力资源，硬件和软件虽然需要事先安排预算，但准备起来相对人力资源还是容易一些，购买之后进行安装、配置就可以使用。而人力资源如果是新招聘的，需要一个较长的时间进行培训，而且需要一个熟悉产品的过程。在做人力资源计划时必须考虑到这些，同时，还要考虑每个人的技术特长、能力、性格、工作风格等，了解这些有利于建立一支成熟、和谐的队伍。

在进行资源安排时，更重要的是设置不同的角色，包括项目经理和组长、系统工程师、测试设计工程师、资深测试工程师、一般测试工程师等，清楚地定义每个人的工作内容和责任。

对于日程安排，主要是设置整个测试周期的项目进程，有产品规格说明书审查结束时间、测试计划初稿完成时间、设计测试用例的时间、完成测试计划表、单元测试完成的日期、首次集成测试的日期、系统测试完成的日期、测试全部完成的日期等。

为了合理、准确地安排日程，对测试工作量要进行正确的估计。除了对工作量的估计之外，还要对参与该项目人员的工作能力进行正确评估。由于涉及到不同的项目、不同的测试人员、不同的前期介入方式，要对每人每天能够完成的平均测试用例数目做出准确的估计确实很困难，但是可以根据以前一些项目测试的经验或历史积累下来的数据进行判断，

然后适当增加 5%~20% 的余量，就基本可以了。

## 2.5.2 设计测试用例

测试计划完成之后，接下来就是根据系统规格说明书、系统设计文档、测试范围、技术特点、程序结构等来设计测试用例。不同测试阶段，可以有不同的测试计划。同样，不同的测试阶段也有不同的设计测试用例的方法。在单元测试阶段，主要用白盒测试方法设计测试用例；在功能测试阶段，主要用黑盒测试方法来设计测试用例。

测试用例是按一定顺序执行的与测试目标相关的测试活动的描述，是确定“怎样”测试。测试用例被看作是有效发现软件缺陷的最小测试执行单元，也被视为软件的测试规格说明书。测试用例的设计是整个软件测试工作的核心。测试用例反映对被测对象的质量要求，决定对测试对象的质量评估。

测试用例定义了为执行测试所需要的条件或环境、输入或操作步骤，以及所期望的结果。

- 测试环境是测试的基础。要尽量模拟软件系统实际应用的环境。
- 输入值。除了正常的输入值，关键是寻找哪些是属于边界条件的输入值和正常输入值。
- 期望结果或标准（criteria）。是根据系统设计规格说明书来确定的输出结果、标准，有时是由经验做出正确判断、理解所确定。

测试用例不是每个人都可以编写的，测试用例的设计者需要对产品的设计、功能规格说明书、用户场景以及程序/模块的结构都有比较透彻的了解。没有经验的测试人员一开始只能执行别人写好的测试用例，随着项目的进度以及测试人员的成熟，他们才逐渐可以参与测试用例的设计。

测试用例具有以下特点。

- 可复用性：良好的测试用例具有重复使用的功能，保证测试的稳定性。
- 有效性：测试是不可能进行穷举测试的，但良好的测试用例将大大节约时间，提高测试效率。
- 可管理性：从测试的项目管理角度来说，测试用例的通过率是检验代码质量的保证。
- 继承性：测试所积累的经验可以通过测试用例传递下去。
- 基础性：是初级测试人员和自动化测试的基础。初级测试人员按照测试用例执行测试，达到良好的效果。

## 2.5.3 执行测试

有了测试用例，就可以执行测试，即按照用例的要求进行人工操作或用测试脚本结合测试工具进行，多数情况下要运行程序，将获得的运行结果与预期结果进行比较和分析，并在测试过程中随时记录、跟踪和管理软件缺陷，最终得到测试报告。

执行测试主要有下列一些活动。

- 建立必要的测试环境。
- 按照所写的测试用例，编写测试脚本。
- 根据测试的对象和目的，构造测试用例的集合（test suite）。
- 运行测试脚本或手工按测试用例进行。
- 记录测试结果。
- 结果比较分析，找出软件缺陷。
- 将软件缺陷记录到缺陷数据库中，清楚地描述该缺陷。
- 跟踪和管理软件缺陷。
- 验证被处理的软件缺陷，并进行回归测试。
- 对测试过程进行管理，保证测试工作执行的正确性，实现资源调拨和相关合作方的协调。对测试中的问题进行全程追踪。

## 2.5.4 测试结果分析和质量报告

如同代码是程序员的成果之一，测试报告是测试人员的主要成果之一。一个好的测试报告建立在测试结果的基础之上，不仅要提供必要测试结果的实际数据，同时要对结果进行分析，发现产品中问题的本质，对产品质量进行准确的评估。

分析的对象和内容：测试的覆盖率、缺陷分析、产品总体质量分析、过程分析等。

### 1. 测试的覆盖率

- 语句覆盖率：检测在软件测试时代码语句执行覆盖率。
- 分支覆盖率：用于分析被测软件在进行软件测试时分支的执行情况。
- 子程序调用覆盖率：判断某一程序是否调用了所有应该调用的子程序，或判断所有的子程序是否被调用过。此指标在系统集成测试时很有用。
- 数据值覆盖率：检测程序中变量在测试时是否包含了所有可能值。
- 面向对象覆盖率：多态类的覆盖、模式化的覆盖、继承的覆盖。类的状态决定它的行为，需要确认每一个对象独立状态的代码覆盖率，或测试每一个类或子系统独立线程的覆盖率。例如，通信协议类有很多状态：初始化状态、正在连接状态、已连接状态和出错状态等。
- MC/DC 代码覆盖率：支持 RTCA DO-178B 标准。

### 2. bug 分析

- bug 分布：在程序模块的横向分布，在时间上的纵向分布。
- 测试的效率：根据丢失的 bug 数目和发现的总 bug 数，可以了解测试的效率。也可以根据执行的总测试用例数，计算出每发现一个 bug 所需要的测试用例数、测试时间等，对不同阶段、不同模块、不同人员等进行对比分析。
- 程序的质量：通过对每千行代码所含的 bug 数分析，了解程序代码质量。

- 开发解决 bug 的能力或状态。

### 3. 产品总体质量分析

传统的软件测试，只针对软件产品开展，找到缺陷之后再加以改正和修补，这是一种“亡羊补牢”的质量管理方式。而针对开发全过程所开展的软件测试和过程度量，则注重事先分析，通过对已发生的数据对比、统计、时间序列等分析，来判断软件产品质量的未来趋势，并提前予以控制和预防，属于一种“防患于未然”的质量管理方式。与传统的软件测试相比，全过程测试管理方式不仅可以有效降低产品的质量风险，而且还可以提前对软件产品缺陷进行规避，这不仅缩短了对缺陷的反馈周期和整个项目的开发周期，而且也会在较大程度上降低软件产品开发用在修正软件缺陷时所支付的成本。

对测试的结果进行整理、归纳和分析，一般借助于 Excel 文件、数据库和一些直方图、圆饼图、趋势图来进行分析和表示。主要的方法有对比分析、根本原因（root cause）查找、问题分类、趋势（时间序列）分析和其他统计分析等。

- 对比分析：用软件执行测试结果与标准输出的对比工作，因为可能有部分的输出内容是不能直接对比的（比如，对运行的时间的记录，对运行的路径记录，以及测试对象的版本数据等），就要用程序进行处理。
- 根本原因查找：“分析”是找出不吻合的地方并指出错误的可能起因。
- 问题分类：“分类”包括各种统计上的分项，例如，对应的源程序位置，错误的严重级别（提示、警告、非失效性错误、失效性错误或别的分类方法），新发现的还是已有记录的错误。
- 趋势（时间序列）分析：根据所发现的软件缺陷历史数据进行分析，预测未来情况。
- 其他统计分析：通过对缺陷进行分类，然后利用一些成熟的统计方法对已有数据进行分析。因为已了解程序开发中主要问题或产生问题的主要原因，从而比较容易提高软件质量。

## 小结

软件测试是软件质量保证的手段，软件测试基于两个最基本的概念来展开，这两个基本概念就是质量和客户。软件质量就是客户的满意度，而测试就是时时刻刻从客户的角度出发，来保证软件产品满足客户的实际需求。软件质量的内涵不仅在于软件产品的质量上，而且还包括软件开发过程的流程质量，以及软件产品所要适应的业务环境质量。

软件测试的主要目的之一就是尽快尽早地发现软件缺陷，而软件缺陷是由软件本身、团队工作和技术问题等多方面因素引起的，而且集中在需求分析、系统设计这两个阶段，代码的错误要比需求分析书、设计规格说明书所存在的问题要少。

测试的方法可以看做白盒测试方法、黑盒测试方法和静态方法、动态方法等组合后的某种形式，可以满足软件产品需求分析、系统设计、代码等审查工作的要求，也可以为单元测试、集成测试、功能测试、系统测试等提供指导思想。

白盒测试方法注重于程序控制结构，一般适合小规模程序的测试，如模块、构件的测试。黑盒测试方法发现功能需求错误，而不考虑程序的内部工作。黑盒测试技术注重于软件的信息域（输入域和输出域）、功能和性能在整个系统中的表现，扩大了测试焦点，因而可以称为“大规模测试”。

可以根据测试范围、测试目的、测试对象、测试过程对软件测试进行分类，从而比较全面地了解测试的内涵、外延和软件测试工作的整体架构。软件测试工作范畴分为两个层次：软件测试工作的组织与管理、测试工作的实施。而其具体内容包括六个方面：测试组织和管理、测试计划、设计测试用例、测试实施、测试结果分析和测试评审与报告。总之，软件测试不仅是一项技术工作，而且是一项软件产品质量的组织和管理工作。

## 思考题

1. 如何看待软件测试在保证软件产品质量中所起的作用？
2. 谈谈测试的所有目标。
3. 需求分析、系统设计所存在的问题在软件缺陷中占有较大比例，对软件测试工作有何启发？
4. 谈谈在测试工作中综合运用测试基本方法的例子。
5. 测试存在的误区对测试工作可能会产生什么影响？
6. 如何描述测试的整体构架？

## 第 3 章 质量保证与测试策略

学完了前两章有关软件开发与测试的基础知识，了解了软件质量与缺陷的概念以及不同阶段的测试类型。你可能会问，这些活动是自发的吗？软件开发的整个过程是怎么组织与管理起来的？这些过程是如何协调、有效进行的？软件质量保证活动对于软件开发过程起着至关重要的作用，它担负着软件开发流程各阶段的监督、审查、评审和协调等工作，通过对过程的管理以保证软件产品在交付前的质量。同时应该有匹配的管理体系支持，以及如何建立合适质量管理与评判体系。

本章将从软件质量保证（Software Quality Assurance, SQA）开始，并将与此关系密切的内容：测试策略、测试计划、软件质量可靠性评估等融入其中。

### 3.1 软件质量保证

生产任何形式的产品我们都会关心其质量，尽管质量体现在制造过程中，不是 QA 部门可以保证的，但质量是可以通过科学的管理活动而得到控制。通过本节学习，您将了解 SQA 的概念、活动原则及内容。

#### 3.1.1 SQA 概述

我们如何保证产品质量呢？任何形式的产品都是多个过程得到的结果，因此对过程进行管理与控制是提高产品质量的一个重要途径。对于一个软件项目，质量保证活动是自始至终的，它的管理对象是软件过程，是对过程的管理。

##### 1. 概念

软件质量保证是通过对软件产品有计划的进行评审和审计来验证软件是否合乎标准的系统工程。

##### 2. 活动原则

- 确保 SQA 要自始至终有计划的进行。
- 审查软件产品是否遵守适用的标准、规程和要求并得到客观验证。
- SQA 和结果要保证全员参与，沟通顺畅。
- 逐级解决不符合问题（因各公司管理层次划分不同略有差别）。

#### 3.1.2 SQA 活动

软件项目早期阶段，软件质量保证组（对于不同公司，SQA 的存在形式不同，可能是

一个小组或一个部门,甚至有些公司只有这种职能的人存在而没有设置独立的机构,以下均假设为组)与软件项目组一起工作,制定计划、标准和规程等,这些计划、标准和规程将满足项目和组织方针上的要求。通过参与制定计划、标准和规程,软件质量保证组帮助确定和确保项目的需求,验证自身对整个软件生命周期中的评审和审计将是有益的。软件质量保证组在整个项目生存周期中评审项目活动,审计软件产品(过程也是产品),并就软件项目是否遵守已制定的计划、标准和规程等给管理者提供可视信息。一般来说,软件质量保证组首先在软件项目内部处理问题,如发现问题,尽可能的就地解决。对于无法在软件项目内部解决的问题,软件质量保证组逐级把该问题上报到恰当的管理层,以求得解决。

### 1. 影响 SQA 活动效果的重要因素

**知识结构:** SQA 人员的职责是审查软件设计、开发人员的活动,验证他们是否将选定的标准、方法和规程应用到活动中去。因此, SQA 工作的有效执行需要 SQA 人员掌握专业的技术,例如质量管理与控制知识、统计学知识等。

**经验:** SQA 人员的经验对任务的实现同样重要。应该选择那些经验丰富的人来做 SQA,同时为 SQA 人员进行专门的培训,以使他们能够胜任这项工作。

**依据:** SQA 人员在工作时要有依据和判断的标准,如果没有这些标准, SQA 人员就无法准确地判断开发活动中的问题,容易引发不必要的争论,因此公司应当建立文档化的开发标准和规程。

**全员参与:** 全员参与至关重要,高层管理者必须重视软件质量保证活动。在一些公司的软件生产过程中,高级管理者不重视软件质量保证活动,对 SQA 人员发现的问题不及时处理。如此一来,软件质量保证就流于形式,很难发挥它应有的作用。

**把握重点:** SQA 人员在工作过程中一定要抓住问题的重点与本质,尽可能避免陷入对细节的争论之中。SQA 人员应集中审查定义的软件过程是否得到了实现,及时纠正那些疏漏或执行得不完全的步骤,以此来保证软件产品的质量。

### 2. SQA 的活动

总的来说是协调、审查、促进和跟踪,获取有用信息,形成分析结果以指导软件过程。

#### ► 提出软件质量需求

软件质量保证部门在新项目的需求分析阶段就开始介入,对形成的软件需求进行分析与评价,并提出可能存在的问题,诸如安全性、可靠性、可扩展性、易用性等,并根据软件本身特性、规模及将来的运行环境等进行综合评定,确定软件要满足的质量要求,记录下来形成正式文档,尽可能对软件周期各个阶段的测量确定一个定量或定性的标准,作为以后各阶段评审的标准和依据。

#### ► 确定开发方案

经过需求分析阶段深入详细的工作,软件质量保证组与开发部门共同研究并确定软件开发方法,选择软件开发所使用的开发工具。各种开发工具都各有所长,各有侧重,根据要开发软件本身的一些特性和功能要求,同时考虑将来的维护,综合平衡考虑软件过程的各个阶段。比如: Delphi 的数据库功能强大, C 比较适合编写底层的程序……。某些情况



下可能会同时使用多个开发工具进行混合编程。

#### ► 阶段评审

阶段评审就是利用在需求分析阶段所选择与制定的标准与规范以及安排的计划，对软件工程各个阶段的进展、完成质量及出现的问题进行正式技术评审，确保过程遵守相应的标准与规范，形成报告。如果发现不符合问题，遵循逐级解决的原则进行解决，将处理结果通知所有相关人员，记录解决的过程及结果，以作为日后改进的重要参考资料。

#### ► 测试管理

对测试管理的好坏，直接关系到测试实施的效果。SQA 必须从宏观上制定并监督执行软件测试策略和测试计划，形成测试完成的标准以作为审查时的依据及制定测试策略时的参考，并组织测试人员制定更详细的测试计划与案例，促使测试有效的进行。

#### ► 文档化管理

提到文档化管理，首先想让读者思考一个问题：一个好的组织机构是如何一天一天地、一年一年地变好的？读者可能会说：经验丰富，管理得好，技术过硬。且不说技术、经验与管理意味着他们做了些什么，其实一个好的管理方式拿来一用不见得就是好的，而是管理体系本身具有自我完善的特性，也就是不断地自我改进。这好像与文档化管理没有直接关系，其实不然，一个公司要使改进成为可能，首先工作要有合理的依据、步骤、方法和解决问题的原则，同时，这些过程所产生的数据必须记录在案，以供总结经验时参考。工作时的依据及评价的标准不断改进，产生新的文档。同样，SQA 的工作也遵循相同的规则，生成软件文档并对文档的改变进行控制，这也是 SQA 一项非常重要的工作。当然不一定所有的文档内容都是自己独立形成的，比如一个 ISO 或 GB 标准与规范也可以列入我们文档的管理范围内，成为自己文档体系的一部分。

#### ► 验证产品与相应文档和标准的一致性

SQA 人员会依据已经形成的相关文档对应所在的阶段，对过程进行审查，检查执行情况并形成报告，对遇到不符合问题依据逐级解决的原则作相应的处理。同时对此问题的处理过程与结果要通知到与问题相关的所有人，并跟踪至问题彻底解决，以确保软件开发过程与相应文档要求一致。

#### ► 建立测量机制

通过以上活动可定性我们的工作，知道工作到什么程度，达到要求的大致情况，存在哪些不足，最终项目是否可以如期按质按量的完成等。但如果客户想知道你完成的项目所量化的数字，比如：质量情况占我们要求的百分比，项目进展情况占完成整个项目（包括质量要求与项目阶段）比例是多少等，你可能就很难准确回答了，所以建立软件质量要素的测量机制是非常重要的，能让 SQA 人员和领导层了解各种指标的量化信息，方便对项目进度的精确控制与资源的调整，紧急时可以做出准确的决策。

#### ► 记录并生成报告

记录并生成报告，其实属于文档化管理的范畴，前述的文档化管理是对所有阶段所有人的要求，在这里单独提出来，是作为 SQA 本身活动一个部分，而且也是 SQA 工作结果的体现之一。

### 3.1.3 SQA 与软件测试的关系

通过前几节的阐述,我们已经对 SQA 与软件测试的关系有了一定的了解。二者之间既存有包含又存有交叉的关系。

我们先看看正规化的测试流程:项目计划检查、测试计划创建、测试设计、执行测试、更新测试文档;而 SQA 的活动可总结为:协调度量、风险管理、文档检查、促进/协助流程改进、监察测试工作。它们的相同点在于二者都是贯穿整个软件开发生命周期的流程。

SQA 的职能是向管理层提供正确的可视化信息,从而促进与协助流程改进。SQA 还充当测试工作的监督者,使得管理与开发人员不必担心谁来管测试。因此有了 SQA,测试工作就可以被客观的检查与评价,同时也可以协助测试流程的改进。

它们的不同之处在于 SQA 侧重对流程中过程的管理与控制,而测试是对流程中各过程管理与控制策略进行实施。

## 3.2 测试策略

通过之前内容的学习,应对软件开发与测试基础知识有了初步了解,明确了软件质量保证在项目中所起的作用及其与软件测试的关系。是不是所有的软件测试都要运用现有的软件测试方法去测试呢?答案是否定的。依据软件本身的性质、规模及应用场合的不同,我们将选择不同的测试方案,以最少的软件、硬件及人力资源投入得到最佳的测试效果,这就是测试策略目标所在。当然,在提高测试效果的方法与手段上,策略只是一个部分,人员的素质、测试的管理、流程的控制等很多方面的工作都将影响测试效果。

### 3.2.1 测试策略的概念

软件测试通常指实际运行被测程序,输入相应的测试用例,判定执行结果是否符合要求,从而检验程序的正确性、可靠性和有效性。软件测试可采用的方法和技术是多种多样的,但通常情况下不论采用什么方法和技术,其测试都是不彻底的,也是不完全的,因为任何一次完全测试或者穷举测试的工作量都太大,在实践上行不通。因此,任何实际测试,都不能够保证被测试程序中不存在遗漏的缺陷。

为了最大程度地减少这种遗漏的错误,同时也为了最大限度地发现已经存在的错误,在测试实施之前,软件测试工程师必须确定将要采用的测试策略和测试方法,并以此为依据制定详细的测试案例。一个好的测试策略和测试方法,必将给软件测试带来事半功倍的效果,它可以充分利用有限的人力和物力资源,高效率、高质量地完成测试。

由此可知,测试策略通常是描述测试工程的总体方法和目标。描述目前在进行哪一阶段的测试(如单元测试、集成测试、系统测试)以及每个阶段内进行的测试种类(如功能测试、性能测试、压力测试等),以确定合理的测试方案使得测试更有效。

### 3.2.3 测试策略的确定

通过以上分析，我们可以对测试策略的确定过程进行归纳。

输入：

- 要求的硬件和软件组件的详细说明，包括测试工具（测试环境和测试工具数据）。
- 针对测试和进度约束（人员和进度表）而需要的资源的角色和职责说明。
- 测试方法（标准）。
- 应用程序的功能性和技术性需求（需求、变更请求、技术性和功能性设计文档）。
- 系统不能够提供的需求（系统局限）。

输出：

- 已批准和签署的测试策略文档、测试计划、测试用例。
- 需要解决方案的测试项目（通常要求客户项目的管理层协调）。

过程：

- 测试策略是关于如何测试系统的正式描述，要求开发针对所有测试级别的测试策略。测试小组分析需求，编写测试策略并且和项目小组一起复审计划。测试计划应该包括测试用例和条件、测试环境、与任务相关的测试、通过/失败的准则和测试风险评估。测试进度表将识别所有要求成功的测试成果任务，活动的进度和资源要求。

那么，究竟如何才能确定一个好的测试策略和测试方法呢？为了阐述更清晰，我们将分两部分介绍，即：基于测试技术的测试策略和基于测试方案的综合测试策略。

基于测试技术的测试策略，著名软件测试专家 Myers 给出了使用各种测试方法的综合策略。

- 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。
- 必要时用等价类划分方法补充一些测试用例。
- 用错误推测法再追加一些测试用例。
- 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。
- 如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法。

对于基于测试方案的综合测试策略，一般来说，应该考虑以下两个方面。

- 根据程序的重要性和一旦发生故障将造成的损失来确定它的测试等级和测试重点。
- 要认真研究，使用尽可能少的测试用例发现尽可能多的程序错误。因为一次完整的软件测试过后，如果程序中遗漏的错误过多并且很严重，则表明本次测试是失败的、是不足的，而测试不足意味着让用户承担隐藏错误带来的危险。反过来说，如果过度测试，则又会浪费许多宝贵的资源。我们需要在这两点上进行权衡，找到一个最佳平衡点。

## 3.3 测试计划

本节将讨论测试前一项非常重要的工作——测试计划。专业的测试必须以一个好的测试计划作为基础，测试计划应该作为测试的起始步骤和重要环节。

### 3.3.1 制定有效的测试计划

通常，测试计划制定的第一步就是将软件分解成较小而且相对独立的功能模块，写出测试需求。

测试需求有很多分类方法，最普通的一种就是按照功能分类。把软件分解成功能模块有几个好处。

- 测试需求是测试设计和开发测试用例的基础，分解功能模块可以更好地进行设计。
- 详细的测试需求是用来衡量测试覆盖率的重要指标。
- 测试需求包括各种测试实际和开发以及所需资源。

一个测试计划应包括：产品基本情况、测试需求说明、测试策略和记录、测试资源配置、计划表、问题跟踪报告、测试计划的评审和结果等。

#### 1. 产品基本情况调研

这部分应包括产品的一些基本情况介绍，例如：产品的运行平台和应用的领域，产品的特点和主要的功能模块等。对于大的测试项目，还要包括测试的目的和侧重点。具体的要点有：

- 目的。重点描述如何使测试建立在客观的基础上，定义测试的策略、测试的配置、粗略地估计测试大致需要的周期和最终测试报告递交的时间。
- 变更。说明有可能导致测试计划变更的事件。包括测试工具改进了，测试的环境改变了，或者是添加了新的功能。
- 技术结构。可以借助画图，将要测试的软件划分成几个组成部分，规划成一个适用于测试的完整系统，包括数据如何存储、如何传递（数据流图），每一部分的测试要达到什么样的目的，每一部分怎么实现数据更新。还有就是常规性的技术要求，比如运行平台、需要什么样的数据库等。
- 产品规格。制造商和产品版本号的说明。
- 测试范围。简单地描述如何搭建测试平台以及测试的潜在风险。
- 项目信息。说明要测试项目的相关资料，如用户文档、产品描述、主要功能的举例说明。

#### 2. 测试需求说明

这一部分要列出所有要测试的功能项，凡是没有出现在这个清单里的功能项都排除在

测试的范围之外。万一有一天你在一个没有测试的部分里发现了一个问题，你应该很高兴你有这个记录在案的文档，可以证明你测了什么，没测什么。具体要点有：

- 功能的测试。理论上测试要覆盖所有的功能项，例如：在数据库中添加、编辑、删除记录等，这会是一个浩大的工程，但是有利于测试的完整性。
- 设计的测试。对于一些用户界面、菜单结构，还有窗体设计是否合理等的测试。
- 综合考虑。这部分测试需求要考虑数据流从软件中一个模块流到另一个模块过程中的正确性。

### 3. 测试的策略和记录

这是整个测试计划的重点所在，要描述如何公正、客观地开展测试，要考虑模块、功能、整体、系统、版本、压力、性能、配置和安装等各个因素的影响。要尽可能地考虑到细节，越详细越好，并制作测试记录文档的模板，为即将开始的测试做准备。测试记录中要包括的部分具体说明如下。

- 声明：要对测试的公正性、遵照的标准做一个说明，证明测试是客观的。整体上，软件功能要满足需求，实现正确，和用户文档的描述保持一致。
- 测试案例：描述测试案例是什么样的，采用了什么工具，工具的来源是什么，如何执行的，用了什么样的数据。测试的记录中要为将来的回归测试留有余地，当然，也要考虑同时安装别的软件对正在测试软件会造成的影响。
- 特殊考虑：有的时候，针对一些外界环境的影响，要对软件进行一些特殊方面的测试。
- 经验判断：对以往的测试中经常出现的问题加以考虑。
- 设想：采取一些发散性的思维，往往能帮助你找到测试的新途径。

### 4. 测试资源配置

制定一个项目资源计划，包含每一个阶段的任务、所需要的资源，当发生类似到了使用期限或者资源共享等事情的时候，要更新这个计划。

### 5. 计划表

测试的计划表可以做成多个项目通用的形式，根据大致的时间估计来制作。操作流程要以软件测试的常规周期作为参考，也可以根据什么时候应该测试哪一个模块来制定。

### 6. 问题跟踪报告

在测试的计划阶段，我们应该明确如何去做一个问题报告以及如何去界定一个问题的性质，问题报告要包括问题的发现者和修改者、问题发生的频率、用了什么样的测试案例测出该问题的，以及明确问题产生时的测试环境。问题描述尽可能是定量的、分门别类地列举。问题有几种。

- 严重问题：严重问题意味着功能不可用，或者是权限限制方面的失误等，也可能是某个地方的改变造成了别的地方出现问题。

- 一般问题：功能没有按设计要求实现或者是一些界面交互的实现不正确。
- 建议问题：功能运行得不像要求的那么快，或者不符合某些约定俗成的习惯，但不影响系统的性能，如界面显示错误、格式不对、含义模糊、混淆的提示信息等。

### 7. 测试计划的评审

又叫测试规范的评审，在测试真正实施之前必须要认真、负责地检查一遍，获得整个测试部门人员的认同，包括部门负责人的同意和签字。

### 8. 结果

计划并不是到这里就结束了，在最后测试结果的评审中，必须要严格验证计划和实际的执行是不是有偏差，体现在最终报告的内容是否和测试的计划保持一致，然后，就可以开始着手制作下一个测试计划了。

## 3.3.2 通用测试计划模板

以下是引自《测试计划（GB8567—88）》的测试计划通用模板，有删改。

### 计划

#### 1. 软件说明

提供一份图表，并逐项说明被测软件的功能、输入和输出等质量指标，作为叙述测试计划的提纲。

#### 2. 测试内容

列出组装测试和确认测试中每一项测试内容的名称标识符、这些测试的进度安排以及这些测试的内容和目的，例如模块功能测试、接口正确性测试、数据存取的测试、运行时间的测试、设计约束和极限的测试等。

#### 3. 测试1（标识符）

给出这项测试内容的参与单位及被测试的部位。

##### ► 进度安排

给出对这项测试的进度安排，包括进行测试的日期和工作内容（如熟悉环境、培训、准备输入数据等）。

##### ► 条件

陈述本项测试工作对资源的要求，包括：

- 所用到的设备类型、数量和预定使用时间；
- 列出将被用来支持本项测试过程而本身又不是被测软件组成部分的软件，如测试驱动程序、测试监控程序、仿真程序、桩模块等；
- 列出在测试工作期间预期可由用户和开发任务组提供的工作人员人数、技术水平

及有关的预备知识,包括一些特殊要求,如倒班操作和数据输入人员。

#### ► 测试资料

列出本项测试所需的资料,如:

- 有关本项任务的文件;
- 被测试程序及其所在的媒体;
- 测试的输入和输出举例;
- 有关控制此项测试的方法、过程的图表。

#### ► 测试培训

说明为被测软件的使用提供培训的计划。规定培训的内容、受训的人员及从事培训的工作人员。

### 4. 测试 2 (标识符)

用与本测试计划相类似的方式说明用于另一项及其后各项测试内容的测试工作计划。

#### 测试设计说明

##### 1. 测试 1 (标识符)

说明对第一项测试内容的测试设计考虑。

###### (1) 控制

说明本测试的控制方式,如输入是人工、半自动或自动引入、控制操作的顺序以及结果的记录方法。

###### (2) 输入

说明本项测试中所使用的输入数据及选择这些输入数据的策略。

###### (3) 输出

说明预期的输出数据,如测试结果及可能产生的中间结果或运行信息。

###### (4) 过程

说明完成此项测试的一个个步骤和控制命令,包括测试的准备、初始化、中间步骤和运行结束方式。

##### 2. 测试 2 (标识符)

用与本测试计划相类似的方式说明第 2 项及其后各项测试工作的设计考虑。

#### 评价准则

##### (1) 范围

说明所选择的测试用例能够接查的范围及其局限性。

##### (2) 数据整理

陈述为了把测试数据加工成便于评价的适当形式,使得测试结果可以同已知结果进行比较而要用到的转换处理技术,如手工方式或自动方式。如果是用自动方式整理数据,还

要说明为进行处理而要用到的硬件、软件资源。

### (3) 尺度

说明用来判断测试工作是否能通过的评价尺度,如合理的输出结果类型、测试输出结果与预期输出之间的容许偏离范围、允许中断或停机的最大次数。

## 3.4 软件质量的可靠性评估

对于软件质量保证、测试策略和测试计划,读者已经有了较全面了解,但执行完所有的工作就可以宣告该软件是合格品了吗?答案是否定的。如前所述,任何一种有效的方法都不可能确保软件是完美的,不可能保证软件在最终用户使用过程没有任何问题存在。通过一系列的质量保证活动之后,我们势必想知道软件产品的一个定性或者定量的质量信息。本节将就此展开论述。

无处不提软件质量,其实我们所说的软件质量多数是指软件在交付前的质量,那么我们去评价软件交付给最终用户前的质量呢,这将是本节要讨论的问题。

### 3.4.1 软件可靠性评估概述

软件可靠性评估 (software reliability assessment) 指根据软件系统可靠性结构 (单元与系统间可靠性关系)、寿命类型和各单元的可靠性试验信息,利用概率统计方法,评估出系统的可靠性特征量。

#### 1. 软件可靠性评估理论及其发展现状

软件可靠性工程是一门得到普遍承认,但还处于不太成熟的正在发展确立阶段的新兴工程学科。国外从 40 年前就开始加强软件可靠性的研究工作,近 20 年来推出了不少可靠性模型和预测方法,形成较为系统的软件可靠性工程体系,并形成了诸如 AIVEY (软件可靠性和度量标准) 计划、欧洲的 ESPRIT (欧洲信息技术研究与发展战略) 计划、SPMMS (软件生产和维护管理保障) 课题、Eureka (尤里卡) 计划等相关理论。

我国对于软件可靠性的研究工作起步较晚,在软件可靠性量化理论、度量标准 (指标体系)、建模技术、设计方法、测试技术等方面与国外差距较大。国内多数软件的生产方式还处于早期阶段,缺点很明显,主要表现在:透明度差、个人及用户无法在使用前获取软件可靠性信息等。目前,软件可靠性管理方面还没有权威性的管理体系和规范,比如,如何描述软件可靠性、如何测试、如何评估、如何设计、如何提高等。由于目前国内外对于软件可靠性模型的研究多集中在软件的研制阶段,而很少有涉及测试与评估阶段的可靠性模型,所以从事软件可靠性测试与评估研究是一个有理论价值和实际意义并且存在一定难度的课题。

随着计算机软件编制的规范化,必然要将软件可靠性考核纳入科学、规范的轨道。具



体表现在：在软件系统研制中，制定软件可靠性量化指标，使软件考核有明确的标准；建立完善的软件测试、可靠性信息收集系统，使在计算机软件开发中通过科学的软件测试不断减少缺陷；通过研究软件可靠性考核方法，制定相应的软件考核规程、标准；开发软件可靠性评估软件，使软件鉴定更加方便。

## 2. 软件可靠性评估的要素

- 规定的时间

软件可靠性只是体现在其运行阶段，所以将“运行时间”作为“规定时间”度量。“运行时间”包括软件系统运行后工作与挂起（开启但空闲）的累计时间。由于软件运行的环境与程序路径选取的随机性，软件的失效为随机事件，所以运行时间属于随机变量。

- 规定的环境条件

环境条件指软件的运行环境。它涉及软件系统运行时所需的各种支持要素，如支持硬件、操作系统、其他支持软件、输入数据格式和范围以及操作规程等。不同环境条件下软件的可靠性是不同的。具体地说，规定的环境条件主要是描述软件系统运行时计算机的配置情况以及对输入数据的要求，并假定其他一切因素都是理想的。有了明确规定的环境条件，还可以有效判断软件失效的责任在用户方还是研制方。

- 规定的功能

软件可靠性还与规定的任务和功能有关。由于要完成的任务不同，软件的运行剖面会有所区别，调用的子模块就会不同（即程序路径选择不同），其可靠性就可能不同。所以要准确度量软件系统的可靠性必须首先明确它的任务和功能。

## 3.4.2 软件可靠性模型

提到软件可靠性评估，就要涉及软件可靠性模型。软件可靠性模型（software reliability model）是指为预计或估算软件的可靠性所建立的可靠性结构和数学模型。建立可靠性模型是为了将复杂系统的可靠性逐级分解为简单系统的可靠性，以便定量预计、分配、估算和评价复杂系统的可靠性。

一般软件可靠性模型分两大类，即软件可靠性结构模型和软件可靠性预计模型。

### 1. 可靠性结构模型

软件可靠性结构模型是依据系统结构逻辑关系，对系统的可靠性特征及其发展变化规律做出可靠性评价。此模型既可用软件可靠性综合评价又可用于软件可靠性分解。

### 2. 可靠性预计模型

软件可靠性预计模型则是用来描述软件失效与软件缺陷的关系，借助这类模型，可以对软件的可靠性特征作出定量的预计或评估。依据软件缺陷与运行剖面数据，利用统计学原理建立二者之间的数学关系，获取开发过程中可靠性变化、软件在预定工作时间的可靠度、软件在任意时刻发生失效数平均值，以及软件在规定时间间隔内发生失效次数的平均

值。这里需要向读者澄清两词的区别,即评估与预计,评估是对现有的情况进行评价,而预计往往是依据现有的情况及评估结果,对未来可能发生的情况进行科学的推断。预计模型主要有以下几类:

- 面向时间的预计模型,以时间为基准,描述软件可靠性特征随时间变化的规律。
- 面向输入数据的预计模型,描述软件可靠性与输入数据的联系,利用程序运行中的失效次数与成功次数的比作为软件可靠性的度量。
- 面向错误数的预计模型,描述程序中现存错误数的多少预示程序的可靠性。

### 3.4.3 可靠性评估过程

#### 1. 可靠性数据收集

软件可靠性数据是可靠性评估的基础。应该建立软件错误报告、分析与纠正措施系统。按照相关标准的要求,制订和实施软件错误报告及可靠性数据收集、保存、分析和处理的规程,完整、准确地记录软件测试阶段的软件错误报告和收集可靠性数据。

用时间定义的软件可靠性数据可以分为四类:

- 失效时间数据,记录发生一次失效所累积经历的时间。
- 失效间隔时间数据,记录本次失效与上一次失效之间的间隔时间。
- 分组数据,记录某个时间区内发生了多少次失效。
- 分组时间内的累积失效数,记录某个区间内的累积失效数。

这四类数据可以互相转换。

每个测试记录必须包含充分的信息,包括:

- 测试时间。
- 含有测试用例的测试计划或测试说明。
- 所有与测试有关的测试结果,包括所有测试时发生的故障。
- 参与测试的个人身份。

#### 2. 可靠性评估报告

测试活动结束后必须编写《软件可靠性测试报告》,对测试项及测试结果在测试报告中加以总结、归纳。编写时可以参考 GJB 438A—97 中提供的《软件测试报告》格式,并根据情况进行剪裁。测试报告应具备下列内容:

- 产品标识。
- 使用的配置(硬件和软件)。
- 使用的文档。
- 产品说明、用户文档、程序和数据的测试结果。
- 与需求不相符项的列表。
- 测试的最终日期。

这种规范化的过程管理控制有利于获得真实有效的数据,为最终得到客观的评估结果奠定基础。

## 小结

本章由四大部分组成，即 SQA、测试策略、测试计划和软件质量可靠性评估。软件质量保证部分主要讲述 SQA 在软件开发生命同期中的角色与作用及活动内容。测试策略是制订测试计划的重要参考依据，主要讲述针对某一具体软件的特点，如何以最少的人力、物力和时间等资源投入来达到最佳测试效果的综合方法。测试计划主要介绍测试计划的制定方法与步骤以及一个好的测试计划应该包含哪些部分。基于以上三部分的阐述，软件质量可靠性评估的必要性不言而喻。当然，可靠性评估部分的理论发展还不是很成熟，希望读者在提高质量意识的同时还能具有质量度量意识，这将是我们所期望的。

## 思考题

1. 根据本章阐述，试着绘出软件质量保证与测试活动在软件生命周期中的位置图。
2. 用自己的语言表述什么是测试策略，它存在于软件测试的哪个阶段，起什么作用？
3. 领悟过程化事务分析与处理方法，举出一种特例说明通用测试计划模板（见 3.3.2 小节）中没有包含的内容。
4. 查阅关于软件质量可靠性评估的资料，简述软件质量可靠性评估的常用模型。

## 第4章 软件测试依据和规范

本章我们将学习一些与测试有关的软件行业标准、规范和依据。随着社会的进步和科技的迅速发展,“造自己的车、走自己的路”的时代已不复存,现在要站在行业的高度上去开拓属于自己的产品。从管理到技术,甚至更高层次的技术管理,无处不存在行业的标准、约定和惯例。举个简单的例子:你家的某个电器或家具脱落了一个螺丝钉,你可能会很随意地到街市买一个回家安上,这说明什么——制造业的标准在起作用。家用电器是依据标准制造的,所以随之而来的各种标准配件也会很容易找到。同时这些标准与规范大都是行业几年、几十年,甚至成百上千年的经验与技术的结晶,是人类宝贵的财富。软件行业也一样,也是在随着时间的推移不推更新与完善相关标准。

本章将参照 ISO/GB 相关标准,从软件质量标准入手,涉及测试规范、CMM (Capability Maturity Model,能力成熟度模型) 和建立测试管理与评判体系等相关内容。

### 4.1 软件质量标准

在软件质量管理与质量保证体系中,目前国际各行各业广为采用的就是 ISO9000 系列的标准,软件行业也一样。本节主要就 ISO 对软件行业的相关标准进行讨论。

#### 4.1.1 ISO 质量体系标准简介

ISO9000 系列标准是 ISO 国际标准化组织 TC/176 技术委员会制定的所有国际标准,其核心标准是质量保证标准 (ISO9001/2/3) 和质量管理体系标准 (ISO9004)。

ISO 通过它的近 3000 个技术机构开展技术活动。其中技术委员会 (简称 TC) 共 185 个,分技术委员会 (简称 SC) 共 611 个,工作组 (WG) 2022 个,特别工作组 38 个。ISO 技术机构技术活动的成果 (产品) 是“国际标准”。ISO 现已制定出国际标准 300 多个,主要涉及各行各业各种产品 (包括服务产品、知识产品等) 的技术规范。

ISO 制定出来的国际标准除了有规范的名称之外,还有编号。编号的格式是:ISO+标准号+[杠+分标准号]+冒号+发布年号 (方括号中的内容可有可无),例如:ISO8402:1987、ISO9000-1:1994 等。

“ISO9000”不是指一个标准,而是一系列标准的统称。根据 ISO9000-1:1994 的定义:“ISO9000 系列”是由 ISO/TC176 制定的所有国际标准。

什么叫 TC176 呢? TC176 即 ISO 中第 176 个技术委员会,它成立于 1980 年,全称是“质量保证技术委员会”,1987 年又更名为“质量管理和质量保证技术委员会”。TC176 专门负责制定质量管理和质量保证技术的标准。

TC176 最早制定的一个标准是 ISO8402:1986, 名为《质量-术语》, 于 1986 年 6 月 15 日正式发布。1987 年 3 月, ISO 又正式发布了 ISO9000:1987、ISO9001:1987、ISO9002:1987、ISO9003:1987、ISO9004:1987 共 5 个国际标准, 与 ISO8402:1986 一起统称为“ISO9000 系列标准”。

质量保证标准是一个统一各国质量保证标准的产物, 它包含了所有顾客对供方的要求, 是企业建立质量体系取得认证的依据。

ISO9000 系列标准的基本思想, 最主要的有两条: 其一是控制的思想, 即对产品形成的全过程——从采购原材料、加工制造到最终产品的销售、售后服务进行控制。任何一件事物都是由过程组成的, 只要对产品形成的全过程进行控制并达到过程质量要求, 最终产品的质量就有了保证。其二是预防的思想, 通过对产品形成的全过程进行控制以及建立并有效运行自我完善机制达到预防不合格, 从根本上减少或消除不合格产品。

ISO9000 系列标准为 ISO9000 质量管理和质量保证标准选择和使用的导则; ISO9001 质量体系是设计/开发、生产、安装和服务中质量保证模式; ISO9002 质量体系是生产和安装中的质量保证模式; ISO9003 质量体系是最终检验和测试中的质量保证模式; ISO9004 是质量管理和质量体系要素导则。

ISO9000 系列标准的主体部分可以分为两组: 一组是用于“需方对供方要求质量保证”的标准 ISO9001—9003; 一组是用于“供方建立质量保证体系”的标准 ISO9004。

9001、9002 和 9003 之间的区别, 在于其对象的工序范围不同: 9001 范围最广, 包括从设计直到售后服务; 9002 为 9001 的子集, 而 9003 又是 9002 的子集。

## 4.1.2 ISO/GB 软件质量体系标准

ISO9000 系列标准原本是为制造硬件产品而制定的标准, 不能直接用于软件过程。后来 ISO 组织试图将 ISO9001 进行修改用于软件开发方面, 但效果不佳。于是, 以 ISO9000 系列标准的追加形式, 另行制定出 9000-3 标准。这样, 9000-3 就成了用于“使 9001 适用于软件开发、供应及维护”的“指南”。不过, 在 9000-3 的审议过程中, 日本等国家曾先后提出过不少意见。所以, 在内容上与 9001 已有相当多不同。ISO9000-3 (即 GB/T19000.3—94), 全称《质量管理和质量保证标准第三部分: 在软件开发、供应和维护中的使用指南》。

ISO9000-3 作为软件企业实施 ISO9001 质量保证模式标准的实施指南, 通过对软件产品从市场调查、需求分析、软件设计、编码、测试等开发工作直至作为商品软件销售, 以及安装和维护整个过程进行控制, 保障软件产品的质量。现在 ISO9000 系列标准已被各国软件企业广泛采用, 并将其作为建立企业质量体系的依据。

### 1. ISO9000 软件质量体系结构

图 4-1 描绘了一个软件项目从需求到形成产品的全过程, 同时也标识出所涉及的六种类型标准: 过程、产品、工具、技术、人员和材料资源、数据 (包括需求数据、产品数据和工程数据)。另外, 也可按标准的自然属性分为四个类别: 通用标准 (包括术语、组织框

架、参考信息)、原理标准(包括描述各个原理级的关键组织标准)、要素标准(包括带有详细性能要求的标准)、指南和补充(包括如何把原理或要素标准应用于特定场合而提供指南的文档)。

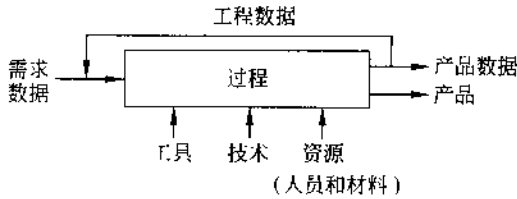


图 4-1 软件质量标准结构

2. ISO9000 与 GB/T19000 的关系

企业不论是申请产品质量认证还是质量体系认证，首先都要贯彻实施 GB/T19000—ISO9000 标准，建立完善的质量体系。GB/T19000—ISO9000 标准是指我国等同采用于国际 ISO9000 标准(有关质量管理和质量保证)的国家标准。等同采用就是把 ISO9000 系列标准的原文翻译过来直接作为国家标准，一般不作任何变动，故 GB/T19000—ISO9000 是 ISO9000 的译文。

国家标准与国际标准编号的对应关系如表 4-1 所示。

表 4-1 国家标准与国际标准编号的对应

GB/T(国标推荐)	Idt(等同于)	ISO 标准
GB/T 6583—1994	idt	ISO8402:1994
GB/T 19000.1—1994	idt	ISO9000 —1:1994
GB/T 19000.2—1994	idt	ISO9000 —2:1994
GB/T 19000.3—1994	idt	ISO9000 —3:1994
GB/T 19001—1994	idt	ISO9001:1994
GB/T 19002—1994	idt	ISO9002:1994
GB/T 19003—1994	idt	ISO9003:1994
GB/T 19004.1—1994	idt	ISO9004 —1:1994
GB/T 19004.2—1994	idt	ISO9004 —2:1991
GB/T 19004.3—1994	idt	ISO9004 —3:1993
GB/T 19004.4—1994	idt	ISO9004 —4:1993
GB/T 19021.1—1993	idt	ISO10011—1:1990
GB/T 19021.2—1993	idt	ISO10011—2:1991
GB/T 19021.3—1993	idt	ISO10011—3:1993
GB/T 19022.1—1994	idt	ISO10012—1:1994

### 4.1.3 ISO9000-3 介绍

我们知道 ISO9000-3 其实是 ISO 质量管理和质量保证标准在软件开发、供应和维护中的使用指南，并不作为质量体系注册/认证时的评估准则，主要考虑软件行业的特殊性制定的标准。参照 ISO9001《质量体系设计、开发、生产、安装和服务的质量保证模式》，并引用 ISO8402《质量管理和质量保证术语》，使得 ISO9000 系列标准应用范围得以拓展。本部分内容主要以 ISO9000-3 标准为蓝本对 ISO9003 进行剖析，其核心内容包括以下这些方面：

- 合同评审
- 需方需求规格说明
- 开发计划
- 质量计划
- 设计和实现
- 测试和确认
- 验收
- 复制、交付和安装
- 维护

#### 1. 合同评审

在投标、接受合同或订单之前，供方应对标书、合同或订单进行评审，以确保：

- 各项要求都有明确规定并形成文件。在以口头方式接到订单，而对要求没有书面说明情况下，供方应确保订单的要求在其接受之前得到同意。
- 任何与投标不一致的合同或订单的要求已经得到解决。
- 供方具有满足合同或订单要求的能力。

#### 2. 需方需求规格说明

在某一具体项目进行开发前，应具有一套该项目的完整、精确、无歧义的功能需求，这些需求应包括需方的所有要求。该需求应足以成为产品验收确认时的依据。

在制订需求规格说明时应注意：

- 双方指定专人负责。
- 需求认可和更改的批准。
- 防止误解，定义好术语，对需求的背景进行说明。
- 记录和评审双方讨论的结果，以备将来查询某些需求、确定原因时使用。

#### 3. 开发计划

在项目进行前制定开发计划，作为总体的策划，指导整个项目有序的进行。开发计划要求包括以下方面：

- 项目定义

- 项目资源组织管理
- 开发阶段
- 进度
- 确定质量保证计划、测试计划、集成计划等

随着项目的进展，开发计划要不断更新，在生命周期模型每一阶段开始之前，都要有该阶段的工作计划，并经过评审后实施。以下详细说明开发计划中应具备的各方面。

#### ► 开发阶段

开发计划应将项目目标转化为最终结果的过程、方法等清楚地描述出来，可以把工作分为几个阶段，比如按照生命周期法划分开发阶段。开发阶段要确定以下项：

- 要执行的开发阶段和每一阶段所需的输入。必须用文档方式确定下来，每一项需求均有明确的定义，以保证完成情况可被检验。
- 每一阶段应产生的输出和验证阶段输出。必须满足以下几点：
  - 满足相应的要求。
  - 有明确的验收准则，作为验收评审的参考。
  - 符合开发惯例和约定。
  - 每一阶段需要执行的验证步骤。
  - 必须有对每阶段输出的验证计划，并在适当的时间进行验证评审。
  - 分析各阶段可能存在的问题或需要解决的问题。

#### ► 项目管理

- 项目开发、实施等过程的时间进度安排。
- 进度的控制方法及活动。
- 确定组织机构职责、各工作组的资源及工作分配。
- 不同工作组间的组织协调方法，并明确技术接口问题。

#### ► 开发方法和工具

- 规定项目活动应共同遵循的方法及使用的工具。
- 开发规范、惯例。
- 开发工具及技术。

#### ► 质量计划

质量计划作为开发计划的一部分，随项目进展而更新，质量计划经正式评审，并得到所有与计划执行有关的组织的统一认可。质量计划应包含或引用以下内容。

- 质量目标。尽可能以定量方式给出。
- 定义每一阶段的输入、输出准则。
- 确定要进行的测试、验证和确认活动类型及详细计划，包括时间、进度等。
- 确定具体质量活动的职责，比如评审和测试、更改控制、对缺陷的控制和纠正措施。

## 4. 设计和实现

设计和实现活动是将需求规格说明转化为软件产品的过程。为保证软件产品的质量，这些活动必须在严格规定的方法下进行，不能依赖于事后的审查监督。



### ► 设计

设计阶段要满足各阶段的共同要求，此外，设计阶段还应考虑如下几方面。

- 选用适合所开发产品类型的设计方法。
- 总结、吸取以往项目的经验教训。
- 设计应考虑软件以后的测试、维护和使用。

### ► 实现

规定编程规则、编程语言、命名约定、编码和注释规则等，要求在实现过程中严格遵守既定开发规则，选用合适的方法和工具实现产品。

### ► 评审

为使需求规格说明得以满足，上述规则方法得以实施，必须以评审的方式加以保证。直到所有被发现的缺陷被消除，或确定缺陷的风险可被控制后，才能进入下一步的设计或实现工作。

## 5. 测试和确认

要具有完整的测试计划，测试计划要经过评审，并以此为依据进行测试活动。

### ► 测试计划

- 包括单元测试计划、集成测试计划、系统测试计划、验收测试计划。
- 制定测试用例、测试数据和预期结果。
- 考虑要进行的测试类型，如功能测试、边界测试、性能测试、可用性测试等。
- 描述测试环境、工具以及测试软件。
- 软件产品是否完成的判断准则。
- 测试所需人员及其要求。

### ► 测试活动

- 记录发现的问题，指出可能受影响的其他部分软件，通知相关负责人员。
- 确定受影响的其他部分软件，并对其进行重新测试。
- 评价测试是否适度 and 适当。
- 在验收和交付产品前，必须尽可能在类似使用环境中进行确认测试。

## 6. 验收

当软件产品已经完成，经过内部确认测试，准备好交付后，应要求需方根据合同中的规定原则判断是否可以进行验收。对于验收中发现问题的处理办法由双方商定并纳入文档。具备验收条件后，应制定验收计划并逐步实施。

验收计划应包括：时间进度、评估规程、软件/硬件环境、验收准则。

## 7. 复制、交付和安装并制定安装分发计划

### ► 复制

制作好安装程序，复制好必要的副本，准备好该交付的操作手册、用户指南等文档。

### ► 交付

交付前应对所交付产品的正确性及完整性进行检验。

### ► 安装

就以下方面双方明确商定各自的作用、责任和义务：

- 时间进度及安排，包括非工作时间及假日的人员安排及工作责任。
- 提供出入便利条件，如通行证等。
- 指定熟练人员的密切配合。
- 提供必要的系统及设备。
- 对每次安装的确认条件需明确规定。
- 对每次安装认可的正式规程。

### 8. 维护

对于软件产品在初次交付及安装后，必须提供的维护应在合同中明确规定。合同中应明确以下各项的维护期：程序、数据、规格说明。

维护工作一般包括：问题的解决、接口的调整、功能扩充和性能改进。

## 4.2 软件测试规范

了解了 ISO 国际标准的基本知识和 ISO/GB 关于软件方面的质量标准后，下面我们来学习更为专业的软件测试方面的规范。

### 4.2.1 概述

软件测试规范就是对软件测试流程的过程化，并对每一个过程的元素进行明确界定，形成完整的规范体系。

我们知道，规范一般形成在标准之后，与规范相比标准显得更为宏观化，而规范往往是标准在某个领域的具体应用中逐步形成的，具有该领域特点更易于操作。软件测试规范也是如此。软件测试规范可分为行业规范与操作规范，行业规范主要是指软件行业长期总结形成的通用规范，而操作规范则指某一公司在长期的软件测试工作中总结出属于自己企业的规范，特别是对于专业提供测试服务的企业，这种操作规范内容与实施情况往往是其取得软件开发商信任的法宝。

本节以 GB\_8566《信息技术软件生存期过程》、ISO/IEC TR 15504 Information technology—Software process assessment 和 CMU/SEI-93-TR-024 Capability Maturity Model for Software 1.1 为参考，剖析软件测试活动规范的核心思想。

### 4.2.2 软件测试规范

一个完整的软件测试规范应该包括规范本身的详细说明，比如规范目的、范围、文档结构、词汇表、参考信息、可追溯性、方针、过程/规范、指南、模板、检查表、培训、

工具、参考资料等。本节主要以一个相对完整的软件测试规范的核心内容作为范例，抛砖引玉，让读者对软件测试规范有较深入的认识。

软件测试过程中一般会从以下几个方面入手来规范过程，并在每个子过程明确角色、职责、活动描述及所需资料。

- 角色的确定
- 进入的准则
- 输入项
- 活动过程
- 输出项
- 验证与确认
- 退出的准则
- 度量

1. 角色

任何项目的实施首先要考虑的是人的因素，对人的识别与确认，软件测试尤其不能例外。在软件测试中会把涉及到的人员按角色进行职责划分。通常会按如表 4-2 所示的方式进行划分。

表 4-2 软件测试的角色职责划分

人员（角色）	职 责
测试设计人员	制定和维护测试计划，设计测试用例及测试过程，生成测试分析报告
测试人员	执行集成测试和系统测试，记录测试结果
设计人员	设计测试需要的驱动程序和稳定桩
编码人员	编写测试驱动程序和稳定桩，执行单元测试

2. 进入准则

进入准则也就是确立软件测试切入点。通过前几章的学习我们知道，软件测试实质上伴随 SQA 在软件开发周期的各个阶段都在进行，因此软件项目立项并等到批准就意味着软件测试的开始。

3. 输入项

软件测试需要相关的文档作为测试设计及测试过程判断符合性的依据和标准，对于需要进行专业单元测试的项目还得有程序单元及软件集成计划相应版本等文档资料。这些文档一并作为测试的输入，如表 4-3 所示。

4. 活动

(1) 制定测试计划

角色：测试设计员

活动描述:

- 制定测试计划的目的是收集和组织测试计划信息，并且创建测试计划。
- 确定测试需求——根据需求收集和组织测试需求信息，确定测试需求。
- 制定测试策略——针对测试需求定义测试类型、测试方法以及需要的测试工具等。
- 建立测试通过准则——根据项目实际情况为每一个层次的测试建立通过准则。
- 确定资源和进度——确定测试需要的软、硬件资源、人力资源以及测试进度。
- 评审测试计划——根据同行评审规范对测试计划进行同行评审。

参考文档：《软件测试计划》模板

表 4-3 作为测试输入的文档

输入名称	输入描述	参考指南、模板
软件项目计划	软件项目计划是一个综合的组装工作，用来收集管理项目时所需的所有信息	《项目开发计划》
软件需求文档	描述软件需求的文档，如软件需求规约(SRS)文档或利用CASE 工具建模生成的文档	《需求规格说明书》
软件构架设计文档	构架设计文档主要描述备选设计方案、软件子系统划分、子系统间接口和错误处理机制等	《概要设计说明书》
软件详细设计文档	详细设计文档主要描述将构架设计转化为最小实施单元，产生可以编码实现的设计	《详细设计说明书》
软件程序单元	包括了所有编码员完成的程序单元源代码	
软件集成计划	软件工作版本的定义、工作版本的内容、集成的策略以及实施的先后顺序等	
软件工作版本	按照集成计划创建的各个集成工作版本	

(2) 测试设计

角色：测试设计员、设计员

活动描述：设计测试的目的是为每一个测试需求确定测试用例集，并且确定执行测试用例的测试过程。

► 设计测试用例

- 对每一个测试需求，确定其需要的测试用例。
- 对每一个测试用例，确定其输入及预期结果。
- 确定测试用例的测试环境配置、需要的驱动界面或稳定桩。
- 编写测试用例文档。
- 对测试用例进行同行评审。

► 开发测试过程

- 根据界面原型为每一个测试用例定义详细的测试步骤。
- 为每一测试步骤定义详细的测试结果验证方法。
- 为测试用例准备输入数据。
- 编写测试过程文档。

- 对测试过程进行同行评审。
- 在实施测试时对测试过程进行更改。
- 设计驱动程序或稳定桩——设计单元测试和集成测试需要的驱动程序和稳定桩。

参考文档：《软件测试用例》模板和《软件测试过程》模板

### (3) 实施测试

角色：测试设计员、编码员

活动描述：实施测试的目的是创建可重用的测试脚本，并且实施测试驱动程序和稳定桩。

- 开发测试脚本——根据测试过程创建测试脚本，并且高度测试脚本。
- 编写驱动程序和稳定桩——根据设计编写测试需要的测试驱动程序和稳定桩。

### (4) 执行单元测试

角色：编码员和测试人员

活动描述：执行单元测试的目的是验证单元的内部结构以及单元实现的功能。

- 执行单元测试——按照测试过程手工执行单元测试或运行测试脚本自动执行测试。
- 记录单元测试结果——将单元测试结果作详细记录，并将测试结果提交给相关组。
- 回归测试——对修改后的单元执行回归测试。

参考文档：《测试日志》和《软件单元测试》

### (5) 执行集成测试

角色：测试员

活动描述：执行集成测试的目的是验证单元之间的接口以及集成工作的功能、性能等。

- 执行集成测试——按照测试过程手工执行集成测试或运行测试，自动执行集成测试。
- 记录集成测试结果——将集成测试结果作详细记录，并将测试结果提交给相关组。
- 回归测试——对修改后的工作版本执行回归测试，或对增量集成后的版本执行回归测试。

### (6) 执行系统测试

角色：测试员

活动描述：执行系统测试的目的是确认软件系统工作版本满足需求。

- 执行系统测试——按照测试过程手工执行系统测试或运行测试脚本，自动执行系统测试。
- 记录系统测试结果——将系统测试结果作详细记录，并将测试结果提交给相关组。
- 回归测试——对修改后的软件系统版本执行回归测试。

### (7) 评估测试

角色：测试设计员和相关组

活动描述：评估测试的目的是对每一次测试结果进行分析评估，在每一个测试阶段提交测试分析报告。

- 分析测试结果——由相关组对一次测试结果进行分析，并提出变更请求或其他处

理意见。

- 分析阶段测试情况：
  - 对每一个阶段的测试覆盖情况进行评估。
  - 对每一个阶段发现的缺陷进行统计分析。
  - 确定每一个测试阶段是否完成测试。
  - 对每一个阶段生成测试分析报告。

5. 输出项（见表 4-4）

表 4-4 输出项内容

输出项	内 容 描 述	形成的文档
软件测试计划	测试计划包含项目范围内的测试目的和测试目标的有关信息。此外，测试计划确定了实施和执行测试时使用的策略，同时还确定了所需资源	软件测试计划模板
软件测试用例	测试用例是为特定目标开发的测试输入、执行条件和预期结果的集合	软件测试用例模板
软件测试过程	测试过程是对给定测试用例（或测试用例集）的设置、执行和结果评估详细描述的集合	软件测试过程模板
测试结果日志	测试结果是记录测试期间测试用例的执行情况，记录测试发现的缺陷，并且用来对缺陷进行跟踪	测试日志模板
测试分析报告	测试分析报告是对每一个阶段（单元测试、集成测试、系统测试）测试结果进行的分析评估	测试分析报告模板

6. 验证与确认（见表 4-5）

表 4-5 验证与确认内容

验证与确认内容	内 容 描 述
软件测试计划评审	由项目经理、测试组及其他相关测试计划进行评审
软件测试用例评审	由测试组及其他相关组对测试用例进行评审
软件测试过程评审	由测试组及其他相关组对测试过程进行评审
测试结果评估	由测试组及其他相关组对测试结果进行评估
测试分析报告评审	由项目经理、测试组及其他相关组对测试分析报告进行评审
SQA 验证	由 SQA 人员对软件测试活动进行审计

7. 退出准则

满足组织/项目的测试停止标准。

8. 度量

软件测试活动达到退出准则的要求时,对于当前版本的测试即告停止。度量工作一般由 SQA 人员通过一系列活动收集数据,利用统计学知识对软件质量进行统计分析,得出较准确的软件质量可靠性评估报告,提供给客户及供方高层领导可视化的质量信息。

## 4.3 CMM 思想和结构体系

CMM (Capability Maturity Model) 即软件能力成熟度模型, 是向软件组织提供如何增加对其开发和维护软件过程的控制能力。设计并实施 CMM 是为了指导软件组织达到以下要求。

- 确定当前过程的成熟度等级, 识别出对软件质量和过程改进至关重要的问题, 选择其过程改进策略。
- 通过关注一组有限的活动, 并为实现它们而积极工作, 组织能稳步地改善其软件过程, 使其软件过程能力持续不断地增长。

### 4.3.1 CMM 的历史

CMM 分阶段的体系结构源于已有 60 多年历史的产品质量原理。Walter Shewart 于 20 世纪 30 年代公布了统计质量控制原理, W. Edwards Deming(Deming 86) 和 Joseph Juran (Juran 88, Juran 89) 的著作又进一步发展和成功地论证了该原理。SEI 已将这些原理改编成为成熟度框架, 该框架为软件过程定量控制建立了项目管理和项目工程的基本原则, 这是过程不断改进的基础。

ITT 的 Philip Crosby 在其书 *Quality is Free* (Crosby 79) 中首先提出将质量原理改编为成熟度框架的思想。Crosby 的质量管理成熟度网格描述了采用质量实践时的五个进化阶段。该成熟度框架又由 IBM 的 Ron Radice 和他的同事们在 Watts Humphrey 指导下进一步改进以适应软件过程 (Radice 85)。1986 年, Humphrey 将此成熟度框架带到软件工程研究所加上了成熟度等级的概念, 研制成当前整个软件产业界所使用的框架基础。

Humphrey 的成熟度框架早期版本发表在 SEI 技术报告 (Humphrey 87a, Humphrey 87b)、文章 (Humphrey 88) 和书 *Managing the software Process* (Humphrey 89) 中。1987 年发表了初步的成熟度提问单 (Humphrey 87b), 它作为工具给组织提供软件过程特征化的一种方法。1987 年又进一步研制出软件过程评估和软件能力评价两种方法, 以便估计软件过程成熟度。自 1990 年以来, 在政府和工业部门许多人的帮助下, SEI 基于几年来将框架运用到软件过程改进方面的经验, 已经进一步扩展和精炼了该模型。

### 4.3.2 CMM 的五个等级及关键过程域

CMM 将软件过程能力成熟度划分为五个等级, 除等级 1 外, 每个成熟度等级被分解成几个关键过程区域 (简称为 KPA), 指明为了改进其软件过程组织应关注的区域。关键过程区域将识别出为了达到基本成熟度等级所必须解决的问题。

每个关键过程区域将识别出一串相关活动, 当这些活动全部完成时, 能达到一组对增强过程能力至关重要的目标。每个关键过程区域按定义存在于单个成熟度等级上。

达到关键过程区域目标的途径可能因项目而异,这是因为在应用领域或环境上有差异。不过,为了使组织实现某个关键过程区域,必须达到该关键域的全部目标。当在连续的基础上对所有项目均已达到一个关键过程区域目标时,可以说,该组织达到了以此关键过程区域为特征的过程能力规范化。

这里经常使用的“关键”一词,蕴含着我们在实现一个成熟度等级过程中,存在并不关键的过程区域和若干过程。CMM 并不仔细描述所有与开发和维护软件有关的过程区域,而是鉴别出过程能力的关键决定因素,CMM 中描述的就是这些因素。尽管其他问题也影响过程的性能,但我们只鉴别出关键过程区域的原因是它们在改进组织软件过程能力上最有效,它们是达到一个成熟度等级的必要条件。为了达到一个成熟度等级,必须实现该等级上的全部关键过程区域。为了实现一个关键过程区域,必须达到该关键区域的每一个目标。目标概括一个关键过程区域的关键实践,可用来确定一个组织或一个项目是否已有效地实现该关键过程区域。目标表明每个关键过程区域的范围、边界和意图。

随着组织晋升到过程成熟度的更高等级,在每个关键过程区域上应进行的具体实践在内容上将有所发展。例如,等级 2 上软件项目策划关键过程区域所描述的项目估计能力中的许多项必须进化,以便能处理在等级 3、4、5 上可得到的、附加的项目数据。当采用已定义软件过程来管理项目时,等级 2 的软件项目策划及软件项目跟踪和监督进化为等级 3 上的集成软件管理。

CMM 的关键过程区域表示一种描述组织如何成熟的方法。确定这些关键过程区域是基于多年来在软件工程和软件管理方面的经验及五年多在软件过程评估和软件能力评价方面的经验。

本节依据 CMM 等级的顺序一一讨论其思想。

- 初始级(等级 1): 软件过程的特点是无秩序的,偶尔甚至是混乱的。几乎没有什么过程是经过定义的,成功依赖于个人的努力。
- 可重复级(等级 2): 已建立基本的项目管理过程去跟踪成本、进度和功能性。必要的过程纪律已经就位,使具有类似应用的项目,能重复以前的成功。
- 已定义级(等级 3): 管理活动和工程活动两方面的软件过程均已文档化、标准化、并集成到组织的标准软件过程。全部项目均采用供开发和维护软件的组织标准软件过程中一个经批准的剪裁本。
- 已管理级(等级 4): 已采集详细的有关软件过程和产品质量的度量。无论软件过程还是产品均得到定量了解和控制。
- 优化级(等级 5): 利用来自过程和来自新思想、新技术先导性试验的定量反馈信息,使持续过程改进成为可能。

### 1. 可重复级

等级 2 上的关键过程区域集中关注软件项目所关心的、与建立基本项目管理控制有关的事情。下面列出对等级 2 上每个关键过程区域的描述。

#### ► 需求管理

目的是在顾客和软件项目之间建立对顾客需求的共同理解,顾客需求将由软件项目处



理。与顾客的协议是策划和管理软件项目的基础。对与顾客关系的控制依靠遵循有效的更改控制过程。

#### ► 软件项目策划

目的是制定进行软件工程和管理软件项目的合理计划。这些计划是管理软件项目的必要基础（正如在软件项目跟踪和监督中所描述的）。没有切合实际的计划不可能实施有效的项目管理。

#### ► 软件项目跟踪和监督

目的是建立对实际进展适当的可视性，使管理者在软件项目性能显著偏离软件计划时能采取有效的措施。

#### ► 软件子合同管理

目的是选择合格的软件子承包商，并有效地管理它们。它把用于基本管理控制的需求管理、软件项目策划、软件项目跟踪和监督等关键过程区域所关注的事情，与软件质量保证以及软件配置管理等过程区域中必不可少的协调结合在一起，并且在合适时对子承包商实施这项管理。

#### ► 软件质量保证

目的是给管理者提供对软件项目正采用的过程和正在构造的产品恰当的可视性。软件质量保证是绝大多数软件工程过程和管理过程不可缺少的组成部分。

#### ► 软件配置管理

目的是在项目的整个软件生存周期中建立和维护软件产品的完整性，软件配置管理是绝大多数软件工程过程和管理过程不可缺少的部分。

## 2. 已定义级

等级3的关键过程区域既阐述项目的问题，又阐述组织的问题，这是因为组织建立起对所有项目有效的软件工程过程和管理过程规范化的基础设施。下面列出对等级3上每个关键过程区域的描述。

#### ► 组织过程焦点

目的是规定组织在改进其整体软件过程能力的活动方面的职责。组织过程焦点活动的主要结果是一组软件过程财富，它们在组织过程定义中加以描述。正如集成软件管理所描述的，这些财富供软件项目使用。

#### ► 组织过程定义

目的是开发和保持一组便于使用的软件过程财富，它们能改进横跨项目的过程性能，并且为组织获得积累性的、长期得益奠定基础。这些财富提供一组稳定的基本原则，通过诸如培训等机制就能使其成为制度。培训在培训大纲中加以描述。

#### ► 培训大纲

目的是培育个人的技能和知识，使他们能有效地和高效率地执行其任务。尽管培训是组织的责任，但是软件项目应该识别出他们所需要的技能，当项目需求独特时，该项目应提供所需要的培训。

### ► 集成软件管理

目的是将软件工程活动和管理活动集成为一个协调的、已定义的软件过程，该过程是剪裁组织的标准软件过程和组织过程定义中所描述的相关过程财富而得到的。剪裁基于项目的经营环境和技术需要，正如在软件产品工程中所描述的那样。集成软件管理是从等级2的软件项目策划以及软件项目跟踪和监督进化而得到的。

### ► 软件产品工程

目的是统一地执行一个妥善定义的工程过程，为了能有效地和高效率地生产正确的、一致的软件产品，该工程过程集成全部软件工程活动。软件产品工程描述项目的技术活动，例如需求分析、设计、编码和测试。

### ► 组间协调

目的是为软件工程组积极参与其他工程组工作制定一种方法，使得项目更能有效地和高效率地满足顾客的需求。组间协调是集成软件管理中涉及多学科的一个方面，它延伸到软件工程之外，不仅应该集成软件过程，而且软件工程组和其他组之间的相互作用也必须加以协调和控制。

### ► 同行评审

目的是及早和高效地除去软件工作产品中的缺陷，一个重要的必然结果是增强对软件工作产品和可预防缺陷的了解。同行评审是一种重要而又有效的工程方法，在软件产品工程中调用此方法，可通过法根式审查（Fagan-style 审查）、（Fagan 86）、结构化走查、或者一些其他的学院式的评审方法（Feedman 90）加以实施。

## 3. 已管理级

等级4上的关键过程区域的关注焦点是建立起对软件过程和正在构造的软件工作产品的定量了解。正如以下所述，该等级上的两个关键过程区域——定量过程管理和软件质量管理——是互相紧密依赖的。

### ► 定量过程管理

目的是定量地控制软件项目的过程性能。软件过程性能表示遵循一个软件过程所得到的实际结果。焦点是在一个可测的稳定过程范围内鉴别出变化的特殊原因，并且适当时改正促使瞬间变化出现的环境。定量过程管理给组织过程定义、集成软件管理、组间协调和同行评审的实践附加一个内容丰富的测量计划。

### ► 软件质量管理

目的是建立对项目软件产品质量的定量了解和实现特定的质量目标。软件质量管理对软件产品工程中所描述的软件工作产品实施内容丰富的测量计划。

## 4. 优化级

等级5上的关键过程区域包括那些为了实施连续不断和可测的软件过程改进，组织和项目都必须解决的问题。下面列出等级5的每个关键过程区域的描述。

### ► 缺陷预防

目的是鉴别缺陷的原因并防止它们再次出现。正如在集成软件管理中所描述的，软件

项目分析缺陷、鉴别其原因并更改项目定义软件进程。正如在过程更改管理中所描述的，应将具有普遍价值的过程更改通知给其他软件项目。

#### ► 技术改革和管理

目的是识别出能获利的新技术（即工具、方法和过程），并以有序的方式将它引进到组织中去，正如在过程更改管理中所描述的那样，技术改革管理的关注焦点是在不断变化的环境里高效率地进行创新。

#### ► 过程更改管理

目的是出于改进软件质量、提高生产率和缩短产品开发周期所采用的持续不断地改进组织的软件过程。过程更改管理既采用缺陷预防的增量式改进，又采用技术改革管理的创新式改进，并使整个组织可以享用这些改进。

### 4.3.3 CMM 与 ISO9001 思想及结构体系的关系

美国 CMM 卡内基·梅隆大学的软件工程研究所（SEI）开发的软件过程能力成熟度模型（CMM）和国际标准化组织（ISO）开发的 ISO9000 标准系列都着眼于质量和过程管理，两者都为了解决同样的问题，直观上是相关的。但是它们的基础却各不相同：ISO9001（ISO9000 标准系列中关于软件开发和维护的部分）确定一个质量体系的最少需求，CMM 则强调持续的过程改进。当然，这种陈述有点主观性，一些国际标准团体坚持认为，如果深入地理解 ISO9001 而不是只停留在表面上，ISO9001 也可以解决持续过程改进的问题，例如，矫正行为可以被解释成持续的改进。

#### 1. ISO9001 和 CMM 既有区别又相互联系

尽管 ISO9001 标准的一些要求在 CMM 中不存在，而 CMM 的一些要求在 ISO9001 标准中也不存在，但不可否认的是，两者之间的关系非常密切。当然，两者之间的差别也很明显，例如，ISO9001 标准的要素 4.7 和 4.15 在 CMM 中没有细述，而 4.19 则是分散在 CMM 的各部分中。ISO9001 的一些要素可以在 CMM 中找到完全对应的部分，另外一些要素则是比较分散的对应。

两者的最大相似之处在于都强调：“该说的要说到，说到的要做到”。对每一个重要的过程应形成文件，包括指导书和说明，并检查交货质量水平。CMM 强调持续改进，ISO9001 的 1994 版标准主要说明的是“合格质量管理体系的最低可接受水平”（ISO9001 的 2000 版标准也增加了持续改进的内容）。

另外，1999 年底，由美国质量协会（ASQ）和 Motorola、Nokia、Bell South 等 100 多家企业、机构共同制定的电信行业（包括电信软件开发企业）质量体系标准 TL 9000 正式发布，在处理已经取得 CMM 和 ISO9001 认证的软件开发企业如何升级到 TL 9000 时，补充审核的要求有很大差异，这从一个侧面说明了它们之间的差别。但很明显，取得 ISO9001 认证对于通过 CMM 评估是有益的，反之，通过 CMM 评估对于获得 ISO9001 认证也是有帮助的。

## 2. 取得 ISO9001 认证并不意味着完全满足 CMM 某个等级的要求

表面上看, 获得 ISO9001 认证的企业应该具有 CMM 第 3 至第 4 级的水平, 但事实上, 有些获得 CMM 第 1 级的企业也获得了 ISO9001 证书, 原因是 ISO9001 强调以顾客的要求为出发点, 不同顾客要求的质量水平不同, 而且各个审核员的水平/解释也有差异。由此可以看出, 取得 ISO9001 认证所代表的质量管理和质量保证能力的高低与审核员对标准的理解及自身水平的高低有很大的关系, 而这并不是 ISO9001 标准本身所决定的。

ISO9001 标准只是质量管理体系的最低可接受准则, 不能说已满足 CMM 的大部分要求, 但有一点可以肯定, ISO9001 认证合格的企业至少能满足 CMM 第 2 级的大部分要求以及第 3 级的一部分要求。

## 3. 通过 CMM 第 2 级(或第 3 级)评估并不代表满足 ISO9001 的要求

CMM 第 2 级的所有关键过程都涉及 ISO9001 的要求, 但都低于 ISO9001 的要求。另外, 一些 CMM 第 1 级的组织在满足了第 2 级和第 3 级的一些关键过程要求后, 也可以获得 ISO9001 认证证书。一些 CMM 第 2 级或第 3 级的企业可能被认为符合 ISO9001 的要求, 但是, 一些通过了 CMM 第 3 级评估的企业也需另外满足 ISO9001 的要素, 才能符合 ISO9001 的要求。

CMM 是专门针对软件开发企业设计的, 因此在针对性上比 ISO9001 要好, 但需要注意的是, CMM 强调的是软件开发过程的管理, 对于国内软件企业涉及较多的“系统集成”并没有考虑, 如果单纯按照 CMM 的要求建立质量体系, 则应该注意补充“系统集成”方面的内容。

本文未明确肯定 CMM 与 ISO9001 相比哪个更好, 因为一个体系的好坏是由很多方面决定的。对于一个软件开发企业来说, 获得什么样的认证只是表面的, 重要的是如何着眼于持续改进以更好地保证软件开发的质量, 满足顾客的要求, 从而获得竞争优势。

# 4.4 建立软件测试管理和评判体系

软件测试是软件质量保证的关键步骤。美国质量保证研究所对软件测试的研究结果表明: 越早发现软件中存在的问题, 开发费用就越低, 在编码后修改软件缺陷的成本是编码前的 10 倍, 在产品交付后修改软件缺陷的成本是交付前的 10 倍; 软件质量越高, 软件发布后的维护费用越低。另外, 根据对国际著名 IT 企业的统计, 它们的软件测试费用占整个软件工程所有研发费用的 50% 以上。

中国软件企业在软件测试方面与国际水准仍存在较大差距。首先, 在认识上重开发、轻测试, 没有认识到软件项目的如期完成不仅取决于开发人员, 更取决于测试人员; 其次, 在管理上随意、简单, 没有建立有效、规范的软件测试管理和评判体系;

最后, 缺少自动化工具的支持, 大多数企业在软件测试时并没有建立软件测试管理与评判体系。

通过前几节的学习, 我们对软件质量标准、测试规范及 CMM 能力成熟度模型等相关内容有了较深的了解, 在此基础上, 我们来学习一下软件测试的管理与评判。

### 1. 建立软件测试管理体系的主要目的

#### ► 软件产品的监视和测量

对软件产品的特性进行监视和测量, 主要依据软件需求规格说明书, 验证产品是否满足要求。所开发的软件产品是否可以交付, 要预先设定质量指标并进行测试, 只有符合预先设定的指标, 才可以交付。

#### ► 对不符合要求产品的识别和控制

对于软件测试中发现的软件缺陷, 要认真记录它们的属性和处理措施, 并进行跟踪, 直至最终解决。在排除软件缺陷之后, 要再次进行验证。

#### ► 产品设计和开发的验证

通过设计测试用例对需求分析、软件设计、程序代码进行验证, 确保程序代码与软件设计说明书一致, 软件设计说明书与需求规格说明书一致。对于验证中发现的不合格现象, 同样要认真记录和处理, 并跟踪解决。解决之后, 也要再次进行验证。

#### ► 软件过程的监视和测量

从软件测试中可以获取大量关于软件过程及其结果的数据和信息, 它们可用于判断这些过程的有效性, 为软件过程的正常运行和持续改进提供决策依据。

### 2. 如何建立测试管理与评判体系

一般应用过程方法和系统方法来建立软件测试管理体系, 也就是把测试管理作为一个系统, 对组成这个系统的各个过程加以识别和管理, 以实现设定的系统目标。同时要使这些过程协同作用、互相促进, 从而使它们的总体作用大于各过程作用之和。其主要目标是在设定的条件限制下, 尽可能发现和排除软件缺陷。测试系统主要由下面 6 个相互关联、相互作用的过程组成。

#### ► 测试规划

确定各测试阶段的目标和策略。这个过程将输出测试计划, 明确要完成的测试活动, 评估完成活动所需要的时间和资源, 设计测试组织和岗位职权, 进行活动安排和资源分配, 安排跟踪和控制测试过程的活动。

测试规划与软件开发活动同步进行。在需求分析阶段, 要完成验收测试计划, 并与需求规格说明一起提交评审; 在概要设计阶段, 要完成和评审系统测试计划; 在详细设计阶段, 要完成和评审集成测试计划; 在编码实现阶段, 要完成和评审单元测试计划。对于测试计划的修订部分, 需要进行重新评审。

#### ► 测试设计

根据测试计划设计测试方案。测试设计过程输出的是各测试阶段使用的测试用例。测

试设计也与软件开发活动同步进行,其结果可以作为各阶段测试计划的附件提交评审。测试设计的另一项内容是回归测试设计,即确定回归测试的用例集。对于测试用例的修订部分,也要求进行重新评审。

#### ► 测试实施

使用测试用例运行程序,将获得的运行结果与预期结果进行比较和分析,记录、跟踪和管理软件缺陷,最终得到测试报告。

#### ► 配置管理

测试配置管理是软件配置管理的子集,作用于测试的各个阶段。其管理对象包括测试计划、测试方案(用例)、测试版本、测试工具及环境、测试结果等。

#### ► 资源管理

包括对人力资源和工作场所,以及相关设施和技术支持的管理。如果建立了测试实验室,还存在其他的管理问题。

#### ► 测试管理

采用适宜的方法对上述过程及结果进行监视,并在适用时进行测量,以保证上述过程的有效性。如果没有实现预定的结果,则应进行适当的调整或纠正。

此外,测试系统与软件修改过程是相互关联、相互作用的。测试系统的输出(软件缺陷报告)是软件修改的输入。反过来,软件修改的输出(新的测试版本)又成为测试系统的输入。

根据上述六个过程,可以确定建立软件测试管理体系的六个步骤。

(1) 识别软件测试所需的过程及其应用,即测试规划、测试设计、测试实施、配置管理、资源管理和测试管理。

(2) 确定这些过程的顺序和相互作用,前一过程的输出是后一过程的输入。其中,配置管理和资源管理是这些过程的支持性过程,测试管理则对其他测试过程进行监视、测试和管理。

(3) 确定这些过程所需的准则和方法,一般应制订这些过程形成文件的程序,以及监视、测量和控制的准则和方法。

(4) 确保可以获得必要的资源和信息,以支持这些过程的运行和对它们的监测。

(5) 监视、测量和分析这些过程。

(6) 实施必要的改进措施。

## 小结

本章前三节主要介绍与软件测试关系较密切的标准、规范及软件过程管理等相关知识,加深读者对软件过程及软件测试的理解。软件测试的管理与评判则以更专业化的描述向读者展示软件测试的综合管理与评判体系。

## 思考题

1. 简述 ISO 体系结构及 ISO9000-3 的总体思想。
2. 以软件测试流程为走向并以 4.2 节所述的规范为依据绘出软件测试活动的流程图。
3. 谈谈你对 CMM 的五个等级及每个等级内的关键过程域的理解。
4. 简要概述建立测试管理与评判体系的六大过程。





## **第 2 部分**

# **软件测试的技术**

# 第5章 单元测试

按阶段进行测试（单元测试、集成测试、系统测试、验收测试）是一种基本的测试策略，单元测试是测试执行过程中的第一个阶段，本章主要从单元测试的定义、目标、过程、技术与方法、评估等方面进行介绍和讨论，并澄清一些在单元测试阶段存在的一些误区。

在测试过程中应该依据每一个阶段的不同特点，采用不同的测试方法和技术，制定不同的测试目标。本章重点论述的是单元测试中最常用到的静态测试技术，也论述了黑盒与白盒的动态测试技术的运用，但如何运用黑盒与白盒测试技术导出具体的测试用例将在第14章中进行详细的论述。

## 5.1 什么是单元测试

单元测试是在软件测试过程中进行的最早期的测试活动。在单元测试活动中，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。

### 5.1.1 单元测试的定义

单元测试是对软件基本组成单元进行的测试。单元测试的对象是软件设计的最小单位——模块。很多参考书中将单元的概念误导为一个具体函数或一个类的方法。一个最小的单元应该有明确的功能、性能定义、接口定义而且可以清晰地与其他单元区分开来。一个菜单、一个显示界面或者能够独立完成的具体功能都可以是一个单元。某种意义上单元的概念已经扩展为组件（component）。

以下几点是在后续各节中需要关注的单元测试的内容。

- 目标：确保模块被正确地编码。
- 依据：详细设计描述。
- 过程：包括设计、脚本开发、执行、调试和分析结果。
- 执行者：由程序开发人员和测试人员共同完成。
- 测试方法：以白盒测试方法为主，辅以黑盒测试方法。
- 评估：通过所有单元测试用例，代码没有严重缺陷。

### 5.1.2 为何要进行单元测试

软件测试的目的之一就是尽可能早地发现软件中存在的错误，从而降低软件质量成本，所以从道理上说，单元测试很重要。但在实践中，因为单元测试的大部分工作由开发人员

完成,而开发人员更多的兴趣在写出代码,而不愿在测试上花比较多的时间,一旦编码完成总是迫切希望进行软件的集成工作。但是如果没有执行好单元测试流程,软件在集成阶段及后续的测试阶段会发现更多的错误,甚至软件根本不能运行。大量的时间将被花费在跟踪那些包含在独立单元里的简单错误上面。所以表面上的进度取代不了实际进度,对于整个项目或系统而言反而会增加额外的工期,导致软件成本的提高。软件中存在的错误发现得越早,则修改和维护的费用就越低,而且难度越小,所以单元测试是早期发现这些错误的最好时机。

另一方面,总有一些自认为很好的程序员,对自己写的程序充满了信心,对单元测试很漠然,希望通过集成测试把问题全找出来。但是规模越大的代码集成意味着复杂性就越高,集成后对每个单元进行全面测试的难度就越高,结果是测试将无法达到它所应该有的全面性。一些缺陷被遗漏,有些缺陷被忽略。如果在后期测试中被发现,将会造成严重影响。所以我们需要完整和规范的单元测试流程。

目前国内软件企业对于软件测试的重视程度有了很大的提高,基于条件限制,许多软件企业还无法开展全面测试。由于单元测试无须太多额外的人员和设备,目前已经被大多数软件企业认知并实施。

## 5.2 单元测试的目标和任务

在讨论单元测试具体的技术和方法时,我们首先要清楚单元测试希望达到的目标,根据这个目标确定要做什么,即单元测试的具体任务是什么。

### 5.2.1 单元测试的目标

确保各单元模块被正确地编码是单元测试的主要目标,但是单元测试的目标不仅测试代码的功能性,还需确保代码在结构上可靠且健全,并且能够在所有条件下正确响应。如果这些系统中的代码未被适当测试,则其弱点可被用于侵入代码,并导致安全性风险(例如内存泄漏或被窃指针)以及性能问题。执行完全的单元测试,可以减少应用级别所需的工作量,并且彻底减少发生误差的可能性。如果手动执行,单元测试可能需要大量的工作,执行高效率单元测试的关键是自动化。

但如果再细分,单元测试需要达到以下一些具体目标。

- 信息能否正确地流入和流出单元。
- 在单元工作过程中,其内部数据能否保持其完整性,包括内部数据的形式、内容及相互关系不发生错误,也包括全局变量在单元中的处理和影响。
- 在为限制数据加工而设置的边界处,能否正确工作。
- 单元的运行能否做到满足特定的逻辑覆盖。
- 单元中发生了错误,其中的出错处理措施是否有效。

单元测试是测试程序代码,为了保证目标的实现,必须制定合理的计划,采用适当的

测试方法和技术, 进行正确评估。

## 5.2.2 单元测试任务

为了实现上述目标, 单元测试的主要任务有:

- 模块接口测试。
- 模块局部数据结构测试。
- 模块边界条件测试。
- 模块中所有独立执行通路测试。
- 模块的各条错误处理通路测试。

### 1. 模块接口测试

只有在数据能正确流入、流出模块的前提下, 其他测试才有意义。对模块接口的检查和确认是单元测试的基础。测试接口正确与否应该考虑下列因素。

- 输入的实际参数与形式参数的个数是否相同。
- 输入的实际参数与形式参数的属性是否匹配。
- 输入的实际参数与形式参数的量纲是否一致。
- 调用其他模块时所给实际参数的个数是否与被调模块的形参个数相同。
- 调用其他模块时所给实际参数的属性是否与被调模块的形参属性匹配。
- 调用其他模块时所给实际参数的量纲是否与被调模块的形参量纲一致。
- 调用预定义函数时所用参数的个数、属性和次序是否正确。
- 是否存在与当前入口点无关的参数引用。
- 是否修改了只读型参数。
- 对全程变量的定义各模块是否一致。
- 是否把某些约束作为参数传递。

如果模块内包括外部输入输出, 还应该考虑下列因素。

文件属性是否正确。

- OPEN/CLOSE 语句是否正确。
- 格式说明与输入输出语句是否匹配。
- 缓冲区大小与记录长度是否匹配。
- 文件使用前是否已经打开。
- 是否处理了文件尾。
- 是否处理了输入输出错误。
- 输出信息中是否有文字性错误。

### 2. 模块局部数据结构测试

检查局部数据结构是为了保证临时存储在模块内的数据在程序执行过程中完整、正确。局部数据结构往往是错误的根源, 应仔细设计测试用例, 力求发现下面几类错误。

- 不合适或不相容的类型说明。
- 变量无初值。
- 变量初始化或默认值有错。
- 不正确的变量名（拼错或不正确地截断）。
- 出现上溢、下溢和地址异常。

### 3. 模块边界条件测试

边界条件测试是单元测试中最重要的一项任务。因为软件经常在边界上失效，采用边界值分析技术，针对边界值及其左、右设计测试用例，很有可能发现新的错误。如果在单元测试中忽略边界条件的测试，那么在以后的测试中很难发现问题，即使被发现后对其跟踪，寻其根源也是一件不容易的事。

### 4. 模块中所有独立执行通路测试

在模块中应对每一条独立执行路径进行测试，单元测试的基本任务是保证模块中每条语句至少能够被执行一次。此时设计测试用例是为了发现因错误计算、不正确的比较和不适当的控制流造成的错误，此时基本路径测试和循环测试是最常用且最有效的测试技术。计算中常见的错误包括：

- 误解或用错了算符优先级。
- 混合类型运算。
- 变量初值错。
- 精度不够。
- 表达式符号错。

比较判断与控制流常常紧密相关，测试用例还应致力于发现下列错误。

- 不同数据类型的对象之间进行比较。
- 错误地使用逻辑运算符或优先级。
- 因计算机表示的局限性，期望理论上相等而实际上不相等的两个量相等。
- 比较运算或变量出错。
- 循环终止条件或不可能出现。
- 迭代发散时不能退出。
- 错误地修改了循环变量。

### 5. 模块的各条错误处理通路测试

一个好的设计应能预见各种出错条件，并预设各种出错处理通路，出错处理通路同样需要认真测试，测试应着重检查下列问题。

- 输出的出错信息难以理解。
- 记录的错误与实际遇到的错误不相符。
- 在程序自定义的出错处理段运行之前，系统已介入。
- 异常处理不当。

- 错误陈述中未能提供足够的定位出错信息。

## 5.3 静态测试技术的运用

静态测试技术是单元测试中最重要的手段之一。其含义是不运行被测程序而对其代码进行分析。适用于新开发的和重用的代码。通常在代码完成并无错误地通过编译或汇编后进行。测试人员主要由软件开发人员及其开发小组成员组成。

### 5.3.1 编码的标准和规范

代码虽然可以正常运行，但是编写可能并不符合某种标准和规范，这相当于写的东西可以被理解，但是不符合语言的语法规则。标准是建立起来和必须遵守的规则——做什么和不做什么。规范是建议最佳做法，推荐更好方式，例如自定义变量和函数的命名。标准没有例外情况，是结构严谨的，规范就相对来说松一点。在一些正规的项目中，经常有一些在测试中表现稳定的软件，因为不符合规范而被认为有问题，为什么呢？至少有三个重要原因可以说明要坚持标准和规范。

- 可靠性：事实证明按照某种标准或规范编写的代码比不这样做的代码更加可靠，软件缺陷更少。
- 可读性和维护性：符合设备标准和规范的代码易于阅读、理解和维护。
- 移植性：代码经常需要在不同的硬件上运行，或者使用不同的编译器编译，如果代码符合标准，迁移到另一个平台就会相对容易，甚至完全没有障碍。

代码中最常用的是表达式，而表达式通常是由变量、函数、常数和运算符组成，通过运算符将变量、函数、常数组合产生一个结果。在变量和函数中，变量通常分为系统变量和自定义变量，自定义变量又分为全局变量和局部变量；函数也分为系统函数和自定义函数两种。因此在检查代码时首先要检查变量定义的对不对，有没有对变量赋予初始值，变量的命名是否正确（新的变量名必须是一个除了系统变量名和已存在的自定义变量名以外的新的名称）以及命名是否符合规范。除了变量和函数外，代码中还有谓词动词语句，例如C语言中的goto、do-while和if-else语句，就有它的编程标准，而目前流行编程语言中，例如，C++、Java、XML都设立了使用它们的标准。例如，著名的MISRA C Coding Standard，这一标准中包括了127条C语言编码标准。通常认为，如果能够完全遵守这些标准，则编写的C代码是易读、可靠、可移植和易于维护的，如：

- 不得使用类型char，必须显式声明为unsigned char或者signed char。
- 所有数字常数应当加上合适的后缀表示类型，例如51L、42U、34.12F等。
- 不得定义与外部作用域中某个标识符同名的对象，以避免遮挡外部作用域中的标识符。
- 具有文件作用域的对象尽量声明为static。
- 在同一个编译单元中，同一个标识符不应该同时具有内部链接和外部链接的声明。

这些标准通常都由4部分组成：标题、标准、解释说明、示例。而规范没有标准严格，有一定灵活性，但却是编写代码时应该遵守的。

每个开发项目由于自身特点都必须符合一组标准，除必须符合计算机语言标准外，还需要符合相应的行业标准，例如金融系统、航天系统的软件都有各自严格的标准。如果想获得大多数计算机语言和信息技术国家的国际标准可以通过以下途径获得。

- 美国国家标准会 (ANST): WWW.ANSI.ORG
- 国际工程协议 (IEC): WWW.IEC.ORG
- 国际标准化组织 (ISO): WWW.ISO.CH
- 计算机机械联合会 (ACM): WWW.ACM.ORG
- 电子电气工程学会 (IEEE): WWW.IEEE.ORG

代码必须清晰易懂，使别的程序员能够容易理解代码所进行的实际工作。在软件工程领域，源程序的风格统一标志着可维护性、可读性，是软件项目的一个重要组成部分。如果没有成文的编码风格文档，程序员就没有一个共同的标准可以遵守，编码风格各异，程序可维护性差，可读性也很差。通过建立代码编写规范，形成开发小组编码约定，提高程序的可靠性、可读性、可修改性、可维护性、可继承性和一致性，可以保证程序代码的质量，继承软件开发成果，充分利用资源，使开发人员之间的工作成果可以共享。

以下是笔者所在过的项目小组曾参考使用过的Java代码的书写规范。由于篇幅较长，略去部分内容以供参考。

案例：

### Java 代码书写规范

#### 1. 目的 (略)

#### 2. 整体编码风格

##### (1) 缩进

缩进建议以4个空格为单位。建议在Tools/Editor Options中设置Editor页面的Block ident为4，Tab Size为8。预处理语句、全局数据、标题、附加说明、函数说明、标号等均顶格书写。语句块的“{ “、” }”配对对齐，并与其前一行对齐，语句块类的语句缩进建议每个“{ “、” }”单独占一行，便于配对。JBuilder默认方式是开始的“{”不是单独一行，建议更改成上述格式（在Project/Default Project Properties中设置Code Style中选择Braces为Next line）。

##### (2) 空格

原则上变量、类、常量数据和函数在其类型、修饰名称之间适当空格并据情况对齐。关键字原则上空一格，如，if (...) 等。运算符的空格规定如下：“::”、“->”、“[”、“]”、“++”、“—”、“~”、“!”、“+”、“-”（指正负号）、“&”（引用）等几个运算符两边不加空格（其中单目运算符系指与操作数相连的一边），其他运算符（包括大多数二目运算符和三目运算符“?:”两边均加一空格，在作函数定义时还可据情况多空或不空格来对齐，但在函数实现时可以用不用。“,”运算符只在其后空一格，需对齐时也可不空或多空格。不论是否有括号，

对语句行后加的注释应用适当空格与语句隔开并尽可能对齐（个人认为此项可以依照个人习惯决定遵循与否）。

### (3) 对齐

原则上关系密切的行应对齐，对齐包括类型、修饰、名称、参数等各部分对齐。另外，每一行的长度不应超过屏幕太多，必要时适当换行，换行时尽可能在“,”处或运算符处，换行后最好以运算符打头，并且以下各行均以该语句首行缩进，但该语句仍以首行的缩进为准，即如其下一行为“{”应与首行对齐。

变量定义最好通过添加空格形成对齐，同一类型的变量最好放在一起。如下例所示。

```
int      Value;
int      Result;
int      Length;
Object   currentEntry;
```

（个人认为此项可以依照个人习惯决定遵循与否）。

### (4) 空行

不得存在无规则的空行，比如说连续 10 个空行。程序文件结构各部分之间空两行，若不必要也可只空一行，各函数实现之间一般空两行，由于每个函数还要有函数说明注释，故通常只需空一行或不空，但对于没有函数说明的情况至少应再空一行。对自己写的函数，建议也加上“//...”作分隔。函数内部数据与代码之间应空至少一行，代码中适当处应以空行空开，建议在代码中出现变量声明时，在其前空一行。类中四个“p”之间至少空一行，在其中的数据与函数之间也应空行。

### (5) 注释

注释是软件可读性的具体体现。程序注释量一般占程序编码量的 20%，软件工程要求不少于 20%。程序注释不能用抽象的语言，类似于“处理”、“循环”这样的计算机抽象语言，要精确表达出程序的处理说明。例如：“计算净需求”、“计算第一道工序的加工工时”等。避免每行程序都使用注释，可以在一段程序的前面加一段注释，具有明确的处理逻辑。

注释必不可少，但也不应过多，不要被动地为写注释而写注释。以下是四种必要的注释：

A. 标题、附加说明。

B. 函数、类等的说明。对几乎每个函数都应有适当的说明，通常加在函数实现之前，在没有函数实现部分的情况下则加在函数原型前，其内容主要是函数的功能、目的、算法等说明，参数说明、返回值说明等，必要时还要有一些如特别的软硬件要求等说明。公用函数、公用类的声明必须由注解说明其使用方法和设计思路。当然，选择恰当的命名格式能够帮助你把事情解释得更清楚。

C. 在代码不明晰或不可移植处必须有一定的说明。

D. 少量的其他注释，如自定义变量的注释、代码书写时间等。

注释有块注释和行注释两种，分别是指：“/\*\*/”和“//”。建议对 A 用块注释，D 用行注释，B、C 则视情况而定，但应统一，至少在一个单元中 B 类注释形式应统一。具体对



不同文件、结构的注释会在后面详细说明。

#### (6) 代码长度

建议对于每一个函数尽可能控制其代码长度为 53 行左右, 超过 53 行的代码要重新考虑将其拆分为两个或两个以上的函数。函数拆分规则应该是不破坏原有算法为基础, 同时拆分出来的部分应该是可以重复利用的。对于在多个模块或者窗体中都要用到的重复性代码, 完全可以将起独立成为一个具备公用性质的函数, 放置于一个公用模块中。

#### (7) 页宽

页宽应该设置为 80 字符。源代码一般不会超过这个宽度, 否则会导致无法完整显示, 但这一设置也可以灵活调整。在任何情况下, 超长的语句应该在一个逗号或者一个操作符后折行, 一条语句折行后, 应该比原来的语句再缩进 2 个字符。

#### (8) 行数

一般的集成编程环境下, 每屏大概只能显示不超过 50 行的程序, 所以这个函数大概要 5~6 屏显示, 在某些环境下要 8 屏左右才能显示完。这样一来, 无论是读程序还是修改程序, 都会有困难。因此建议把完成比较独立功能的程序块抽出, 单独成为一个函数。把完成相同或相近功能的程序块抽出, 独立为一个子函数。可以发现, 这是好程序的一个标志是: 越是上层的函数越简单, 就是调用几个子函数, 越是底层的函数完成的越是具体的工作。这样, 我们就可以在较上层函数里容易控制整个程序的逻辑, 而在底层的函数里专注于某方面的功能的实现了。

3. 代码文件风格 (略)

4. 函数编写风格 (略)

5. 符号风格 (略)

## 5.3.2 走查

走查 (walk through) 是一种使用静态分析方法的非正式评审过程。在此过程中, 设计者或程序员引导小组部分成员已通读过书写的设计和编码。其他成员提出问题并对有关技术、风格、可能的错误、是否有违背开发标准和规范的地方进行评论。走查过程是让与会成员充当计算机, 由被指定作为测试员的小组成员提出一批测试实例, 在会议上对每个测试实例用头脑来执行程序, 在纸上或黑板上监视程序的状态 (即变量的值)。在这个过程中, 测试实例并不起关键作用, 它们仅作为怀疑程序逻辑与计算错误的参照。大多数走查中, 在怀疑程序的过程中所发现的缺陷比通过测试实例本身发现的缺陷更多。编程者对照讲解设计框图和源码图, 特别是对两者相异之处加以解释, 有助于验证设计和实现之间的一致性。

进行走查时要注意限时和避免现场修改。限时是为了避免跑题, 不要针对某个技术问题进行无休止的讨论。发现问题时不要现场修改, 适当地记录, 会后再进行修改是必要的, 否则浪费了大家的时间, 以后就没有人愿意参加该活动了。会议主持人要牢记会议的宗旨和目标。检查的要点是代码编写是否符合标准和规范, 是否存在逻辑错误。

### 5.3.3 审查

审查 (inspection) 是一种正式的检查 and 评估方法, 最早是由 IBM 公司提出, 经实践证明是一种有效的检查方法, 从而得到软件工程界的普遍认同。它是用逐步检查源代码中有无逻辑或语法错误的办法来检测故障。可以认为它是拿代码与标准和规范对照的补充, 因为它不但需要软件开发者自查, 因此要组织代码检查小组进行代码检查。代码检查小组通常由独立的仲裁人、程序编写小组, 其他组程序员和测试小组成员组成。代码检查程序如下: 仲裁人提前把程序目录表和设计说明分配给小组各成员, 小组成员在开会前先熟悉这些材料, 然后开会, 在会议上进行以下工作。

(1) 由程序编写小组成员逐句阐明程序的逻辑, 在此过程中可由程序员或测试小组成员提出问题, 追踪缺陷是否存在。这样简单的工作却是非常有效的缺陷检测方法。

(2) 利用公用程序设计缺陷表来分析讨论。仲裁人负责保证讨论沿着建设性方向进行, 而其他人员则集中注意力发现缺陷。在会议之后把发现的缺陷填入表中交给程序开发小组。如发现重大缺陷, 那么在改正缺陷之后, 还要重新开审议会议。

检查过程所采用的主要技术是设计与使用缺陷检查表。这个表通常是把程序设计中可能发生的各种缺陷进行分类, 以每一类列举尽可能多的典型缺陷, 然后把它们制成表格, 以供在会议中使用, 并且在每次审议会议之后, 对新发现的缺陷也要进行分析、归类, 不断充实缺陷检查表。缺陷检查表根据项目不同而不同, 在实际工作中不断积累完善, 使用缺陷检查表的目的是防止人为的疏漏。下面就是代码检查表的一个示例, 这个示例只对结构化编程测试具有普遍和通用的意义, 不包括特殊应用领域和面向对象软件的测试 (关于面向对象软件的测试, 将在第 8 章详细介绍)。

#### 1. 格式

- 嵌套的 IF 是否正确地缩进。
- 注释是否准确并有意义。
- 是否使用有意义的标号。
- 代码是否基本上与开始时的模块模式一致。
- 是否遵循全套的编程标准。

#### 2. 程序语言的使用

- 是否使用一个或一组最佳动词。
- 模块中是否使用完整定义的语言的有限子集。
- 是否使用了适当的转移语句。

#### 3. 数据引用错误

- 是否引用了未初始化的变量。
- 数组和字符串的下标是否是整数值。下标是否总是在数组和字符串大小范围之内。

- 是否在应该使用常量的地方使用了变量，例如在检查数组范围时。
- 变量是否被赋予了不同类型的值。
- 是否为引用的指针分配了内存。
- 一个数据结构是否在多个函数或者子程序中引用，在每一个引用中是否明确定义了结构了。

#### 4. 数据声明错误

- 所有变量是否都赋予正确的长度、类型和存储类，例如，把本应声明为字符串的变量声明为字符数组。
- 变量是否在声明的同时进行了初始化，是否正确初始化并与其类型一致。
- 变量是否有相似的名称吗，是否自定义变量使用了系统变量名。
- 是否存在声明过，但从未引用或者只引用过一次变量。
- 在特定模块中所有变量是否都显式声明了，如果没有，是否可以理解为该变量与更高级别的模块共享。

#### 5. 计算错误

- 计算中是否使用了不同数据类型的变量，例如将整数与浮点数相加。
- 计算中是否使用了不同数据类型相同但长度不同的变量，例如，将字节与字相加。
- 计算时是否了解和考虑到编译器对类型和长度不一致的变量的转换规则。
- 赋值的目的是否小于赋值表达式的值。
- 在数值计算过程中是否可能出现溢出。
- 除数/模是否可能为 0。
- 对于整型算术运算，特别是除法的代码处理是否会丢失精度。
- 变量的值是否超过有意义的范围。
- 对于包含多个操作数的表达式，求值的次序是否混乱，运算优先级对吗。

#### 6. 比较错误

- 比较是否正确。虽然听起来容易，但是比较中应该是大于还是小于或等于常常发生混淆。
- 是否存在分数或者浮点值之间的比较，如果有，精度问题会影响比较吗。
- 每一个逻辑表达式都正确表达了，逻辑计算如期进行了，求值次序是否有疑问。
- 逻辑表达式的操作数是逻辑值，例如，是否包含整数值的整型变量用于逻辑计算中。

#### 7. 入口和出口的连接

- 初始入口和最终出口是否正确。
- 对另一个模块的每一次调用是否恰当，例如，全部所需的参数是否传送给每一个被调用的模块；被传送的参数值是否正确地设置；对关键的被调用模块的意外情况（如丢失、混乱）是否处理。

- 每个模块的代码是否只有一个入口和一个出口。

#### 8. 存储器的使用

- 每个域，在其第一次被使用前是否正确初始化。
- 规定的域正确否。
- 每个域是否有正确的变量类型声明。

#### 9. 控制流程错误

- 如果程序包含 begin-end 和 do-while 等语句组，end 是否对应。
- 程序、模块、子程序和循环能否终止，如果不能，是否可以接受。
- 是否可能存在永远不停的循环。
- 是否存在循环从不执行的情况，如果是这样，是否可以接受。
- 如果程序包含像 switch-case 语句这样的多个分支，索引变量能超出可能的分支数目，如果超出，该情况能否正确处理。
- 是否存在“丢掉一个”错误，导致意外进入循环的情况。
- 代码执行路径是否已全部覆盖，是否能保证每条源代码语句至少执行一次。

#### 10. 子程序参数错误

- 子程序接收的参数类型和大小与调用代码发送的是否匹配，次序是否正确。
- 如果子程序有多个入口点，引用的参数是否与当前入口点没有关联。
- 常量是否当作形式参数传递，意外在子程序中改动。
- 子程序是否更改了仅作为输入值的参数。
- 每一个参数的单位是否与相应的形参匹配。
- 如果存在全局变量，在所有引用子程序中是否有相似的定义和属性。

#### 11. 输入输出错误

- 软件是否严格遵守外部设备读写数据的专用格式。
- 文件或者外设不存在或者未准备好的错误情况是否处理。
- 软件是否处理外部设备未连接、不可用或者读写过程中存储空间占满等情况。
- 软件是否以预期方式处理预计的错误。
- 是否检查错误提示信息的准确性、正确性、语法和拼写了。

#### 12. 逻辑和性能

- 全部设计是否已实现否了。
- 逻辑是否被最佳地编码了。
- 是否提供正式的错误/例外子程序。
- 每一个循环是否执行正确的次数。

### 13. 维护性和可靠性

- 清单格式是否适于提高可读性。
- 标号和子程序是否符合代码的逻辑意义。
- 对从外部接口采集的数据是否确认。
- 是否遵循可靠性编程要求。
- 是否存在内存泄露的问题。

在审查会前项目经理要做这个检查表，以表的内容为检查依据，检查表的内容是检查的要点。在审核会上项目组的每一个人员都能看到自己和其他人员的编码问题，从而起到预防的作用。这些问题都要被解决，并且解决的结果要在审议会上被确认。审查结束后要完成静态分析错误报告。

表 5-1 对走查与审查作了比较。

表 5-1 走查与审查对比

项 目	走 查	审 查
准备	通读设计和编码	应准备好需求描述文档、程序设计文档、程序的源代码清单、代码编码标准和代码缺陷检查表
形式	非正式会议	正式会议
参加人员	开发人员为主	项目组成员，包括测试人员
主要技术方法	无	缺陷检查表
注意事项	限时，不要现场修改代码	限时，不要现场修改代码
生成文档	会议记录	静态分析错误报告
目标	代码标准规范，无逻辑错误	代码标准规范，无逻辑错误

### 5.3.4 评审

评审（review）通常在审查会后进行，审查小组根据代码审查的错误记录来评估该程序，决定是否需要重新进行审议。《静态分析错误报告》中必须写明错误的类型、影响域、位置和原因等，需交给程序编写者并同时存档。

评审通过的准则是：

- 充分审查了所规定的代码，并且全部编码准则被遵守。
- 审查中发现的错误已全部修改。

## 5.4 动态测试技术的运用

完成静态测试后，还需真正地将程序运行起来完成动态测试。这就需要设计系列的测试用例确保测试的完整性和有效性。在测试用例的设计中，通常会综合白盒与黑盒测试

方法。

### 5.4.1 白盒测试方法

白盒测试技术的理论性比较强，也比较成熟，现实中不是每个单元都可以很好地完整运用白盒测试技术。如不是关键的单元也没有必要将所有的理论付诸于实践。本节主要讲述白盒测试的基本方法和准则。具体如何运用白盒测试技术导出测试用例，请参见第14章。

白盒测试在单元测试中的应用技术有逻辑驱动法和基本路径测试法。这两种方法不考虑软件的功能实现。基于对模块内部结构的清晰了解，要求做到验证内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序，检验程序中的每条通路是否都能按预定要求正确工作。

- 语句覆盖：选择足够的测试用例，使得程序中每一条可执行语句至少被执行一次。
- 判定覆盖：选择足够的测试用例，使得程序中每一个分支判断的每一种可能结果都至少被执行一次。判定覆盖也叫分支覆盖。
- 条件覆盖：选择足够的测试用例，使得程序中每一个分支判断中每一个条件的可能结果都至少被执行一次。
- 判定/条件覆盖：选择足够的测试用例，使得同时满足判定覆盖和条件覆盖。
- 条件组合覆盖：选择足够的测试用例，使得程序中每一个分支判断中每一个条件的每一种可能组合结果都至少被执行一次。
- 路径覆盖：选择足够的测试用例，使得程序中所有的可能路径都至少被执行一次。
- 循环测试：在上下边界及可操作范围内运行所有的循环。

### 5.4.2 黑盒测试方法

黑盒测试方法主要运用于单元的功能和性能方面的测试。如前面所说，单元测试除了测试其功能性之外，还需确保代码在结构上可靠、健全，并且能够有良好的响应，在所有条件下正确响应在单元测试中应用较少，但有时却又是必不可少的。运行被测试单元，有时需要基于被测试单元的接口，开发相应的驱动模块（driver）和桩模块（stub），如图5-1所示。

- 驱动模块（driver）：对底层或子层模块进行集成测试时，所编写的调用这些模块的程序。
- 桩模块（stub）：对顶层或上层模块进行集成测试时，所编写的替代下层模块的程序。

黑盒测试常用的技术和方法包括（详见第14章）：

- 等价类划分法。
- 边界值分析法。
- 错误推测法。

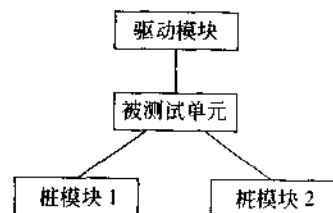


图 5-1 单元测试中驱动模块和桩模块

- 因果图法。
- 功能图法。

在功能性测试方面我们通常会利用三种数据来进行测试，即正常数据、边缘数据和错误数据。

- 正常数据：在测试中所用的正常数据的量是最大的，而且也是最关键的。少量的测试数据不能完全覆盖需求，但要从中提取出一些具有高度代表性的数据作为测试数据，以减少测试时间。
- 边缘数据：是介于正常数据和错误数据之间的一种数据。它可以针对某一种编程语言、编程环境或特定的数据库而专门设定。例如，若使用 SQL Server 数据库，则可把 SQL Server 关键字（如：'; AS; Join 等）设为边缘数据。其他边缘数据还有 HTML 的 HTML; <>等关键字以及空格、@、负数、超长字符等。边缘数据要靠测试人员的丰富经验来制定。
- 错误数据：显而易见，错误数据就是编写与程序输入规范不符的数据从而检测输入筛选、错误处理等程序的分支。

另外，还得考虑接口测试、性能测试、内存测试等。

- 性能分析：代码运行缓慢是开发过程中的一个重要问题。一个应用程序运行速度较慢，程序员不容易找到是在哪里出现了问题。如果不能解决应用程序的性能问题，将降低并极大地影响应用程序的质量，于是查找和修改性能瓶颈成为调整整个代码性能的关键。
- 内存分析：内存泄漏会导致系统运行的崩溃，尤其对于资源比较匮乏、应用非常广泛的嵌入式系统，可能导致无法预料的重大损失。通过测量内存使用情况，可以了解程序内存分配的真实情况，发现对内存的不正常使用，在问题出现前发现征兆，在系统崩溃前发现内存泄露错误。发现内存分配错误，并精确显示发生错误时的上下文情况，指出发生错误的原由。

一位优秀的开发人员会主动要求测试人员对其代码进行充分的测试。一位优秀的测试人员也会对关键的单元十分敏感，进行充分的测试。下面就是笔者所在项目组的一个真实事例。

### 案例：

公司正在进行一项大型的网络服务系统的开发，项目组承担的是服务器端的软件开发。其中有个项目负责多台数据库服务器的数据复制。服务系统是实时的，对数据复制的性能要求当然很高。当开发人员完成了数据传输模块时（还未编写和数据库相关的模块），就主动要求对其性能进行单元测试。

进行这样的性能测试，不需要详细了解该单元的结构，但首先要掌握设计文档中相关的性能指标和运行的网络环境及服务器环境等指标。以便搭建相应的测试环境。

其次，要求开发人员提供相应的程序接口，设计驱动程序和桩程序用来运行并测试该单元程序。此处我们编写了一个功能简单的小程序，即作为驱动模块也是桩模块。该驱动程序在服务器端运行模拟数据库提供和接收需复制的数据，它能够随机产生可设置大小的

数据包,按设置好的单位时间发包数量进行数据包的发送,同时它也是接收端,能对接收到的数据包的数量和大小进行简单的统计,以便实现简单的验证,如图 5-2 所示。

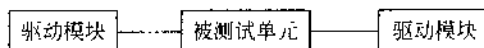


图 5-2 具有桩模块作用的驱动程序

接着就要设计测试用例并实施测试。设计测试用例时要求:

- 根据指标考虑数据包的大小和频率,如大包低频或小包高频。
- 考虑两个驱动程序的数据对发。
- 从两个驱动程序变为多个驱动程序的数据对发。
- 从同一网段变为多个网段,验证代理服务器或网关造成的影响。

发现问题后,得先排除网络等环境因素,再报告开发人员进行调试。

这样会确保该单元不会是该项目的性能瓶颈,也避免了后续开发的盲目性。很多参考书中误导人们认为单元测试采用的是白盒测试技术,由开发人员完成。这很片面,在有些情况下是完全不对的。从另一方面来说,该案例的测试工作也可由开发者完成,但在开发的初期,测试人员并没有大的测试压力,而开发者面临着大量代码编写压力,浪费开发者的时间直接影响项目的进度。何况开发者与测试者的心理状态的不同,还可能直接影响测试结果的可靠性。

## 5.5 调试与评估

调试与测试的对象及采用的方法由很相似,调试还用断点控制等排错方法,但其目的却完全不同。测试是为了找出软件中存在的缺陷,而调试是为了解决存在的缺陷。当程序员尝试多种方法解决了棘手的问题时,会有满足感,许多程序员乐在其中。调试工作是程序员的职责,在此就不多加叙述。

何时可以结束单元测试?测试是否充分足够?如何评估测试的结果?每个项目都有自己的特性和特殊需求,但通常除了代码的标准和规范,单元测试中主要考虑的是对结构的测试,测试用例的覆盖率。在某种意义上说,测试用例设计的好坏决定了测试结果的好坏。下面给出是否通过单元测试的一般准则。

- 软件单元功能与设计需求一致。
- 软件单元接口与设计需求一致。
- 能够正确处理输入和运行中的错误。
- 在单元测试中发现的错误已经得到修改并且通过了测试。
- 达到了相关的覆盖率的要求。
- 完成软件单元测试报告。

有时我们也会借助《单元测试检查表》来对单元测试进行评估。以下提供一个案例,仅供读者参考。



案例:

### 单元测试检查表

单元名称----- 系统 ----- 构造-----  
任务编号----- 初次测试日期-----

#### 关键测试项是否已纠正

1. 有无任何输入参数没有使用? 有无任何输出参数没有产生?
2. 有无任何数据类型不正确或不一致?
3. 有无任何算法与 PDL 或功能需求中的描述不一致?
4. 有无任何局部变量使用前没有初始化?
5. 有无任何外部接口编码错误? 即调用语句、文件存取、数据库错误。
6. 有无任何逻辑路径错误?
7. 该单元是否有多个入口或多个正常的出口?

#### 额外测试项

8. 该单元中有任何地方与 PDL 与 PROLOG 中的描述不一致?
9. 代码中有任何偏离本项目标准的地方?
10. 代码中有任何对于用户来说不清楚的错误提示信息?
11. 如果该单元是设计为可重用的, 代码中是有可能妨碍重用的地方?

采取的动作和说明

(请用单独的一页或多页。每一项动作必须指出所引用的问题。)

#### 审查结果

1. 如果上述 11 个问题的答案均为“否”, 那么测试通过, 请在此标记并且在最后签名。
2. 如果代码存在严重的问题, 例如多个关键问题的答案为“是”, 那么程序编制者纠正这些错误, 并且必须重新安排一次单元测试。

下一次单元测试的日期: -----

3. 如果代码存在小的缺陷, 那么程序编制者纠正这些错误, 并且仲裁者必须安排一次跟踪会议。

跟踪会议的日期: -----

测试人签名: ----- 日期: -----

## 5.6 单元测试的过程与文档管理

单元测试的过程如图 5-3 所示, 由以下五个步骤组成。

- (1) 在详细设计阶段完成单元测试计划。

- (2) 建立单元测试环境，完成测试设计和开发。
- (3) 执行单元测试用例，并且详细记录测试结果。
- (4) 判定测试用例是否通过。
- (5) 提交《单元测试报告》。

在单元测试的各个过程中，必须遵守一定的规则，以一些设计文档为依据产生报告、分析文档等。管理好文档有利于经验的总结、团队的建设。下面就将文档管理与过程管理结合起来进行阐述。

(1) 计划阶段：在软件详细设计阶段完成。制定单元测试计划的主要依据是《软件需求规格说明书》、《软件详细设计说明书》，同时要参考并符合软件的整体测试计划和集成方案。这一阶段完成时输出《单元测试计划》。

单元测试计划的主要内容包括测试时间表、资源分配使用表、测试的基本策略和方法。例如是否需要执行静态测试，是否需要测试工具，是否需要编制驱动模块和桩模块等。

单元测试计划完成后，并不是立刻进行单元测试，代码可能还未完成，在代码编制时软件详细设计文档发生变化已经不是新鲜事了。所以要对需求变化进行跟踪，及时更新《单元测试计划》。《单元测试计划》完成后要对其进行评审，精心制定……？和严格评审是整个单元测试的基础，及时更新也有利于后续各个阶段的顺利进行。

(2) 设计阶段：《单元测试计划》提交后进入设计阶段。主要任务是测试用例的设计编写、驱动模块和桩模块的设计及代码编制。该阶段进行的主要依据是《单元测试计划》、《软件详细设计说明书》。测试用例完成后生成《单元测试用例》文档，许多大型软件公司对测试用例运用数据库进行管理，将测试用例添加至数据库即可。测试用例在执行阶段也要不断地完善和更新。

设计阶段完成的《单元测试用例》也是测试能够顺利执行的保证。

(3) 执行阶段：在代码完成后进行。主要依据是《单元测试用例》文档及《软件需求规格说明书》、《软件详细设计说明书》。主要任务是执行具体的测试用例，验证《软件需求规格说明书》、《软件详细设计说明书》。对测试中发现的错误或缺陷进行记录，生成《缺陷跟踪报告》。将该报告反馈给开发人员，及时修改。许多大型软件公司对缺陷的跟踪也是运用数据库进行管理的。

如果需进行静态测试，还要用到相应的标准和规范文档，生成《代码审查检查表》，即前面提到的《缺陷检查表》。严格地执行《单元测试计划》是整个单元测试的核心，也是保证质量的根本。

(4) 评估阶段：包括测试完备性评估和代码覆盖率评估。进行评估的依据是《单元测试用例》、《缺陷跟踪报告》、《缺陷检查表》等。有时也会借助《单元测试检查表》来对单元测试进行评估。评估的目的是帮助我们判定单元测试是否足够，对该单元的质量给以评价。

(5) 通过单元测试的评估，正式填写并提交《单元测试报告》。

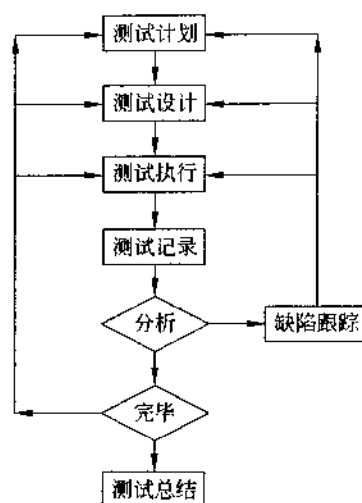


图 5-3 单元测试的过程

## 5.7 单元测试的常用工具简介

单元测试主要由开发人员来做，所以有些开发的集成环境就提供了一些测试的工具，虽然开发的调试和测试有很大的区别，但有些调试的工具可以作为测试工具，软件编译系统也可以看做是一种代码的测试工具。当然，这里主要讨论的专业测试工具，它们按照测试的范围和功能，可以分为下列一些种类。

- 静态分析工具；
- 代码规范审核工具；
- 内存和资源检查工具；
- 测试数据生成工具；
- 测试框架工具；
- 测试结果比较工具；
- 测试度量工具；
- 测试文档生成和管理工具。

使用单元测试工具可以提高工作效率，但要根据项目的特点选择合适的自动测试工具。在选择时通常注意以下两点。

- 自动测试工具的体系结构和文件格式应该是开放的，可以很容易地与其他技术或工具进行交互和集成。
- 自动测试工具厂商应该有比较完善的科室培训和技术支持机制，能够为自动测试工具的实施提供咨询和支持。

下面以 Rational PurifyPlus 为例子，介绍单元测试工具的特点，其他内容参见第 11 章。

Rational PurifyPlus 是一个完整的自动化运行分析工具，用来提高应用程序的性能和质量。它为那些需要进行创建和配置可靠的应用程序的开发者设计，支持 UNIX 平台的 C/C++ 和 Java，以及 Windows 平台上的 Visual C++、C#、Visual Basic.NET、Visual Basic 等。

PurifyPlus for Windows 对于 Java 的服务器端和客户端提供一样的支持。安装在 Web 服务器上面以后，可以在服务器针对诸如 IBM WebSphere、BEA WebLogic 和 Apache Jakarta Tomcat 上的 Java Server Pages (JSPs) 和 Java servlets 使用 PurifyPlus。

Rational PurifyPlus 由 Rational Purify、Quantify 和 PureCoverage 的组成。

- Rational Purify：内存和资源检查测试，用来探测内存泄漏和代码错误。
- Rational Quantify：性能瓶颈检查测试，用来发现性能瓶颈。
- Rational PureCoverage：代码覆盖测试，可标识未测试代码。

PurifyPlus 为 Windows/UNIX 开发者提高了生产率，因为它完全集成在进程当中。它不要求重新编译目标应用程序，不会降低测试进度。PurifyPlus 帮助用户可视化地执行代码，提供便于理解和可重复信息，可以结合或者独立于源代码（包括各种第三方组件）。

## 小结

单元测试的对象是程序系统中的最小单元——模块或组件，其目标不仅测试代码的功能性，还需确保代码在结构上可靠且健全。单元测试是测试执行的开始阶段，而且与程序设计和实现有非常紧密的关系，所以单元测试一般由编程人员和测试人员共同完成，编程人员有时起了主要作用。单元测试的主要任务有：

- 模块接口测试。
- 模块局部数据结构测试。
- 模块边界条件测试。
- 模块中所有独立执行通路测试。
- 模块的各条错误处理通路测试。

静态测试技术是静态分析，是单元测试中最重要的手段之一，如审查（inspection）、走查（walk through）、评审（review）等。

单元测试动态技术主要是白盒测试方法，辅之以黑盒测试方法。白盒测试主要从程序的内部结构出发设计测试用例，检查程序模块或组件已实现的功能与定义的功能是否一致以及编码中是否存在错误。白盒测试方法有逻辑驱动法和基本路径测试法。

由于模块规模小、功能单一、逻辑简单，测试人员有可能通过模块说明书和源程序，清楚地了解该模块的 I/O 条件和模块的逻辑结构，采用结构测试（白盒法）的用例，尽可能达到彻底测试，使之对任何合理和不合理的输入都能鉴别和响应。高可靠性的模块是组成可靠系统的坚实基础。

## 思考题

1. 单元测试的对象不可能是一组函数或多个程序的组合吗？
2. 单元测试是由开发人员完成，采用白盒测试技术吗？
3. 为什么要进行单元测试？
4. 单元测试的主要任务有哪些？
5. 静态测试的审查过程所采用的主要技术是什么？
6. 简述单元测试的各个阶段及涉及的文档。

## 第 6 章 集成测试和系统测试

在将所有功能基本独立的模块经过严格的单元测试以后，接下来需要进行集成测试（integration test）。集成测试是将已分别通过测试的单元按设计要求组合起来再进行的测试，以检查这些单元之间的接口是否存在问题。经过集成测试之后，分散开发的模块被联接起来，构成相对完整的体系，其中各模块间接口存在的种种问题都已基本消除，测试开始进入到系统测试（system test）阶段。系统测试一般由若干个不同测试组成，目的是充分运行系统，验证系统各部件是否都能正常工作并完成所赋予的任务。这里所谓的系统不仅仅包括软件本身，还包括计算机硬件及其相关的外围设备，数据以及收集传输机构，甚至掌握计算机运行的人员及其操作等。通常意义上的系统测试包括压力测试、容量测试、性能测试、安全性测试等。

### 6.1 系统集成的模式与方法

在做软件测试时常会遇到这样的情况，在单元测试中每个模块都能单独工作，但这些模块集成在一起之后会出现有些模块不能正常工作。这主要因为模块相互调用时接口会引入许多新问题。例如，数据经过接口可能丢失；一个模块对另一模块可能造成不应有的影响；几个子功能组合起来不能实现主功能；误差不断积累达到不可接受的程度；全局数据结构出现错误等。

#### 6.1.1 集成测试前的准备

为了更好地做集成测试，先介绍一下集成测试前需要考虑的一些主要因素。

##### 1. 人员安排

集成测试既要求参与的人熟悉单元的内部细节，又要求能够从足够高的层次上观察整个系统。一般由有经验的测试人员和软件开发者共同完成集成测试的计划和执行。

##### 2. 测试计划

集成测试计划在系统设计阶段就开始制定，随着系统设计、开发过程不断细化，最终在系统实施集成之前完成。在这份计划里主要包含的内容有测试的描述和范围、测试的预期目标、测试环境、集成次序、测试用例设计思想、时间表等。

### 3. 测试内容

经过了单元测试后, 需要将所有单元集成到一起, 组成一个完整的软件系统。其重点测试内容包括各单元的接口是否吻合, 代码是否符合规定的标准, 界面标准是否统一等。

### 4. 集成模式

集成方式的选择, 可以是把所有模块按设计要求一次全部组装起来进行测试, 也可以是在模块一个一个地扩展下进行, 使测试的范围逐步增大。

### 5. 测试方法

集成测试阶段是以黑盒测试为主。在自底向上集成的早期, 白盒测试占较大的比例, 随着集成测试的不断深入, 这种比例在测试过程中将越来越少, 渐渐地, 黑盒测试占据主导地位。

## 6.1.2 集成测试的模式

集成模式是软件集成测试中的策略体现, 其重要性是明显的, 直接关系到测试的效率、结果等, 一般要根据具体的系统来决定采用哪种模式。集成测试基本可以概括为以下两种。

- 非渐增式测试模式: 先分别测试每个模块, 再把所有模块按设计要求放在一起合成所要的程序, 如大棒模式。
- 渐增式测试模式: 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, 测试完以后, 再把下一个应该测试的模块结合进来测试。

把所有模块按设计要求一次全部组装起来, 然后进行整体测试, 这称为非增量式集成。这种方法容易出现混乱, 因为测试时可能发现很多错误, 为每个错误定位和纠正非常困难, 并且在改正一个错误的同时又可能引入新的错误, 新旧错误混杂, 更难断定出错的原因和位置。与之相反的是增量式集成模式, 程序一段一段地扩展, 测试的范围一步一步地增大, 错误易于定位和纠正, 界面的测试亦可做到完全、彻底。两种模式中, 增量式集成模式有一定的优势, 但它们有各自的优缺点。

- 渐增式测试模式需要编写的软件较多, 工作量较大, 而非渐增式测试工作量较小。
- 渐增式测试模式发现模块间接口错误早, 而非渐增式测试模式发现晚。
- 非渐增式测试模式发现错误, 较难诊断, 而使用渐增式测试模式, 如果发生错误则往往和最近加进来的那个模块有关。
- 渐增式测试模式测试更彻底。
- 渐增式测试模式需要较多的机器时间。
- 使用非渐增式测试模式, 可以并行测试。
- 针对这两种模式, 其集成方法有自顶向下、自底向上、大棒、三明治等。在实际测试中, 应该将两种模式有机结合起来。

### 6.1.3 自顶向下和自底向上集成方法

在讲解自顶向下与自底向上集成方法之前，先来看看两个经常用到的概念：当对两个以上模块进行集成时，不可能忽视它们和周围模块的相互联系。为模拟这些联系，需设置若干辅助测试模块，也就是连接被测测试模块的程序段。辅助模块有下面两种。

- 驱动模块（driver）：用以模拟被测模块的上级模块。驱动模块在集成测试中接受测试数据，把相关的数据传送给被测模块，启动被测模块，并打印出相应的结果。
- 桩模块（stub）：也称为存根程序，用以模拟被测模块工作过程中所调用的模块。桩模块由被测模块调用，它们一般只进行很少的数据处理，例如打印入口和返回，以便于检验被测模块与其下级模块的接口。

#### 1. 自顶向下法（top-down integration）

自顶向下法，从主控模块（“主程序”）开始，沿着软件的控制层次向下移动，逐渐把各个模块结合起来。在组装过程中，可以使用深度优先的策略或宽度优先的策略，如图 6-1 所示。其具体步骤是：

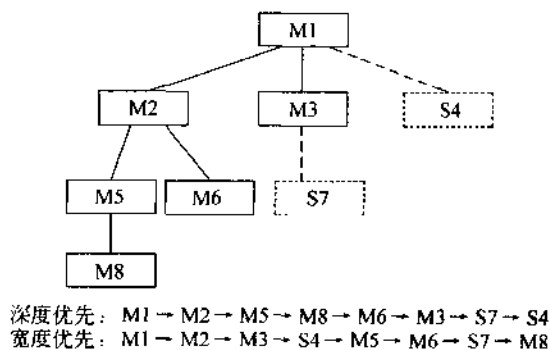


图 6-1 自顶向下集成方法示意图

- (1) 对主控模块进行测试，测试时用桩程序代替所有直接附属于主控模块的模块。
- (2) 根据选定的结合策略（深度优先或宽度优先），每次用一个实际模块代替一个桩程序（新结合进来的模块往往又需要新的桩程序）。
- (3) 在结合下一个模块的同时进行测试。
- (4) 为了保证加入模块没有引进新的错误，可能需要进行回归测试（即全部或部分地重复以前做过的测试）。

从第（2）步开始不断地重复进行上述过程，直至完成。

自顶向下法的主要优点：不需要测试驱动程序，能够在测试阶段的早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误。其缺点是：需要桩程序，可能遇到与此相联系的测试困难，低层关键模块中的错误发现较晚，而且用这种方法在早期不能充分展开人力。

2. 自底向上法 (bottom-up integration)

自底向上测试从“原子”模块（即在软件结构最低层的模块）开始集成以进行测试，如图 6-2 所示。具体步骤是：

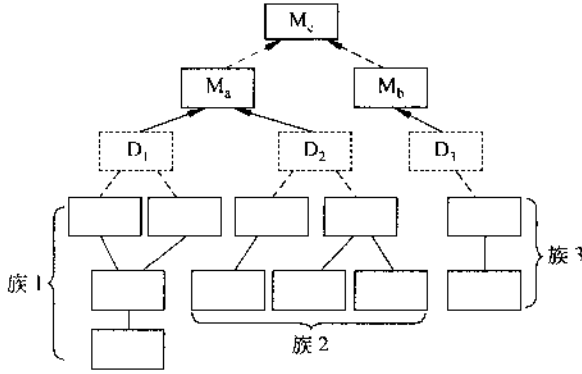


图 6-2 自底向上集成方法示意图

- (1) 把低层模块组合成实现某个特定软件子功能的族。
- (2) 写一个驱动程序（用于测试的控制程序），协调测试数据的输入和输出。
- (3) 对由模块组成的子功能族进行测试。
- (4) 去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成更大的子功能族 (cluster)。

从第(2)步开始不断地重复进行上述过程，直至完成。

自底向上法的优缺点与自顶向下法正好相反。

3. 混合策略 (modified top-down integration)

在具体测试中，采用混合策略：

- 改进的自顶向下法：基本使用“自顶向下”法，但在测试早期，使用“自底向上”法测试少数的关键模块。
- 混合法：对软件结构中较上层使用“自顶向下”法；对软件结构中较下层使用“自底向上”法，两者相结合，如图 6-3 所示。

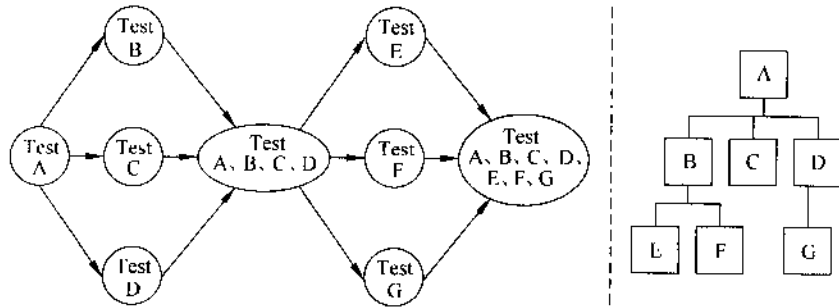


图 6-3 混合策略集成示意图



### 6.1.4 大棒与三明治集成方法

#### 1. 大棒集成方法 (big-bang integration)

采用大棒集成方法, 先对每一个子模块进行测试 (单元测试阶段), 然后将所有模块全部集成起来一次性进行集成测试, 如图 6-4 所示。因为所有模块一次集成, 所以很难确定出错的真正位置、所在的模块、错误的原因。

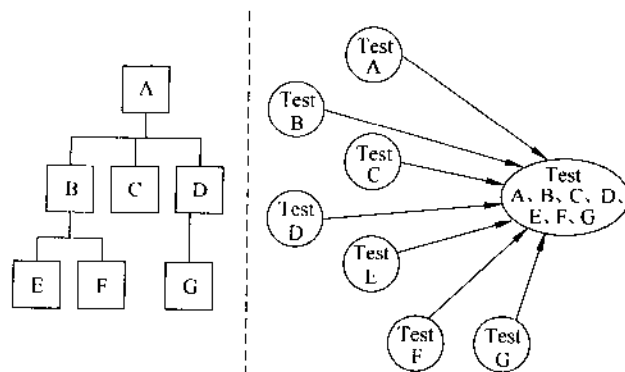


图 6-4 大棒集成方法示意图

由于这样, 这种方法并不推荐在任何系统中使用, 但适合在规模较小的应用系统中使用。

#### 2. 三明治集成方法 (sandwich integration)

三明治集成方法自两头向中间集成, 如图 6-5 所示。

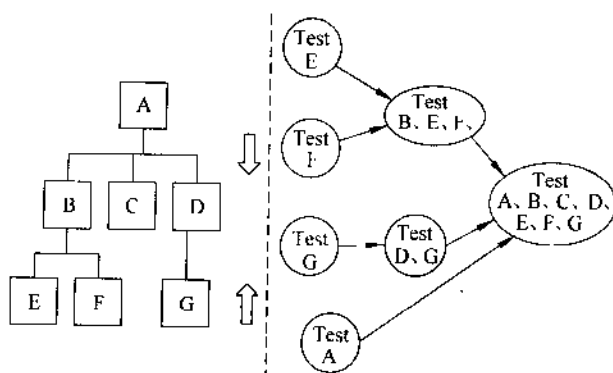


图 6-5 三明治集成方法示意图

采用三明治方法的优点是: 它将自顶向下和自底向上的集成方法有机地结合起来, 不需要写桩程序, 因为在测试初期自底向上集成已经验证了底层模块的正确性。采用这种方法的主要缺点是在真正集成之前每一个独立的模块没有完全测试过。

3. 改进的三明治集成方法 (modified sandwich integration)

改进的三明治集成方法, 不仅自两头向中间集成, 而且保证每个模块得到单独的测试, 使测试进行得比较彻底, 如图 6-6 所示。

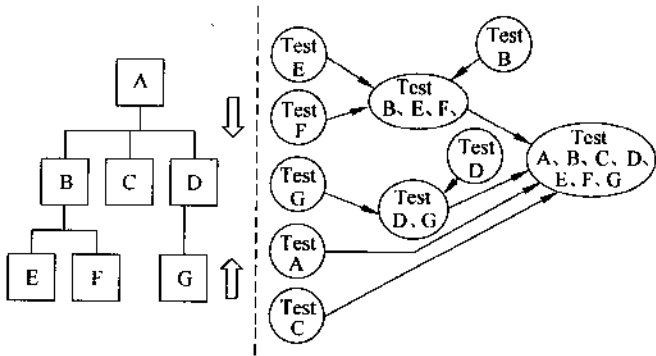


图 6-6 改善的三明治集成方法示意图

最后, 将以上介绍的几种集成测试方法进行一个系统的比较, 见表 6-1。

表 6-1 集成方法性能的比较

名 称	自底向上	自顶向下	混合策略	大棒	三明治	改进三明治
集成	早	早	早	晚	早	早
基本程序能工作时间	晚	早	早	晚	早	早
需要驱动程序	是	否	是	是	是	是
需要桩程序	否	是	是	是	是	是
工作并行性	中	低	中	高	中	高
特殊路径测试	容易	难	容易	容易	中等	容易
计划与控制	容易	难	难	容易	难	难

6.1.5 持续集成

通常系统集成都会采用持续集成的策略, 软件开发中各个模块不是同时完成, 根据进度将完成的模块尽可能早地进行集成, 有助于尽快发现缺陷, 避免集成中大量缺陷涌现。同时自底向上集成时, 先期完成的模块将是后期模块的桩程序; 而自顶向下集成时, 先期完成的模块将是后期模块的驱动程序, 从而使后期模块的单元测试和集成测试出现了部分的交叉, 不仅节省了测试代码的编写, 也有利于提高工作效率。

在没有采用持续集成策略的开发中, 开发人员经常需要集中开会来分析软件究竟在什么地方出了错。因为某个程序员在写自己这个模块代码时, 可能会影响其他模块的代码, 造成与已有程序的变量冲突、接口错误, 结果导致被影响的人还不知道发生了什么, 缺陷就出现了。这种缺陷是最难查的, 因为问题不是出在某一个人的领域里, 而是出在两个人

的交流上面。随着时间的推移,问题会逐渐恶化。通常,在集成阶段出现的缺陷早在几周甚至几个月之前就已经存在了。结果,开发者需要在集成阶段耗费大量的时间和精力来寻找这些缺陷的根源。

如果使用持续集成,这样的缺陷绝大多数都可以在引入的第一天就被发现。而且,由于一天之中发生变动的部分并不多,所以可以很快找到出错的位置。如果找不到缺陷究竟在哪里,也可以不把这些出错的代码集成到产品中去。

所以,持续集成可以减少集成阶段消灭缺陷所消耗的时间,从而最终提高软件开发的质量与效率。

## 6.2 功能测试

在单元测试中也讲到了功能测试,当时目的是保证所测试的每个独立模块在功能上是正确的,主要是从输入条件和输出结果来进行判断是否满足程序的设计要求。在单元测试时主要采用白盒测试法,从程序的角度来判断模块内部有没有错误或是潜在的隐患,以消除局部模块逻辑和功能上的错误及缺陷。这里所说的功能测试,是在系统集成过程中和系统集成之后所进行的系统功能测试,不仅要考虑模块之间的相互作用,而且要考虑系统的应用环境,其衡量标准是实现产品规格说明书上所要求的功能。

### 6.2.1 功能测试的目的和内容

在集成测试的时候,要确保集成后的各个新模块不会出现错误,即原来每个模块功能不会因为集成而消失。在软件集成的功能测试时,多采用黑盒测试方法,所以黑盒测试也常被称为功能测试,虽然这不是一种准确的说法。在功能测试的时候,有时也有必要查看源代码、数据库的值,配以相应的白盒测试方法。

功能测试比较容易理解,主要是根据产品规格说明书,来检验被测试的系统是否满足各方面功能的使用要求,主要包括:

- 程序安装、启动正常,有相应的提示框、错误提示等。
- 每项功能符合实际要求。
- 系统的界面清晰、美观。
- 菜单和按钮操作正常、灵活,能处理一些异常操作。
- 能接受正确的数据输入,对异常数据的输入可以进行提示、容错处理等。
- 数据的输出结果准确,格式清晰,可以保存和读取。
- 功能逻辑清楚,符合使用者习惯。
- 系统的各种状态按照业务流程而变化,并保持稳定。
- 支持各种应用的环境。
- 能配合多种硬件周边设备。
- 软件升级后,能继续支持旧版本的数据。

- 与外部应用系统的接口有效。

对于功能测试，这里列出的是一般情况，针对不同的应用系统，其测试内容的差异很大，但都可以归为界面、数据、操作、逻辑、接口等几个方面。至于安全性、兼容性、可靠性等测试归为系统测试的范畴，在后面几节介绍。

## 6.2.2 功能测试的方法

目前在功能测试中，常用的黑盒测试方法有等价类划分法、边界值划分法、错误推测法、因果图法和组合分析法。下面将一一给予简单介绍，关于其测试用例设计，请参考第14章。

### 1. 等价类划分法

数据测试是功能测试的主要内容，或者说功能测试最主要的手段之一就是借助数据的输入/输出来判断功能能否正常运行。在进行数据输入测试时，如果需要证明数据输入不会引起功能上的错误、或者其输出结果在各种输入条件下都是正确的，就需要将可输入数据域内的值完全尝试一遍（即穷举法），这实际是不现实的。

假如一个程序P有输入量I1和I2及输出量O，在字长为32位的计算机上运行。如果I1和I2均取整数，则测试数据的最大可能数目为： $2^{32} \times 2^{32} = 2^{64}$ 。

如果测试程序P，采用穷举法力图无遗漏地发现程序中的所有错误，且假定运行一组(I1, I2)数据需1毫秒，一天工作24小时，一年工作365天，则 $2^{64}$ 组测试数据需5亿年。说明穷举的黑盒测试通常是不能实现的，因此只能选取少量有代表性的输入数据，以期用较小的代价暴露出较多的错误。

为了解决这个问题，人们就设想是否可以用一组有限的数去代表近似无限的数据，这就是等价类划分法的基本思想。等价类划分法选择适当的数据子集来代表整个数据集，通过降低测试的数目实现“合理的”覆盖，覆盖了更多的可能数据，以发现更多的软件缺陷。

等价类划分法基于对输入或输出情况的评估，然后划分到两个或更多子集来进行测试，即将所有可能的输入数据（有效的或无效的）划分成若干个等价类，从每个等价类中选择一定的代表值进行测试。其中有一个假定：等价类中的所有数据对于暴露程序中的错误是等效的，即如果用这个等价类中的代表值作为测试用例未发现程序错误，那么该类中其他的测试用例也不会发现程序的错误。有时，在确定输入数据的等价类时常要分析输出数据的等价类，以便根据输出数据的等价类导出对应的输入数据等价类。这样就将漫无边际的随机测试变为具有针对性的有效测试，极大地提高了测试效率。

先举一个抽象并能完全描述等价类划分法的例子，假设函数需要3个参数(A、B、C)，每个参数的输入域分别在以下的子集里（参见图6-2）：

$$A = A1 \cup A2 \cup A3; \quad B = B1 \cup B2 \cup B3; \quad C = C1 \cup C2$$

表 6-2 等价类划分法分析

用 例	参数 1	参数 2	参数 3
1	A	B	C
2	≠A	B	C
3	A	≠B	C
4	A	B	≠C

下面再举一个例子来说明等价类划分法的具体意义。例如，有一报表处理系统，要求用户输入处理报表的日期。假设日期限制在 2000 年 1 月至 2020 年 12 月，即系统只能对该段时期内的报表进行处理。如果用户输入的日期不在这个范围内，则显示错误信息。并且此系统规定日期由年月的 6 位数字组成，前 4 位代表年，后 2 位代表月，则检查日期时，可用表 6-3 进行等价类划分和编号。

表 6-3 等价类划分的一个实例

输 入	合理等价类	不合理等价类
报表日期	1. 6 位数字字符	2. 有非数字字符 3. 少于 6 个数字字符 4. 多于 6 个数字字符
年份范围	5. 在 2000 ~ 2020	6. 小于 2000 7. 大于 2020
月份范围	8. 在 1 ~ 12	9. 等于 0 10. 大于 12

在进行功能测试时，只要对合理等价类和不合理等价类测试进行测试，覆盖 1，5，8 三个合理等价类测试，只要用一个值 201006 即可；对不合理等价类的测试则要分别输入 7 个非法数据，如 200a0b、20102、1012012、198802、203011、200000、202013。合起来只要完成 8 个数据的输入就可以了。如果不用等价类划分法，其测试的输入值是几百、上千个，可见等价类划分法提高了测试效率。

等价类划分法优点是基于相对较少的测试案例，就能够进行完整覆盖，很大程度上减少了重复性；缺点是缺乏特殊案例的考虑，同时需要深入的系统知识，才能做到有效地处理。

2. 边界值分析法

实践证明，程序往往在输入输出的处理边界情况下发生错误。边界情况指输入等价类和输出等价类边界上的情况，检查边界情况的测试用例是比较高效的，可以查出更多的错误。如上面介绍的处理报表日期的例子，等价类划分法就忽略了几个边界条件，如 200001（边界有效最小值）、202012（边界有效最大值）以及边界无效值 199901、199912、202101、202112 等，而程序往往会在这些地方出错。这就是下面要讨论的边界值分析法的优点。

边界值分析法就是在某个变量范围的边界上，验证独立的输入/输出是否正确的测试方

法。边界值分析法取决于变量的范围和范围的类型，确认所有输入的边界条件或临界值，然后选择这些边界条件/临界值及其附近的值来进行相关功能的测试。边界值分析法处理技巧有：

- 如果输入条件规定了值的范围，则取刚达到这个范围的边界值（如上述 200001、202012），以及刚刚超过这个范围边界的值（如上述 199912、202101）。
- 如果输入条件规定了值的个数，则用最大个数、最小个数、比最大个数多 1 个、比最小个数少 1 个的数等作为测试数据。
- 根据规格说明的每一个输出条件，分别使用以上两个规则。
- 如果程序的规格说明给出的输入域或输出域是有序集合（如有序表、顺序文件等），则应选取集合的第一个和最后一个元素作为测试数据。
- 如果程序用了一个内部结构，应该选取这个内部数据结构的边界值作为测试数据。
- 分析规格说明，找出其他可能的边界条件。

这里给出测试一个排序程序的边界值分析法的例子，其边界条件有：

- 排序序列为空。
- 排序序列仅有一个数据。
- 排序序列为满，用猜错法补充一下测试用例。
- 排序序列已经按要求排好序。
- 排序序列的顺序与要求的顺序恰好相反。
- 排序序列中的所有数据全部相等。

因为错误最容易发生在边界值附近，所以边界值分析法对于多变量函数的测试很有效，尤其是对于像 C/C++ 这种数据类型要求不是很严格的语言有用。缺点是对布尔值或逻辑变量无效，也不能很好地测试不同的输入组合。边界值分析法也不具有随机性，常被看做是等价类划分法的一种补充，两者结合起来使用更有效。

### 3. 错误推测法

有经验的测试人员往往可以根据自己的工作经验和直觉推测出程序可能存在的错误，从而有针对性地进行测试，这就是错误推测法。错误推测法是测试者根据经验、知识和直觉来发现软件错误，“只可意会，不能言传”，就是表明这样一个道理。

错误推测法基于这样一个思想：某处发现了缺陷，则可能会隐藏更多的缺陷，在实际操作中，列出程序中所有可能的错误和容易发生的特殊情况，然后依据测试者经验做出选择。如等价类划分法和边界值分析法通过选择有代表性的测试数据来暴露程序错误，但不同类型、不同特点的程序通常又有一些特殊的容易出错的情况。并且，有时分别使用某些测试数据或用例进行测试时程序工作正常，但其组合可能会使程序出错。例如，程序中两个模块使用并修改某些共享的变量，则在程序运行测试过程中应对这些共享的变量赋值来回来验证这两个模块，一般说来，可能的输入组合数目往往很多。因此，测试人员应依靠经验和直觉，从各种可能的方案中选出最可能引起程序出错的方案。

错误推测法能充分发挥人的直觉和经验，在一个测试小组中集思广益，方便实用，特别是在软件测试基础较差的情况下，很好地组织测试小组进行错误猜测，是一种有效的测



系列的严格有效的测试来发现软件的潜在问题，保证系统的运行。

系统测试明显区别于功能测试。功能测试主要是验证软件功能的实现情况，不考虑各种环境以及非功能问题，如安全性、可靠性、性能等，而系统测试是在更大的范围内进行的测试，着重对系统的性能、特性进行测试。

### 6.3.1 系统测试的内容

系统测试应该由若干个不同测试组成，目的是充分运行系统，验证系统各部件是否都能正常工作并完成所赋予的任务。这里所谓的系统不仅仅包括软件本身，而且还包括计算机硬件及其相关的外围设备、实际运行时大批量数据、非正常操作（如黑客攻击）等。通常意义上的系统测试包括压力测试、容量测试、性能测试、安全测试、容错测试等。这里先简单介绍一下它们的概念，使读者有一个整体的认识。

- **压力测试 (stress test)**: 也称为强度测试、负载测试。压力测试是模拟实际应用的软硬件环境及用户使用过程的系统负荷，长时间或超大负荷地运行测试软件，来测试被测系统的性能、可靠性、稳定性等。压力测试的目的就是在软件投入使用以前或软件负载达到极限以前，通过执行可重复的负载测试，了解系统可靠性、性能瓶颈等，以提高软件系统的可靠性、稳定性，减少系统的宕机时间和因此带来的损失。
- **容量测试 (capacity test)**: 预先分析出反映软件系统应用特征的某项指标的极限值，如某个 Web 站点可以支持多少个并发用户的访问量、网络在线会议系统的与会者人数。知道了系统的实际容量，如果不能满足要求，就应该寻求新的解决方案，以提高系统的容量。若一时没有新的解决方案，就有必要在产品发布说明书上明确这些容量的限制，避免引起软件产品使用上的纠纷。如果实际容量已满足要求，就能帮助用户建立对产品的信心。
- **性能测试 (performance test)**: 通过测试确定系统运行时的性能表现，如得到运行速度、响应时间、占有系统资源等方面的系统数据。对于那些实时或嵌入式系统，系统有时满足了功能要求，但未必能够满足性能要求，如某个网站可以被访问，而且可以提供预先设定的功能，但每打开一个页面都需要 1~2 分钟，用户不可忍受，其结果没有用户愿意使用这个网站所提供的服务。
- **安全测试 (security test)**: 检查系统对非法侵入的防范能力。安全测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。系统安全设计的准则是，使非法侵入的代价超过被保护信息的价值。
- **容错测试 (recovery test)**: 主要检查系统的容错能力。当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。容错测试首先要通过各种手段，让软件强制性地发生故障，然后验证系统是否能尽快恢复。对于自动恢复需验证重新初始化、检查点、数据恢复和重新启动等机制的正确性；对于人工干预的恢复系统，还需估测平均修复时间，确定是否在可接受的范围内。



### 6.3.2 回归测试

无论在进行系统测试还是功能测试,当发现一些严重的缺陷需要修正时,会构造一个新的软件包(full build)或新的软件补丁包(patch),然后进行测试。这时的测试不仅要验证被修复的软件缺陷是否真正被解决了,而且要保证以前所有运行正常的功能依旧保持正常,而不要受到这次修改的影响。不希望已发现的程序缺陷被修复了,但又出现新的软件缺陷,甚至更严重的缺陷也没有被发现,结果产品发布出去了,从而引起很大的问题。这就是本节要讨论的回归测试。

#### 1. 回归测试的目的

回归测试的目的是在程序有修改的情况下保证原有功能正常的一种测试策略和方法,因为这时的测试不需要进行全面测试,从头到尾测一遍,而是根据修改的情况进行有效测试。程序在发现严重软件缺陷要进行修改或版本升级要新增功能,这时需要对软件进行修改,修改后的程序要进行测试,要检验对软件所进行的修改是否正确,保证改动不会带来新的严重错误。这里所说的关于软件修改的正确性有两层含义:

- 所做的修改达到了预定的目的,如错误得到了改正,新功能得到了实现,能够适应新的运行环境等。
- 不影响软件原有功能的正确性。

在软件生命周期中的任何一个阶段,只要软件发生了改变,就可能给该软件带来新的问题。软件的改变可能是源于发现了缺陷并做了修改,也有可能是因为在集成或维护阶段加入了新的功能或增强原有的功能而引发。当软件中所含错误被发现时,如果错误跟踪与管理不够完善,就可能会遗漏对这些错误的修改;而开发者对错误理解的不够透彻,也可能导致所做的修改只修正了错误的外在表现,而没有修复错误本身,从而造成修改失败;修改还有可能产生副作用从而导致软件未被修改的部分产生新的问题,使本来工作正常的功能产生错误;同样,在有新代码加入软件的时候,除了新加入的代码中有可能含有错误外,新代码还有可能对原有的代码带来影响。因此,每当软件发生变化时,就必须重新测试现有的功能,以便确定修改是否达到了预期的目的,检查修改是否损害了原有的正常功能。同时,还需要补充新的测试用例来测试新的或被修改了的功能。为了验证修改的正确性及其影响就需要进行回归测试。

回归测试作为软件生命周期的一个组成部分,在整个软件测试工作量中占有很大的比重,软件开发的各个阶段都可能需要进行多次回归测试。在渐进和快速迭代开发中,新版本的连续发布使回归测试进行的更加频繁,而在极端编程方法中,更是要求每天都进行若干次回归测试。因此,通过选择正确的回归测试策略来改进回归测试的效率和有效性是非常有意义的。

#### 2. 回归测试的方法

在软件生命周期中,即使一个得到良好维护的测试用例库也可能变得相当大,使得每

次回归测试都重新运行完整的测试包变得不切实际,时间和成本的限制也不允许进行完全的测试,需要从测试用例库中选择有效的测试用例,构造一个缩减的测试用例组来完成回归测试。

回归测试的价值在于它是一个能够检测到回归错误的受控实验。当测试组选择缩减的回归测试时,有可能忽略了将发现揭示回归错误的测试用例,而错失了发现回归错误的机会。然而,如果采用了代码相依性分析等安全的缩减技术,就可以决定哪些测试用例可以被删除而不会影响回归测试的结果。

选择回归测试方法应该兼顾效率和有效性两个方面,下面有几种方法,在效率和有效性方面的侧重点是不同的。

- 再测试全部用例:选择测试用例库中的全部测试用例构成回归测试包,这是一种比较安全的方法,遗漏回归错误的风险最低,但测试成本最高。再测试全部用例几乎可以应用到任何情况下,基本上不需要进行用例分析和设计,但是随着开发工作的进展,测试用例不断增多而带来相当大的工作量,会受预算和进度的限制。
- 基于风险选择测试:基于一定的风险标准来从测试用例库中选择回归测试包。首先运行最重要的、关键的和可疑的测试,而跳过那些次要的、例外的测试用例或那些功能非常稳定的模块。运行那些次要用例即便发现缺陷,这些缺陷的严重性也较低。
- 基于操作剖面选择测试:如果测试用例是基于软件操作剖面开发的,测试用例的分布情况反映了系统的实际使用情况。回归测试所使用的测试用例个数可以由测试预算确定,回归测试可以优先选择那些针对最重要或最频繁使用功能的测试用例,释放和缓解最高级别的风险,有助于尽早发现那些对可靠性有最大影响的故障。
- 再测试修改的部分:当测试者对修改的局部有足够的信心时,可以通过相依性分析识别软件的修改情况并分析修改的影响,将回归测试局限于被改变的模块和它的接口上。通常,一个回归错误一定涉及被修改的或新加的代码。在允许的条件下,回归测试尽可能覆盖受到影响的部分。这种方法可以在一个给定的预算下最有效地提高系统可靠性,但需要丰富的经验和深入地代码分析。

综合运用多种测试技术是常见的,在回归测试中也不例外,测试者也可能希望采用多于一种回归测试策略来增强对测试结果的信心。不同的测试者可能会依据自己的经验和判断选择不同的回归测试技术和方法。

### 3. 回归测试的组织和实施

回归测试可遵循下述基本过程进行。

- (1) 识别出软件中被修改的部分。
- (2) 从原基线测试用例库 T 中,排除所有不再适用的测试用例,确定那些对新的软件版本依然有效的测试用例,其结果是建立一个新的基线测试用例库 T0。
- (3) 依据一定的策略从 T0 中选择测试用例测试被修改的软件。
- (4) 如果回归测试包不能达到所需的覆盖要求,必须补充新的测试用例使覆盖率达到

规定的要求,生成新的测试用例集 T1,用于测试 T0 无法充分测试的软件部分。

(5) 用 T1 执行修改后的软件。

第(2)和第(3)步测试验证修改是否破坏了现有的功能,第(4)和第(5)步测试验证修改工作本身。

## 6.4 压力测试、容量测试和性能测试

压力测试、容量测试和性能测试的测试目的虽然有所不同,但其手段和方法在一定程度上比较相似,通常会使用特定的测试工具,来模拟超常的数据量、负载等,监测系统的各项性能指标,如 CPU 和内存的使用情况、响应时间、数据传输量等。所以在这一节中一起来介绍。

### 6.4.1 压力测试

压力测试是在一种需要反常数量、频率或资源的方式下,执行可重复的负载测试,以检查程序对异常情况的抵抗能力,找出性能瓶颈。从本质上来说,测试者是想要破坏程序,难怪在进行压力测试时常常问自己:“我们能够将系统折腾到什么程度而又不会出错?”这种系统折腾,就是对异常情况的检测。异常情况主要指的是峰值(瞬间使用高峰)、大量数据的处理能力、长时间运行等情况。压力测试总是迫使系统在异常的资源配置下运行,例如以下几种情况。

- 当中断的正常频率为每秒 1~2 个时,运行每秒产生 10 个中断的测试用例。
- 定量地增长数据输入率,检查对数据处理的反应能力。
- 运行需要最大存储空间(或其他资源)的测试用例。
- 运行可能导致虚存操作系统崩溃或大量数据对磁盘进行存取操作的测试用例等。

#### 1. 测试压力估算

根据产品说明书的设计要求或以往版本的实际运行经验对测试压力进行估算,给出合理的估算结果。例如,单台服务器实际使用时一般只有 100 个并发用户,但在某一时间段的用户峰值可达到 500 个。那么事先预测要求的压力值为 500 个用户的 1.5~2 倍。而且要考虑到每个用户的实际操作所产生的事务处理和数据量。如果产品说明书已说明最大设计容量,则最大设计容量为最大压力值。

#### 2. 测试环境准备

测试环境准备包括硬件环境(服务器、客户机等)、网络环境(网络通信协议、带宽等)、测试程序(能正确模拟客户端的操作)、数据准备等。

分析压力测试中系统容易出现瓶颈的地方,从而有目的地调整测试策略或测试环境,使压力测试结果真实地反映出软件的性能。例如,服务器的硬件限制、数据库的访问性能

设置等常常会成为制约软件性能的重要因素，但这些因素显然不是用户最关心的，在测试之前就要通过一些设置把这些因素的影响调至最低。

- 压力稳定性测试：在选定的压力值下，持续运行 24 小时以上进行稳定性测试。客户端通常由测试工具模拟真实用户不停地各种操作。监视服务器和真实客户端的必要性能指标。通过压力测试的标准使各项性能指标在指定范围内无内存泄漏、无系统崩溃、无功能性故障等。
- 破坏性加压测试：在压力稳定性测试中可能会出现一些问题，如系统性能明显降低，但仅从以上的测试中很难暴露出真实的原因。通过不断加压的破坏性手段，往往能快速造成系统的崩溃或让问题明显地暴露出来。

### 3. 问题的分析

在压力测试中通常采用的是黑盒测试方法，测试人员很难对出现的问题进行准确的定位。报告中只有现象会造成调试修改的困难，而开发人员又没有相应的环境和时间去重现问题，所以适当的分析和详细的记录是十分重要的。

- 查看服务器上的进程及相应的日志文件可能立刻找到问题的关键（如某个进程的崩溃）。优秀的程序员会给程序加上保护、跟踪机制及错误处理机制，备份日志文件以供参考。
- 查看监视系统性能的日志文件，找出问题出现的关键时间。此时的联机用户数量、系统状态等也是很有价值的参考材料。
- 检查测试运行参数，进行适当调整重新测试，看看是否能够再现问题。
- 对问题进行分解，屏蔽某些因素或功能，试着重现问题。例如，客户端与服务器有三种连接方式：TCP、HTTP、HTTPS，则只保留 HTTP 或 TCP 连接方式，如问题仍然存在，也许是代理服务器或网关等造成的，把 MS 代理换成 SQLID 代理再次检测。

### 4. 累积效应

有些测试人员在压力测试中喜欢让整个系统重启（如服务器 reboot），以确保后续的测试能在一个“干净”的环境中进行。这样确实有利于问题的分析，但不是一个好的习惯，因为这样往往会忽略掉累积效应，使得一些缺陷无法被发现。有些问题的表现并不明显，但日积月累就会造成严重问题，特别是服务器端的压力测试。例如某进程每次调用时申请占用的内存在运行完毕时并没有完全释放，平常的测试中无法发现，但最终可能导致系统的崩溃。

## 6.4.2 容量测试

容量测试目的是通过测试预先分析出反映软件系统应用特征的某项指标极限值（如最大并发用户数、数据库记录数等），系统在其极限值状态下还能保持主要功能正常运行。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。例如，如果测

试对象正在为生成一份报表而处理一组数据库记录,那么容量测试就会使用一个具有十几万条、甚至几百万条记录的大型测试数据库,检验该软件是否能正常运行并生成正确的报表。

容量测试有时候进行一些组合条件下的测试,如核实测试对象在以下高容量条件下能否正常运行:

- 连接或模拟了最大(实际或实际允许)数量的客户机。
- 所有客户机在长时间内执行相同的、可能性能不稳定的重要业务功能。
- 已达到最大的数据库大小(实际的或按比例缩放的),而且同时执行多个查询或报表事务。

容量测试的完成标准可以定义为:所计划的测试已全部执行,而且达到或超出指定的系统限制时没有出现任何软件故障。

当然需要注意,不能简单地说在某一标准配置服务器上运行某软件的容量是多少。选用不同的加载策略可以反映不同状况下的容量。举一个简单的例子,网上聊天室软件的容量是多少?在一个聊天室内有1000个用户,和100个聊天室每个聊天室内有10用户。同样的1000个用户,在性能表现上可能会出现很大的不同,在服务器端数据处理量、传输量是截然不同的。在更复杂的系统内,就需要分更多种情况提供相应的容量数据作为参考。

对软件容量的测试,能让软件开发商或用户了解该软件系统的承载能力或提供服务的能力,如某个电子商务网站所能承受同时进行交易或结算的联机用户数。知道了系统的实际容量,如果不能满足设计要求,就应该寻求新的技术解决方案,以提高系统的容量。有了对软件负载的准确预测,不仅能对软件系统在实际使用中的性能状况充满信心,同时也可以帮助用户经济地规划应用系统,优化系统和网络配置。

### 6.4.3 性能测试

对于那些实时和嵌入式系统,软件部分即使满足功能要求,也未必能够满足性能要求,虽然从单元测试起,每一测试步骤都包含性能测试,但只有当系统真正集成之后,在真实环境中才能全面、可靠地测试系统性能,系统性能测试就是为了完成这一任务。

性能测试经常和压力测试一起进行,而且常常需要硬件和软件测试设备。这就是说,在一种苛刻的环境中衡量资源(比如处理器周期、内存)的使用常常是必要的。外部的测试设备可以监测执行的间歇,当出现情况(比如中断)时记录下来。通过对系统的检测,测试者可以发现导致效率降低和系统故障的情况。

为记录性能需要在系统中安装必要的测量仪表,或是为度量性能设置相应的软件或程序。大多数性能测试,特别是基于C/S结构的应用软件的性能测试只有借助于测试工具才能完成,同时,也需要测试工程师灵活地运用才能让测试工具充分发挥作用。

例如,如果对一个C/S结构的网络实时在线培训系统软件进行测试。系统性能测试的主要焦点在各种情况下服务器的CPU及内存的使用和占用率、客户端的各种响应时间。先让客户端与服务器全部采用HTTP的连接方式,建立表6-5,进行测试并取得数据。

表 6-5 HTTP 连接性能表

HTTP	1×5	1×50	1×100	1×250	1×500	1×600	1×700	1×800	1×900	...	10×5	50×5
CPU (%)	1.2	2.5	4.5	11	20	20	28	23	25	...	4	24
物理内存 (MB)	55	45	38	38	32	48	75	46	37	...	178	232
虚拟内存 (MB)	836	841	831	855	865	858	867	874	884	...	871	1, 472
加入时间 (s)	12.04	12.14	11.6	15.48	126.1	104.76	168.1	123.7	218.11	...	12.01	9.17
建会时间 (s)	12.01	11.35	12.38	13.32	13.63	14.06	16.35	14.98	17.68	...	10.9	11.39
延时 (s) ...	...	...	...	...	...	...	...	...	...	...	...	...
断开时间 (s)	8.58	9.11	7.94	9.09	8.26	8.35	8.46	11.41	11.1	...	8.79	8.22

测试过程中,压力的不断增加在系统性能上的表现越来越明显,加载过程中每到一个测试点时注意让系统平稳地运行一段时间后再获取数据,否则加压的操作可能会影响数据的可靠性。表中 1×5 表示的是 1 个课程,5 个联机用户;10×5 表示 10 个课程,每个课程有 5 个在线用户。在表中,同样是 250 个用户,1×250 与 50×5 的性能表现差别很大。也可以获得 TCP 等连接方式下的数据,然后根据这些数据生成各项指标的性能曲线图,如图 6-8 所示为 CPU 使用率的对照曲线图。

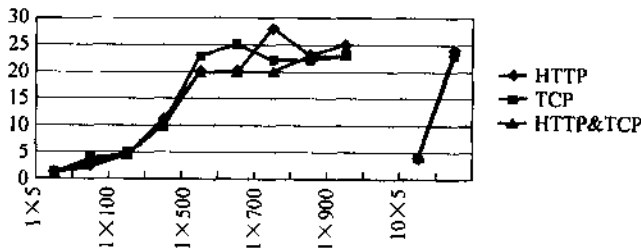


图 6-8 CPU 使用率的对照曲线图

根据这些图表能很清楚地了解系统的性能状况。在什么样的压力下系统达到最佳状况,什么压力是系统的极限。实际上在做性能测试的同时也完成了部分容量测试。

综上所述,压力测试、容量测试和性能测试的手段和方法很相似,有时可以交织在一起进行测试。压力测试的重点在于发现功能性测试所不易发现的系统方面的缺陷;容量测试和性能测试更着力于提供性能与容量方面的数据,以供软件开发商参考,改进或进行广告宣传。

## 6.5 安全性、可靠性和容错性测试

安全性测试、可靠性测试和容错性测试的测试目的不同,其手段和方法也不相同,但它们都属于系统测试的范畴,有一定的联系,如软件可靠性测试通常包括了安全性的要求。而且,安全性测试、可靠性测试和容错性测试的技术比较深、实施起来比较难,但在计算机应用系统中其作用越来越大。

## 6.5.1 安全性测试

根据 ISO8402 的定义, 安全性是“使伤害或损害的风险限制在可接受的水平内”。所以直观地说, 软件的安全性是软件的一种内在的属性。安全性的英文术语是 **safety**, 另一个英文术语 **security**, 也有安全的含义, 但是它主要是指文件、数据、资料的保密问题。

软件安全性和可靠性有非常紧密的联系, 安全事故是危害度最大的失效事件, 因此软件可靠性要求通常包括了安全性的要求。但是软件的可靠性不能完全取代软件的安全性, 因为安全性要求包括在非正常条件下不发生安全事故的能力。

安全性测试是检查系统对非法侵入的防范能力。安全测试期间, 测试人员假扮非法入侵者, 采用各种办法试图突破防线。例如:

- 想方设法截取或破译口令。
- 专门开发软件来破坏系统的保护机制。
- 故意导致系统失败, 企图趁恢复之机非法进入。
- 试图通过浏览非保密数据, 推导所需信息等。

理论上讲, 只要有足够的时间和资源, 没有无法进入的系统。因此系统安全设计的准则是使非法侵入的代价超过被保护信息的价值, 此时非法入侵者已无利可图。

### 1. 两种级别的安全性

安全性一般分为两个层次, 即应用程序级别的安全性和系统级别的安全性, 它们的关系如下:

- 应用程序级别的安全性, 包括对数据或业务功能的访问; 系统级别的安全性, 包括对系统的登录或远程访问。
- 应用程序级别的安全性可确保在预期的安全性情况下, 操作者只能访问特定的功能或用例, 或者只能访问有限的的数据。例如, 某财务系统可能会允许所有人输入数据, 创建新账户, 但只有管理员才能删除这些数据或账户。如果具有数据级别的安全性, 测试就可确保“用户类型一”能够看到所有客户消息(包括财务数据), 而“用户类型二”只能看见同一客户的统计数据。
- 系统级别的安全性可确保只有具备系统访问权限的用户才能访问应用程序, 而且只能通过相应的网关来访问。

### 2. 测试目标

应用程序级别的安全性: 核实操作者只能访问其所属用户类型已被授权访问的那些功能或数据。

系统级别的安全性: 核实只有具备系统和应用程序访问权限的操作者才能访问系统和应用程序。

### 3. 测试范围

确定并列各用户类型及其被授权访问的功能或数据。为各用户类型创建测试，并通过创建各用户类型所特有的事务来核实其权限，修改用户类型并为相同的用户重新运行测试。对于每种用户类型，确保正确地提供或拒绝这些附加的功能或数据。

### 4. 完成标准

各种已知的操作类型都可访问相应的功能或数据，而且所有事务都按照预期的方式运行，并在先前的应用程序功能测试中运行了所有的事务。

## 6.5.2 可靠性测试

可靠性 (reliability) 是产品在规定的条件下和规定的时间内完成规定功能的能力，它的概率度量称为可靠度。软件可靠性是软件系统的固有特性之一，它表明了一个软件系统按照用户的要求和设计目标，执行其功能的可靠程度。软件可靠性与软件缺陷有关，也与系统输入和系统使用有关。理论上说，可靠的软件系统应该是正确、完整、一致和健壮的。但是实际上任何软件都不可能达到百分之百的正确，而且也无法精确度量。一般情况下，只能通过对软件系统进行测试来度量其可靠性。

对软件可靠性的定义：“软件可靠性是软件系统在规定的时间内及规定的环境条件下，完成规定功能的能力”。根据这个定义，软件可靠性主要包含以下三个要素：

- 规定的时间：软件可靠性只是体现在其运行阶段，所以将“运行时间”作为“规定的时间”度量。“运行时间”包括软件系统运行后工作与挂起（开启但空闲）的累计时间。由于软件运行的环境与程序路径选取的随机性，软件的失效为随机事件，所以运行时间属于随机变量。
- 规定的环境条件：环境条件指软件的运行环境。它涉及软件系统运行时所需的各种支持要素，如支持硬件、操作系统、其他支持软件、输入数据格式和范围以及操作规程等。不同的环境条件下软件的可靠性是不同的。具体地说，规定的环境条件主要是描述软件系统运行时计算机的配置情况以及对输入数据的要求，并假定其他一切因素都是理想的。有了明确规定的环境条件，才可以有效判断软件失效的责任在用户方还是研制方。
- 规定的功能：软件可靠性还与规定的任务和功能有关。由于要完成的任务不同，软件的运行剖面会有所区别，调用的子模块就不同（即程序路径选择不同），其可靠性也就可能不同。所以要准确度量软件系统的可靠性必须首先明确它的任务和功能。

### 1. 可靠性测试方法

在第3章已经介绍了可靠性的数据收集、可靠性的评估报告。可靠性数据收集、保存是可靠性测试的基础。在可靠性测试中，可以考虑进行“强化输入”，即输入比正常输入更



恶劣（合理程度的恶劣）的数据。如果软件在强化输入下可靠，就能说明比正规输入下可靠得多。同时为了获得更多的可靠性数据，应该采用多台计算机同时运行软件，以增加累计运行时间。

## 2. 可靠性测试结果的评估

软件系统的可靠性是系统最重要的质量指标。ISO9000 国际质量标准（ISO/IEC 9126-1991）规定，软件产品的可靠性含义是：在规定的一段时间和条件下，软件能维持其性能水平的能力有关的一组属性，可用成熟性、容错性、易恢复性三个基本子特性来度量。

成熟性度量可以通过错误发现率 DDP（defect detection percentage）来表现。在测试中查找出来的错误越多，实际应用中出错的机会就越小，软件也就越成熟。

$$DDP = \text{测试发现的错误数量} / \text{已知的全部错误数量}$$

已知的全部错误数量是测试已发现的错误数量加上可能会发现的错误数量之和。

容错性的测试在下面一节介绍。恢复性的测试先设法（模拟）使系统崩溃、失效，然后计算其系统和数据恢复的时间来评估易恢复性。

## 6.5.3 容错性测试

容错性测试是检查软件在异常条件下自身是否具有防护性的措施或某种灾难性恢复的手段。当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。容错性测试包括两个方面：

- 输入异常数据或进行异常操作，以检验系统的保护性。如果系统的容错性好，系统只给出提示或内部消化掉，而不会导致系统出错甚至崩溃。
- 灾难恢复性测试。通过各种手段，让软件强制性地发生故障，然后验证系统已保存的用户数据是否丢失，系统和数据是否能尽快恢复。

对于自动恢复需验证重新初始化、检查点、数据恢复和重新启动等机制的正确性；对于人工干预的恢复系统，还需估测平均修复时间，确定其是否在可接受的范围内。容错性好的软件能确保系统不发生无法意料事故。

从容错性测试的概念可以看出，当软件出现故障时如何进行故障的转移与恢复有用的数据是十分重要的。

### 1. 故障转移与数据恢复

故障转移是确保测试对象在出现故障时能成功完成故障的转移，并能从导致意外数据损失或数据完整性破坏的各种硬件、软件和网络故障中恢复。数据恢复可确保：对于必须持续运行的系统，一旦发生故障，备用系统将不失时机地“顶替”发生故障的系统，以避免丢失任何数据或事务。容错测试是一种对抗性的测试过程。在这种测试中，将把应用程序或系统置于（模拟的）异常条件下，以产生故障。例如设备输入/输出（I/O）故障或无效的数据库指针和关键字等。然后调用恢复进程并监测和检查应用程序和系统，核实系统和数据已得到了正确的恢复。

## 2. 测试目标

确保恢复进程将数据库、应用程序和系统正确地恢复到预期的已知状态。测试中将包括以下各种情况：

- 客户机断电、服务器断电。
- 通过网络服务器产生的通信中断或控制器被中断。
- 断电或与控制器的通信中断周期未完成（数据过滤进程被中断，数据同步进程被中断）。
- 数据库指针或关键字无效，数据库中的数据元素无效或遭到破坏。

## 3. 测试范围

应该使用为功能和业务周期测试创建的测试来创建一系列的事务。一旦达到预期的测试起点，就应该分别执行或模拟以下操作：

- 往软盘保存时，不插入软盘，或将软盘加写保护。
- 不接打印机，但进行打印操作。
- 客户机断电和服务器断电。
- 网络通信中断，如可以断开通信线路的连接，关闭网络服务器或路由器的电源。
- 控制器被中断、断电或与控制器的通信中断，模拟与一个或多个控制器及设备的通信，或实际取消这种通信。

一旦实现了上述情况（或模拟情况），就应该执行其他事务。而且一旦达到第二个测试点状态，就应调用恢复过程。在测试不完整的周期时，所使用的技术与上述技术相同，只不过应异常终止或提前终止数据库进程本身。对以下情况的测试需要达到一个已知的数据库状态。当破坏若干个数据库字段、指针和关键字时，应该以手工方式在数据库中（通过数据库工具）直接进行。其他事务应该通过使用“应用程序功能测试”和“业务周期测试”中的测试来执行，并且应执行完整的周期。

## 4. 完成标准

在所有上述情况中，应用程序、数据库和系统应该在恢复过程完成时立即返回到一个已知的预期状态。此状态包括仅限于已知损坏的字段、指针或关键字范围内的数据损坏，以及表明进程或事务因中断而未被完成的报表。

## 5. 需考虑的特殊事项

恢复测试会给其他操作带来许多的麻烦。断开缆线连接的方法（模拟断电或通信中断）可能并不可取或不可行。所以，可能需要采用其他方法，例如诊断性软件工具。需要系统（或计算机操作）、数据库和网络组中的资源。这些测试应该在工作时间之外或在一台独立的计算机上运行。

## 小结

本章首先从系统集成不同模式的比较以及自底向上、自顶向下、混合各自优缺点比较说明集成测试的必要性，以及持续集成的好处。在系统测试前，为了确保每一个模块功能的实现，要进行功能测试。在系统测试中：

- 为了测试系统的限制和故障恢复能力，要进行压力测试、容量测试与性能测试。
- 为了保证加入模块没有引进新的错误，需要进行回归测试。
- 为了确保使用者信息的安全以及和软件自身的安全需要进行安全性测试。
- 为了保证软件的健壮性要有可靠性测试与容错性测试。

从各个不同的角度实施相应的测试策略来确保软件的质量。

通过本章的学习，应该知道集成测试在整个软件测试中的重要性，因为单元测试中的模块虽然个个能用，但不能保证经过两三次甚至更多次的集成后不会出错误，而在集成中比较重要的就是模块之间的接口，所以对接口数据进行大量测试相当必要。另外，软件中可能有一些关键的模块，它们的存在影响其他模块的运行，它们一般都具有下述一或多个特征：对应几条需求；具有高层控制功能；复杂、易出错；有特殊的性能要求。所以对于关键模块应尽早测试，并反复进行回归测试。

对测试可以用表 6-6 做一个总结。

表 6-6 从三个质量纬度看系统测试

质量维度	测试类型
可靠性	<p><b>完整性测试：</b>侧重于评估测试对象的强壮性（防止失败的能力），语言、语法的技术兼容性以及资源利用率的测试。该测试针对不同的测试对象实施和执行，包括单元和已集成单元</p> <p><b>结构测试：</b>侧重于评估测试目标是否符合设计和构造的测试。通常对基于 Web 的应用程序执行该测试，以确保所有链接都已连接、显示正确的内容以及没有孤立的内容</p>
功能	<p><b>配置测试：</b>侧重于确保测试对象在不同的硬件/软件配置上按预期运行的测试。该测试还可以作为系统性能测试来实施</p> <p><b>功能测试：</b>侧重于核实测试对象按计划运行，提供需求的服务、方法或用例的测试。该测试针对不同的测试对象实施和执行，包括单元、已集成单元、应用程序和系统</p> <p><b>安装测试：</b>侧重于确保测试对象在不同的硬件/软件配置上，以及在不同的条件下（磁盘空间不足或电源中断）按预期安装的测试。该测试针对不同的应用程序和系统实施和执行</p> <p><b>安全测试：</b>侧重于确保只有预期的用户才可以访问测试对象、数据（或系统）的测试。该测试针对多种测试对象实施和执行</p> <p><b>容量测试：</b>侧重于核实测试对象对于大量数据（输入和输出或驻留在数据库内）的处理能力的测试。容量测试包括多种测试策略，如创建返回整个数据库内容的查询；或者对查询设置很多限制，以至不返回数据；或者返回每个字段中最大数据量的数据条目</p>

续表

质量维度	测 试 类 型
性能	<b>基准测试：</b> 一种性能测试，该测试将比较（新的或未知的）测试对象与已知的参照负载和系统的性能。
	<b>竞争测试：</b> 侧重于核实测试对象对于多个用户对相同资源（数据记录、内存等）的请求处理是否可以接受的测试
	<b>负载测试：</b> 一种性能测试，用于在测试系统保持不变的情况下，核实和评估系统在不同负载下操作极限的可接受性，评测包括负载和响应时间的特征。如果系统结合了分布式构架或负载平衡方法，将执行特殊的测试以确保分布和负载平衡方法能够正常工作
	<b>性能曲线：</b> 在该测试中，将监测测试对象的计时配置文件，包括执行流、数据访问、函数和系统调用，以确定并解决性能瓶颈和低效流程
	<b>强度测试：</b> 一种性能测试，侧重于确保系统在遇到异常条件时按预期运行。系统面对的工作强度可以包括过人的工作量、不充足的内存、不可用的服务/硬件或过低的共享资源

思考题

- 1. 系统集成测试常见有哪几种不同模式？各自优缺点是什么？
- 2. 什么是桩模块？什么是驱动模块？在采用什么样集成方法时能用到它们？
- 3. 从自己的角度出发，你认为在进行集成测试时需要着重注意什么？
- 4. 黑盒测试常用哪几种方法？各自是从什么角度出发来进行测试的？
- 5. 某一 C 语言版本中规定：“标识符是以字母或下划线开头，后跟字母、数字或下划线的任意组合构成有效字符为 16 个；且标识符不能是保留字。”试用等价分类法设计测试用例。
- 6. 集成测试、单元测试与系统测试的联系与区别是什么？
- 7. 如果给你一个网站进行测试，利用现在所学的知识可用到哪些测试方法？因为是针对网站测试，使用每种测试方法应该注意哪些问题？

# 第7章 验收测试

验收测试（acceptance test）是软件产品完成了功能测试和系统测试之后，在产品发布之前所进行的软件测试活动，它是技术测试的最后一个阶段，通过了验收测试，产品就会进入发布阶段。验收测试一般根据产品规格说明书严格检查产品，逐行逐字地对照说明书上对软件产品所做出的各方面要求，确保所开发的软件产品符合用户的各项要求。验收测试检验产品和产品规格说明书（包括软件开发的技术合同）的一致性，同时考虑用户的实际使用情况。验收测试和确认测试比较接近，但两者并不相同。确认测试是从质量的角度，依据国家相应标准（如附录 C 及 GB / T16260-1996《信息技术 软件产品评价 质量特性及其使用指南》）在功能、性能、可靠性、易用性等方面做全面的质量检测。两者提交的报告也不同。

## 7.1 验收测试的过程和主要内容

如前所述，验收测试在功能测试和系统测试之后进行，所以验收测试进行的前提条件是系统或软件产品已通过了系统测试，而且要求软件系统必须在真实的环境下运行。验收测试的内容和过程又是什么呢？

### 1. 测试内容

验收测试用来验证系统是否达到了用户需求规格说明书（可能包括项目或产品验收准则）中的要求，测试希望尽可能地发现软件中存留的缺陷，从而为软件进一步改善提供帮助，并保证系统或软件产品最终被用户接受。其主要包括易用性测试、兼容性测试、安装测试、文档（如用户手册、操作手册等）测试等几个方面的内容。

### 2. 测试步骤

(1) 测试计划在需求分析阶段建立，主要了解软件功能和性能要求、软硬件环境要求等，并特别要了解软件的质量要求和验收要求。根据软件需求和验收要求编制测试计划，制定需测试的测试项，制定测试策略及验收通过准则，并经过客户参与的计划评审。

(2) 建立测试环境。根据验收测试计划、项目或产品验收准则完成测试用例的设计，并经过评审。

(3) 准备测试数据、执行测试用例，记录测试结果。

(4) 分析测试结果。根据验收通过准则分析测试结果，作出验收是否通过及测试评价。通常会有四种情况：

- 测试项目通过。

- 测试项目没有通过，并且不存在变通方法，需要作很大的修改。
- 测试项目没有通过，但存在变通方法，在维护后期或下一个版本改进。
- 测试项目无法评估或者无法给出完整的评估。此时必须给出原因。如果是因为该测试项目没有说清楚，应该修改测试计划。

(5) 提交测试报告。根据产品设计说明书、详细设计说明书、验收测试结果和发现的错误信息，评价系统的设计与实现，最终通过验收测试报告和缺陷报告等体现出来。

### 3. 验收测试完成标准

- 完全执行了验收测试计划中的每个测试用例。
- 在验收测试中发现的错误已经得到修改并且通过了测试。
- 完成软件验收测试报告。

### 4. 注意事项

- 必须编写正式的、单独的验收测试计划。
- 验收测试必须在实际的用户运行环境中进行。
- 由用户和测试部门共同执行比较好。如果是公司自我开发的产品，由测试人员和产品设计部门、市场部门等共同进行，可能还包括技术支持、产品培训部门。

## 7.2 产品规格说明书的验证

产品规格说明书是综合客户的需求以及没有提出但必须实现的需求，定义产品的目的、适应范围，有哪些功能，界面表现形式和外观等的文档。通常是用文字和图形来描述产品的 Word、PDF 或 HTML 文档，其格式没有特别的要求。

### 7.2.1 产品规格说明书的审核

产品规格说明书的审核不是验收测试阶段的主要任务，它应该在整个产品生命周期的初期，即产品说明书生成后就进行。应该在产品设计前基本完成，以后在测试过程中尽量不做改动，如果确实需要改动，要通过一定的严格流程去完善。产品规格说明书很重要，是测试的标准，没有它，会让测试工作无从着手，谁都不清楚产品应该是什么样，最终会是什么样，从而使产品开发陷入无尽的困难之中。

产品说明书的特性决定了对其审核的重要性。经验证明充分的审核能排除约 60% 的错误，而在此阶段找出未来软件在设计和需求上的缺陷极有可能为项目节省大笔的开销。审核是为了查验根本性的问题，避免疏忽和遗漏。通常要注意以下几点：

- 从客户的角度和立场进行审核工作。
- 检验应用标准的正确性，不要和行业标准相抵触。
- 审查、研究同类产品。

- 验证产品说明书的完整性、准确性、一致性、合理性等特性。

因为软件开发周期较长，而整个行业变化很快，种种原因会导致新的功能不断增加，产品说明书不断地更新。这是软件开发人员痛恨而又不得不面对的难题。例如，竞争对手推出新的吸引人的功能，客户提出新的要求以适应新的形势，原先的设计很难或无法实现等。

软件测试人员必须要想到产品说明书可能改变，未曾计划的特性会增加，经过测试并报告软件缺陷的特性可能发生变化，甚至被删除。要针对产品说明书的变化，不断地验证并灵活地调整测试计划和测试方法。

## 7.2.2 产品说明书的验证

产品说明书的验证大部分属功能性测试的范畴。早在集成测试完成后，系统测试开始前，我们就开始对软件系统进行功能性测试。但早期的功能性测试旨在验证产品说明书定义的功能，以保证后续测试的进行，并尽早地发现问题。

在验收测试阶段，产品说明书的验证将更加严格和彻底地进行。但大多数情况下，测试人员并不能得到完美的产品说明书，尽管它已经通过了审核。由于审核上的缺陷、计划的变更、新特性的追加等因素，都会导致产品说明书不得不作相应更改。所以测试人员不仅要根据产品说明书的每一个特性导出测试用例，还要针对上述因素进行分析，设计测试用例，确保产品说明书验证的完整性。

测试人员应根据产品说明书，逐行逐句地验证其中的每一项特性。用英语表示为“Line by Line, Word by Word”，更加贴切地形容出对验证每一个特性的严格和彻底。并在验证结束时提交基于产品说明书的验证报告。

验证报告的内容应包括：

- 已经实现的特性标识为通过。
- 特性没有实现，报告 bug 并在报告中体现。
- 特性基本实现，但与产品说明书中内容不相一致，报告 bug 并在报告中体现。
- 特性基本实现，但存在一些问题或错误。

## 7.3 用户界面和可用性测试

用于软件程序交互的方式称为用户界面（UI, User Interface）。与早期的软件相比，我们现在使用的个人计算机都有复杂的图形用户界面（GUI）。虽然 UI 各不相同，但是从技术上讲，它们与计算机进行同样的交互：提供输入和接受输出。

许多产品都应用人体工程学的研究成果，使产品更具人性化，其使用更加灵活、舒适。软件产品也是一样，应以软件的最终使用者——客户为出发点。好的用户界面包括 7 个要素：符合标准和规范、直观性、一致性、灵活性、舒适性、正确性、实用性。

### 1. 符合标准和规范

软件在现有的平台上运行,通常标准是已经确立的(如 Mac 或者 Windows)。这些标准和规范是由大量正式测试、经验、技巧和错误得出的方便用户的规则。如软件应该有什么样的外观,何时使用复选框,何时使用单选按钮,何时使用提示信息、警告信息或者严重警告信息等,如图 7-1 所示。



图 7-1 Windows 的三种信息使用方式

由于多数用户已经熟悉并接受了这些标准和规范,或已经认同了这些信息所代表的意义,所以如果用“提示信息”代表严重警告,很难引起用户的重视,可能会被随手关闭,而且造成严重后果后,用户自己可能还不知道,这样自然得不到用户的认同。测试人员就应该将此类问题报告为 bug。如果软件在某一个平台上运行,就需要把该平台的标准和规范作为产品说明书的补充内容,在建立测试案例时和产品说明书一样作为依据。

平台也可能没有标准,也许测试的软件本身就是平台。那么软件设计者就应创立一套标准,贯穿于整个软件的设计开发过程,保持软件与行业标准、规范或约定相一致。

### 2. 直观性

考虑用户界面的直观性,应首先了解所需的功能,期待的响应应该明显,并在预期的地方出现。例如,执行结果已经显示出来,但因其不明显,客户使用时还在焦急地等待结果的出现。其次要考虑用户界面的组织和布局是否合理,界面是否洁净、不拥挤,以及是否有多余的功能,是否太复杂而难以掌握等因素。有一个例子,新软件中一个非关键的图标用了软件编程中常用的术语缩写,开发人员和测试人员往往因为熟悉而忽略,但真正的用户很难理解其含义,从而会产生猜测影响其直观性。

### 3. 一致性

包括软件本身的一致性,以及软件与其他软件的一致性。字体是否一致、界面的各元素风格是否一致是比较容易判定的。另外的一致性问题的通常体现在平台的标准和规范上。用户习惯于将某一程序的操作方式带到另一个程序中使用。例如,在 Windows 平台上客户已经习惯用 Ctrl+C 键表示复制操作,而在软件中将复制操作的快捷键定义为其其他键必定会给用户造成不适应感。如果在同一软件中不同的地方做了不同的定义那更是一件糟糕的事情。

### 4. 灵活性

用户喜欢可以灵活选择的软件,软件可以选择不同的状态和方式完成相应的功能。但灵活性也可能发展为复杂性,多种状态之间的转换,太多状态和方式的选择增加的不仅仅是用户理解和掌握的难度,而且增加了编程的难度,更增加了软件测试人员的工作量。图



7-2 中 Windows 计算器程序两种方式：标准型和科学型，充分体现了灵活性。



图 7-2 Windows 计算器程序的灵活性

### 5. 舒适性

舒适性的定义是含糊的。人们对舒适的理解各不相同，总体上说恰当的表现，合理的安排，必要的提示或更正能力都是要考虑的因素。

Windows 的 UNDO/REDO 特性让用户觉得方便，图 7-3 中所示的状态信息让用户清楚目前的工作状态，左手鼠标的设置给惯用左手的人带来便利也为右手十分劳累时提供了另一种使用方法。



图 7-3 复制文件的状态

### 6. 正确性

正确性的问题一般都很明显，比较容易发现。通常我们得注意是否有多余或遗漏的功能，功能是否被正确地实现，语言或拼写是否无误，在不同媒体上的表现是否一致，所有界面元素的状态是否都正确无误等。例如，根据用户的权限自动屏蔽某些功能，将密码输入内容显示为\*号。

### 7. 实用性

实用性不是指软件本身是否实用，而是指具体特性是否实用。在产品说明书的审查、准备测试、实际测试等各阶段都应考虑具体特性对软件是否具有实际价值，是否有助于用户执行软件设计的功能。如果认为没有必要，就要研究其存在于软件中的原因。无用的功能只会增加程序的复杂度，产生不必要的软件缺陷。

在大型软件的开发或周期较长经过几次反复修改的软件开发中容易产生一些没有实用性的功能。例如，由于某项功能的更改，可能导致原先设计界面上的图标或按钮没有存在的意义，也可能导致传输一些无用的参数、产生一些无用的数据。

软件易用性测试没有一个具体量化的指标，主观性较强。当前面的7个要素处理好了，易用的属性自然就好了。如果界面清晰美观，各元素布置合理，符合常用软件的标准和规范，用户能够在不需要其他帮助的情况下完成各项主要功能，我们就认为软件达到了易用性测试的标准。

## 7.4 兼容性测试

软件兼容性测试是指验证软件之间是否能正确地交互和共享信息。交互可以是同时运行于同一台计算机上，或在相隔甚远的不同计算机上的两个程序之间进行。

软件设计要求与何种平台（操作系统、Web 浏览器或者操作环境）和应用软件保持兼容？如果要测试的软件本身就是一个平台，那么设计要求哪些应用程序可以在其上运行？应该遵守何种软件之间的标准和规范？软件使用何种数据与其他平台和软件交互和共享信息？

以上问题实际上是程序管理或市场定位的任务，在接受兼容性测试任务时，应详细了解产品说明书中的有关内容并和相关人员进行沟通。从项目管理的角度出发，使平台清单在满足客户要求的前提下尽可能小是十分重要的，否则将会给编码和测试带来巨大的工作量。并不是所有的平台均需兼容，现在在 Windows 2000 下开发的应用程序，基本上不必考虑兼容 Windows 3.x。我们经常在软件包装、安装说明书或启动界面上看到一些类似的通告：“Work best with Netscape 4.0”、“Requires Windows 95 or greater”等。

兼容性包括与硬件兼容、软件之间兼容、数据之间兼容，测试应该按这三部分展开。

### 1. 向前和向后兼容

向后兼容是指可以使用软件的以前版本；向前兼容指的是可以使用软件的未来版本。例如，在 Windows 98 下开发的字处理软件，是否能够向后兼容以前在 Windows 95，Windows 3.1，甚至 MS-DOS 下运行的字处理软件所有版本的文件格式。而向前兼容指的是 Windows 2000，甚至未来的新版本。

### 2. 多版本的测试

当前流行的操作系统，已经有数百万个应用程序在上面运行。现在程序员修复了大量软件缺陷，改善了性能，并增加了许多有用的新特性。新操作系统的目标是百分之百兼容那数百万个应用程序。这样一个庞大而又艰巨的任务，需要对所有可能的软件组合等价分配，验证软件之间正确交互的最小有效集合。

通常我们的做法是：

- 将软件分类。例如，字处理、电子表格、数据库、图形处理、游戏等。

- 按软件的流行程度选择较流行的软件。
- 按年份选取一定年份内的程序和版本。

### 3. 一个典型的例子

每一个浏览器版本支持的特性都有细微的差别，在不同操作系统上的表现也不一样。一个网站可能在某个浏览器的某个版本上表现极佳，但是在另一种环境中就存在许多问题，甚至无法显示。

程序员可以选择只使用最普通的特性，以便在所有浏览器中显示同样的效果，也可以选择为每一个浏览器编写专用代码，使站点以最佳方式工作。浏览器的插件可以获得音频和视频播放功能。浏览器自身有各种设置选项（安全性等）。在不同的平台上屏幕分辨率和颜色模式设置的不同均会影响到网站的测试。为了保证很好地为预定的客户服务，就要研究他们可能拥有的配置。表 7-1 给出了在设计测试计划时常用的一个矩阵表。

表 7-1 测试设计矩阵表

浏览器版本	PC				UNIX/Mac			
	Windows 98	Windows Me	Windows NT	Windows 2000	Solaris	HP-UX	OS IX	OS X
Internet Explorer 5	✓			✓				
Internet Explorer 5.5	✓	✓		✓				
Internet Explorer 6	✓	✓		✓				
Netscape 4.7	✓	✓		✓	✓			
Netscape 6.0				✓	✓			
...								

专业的测试单位负责客户端测试的人员每人拥有 6 台以上测试用机，每台机器配置不同的操作系统和浏览器，每台机器均采用活动硬盘架，可很快更换备用硬盘来测试不同的系统环境。测试任务的艰巨由此可见一斑。

## 7.5 可安装性和可恢复性测试

软件产品的日益丰富，可获得软件的途径也多种多样，软件的安装方式也发生了很大的变化。有系统软件的安装、应用软件的安装、服务器的安装、客户端的安装、还有产品的升级安装等。

安装测试时要注意以下几点：

- 是否需要专业人员安装。需要专业人员安装的软件通常只有 Readme 文档，或者安装说明书相对简单，依赖安装人员的专业水平。测试的工作量相对较小。而需普通用户自行安装的软件则必须提供安装说明书，并必须以其为基础展开安装测试。

- 软件的安装说明书有无对安装环境做限制和要求。至少在标准配置和最低配置两种环境下安装。曾经有过这样的例子，某客户端产品进行安装测试时十分顺利，在准备发布之前的一次演示中，按安装说明书进行安装时意外发现无法通过，提示没有安装 Java 程序。让主管经理们对测试结果产生了很大的疑问。真正的原因就是测试人员的测试用机都按习惯在装操作系统时默认安装了 Java 程序，造成了测试上的疏漏。
- 安装过程是否简单，容易掌握。软件的安装说明书与实际安装步骤是否一致。对一般用户而言，长长的安装文档，复杂的操作步骤往往造成畏惧心理。如果实际步骤与安装说明再有出入，就容易让用户缺乏信心，增加技术支持的成本。
- 安装过程是否有明显的、合理的提示信息。相应的信息是否合理、合法；插入碟片，选择、更改目录，安装的进程和步骤等均应有明显的、合理的指示。用户许可协议的条款要保证其合理、合法。
- 安装过程中是否会出现不可预见的或不可修复的错误。安装过程中（特别是系统软件）对硬件的识别能力；检查系统安装是否会破坏其他文件或配置；检查系统安装是否可以中止并恢复原状。
- 安装程序是否占用系统资源与原系统冲突，是否会影响原系统的安全性。检查系统是否能够安装所有需要的文件/数据并进行必要的系统设置。
- 软件安装的完整性和灵活性。大型的应用程序会提供多种安装模式（最大、最小、自定义等），每种模式是否能够正确的执行，安装完毕后是否可以进行合理的调整。
- 软件使用的许可号码或注册号码的验证。
- 升级安装后原有应用程序是否可正常运行。
- 卸载测试也是安装测试的一部分。卸载后，文件、目录、快捷方式等是否清除；卸载后，占用的系统资源是否全部释放；卸载后，是否影响其他软件的使用。

### 恢复测试

许多基于计算机的系统必须在限定的时间内从失效状态中恢复过来，然后继续运行。在有些情况下，一个系统必须是可以容错的，这就是说，运行过程中的错误不能使得整个系统的功能都停止。在某些情况下，一个系统错误必须在一个特定的时间段内改正，否则就会产生严重的经济损失。

恢复测试主要检查系统的容错能力。比如，当系统出错时，能否在指定时间间隔内修正错误并重新启动系统。恢复测试首先要通过各种手段，让软件强制性地发生故障，然后验证系统是否能尽快恢复。对于自动恢复需验证重新初始化、检查点、数据恢复和重新启动等机制的正确性；对于人工干预的恢复系统，还需估测平均修复时间，确定其是否在可接受的范围内。

基于服务器/客户端结构的应用是测试工作中常常遇到的。下面就是一个简单的示例，但从中可以得到很好的启发。

先分析服务器端的恢复测试，通常服务器上会有一个进程对其他服务进程进行维护和管理。本例是一台 Linux 系统的服务器。使用 `pgrep -fl svr` 命令列出所有服务进程如下所示，

其中 `atmmsvr` 为维护管理进程，其他均为各种服务进程。

```
[root@lnx2210 root]# pgrep -fl svr
12063 /opt/...../ammsvr
12137 /opt/...../apngsvr 192.168.2.211
12138 /opt/...../acblsvr 192.168.2.213
12139 /opt/...../acb2svr 192.168.2.214
12140 /opt/...../arassvr 192.168.2.215
12141 /opt/...../alicsvr 192.168.2.212
12142 /opt/...../alogsvr 192.168.2.212
12143 /opt/...../achatsvr 192.168.2.213
12144 /opt/...../aassvr 192.168.2.213
12145 /opt/...../adtsvr 192.168.2.213
12146 /opt/...../achatsvr 192.168.2.214
12147 /opt/...../aassvr 192.168.2.214
12148 /opt/...../adtsvr 192.168.2.214
.....
12290 /opt/...../wmssvr
12378 /opt/...../arassvr 192.168.2.215
12592 /opt/...../apngsvr 192.168.2.211
12593 /opt/...../apngsvr 192.168.2.211
[root@lnx2210 root]# kill -9 12138
```

如果我们对其中进程号为 12138 的 `acblsvr` 进行恢复测试，可以使用“`kill -9 12138`”命令将该进程杀掉。立刻通过客户端验证该项服务的丧失，在恢复时间内监控服务器的进程，直到 `acblsvr` 进程被重新启动。再通过客户端验证该项服务的恢复，服务器端系统资源不应该出现较大的变化。

客户端的恢复测试可以用一个更简单的例子进行说明。手工拔下网线，在许可的时间范围内再插上。从客户的角度，服务的丢失和重新获得不能太麻烦，也不能太困难，状态不能发生大的变化，数据能够重新获得。

## 7.6 文档测试

在 20 世纪 80 年代，许多应用软件仅仅有一个叫 `Readme` 的文本文件，进行文档测试未免有点小题大做。但十几年过去了，现在的软件文档已成为软件的一个重要组成部分，而且种类繁多，对文档的测试也变得必不可少。

## 7.6.1 文档的种类

### ● 联机帮助文档或用户手册

这是人们最容易想到的文档。用户手册是随软件发布而印制的小册子，通常是简单的软件使用入门指导书。而详细的帮助指导内容通常以联机帮助文档的形式出现，有索引和搜索功能，用户可以方便、快捷地查找所需信息。Microsoft Word 的联机帮助文档内容非常全面，如图 7-4 所示。多数情况下联机帮助文档已成为软件的一部分，有时也在网站上发布。

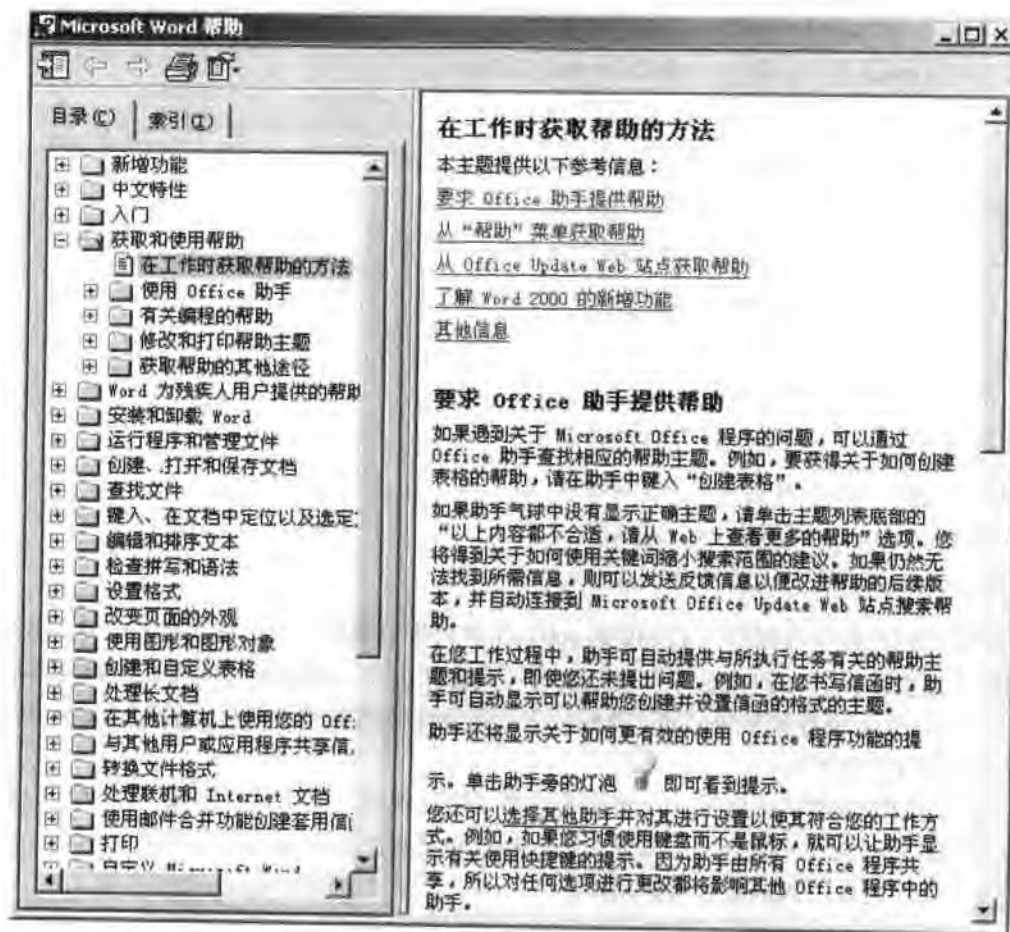


图 7-4 Microsoft Word 的联机帮助

### ● 指南和向导

是程序和文档融合在一起形成的，可以引导用户一步一步完成任务的一种工具，如 Microsoft Office 助手。

### ● 安装、设置指南

简单的可以是一页纸，复杂的可以是一本手册。

- 示例及模板

例如，某些系统提供给用户填写的表单模板。

- 错误提示信息

常常被忽略，但确属于文档。一个较特殊的例子，服务器系统运行时检测到系统资源达到临界值或受到攻击时，给管理员发送的警告邮件。

- 用于演示的图像和声音。
- 授权/注册登记表及用户许可协议。
- 软件的包装、广告宣传材料

有些用户会认真对待，并很好地利用它，因为错误或缺少必要的信息可能带来麻烦。甚至标签上的信息等均为文档测试的内容。

## 7.6.2 文档测试的重要性

对于用户来说，软件文档是软件的一部分，所以文档的错误也是软件缺陷。错误的解释可能会引导用户无法完成某些软件已具有的功能。如果安装文档不正确，用户无法进行安装，肯定是软件的 bug。

好的文档能达到提高易用性、提高可靠性、降低技术支持费用的目的，从而提高了产品的整体质量。用户通过文档可以掌握具体的使用方法，这提高了产品的易用性，避免了用户在摸索使用中一些不可预期的操作，也就相对避免了一些不可预期的错误的发生，从而提高了产品的可靠性。当用户在遇到问题时，多数会向朋友或同事询问解决方法，再就是通过帮助文档或请求公司帮助。约 30% 的用户通过文档解决了问题，也就避免了公司提供费用不菲的技术支持。

## 7.6.3 怎样进行文档测试

非代码的文档测试主要检查文档的正确性、完备性和可理解性。软件驱动文档还得像程序一样运行测试。

正确性是指不要把软件的功能和操作写错，也不允许文档内容前后矛盾。

完备性是指文档不可以“虎头蛇尾”，更不许漏掉关键内容。文档中很多内容对开发者可能是“显然”的，但对用户而言不见得都是“显然”的。

文档要让大众用户看得懂，能理解。术语、缩写用户是否理解？内容和主题是否一致？

很多程序员能编写出好程序，却写不出清晰的文档。与文档作者密切合作，对文档仔细阅读，跟随每个步骤，检查每个图形，尝试每个示例是进行文档测试的基本方法。

## 7.7 验收测试报告和用户验收测试

验收测试是 QA 在整个产品测试中的最后一个环节，完成并通过验收测试后我们需要提交验收测试报告，有时也称为发布报告（release report）。在报告中要综合分析各阶段所有的测试内容，有充分的信心保证产品的质量，并指出可能存在的问题。当然没有 bug 的软件是不存在的，我们不能宣称找出并修正了软件中的所有错误和缺陷，有时迫于市场压力和时间上的考虑，我们会允许即将发布的软件中存在部分级别较低、对用户影响不大的缺陷。

事实上，测试人员不可能完全预见用户实际使用程序的情况，也就不可能发现所有的错误。例如，用户可能错误的理解命令，或提供一些奇怪的数据组合，也可能对设计者自认明了的输出信息迷惑不解等。因此，软件是否真正满足最终用户的要求，应由用户进行一系列“验收测试”。用户验收测试既可以是非正式的测试，也可以有计划、有系统的测试。

用户验收测试由用户完成，验收测试由测试人员完成。原因有以下几点：

- 有时验收测试长达数周，甚至数月，不断暴露错误，导致开发延期。而且大量的错误可能吓跑用户。
- 即使用户愿意做验收测试，他们消耗的时间、花费的金钱大多比测试小组要高。
- 一个软件产品可能拥有众多用户，不可能由每个用户都进行验收，此时多采用称为 $\alpha$ 、 $\beta$ 测试的过程，以期发现那些似乎只有最终用户才能发现的问题。

$\alpha$ 测试是指软件开发公司组织内部人员模拟各类用户对即将面世的软件产品（称为 $\alpha$ 版本）进行测试，试图发现错误并修正。 $\alpha$ 测试的关键在于尽可能逼真地模拟实际运行环境和用户对软件产品的操作，并尽最大努力涵盖所有可能的用户操作方式。经过 $\alpha$ 测试调整的软件产品称为 $\beta$ 版本。紧随其后的 $\beta$ 测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用 $\beta$ 版本，并要求用户报告异常情况、提出批评意见。然后软件开发公司再对 $\beta$ 版本进行改错和完善。

通过 $\alpha$ 测试和 $\beta$ 测试，软件产品就可以发布了。测试人员终于松了一口气，后面的工作让技术支持和维护人员去忙吧！不行，又要进行下一版本？啊，还要测试补丁包……

### 小结

验收测试（acceptance test）是在软件产品完成了功能测试和系统测试之后、产品发布之前所进行的软件测试活动，它是技术测试的最后一个阶段，通过了验收测试，产品就会进入发布阶段。验收测试一般根据产品规格说明书，严格检查产品，逐行逐字地对照说明书上对软件产品所做出的各方面要求，确保所开发的软件产品符合用户预期的各项要求。



## 思考题

1. 验收测试是由用户完成的吗？为什么？
2. 进行验收单元的条件是什么？通过标准是什么？
3. 如何进行产品说明书的验证？
4. 用户界面测试有哪些要素？
5. 简述文档测试的重要性。

# 第 8 章 面向对象软件的测试

思维方式决定解决问题的方式。传统软件开发采用面向过程、面向功能的方法，将程序系统模块化，在此基础上还可以再分成若干个单元，这些单元可以通过一系列程序过程实现，也产生相应的单元测试、集成测试等方法。面向对象程序的结构不再是传统的功能模块结构，它将开发分为面向对象分析（OOA）、面向对象设计（OOD）和面向对象编程（OOP）三个阶段。分析阶段产生整个问题空间的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。针对面向对象软件的开发特点，其测试方法和技术也必然要做相应的改变，从而形成面向对象的测试模型、测试的层次与数据流、面向对象的单元和集成测试方法等，这些都是本章要叙述的内容。

## 8.1 面向对象软件的特点

我们生活在一个对象的世界里，每个对象有一定的属性，把属性相同的对象进行归纳就形成类。例如，家具就可以看做类，其主要的属性有价格、尺寸、重量、位置和颜色等。无论我们谈论桌子、椅子，还是沙发、衣橱，这些属性总是可用的，因为它们都是家具，它们继承了为类定义的所有属性。实际上，计算机软件所创建的面向对象思想同样来源于生活。

除了属性之外，每个对象可以被一系列不同的方式操纵，它可以被买卖、移动、修改(如漆上不同的颜色)。这些操作或方法将改变对象的一个或多个属性。这样所有对类的合法操作可以和对象的定义联系在一起，并且被类的所有实例继承。

在面向对象语言中，类是创建对象的关键。类描述了一组对象的公共特征和操作，而对象则是具体实现的类。面向对象可以定义为：

面向对象(object oriented) = 对象 + 分类 + 继承 + 通信

面向对象技术导致程序构件的复用，而复用导致更快的软件开发和高质量的程序。面向对象软件易于维护，因为它的结构是内紧外松，这样当进行修改时，影响面小。此外，面向对象系统易于进行适应性修改及伸缩。归纳起来其优点有：

- 可重用性。从一开始对象的产生就是为了重复利用，完成的对象将在今后的程序开发中被部分或全部地重复利用。
- 可靠性。由于面向对象的应用程序包含了通过测试的标准部分，因此更加可靠。由于大量代码来源于成熟可靠的类库，因而新开发程序的新增代码明显减少，这是程序可靠性提高的一个重要原因。
- 连续性。具有面向对象特点的 C++ 与 C 语言有很大的兼容性，C 程序员可以比较

容易地过渡到 C++ 语言开发工作。

### 1. 对象的封装和抽象

为有效使用面向对象的程序方法，首先需要解决程序的结构设计问题。在程序设计过程中最重要的是抽象，也就是说，从现实世界中抽象出合理的对象结构。在面向对象思想中，抽象决定了对象的属性、内部结构以及处理对象的外部接口。

从程序语言角度来看，在一个对象中代码和（或）数据可以是这个对象私有的，不能被对象外的部分直接访问。因而对象提供了一种高级保护以防止程序被无关部分错误修改或错误地使用了对象的私有部分。当从对象外部试图直接对受保护的内部数据进行修改时，将被程序拒绝，只有通过对象所提供的对外服务函数才能够对其内部数据进行必要的加工，从而保证了数据加工的合法性。从这一意义上讲，把这种代码和数据的联系称为“封装”。换句话说，封装是将对象封闭保护起来，是将内部细节隐蔽起来的能力。

### 2. 继承性与多态性

面向对象设计方法的另一个重要贡献是关于继承性与多态性的处理。所谓继承是指从现存对象出发建立一个新的对象类型，使它具有原对象的特点和功能。同时，新的对象类型又具有某种新特点和新功能。这样可以采用对象继承的方法建立一个有层次的对外部世界的描述。例如，我们可以想像有一组二维曲面分块的类族，从四条三维空间直线定义的简单曲面，到四条复杂三维空间曲线定义的曲面，呈现复杂的分层次的多态性，但它们都有共同的接口函数形式。

- 访问控制。对象必须能够对其内部的某些元素进行保护，使它们只能被内部使用，而不受外部干扰。反过来，对象又必须同其他外部元素进行联系，以便对对象进行操作。在 C++ 中，类分为私有的（private）、保护的（protected）和公有的（public）三种访问机制。
- 继承性。通过对已有对象进行增加或部分修改的方法建立新的对象，对已有对象可以增加数据和过程，也可以对其中某些过程进行重新定义。最初的类被称为基类，从基类扩展出来的类称为派生类。从已有类派生出新类是为了获得更强的针对性。
- 多态性。正像生态系统一样，继承构成了类族。通常这些类族中的类具有同名的成员函数，例如，OD 分布类族，具有一个通用基类、两个派生类——增长系数 OD 分布类和重力模型 OD 分布类，这几个类都有同名的 Exec 成员函数。多态性意味着存在多种形式，能使人们在程序中激活任何属于 OD 分布类族的 Exec 成员函数，甚至在编译可以不必具体知道对象是属于增长系数 OD 类还是重力模型 OD 分布类。

### 3. 面向对象方法程序开发示例

为了进一步帮助读者了解面向对象软件的特点，我们在此介绍一个应用示例。该示例以交通规划、道路项目决策支持系统的一个基础类库为背景，这个类库中有一个用于管理

道路或交通网络数据的类族，它为交通分配、数据检查、图形输出等多个子系统提供支持。作为整个类族的基础，建立了一个拓补网络，它只描述节点之间的连接特性，各边均没有长度、等级等方面的特性。对于这个拓补网，建立了基类 NetworkManager，其具体定义如下所示：

【清单 8-1】 NetworkManager 的说明

```
class NetworkManager {
protected :
    ...
public :
    构造函数及析构函数...
    int GetSetFlag (void);           //获取设置状态标志
    int NWalloc (void);              //进行系统内存分配
    void SetNetworkType (int type);  //设置网络类型
    void SetMaxLink (int n);         //设置最大边数
    void SetMaxNode (int n);         //设置节点的最大编号×
    int SetNetwork(int n1, int n2);  //在网络中增设一条边
    int SetEnd(void);                //网络数据设置完成后的处理
    void Reverse(void);              //将网络中的所有有向边反向
    int GetNumLink(void);            //获取网络中的边数
    int GetNumNode(void);            //获取网络中的节点数
    int GetLinkStat(int n1, int n2); //获取两指定节点间有几条连接边
    int GetLink(int n1, int n2, int no); //获取指定节点间第no条边的内存序号
    int GetNode1(int l);              //获取内存序号为 l 的边的起点编号
    int GetNode2(int l);              //获取内存序号为 l 的边的终点编号
    int GetNumOutLink(int n);         //获取以节点 n 为起点的边的数量
    int GetOutLink(int n, int no);    //获取以节点 n 为起点的第 no 条边的内存序号
    int GetNumInLink(int n);          //获取以节点 n 为终点的边的数量
    int GetInLink(int n, int no);     //获取以节点 n 为终点的第 no 条边的内存序号
    ...
};
```

说明符 protected 下包含了一些数据以及内部函数的定义，但对于外部调用来说，这些内容都是不可见的，以防止使用者无意中对此部分内容进行了错误的操作，这就是类的屏蔽。而在说明符 public 下列出的函数均是对外开放的服务函数，从外部可以调用它们获得所需的支持。

在这些函数中，GetNode1()、GetNode2()、GetNumOutLink()/ GetNumInLink()、GetOutLink()/GetInLink ()等是必须的、具有真正服务功能的函数。它们分别可以通过给定边的编号查询该边的起（终）点编号，根据给定节点的编号查询从该节点出去或进来的边的编号，以及查询给定节点出去或进来的第 n 条边的编号。这类函数还有 Reverse()、GetNumLink()、GetNumNode()、GetLinkStat()、GetLink()等。

而 `SetNetwork()` 起的作用是告诉类网络中的某一条边的起点和终点, 换句话说是将外部的数据输入到 `NetworkManager` 中间去。由于这种处理方法, 这个类与具体数据形式脱钩了, 从而增强了其通用性。调用者可以根据具体情况从数据文件或数据库中获取网络数据, 然后调用 `SetNetwork` 一条边一条边地把数据送进网络管理类中。这类函数还有 `SetNetworkType()`、`SetMaxLink()`、`SetMaxNode()` 等。

类族结构如图 8-1 所示, 从 `NetworkManager` 逐步派生出整个类族。`CTPSNetworkForWin` 与 `CTPSNetworkForDOS` 在内存申请处理方面有所区别, 分别为 Windows 和 DOS 环境下的程序开发提供服务, 与 `NetworkManager` 相比, 对各边增加了长度和等级的特征值。

`CTPSNetworkCost` 新增加了有关各边的费用函数处理功能, 以及能够对各边进行流量、运行车速等数据存取。`NetworkCost` 与其父类相比, 增加了一些有关网络总体状态合计处理的功能, 例如计算整个网络的车公里合计值等。`NetworkNode` 则是在节点上增加了数据存储功能, 因而能够记录节点的坐标值等数据。`NetworkFlowNode` 也是增加了节点数据的存储处理功能, 但由于其父类的不同, 因而能够处理许多有关流量的信息。

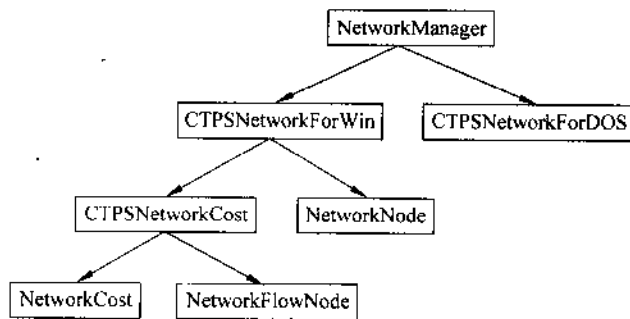


图 8-1 网络类派生关系图

我们进一步以 `CTPSNetworkForWin` 为例, 说明派生类与父类之间的关系。对于父类已有的对外服务函数, 派生类仍然可以继续使用。也就是说, 派生类是在父类的基础上, 增加一些新功能而建立的。这就使得我们能够根据需要逐步通过派生的方法建立既有相同功能, 又有所区别类族, 以满足不同的要求。

#### 【清单 8-2】 CTPSNetworkForWin 的说明

```

class CTPSNetworkForWin : public NetworkManag {
    //本网络类是针对 Windows 环境所开发的派生类, 与 NetworkManag 相比, 网络边增加了
    //长度及等级特征
public:
    构造及析构函数...
    int SetLinkLength (int n1, int n2, int no, double x);
                                     //设置指定节点间第 no 条边的长度
    int SetLinkLength(int n, double x); //根据边的内存序号设置长度
    int SetLinkClass (int n1, int n2, int no, int x);
                                     //设置指定节点间第 no 条边的等级
    int SetLinkClass (int n, int x); //根据边的内存序号设置长度
    double GetLinkLength (int n); //根据边的内存序号获取边的长度
    double GetLinkLength (int n1, int n2, int no);
  
```

```

//获取指定节点间第 no 条边的长度
int GetLinkClass (int n1, int n2, int no);

//获取指定节点间第 no 条边的等级
//根据边的内存序号获取边的等级
int GetLinkClass (int n);
};

```

在清单 8-2 中, 标志 `class CTPSNetworkForWin : public NetworkManag` 说明 `class CTPSNetworkForWin` 是由 `class NetworkManag` 中公有派生出来的, 这是 C++ 中类属关系的句法表达式, 它告诉编译器 `CTPSNetworkForWin` 继承了 `NetworkManag` 的基本特征, 所作的改变将在 {} 中加以说明。

## 8.2 面向对象测试的层次与数据流

面向对象软件的开发与传统软件的开发有许多不同, 上一节主要讲了面向对象软件的特点并通过一个示例来说明面向对象软件开发的过程, 这一节我们将介绍面向对象软件测试的层次与数据流, 这是面向对象软件的单元测试与集成测试的基础。

### 8.2.1 类与子类的测试

当类的实现从上往下时, 实现起来就比较简单。同样, 在测试继承层次中的类时, 如果采用从上往下的方法时通常比较容易。在测试第一层次的类时, 可以着重于通用接口和代码, 然后针对每一个子类生成专门的测试驱动代码。为了方便说明, 假设采用从上往下的方法对继承关系中的类型进行测试。我们将着重测试一个其父类已经测试过的子类, 如图 8-2 所示, 类 D 是类 C 的子类, 类 C 已经进行了充分的测试。下面开始对子类 D 进行测试。

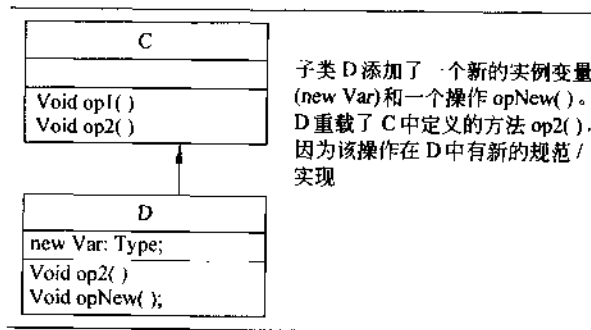


图 8-2 类与子类的层次与继承关系

由于类 D 至少从类 C 继承了它的部分规范以及实现, 因此可以合理地假设 C 的有些测试软件可以在测试 D 时复用。考虑下面这个例子: D 从 C 继承了退化功能, 并且没有做任何修改, 因此, D 在规范和实现上与 C 相同。如果我们可以假设编译器能够正确地处理代码, 那么类 D 根本就不需要测试。在这样的假设下, 如果 C 通过了所有的测试, 那么类 D

也一定能够通过。比较常见的情况是，D 包含对 C 的增量变化。通过复用 C 的测试用例和部分测试驱动程序可以大大减少测试 D 所花费的代价。

## 8.2.2 分层与增量

类 C 和其派生类 D 间的增量变化能够用来帮助确定需要在 D 中测试什么。由于 D 是 C 的子类，那么所有用于 C 的基于规范的测试用例也都适用于 D。引入术语“继承的测试用例”来代表从父类测试用例中选取出来的、用于子类的测试用例。可以通过简单的分析来确定继承的测试用例中哪些适用于测试子类、哪些在测试子类时不必执行。在这里需要注意以下几个方面：

- D 的接口中添加一个或多个新的操作，并且有可能是 D 中的一个新方法实现一个新操作。新操作引入了新的功能和新的代码，这些都需要测试。一个新的操作对现存的、继承来的操作或方法没有直接的影响时，只需要为每个新操作添加基于规范的测试用例。如果操作不是抽象的并且有具体的实现，那么为了服从测试计划的覆盖标准，需要添加基于规范和基于交互的测试用例。
- 在 D 中改变那些在 C 中声明的操作规范，需要为操作添加新的基于规范的测试用例。附加的测试用例提供了新的输入，这些输入符合任何削弱了的前置条件，并且对由任何加强了后置条件导致的新期望结果进行检查。C 中定义的这个操作规范仍然适用，但是必须重新运行。另外还需要向用于测试类 C 中的这个操作测试的每一个用例输出添加强化的后置条件需求。
- 在 D 中覆盖那些在 C 中实现了某个操作并且被 D 继承了的方法，可以复用于该方法所有继承来的基于规范的测试用例。因为有新的代码需要测试，我们还需要对基于实现的测试用例进行检查，如果需要的话，对它们进行修正和添加以符合覆盖率的测试标准。
- 在 D 中添加一个或多个新的实例变量来实现更多的状态和/或属性。添加新的变量最有可能与新的操作和/或重载方法中代码有关，而且对测试的处理也与它们有关。如果新变量没有在任何方法中使用，那么我们就不必作任何改变。
- 在 D 中改变类常量。类常量累计成每个测试用例的附加后置条件。我们宁愿将它们看做是隐含的后置条件，并且在没有对常量限制任何显示引用的情况下编写测试用例。测试用例的输出受常量约束的支配——即“类常量句柄”在每个测试用例的输出中是显示的。因此，如果一个类常量变化了，那么我们需要重新运行所有继承的测试用例以验证新常量句柄。

## 8.2.3 面向对象层次结构测试重点

在掌握类与子类的测试、分层与增量之后，我们要讨论面向对象层次结构测试的要点，主要从以下三个方面去展开：

- 对认定的对象的测试。

- 对认定的结构的测试。
- 对构造的类层次结构的测试。

### 1. 对认定对象的测试

OOA（面向对象分析测试）中认定的对象是对问题空间中的结构、其他系统、设备、被记忆的事件、系统涉及的人员等实际实例的抽象。对它的测试可以从如下方面考虑：

- 认定的对象是否全面，其名称应该尽量准确、适用，是否问题空间中所涉及到的实例都反映在认定的抽象对象中。
- 认定的对象是否具有多个属性。只有一个属性的对象通常应看做其他对象的属性而不是抽象为独立的对象。
- 对认定为同一对象的实例是否有共同的、区别于其他实例的共同属性，是否提供或需要相同的服务，如果服务随着不同的实例而变化，认定的对象就需要分解或利用继承性来分类表示。
- 如果系统没有必要始终保持对象代表的实例信息，提供或者得到关于它的服务，认定的对象也无必要。

### 2. 对认定结构的测试

认定的结构指的是多种对象的组织方式，用来反映问题空间中的复杂实例和复杂关系。认定的结构分为两种：分类结构和组装结构。分类结构体现了问题空间中实例的一般与特殊的关系；组装结构体现了问题空间中实例整体与局部的关系。

认定的分类结构测试要点：

- 对于结构中的一种对象，尤其是处于高层的对象，是否在问题空间中含有不同于下一层对象的特殊可能性，即是否能派生出下一层对象。
- 对于结构中的一种对象，尤其是处于同一低层的对象，是否能抽象出在现实中有意义的更一般的上层对象。
- 对所有认定的对象，是否能在问题空间内向上层抽象出在现实中有意义的对象。
- 高层对象的特性是否完全体现下层的共性，低层的对象是否有高层特性基础上的特殊性。

认定的组装结构的测试要点：

- 整体（对象）和部件（对象）是否在考虑的问题空间中有实际应用，其组装关系是否符合现实的关系。
- 整体（对象）中是否遗漏了反映在问题空间中的有用部件（对象）。
- 部件（对象）是否能够在空间中组装新的有现实意义的整体（对象）。

### 3. 对构造类层次结构的测试

为了能充分发挥面向对象继承共享特性，OOD（面向对象设计测试）的类层次结构通常基于 OOA 中产生的分类结构原则来组织，着重体现父类和子类间的一般性和特殊性。在当前的问题空间，对类层次结构的主要要求是能了解空间构造实现全部功能的结构框架。



为此测试要注意如下几个方面:

- 类层次结构是否涵盖了所有定义的类。
- 是否能体现 OOA 中所定义的实例关联、消息关联。
- 子类是否具有父类没有的新特性。
- 子类间的共同特性是否完全在父类中得以体现。

## 8.3 面向对象的单元测试

尽管许多常用的面向对象软件的测试方法和技巧与过程软件相同,或者可以从传统的测试方法和技巧中演化出来,但实践和研究表明它们之间还是存在许多的不同,面向对象的软件测试要面对许多新的挑战,如面向对象的编程语言中继承性和多态性对测试者来说就是一个新的技术难点。与此同时,作为增量开发过程的一部分,设计良好的面向对象软件为改善传统测试过程提供了机遇,可以提高软件的可维护性、复用性和灵活性等。

### 8.3.1 面向对象软件测试与传统软件的不同

在编程语言中,面向对象编程的特性很显然对测试的某些方面有影响。比如,类的继承和接口等特性支持多态,使得代码可以控制对象而不管对象到底是什么。测试者必须确保代码能正常工作而不管与那个对象确切相关的类是什么。支持和加强数据隐藏的特性可能使测试复杂化,因为有时为了支持测试必须向一个类的接口添加操作。与此同时,这些特性都能更好地、重复地使用测试软件。

传统的单元测试在面向对象的软件测试中其实是对类的测试,因为类是面向对象软件的基本单位,而不再是以前的模块。传统的集成测试:自顶向下、自底向上等测试在面向对象中基本不适用,面向对象的集成测试主要是两个方面:

- 类的线性测试、交互测试。
- 类的独立性测试(跨平台)。

### 8.3.2 类测试

面向对象程序的基本单位是类。类测试是由那些与验证类的实现是否和该类的说明完全一致的相关联活动组成的。如果类的实现正确,那么类的每一个实例的行为也应该是正确的。

#### 1. 类测试的方法

通过代码检查或执行测试用例能有效地测试一个类的代码。在某些情况下,用代码检查代替基于执行的测试方法是可行的。但是和基于执行的测试方法相比,代码检查有两个不利之处:

- 代码检查易受人为错误的影响。
- 代码检查在新产品开发时明显需要更多的工作量。

尽管基于执行的测试方法克服了这些缺点，但确定测试和开发测试驱动程序也需要很大的工作量。在某些情况下，为某个类构造一个测试驱动程序所需要的工作量可能比开发这个类所需要的工作量要大得多。但这种情况不是面向对象编程独有的，当有许多子程序被上一层调用时，在传统开发过程中，也会出现类似的情形。一旦确定了一个类的可执行测试用例，测试驱动程序创建一个或多个类的实例来运行一个测试用例，我们就必须执行测试驱动程序来运行每个测试用例，并给出每个测试用例运行的结果。

## 2. 类测试的组成部分

作为每个类，决定是将其作为一个单元进行独立测试，还是以某种方式将其作为系统某个较大部分的一个组件进行独立测试，需要基于以下因素进行决策：

- 这个类在系统中的作用，尤其是与之相关联的风险程度。
- 这个类的复杂性（根据状态个数、操作个数以及关联其他类的程度等进行衡量）。
- 开发这个类测试驱动程序所需的工作量。

假如一个类是某个类库不可缺少的部分，即使测试驱动程序的开发成本可能很高，对它进行充分的测试也是值得的，因为它的正确操作是最重要的。在进行类测试时，一般要考虑以下几个方面：

- **测试人员。**如同传统的单元测试是由开发人员来执行，类的测试通常也由开发人员来进行。因为测试人员对代码极其熟悉，开发人员可以使用测试驱动程序来调试他们编写的代码，方便了基于执行的测试方法。
- **测试内容。**对一个类进行测试以检查它是否只做了规定的事情，确保一个类的代码能够完全满足类说明所描述的要求。在运行了各种测试用例后，如果代码的覆盖率不完整，这可能意味着该类设计过于复杂，需要简化成几个子类。
- **测试时间。**类测试可以在开发过程的不同位置进行。在一个递增的、反复的开发过程中，一个类的说明/实现可能会发生变化，所以应该在软件其他部分使用该之前执行类的测试。每当一个类的实现发生变化时，就应该执行回归测试。所以类的测试要和类的设计、开发保持同步，因为确定早期测试用例有助于对类说明的理解保持一致，也有助于获得独立代码检查的反馈。若一个类的开发人员不能设计充分和准确的测试用例，其测试结果会给人一个错觉——即该类通过了所有测试。但是当该类被集成到某个较大的系统时，将会导致严重的问题。
- **测试过程。**类的测试通常要借助测试驱动程序，这个驱动程序创建类的实例，并为那些实例创造适当的环境以便运行一个测试用例。驱动程序向测试用例指定的一个实例发送一个或多个消息，然后根据参数、响应值、实例发生的变化来检查那些消息产生的结果。如果编程语言（例如 C++）具有程序员管理存储分配的机制，那么测试驱动程序需要删除它所创建的那些实例。

- **测试程度。**可以根据已经测试了多少类实现和多少类说明来衡量测试的充分性。对于类测试来说,要测试操作和状态转换的各种组合情况,但有时穷举法是不可能的,此时就应该选择配对系列的组合情况,如果能结合风险分析进行选择,效果会明显些。

### 3. 构建测试用例

首先看怎样从类说明中确定测试用例,然后根据类实现引进的边界值来扩充附加的测试用例。类说明通常可以用多种方式进行描述,如自然语言和状态图等,假如要测试类的说明不存在,那么就可通过“逆向工程法”产生一个说明,并在开始测试之前让开发人员对之进行检查。

根据前置条件和后置条件来构建测试用例的总体思想是:为所有可能出现的组合情况确定测试用例需求。在这些可能出现的组合情况下,可以满足前置条件,也能够达到后置条件。接下来创建测试用例来表达这些需求,根据这些需求还可以创建拥有特定输入值(包括常见值和边界值)的测试用例,并确定它们的正确输出。最后,还可以增加测试用例来阐述违反前置条件所发生的情况。

### 4. 类测试系列的充分性

在某些情况下,可以使用穷举法来测试每个类,即用所有可能的值来测试,以确保每一个类都符合它的说明。在这种情况下,穷举测试法所带来的好处就超过了编写测试驱动程序以运行更多测试用例所花的代价。

但是穷举测试法一般是不可能实现的,如果不能使用穷举测试法时,就不能保证一个类的每一方面都符合它的说明,但能够运用某个充分性的标准来使我们对测试系列的质量抱有高度的信心。充分性的3个常用标准是:基于状态的覆盖率、基于限制的覆盖率、基于代码的覆盖率。最低限度地符合这些标准将会产生若干不同的测试系列。将所有3个标准用于测试系列,将会提高我们对测试充分性的信任度。

- **基于状态的覆盖率。**以测试覆盖了多少个状态转换为依据。假如测试没有覆盖一个或一个以上的状态转换,那么其类的测试就不充分。即使测试用例对所有的状态都覆盖了一次,测试的充分性仍值得怀疑,因为状态通常包含了各种对象属性的值域。这样,必须测试这些值域里的所有值,包括典型值和边界值。
- **基于约束的覆盖率。**与基于状态转换的充分性类似,还可以根据有多少对前置条件和后置条件被覆盖来表示充分性。例如,如果一个操作的前置条件是 `pre1` 或者 `pre2`,而后置条件是 `post1` 或者 `post2`,充分的测试则需要包含所有有效的组合情况(即 `pre1=true, pre2=false, post1=true, post2=false`; `pre1=false, pre2=true, post1=true, post2=false` 等)的测试用例。假如生成的测试用例满足了每一个需求,那么就符合这个充分性的标准。
- **基于代码的覆盖率。**当所有的测试用例都执行结束时,确定实现一个类的每一行代码,或代码通过的每一条路径至少执行了一次,这是一种很好的思想。即使代

码覆盖率是 100%，也不一定能满足基于状态覆盖率或基于约束的覆盖率是 100%。基于代码的覆盖率不够充分，所以使用哪些度量标准中的某一种来确定充分性是很重要的。

### 5. 构建测试的驱动程序

测试驱动程序是一个运行测试用例并收集运行结果的程序。测试驱动程序的设计应该相对简单，因为我们很少有时间和资源来对驱动程序软件进行基于执行的测试（否则会进入一个程序测试的递归的、无穷之路），而是依赖代码检查来检测测试驱动程序。所以，测试驱动程序必须是严谨的，结构清晰、简单，易于维护，并且对所测试类的说明变化具有很强的适应能力。理想情况下，在创建新的测试驱动程序时，应该能够复用已存在的驱动程序代码。

### 6. Tester 类的设计

由于 Tester 类提供了一些操作来帮助给出测试用例的结果，一个具体的 Tester 类的主要任务就是运行测试用例和给出结果。类接口的主要组成部分是建立测试用例的操作、分析测试用例结果的操作、执行测试用例的操作和创建用于运行测试用例的输出实例的操作。在具体的 Tester 类中，为每一个测试用例定义了一个方法，被称为测试用例方法。这些方法给测试计划提供了可跟踪性——每一个测试用例或每一组紧密联系的测试用例都有一个方法。测试用例方法的目的是通过创建输入状态、生成事件序列并检查输出状态来执行测试用例。

测试用例方法的任务是为某个用例构建输入状态。例如，通过将一个输出和作为参数传递的对象实例化，然后生成测试用例指定的事件。图 8-3 显示了一个满足了这些需求的 Tester 类的模型。

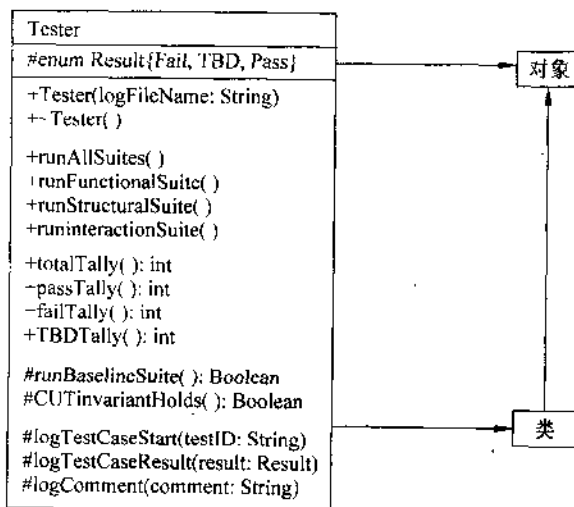


图 8-3 Tester 类需求的一个类模型

公共接口提供了一些操作来运行各种不同的测试，根据测试用例的来源组织测试系列。如果测试用例是根据类说明确定的，那么就是功能性测试用例；如果测试用例是根据代码确定的，那么就是结构性测试用例；如果测试用例是测试一个事件序列对一个对象的操作（例如几对输入输出转换是否正确），那么就是交互性测试用例。我们确定这些范畴是为了便于测试的维护。

## 8.4 面向对象的集成测试

面向对象的程序是由若干对象组成的，这些对象互相协作以解决某些问题。对象的协作方式决定了程序能做什么，从而决定了这个程序执行的正确性。例如，可信任的原始类实例可能不包含任何错误。因此，一个程序中对象的正确协作即交互对于程序的正确性是非常关键的。

交互测试的重点是确保对象（这些对象的类已被单独测试过）的消息传送能够正确进行。交互测试的执行可以使用嵌入到应用程序中的交互对象，或者在独立测试工具（例如一个 `Tester` 类）提供的环境中，交互测试通过使得该环境中的对象相互交互而执行。

### 8.4.1 对象交互

对象交互只不过是一个对象（发送者）对另一个对象（接收者）的请求，发送者请求接收者执行接收者的一个操作，而接收者进行的所有处理工作就是完成这个请求。在大多数面向对象的语言中，对象交互涵盖了程序中的绝大部分活动。它包含了对象及其组件的消息，还包含了对象和与之相关的其他对象之间的消息。假设这些其他对象是一些类的实例，这些类已经被独立测试过了，并且它们的实现已被证明是完整的。因为在处理接收对象上任意某个方法的调用期间都可能发生多重的对象交互，所以希望考虑一下这些交互对接收对象内部状态的影响，还有对那些与接收对象相关联的对象的影响。

对象交互的测试方法，按原始类、汇集类和协作类来进行讨论。原始类的测试使用类的单元测试方法，前面已经介绍。

#### 1. 汇集类测试

有些类在它们的说明中使用对象，但是实际上从不和这些对象中的任何一个进行协作，也就是说，它们从来不请求这些对象的任何服务。相反，它们会表现出以下的一个或多个行为：

- 存放这些对象的引用（或指针），程序中常表现为对象之间一对多的关系。
- 创建这些对象的实例。
- 删除这些对象的实例。

可以使用测试原始类的方法来测试汇集类，测试驱动程序要创建一些实例，这些实例作为消息中的参数被传递给一个正在测试的集合。测试的目的主要是保证那些实例被正确从集合中移出。有些测试用例会说明集合对其容量所做的限制。假如在实际应用中可能要加入 40 或 50 条信息，那么生成的测试用例至少要增加 50 条信息。如果无法估算出一个有代表性的上限，那么就使用集合中的大量对象进行测试。

## 2. 协作类测试

凡不是汇集类的非原始类就是协作类，该类的一个或多个操作中使用其他的对象并将其作为它们的实现中不可缺少的一部分。当类接口中的一个操作的某个后置条件引用了一具对象的实例状态，并且（或者）说明那个对象的某个属性被使用或修改了，那么这个类就是一个协作类。协作类测试的复杂性远远高于汇集类或原始类的测试。

## 8.4.2 面向对象集成测试的常用方法

面向对象集成测试除了要考虑对象交互特征面进行分类之外，还需要一些具体的测试技术去实现测试的要求。在测试中，希望运行所有可能出现的组合情况，而达到 100% 的覆盖率，这就是穷举测试法。穷举测试法是一种可靠的测试方法，然而在许多情况下，却没有办法实施，因为对象交互作用的组合数量太多了，没有足够的时间去构建和执行这些测试。所以人们希望使用更有效的测试方法。主要有：抽样测试、正交阵列测试。

### 1. 抽样测试

抽样测试提供了一种运算法则，它使我们能够从一组可能的测试用例中选择一个测试系列。但并不要求一定要首先明确如何来确定测试用例的总体。测试过程的目的在于定义感兴趣的测试总体，然后定义一种方法，以便在这些测试用例中选择哪些被构建、哪些被执行。

### 2. 正交阵列测试

正交阵列测试提供了一种特殊的抽样方法，这种方法通过定义一组交互对象的配对方式组合，以尽力限制测试配置的组合数目激增。由交互所产生的大多数错误要归咎于双向交互。挑选某个样本的一种特定测试技术就是正交阵列测试系统（orthogonal array testing system, OATS）。正交阵列（orthogonal array）是一个数值矩阵，其中的每一列代表一个因素（factor），即实验中的一个变量。在一个正交阵列中，将各个因素组合成配对情况。例如，假设有 3 个因素——即 A、B、C，每个因素有 3 个级别——即 1、2 和 3，那么这些值就有 27 种可能的组合情况——A 的 3 种组合情况  $\times$  C 的 3 种组合情况。如果使用配对方式的组合，也就是说，如果仅仅考虑这些组合情况：一个给定级别仅出现两次，那么就只有表 8-1 所示的 9 种情况。

表 8-1 3 个因素（每个因素有 3 个层次）配对方式的组合情况

	1	2	3	4	5	6	7	8	9
A	1	1	1	2	2	2	3	3	3
B	1	2	3	1	2	3	1	2	3
C	3	2	1	2	1	3	1	3	2

### 8.4.3 分布式对象测试

如今很少有设计单个进程在单个处理机上执行的系统，为了获得灵活性和伸展性，许多系统都被设计成多个充分独立的部件，每个部件可以存在于一个独立的进程中，而整个系统的运行会根据需要启动多个进程。如果这些进程不是分布在一台机器上，而是分布在多台机器上，借助于计算机通信或网络实现它们相互之间的协作，从而构成一个分布式的系统。客户机/服务器模型是一种简单的分布式系统，在这种模型中，客户机和服务器部件被设计成存在于独立的进程中，服务器提供数据计算、处理、存储等管理工作，客户端接受用户的输入、请求、显示结果等工作，两者分工不同。随着计算机技术的发展，现在可以构造一个分布式的服务器集群，通过并行技术实现复杂的或巨量的计算；也可以构造没有服务器的、分布式的、由客户端构成的对等网络 (P2P) 系统。

#### 1. 分布式对象的概念和特点

线程是一个操作系统进程内能够独立运行的内容，它拥有自己的计数器和本地数据。线程是能够被调度执行的最小单位。面向对象语言通过隐藏接口的属性或在某些情况下使线程对对象做出反应，以此提供一些简单的同步手段。这就意味着在对象接口中同步是可见的（如 Java 的 `synchronize` 关键字），而传递消息是同步中最关键的一环。在这种情况下，类测试并不能发现很多同步错误，只有当一系列对象交互作用时才真正有机会发现错误。

当软件包含多个并发进程时，其特点是不确定性，完全地重复运行一个测试是很困难的。线程准确的执行是由操作系统安排的，与系统测试无关的程序变化可能会影响测试中的系统线程执行顺序。这就意味着如果出现失败，缺陷就必须被隔离并修复，并重新测试。不能因为在一个特定执行中没有发生错误就肯定缺陷被消除了，我们必须使用下列技术之一来进行测试。

- 在类的层次上进行更彻底的测试。对用来产生分布式对象的类进行设计检查，应该确定类设计中是否提供了恰当的同步机制。动态类测试应该确定在受控制的测试环境中同步是否正常。
- 在记录事件发生顺序的同时，执行大量的测试用例。这就增大了执行所有顺序的可能性。而努力想发现的问题正是来源于事件执行的顺序。如果一一执行所有的顺序，就能找到这些问题。
- 指定标准的测试环境。从一台尽可能简单的机器开始，包括尽可能少的网络、调制解调器或其他共享设备互联。并确定应用程序能够在这个平台上运行。然后安

装一套基本的应用程序，它将一直在此机器上运行。每个测试用例都应该描述在标准环境下所做的任何修改，还要包括进程开始的顺序。标准环境下的程序调试应允许测试者控制线程的创建、执行和删除的顺序。环境越大，共享和网络化的程度越高，要保持环境的一致性就越难。不论在哪我们都应该有测试实验室，并把测试机器与公共网络的其他部分隔离，这些机器专用于测试进程。

## 2. 测试中需要注意的情况

- **局部故障。**由于以分布式系统为主的机器上的软件或硬件可能出错，分布式系统的部分代码也许就不能执行，而运行在单一机器上的应用程序是不会遇到这类问题的。局部出错的可能性使我们应考虑针对网络连接的断开、失灵或关闭网络上的一个节点而发生故障的这类测试。这一点可以在前面提到的实验室进行实现。
- **超时。**当一个请求发送到另一个系统时，网络系统通过设置定时器来避免死锁。如果在指定的时间内没有得到任何的回应，系统就放弃这个请求。这可能是由于系统的死锁，或是网络上的机器太忙以致反映的时间比规定的时间要长。当出现请求被回答或未被回答时，软件应该能够做出正确的反映。当然在这两种情况下，反映是不同的。在网络机器上运行测试时，测试必须加载多种配置。
- **结构的动态性。**分布式系统通常具有依靠多种机器的加载来改变自身配置的能力，比如特定请求的动态定向。系统的设计要允许多种机器参与进来，而且系统也需要根据大量的配置来重复测试。如果存在一组大量配置，对这些配置进行全部测试是可行的。另外，可以使用比如正交阵列测试系统这样的技术来选择一套特殊的测试配置。
- **线程。**作为时间调度的计算单元，引进了线程的概念。在设计中，基本的权衡是以线程的数量为中心。增加线程的数量可以简化一定的算法和技术，但线程执行的顺序出现风险的机会更大。减少线程的数量可以减少这种顺序问题，但会使软件更为刻板而且通常效率会更低。
- **同步。**当两个或两个以上的线程都必须访问一个存储空间时，就需要一定的机制来避免两个线程相互冲突。而且两个线程可能会同时对数据进行修改。一些语言（如 Java）提供了语言关键字来自动添加避免同时访问的机制。而其他语言，如 C++，则要求每个开发者必须自行构造以达到这一目的。在面向对象的语言中同步会显得更为简单，因为这种机制局限于一般数据属性的修改方法，而且有不正一个特定的方法来避免实际数据的直接存取。

## 小结

本章首先介绍面向对象软件与传统软件设计方法与测试方法的不同，描述了面向对象中数据流与层次、类与子类、分层与增量等，然后着重介绍面向对象软件的单元测试与集成测试的方法，面向对象软件的单元测试与集成测试和传统软件测试的不同之处。



## 思考题

1. 面向对象的软件测试与传统的软件测试有何不同?
2. 测试者希望开发者为系统做出什么样的规定?
3. 了解一下你想要测试系统的可用的文档和模型。对于缺少的部分,设想一下你能怎样提供这些必要的信息。
4. 类测试环境下衡量类测试的 5 个方面是什么?如何衡量类测试的充分性?
5. 有没有可能设计一个类,使测试变得更简单?
6. 为什么要做交互测试?
7. 对于抽样测试我们怎样衡量它的覆盖程度与优越程度?
8. 分层、增量测试(HIT)近似于采用这种方法进行测试,即通过少量的分析,我们能够避免重复测试已经测试过的代码。请从你所做项目的继承层次中选择一些类并进行 HIT 分析。估计一下为充分测试继承层次中的所有类而实现足够的测试用例所需的工作量,并将你的估计与你进行 HIT 分析的工作量作比较。
9. 确定一个待测试的网站。基于网站上的应用,描述一个访问者在网站中从头到尾参与可能执行的一系列典型活动。尽可能地包括数据输入页面,写出包含所有可能性的测试用例。

## 第9章 基于应用服务器的测试

从一般意义上说,服务器是指一些通用的、具有很强的集中处理能力、可靠性、可扩展性和可管理性的计算设备。随着网络的不断发展,服务器的概念有所拓展,其应用范围越来越广,除了一些专用的服务器,如网关服务器、代理服务器等之外,大多数服务器都被用来提供业务或商业应用的计算服务,成为基于网络的应用系统的核心。

应用服务器一般用于执行单一的或者是专用成套的功能,并为最终用户提供一个完整的软硬件解决方案,其中常见的一些应用服务器有:Web 服务器、数据库服务器、FTP 服务器、邮件服务器、文件共享服务器等。本章就讨论各种应用服务器的测试方法、技术和经验。

### 9.1 应用服务器的分类和特征

应用服务器根据其结构和使用领域的不同有所分类,在接下来的小节中,将对常见的应用服务器分类以及浏览器/服务器(B/S, Browser/Server)、客户端/服务器(C/S, Client/Server)结构、多层结构的应用服务器进行说明。

#### 9.1.1 应用服务器的分类

根据应用服务器的应用领域和范围,一般分为:

- Web 服务器。通过 MS IIS、Bea Weblogic 和 Apache 等中间件、插件,提供 Internet/Intranet Web 服务,实现与众多客户之间的数据交换和共享。
- 数据库服务器。顾名思义,其主要功能是提供数据库查询、处理的平台,通过 Oracle/SQL Server/Informix/DB2 等大中型的数据库管理系统来构建。
- 实时通信服务器。提供数据实时通信、消息传递等服务,如著名的实时通信平台 MSN、Yahoo message 和 OICQ 等专用服务器。
- 邮件服务器。实现邮件管理的服务器系统,如 MS Exchange Server。
- 群件服务器。提供工作群组之间的协作服务,如著名的 Lotus Domino 平台。
- 文件/打印服务器。实现文件共享和打印功能的服务器系统。

当然,以 Java 技术为基础的 J2EE 构架的服务器也可以算是一种应用服务器,在本章中也会单独说明。

## 9.1.2 C/S 和 B/S 结构简述

C/S 结构是目前常用的应用服务模式之一,它使用客户/服务模型进行工作。在服务端,一般采用高性能的 PC、工作站或者专用服务器,并根据需要采用大型的数据系统,如 Oracle、Sybase、Informix 或者 MS SQL Server;而客户端则需要安装专用的客户端软件。C/S 结构是一种非常常见的结构,例如客户端基于 Outlook Express,服务端基于 Outlook Exchange Server,又如 MSN、Yahoo message、网络游戏等都是典型的 C/S 结构的应用。

C/S 结构充分发挥了客户端 PC 的处理功能,将很多部分的工作,如计算、数据采集等通过客户端处理以后再提交给服务器,这样相对就减少了服务器的压力,从而能很快响应客户端的请求。C/S 结构也有着很大的限制,客户端需要安装用户专用的客户端软件,这样给开发、安装、升级、维护,以及数据存储都带来一系列问题。同时,客户端程序可能会受到操作系统的限制,如果应用程序不支持跨平台特性,只能运行在 Windows 系统上,就无法在 Linux、Solaris 或者其他平台上运行。为了解决这一问题,就出现了 B/S 结构。

B/S 结构克服了 C/S 结构的上述缺点——安装维护不方便、需要在客户端机器上安装程序,B/S 结构不需要额外的客户端程序支持,而是通过浏览器与服务器进行通信和数据传输,容易维护和升级。目前一般操作系统自带浏览器,包括 Windows、Mac OS、UNIX、Linux 等平台都已安装浏览器,最常见的浏览器有 Microsoft 的 Internet Explorer、Netscape 的 NS 等。B/S 的结构应用也非常广泛,如搜狐、新浪门户网站、各种电子商务网站就使用 B/S 结构。

## 9.1.3 三层和多层结构

随着计算机技术的发展,传统的 C/S 结构或 B/S 结构越来越难以实现复杂的数据处理过程和日益增长的客户请求,因此,软件界提出了三层或多层体系结构的概念。

三层结构主要是将应用层分离出几个相互隔离的逻辑层,如图 9-1 所示,每一层都定义好一套接口集。第一层,也就是表示层,主要由类似于图形用户界面的部分组成;中间层,即业务层,由应用逻辑和业务逻辑构成;而第三层即数据层,包括了应用程序中所需要的数据。

中间层(应用逻辑)代码由用户调用(通过表示层)来获取需要的数据,表示层接收数据并且按照适当的格式显示出来。从用户界面中分离出应用逻辑,极大地增强了应用程序设计的灵活性。在应用逻辑对表示层提供了一套定义清晰的接口的情况下,甚至不用改变应用逻辑部分,就可以建立和分发多种用户界面。第三层包含了应用程序所需要的数据。这些数据可以由不同的信息源组成。例如可以是 Oracle、Informix 这样的数据库,也可以是 XML 文档集,还可以是 LDAP 服务器的目录服务。

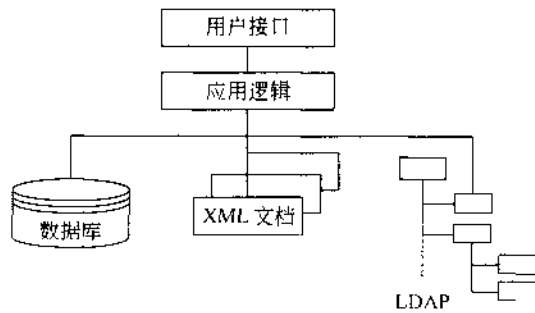


图 9-1 服务器三层结构示意图

三层结构的提出对应用的层次划分还没有终止，最终的目的是为了创建多层体系结构。在多层体系结构中，应用逻辑的划分是根据功能而不是根据物理方面来进行的。多层结构（参见图 9-2）的划分方式如下。

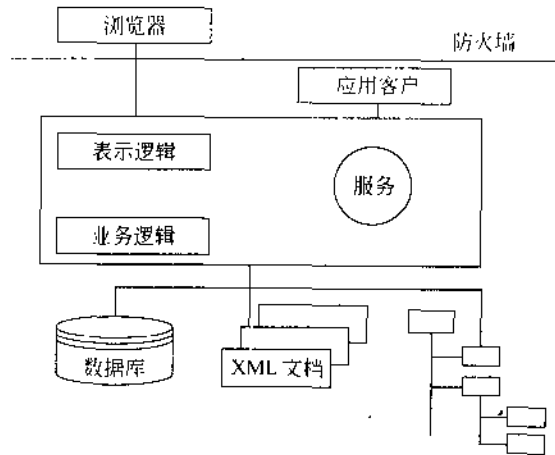


图 9-2 服务器多层结构示意图

- 用户接口层：负责处理用户与应用程序之间的交互过程。它可以是一个通过防火墙的 Web 浏览器，也可以是一般的桌面应用程序，甚至是无线设备或其他设备。
- 表示逻辑层：定义了用户界面要显示的内容和如何处理用户的请求。对于相应的客户，表示逻辑层可以有不同的版本。
- 业务逻辑层：通过与应用数据的通信，对应用的业务规则实现建模。
- 基础框架服务层：提供了应用系统需要的其他功能，如消息传输。
- 数据层：存放数据的数据库。

采用这种模式的目的是为了把数据和如何表示这些数据的部分分离出来，它们之间通过应用/业务逻辑来控制信息的流向。三层结构和多层结构的经典例子是 J2EE 服务器架构，在后面的章节将具体描述。

基于以上的各种服务器应用中，软件测试需要注意哪些呢？该如何进行基于服务器应用软件的测试？在接下来的章节里面，我们将对其中的重点部分做详细讨论。

## 9.2 基于 Web 服务器应用的测试

随着互联网络的普及和广泛应用, Web 的应用越来越广泛, 基于 Web 服务器的应用程序也变得非常普及, 因此, 对 Web 的要求也越来越高。

在 Web 测试中, 一般要从以下几个方面考虑。

- 功能测试: 主要测试页面功能和逻辑是否满足功能规格设计说明书的需求, 包括基本的逻辑、页面特性等。
- 用户界面测试: 包括用户页面是否和设计保持一致, 一些基本的页面元素(如按钮、表单、图片、文字等)的尺寸大小、边距、间距、布局是否和功能规格说明书相符, 文字是否有错误拼写等。
- 负载/压力测试: 检验 Web 程序的负载效率, 当大的客户访问量出现时, 需要对页面的执行效率进行模拟, 以实现整个系统处理交易的能力, 这也是为了保证系统的稳定性。在负载/压力测试过程中, 一般需要一些辅助的测试工具进行模拟测试。
- 安全性测试: Web 的安全性同样是 Web 测试的一个重要考虑因素, 包括页面、Web 服务器是否存在安全性漏洞、加密信息的保密性、防止黑客攻击能力等。
- 常用 Web 元素测试: 包括页面的链接、图片/文字信息、表单处理、脚本语言错误校验等。
- 兼容性测试: 各种操作系统(Windows、Mac OS、Solaris、Linux)与不同的浏览器以及浏览器版本的组合(Internet Explorer、Netscape 不同的版本)是否能正常执行, 在做兼容性测试时, 需要对一些已知的因素进行规约(例如 Netscape 浏览器本身的一些缺陷、Mac OS 的限制等)。
- 网络链接测试: 包括不同的网络链接方式的速度、效率, 同时需要考虑是否需要代理服务器, 在代理服务器上怎样执行 Internet 链接等。
- 其他方面的测试: 如系统的不同分辨率下的页面显示、流量测试。

### 9.2.1 常用的 Web 元素功能测试

构成常用的 Web 元素主要包括超级链接、图片、文字、HTML 语言、脚本语言、表单等。在基于 Web 的应用中, 我们对网页的功能测试需要注意以下几个方面。

#### 1. 页面链接测试

页面的链接是使用户从一个页面浏览到另外一个页面的重要手段, 在做页面链接测试的时候, 需要验证两个问题:

- 该页面是否存在, 如页面不能显示信息, 则视为页面链接无效。引起页面无效的因素有很多种, 主要有页面文件在 Web Server 不存在、链接的地址不正确等。
- 该页面是否跳转到所规定的页面, 主要使验证页面正确性, 这种测试也应该在 Web

功能测试部分被考虑。

## 2. 设计语言测试

这里的设计语言主要指 HTML 语言和不同的脚本语言,在某些情况下,HTML 语言随着客户浏览器的不同可能会产生不同的效果,因此,这也是测试中需要考虑的因素。如在 Netscape 4.7 里面,不能将表单内容限定成为只读属性,这样当表单的内容需要禁用或者限制使用的时候,程序必须考虑其他的方式来实现,比如利用 JavaScript 脚本进行处理。

## 3. Web 图形测试

Web 图形是一种常见的显示信息的手段,如 GIF 图片、Flash 等。很多时候,图形是和文本混合在一起使用的,因此,在 Web 图形测试的时候,不仅要确认文本是否正确,同时需要确认图片的内容和显示,如文字是否正确地环绕图片,图片的文字提示是否正确,图片所指向的链接是否正确等。当然,页面的负载测试中,图片显示也是一个重要因素,某些时候,在网络状态不好且图片文件比较大的时候,可能会遇到链接超时的错误,这些也需要被考虑在图形测试之内。图形测试还应当考虑显示问题,例如不同分辨率下的图形显示是否正确,需要浏览器附加程序支持的图形(如 Flash 动画)是否能正确加载等。

## 4. 表单测试

从设计的角度来看,表单是在访问者和服务器之间建立了一个对话,允许使用文本框、单选按钮和选择菜单来获取信息,而不是用文本、图片来发送信息。通常情况下,要处理从站点访问者发来的响应(即表单结果),需要使用某种运行在 Web 服务器端的脚本(如 PHP、JSP),同时在提交访问者输入表单的信息之前也可能需要用浏览器运行在客户端的脚本(通常是使用 JavaScript)。在进行表单测试的时候,需要保证应用程序能正确处理这些表单信息,并且后台的程序能够正确解释和使用这些信息。举个例子,用户可以通过表单提交来实现联机注册,当注册完毕以后,应该从 Web 服务器上返回注册成功的消息。整个程序处理过程如图 9-3 所示。

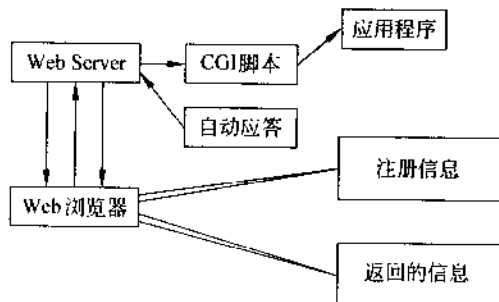


图 9-3 联机注册程序处理过程示意图

- (1) Web 服务器将表单传送到 Web 浏览器。
- (2) Web 浏览器显示需要访问者填写的注册信息表单。

- (3) 访问者将提交按钮和数据传送到 Web 服务器。
- (4) Web 服务器将表单数据传送给 CGI 脚本。
- (5) 处理表单结果的 CGI 脚本将数据格式化并将其发送给相应应用程序处理。
- (6) CGI 脚本产生一个验证消息并将其传送给 Web 服务器。
- (7) Web 服务器将验证消息发送给 Web 浏览器，以便显示。

## 9.2.2 Web 安全性测试

随着 Internet 的普及，网上购物、网上交易、电子银行等新的信息交易方式走进人们的生活，同时网络安全越来越不容忽视。在这些应用中，通常要使用 Web 页面来传送一些重要的信息，如信用卡信息、用户资料信息等，一旦这些信息被黑客捕获，后果将不堪设想。在 Web 的安全性测试中，通常需要考虑下列的情形。

- **数据加密：**某些数据需要进行信息加密和过滤后才能进行数据传输，例如用户信用卡信息、用户登录密码信息等。此时需要进行相应的其他操作，如存储到数据库、解密发送到用户电子邮箱或者客户浏览器。目前的加密算法越来越多，越来越复杂，但一般数据加密的过程是可逆的，也就是说能进行加密，同时需要能进行解密。
- **登录：**一般的应用站点都会使用登录或者注册后使用的方式，因此，必须对用户名和匹配的密码进行校验，以阻止非法用户登录。在进行登录测试的时候，需要考虑输入的密码是否对大小写敏感、是否有长度和条件限制，最多可以尝试多少次登录，哪些页面或者文件需要登录后才能访问/下载等。
- **超时限制：**Web 应用系统需要有是否超时的限制，当用户长时间不作任何操作的时候，需要重新登录才能使用其功能。
- **SSL：**越来越多的站点使用 SSL 安全协议进行传送。SSL 是 Security Socket Layer（安全套接字协议层）的缩写，是由 Netscape 首先发表的网络数据安全传输协议。SSL 是利用公开密钥/私有密钥的加密技术（RSA），在位于 HTTP 层和 TCP 层之间，建立用户与服务器之间的加密通信，确保所传递信息的安全性。SSL 是工作在公共密钥和私人密钥基础上的，任何用户都可以获得公共密钥来加密数据，但解密数据必须要通过相应的私人密钥。进入一个 SSL 站点后，可以看到浏览器出现警告信息，然后地址栏的 http 变成 https，在做 SSL 测试的时候，需要确认这些特点，以及是否有时间连接限制等一系列相关的安全保护。
- **服务器脚本语言：**脚本语言是最常见的安全隐患，如有些脚本语言允许访问根目录，经验丰富的黑客可以通过这些缺陷来攻击和使用服务器系统，因此，脚本语言安全性在测试过程中也必须被考虑到。
- **日志文件：**在服务器上，要验证服务器的日志是否正常工作，例如 CPU 的占用率是否很高、是否有例外的进程占用、所有的事务处理是否被记录等。
- **目录：**Web 的目录安全是不容忽视的一个因素。如果 Web 程序或 Web 服务器的处理不适当，通过简单的 URL 替换和推测，会将整个 Web 目录完全暴露给用户，

这样会造成很大的风险和安全性隐患。我们可以使用一定的解决方式，如在每个目录访问时有 index.htm，或者严格设定 Web 服务器的目录访问权限，将这种隐患降低到最小程度。

### 9.2.3 Web 负载测试

负载测试的作用就是在软件投入使用以前或软件负载达到极限以前，通过执行可重复的负载测试，预先分析出软件可承受的并发用户极限值和性能瓶颈，以便优化程序。Web 的负载测试是获得 Web 站点、程序性能、可靠性、稳定性等信息的重要手段。Web 的负载测试一般使用自动化工具来实现。

## 9.3 基于数据库应用服务器的测试

数据库应用服务器是构成数据库系统的重要部分，数据库系统应该包含计算机硬件、数据库、数据库管理系统、应用程序系统以及数据库管理员。一个完整的数据库应用服务器组成如图 9-4 所示。图中只表示由内到外的逻辑依赖关系，不表示包含关系。

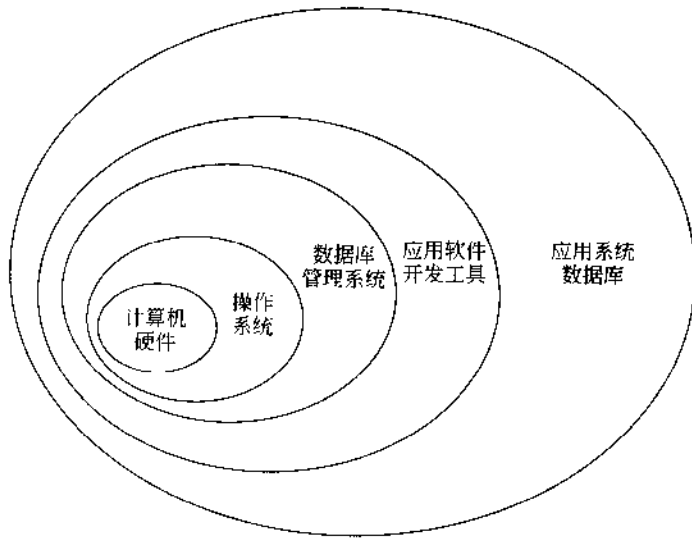


图 9-4 数据库应用服务器组成示意图

在本节中，我们将主要针对数据库服务器性能测试（大数据量测试，压力测试）以及并发控制过程的测试进行讨论。

### 9.3.1 数据库服务器性能测试

数据服务器性能测试主要从两个方面考虑，一个是大数据量测试，另一个是大容量的数据测试。



## 1. 大数据量测试

数据库的容量是表征数据库服务器性能的一个重要标准,在测试中,当大数据量在数据库中存在时,系统的性能肯定会受到影响,合理的数据库服务器管理程序以及数据库结构将会将这种变化降低到最小,例如在大数据量(成千上万、几十万条记录)处理时,通过数据库表的索引定义、数据库表空间、log 大小将会直接影响到数据库的存储/检索性能和速度,另外,数据库服务器的 CPU 占用率也会受到影响,在数据库服务器上,不应该由于这样的操作导致数据库系统的负荷超载甚至崩溃。

例如有这样一个查阅书籍销量的 Web 程序,使用 B/S 结构,用户可以通过浏览器页面来访问 Web 服务器,Web 服务器从另外的 Linux 上的 Oracle 数据库服务器上读取数据,页面上显示所有当天/当月的售书信息,其表结构如表 9-1 所示。

表 9-1 表的结构 saledbooksinfo

字段名	类型定义	含义	注释
saled_date	date	售出的日期	不为空
saler_id	int	销售者 id	不为空
book_ISBN	varchar	书的 ISBN	不为空
book_name	varchar	书名	
book_price	double	书价	
coupon_price	double	折扣金额	
invoice_id	int	发票 id	

进行数据库性能测试时,需要使用一个自己编写的工具软件来动态给数据库加压,从不同的数据量和执行的语句,来测试数据库的执行速度,必要时甚至需要测试 CPU/内存占用率等。可以进行如下测试设计:

- 所有页面可能提交 SQL 语句跟踪,例如我们可以通过工具来取得执行每个页面刷新或者新链接以及其他操作(删除,排序等)的 SQL 语句,例如“select \* from saledbooksinfo where”(页面用户操作的时间和排序条件)。
- 通过向数据库里面批量加入适当的满足条件的语句。
- 执行操作,通过监控程序来获取执行速度,进程占用 CPU 和内存信息以及其他我们需要分析的信息。
- 获取测试结果,找到数据库结构或者程序需要优化的地方。
- 在本例中,我们可以使用 Oracle 本身提供的 SQL PLUS 工具来实现执行速度的跟踪(set timing on),然后输出(spool)到一个指定的文件里面,获取执行每一个 SQL 的 CPU 占用或者内存占用情况(top 命令),最后形成 Excel 表格,如表 9-2 所示。

## 2. 大容量数据测试

在数据库性能测试过程中,还需要考虑大容量数据测试。例如在测试中,我们可以使

用带有视频、图片的数据，假设做这样的测试，在数据库中插入 1000 个图片文件，每个图片文件的大小为 4MB，这样，数据库在插入这些数据后的容量为  $1000 \times 4\text{MB} = 4000\text{MB} = 4\text{GB}$ ，我们可以通过在空表中插入 (insert) 1000 条数据，选择 (select) 900 条数据、更新 (update) 900 条数据、删除 (Delete) 900 条数据这样一个过程来进行测试，跟踪所需要的时间、进程占用的 CPU 及内存信息等，最后得出结果、形成表格，从而分析数据库的性能情况。

表 9-2 大数据量测试结果记录表

执行的操作项	操作的数据数量	执行使用的时间	进程占用 CPU	进程占用的内存
Select * from saledbooksinfo where (用户条件)	1000			
	5000			
	10 000			
	100 000			
Insert saledbooks Values (用户值)	1000			
	5000			
	10 000			
	100 000			
Delete from saledbooks where (用户条件)	1000			
	5000			
	10 000			
	100 000			
Update saledbooks set (用户设定) where (用户条件)	1000			
	5000			
	10 000			
	100 000			

数据库容量和性能测试是至关重要的，不合理的表结构以及程序中不合理的代码将使数据库的性能降低，甚至崩溃，因此，性能测试为数据库的优化过程提供了指导方向。

### 9.3.2 数据库并发控制测试

数据库的并发控制能力是指在处理多个用户在同一时间内对相同数据同时进行访问的能力。一般的关系型数据库都具备这种能力，日常应用系统中也随处可见，例如火车的售票系统、银行数据库系统。举个例子来说明，在火车的售票系统中，有这样一个过程。

- (1) A 售票点通过网络从源数据库读出某车次的车票剩余张数为  $n$  ( $n=100$ )。
- (2) B 售票点通过网络从源数据库读出该车次的车票剩余张数也为  $n$  ( $n=100$ )。
- (3) A 售票点卖出一张该车次的车票，将  $n-1$  (99) 写回源数据库。
- (4) B 售票点也卖出一张该车次的车票，将  $n-1$  (99) 写回源数据库。

这样就存在并发控制的问题，卖出了两张票，而数据库里面只有 1 条数据减少。这样，下次读取数据的时候，源数据就不准确了，从而会带来数据的错误。如果按照上面的操作

顺序执行，A 对源数据库的修改就被丢失。

并发控制带来数据的不一致问题，被称为“数据库并发控制过程冲突”，如图 9-5 所示，在实际的测试过程中，我们必须对这样的冲突进行测试设计。我们主要是通过逻辑判定来设计测试用例。在并发控制过程冲突中，主要包括三类：丢失数据，不可重复读数据和读“脏”数据。



图 9-5 火车票系统的数据并发过程冲突示意图

### 1. 丢失数据

刚才的例子就是一个典型。当事务 A 和 B 对同一个数据源进行修改，B 提交的结果破坏了 A 提交的结果，导致 A 对数据库的修改失效，如图 9-5 所示。

### 2. 不可重复读数据

不可重复读数据是指事务 A 在读取数据后，事务 B 对其进行了修改并执行了更新操作，当事务 A 无法再现前一次读取的结果。举个例子来说明：

- 事务 A 从数据库表中读出整数  $X=10$ ， $Y=20$  值进行求和运算  $Z=X+Y=30$ 。
- 事务 B 从相同的数据库中读出 X 值  $X=10$ ，对 X 乘以 5 后写入原 X 值 ( $X=X*5=50$ )，提交事务 B。

此时，事务 A 处理的结果是： $Z=X+Y=10+50=60$ ，事务 B 对数据的操作已经影响了原来的结果。

### 3. 读“脏”数据

读脏数据是指事务 A 修改某一数据（或者执行某一操作），并将其写到数据库，事务 B 读取相同数据（或者执行某一关联操作）的时候，事务 A 由于某种原因撤销操作（即进行事务回滚），此时事务 A 已经将原来的数据还原，而事务 B 所读取的数据和数据库中真实记录就不一致，此时事务 B 读到的数据就是脏数据。例如，事务 A 从数据库表中读出整数  $X=10$  并乘以 5 写入 A。事务 B 读取 X 的值为 50，此时，事务 A 执行回滚操作，将 X 的值恢复成 10，此时 B 读取的数据 50 就是“脏”数据。

在数据库应用系统中，一般会使用加锁（locking）技术来进行并发控制。如图 9-6 所示，在火车售票系统中，当 A 事务对数据进行修改之前先对数据库表进行加锁，当 B 事务请求对数据库表进行修改时需要重新请求加锁，此时若 A 事务锁未被释放，B 将不能对数据库进行修改操作，这样，B 对数据库表进行操作时就是对已经更新的数据进行操作，此时防止了数据库并发控制的冲突产生，如图 9-6 所示。按照加锁的等级和操作权限，数据库加锁可以分为一级/二级/三级封锁协议，具体请参考数据库相关方面的书。

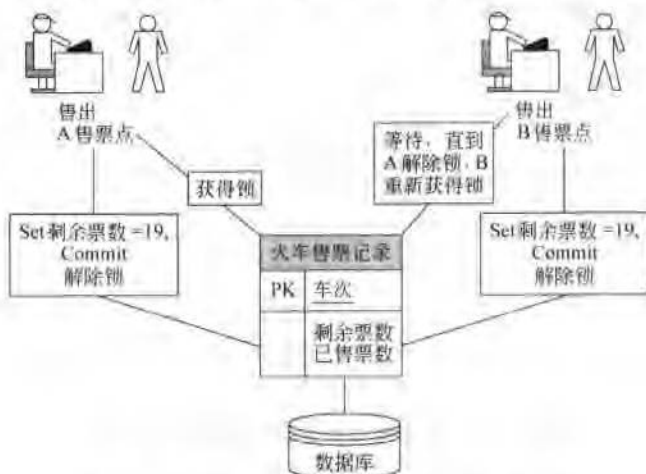


图 9-6 火车票系统的加锁技术的并发控制示意图

在数据库并发控制测试过程中，我们需要针对程序控制的流程来设计测试。测试的重点是并发控制逻辑分析以及锁控制的逻辑分析，设计并发控制的测试过程分为两个部分：

- 并发流程分析。按照数据库处理的流程来设计测试的逻辑重点，分析并发控制的点、事务锁的使用。
- 并发控制测试分析。按照并发控制的实现过程以及事务锁的基本机制，设计相应的测试过程以及测试用例。

当然，在并发控制测试中，我们可能要涉及到更为复杂的测试过程，例如多线程应用程序的并发控制处理、数据的死锁控制以及分析等，这里不再多描述。

## 9.4 基于 J2EE 平台的测试

以 Java 技术为基础的 J2EE 构架为企业提供了一个快速构造大型、可伸缩的、分布式的电子商务框架。本节主要讨论 J2EE 平台应用系统的测试方法和技术。

### 9.4.1 J2EE 概述

什么是 J2EE? J2EE 是 Java 2 Enterprise Edition 的缩写，J2EE 平台为企业级的应用提供了多点分布式应用模型。按照功能，应用程序的逻辑被分割成不同的构件，这些不同的应

用程序构建共同组成了 J2EE 应用程序，并依据 J2EE 环境中不同的层次来安装在不同的机器上，图 9-7 描述了基于下述清单中 J2EE 的应用。

- 客户机器上加载客户级组件。
- J2EE 服务器上加载 Web 应用组件。
- J2EE 服务器上加载业务组件。
- 在 EIS (Enterprise Information System, 企业级信息系统) 上加载信息系统软件。

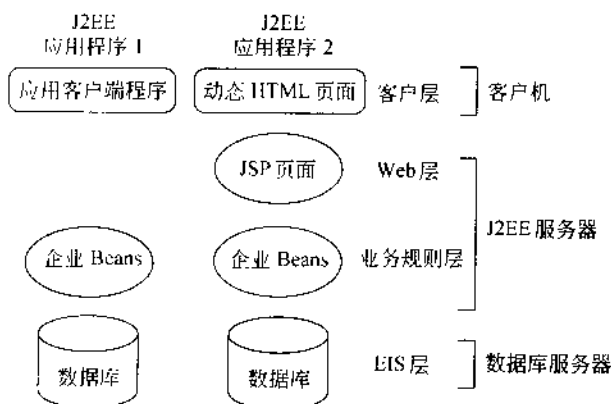


图 9-7 J2EE 平台应用的四个层次

### 1. J2EE 组件

J2EE 应用是由多个组件构成。一个 J2EE 组件是具有自我包含功能的软件单元，它使用类和文件来关联其他的组件并嵌入到一个 J2EE 应用程序中。J2EE 规格说明中定义了下列组件：

- 在客户端运行的 Java 应用程序和 Applet。
- 在服务端运行的 Java Servlet 和 Java Server Pages (JSP) 等 Web 组件。
- 在服务端运行的企业级的 JavaBeans (EJB) 组件。

**Web 组件：**J2EE 的 Web 组件可以是 Servlets 或者 JSP，Servlets 用来动态过程请求和构成相应的 Java 语言类库，JSP 页面基于文本的文档来执行类似 Servlets 的功能，但 JSP 能更加简单自然的创建静态类库。

在应用程序编译阶段，静态的 HTML 页面及 Applet 能和 Web 组件绑定，但我们不能认为 Web 组件属于 J2EE 的规格需求，服务端的有效类也可以和 Web 组件绑定，和 HTML 页面一样，它们也不能看做是 Web 组件。

和客户层一样，如图 9-8 所示，Web 层必须包含 JavaBeans 组件用来管理用户的输入，并将这些输入发送给运行在业务层的 JavaBeans 来处理。

**业务组件：**业务层的代码用于解决通用的业务领域，如银行、零售业、金融业需求逻辑性的编码，它由运行在业务层的 JavaBeans 来控制，如图 9-9 所示，JavaBeans 从客户程序中接受数据，处理（如果需要的话）并发送到企业信息系统层（EIS）来进行存储，反之，JavaBeans 也可以从存储的信息中获取数据，处理并返回到客户端。

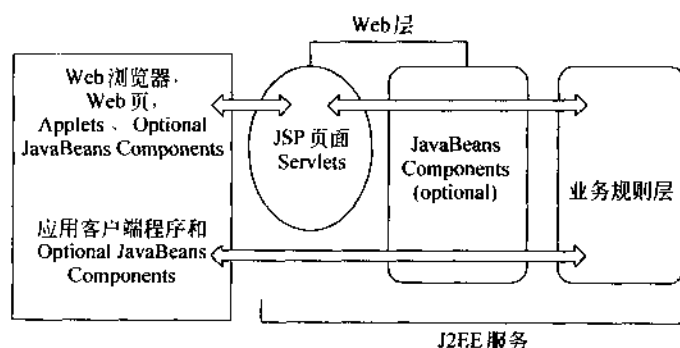


图 9-8 Web 层和 J2EE 应用

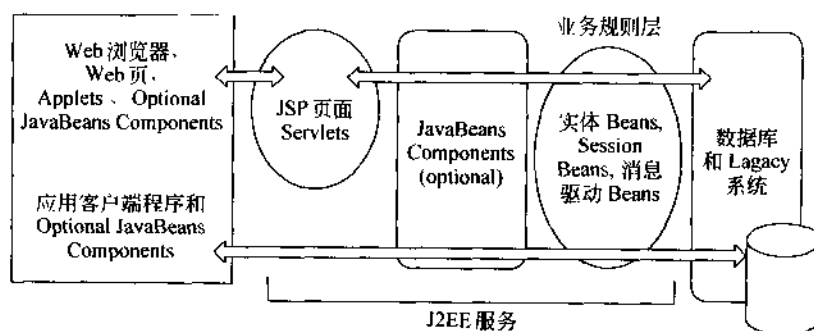


图 9-9 业务层和 EIS 层

## 2. J2EE 客户端

J2EE 客户端可以是 Web 客户端或者应用程序客户端。

- **Web 客户端。**一个 Web 客户端包含两个部分：由各种 Web 的混合语言（HTML，XML 等）组成的 Web 页面和 Web 浏览器。Web 页面通过运行在 Web 层的组件产生，Web 浏览器将负责与 Web 服务器的页面数据交互。一个 Web 客户端有时也被称为瘦客户（thin client），瘦客户通常什么都不做，只负责做一些简单的工作，如数据库查询，执行复杂的业务规则等操作不会由瘦客户来执行。当使用瘦客户的时候，这些繁重的工作将在 J2EE 服务端执行，基于 J2EE 的服务器技术能在安全性、速度、服务、可靠性之间做出平衡。
- **Applet。**Web 页面可以通过嵌入的 Applet 程序来接受来自 Web 层的数据，一个 Applet 程序是由 Java 编写的小应用程序，它负责在 Web 浏览器上执行。
- **应用程序客户端。**一个 J2EE 应用客户端运行在客户机上，它为用户提供了一种方法，用以控制一些由富裕的用户接口提供的任务需求。典型的应用是使用 Swing 类或者抽象窗口工具（AWT）提供的 API，来创建用户图形接口（graphical user interface），当然基于命令行的程序也能如此。应用客户端程序能通过运行在业务层来访问企业级的 JavaBeans，然而，如果应用程序需要授权，J2EE 程序也可以和运行在 Web 层的 Servlet 打开 HTTP 连接进行通信。

### ► JavaBeans 组件架构

服务层和客户层必须使用基于 JavaBeans 组件架构的组件,用以管理在应用客户端(或者 applet)和运行在 J2EE 服务器上的组件进行通信,或者是服务组件和数据库之间的通信。在 J2EE 的规范中,JavaBeans 组件没有被包括在 J2EE 组件中。JavaBeans 有一些常量以及使用 get/set 方法来访问这些常量,JavaBeans 组件在设计和实现中是典型的、简单的使用方法,但必须符合 JavaBeans 组件结构大纲的要求来进行命名和设计。

### ► J2EE 服务通信

图 9-10 中清晰地描绘了组成客户层的不同元素,客户层和 J2EE 服务器的业务层使用直接或者间接的方式进行通信。图 9-10 中客户端运行在浏览器上,通过 JSP 页面或者 Servlet 来被执行。

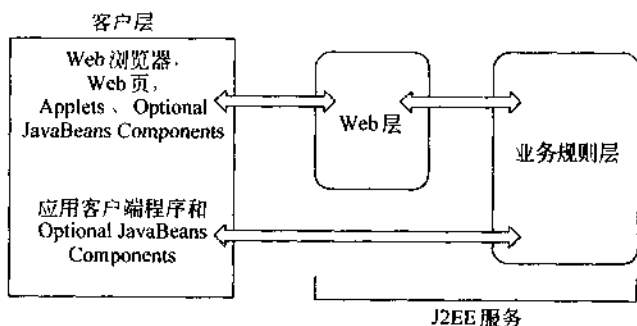


图 9-10 业务层和 EIS 层

## 9.4.2 基于 J2EE 应用的单元测试技术

基于 J2EE 架构的测试非常昂贵和复杂,尤其是进行系统测试时,这里我们只讨论一些白盒测试方法来实现 J2EE 单元测试的过程和方法。

### 1. 测试原则

Java 语言是一种支持面向对象的语言,通常情况下,我们可以将程序的一个单元看成一个独立的类。因此进行单元测试的重点就是针对这些类进行测试。

最佳的方法就是测试类中定义的方法。如果父类为抽象类,为进行测试可以简单定义一个具体子类,它只利用了父类的抽象方法,而无其他行为。然而,这样做可能会忽略建立的子类将随着父类结构变化而变化,以及编译器或者其他测试过程中不能捕获的信息,从而导致一些遗漏,我们需要遵循一些基本的原则来进行测试:

- 不需要测试 get 和 set 这样的行为。
- 一个方法至少要测试一次。
- 各种访问、修改器也对测试产生影响。

### 2. 测试步骤

- 判断组件的功能:通过定义应用的整体需求,然后将系统划分成几个对象,我们

需要对组件的基本功能十分清楚。因此，J2EE 单元测试实际上也属于设计过程的一部分。

- 设计组件行为：依据所处理的过程，可以通过一个正规或者非正规的过程实现组件行为的设计，我们可以使用 UML 或者其他文档视图来设计组件行为，从而为组件的测试打下基础。
- 编写单元测试程序（或测试用例）确认组件行为：这个阶段，应该假定组件的编码已经结束而且组件工作正常，我们需要编写单元测试程序来确定其功能是否和预定义的功能相同，测试程序需要考虑所有正常和意外的输入，以及特定的方法能产生溢出。
- 编写组件并执行测试：首先，创建类及其所对应的方法标识，然后遍历每个测试实例，为其编写相应代码使其顺利通过，然后返回测试，继续这个过程直至所有实例通过。此时，停止编码。
- 测试替代品：对组件行为的其他方式进行考虑，设计更周全的输入或者其他错误条件，编写测试用例来捕获这些条件，然后修改代码使得测试通过。
- 重整代码：如果有必要，在编码结束时，对代码进行重整和优化，改动后，返回单元测试并确认测试通过。
- 当组件有新的行为时，编写新的测试用例：每次在组件中发现故障，编写一个测试实例重复这个故障，然后修改组件以保证测试实例通过。同样，当发现新的需求或已有的需求改变时，编写或修改测试实例以响应此改变，然后修改代码。
- 代码修改，返回所有的测试：每次代码修改时，返回所有的测试以确保没有打乱代码。

### 3. JUnit 框架简介

JUnit 是一种可重复编写 Java 程序测试的框架，主要用于单元测试。

#### ► JUnit 的目标

- 创建一个通用的测试框架，将测试代码封装入对象中，从而使开发者同步设计并配置自己的单元测试。
- 让测试代码不会因为时间推移而变化，具有保值性，使测试代码标准化，从而保证编写原始测试代码之外的人也可以执行和维护测试，这样，多人的测试用例联合和执行成为可能而不会导致混乱。
- 在新创建的测试用例和旧的测试用例之间起到杠杆作用，JUnit 框架能够为重复测试提供便利。
- 将测试代码从系统代码中剥离开，二者可同步发展。

#### ► JUnit 的框架成员逻辑分析

- 被测试的对象（类、多个类、子系统）。
- 对测试目标进行测试的方法与过程集合，可将其称为测试用例（TestCase）。
- 测试事务的集合，可容纳多个测试用例，将其称作测试组件（TestSuite）。
- 测试结果（TestResult）的描述与记录。



- 每一个测试方法所发生的与预期不一致状况的描述, 称其测试失败 (TestFailure)。
- JUnit Framework 中的出错异常 (AssertionFailedError)。

► JUnit 框架功能以及原理描述

JUnit 框架的基本结构如图 9-11 所示。

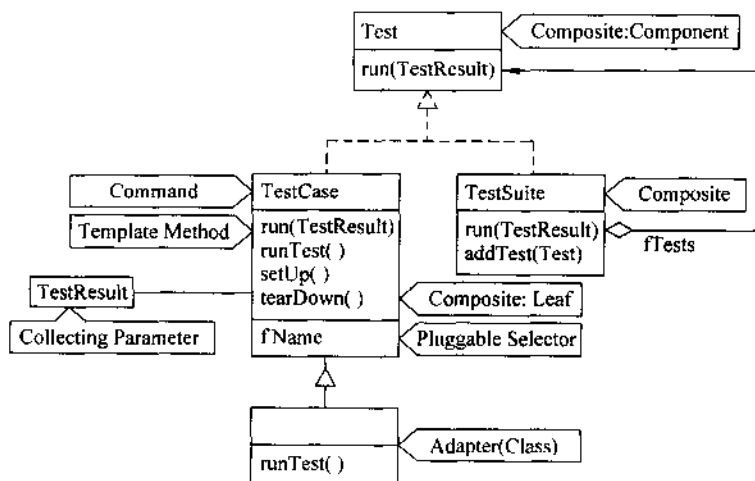


图 9-11 JUnit 框架的基本结构

- 测试接口与测试用例 (TestCase)、测试组件 (TestSuite) 形成了复合 (Composite) 结构, Run (TestResult) 则是 Composite Method。Test 为 Component, 派生出的 TestCase 为 Leaf, 是测试的执行元素; TestSuite 为 Composite, 可通过 addTest (Test) 来容纳 Test 组合 (TestCase or TestSuite) 形成测试包。
- TestCase 可在框架中视为测试单元的运行实体。用户可以通过它派生自定义的测试过程与方式 (单元), 利用 Command Pattern 与 Composite Pattern 使其形成可组合装配的可扩展测试批处理。TestCase 本身的运作操作为 run (TestResult), 其中分别执行 setUp(), runTest(), tearDown() 来架构测试过程。Template Method Pattern 使用户无须了解执行框架的过程细节, 而只需重定义特性化的测试预处理、测试单元过程以及测试完毕, 测试三个 Template Method 就能使测试正确工作。
- 用户层可通过 Java 的匿名内部类 (anonymous inner classes) 来集成化重载 runTest() 形成特性测试类, JUnit 3.0 版本以后则支持利用 Java 的 Class 属性来动态框架后台生成这些特性测试单元类, 用户只需在 TestCase 派生类的 testXXX() 即可, 在 Suite 中使用 return new TestSuite (MyTestCase.class) 后, 该 TestSuite 就包含了所有 testXXX 为测试过程的 TestCase 类对象集合。
- Assert 类包含了 assertEquals (...), assertEquals (...), assertEquals (...) 等静态工具方法 (static tools methods), 为使用户对系统所了解的类型尽可能少, JUnit 框架将 Assert 作为了 TestCase 的超类, TestCase 同时继承了 Assert 的实现与 Test 接口 (class Adapter pattern)。用户可在 TestCase 中直接调用这些 assertXXX (...) 等静态工具。
- TestResult 描述了整个测试执行过程的测试结果, JUnit 框架将其作为参数传递与

Test 的 run(.) 间, 当测试任务执行完毕后, TestResult 内包含了所有 TestCase 的测试结果。框架默认的 TestResult 为 TextTestResult, 用户也可以自定义其 TestResult 的回显格式, 譬如 HTMLTestResult。

TestFailure 描述了测试过程中的错误信息, TestResult 通过 Vector 容纳了测试过程中生成的 TestFailure, 将其作为测试结果的参数依据。

#### ► 使用 JUnit 编写测试

使用 JUnit 编写单元测试用例主要从以下几个方面考虑。

##### ● 编写单个测试

JUnit 测试不需要人工的判断和解释, 而且它可以在同一时间内同时执行多个测试。在实际编写 JUnit 单个测试程序中, 我们需要按照以下的步骤进行:

- (1) 创建一个 TestCase 的实例 (如图 9-12)。
- (2) 创建一个构造器, 它能接受字符串 (String) 类型作为参数并可以传送给父类。
- (3) 重载 runTest() 方法。

(4) 当需要检验一个值或者变量时, 调用 assertTrue() 方法, 它将返回一个布尔型值来标识测试成功 (TRUE) 或者失败 (FALSE)。

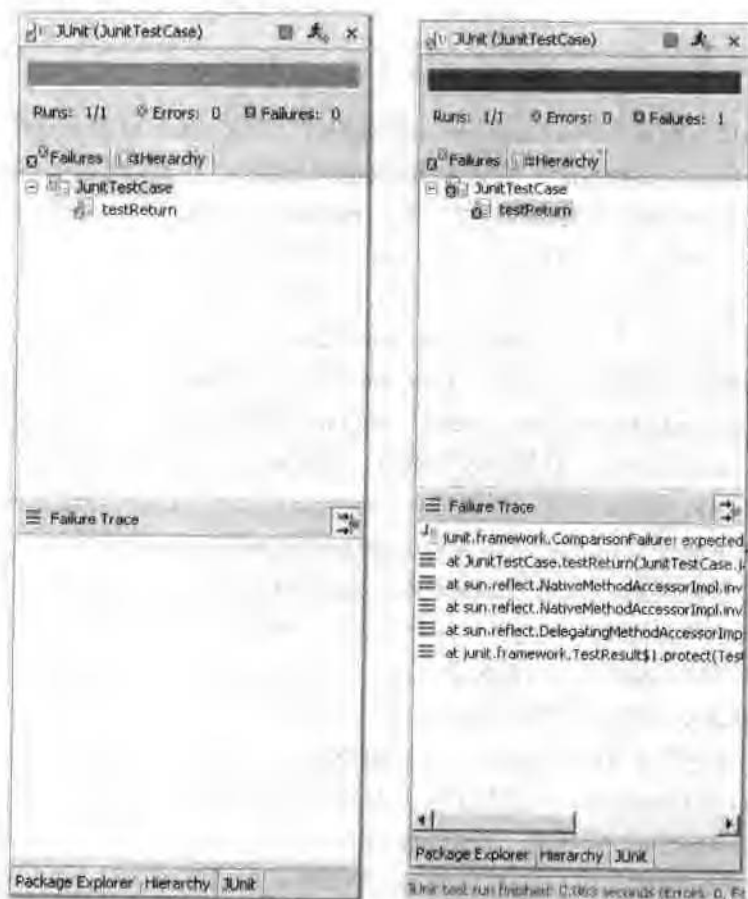


图 9-12

如果你想写一个类似于已经完成的测试实例，可以使用固件（注：原文为 *fixture*）；而如果需要编写多个测试实例，则需要创建测试组件（*Suite*）。

- 编写固件

- (1) 创建一个 *TestCase* 的子类。
- (2) 创建一个构造器，它能接受字符串（*String*）类型作为参数并可以传送给父类。
- (3) 在固件的每个部分各加入一个实例变量。
- (4) 重载 *setup()* 方法来初始化变量。
- (5) 重载 *teardown()* 方法来释放在 *setup()* 过程中所使用的资源。

- 编写 *TestCase*

- ◆ 在固件类中加入测试实例的方法，确认其为 *public*，并不能通过映射来调用。
- ◆ 创建 *TestCase* 类的一个实例，通过测试实例方法把名称传递给构造器。

- 编写 *TestSuite*

*TestSuite* 是为了同时运行多个 *TestCase* 而实现的。在 *JUnit* 里面，*TestSuite()*、*TestCase()* 都是类模块，其含义和前面广义上提到的测试组件和测试用例概念是不同的，但就其内涵来说，两者是一致的。

关于 *JUnit* 更多的学习，请参阅 *JUnit* 的主页（[www.JUnit.org](http://www.JUnit.org)），在下面的例子中，我们将基于 *JUnit* 框架来实现单元测试的分析。

### 9.4.3 实用对象的单元测试

实用对象的操作大多独立于系统，一般低层次操作可用于很多对象，例如，对字符串的解析功能，我们可以使用一个实用类来执行。

当我们在使用 *HTML* 访问数据的时候，通常会遇到字符串转义的问题，如从数据库里面检索出来的 *>*、*<* 符号，会干扰 *HTML* 的正常显示，由此可以用 “*&gt;*” 和 “*&lt;*” 来分别替换大于和小于符号，使用一个实用类来解决这个问题。

从上面的描述中可以知道组件的基本需求，下面需要进行组件设计。例子中，组件的惟一功能就是对字符串进行解析和清理，我们假定存在一个方法，接受字符串输入，返回值是清理后的字符串。将整个组件看成是一个提供了标准的输入输出接口的黑盒子。假定调用的组件名称为 *HMTLScriber*，调用的方法为 *scrub()*。

接下来，我们需要编写单元测试。测试类为 *HMTLScriber*，测试用例为 *HMTLScriber*。*HMTLScriberTest*，扩展名为 *TestCase*，使用 *JUnit*、*textui*、*TextRunner* 为运行器（注：*JUnit* 提供了两种运行器，即文本运行器和图形运行器）。测试用例代码设计如下：

```
import JUnit.Framework.TestCase;

public class HMTLScriberTest extends TestCase {
    public HMTLScriberTest(String arg0) {
        super(arg0);
    }
}
```

```

    public static void main(String[] args) {
        JUnit.textui.TestRunner.run(HTMLScrubber.class);
    }
}

```

我们需要在测试实例中加入调用方法，在本例中，需要测试组件以“&gt;”替换“>”，以“&lt;”来替换“<”，并可以处理空字符串以及空输入情况。

参考 14.4.1 节中测试用例设计原则，在测试用例中定义如下的测试方法。

```

public void testAlreadyScrubbed() throws Exception{} //已经替换的字符串测试
public void testEmptyString() throws Exception{} //字符输入为空测试
public void testGreaterThanAndLessThanScrubbing ()
throws Exception{} //字符串中">"和"<"并存时的测试
public void testGreaterThanScrubbing() throws Exception{}
//字符串中含">"符号替换测试
public void testLessThanScrubbing() throws Exception{}
//字符串中含"<"符号替换测试
public void testMultipleGreaterThanScrubbing() throws Exception{}
//字符串中含多个">"符号时的测试
public void testMultipleLessThanScrubbing() throws Exception{}
//字符串中含多个"<"符号时的测试
public void testNull() throws Exception{} //空输入测试

```

每个测试类别都描述名称，类型为 `public`，溢出类型为 `exception`，现在需要进一步定义每一个实体。以 `testLessThanScrubbing()` 为例：

```

public void testLessThanScubbing() throws Exception{
    HTMLScrubber scrubber=new HTMLScrubber();

    String before="test < test"; //定义传入 scrub() 方法的字符串，用于输入
    String after="test &lt; test"; //定义预计的输出字符串

    assertEquals("String should be equal", after, scrubber.scrub(before));
    //通过 JUnit 的 assertEquals() 方法来进行测试判定
}

```

字符串 `before` 标识发送到 `scrub` 方法的值，而 `after` 表示预想的输出，将被用于测试输出。在 `assertEquals()` 方法中，第一个参数为描述行字符串，第二个参数为预计值，第三个值为组件的实际取值，如果实际值（从 `scrub` 方法中返回）匹配预计值，则测试通过，如果不一致，则测试失败。

同样，按照上面的步骤，我们定义其他的方法。这样，我们就已经完成了 `scrub()` 方法

所有正常情况的测试。

接下来,我们需要对需求中的一些异常情况进行测试,例如空字符串。当输入为空(`null`)时,应该产生溢出 `NullPointerException`, `scrub()` 方法应返回空字符串。需要编写确认此行为的测试代码,如下所示:

```
public void testEmptyString() throws Exception{
    HTMLScrubber scrubber=new HTMLScrubber();
    assertEquals("String should be equal", "", scrubberscrub(""));
} //空字符串被程序处理后应返回为""

public void testNullThrowException() throws Exception{
    HTMLScrubber scrubber=new HTMLScrubber();
    try{
        scrubberscrub(null);
        assert(false); //空输入时抛出异常,并返回测试失败
    }
    catch(NullPointerException e){
    }
}
```

最后,定义 `HTMLScrubber` 类如下:

```
public class HTMLScrubber {
    public String scrub(String s){
        StringBuffer buf=new StringBuffer();
        for (int x=0; x<s.length(); x++){
            buf.append(scrubbed(s.charAt(x)));
        }
        return buf.toString();
    }
    private String scrubbed(char e)
    {
        //此处定义字符转换函数
    }
}
```

## 9.4.4 事务对象的单元测试

事务对象主要用于在系统中存储数据,用户对象即为其中的一个例子。在事务对象中,一旦剔除了 `getter` 和 `setter` 方法,事务对象中需要测试的地方就很少了。举个例子来说明,假定作为应用的一部分要创建一个用户对象,此对象负责存储用户名以及 `String` 类型的一

个 `Vector`，表示应用中用户访问过的广告主页。另外，用户对象必须能够表示成用逗号分割的 `Vector` 字符串，也必须接受此格式的字符串，并将其转换为一个 `Vector`。

我们定义的事务对象类命名为 `user`；假定它有一个成员变量是 `name`，使用一个 `SetPathString()` 方法来实现字符串到 `vector` 的转化，使用 `GetPathString()` 来接受用户输入。按照前面提到的原则，成员变量 `name` 的 `setter` 和 `getter` 方法是不必测试的，但负责转化的 `SetPathString()` 则必须要测试，代码如下：

```
public class user{
... //基本的事务对象类 user 以及其变量和方法定义
public String GetPathString( )
{
...//获取用户输入
    Vector v=getPath( );
    if(v==null){
        return "";
    }

    StringBuffer b=new StringBuffer();

    for(int x=0;x<v.size();x++){
        b.append(v.elementAt(x));

        if(x+1!=v.size()){
            b.append(",");
        }
    }
    return b.toString();
}

public void SetPathString(String newPathString){
...//转换用户输入
    if (newPathString==null)
    {
        setPath(null);
        return;
    }

    StringTokenizer t=new StringTokenizer(newPathString,",");

    Vector v=new Vector();
    while(t.hasMoreTokens()){
        v.addElement(t.nextToken());
    }
}
```

```

    setPath(v);
}

```

用户类以及所有的事务属性已经定义好，我们接下来做测试实例代码的编写，这里，我们使用一个测试过程 `testStringPathEqualsStringPassOut()` 来判断事务处理结果是否正确（假定的 `String` 输入为 1~7 字符串索引）。

测试实例的代码如下：

```

public void testStringPathEqualsStringPassOut() throws Exception{
    User u=new User();           //创建用户对象实例
    String in="1, 2, 3, 4, 5, 6, 7"; //预定义的输入
    u.SetPathString(in);         //调用事务对象处理过程
    assertEquals(in, u.GetPathString()); //判断事务对象处理结果是否在期望的定义中
}

```

事务对象的测试过程和实用对象的测试过程基本类似，主要通过对事务对象的基本特征（调用的方法和过程）和事务对象处理逻辑（`get` 和 `set` 方法）的把握来设计有效的测试。

### 9.4.5 servlet 的单元测试

根据 `servlet` 的定义，一个 `servlet` 是在请求/应答环境下实现的。一个 `servlet` 的 `service()` 方法对请求及其响应实施动作，我们可以使用任何可能出现的参数组装请求对象，然后检验相应对象以确保出现预计结果。

这种方法需要对请求和相应对象进行分类，并在子类中加入测试支持，如果一个 `servlet` 从响应中得到输出流，并向其写入信息，应该确保其实际写入了正确的数据，因此可以覆盖响应以提供一个输出流，可以将其存储在测试方法中检验的响应对象里。

举个例子，我们需要编写一个 `servlet`，带有一个参数 `id`，对应即将显示的产品序号，然后 `servlet` 输出适当的 `HTML` 显示指定序号的产品信息。对此 `servlet`，只有 1、2、3 序号有效。我们使用四个部分来完成基本定义。

首先，我们需要用一个程序类来定义 `servlet` 的行为，以显示产品信息，在这个类中，提供 `service()` 方法来定义针对 `servlet` 的请求和响应方法并通过 `log` 输出。程序代码如下：

**DisplayServlet.java:**

```

public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException{
    //定义 service 方法调用两个参数，这两个参数是 javax.servlet.ServletRequest 类和
    //ServletResponse 类的实例，分别表示 servlet 的请求和相应
    String body =null;
    int prodID=0;
    try{
        prodID=Integer.parseInt(request.getParameter("id"));
    }
}

```

```

    }

    catch (NumberFormatException e) {
        WriteLog(response.getWriter(), "You must specify a product id.");
        return;
    }

    if (prodID>1 && prodID<3){
        body="Product id is"+ prodID;
    }
    else{
        body="The product id is Invalid.";
    }
    WriteLog(response.getWriter(), body);

}

protected void WriteLog(PrintWriter out, String msg){
//WriteLog 函数将 ServletResponse 类中 printWriter() 的输出打印出来
    Out.println(msg);
}
}

```

接下来, 需要对这个 servlet 进行测试, 测试通过三个过程来进行 (实际上, 就是三个测试实例), 即测试有效输入 (testValidID()), 无效输入 (testInvalidID()) 和异常输入 (testNoID()), 利用 JUnit 的 assertEquals() 方法来判定 servlet 的请求是否能返回正确的响应。

```

public void testValidID() throws Exception{
    RequestTest request=new RequestTest();
    Request.setParameter("id","2");
    ResponseTest response=new ResponseTest();
    DisplayServlet servlet=new DisplayServlet();
    Servlet.service(request, response);
    String expected="Product id is 2";
    assertEquals(expected, ((TestWriter) response.getWriter()).getOutput());
}
//这里以其中的 testValidID() 为例, 其余的两个测试类同

```

在上面的程序中, 我们还使用了一个 RequestTest 类, 这个类主要是通过 getParameter() 和 setParameter() 方法来实现 servlet 请求的实现过程。

**RequestTest.java:**

```

public String getParameter(String key){
    return (String)paramMap.get(key);
}

```



```
}  
.  
  
public void setParameter(String key, String value){  
    paramMap.put(key, value);  
}  
...//关于 ServletRequest 的其他方法调用, 都使用缺省值, 此处省略。
```

## 9.4.6 JSP 单元测试

单元测试 JSP 程序比较难, 尤其是 JSP 程序里面嵌入了大量 Java 代码的时候, 因为 JSP 不是一个 Java 对象, 不能使用 JUnit 框架来测试它, 最好的办法是使用自动化工具来根据预定义的脚本检验 JSP 页面, 如使用 Rational Robot。请参阅本书的关于自动化测试的内容。

## 9.4.7 数据库访问层的单元测试

测试数据库访问代码的方式像数据库访问策略一样多, 从带有内嵌于代码中的 SQL 语句 JDBC 到第三方软件解决方案, 存在多种编写数据库访问层的方案。本节定位于测试此类代码的一般方法。

在一个典型应用中, 存在一些代码执行数据库访问, 即读写数据库, 并进行处理, 例如一个数据库驱动层软件, 使用 JDBC 和 Oracle 数据库相连, 并从中取出数据处理到另外的一个 MySQL 的数据库中。这样, 我们就需要考虑 JDBC 与 Oracle 以及和 MySQL 数据库的连接引擎、数据处理等一系列复杂的过程。在数据库访问测试中, 进行有效单元测试的第一步是判断哪些组件是访问数据库的, 哪些组件用于数据处理。任何访问数据库的组件都可以当作实用对象或事务对象进行单元测试, 测试的方法和前面的描述相类似。在 JUnit 中, 提供了两种运行与测试执行之前和之后的方法, 即 setUp()和 tearDown(), setUp()方法刚好运行在每个 test()方法之前, 而 tearDown()运行于其后。使用 setUp()方法可以在其中启动一个事务处理, 并在 tearDown()中执行 rollback 操作, 这样, 就不会因为测试造成数据库系统的损坏, 从而产生错误的输出。

当然, 由于 J2EE 应用的复杂性, 基于 J2EE 的单元测试还有很多, 如 JavaBeans、EJB 和 RMI 对象的测试, 另外, J2EE 性能测试也是非常重要的一个环节, 针对不同的应用服务器, 如 Tomcat、Jrun、Weblogic 以及对应的应用/部署/程序的规模大小, 各自的测试重点又会有不同, 这里就不再描述, 请参考其他相关的书籍。

## 9.5 其他应用服务器应用的测试

前面主要提到了基于常用的服务器应用测试, 除了前面提到的各种服务器应用外, 还有很多的其他类型服务器的应用, 如邮件服务器、群件服务器、FTP/文件服务器、游戏服务器等, 这些服务器测试也各有自己不同的重点和角度, 如在邮件服务器中, 容量、吞吐

量、邮件的防病毒攻击、保护等是测试的重点；在群件服务器中，功能性、数据完整性、稳定性是测试的重点；而在实时通信系统中，测试重点在数据的及时性以及故障恢复处理上。

总而言之，基于应用服务器应用的测试是个非常复杂的过程，要设计好有效的测试，建立正确的测试过程，编写测试用例，需要对不同的服务器特征、性能以及应用的领域和应用方法都有比较深刻的认识，才能有目标、有效率的组织测试。

## 小结

应用服务器的种类多种多样，基于应用服务器应用的测试也根据不同服务器类型而不同，在本章，主要针对最常见的几类应用服务器测试进行了介绍，包括 Web 服务器，数据库服务器和 J2EE 服务器。

通过本章的学习，用户应该熟悉以下基本的内容：

1. 基本的 Web 测试的技巧。
2. 数据库性能测试设计方法和考虑因素。
3. 基于 JUnit 框架的 J2EE 单元测试。

# 第 10 章 软件本地化测试

随着我国软件开发水平的提高和向国外软件市场的扩张（如软件外包领域的兴起），各种语言版本的软件得到广泛的传播和应用，很多软件在开发的时候都需要充分考虑到国际市场的需求，将不同语言市场的需求对软件的语言版本做相应的定制，这就产生了软件本地化的概念。

本章主要介绍什么是软件本地化、软件本地化的翻译问题和软件本地化测试重点，让读者深入了解软件本地化的过程和测试人员如何测试本地化软件。

## 10.1 什么是软件本地化

软件本地化是将一个软件产品按特定国家或语言市场的需要进行全面定制的过程，它包括翻译、重新设计、功能调整以及功能测试、是否符合当地的习俗、文化背景、语言和方言的验证等。在开始讨论之前，我们先来介绍几个关键术语。

- **L10n**: 由英文的 **Localization** 一词的简写，意即本地化，由于首字母“L”和末尾字母“n”间有 10 个字母，所以简称 L10n。
- **I18n**: 是 **internationalization** 的简写，意为国际化，由于首字母“i”和末尾字母“n”间有 18 个字符，所以简称 i18n。**Internationalization** 指为保证所开发的软件能适应全球市场的本地化工作，不需要对程序做任何系统性或结构性变化的特性，这种特性通过特定的系统设计、程序设计、编码方法来实现。
- **locale**: 场所、本地，简单来说是指语言和区域进行特殊组合的一个标志。
- **Globalization**: 即全球化，它是一个概念化产品的过程，它基于全球市场考虑，以便一个产品只做较小的改动就可以在世界各地出售。全球化可以看做国际化和本地化两者合成的结果。

它们之间的关系可用图 10-1 表示。在这里我们强调国际化是核心工作，只有满足国际化的要求之后才能做本地化，翻译只是本地化工作的一部分，全球化是一个产品市场的概

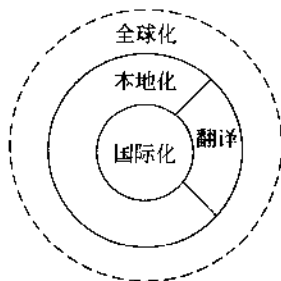


图 10-1 翻译、本地化与国际化、全球化之间的关系

念。下面就来详细说明翻译、本地化与国际化等之间的关系。

### 10.1.1 软件本地化与国际化

人们常说的“国际化”是指任何涉及撰写或改变“原始产品”源代码的行为，从而使该产品在另一个国家能够顺利销售。通常，源代码的国际化必须在本地化之前进行，它为本地化所需要的源代码做准备。国际化的内容包括：

- 支持 Unicode 字符集。
- 分离程序代码和显示内容（文本、图片、对话框、信息框和按钮等），如资源文件（\*.rc）。
- 消除硬代码（hard code，指程序代码中包含一些特定的数据，它们本应该作为变量处理，其数据应该存储在数据库或初始化文件中）。
- 使用 header files 去定义经常被调用的代码段。
- 改善翻译文本尺寸，具有调整的灵活性，如在资源文件中可以直接具有调整用户界面的灵活性来适应翻译文本尺寸。
- 支持组合键。
- 支持各个国家的键盘设置。
- 支持双字节的字符。
- 支持文字排序和大小写转换。
- 支持各个国家的度量衡、时区、货币单位格式等。
- 国际化用户界面设计。

软件本地化是把软件从源语言转换成一种或多种目标语言的过程，同时考虑产品的外观、功能设置等方面在目标国家的相应设计，如：

- 软件用户界面（UI）。
- 联机文档（帮助文档和功能性的 PDF 文档）。
- Web 站点（及其相关的 HTML/SGML 页面和 scripts、applets 等）。
- 组合键设置。
- 度量衡和时区等。

本地化的这些内容都将在以下各节详细讨论。

国际化与本地化是一个辩证的关系。国际化是为了解决软件能在各个不同语言、不同风俗的国家和地区使用的问题，对计算机设计和编程做出的某些规定。简言之，国际化是本地化的基础和前提，为本地化做准备，使本地化过程不需要对代码做改动就能完成。另一方面，本地化是国际化向特定本地语言环境的转换，本地化要适应国际化的规定。

### 10.1.2 软件本地化与翻译

提到本地化，大家首先想到的就是翻译问题，毋庸置疑，翻译在本地化工作中占据着很重要的地位，但是我们绝不能把翻译等同于本地化，它和本地化还有很大的差距。

从图 10-1 可以看出翻译是本地化的子集，它的主要任务是把手语言转换到另一种目标语言。然而，当文字被翻译后，必然要对产品进行许多其他相应的更改。这些更改包括技术层面和文化层面的更改。

### 1. 技术层面的更改

- 调整大小：由于翻译时常引起输入框、文本框、表格不适应的问题。
- 调整默认设置：如文本方向和大小写等，需要根据文化习惯而设置。
- 重新编译：不适合的功能要重新设置，如相应的组合键等。
- 重新创建图标：应该放弃与本地文化相冲突的图标，创建适合本地文化的图标。
- 创建新的图形：带有文字的图形及标签等，也应该把其中的文字本地化。
- 重新编排文档格式。

### 2. 文化层面的更改

- 包装：由于民族风俗的不同，有些国家对颜色和每组的包装数量都有特定的要求。
- 图标：文化背景的区别，应该慎用动物图案。
- 宣传：应该选择适合本地文化的宣传方式。
- 样品：提供的免费样品要能够引起目标客户的足够兴趣。
- 政治敏感的术语、地方规章和宗教信仰。

## 10.1.3 软件本地化基本步骤

要做好软件本地化的测试工作，有必要了解软件本地化的步骤。软件本地化的基本工作是假定建立在软件国际化的基础上，或者说，软件本地化的第一项工作就是规范甚至是迫使原始语言的软件开发遵守软件国际化的标准。在此基础上，依次做好版本管理、建立专业术语表、翻译、调整 UI 等工作。

在软件全部翻译完毕，对技术部分做了必要的调整之后，还有一个软件测试的问题，不论原来的软件产品有多成熟，本地化之后的产品在经过了很多人的重新创造后，除了产品本来存在的问题之外，还有可能产生一些意想不到的新问题，所以进行本地化测试也是本地化非常重要的一个环节。

以下是本地化的基本步骤，在具体操作时可能会有不同，但这些步骤基本是不可省略的：

- (1) 建立一个配置管理体系，跟踪目标语言各个版本的源代码。
- (2) 创造和维护术语表。
- (3) 从源语言代码中分离资源文件、或提取需要本地化的文本。
- (4) 把分离或提取的文本、图片等翻译成目标语言。
- (5) 把翻译好的文本、图片重新插入目标语言的源代码版本中。
- (6) 如果需要，编译目标语言的源代码。
- (7) 测试翻译后的软件，调整 UI 以适应翻译后的文本。

(8) 测试本地化后的软件，确保格式和内容都正确。

### 10.1.4 软件本地化测试

软件本地化测试检查为适应某一特定文化或地区本地化的产品质量。这个测试是基于国际化测试的结果而进行的，国际化测试验证对特定文化或地区的功能性支持。

本地化测试应该着重于：

- 受本地化影响的部分，如用户界面和内容。
- 特殊的文化和地理位置、特殊的语言环境、特定的地区。
- 翻译的正确性。

此外，本地化测试还应该包括：

- 基本的国际化测试。如主要功能性测试，有些传递参数、数据库的默认值会对系统的函数、功能产生一些影响，特别是单字节版本向多字节版本的转换。
- 在本地化环境中的安装和升级测试。由于语言版本操作系统等环境不一样，安装、升级常常受影响。
- 根据产品的目标区域而进行的应用程序和硬件兼容性测试。其应用程序的接口、标准可能不同，硬件流行种类更会有差异。

具体测试的时候，可以选择任何语言版本的操作系统作为测试平台，但是在浏览器上必须安装对目标语言的支持。除此之外，要进行用户界面和语言文化方面的测试，其内容应覆盖以下几个方面：

- 应用程序源文件的有效性。
- 验证语言的准确性和源代码的属性。
- 排版错误。
- 检查印刷文档和联机帮助、界面信息的一致性以及命令键的顺序等。
- 用户界面的可用性。
- 文化适用性的估计。
- 政治敏感内容的检查。

当发布一个本地化产品时，应该确保本地化文档（用户手册、在线帮助、文本帮助等）都包含在其中。同时应该检查：

- 翻译的质量。
- 翻译的完整性。
- 在所有的文档和应用程序界面中术语使用的一致性。

所以概括起来，本地化测试的内容包括以下六个方面：

- 功能性测试，所有基本功能、安装、升级等测试。
- 翻译测试，包括语言完整性、术语准确性等的检查。
- 可用性测试，包括用户界面、度量衡和时区等。
- 兼容性调试，包括硬件兼容性、版本兼容性等测试。
- 文化、宗教、喜好等适用性测试。
- 手册验证，包括联机文件、在线帮助、PDF 文件等测试。

由此可见, 整个软件本地化的过程, 其实是一个再创造的过程。文字翻译仅仅只做了本地化工作的一部分, 要真正完成软件本地化确实有很多工作要做。

## 10.2 软件本地化的翻译问题

当一个软件产品需要在全球范围应用时, 就得考虑在不同的地域和语言环境下的使用情况, 最简单的要求就是用户界面上的内容能用本地化语言来显示, 这就是我们将要说到的翻译问题。当然一个优秀的全球化软件产品关于国际化和本地化的要求远远不止于此。我们所说的本地化不仅是界面的本地化, 还包括内核的本地化。

如前所述, 本地化不仅仅是简单的文字翻译转换, 还应该根据目标语言国家的市场特点、文化习惯、法律等情况进行本地特性开发、界面布局调整等工作。因此翻译也不是单纯的翻译, 还必须立足于文化和市场的角度来考虑用户, 兼顾目标语言的文化心理。为方便起见, 把文化部分的本地化也放在本节讨论。

### 1. 翻译的内容

这是软件本地化要做的第一项任务。一般来说, 需要翻译的内容大致分为三个部分: 用户界面、联机文档和用户手册等。首先, 我们需要从源代码中把需要翻译的资源提取出来, 不论使用的是什么编程语言或平台 (Windows、Mac 或 UNIX), 都可以使用 Visual C++ 或其他有效工具把资源从源代码中分离开来。

翻译完毕, 再把翻译好的文字替换到相应的位置, 可以说其他工作都是在完成这一步的基础上才展开的。这个阶段的要求是翻译准确, 能够照顾到目标语言的文化和习惯。

我们已经知道了翻译是本地化的一个重要的步骤, 那么本地化不仅仅是翻译、不同的文化使用不同的语法和句子结构, 所以直接的词对词的翻译远远不够。相反, 在保持原有意思和风格的基础上, 还必须把源语言格式替换为目标语言的格式。

联机文档常用的格式有 PDF、HTML 和 HTML Help 文件等, 本地化翻译人员也应该把翻译后的文档转换成相应的格式。测试人员也要注意其转换后的格式是否能够正常显示、各部分内容和相关的链接是否都正常等。

除此之外, 软件中的按钮、图标和插图等上面的文字也需要翻译, 本地化测试人员应该指出翻译人员没有翻译的部分, 协助其尽快完成。

关于测试翻译内容的几点建议:

- 翻译时, 应该尽量使用简单的句子结构和语法, 选择意义明确的词。
- 检查翻译的内容是不是断章取义、是否会导致词不达意。
- 如果在源文件中使用了缩写词, 检查缩写词在第一次出现的时候是否正确地标出了它的全称, 以使用户能够明白它的意思。这样, 在其后的文本中即使一直用缩写词来表示它, 也没有关系。
- 检查在不同的国家标点符号、货币单位等是否显示正确。

在国际化基础上, 本地化过程就相对简单。如果国际化没有做好, 翻译和本地化的过

程就会变为一个相当冗长的过程,其中包括将屏幕、对话框等重新设定,而且需要重建在线文件,图像和插图也可能需要更改。最后, 计算机程序还可能需要做出某些修改去适应那些使用双字节字符的语言。

## 2. 目标语言的文化心理

翻译的时候要照顾到目标语言的文化心理,尤其是翻译其宣传品的时候,更要注意把它转换为与目标市场相适应的宣传。包装的规格和包装的颜色也应该留意,比如日本人比较忌讳数字 4,就连 4 个一组包装的产品都不容易卖出去;美国人不太喜欢鲜艳的红色,那么宣传资料和包装纸就应该尽量避免大红的颜色。

翻译不能单纯地追求字与字的对译,这是没有必要的,也是不科学的。对于一些涉及到文化方面的内容,最好能用本民族中相应的内容来替换,如中国人以红色为喜庆的颜色,中国人的结婚礼服都是红色的,而英美等国家则是白色的礼服,这里的红色和白色是一组对应物。本地化时,要把内容做相应的替换。

## 3. 特殊符号

把一种语言翻译成另一种语言,同时还要注意目标语言的特殊符号,比如标点符号、货币符号以及该目标语言所特有的其他符号。英语中的标点符号和亚洲语言的标点符号不太相同,英文的句号是一个圆点(单字节符号),而汉语和日语的句号都是一个小圆圈(双字节符号)。汉语中的标点符号是比较完备的,英文中通常用斜体表示书名,汉字则用“《》”表示书名。

几乎各国都有表示自己货币的货币符号,如美元\$、人民币¥。在翻译的过程中,这些符号是绝对不能出差错的。如果一个金融软件把本该用¥表示的地方用了\$,后果将是不堪设想的。这些也都是本地化测试所应该特别注意的细节问题。

## 4. 对本地化翻译人员的要求

软件是一个面对广大用户的产品,所以一个合格的本地化翻译人员必须具有以下素质。

- 流利的源文字表达水平。
- 对目标语言透彻的理解。
- 良好的写作能力,包括对目标语言语法的掌握能力及不同写作格式和风格的了解。
- 熟悉相关领域的术语。
- 对本地化的流程有相当的了解。
- 会使用相关的工具,比如最新的计算机和多种软件应用程序。
- 对语言和文化差异的敏感性。

此外,翻译者必须有渊博的知识,包括应有软件所涉及的、特定领域的知识。更需要语言之外的技能和技术能力,因为软件本地化的翻译已不再是单纯意义上的翻译。对本地化翻译人员的技术要求有以下一些方面:

- 计算机、数据库或网络相关的技术背景和基础。
- 软件本地化通常还包括图片的处理,所以应该具备图像图形方面的技能和知识,



能够应用图形图像软件来处理所需的图片。

- 有较好的技术自我学习能力，随时去掌握新领域的技术知识和能力。

## 10.3 软件本地化测试的技术问题

完成了语言的转换，对于整个本地化过程来说，才只是完成了第一阶段。要使该软件真正投入使用，还有很多技术方面的问题有待解决，主要有：

- 字符集问题。
- 数据格式。
- 页面显示和布局。
- 配置和兼容性问题。

### 10.3.1 字符集问题

首先要讨论的问题就是不同字符集的使用。西方语言，如英语、法语和德语，使用不到 256 个字符，所以它们可以用单字节编码表示。可是亚洲语言，比如中文和日文，却有几万个字符，所以需要双字节编码。因此在做本地化测试的时候，应该检查开发人员是否使用了正确的字符编码。

不同的操作系统采用不同的方法输入和输出字节，这些不同的方法使字符的编码规范化从而代表不同的语言，这样我们就很容易看出在特定的编码中哪些字符是被支持的，哪些字符是不被支持的。比如，英文字符几乎被所有的编码作为子集而提供支持，但是某些重音字符（如 é, è）和所有其他的扩展符号却不一定被支持。

字符集是操作系统中所使用的字符映射表，例如，某些 UNIX 系统使用只包含 128 个字符的 7 位 ASCII 字符集（包括 Tab、空格、标点、符号、大小写字母、数字和回车键等）。

然而对于很多语言来说，7 位 ASCII 字符集远远不够，因为它不包含特殊字符（比如 é, â 或 â）。所以一个新的标准 8 位 ASCII 诞生了，它包含 256 个字符。微软的 Windows 使用的就是 8 位 ASCII 字符集，对于 UNIX 计算机，还有一个 ISO 标准（ISO8859-X），微软和 ISO 标准极为相似。但是即使拥有 256 个字符，8 位 ASCII 还是无法满足所有语言的需求。汉语、日语和韩语这些语言的字符都很多，仍无法适用扩展后的 ASCII 字符集，对于这些语言，可以使用 16 位字符集（双字节、多字节或变数字节），这就是后来产生的统一的字符编码标准 Unicode，采用双字节对字符进行编码，几乎包含了所有语言的每个字符。目前，很多操作系统都支持 Unicode，如 Windows NT 4.0、Windows 2000、Windows XP 和 UNIX 系列。

### 10.3.2 数据格式

数字、货币和日期的表达方法在不同的国家格式也不尽相同，所以在把软件本地化时，

也应该特别注意这些方面的问题，考虑到本地化格式的要求，否则就有可能出现错误。幸运的是，今天可以使用标准 APIs（比如微软提供的）来处理这类转换的问题。如果是由自己设计的显示方式或模式，就必须设计好其变量含义和处理方式、数据存储方式等去适应这种显示的要求。

在程序设计、编程时，可以通过一些特殊的函数来处理不同语言的数据格式。例如，使用自定义函数 `LocLongdate()`、`LocShortdate()`、`LocTime()`、`LocNumberFormat()` 等替换原来的 `date()` 函数，来处理日期的完整显示、简写、数字等不同的显示格式。后面将会列举一些具体的例子做详细介绍。现在我们先来看看不同地方表达数字、货币和日期等的不同格式，以供本地化测试人员参考。

### 1. 数字

很多欧洲语言使用逗号而不是小数点来表示千位，有的则使用句号或空格代替逗号。所以，本地化的软件也必须注意这个问题，如若不然，有可能一个顾客存入 5000 欧元，而却只能取出 5 美元。比如，同一个数字 7582 在美国、意大利和瑞士有三种不同的表达方式。

美国：7,582

意大利：7.582

瑞士：7582

### 2. 货币

除了数字转换外，几乎每一个国家都有标志自己货币的特殊符号，这些符号出现在金额的前后也各不相同。比如：

美国 Dollar \$ 或 US\$

英国 Pound £

日本 Yen ¥

### 3. 时间

各国时间的习惯表达方式也总是不一样的，美国习惯上使用 12 小时来表达时间，而欧洲国家使用 24 小时模式来表达时间。如，同样是晚上 10:45，各国的表示方式分别如下。

美国：10:45pm

德国：22:45

加拿大法裔：22 h 45

### 4. 日期格式

同样，不同国家的日期显示格式也是不一致的。美国的标准是 MM/DD/YY 来显示月、日、年，也有很多不同的分割符号（如 “/” 和 “-”）；欧洲（除少数例外）的标准是日、月、年（DD/MM/YY）；中国的标准则是年、月、日。下面以 2003 年 2 月 14 日为例来说明。

美国：2/14/2003

英国：14.2.2003

中国: 2003/2/14

即使是一个星期的起始天各国也不相同, 如美国一个星期的第一天是星期天, 而法国日历的第一天都以星期一开头。

现在来看一个具体的例子, 一个英文日期, 如 7/22, 2003 或 7-22-2003, 本地化为中文版本后, 日期显示变为“7月.12, 2003”, 显然不正确, 其正确的中文显示应该是“2003年7月12日”。

现在让我们来了解一下正确的编码。从编码中可以看到在本地化的时候, 有时必须应用自定义函数 `LocLongDate()` 来解决日期显示的问题。这就要求本地化测试人员不只是发现问题, 还是站在更高的层次来分析问题, 并提出解决问题的建议。

根据语言版本取完整日期格式的处理函数 (以下程序设计语言为 PHP 语言)。

```
function LocLongDate($UNIXTime,$RegionID,$DisplayWeek="Yes")
{
    global $glbRegion;
    ... ..

    InternationInit();
    if(!IsExistRegionID($RegionID))
        $RegionID = $glbDefaultRegionID;           //取得本地区域代码
    if("". $glbRegion[$RegionID][LONGDATEFORMAT]=="") //如果是长日期型
        $glbRegion[$RegionID][LONGDATEFORMAT]="WWW,MMM d,yyyy";
    $strFormat=FormatLocToFormatPhp($glbRegion[$RegionID][LONGDATEFORMAT]);
    if($DisplayWeek=="NoWeek")                       //处理日期格式
    {
        $strFormat = eregi_replace("l","", $strFormat);
        $strFormat = eregi_replace("^","", $strFormat);
    }
    $LongDateString=date($strFormat,$UNIXTime);
    if(strpos(strtolower($glbRegion[$RegionID][LONGDATEFORMAT]),"www"))
    $LongDateString=str_replace(date("l",$UNIXTime),$ARR_FULLWEEKDAY
        [date("w",$UNIXTime)], $LongDateString); //获得星期显示字符串
    if(strpos(strtolower($glbRegion[$RegionID][LONGDATEFORMAT]),"mmmm"))
    $LongDateString=str_replace(date("F",$UNIXTime),ARR_FULLMONTH[date("n",
        $UNIXTime)-1], $LongDateString); //获得日期显示字符串
    return $LongDateString;
}
```

## 5. 度量衡的单位

美国以外的很多国家使用公制度量系统, 因此, 国际化的软件必须能够解决公制度量单位的问题。度量衡的单位在工程和科学软件中尤为敏感, 如果在转换的过程中出现错误, 后果将不堪设想, 所以在转换英式度量单位和公制度量单位时要倍加小心。这是本地化测



是姓前名后，而且中间无须空格。由于这个区别，在做本地化测试的时候，一定要确保受影响的部分都做了相应的改动，否则会导致显示和查找的时候产生错误。在测试的时候如果发现此类错误，可以建议编码人员根据不同国家和地区的语言习惯考虑姓、名以及全名之间的关系，自定义一些函数来处理此类问题。

这里我们定义了函数 `GetFullNameforMultipleLanguage` 来帮助我们识别姓名的类型，从而选取正确的显示格式。

```
function GetFullNameforMultipleLanguage($FirstName,$LastName="")
    //根据语言版本取得相应的姓名格式
{
    //如果全名中包含英文大写字母 A-Z 或小写字母 a-z, 则保留名和姓之间的空格
    $strFullName=trim($FirstName." ".$LastName);
    if(ereg("[a-z]",$strFullName[0])) //如果姓名是英文字符, 则立刻返回其值
        return $strFullName;
    //如果当前语言是繁体中文, 则删除其中的空格
    $LanguageID=GetCookie("CK_LanguageID_".GetSiteConfig("SiteID"));
    if(intval($LanguageID)==0)
        $LanguageID=GetLanguageIDFromUrl(BrandName());
    if($LanguageID == 4||$LanguageID == 5) // 针对繁体中文和日文
        return ereg_replace(" *","",$strFullName);
    if(ereg("[a-z]",$strFullName))
        return trim($strFullName);
    else
        return ereg_replace(" *","",$strFullName);
}
```

## 8. 复数问题

生成复数的规则因语言的不同而有差异。即使在英语中，复数的规则也并不是始终如一，如 `bed` 的复数是 `beds`，而 `leaf` 的复数却不是 `leafs`，以下例子说明了复数的问题。如：

```
"%d program%s searched"和"%d file%s searched"
```

如果 `%d` 大于 1，`%s` 将把 `s` 插入到该单词中去，从而组成其复数形式，该信息显示格式如下：

```
"1 program searched"和"1 file searched"
```

或者

```
"3 programs searched"和"3 files searched."
```

在英语中，这样编码是没有问题的，但是对于德语和多数其他欧洲语言，它们的复数规则却不是这样的，如：

```

program = programma
programs = programma's
file = bestand
files = bestanden

```

在做本地化测试的时候，一定要注意这些地方是否被充分地考虑并做了适当的修改。

### 10.3.3 页面显示和布局

在有些本地化软件中，有时会发现乱码的问题，这是由于没有设置相应的本地化字符集或字符编码方式不支持本地化语言所致。不同的浏览器或邮件接收软件的编码解码方式不同，解决这类问题的方法是：

- 开发本地化时应用自定义函数 `GetCurCharset()`。如：

```

function GetCurCharSet() // bind charset to language
{
    $CharSet="iso-8859-1";                //标准字符集
    $LanguageID = GetCurLanguageID();

    if($LanguageID==3) $CharSet="gb2312";    //简体中文版
    if($LanguageID==4) $CharSet="big5";      //繁体中文版
    if($LanguageID==5) $CharSet="shift_jis"; //日文版
    if($LanguageID==6) $CharSet="euc-kr";    //韩文版
    return $CharSet;
}

```

我们可以看到在这个函数中调用了另一个自定义函数 `GetCurLanguageID()`，这个函数的值是通过本机的 cookies 取到的，这样可以通过在函数 `GetCurCharset()` 中调用该函数来判断采用的字符集。

```

function GetCurLanguageID()                //取得相应的语言版本
{
    $LanguageID=GetCookie("CK_LanguageID_".GetSiteConfig("SiteID"));
    if(intval($LanguageID)==0)
        $LanguageID=GetLanguageIDFromUrl(BrandName());

    return intval($LanguageID);
}

```

- 针对不同的浏览器采取不同的解码方法。如下例：

```
Function Preview( form ) {  
    var NS4=(document.layers && !dom)? 1:0;  
    var NS6=(navigator.vendor=="Netscape6" || navigator.product  
        ==("Gecko"));  
    var message;  
    var re=^+/g;  
    if(NS4>0)  
        message=escape(form.welmsg.value);  
    else if(NS6>0)  
        message=form.welmsg.value;  
    else  
        message=form.welmsg.innerHTML;  
  
    if(message.indexOf("+")!=-1){  
        var wmessage=message.replace(re,"%2B");  
    }  
    else wmessage=message;  
    form.AT.value="Submit";  
    form.preview.value="true";  
  
    var sTemp;  
    if(NS4>0 || NS6>0)  
        sTemp=form.welmsg.value;  
    else  
        sTemp=form.welmsg.innerHTML;  
    if(getStrLength(sTemp)>128)  
    {  
        alert("欢迎消息不能超过 128 个字符。"); //给出警告提示  
        form.welmsg.focus();  
    }  
    else  
        window.open("/<?=PersonalLobbyPath()??>index.php?username=<?=  
rawurlencode(str_replace("&quot;","\"",$repinfo["UserName"]))??>  
&preview=true&welmsg="+wmessage);  
}  
}
```

由于源代码没有充分考虑到国际化 (I18N) 版本的要求, 很多软件本地化之后在页面的外观上会出现一些不尽如人意的地方。例如, 没有翻译的字段、对齐问题、大小写问题、文字遮挡图像问题、乱码显示问题等。这些有表格设置所产生的问题, 也有未考虑翻译后的文字扩展而产生的设计问题。本地化、国际化测试工程师应该指出这些地方, 让开发人员尽快修改。

### 10.3.4 配置和兼容性问题

测试本地化软件的时候，其配置和兼容性也是必须考虑的问题。配置性包括键盘布局设计，它是语言依赖性最大的硬件配置。软件可能会用到的任何外设都要在平台配置和兼容性测试的等价区间中考虑。兼容性包括与硬件的兼容性、与上一版本的数据兼容及与其他本地化软件的兼容性等。兼容不单单是同一版本向前或向后的兼容，同时还要考虑同其他软件的兼容，也要考虑同其他本地化软件的兼容。

#### 1. 数据库问题

软件本地化同时也涉及到数据库的改动，比如由于文本的 `maxlen` 属性只限制输入字符而非字节长度或非 ASCII 码，特别是多字节字符解析成 NCR 形式 (`&#dxxx;`)，导致输入的字符长度超出数据库字段宽度，这就是由数据库而产生的问题。

在本地化过程中，我们应视情况而定。比如我们可以在输入页面提交之前，检测输入字符的宽度是否超长或显示数据库操作错误，如下所示：

```
.....
<form name=AddVis>
<input name="V_Name" type=text value="">
<input type=submit value="ok" onsubmit="javascript:checkinput()">
.....

function getStrLength(StrTemp) //取字符长度
{
    var strInput=""+StrTemp;
    var i,sum;
    sum=0;
    //alert("string lengh="+strInput.length);
    for(i=0;i<strInput.length;i++)
    {
        if((strInput.charCodeAt(i)>=0)&&(strInput.charCodeAt(i)<=255))
            sum=sum+1;
        else
            sum=sum+2;
    }
    //alert("actual lengh="+sum);
    return sum;
}

function checkinput() //检查输入字符的长度
{
    if( getStrLength(document.AddVis.V_Name.value)>64)
    {
```



```
    alert("The input string must be less than 64. "); //给出警告提示
    document.AddVis.V_Name.focus();
    return false;
}
}
```

## 2. 组合键

在做本地化测试的时候，还有一个不能忽略的问题——组合键问题。许多程序都为不同的命令设置了组合键（键盘快捷方式）。比如，在微软的 Word 中，可以同时按下 Ctrl 和 F 键打开“查找”对话框。组合键 Ctrl+F 就是代替鼠标选择 Word 编辑菜单中查找命令的快捷方式。通常，文字被翻译之后，原来的组合键很可能不再适用，我们需要为翻译过的文本设定新的组合键，比如，当 Close 被翻译成德语 Schließen 之后，原有的组合键 Alt+C 也应该相应地变为 Alt+S。新的组合键应该和本地操作系统环境相匹配，确保所有的组合键都是惟一的。不过中国、日本和韩国的版本，都沿用英文原有的组合键，所以本地化之后不存在这个问题。

此外，还有很多应该注意的技术问题，如对于欧洲语言的本地化，还有大小写字母转换的问题、连字符连接规则、键盘的问题等。对于有些国家的本地化，例如希伯来文和阿拉伯文还要考虑文字方向的问题等，这些都是在实际工作中会遇到的具体问题，这里就不一一详述了。

# 10.4 本地化测试的重点

软件本地化之后，应该把它当做一个全新的产品来对待，所以本地化测试的方法和步骤与其他软件的测试都是相通的。但是它还有自己的特性，本地化测试人员应该特别注意在第 10.2、10.3 节提到的翻译问题和技术问题，这些基本覆盖了本地化测试实践中所遇到的应该注意的问题。此外，我们还想提醒本地化测试人员一些其他方面的注意事项。前面说过，软件本地化是一个再创造的过程，这其中不仅包括翻译人员的劳动、技术人员的再加工，更应该包括本地化测试人员的层层把关。在测试本地化的软件时，不能单纯地看功能是否实现，还要看是否符合目标用户的习惯，是否与当地的文化和风俗相符合。

## 10.4.1 翻译所产生的问题

10.2 节已讨论了翻译的问题，这里是从测试的角度进一步探讨翻译可能带来的问题。很多本地化测试人员都有这样的误解，认为翻译部分都是翻译人员的工作，测试人员的工作只是保证基本功能的实现。其实不然，整个本地化过程是翻译、编程和测试人员共同劳动的结晶，三者是共同协作、共同促进的关系，任何一方都不能只单纯的注意自己的那一部分，而应该多方兼顾。如果一个测试人员只注重基本功能的实现，而对翻译的问题置之不理，就称不上是一个合格的本地化测试工程师。

帮助翻译人员发现翻译中的错误和不妥之处、指出开发人员技术上未能实现的部分，这些都是测试人员在工作中应该做到的。一个合格的本地化测试工程师应该积极地参与到产品的各个阶段中去，从翻译编程到测试，都包含测试工程师的大量辛勤劳动。

把一种语言翻译成另外一种语言，难免会有晦涩或表达不准确的地方，由于大量的翻译工作，翻译人员可能会遗漏或疏忽某些地方，这就需要测试人员及时的提醒和配合。此外，对于不符合本地文化和软件产品的措辞，测试工程师也应该及时地指出来，使其尽量和本地语言对应。

软件展示给用户的首先就是它的用户界面（UI）、包装，所以 UI 的重要性是不言而喻的。如果在 UI 上出现错误或者有歧义会让用户对该软件产生不信任的心理，这些都应该坚决杜绝。再则，由于不同民族、不同信仰之间的差别，各民族在色彩、禁忌和其他习惯上也有很多区别。测试工程师在这个环节中肩负着重任，不单单是找出错误，找出不符合民族习惯的地方，还应该能够提出有建设性的修改方案。

## 10.4.2 功能的实现

任何一件产品，人们最关心的还是它所能提供的服务，所以功能的实现总是很重要。要验证一个软件是否被正确地本地化，要在相对真实的环境下对软件的所有功能进行测试。这个过程可能会需要很多人的参与，比如市场销售人员、国内用户群体等。有条件的话，在本地化软件向市场发布之前，还应该让目标语言的语言专家来最后审稿。

关于本地化软件的功能测试，可以同原软件相对比来进行。此外，还要注意是否能够正确地输入目标语言，输入之后是否能够正确显示等。

### 1. 联机文档的功能测试

就像打印好的文档一样，测试人员应该验证任何一个在线文档的有效性、可用性。本地化软件测试人员应该对它们进行功能测试，以确保它们能够正常工作，并且与目标市场的要求一致。

不论是 PDF 还是 HTML 格式的在线文档都应该在目标语言的操作系统下测试，确保其功能能够实现、字符能够正确显示。一般来说，主要检测这些文件的下述方面：

- 与目标语言操作系统的兼容性。
- 字体和图形能够正确显示。
- 与本地化的 Acrobat Reader 版本和 HTML 浏览器兼容。
- 超级链接的正常跳转。

根据 PDF 和 HTML 文件的高级特征，在对在线文档进行功能测试的时候，还可以加入其他的测试项目。

### 2. 页面内容和图片

很多页面的内容都是文字和图片，正如我们此前所介绍的一样，同样的规则也适用于这里，测试人员应该时刻谨记 HTML 页面上有些文字不是一眼就能看到的，包括：

- 显示在浏览器界面顶部的页面标题。
- 图片的标题，当图片正在下载或者用户鼠标指向该图形时所显示的 ALT 属性。
- 超级链接的标题。

还要确保站点上包含文字的图片也同时进行了本地化。

### 3. Web 链接和高级选项

测试员需要关注页面上的超级链接未被本地化或者链接到其他未被本地化的站点上去。如果可能的话，建议开发人员去修改这些链接，可以把它指向特定的站点（如果该链接所指向的站点有相应的本地化版本），或者用目标语言在这些链接旁给出提示，指出这些站点是外文的。

同时测试人员还要关注很多站点日益增多的动态效果，以及应该去检查 CGI 脚本、Java 代码或脚本和 ActiveX applets 是否受到影响。

## 10.4.3 对本地化测试人员的要求

测试是软件本地化的最后一个环节，能否发布一个高质量的产品，测试工程师的把关是很重要的。本地化测试工程师跟踪产品的各个阶段，对产品实现什么功能、如何实现，都应有很清楚的了解。为了做好本地化的测试工作，本地化测试工程师应该具备下述能力：

- 具有基本软件编码、标准字符集和本地化计算机操作系统方面的相关知识。
- 掌握本地化软件的测试技能和流程。
- 具有计算机技术方面较为全面的知识和技能。
- 具备相应的外语理解能力和汉语表达能力，至少要基本熟悉所测试的语言，如看懂显示的文字、输入必要的文字。
- 对语言、文字和界面具有高度的敏感性。

其实，软件本地化工程师，无论是翻译人员、测试工程师、编辑人员还是开发工程师都应该具备多方面的能力，包括计算机知识、相关技术知识、外语能力和良好的语言文学修养等。

## 小结

国际化是本地化的基础和前提，本地化是国际化向特定本地语言环境的转换，其理想的状态是，源语言版本要按照国际化版本的要求去做，本地化本身不应该再给该软件增加新的功能缺陷。然而，如果该软件没有充分地国际化，在本地化过程中就很有可能会产生新的功能性方面的问题，包括功能调用出错、输入和输出问题等。

翻译仅是软件本地化的一部分工作，软件本地化实际是一项技术工作，要处理字符集问题、数据格式、页面显示和布局、配置和兼容性问题。在测试过程中，应该特别注意这些方面的问题，特别是时区、日期、时间、货币、度量衡、姓名、复数等方面的处理和

显示。为确保本地化后软件产品的质量，本地化的产品应该在配置有目标语言操作系统的计算机上进行测试。本地化的翻译人员、培训人员和技术支持人员也最好参与到本地化的测试中来，以保证该测试的全面性和完整性。

## 思考题

1. 为什么要进行软件本地化？
2. 软件本地化和软件国际化有什么关系？
3. 为什么说软件本地化不等同于是翻译？
4. 软件本地化测试中应该着重于哪些方面？
5. 进行软件本地化测试是否必须通晓目标语言？为什么？
6. 请简要阐述软件本地化测试与其他测试的异同。
7. 作为一名本地化测试工程师，你认为最重要的素质是什么？
8. 本地化测试工程师应如何配合软件开发工程师和翻译人员？
9. 请用图表简述软件本地化的一般流程。
10. 假设需要测试某一软件的日语本地化版本，请问需要做哪些方面的准备？请一一列举。

# 第 11 章 软件测试自动化

软件测试是一项艰苦的工作，需要投入大量的时间和精力，据统计，软件测试会占用整个开发时间的 40%。一些可靠性要求非常高的软件，测试时间甚至占到总开发时间的 60%。但是软件测试具有一定的重复性，我们知道，软件在发布之前要进行几轮测试。在测试后期所进行的回归测试中大部分测试工作是重复的，回归测试就是要验证已经实现的大部分功能，这种情况下，是为了解决软件缺陷、需求变化，代码修改很少，针对代码变化所做的测试相对比较少。而为了覆盖代码改动所造成的影响需要进行大量的测试，虽然这种测试找到软件缺陷的可能性小，效率比较低，但又是必要的。此后，软件不断升级，所要做的测试重复性也很高，所有这些因素驱动着软件自动化的产生和发展。

本章将主要介绍软件测试自动化的基本内涵、如何在测试中引入自动化方法、测试自动化的基本结构和基本方法，以及测试工具的分类和流行工具，最后以基于 Rational 产品的整体解决方案使读者对软件测试自动化有一个完整的、感性的认识。

## 11.1 测试自动化的内涵

软件测试自动化是相对手工测试而存在的，主要是通过所开发的软件测试工具、脚本（script）等来实现，具有良好的可操作性、可重复性和高效率等特点。测试自动化是软件测试中提高测试效率、覆盖率和可靠性的重要测试手段。也可以说，测试自动化是软件测试不可分割的一部分。

### 11.1.1 软件测试自动化的意义

要理解为什么要软件测试自动化，可以从两个方面考虑：一是手工测试的局限性；二是软件测试自动化所带来的好处。

#### 1. 手工测试的局限性

测试人员进行手工测试时，具有创造性，可以举一反三，从一个测试用例想到另外一些测试用例，特别是可以考虑到测试用例不能覆盖的一些特殊的或边界的情况。同时，对于那些复杂的逻辑判断、界面是否友好，手工测试具有明显的优势。但是手工测试在某些测试方面，可能还存在着一定的局限性，包括：

- 通过手工测试无法做到覆盖所有代码路径。
- 简单的功能性测试用例在每一轮测试中都不能少，而且具有一定的机械性、重复性。其工作量往往较大，却无法体现手工测试的优越性。

- 许多与时序、死锁、资源冲突、多线程等有关的错误通过手工测试很难捕捉到。
- 在系统负载、性能测试时，需要模拟大量数据或大量并发用户等各种应用场合时，也很难通过手工测试来进行。
- 在进行系统可靠性测试时，需要模拟系统运行 10 年、几十年，以验证系统能否稳定运行，这也是手工测试无法模拟的。
- 如果有大量（几千）的测试用例，需要在短时间内（1 天）完成，手工测试几乎不可能做到。
- 测试可以发现错误，并不能表明程序的正确性。因为不论黑盒、白盒方式都不能实现穷举测试。对一些关键程序，如导弹发射软件，则需要考虑利用数学归纳法或谓词演算等进行证明。

## 2. 软件测试自动化所带来的好处

由于手工测试的局限性，软件测试借助测试工具极为必要，并向软件测试全面自动化方向发展，将测试工具和软件测试自动化结合起来，可以解决上述局限性，并且会带来一些好处：

- 缩短软件开发测试周期。软件测试具有速度快、效率高的特点，对上千个测试用例，软件测试自动化工具可以在很短时间内完成，还可以在很短的时间内运行同样测试用例 10 遍、100 遍等。
- 测试效率高，充分利用硬件资源。可以在运行某个测试工具的同时运行另一个测试工具，也可以在一面运行某个测试工具一面思考新的测试方法或设计新的测试用例，能够把大量测试个案分配到各台机器上同时运行，从而节省大量的时间。也可以把大量的系统测试及回归测试安排到夜间及周末运行，这样能提高效率，如在下班前将所有要运行的测试脚本（用脚本语言，写成的一些短小程序）准备好，并启动测试工具，第二天一上班就能拿到测试结果。
- 节省人力资源，降低测试成本。在回归测试时，如果是手工方式，就需要大量的人力去验证大量稳定的旧功能，而通过测试脚本和测试工具，只要一个人就可以了，可以节省大量的人力资源。同样的测试用例，需要在很多不同的测试环境（如不同的浏览器、不同的操作系统、不同的连接条件等）下运行，这也正是测试工具大展身手的时候。
- 增强测试的稳定性和可靠性。通过测试工具运行测试脚本，能保证 100% 进行。但是，有时个别测试人员并没有执行那些测试用例，但他可能告诉你，他已经运行了。
- 提高软件测试的准确度和精确度。软件测试自动化的结果都是数量化，能够同所预期结果或规格说明书规定的标准进行量化对比。
- 软件测试工具使测试工作相对比较容易，但能产生更高质量的测试结果。
- 手工不能做的事情，软件测试自动化能做，如负载、性能测试。

软件测试实行自动化进程，绝不是因为厌烦了重复的测试工作，而是因为测试工作的需要，更准确地说是回归测试和系统测试的需要。

## 11.1.2 自动化测试的引入和应用

在了解软件测试自动化的重要意义之后,就要开始启动软件测试自动化进程。在进行自动化测试之前,首先要建立一个对软件测试自动化的认识观。软件测试工具能提高测试效率、覆盖率和可靠性等,软件测试自动化虽然具有很多优点,但它只是测试工作的一部分,是对手工测试的一种补充。软件测试自动化绝不能代替手工测试,它们各有各的特点,其测试对象和测试范围都不一样:

- 在系统功能逻辑测试、验收测试、适用性测试、涉及物理交互性测试时,多采用黑盒测试的手工测试方法。
- 单元测试、集成测试、系统负载或性能测试、稳定性测试、可靠性测试等比较适合采用自动化测试。
- 那种不稳定软件的测试、开发周期很短的软件、一次性的软件等不适合自动化测试。
- 工具本身并没有想象力和灵活性,根据报道,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。
- 自动化测试工具在进行功能测试时,其准确的含义是回归测试工具,这时工具不能发现更多的新问题,但可以保证对已经测试过部分的准确性和客观性。

多数情况下,手工测试和自动化测试应该相结合,以最有效的方法来完成测试任务。

### 1. 找准测试自动化的切入点

不管是自己开发测试工具,还是购买第三方现成的工具产品,当开始启动测试自动化时,不要希望一下子就能做很多事情。必须从最基本的测试工作切入,如验证新构建的软件包(Build)是否有严重的或致命的问题,即验证构建的软件包所有基本功能是否正常工作,或者可以从某一个模块开始,如果这个模块做成功了,再向其他模块推进。

### 2. 把测试开发纳入整个软件开发体系

测试用例设计完成之后,就可以进行手工测试,但要用测试工具,还必须将测试用例转化成测试脚本或编写特殊的测试程序,测试脚本也是程序,所以应该要遵守已有的、规范的编程标准和规则。用编程语言或脚本语言写出短小的程序来产生大量的测试输入(包括输入数据与操作指令),或同时也按一定的逻辑规律产生标准输出。输入与输出的文件名,同开发中其他环节一样,进行统一规划,按规定进行配对,以便进行自动化测试的结果对比分析。自动测试应该是整个开发过程中的一个有机组成部分。自动测试要依靠配置管理来提供良好的运行环境,同时它必须要与开发中的软件构建紧密配合。

只要是程序,就可能存在缺陷,所以测试脚本或测试程序也要进行测试,在实际运行测试之前,要保证测试工具或测试脚本的正确性。当然,并不是说要一层的测试下去,而进入程序测试递归的死胡同。相对来说,测试脚本或测试工具简单些,其测试也容易些。一旦测试中发现问题,要么是被测试的对象有问题,要么是测试脚本或测试工具有问题,

总之，问题容易发现。

为了使测试自动化的脚本能多次重复进行，测试用例和测试脚本要写入数据库，进行动态管理。

### 3. 测试自动化依赖测试流程和测试用例

不管是手工测试和自动化测试，关键是测试流程的建立和测试用例的设计，只有在良好的测试用例基础上，编写测试脚本、执行测试或运行测试脚本，才能保证测试的执行效果。为了适合测试自动化的测试脚本的编程，可以使测试用例转化为用例矩阵化（Case Matrix），使测试脚本容易实现结构化。

### 4. 软件测试自动化的投入较大

对于软件测试自动化要有一个正确的理解，才能做到事半功倍。由于软件测试自动化在前期的投入要比手工测试的投入大得多，除了在购买软件测试工具或成套工具系统所投入的资金（一般这类工具软件还比较贵）和大量的人员培训之外，还要花很多时间去写测试脚本、维护脚本等。

### 5. 进行资源的合理调度

在开发中的产品达到一定程度的时候，就应该开始进行每日构造新版本并进行自动化的验证测试。这种做法能使软件的开发状态得到频繁的更新，及早发现设计和集成的缺陷。为了充分利用时间与设备资源，下班之后进行自动的软件构建，紧接着进行自动测试（这里多数指的是系统测试或回归测试）是一个非常行之有效的方法。如果安排得好，到第二天上班时，测试结果就已经在各人的电子邮箱里面了。

## 11.1.3 测试自动化的基本结构

测试自动化引入是第一步，做什么事首先要做什么，做了之后就有感觉，对测试自动化有了更深理解，就可以好好设计一下测试自动化的框架，在此框架下就可以全面启动测试自动化的工作。测试自动化的基本结构由六部分组成：

- 构建、存放程序软件包和测试软件包的文件服务器，在这个服务器上进行软件包的构建，并使测试工具可以存取这些软件包。
- 存储测试用例和测试结果的数据库服务器提高过程管理的质量，同时生成统计所需要的数据。
- 执行测试的运行环境——测试试验室，或一组测试用的服务器或 PC 计算机。单元测试或集成测试可能多用单机运行。但对于系统测试或回归测试，就极有可能需要多台计算机在网络上同时运行。
- 控制服务器，负责测试的执行、调度，从服务器读取测试用例，向测试环境中代理（Agent）发布命令。
- Web 服务器负责显示测试结果、生成统计报表、结果曲线。作为测试指令的转接



点,接受测试人员的指令,向控制服务器传送。同时,根据测试结果,自动发出电子邮件给测试或开发的相关人员。**Web 服务器**,让开发团体的任何人员都可以方便地查询测试结果,也方便测试人员在自己办公室运行测试。

- 客户端程序,测试人员在自己计算机上安装的程序,许多时候,要写一些特殊的软件来执行测试结果与标准输出的对比工作或分析工作,因为可能有部分的输出内容是不能直接对比的,此时就要用程序进行处理。

如图 11-1 所示,理想的测试工具可以在任何一个路径位置上运行,可以到任何路径位置取得测试用例,同时也可以把测试的结果输出到任何路径位置上去。这样的设计,可以使不同的测试运行能够使用同一组测试用例而不至于互相干扰,也可以灵活使用硬盘的空间,并且使备份保存工作易于控制。

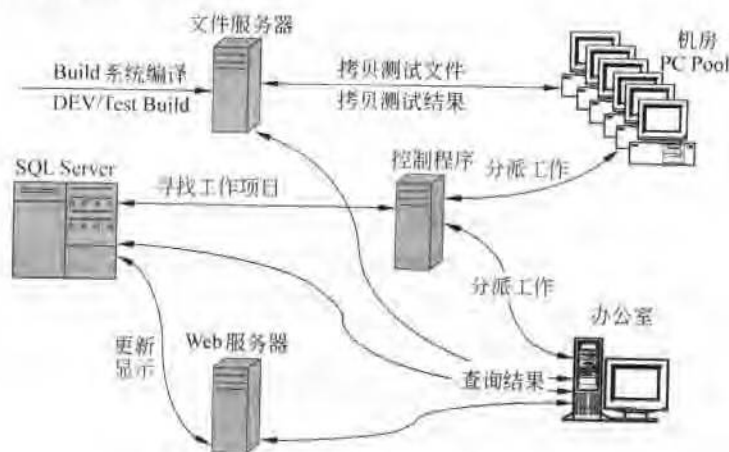


图 11-1 测试自动化的基本结构

同时,软件自动测试工具必须能够有办法方便地选择测试用例库中的全部或部分内容来运行,也必须能够自由地选择被测试的产品或中间产品作为测试对象。

### 11.1.4 测试自动化的原理和方法

基于上面测试自动化框架,可以购买第三方软件测试工具,或自己开发软件测试工具。为了选择软件测试工具或指导自己开发软件测试工具,有必要了解一下软件测试自动化的原理。

软件测试自动化实现的基础是可以通过设计的特殊程序模拟测试人员对计算机的操作过程、操作行为,或者类似于编译系统那样对计算机程序进行检查。软件测试自动化实现的原理和方法主要有:直接对代码进行静态和动态分析、测试过程的捕获和回放、测试脚本技术、虚拟用户技术和测试管理技术。

#### 1. 代码分析

代码分析类似于高级编译系统,一般针对不同的高级语言去构造分析工具,在工具中

定义类、对象、函数、变量等定义规则、语法规则等；在分析时对代码进行语法扫描，找出不符合编码规范的地方；根据某种质量模型评价代码的质量，生成系统的调用关系图等。为了更好地进行代码分析，可以在代码中插入一些“断点”，即向代码生成的可执行文件中插入一些监测代码，随时了解这些关键点/关键时刻的某个变量的值、内存/堆栈状态等。

## 2. 捕获和回放

代码分析是一种白盒测试的自动化方法，捕获和回放则是一种黑盒测试的自动化方法。首先将用户每一步操作都记录下来。这种记录的方式有两种：程序用户界面的像素坐标或程序显示对象（窗口、按钮、滚动条等）的位置，以及相对应的操作、状态变化、或属性变化。所有的记录转换为一种脚本语言所描述的过程，以模拟用户的操作。

回放时，将脚本语言所描述的过程转换为屏幕上的操作，然后将被测系统的输出记录下来同预先给定的标准结果比较。这可以大大减轻黑盒测试的工作量，在迭代开发的过程中，能够很好地进行回归测试。

## 3. 脚本技术

脚本是一组测试工具执行的指令集合，也是计算机程序的一种形式。脚本可以通过录制测试的操作产生，然后再做修改，这样可以减少脚本编程的工作量。当然，也可以直接用脚本语言编写脚本。测试工具脚本中可以包含数据和指令，并包括下面一些信息：

- 同步（何时进行下一个输入）。
- 比较信息（比较什么、如何比较以及和谁比较）。
- 捕获何种屏幕数据及存储在何处。
- 从另一个数据源读取数据时从何处读取。
- 控制信息等。

脚本技术围绕着脚本的结构设计，实现测试用例，在建立脚本的代价和维护脚本的代价中得到平衡，并从中获得最大益处。

脚本技术可以分为以下几类。

- 线性脚本：是录制手工执行的测试用例得到的脚本。这种脚本包含所有的击键、移动、输入数据等，所有录制的测试用例都可以得到完整的回放。对于线性脚本，也可以加入一些简单的指令，如时间等待、比较指令等。线性脚本适合于简单的测试（如 Web 页面测试）、一次性测试，多数用于脚本的初始化（录制的脚本用于以后修改），或者用于演示等。
- 结构化脚本：类似于结构化程序设计，具有各种逻辑结构，包括选择性结构、分支结构、循环迭代结构，而且具有函数调用功能。结构化脚本具有很好的可重用性、灵活性，所以结构化脚本易于维护。
- 共享脚本：是指某个脚本可以被多个测试用例使用，即脚本语言允许一个脚本调用另一个脚本。可以将线性脚本转换为共享脚本。
- 数据驱动脚本：将测试输入存储在独立的（数据）文件中，而不是存储在脚本中。这样的脚本可以针对不同的数据输入实现多个测试用例。

- 关键字驱动脚本：是数据驱动脚本的逻辑扩展。

实际上，在建立脚本时，都是将几种技术结合起来应用，如数据驱动脚本技术和关键字驱动脚本技术经常是一起使用的。

脚本技术不仅可以用在功能测试上模拟用户的操作，然后进行比较，而且可以用在性能、负载测试上，虚拟用户同时进行相同或不同的操作，给系统或服务器足够的数据、操作，以检验系统或服务器的响应速度、数据吞吐能力等。

#### 4. 自动比较

自动测试时，预期输出是事先定义的，或插入脚本中，然后在测试过程中运行脚本，将捕获的结果和预先准备的输出进行比较，从而确定测试用例是否通过。所以，自动比较在软件测试自动化中就非常重要。自动比较可以对比分析屏幕或屏幕区域图像、比较窗口或窗口上控件的数据或属性、比较网页、比较文件等。

- 静态比较和动态比较：动态比较是在测试过程中进行比较。静态比较在测试过程中并不作比较，而是将结果存入数据库或文件中，然后通过另外一个单独的工具来进行结果比较。
- 简单比较和复杂比较：简单比较要求实际结果和期望结果完全相同，而复杂比较是一种智能比较，允许实际结果和期望结果有一定的差异。智能比较需要使用屏蔽的搜索技术，来排除输出中预期会出现差异部分，忽略特定的差异。
- 敏感性测试比较和健壮性测试比较：敏感性测试比较要求比较尽可能多的信息，如在执行测试用例的每一步就比较整个屏幕的信息，屏幕输出中或多或少的变化就可能导致不匹配，而标志此测试用例失败。健壮性测试只比较最少量、最需要的信息，如屏幕的最后输出。
- 比较过滤器：就是在对实际输出结果和期望输出结果进行预先处理，执行过滤任务之后，再进行比较。这样可以使比较标准化，测试结果可靠。

#### 5. 测试管理

测试管理是指对测试输入、执行过程和测试结果进行管理。除了对和手工测试共性的东西，如测试计划、测试用例、测试套件、缺陷、产品功能和特性、需求变化等实施管理之外，还要对自动化测试中特有的东西进行跟踪、控制和管理，主要有测试数据文件、测试脚本代码、预期输出结果、测试日志、测试自动比较结果等。由于是进行自动化管理，文档性管理已不能满足其需要，应该使用数据库技术、XML 技术或严格的数据格式文件进行管理。

### 11.1.5 测试自动化普遍存在的问题

对测试工具能够发挥的作用大家都已经了解了，但是很多引入测试软件的公司并没有能够让测试软件发挥应有的作用，其主要原因有以下几个方面。

### 1. 不正确的观念或不现实的期望

没有建立一个正确的软件测试自动化观念操之过急,认为测试自动化可以代替手工测试,认为测试自动化可以发现大量新缺陷,或不够重视而不愿初期花费产生比较大的开支等。多数情况下,对软件测试自动化存在过于乐观的态度、过高的期望,人们期望通过这种测试自动化的方案能解决目前遇到的所有问题。同时,开发测试工具的软件厂商自然会强调有利的或成功的一面,对取得这种成功所要做出的持久不懈努力却只字不提。结果最初的期望实现不了。

### 2. 缺乏具有良好素质和有经验的测试人才

有些软件公司舍得花几十万元去买测试工具软件,但缺乏具有良好素质和有经验的测试人才。软件测试自动化并不仅是简简单单地使用测试工具,还需要有良好的测试流程、全面的测试用例配合脚本的编写,这就要求测试人员不仅要熟悉产品的特性和应用领域、熟悉测试流程,而且要很好地掌握测试和编程技术。

### 3. 测试工具本身的问题影响测试的质量

一般自动测试的脚本不会再做大规模的测试,所以自动测试的脚本质量往往依赖于测试人的经验和工作态度。如果自动测试工具的质量得不到保证,将直接影响到测试结果的正确性。

一般来说,通过自动测试工具测试的用例是不需要再进行手工测试的。将自动测试与手工测试有效的结合,并在最终的测试报告中体现自动测试的结果是比较好的方向。

### 4. 没有进行有效、充分的培训

人员和培训是相辅相成的,有了良好的人员而没有良好、有效、充分的培训,测试人员对测试工具的了解缺乏深度和广度,将导致其使用效率很低,结果不理想。这种培训是一个长期的过程,不是通过一两次讲课就能做到的。而且,在实际使用测试工具的过程中,测试工具的使用者还可能存在着这样那样的问题,这也需要有专人负责解决,否则的话,会严重影响使用测试工具的积极性。

### 5. 不考虑公司的实际情况,盲目引入测试工具

有一点很明确,不同的测试工具面向不同的测试目的,具有各自的特点和适用范围,所以不是所有优秀的测试工具都能适应不同公司的需求。某公司怀着美好的愿望花了不小的代价引入测试工具,半年或一年以后,测试工具却成了摆设。究其原因,就是没有考虑公司的现实情况,不切实际地期望测试工具能够满足公司的需要,从而导致了失败。

例如,国内多数软件公司是针对最终用户进行项目开发——工程性质的软件,而不是产品开发。项目开发周期短,不同的用户需求不一样,而且在整个开发过程中用户需求和界面变动较大,这种情况下就不适合引入黑盒测试软件,因为黑盒测试软件的基本原理是录制/回放,对于不停变化的需求和界面,将导致修改和录制脚本的工作量大过测试实施,

运用测试工具不但不能减轻工作量，反而加重了测试人员的负担。这种情况可以考虑引入白盒测试工具提升代码质量。

### 6. 没有形成一个良好的测试工具使用环境

测试工具应用环境需要测试流程和管理机制做出相适应的变化，只有这样，测试工具才能真正发挥作用。例如，对于基于 GUI 录制/回放的自动测试来说，产品界面的改变对脚本正常运行影响较大。再者，白盒测试工具一般在单元测试阶段使用，多数公司单元测试是由开发人员自己完成的，如果没有流程来规范开发人员的行为，在项目进度压力比较大的情况下，开发人员很可能就会有意识地不使用测试工具来逃避问题。所以，有必要将测试工具的使用在开发和测试的流程中明确，如在项目各个阶段所提交的文档中必须包含某些测试工具生成的报告，如集成测试时 DevPartner 工具生成的测试覆盖率报告、Logiscope 生成的代码质量报告。

### 7. 其他技术问题和组织问题

软件测试自动化所需要的测试脚本维护量很大，而且软件产品本身代码的改变也需要遵守一定的规则，从而保证良好的测试脚本使用重复性，也就是说测试自动化和软件产品本身不能分离。

其次，提供软件测试工具的第三方厂家，对客户的应用缺乏足够理解，很难提供强有力的技术支持和解决具体问题的能力。也就是说，软件测试工具和被测试对象——软件产品或系统的互操作性会存在或多或少的问题，加上技术环境的不断变化，所有这些对测试自动化的应用推广和深入都将带来很大的影响。

再者还有安全性的错觉，如果软件测试工具没有发现被测软件的缺陷，并不能说明软件中不存在问题，可能测试工具本身不够全面或测试的预期结果设置不对。

## 11.2 测试工具的分类和选择

在认真分析公司的实际需求之后，就要开始启动测试自动化工作。首先要做的工作是选择测试工具，除了特殊应用的测试工具之外，一般不建议自己开发，选用第三方专业软件测试工具厂家的产品是一种比较明智的做法。要选择测试工具，就要对测试工具的分类和选择的标准有一个清楚的认识。

### 11.2.1 测试工具的分类

测试工具可以从两个不同的方面去分类：

- 根据测试方法不同，分为白盒测试工具和黑盒测试工具。
- 根据测试的对象和目的，分为单元测试工具、功能测试工具、负载测试工具、性能测试工具和测试管理工具等。

### 1. 白盒测试工具

白盒测试工具是针对程序代码、程序结构、对象属性、类层次等进行测试，测试中发现的缺陷可以定位到代码行、对象或变量级。根据测试工具原理的不同，又可以分为静态测试工具和动态测试工具。

- 静态测试工具对代码进行语法扫描，找出不符合编码规范的地方，根据某种质量模型评价代码的质量，生成系统的调用关系图等。它直接对代码进行分析，不需要运行代码，也不需要代码编译链接、生成可执行文件。静态测试工具的代表有 Telelogic 公司的 Logiscope 软件、PR 公司的 PRQA 软件。
- 动态测试工具与静态测试工具不同，需要实际运行被测系统，并设置断点，向代码生成的可执行文件中插入一些监测代码，掌握断点这一时刻程序运行数据（对象属性、变量的值等）。动态测试工具的代表有 Compuware 公司的 DevPartner 软件、Rational 公司的 Purify 系列。

单元测试工具多属于白盒测试工具。

### 2. 黑盒测试工具

黑盒测试工具适用于系统功能测试和性能测试，包括功能测试工具、负载测试工具、性能测试工具等。黑盒测试工具的一般原理是利用脚本的录制（Record）/回放（Playback），模拟用户的操作，然后将被测系统的输出记录下来同预先给定的标准结果比较。黑盒测试工具可以大大减轻黑盒测试的工作量，在迭代开发的过程中，能够很好地进行回归测试。

黑盒测试工具的代表有 Rational 公司的 TeamTest、Robot、Compuware 公司的 QACenter。另外，专用于性能测试的工具包括 Radview 公司的 WebLoad、Microsoft 公司的 WebStress 等工具。

### 3. 其他测试工具

在上述两类测试工具之外还有测试管理工具，这类工具负责对测试计划、测试用例、测试实施进行管理、对产品缺陷跟踪管理、产品特性管理等。测试管理工具的代表有 Rational 公司的 Test Manager、Compuware 公司的 TrackRecord 等软件。

除了上述的测试工具外，还有一些专用的测试工具，例如，针对数据库测试的 TestBytes，对应用性能进行优化的 EcoScope 等工具。

## 11.2.2 测试工具的选择

根据软件产品或项目的需要确定要用哪一类工具，是白盒测试工具还是黑盒测试工具，是功能测试工具还是负载测试工具。即使在特定的一类工具中，也需要从众多不同的产品中做出选择。总的来说，测试工具的选择是测试自动化的一个重要步骤之一，选择一个产品，不外乎根据自己的需求、对不同产品的功能、价格、服务等进行比较分析，选择比较适合自己需要、性能价格比好的 2~3 种产品作为候选对象。然后请这 2~3 种产品的生产

厂家来做演示，并让他们帮助解决实际中的几个比较难或比较典型的测试用例。最后根据演示的效果、商业谈判的价格/服务结果等做出选择。在引入/选择测试工具时，还要考虑测试工具引入的连续性，也就是说，对测试工具的选择必须有一个全盘考虑，分阶段、逐步地引入测试工具。如图 11-2 所示，一般来说，测试工具的选择步骤是：

- (1) 成立小组负责测试工具的选择和决策，制订时间表。
- (2) 确定自己的需求，研究可能存在的不同解决方案，并进行利弊分析。
- (3) 了解市场上满足自己需求的产品，包括基本功能、限制、价格和服务等。
- (4) 根据市场上产品的功能、限制和价格，结合自己的开发能力、预算、项目周期等因素决定是自己开发还是购买。建议购买。
- (5) 对市场上的产品进行对比分析，确定 2~3 种产品作为候选产品。
- (6) 请候选产品的厂商来介绍、演示，并解决几个实例。
- (7) 初步确定。
- (8) 商务谈判。
- (9) 最后决定。

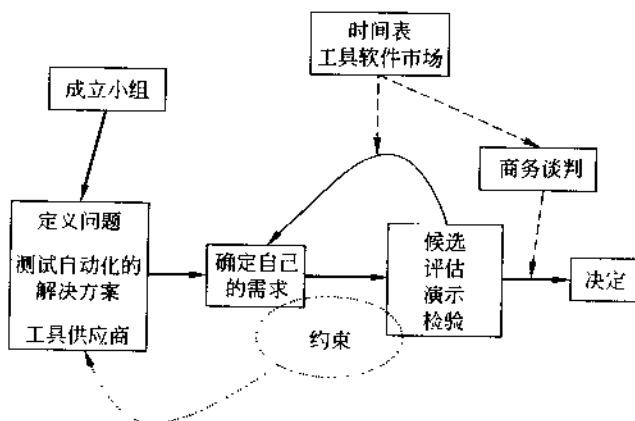


图 11-2 自动化测试的工具选择示意图

至于价格或商业策略，不是本节要讨论的。在本节，主要以功能测试工具为例，说明自动化测试所特有的功能要求，这也是选择测试工具最关键的内容之一。在实际的选择过程中，预算是基础，解决问题是前提，质量和服务是保证，适用才是根本。为不需要的功能花钱是不明智的，同样，仅仅为了省钱而忽略产品的关键功能或服务质量，也不能说是明智的行为。如何评价其功能？或者说，在比较不同产品之间的功能时，要注意哪些方面呢？下面将介绍自动化测试工具的几条关键特征。

### 1. 支持脚本语言 (Script Language)

这是最基本的要求，脚本语言具有与常用编程语言类似的语法结构，可以对已录制好的脚本进行编辑修改。具体来讲，应该至少具备以下功能：

- 支持多种常用的变量和数据类型。

- 支持各种条件逻辑（IF、CASE 等语句）、循环（FOR WHILE）结构等。
- 支持函数的创建和调用。

如果所使用的脚本语言和所熟悉的通用语言（如 Visual Basic、C 语言等）比较接近或一致，测试自动化也就成功了一半。脚本语言的功能越强大，就越能够为测试开发人员提供更灵活的使用空间，而且具备一个良好的、类似于 C 语言的脚本语言结构（头文件、库文件等）也是很重要的，它会大大减少测试脚本的维护工作量。总之，要确认脚本语言的功能是否可以满足测试的实际需求。

## 2. 脚本语言是否支持外部函数库、函数的可重用

如果支持函数调用，就可以建立一套比较通用的函数库。一旦程序做了修改，我们只需把原脚本中的相应函数进行更改，而不用改动所有可能的脚本，从而可以节省大量工作。如果这种函数调用比较容易通过函数之间的参数传递来实现，这也是我们需要的。例如，对于用户登录函数，每次调用时可能都需要使用不同的用户名和口令，此时就必须通过参数的传递，将相关信息在函数内部执行。

除了针对被测系统建立库函数外，一些外部函数同样能够为测试提供更强大的功能，如 Windows 程序中对 DLL 文件的访问，Client / Server 程序中对数据库编程接口的调用等。举一个很简单的例子，完成向数据库插入一条记录的操作，程序可以提示插入成功，但数据是否正确写入数据库中，通常需要手工去数据库里进行检查，以确认功能是否完全正确实现。如果能够在测试脚本中插入检查点，通过调用数据库提供的编程接口检查刚才的操作是否正常执行，这样就无须人工检查，测试程序可以自动完成一些校验。

如果能够通过命令行方式运行测试脚本，可以为测试的执行带来更大的灵活性，这样程序 Build 后就可以自动启动测试脚本，并且可以同时向一组机器发布命令，让它们同时运行不同的测试脚本。

## 3. 对程序界面中对象的识别能力

测试工具必须能够将程序界面中的相应对象（如窗口、按钮、滚动条等）区分并识别出来，录制的测试脚本才具有良好的可读性、修改的灵活性和维护的方便性。如果只支持通过位置坐标来区分对象，问题就会比较多，程序界面稍做改变或者是在不同的屏幕分辨率下，在不同的操作系统或测试环境下（因为有时测试需要在不同的环境下进行，这也是测试工具的优势之一），原有的测试脚本就不能用了。

对于用一些比较通用的开发工具（如 Visual C++、Visual Basic、PowerBuilder、Delphi 等）写的程序，多数测试工具都能区分和标识出程序界面里的所有元素，但对一些不太普及的开发工具或库函数，工具的支持会比较差，因此，在测试工具选择方面，对开发语言的支持也是很重要的一项。当然程序难免会存在一些确实比较难于标识的对象，如位图（Bitmap）对象等，而这些对象在程序中可能还要完成一些功能或者执行相应的操作，那么这些可能出现的问题在产品软件设计阶段就应该考虑。也就是前面所说的，软件开发和测试要相互照应，一方面要实现软件必须实现的功能，还要保证软件的可测试性，为了在功能的灵活性和可测试性之间达到平衡，有时会采取不得而为之的折中处理方法。



#### 4. 抽象层

抽象层和对象识别能力有一定的关系，也是解决捕捉、回放过程中针对程序界面进行存在的问题，在被测应用程序和录制生成的测试脚本之间增加一个抽象层。

抽象层用于将程序界面中存在的所有对象实体——映射成逻辑对象，测试就可以针对这些逻辑对象进行，而不需要依赖于界面元素的变化，以减少测试脚本建立和维护的工作量。有些工具称抽象层为测试映射图（Test Map）、测试帧（Test Frame）或取值映射图（Get Map）。举个例子，就比较容易理解抽象层的作用。如程序中经常有的用户登录窗口，一般需要输入用户名和口令，我们可以在脚本中将这两条信息标识为“NAME”和“PASSWORD”，对应程序中这两个变量名。这就是所建立的抽象层。在很多脚本里，有同样的信息标识去处理不同的用户登录操作。如果下一版本的程序中，登录窗口中两条输入信息的标识变成了“USERNAME”和“PWORD”，这时就不需要把所有脚本都修改一遍，只要简单地对抽象层中这两个对象的标识进行修改就可以了。脚本执行时通过抽象层自动会使用新的对象标识。有些工具软件支持程序界面对象自动搜索，建立所发现对象的抽象层，当然也可以脚本手工编程定义所需要的抽象层。

#### 5. 分布式测试（Distributed Test）的网络支持

在互联网上的应用软件，如网上会议系统、远程培训系统、聊天系统等，一般是用于协同工作、相互通信等模式中，支持多用户来共同操作软件，这时对软件自动化测试有更高的要求，包括：

- 测试工具进行测试时自身传输的数据量很小，具有很强的独立性，这样对被测的软件影响很小。
- 能按照事先设置的任务执行时间表进行，即在指定时间、指定设备上执行指定的测试任务；或者是按照任务设定的先后次序、相互的依赖性进行，如一个用户要加入网络会议，只能在某个用户已经启动一个网络会议之后进行，Rational Robot就有这样的功能。
- 当两个测试任务要同时打开一个文件时，能保持协调或协同处理，避免出现资源竞争问题。

#### 6. 支持数据驱动测试（Data-Driven Test）

测试工具能支持对流行的数据库（Oracle、SQL Server、Access 等）格式文件的操作，这样有利于测试脚本的代码和数据输入分离，减少代码的编程和维护工作量，也有利于测试用例的扩充和完善。在数据驱动测试中，测试脚本通过从事先准备的数据库或文件中读取数据，在执行测试过程中将结果数据写入数据库或文件中，或直接将结果和事先保存在数据库中的预期结果值进行比较。

#### 7. 具有脚本开发良好的环境

对于可以自动执行的测试任务，通常我们会在上班时将任务定制好，下班后启动任

务执行，第二天上班再来检查测试执行的结果。但是，经常会出现本来需要执行整晚的工作，第二天发现才执行了5分钟就因为程序一些异常错误而终止了，而且这种情况经常发生。因此，测试工具应有相应的容错处理系统，可以自动处理一些异常情况而对系统进行复位，或者允许用户设置是否可以跳过某些错误，然后继续执行下面的任务。

其次，能提供类似软件集成开发环境中的调试功能，支持脚本单步运行、设置断点、得到变量返回结果等，可以更有效地对测试脚本的执行进行跟踪、检查，迅速定位问题。

最后，测试脚本的开发，经常也是多个工程师共同工作，需要一个团队的开发环境，即对脚本代码能很好控制、管理，可以对测试数据文件、测试脚本、对象抽象层进行统一管理。这一点对大型测试项目尤其重要，有利于自动化测试管理、缩短测试开发周期、减少测试脚本的错误。

## 8. 其他功能

- 图表功能。测试工具生成的结果可以通过一些统计图表表示，这样做会更直观些。但问题最终要由人进行解释，而且，查看最终报告的人员不一定对测试很熟悉，因此，测试工具能否生成结果报表，能够以什么形式提供报表是需要考虑的因素。
- 测试工具的集成能力。测试工具的引入是一个长期的过程，应该是伴随着测试过程改进而进行的一个持续的过程。因此，测试工具的集成能力也是必须考虑的因素。这里的集成包括两个方面的意思，首先，测试工具能否和开发工具进行良好的集成；其次，测试工具能够和其他测试工具进行良好的集成。
- 操作系统和开发工具的兼容性。测试工具可否跨平台，是否适用于公司目前使用的开发工具，这些问题也是在选择一个测试工具时必须考虑的问题。

# 11.3 测试工具的主流产品介绍

谈到软件测试工具，肯定会提到 IBM-Rational、Compuware、Mercury Interactive (MI) 这3家世界著名的软件公司，其中任何一家公司的产品就可以构成一个较完整的解决方案，所以在本章后面几节会相继介绍这3家公司的整体解决方案。在介绍这些整体解决方案之前，先按工具所属的类别来介绍测试自动化工具，主要涉及其他一些公司的产品。

## 11.3.1 面向开发的单元测试工具

单元测试一般采用白盒测试方法，要根据程序内部的实现来完成测试，所以必须和特定的语言结合起来进行，所以针对语言不同，单元测试工具也不同。而且一些开发环境，如 Microsoft Visual Studio 和 Borland 公司的 JBuilder 都含有单元测试工具软件。单元测试工具可以根据不同的语言进行分类，如：

- 功能强大的自动化 C/C++ 单元级测试工具 Panorama C++、C++Test、Numega。
- JUnit 是一个开发源代码的 Java 测试框架，用于编写和运行可重复的测试。

单元测试工具还可以根据工具的功能特点进行分类,如:

- 内存资源泄漏检查工具 Numega 中的 BounceChecker、Rational 的 Purify 等。
- 代码覆盖率检查工具 Numega 的 TrueCoverage、Rational 的 PureCoverage、TeleLogic 公司的 Logiscope。
- 代码性能检查工具 Logiscope 和 Macabe 等。
- 软件纠错工具 Rational Purl 等。

第9章详细介绍了目前流行的单元测试工具JUnit,本小节介绍3种单元测试工具JTest、C++ test 和Test。

### 1. JTest

JTest 通过自动生成和执行能够全面测试类代码的测试用例,自动测试类的所有代码分支,从而彻底检查被测类的结构,使白盒测试完全自动化。JTest 使用一个符号化的虚拟机执行类搜寻未捕获的运行时异常。对于检测到的每个异常情况,JTest 报告一个错误,并提供导致错误的栈轨迹和调用序列。

JTest 报告下列未捕获的运行时异常。

- 行为错误的方法:这些方法对于某些特定输入不会产生异常,必须修改这些代码。
- 非预期参数:这一问题出现在当某方法遇到非预期的输入(不知任何处理)而产生一个异常。这些问题的修正可以通过检查输入并产生一个 `IllegalArgumentException` (IAE) (假如该输入是非法的)。改正这类问题可以使代码更清晰、更易维护。
- 行为正确的方法:这时,方法的正确输出是产生一个异常。在这种情形下,建议开发人员修改代码,将这类异常的产生置于方法的 `throw` 子句中。这会得到更清晰的代码,并易于维护。
- 仅为开发人员使用的方法:在这种情况下,这些方法“不被假设”成处理 JTest 生成的输入,开发人员是这些方法的惟一使用者,并且不传递这些输入参数。最好的办法是修改这些代码,让它产生一个 IAE。这将带来额外的好处,使代码更易阅读。

### 2. C++Test

C++Test 是一个功能强大的自动化 C/C++单元级测试工具,可以自动测试任何 C/C++函数、类,自动生成测试用例、测试驱动函数或桩函数,在自动化的环境下完成单元测试。其单元级的测试覆盖率可以达到 100%。C++Test 能够自动测试代码构造(白盒测试)、测试代码的功能性(黑盒测试)和维护代码的完整性(回归测试)。C++Test 具有的特性有:

- 即时测试类/函数。
- 支持极端编程模式下的代码测试。
- 自动建立类/函数的测试驱动程序和桩调用。
- 自动建立和执行类/函数的测试用例。
- 提供快速加入、执行说明和功能性测试的框架。

- 执行自动回归测试和组件测试（COM）。

3. .Test

.Test 是专为 .NET 开发而推出的、使用方便的自动化单元级测试与静态分析工具。

- 使用超过 200 条的工业标准代码规则对所写代码自动执行静态分析。这些规则有助于将 .NET 全面的编程技术和领域知识应用到代码中，防止出现错误。
- 自动测试代码构造与功能。.Test 具有智能特性，能提取代码、审查代码，生成测试用例，自动完成单元测试。.Test 产生的单元测试可以由用户自定义。
- 通过自动衰减测试自动地维持代码完整性。
- 可用于任何 Microsoft .NET 框架的语言，如 C#、Visual Basic.NET 和 Managed C++。

11.3.2 负载和性能测试工具

对于联机事务处理（OLTP）方式数据库应用、Web 浏览和视频点播等应用系统，即使通过了单元测试、集成测试、功能测试，用户还常会有疑问，如这套系统能不能承受大量的并发用户同时访问？到底能承受多少个用户同时访问？如果同时有 1 万个用户在一个小时内访问服务器会不会导致服务器崩溃？如果数据库中有 100 万条记录，这时用户的操作是不是慢得像蜗牛爬行？这些疑问的解决要借助于软件测试手段，而这种测试手段通过手工模拟是很难进行的，这就需要借助负载和性能测试工具。

负载和性能测试工具在软件测试自动化中占据着很重要的位置，因为负载和性能测试是软件手工测试的弱项，是工具的强项。优秀产品主要有 MI 公司的 LoadRunner、Compuware 的 QALoad、Rational 的 SQA Load、Performance、Visual Qualityfy。在后面几节相继会介绍 LoadRunner、QALoad，这里给出 3 个主流产品的对照表，如表 11-1 所示，有利于用户在购买工具产品时做出选择。

表 11-1 主流商业性能测试工具的比较

属性	LoadRunner	QALoad	WebLoad
出品公司	MI	Compuware	Radview
价格	昂贵	较贵	一般
安装配置的复杂性	简单	简单	一般
操作性	较复杂	简单	简单
支持测试对象	各种中间件/数据库/应用服务器的性能监控/企业架构（J2EE 和 .NET）的测试	客户/服务器系统、企业资源配置（ERP）和电子商务应用	Web Application
支持平台	Windows、UNIX 或 Linux	HP-UX、IBM AIX、Sun Solaris、Linux、NT/2000	UNIX、Windows
支持数据库	DB2、SQL Server、Oracle、Sybase	ADO、DB2、Oracle、Sybase、SQL Server、ODBC	ADO、DB2、Oracle、Sybase、SQL Server、ODBC

续表

属性	LoadRunner	QALoad	WebLoad
支持协议	Web、HTTP(s)、SOAP、Streaming、WAP、Winsock、XML	HTTP、SSL、SOAP、XML、Streaming、Media	XML、Java、EJB、ActiveX、WAP、HTTP、SNMP、Real/m\$streaming
脚本语言	类似 C++	C/C++ 和 Visual C++	JavaScript
自动数据生成	Y	Y	Y
脚本调试	Y	Y	Y
报表定制功能	Y	Y	Y
功能点	创建虚拟用户，创建真实的负载，定位性能问题，重复测试，保证系统发布的高性能等	预测系统性能、通过重复测试寻找瓶颈问题、从控制中心管理全局负载测试、快速创建仿真的测试、验证应用的可扩展性	强大的专业网站性能测试，虚拟多用户
虚拟用户上限数量	成千上万	成百上千	理论上无限，不过受机器的限制，同时运行太多会影响结果的准确性

### 11.3.3 GUI 功能测试工具

基于 GUI 功能测试工具在软件测试自动化中占有重要的地位，其基本原理是：将操作应用程序的各种动作和输入记录下来，包括键盘操作、鼠标单击等，生成一个脚本文件。这个脚本以后可以被“回放（playback）”，也就是能重复上一次所操作的动作，实现自动运行和测试。在实际测试过程中，还要根据测试需求对录制的脚本进行一些必要的修改或加入一些参数，如选择不同的测试数据、脚本中插入检查点（Check Point）进行跟踪调试等。

基于 GUI 功能测试工具主要适合回归测试阶段。当一个应用开发基本完成后，程序界面基本定型，虽然业务的需求会频繁变化，但测试脚本结构基本不需要改动，只需要做些小调整，就可以自动运行，可大大提高测试的效率和测试的准确性。

目前主要产品有 MI 公司的 WinRunner、Compuware 的 QARun、Rational 的 SQA Robot、MS Visual Test Suite 等。

### 11.3.4 基于 Web 应用的测试工具

现在 Web 应用非常广泛，相应的测试工具也比较多。基于 Web 应用的测试工具主要进行链接检查、HTML 检查、Web 功能和 Web 站点安全性等各个方面的测试。主要 Web 测试工具有 MI 公司的 Astra 系列（如 Astra QuickTest）、RSW 公司的 E-Test Suite 等，Web 系统测试工具有 WorkBench、Web Application Stress（WAS）Tool、页面链接测试 Link Sleuth 等。

## 1. Web Application Stress (WAS) Tool

微软的 WAS 允许以不同的方式创建测试脚本：可以通过使用浏览器走一遍站点来录制脚本，可以从服务器的日志文件导入 URL，或者从一个网络内容文件夹选择一个文件。当然，也可以手工地输入 URL 来创建一个新的测试脚本。

WAS 可以使用任何数量的客户端运行测试脚本，全部都由一个中央主客户端来控制。在每一个测试开始前，主客户机透明地执行以下任务：

- 与其他所有的客户机通信。
- 把测试数据分发给所有的客户端。
- 在所有客户端同时初始化测试。
- 从所有的客户端收集测试结果和报告。

这个特性非常重要，尤其对于要测试一个需要使用很多客户端的服务器群的最大吞吐量时非常有用。除此之外，WAS 是被设计用于模拟 Web 浏览器发送请求到任何采用了 HTTP1.0 或 1.1 标准的服务器，而不考虑服务器运行的平台。除了它的易用性外，WAS 还有很多其他有用的特性，包括：

- 对于需要署名登录的网站，它允许创建用户账号。
- 允许为每个用户存储 Cookies 和 Active Server Pages (ASP) 的会话信息。
- 支持随机的或顺序的数据集，以用在特定的名称-值对。
- 支持带宽调节和随机延迟（“思考的时间”）以更真实地模拟显示情形。
- 支持 SSL (Secure Sockets Layer) 协议。
- 允许 URL 分组和对每组的点击率的说明。
- 提供一个对象模型，可以通过 Microsoft Visual Basic Scripting Edition (VBScript) 处理或者通过定制编程来达到启动、结束和配置测试脚本的效果。

## 2. WebKing

WebKing 是一个独特的工具，用以帮助开发人员防止和检测多层次 Web 应用中的错误。WebKing 采用已经证明能够有效地改善 C/C++ 和 Java 代码质量的测试技术，可以自动进行白盒、黑盒和回归测试，以及网盒 (Web-box) 测试——一种全新的针对动态网页的单元测试方法。

- WebKing 的白盒测试用例对表格对象自动生成一组用户输入，然后显示生成的目录和动态页，分析网站的结构，找到测试的最佳途径。简单地单击 Test All 按钮，WebKing 将自动测试每一个静态和动态网页，并发现其中的构造错误。同时 WebKing 的 CodeWizard 功能执行最重要的 HTML、CSS 和 JavaScript 编码标准检查，WebKing 还检查所有的链接。
- WebKing 执行黑盒测试以保证网站应用能满足功能要求，WebKing 的路径视图显示所有潜在的路径，包括一组默认的路径。可以建立自己的测试用例以测试特定的路径。WebKing 还将测试所有的链接，并执行编码标准检查。
- 独有的 RuleWizard 特性让用户使用图形脚本语言建立监视动态网页内容的规则。

虽然某些动态网页的内容依赖于用户输入, RuleWizard 标识出不变的部分。CodeWizard 自动执行这些规则以保证动态网页能够符合设计要求。

- 网盒测试 (Web-box testing) 是为 Web 应用开发而特别设计的创新技术, 能自动化修改、编译、分发和测试网站等生命周期活动, 帮助用户建设一个建立、发布和测试每个程序或脚本语言的基础结构。当每次修改网站时, WebKing 能够自动执行回归测试, 如果发现问题, 会取消发布修改内容。当发布时, WebKing 自动显示动态页, 验证页面的准确性。同时, 它检查链接、拼写、HTML、CSS 和 JavaScript。
- WebKing 的特性如下:
  - ◆ 防止和检测动态网站中的错误。
  - ◆ 测试一个动态网站中所有可能的路径。
  - ◆ 强化 HTML、CSS 和 JavaScript 编程标准。
  - ◆ 帮助建立自动监视动态页面内容的规则。
  - ◆ 检查中断的链和孤立的文件。
  - ◆ 防止含有错误的页面。
  - ◆ 记录有关网站使用的各类文件统计信息。
  - ◆ 集成各类插件和第三方工具。
  - ◆ 发布网站时自动执行许多基本命令。

### 3. SOAPtest

SOAPtest 是一个测试 Web 服务程序的工具, 运行服务功能测试、负载测试和客户测试来预防错误。SOAPtest 使服务的功能测试很容易实现, 它通过来源于 WSDL 自动创建的测试套件。用于服务功能的测试套件同样适用于负载测试, 不但可以测试服务对并发请求的响应, 而且可以确认测试负载是否导致产生功能上的问题。可以让 SOAPtest 仿效服务组件接受客户发送的适当请求。客户处理服务器返回的响应。可以用 SOAPtest 当做一个代理服务器, 显示 Web 服务器和客户之间的消息。

SOAPtest 的特性如下:

- 彻底了解 SOAP 协议和 Web 服务方面的问题。
- 从 WSDL 文档中自动创建测试套件, 用于客户端和服务端测试。
- 自动服务程序功能测试、负载测试和客户测试。
- ◆ 使用 XSLT 到 SOAP 消息实现客户端和服务端正确的事务。

## 11.3.5 软件测试管理和其他工具

在测试过程中, 要涵盖单元测试、集成测试、系统测试、回归测试、交付测试等各个阶段, 如何有效地组织管理这些不同阶段的测试尤为重要, 这就需要软件测试管理工具。软件测试管理工具能管理整个测试过程, 从测试计划、测试例程、测试执行、测试结果到测试报告, 提供一个基于中央数据库的、协同合作的环境。虽然测试人员分布在各地, 但

不管在何时何地都能参与整个测试过程。软件测试管理工具主要有 MI 的 Test Director、Rational 公司的 Test Manager、Silicon Valley Networks 公司的 TestExpert 等。

### 1. 嵌入式测试工具

- ATTOLTestW8fC 公司自动生成测试代码的软件测试工具, 特别适用于嵌入式实时应用软件单元和通信系统测试, 在法国市场居领先地位。
- CodeTest 是 Applied Microsystems 公司的产品, 是广泛应用的嵌入式软件联机测试工具。CodeTest 为追踪嵌入式应用程序, 分析软件性能, 测试软件的覆盖率以及存储体的动态分配等提供了一个实时联机的高效解决方案。CodeTest 还是一个可共享的网络工具, 它将给整个开发和测试团队带来高品质的测试手段。CodeTest 支持所有的 16/32 位 CPU 和 MCU, 支持总线频率高达 100 MHz, 可通过 PCI/VME 总线、MICTOR 插头对嵌入式系统进行联机测试, 无须改动用户的 PCB, 与用户系统的连接方便。
- GammaRay 系列产品主要包括软件逻辑分析仪 GammaPfiler、可靠性评测工具 GammaRET 等。
- Logiscope 是 TeleLogic 公司的工具套件, 用于代码分析、软件测试、覆盖测试。
- LynxInsure++ 是 Lynx Real-Time Systems 公司的产品。基于 LynxOS 的应用代码检测与分析测试工具。
- MessageMaster 是 Elvior 公司的产品。它是测试嵌入式软件系统工具, 向环境提供基于消息的接口。
- VectorCast 是 Vector Software 公司产品, 由 6 个集成的部件组成, 自动生成测试代码, 为主机和嵌入式环境构造可执行的测试架构。

### 2. 数据库测试工具 DataRecon

DataRecon 是一个自动数据库认证和监视工具, 不仅能确认数据源, 还可以生成和管理数据库测试用例。DataRecon 允许数据库设计认证、数据有效性校验以防止数据重复, 结构比较测试使得数据库开发到产品更平滑。另外, 它还是一个监视工具, 可以生成网页格式的报告放到网上让人家共享。

DataRecon's SQL 测试特性联合数据库分析工具可以快速、容易和重复进行数据有效性校验。DataRecon's Database Structure Test 可以让用户比较在开发环境和产品环境下的数据结构。其特点有:

- 在局域网内发布网页报告。
- 方便的多用户共享。
- 和所有主要的关系数据库兼容。
- 预防数据库表失败。
- 完成实时表捕获而不需要 SQL 知识。
- 快速的数据库结构有效性检验。
- 排除数据和结构上的错误。



- 确定数据库容量。
- 强制编码标准。
- 自动和可重复的数据库测试。

### 3. 缺陷跟踪工具

缺陷跟踪工具主要有 Rational ClearQuest、Compuware 公司的 TrackRecord、WEBsina 公司的 bugzero 和 SilkRadar，以及著名的免费软件 bugzilla。这里以 SilkRadar 为例，对缺陷跟踪工具做一个简单的介绍。

SilkRadar 对软件缺陷进行记录及缺陷处理结果状态进行自动跟踪、记录、归类处理，能够灵活满足各种业务环境和各种产品的需求。SilkRadar 提供行为驱动的工作流程，能够帮助开发人员自动完成对缺陷管理的相关处理。

- 可以记录软件测试过程中失败的测试结果以及用户报告的问题。
- 通过电子邮件通知、自动分配规则、预先定义的优先级等对问题进行分解。
- 能够按照预先设定的规则对各个缺陷状态按照其生命周期进行相应处理，迅速地将任务分派给相关人员进行处理。
- 定制处理的卡片帮助开发/测试人员关注对自己最重要的信息，如每个缺陷可能造成的风险等。

SilkRadar 允许用户通过 Web 方式使用。这样有利于不同地点间甚至跨国的各个开发团队间进行的缺陷管理。在浏览器中，允许用户进行自定义链接，访问自己所关心的区域。这样，用户可以在任意时刻快速找到关键信息，从而提高工作效率。

SilkRadar 提供了许多预定义的、用户定制的报表、图表以及用于有效提高表达项目状态的语汇（query），使用统一定义的信息提高沟通效率，有助于解决缺陷管理问题。此外，可以保存并重用个人级、工作组级、公司级的各种查询条件。通过使用 SQL 语言，可以从不同数据库中，提取复杂的、跨产品的问题或信息。

## 11.4 IBM-Rational 产品的整体解决方案

IBM-Rational（瑞理）软件公司提供一种软件发展平台。该平台的使用加快了软件产品开发的速度，提高了质量和增强了可预见性。它将完整的生命周期解决方案、软件工程的最佳经验、市场领先的工具以及专业服务完美地结合在一起。

### 11.4.1 Rational 测试产品结构

#### 1. 具有代表性的测试产品

- Rational Robot：对整个应用程序进行全面的测试。
- Rational TestManager：管理所有测试活动和工件的核心平台。
- Rational TestFactory：无须用户介入，自动查找运行时错误，并生成优化脚本以用

于回归测试。

- **Rational PureCoverage**: 提供代码覆盖分析, 找出未经测试的程序代码。
- **Rational Quantify**: 快速识别应用程序性能瓶颈。
- **Rational Team Test**: 确实做好产品发布的准备。
- **Rational Purify**: 查找造成程序崩溃的那些难以发现的运行时错误。

## 2. 其他测试相关产品

- **Rational Suite TestStudio**: 测试各种工具套件产品, 保证产品发布的质量。
- **Rational ClearQuest**: 集成的缺陷跟踪及变更请求管理。
- **Rational ClearCase**: 简化变更过程, 帮助团队控制在开发过程中不断完善的所有工件。
- **Rational Unified Process**: 切实可行的流程, 通过浏览器访问的知识库, 提示团队使用开发最佳经验。
- **Rational RequisitePro**: 将需求管理嵌入整个软件开发生命周期。

## 11.4.2 Rational 全套测试解决方案的特点

**Rational Suite TestStudio** 主要提供快速提升功能(包括回归测试)、可靠性、性能(包括负载测试、压力测试等)和测试管理等综合测试能力的解决方案。

**Rational Suite TestStudio** 提供了一种集成测试解决方案, 使测试人员就产品的功能性、可靠性和性能, 进行全方位的质量测试。所有操作都由 **Rational TestManager** 进行管理和控制, 该工具能整合测试计划、执行和分析等活动, 包括遗留的和专用的测试资料。它包括一整套自动化测试和缺陷跟踪工具以及 **Rational Team Unifying Platform** (**Rational** 团队统一平台), 该平台通过提供对项目需求、变更请求、测试资料及其他数据的共享, 提高了团队的工作效率。

### 1. 满足项目需求的功能性测试

基于 **IBM Rational** 的测试解决方案, 可以方便地解决以下功能测试问题:

- 回归测试。
- 利用数据池方便地解决大批量数据驱动的功能测试。
- 完善的功能测试管理平台。

功能性测试, 可以确保应用程序满足产品规格说明和测试计划的每一条业务需求。

**Rational Suite TestStudio** 的目标, 是使功能性测试变得更简单、有效并可重复执行。

### 2. 功能强大的 Robot

**Rational Robot** 可以对在各种独立开发环境(IDE)中开发的应用程序, 创建、修改并执行功能测试、分布式功能测试、回归测试以及整合测试, 记录并回放能识别业务应用程序对象的测试脚本。可以快速、有效地跟踪、报告与质量保证测试相关的所有信息, 并将

这些信息绘制成图表。Robot 的回归测试与 Purify® 结合使用完成可靠性测试, 与 PureCoverage® 结合使用完成代码覆盖计算, 与 RationalQuantify® 结合使用完成应用程序性能测试。

Rational TestFactory® 使可靠性测试自动化, 该工具可以自动找出应用程序中的缺陷, 将在自动认知用户界面的基础上, 创建测试流程, 对应用程序进行严格的测试。

### 3. 进行功能测试的同时自动完成全面的可靠性测试

Rational TestStudio 中包含了可靠性测试, 即通过 PurifyPlus 在短期内快速提升单元测试能力和运行时分析能力的团队, PurifyPlus 包含 Rational Purify、Quantify 和 PureCoverage 三个产品。Purify 主要针对软件开发过程中难于发现的内存错误、运行时错误, 能自动地发现错误、准确地定位错误、提供完备的错误信息, 从而减少调试时间, 帮助开发团队找出缺陷并最终形成高质量的产品。Quantify 主要解决软件开发过程中的性能问题, 方便地查明并显示应用程序的性能瓶颈, 提供了一个性能数据的全局图形化视图。PureCoverage 提供应用程序的测试覆盖率信息, 能自动找出代码中未经测试的代码, 保证代码测试覆盖率, 可针对每次测试生成全面的覆盖率报告, 可以归并程序多次运行所生成的覆盖数据, 并自动比较测试结果, 以评估测试进度、测试工作的效果。

### 4. 查找响应时间瓶颈的性能测试

基于 IBM Rational 的测试解决方案, 可以方便地解决以下性能测试问题:

- 准确地模拟性能测试的负载模型。
- 方便地模拟大批量的虚拟用户。
- 方便地为不同的虚拟用户提供所需的大批量的测试数据。
- 准确地提供各种性能分析报告。

Rational Suite TestStudio 提供三种级别的诊断信息, 开发人员可以对导致性能不佳的业务事务处理、底层客户端调用和系统资源进行分析, 来找出产生性能瓶颈的原因。

### 5. 提供完备的、高度集成的测试管理平台

Test Manager 是一种开放的、可扩展的框架, 将所有测试工具、工件和数据组合在一起, 帮助团队制订并优化其质量目标。Test Manager 作为测试管理的核心平台, 统一管理从测试输入、测试计划、测试设计、测试执行和测试结果分析等整个测试过程。通过它, 团队可以创建、维护或对照测试计划, 来组织测试用例、迭代和配置, 以及外部工件 (例如文档、模型、变更请求和电子表格), 创建测试报告, 提交测试结果。作为一种集成解决方案, TestManager 与 Rational 其他工具一起提供即时的可追踪性, 用于测试诸如 RequisitePro 需求、RationalRose 模型和 ClearQuest 变更请求等输入。

TestManager 还具有开放的可扩展 API, 使测试人员可以为自己专有的输入类型编写适配程序。还可以使用诸如 Rational Robot 和 TestFactory 等自动化工具创建测试用例脚本, 或使用自己的脚本 (包括手动测试脚本)。

# 11.5 Mercury Interactive 产品的 整体解决方案

Mercury Interactive 是从事企业测试和性能管理解决方案的供应商,其自动测试软件和网络管理联机服务帮助客户提供和维护高性能的应用系统。

## 11.5.1 MI 产品结构

Mercury Interactive (MI) 为行业提供一整套综合自动软件测试解决方案,见表 11-2。

表 11-2 Mercury Interactive 的主要应用

应用系统测试方案	应用系统性能管理方案
负载测试: LoadRunner, Astra LoadTest	性能管理: TOPAZ
功能测试: WinRunner, Astra QuickTest	
测试管理: TestDirector, Astra FastTrack	

其测试解决方案主要集中在 IT 行业,包括电子商务/网站应用、商品化软件和客户/服务器应用测试,主要产品包括(如图 11-3):

- TestDirector——基于 Web 集成的测试管理工具,组织和管理整个测试过程。
- WinRunner/XRunner——功能测试工具,测试 Windows 和 X-Windows 应用是否能正确工作。
- QuickTest——一个独创性的技术,能简化、加速测试。
- LoadRunner——负载测试工作,预测系统的性能和表现。
- TestSuite Enterprise——一整套的自动测试软件包,包括 LoadRunner、Win/Runner、和 TestDirector。

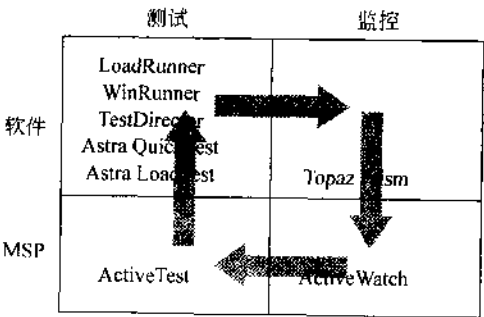


图 11-3 MI 产品结构体系

## 11.5.2 MI 3 个重量级产品的介绍

Mercury Interactive 的 LoadRunner、WinRunner 和 TestDirector 堪称是这家公司的 3 个重量级产品，也是软件测试领域世界级著名产品。

### 1. 负载测试工具 LoadRunner

Mercury Interactive 的 LoadRunner 是一种适用于企业级系统、各种体系架构的自动负载测试工具，通过模拟实际用户的操作行为和实行实时性能监测，帮助更快地查找和发现问题，预测系统行为并优化系统性能。此外，LoadRunner 能支持广范的协议和技术，为一些特殊环境提供特殊的解决方案。

- LoadRunner 可以记录下客户端的操作，并以脚本的方式保存，然后建立多个虚拟用户，在一台或几台 PC 上模拟上百或上千虚拟用户同时操作的情景，同时记录下每一事务处理的时间、中间件服务器峰值数据、数据库状态等，并根据测试结果分析系统瓶颈，输出各种定制压力测试报告。
- 使用其 Virtual User Generator，能简便地创立起系统负载。该引擎能够生成虚拟用户，以虚拟用户的方式模拟真实用户的业务操作行为。它先记录下业务流程，然后将其转换为测试脚本，并且可以对所建立的测试脚本进行参数化操作，使之利用几套不同的实际发生数据来测试应用程序，来匹配多个实际用户的操作行为，从而反映出系统真正的负载能力。
- 创建真实的负载。LoadRunner 的 Controller 能组织起多用户的测试方案，其 Rendezvous 功能提供一个互动的环境，能建立持续且循环的负载，限定负载又能管理和驱动负载测试方案。而且可以利用它的日程计划服务来定义用户在什么时候访问系统以产生负载，这样使测试过程高度自动化。使用 AutoLoad 技术可以提供更多的测试灵活性，可以根据目前的用户人数事先设定测试目标，优化测试流程。例如，可以控制应用系统承受的每秒点击数或每秒的交易数量、频率、用户的思考时间和连接速度等。
- 定位性能问题。LoadRunner 内含集成的实时监测器，在负载测试过程的任何时候，可以观察到应用系统的运行性能，实时显示交易性能数据（如响应时间）和其他系统组件（包括 Application Server、Web Server、网路设备和数据库等）的实时性能。再者，利用 ContentCheck，在 Virtual users 运行时，检测应用程序的网络数据包内容，从中确定是否有错误内容传送出去，可以判断负载下的应用程序功能正常与否。
- 分析结果以精确定位问题所在。测试完毕后，LoadRunner 收集、汇总所有的测试数据，提供高级的分析和报告工具，以便迅速查找到问题并追溯原由。如 Web 交易细节监测器记录所有的图像、框架和文本下载到每一网页上所需的时间，然后分解用于客户端、网络和服务端上端到端的反应时间，便于确认问题，定位查找真正出错的组件，或者将网络延时进行分解，以判断 DNS 解析时间，连接服务器或 SSL 认证所花费的时间。

- 其他功能。LoadRunner 完全支持基于 Java 平台应用服务器 Enterprise Java Beans 的负载测试, 支持无线应用协议 WAP 和 I-mode, 支持 Media Stream 应用, 可以记录和重放任何流行的多媒体数据流格式来诊断系统的性能问题, 查找原由、分析数据的质量。

## 2. WinRunner

Mercury Interactive 公司的 WinRunner 是比较常用的自动化功能测试软件, 用于检测应用程序是否能够达到预期的功能及正常运行, 对复杂的企业级应用 (包括 ERP 系统、CRM 系统等) 不同发布版进行测试, 提高测试人员的工作效率和质量。

(1) 用 WinRunner 创建一个测试, 按照预期设计执行并将业务处理过程记录到测试脚本, 支持测试脚本的编辑、扩展、执行和报告测试结果, 并且保证测试脚本的可重复使用, 自动记录操作并生成所需的脚本代码。

(2) 插入检查点来检验数据。在脚本中可以插入不同类型的检查点, 包括文本、位图和数值等, 并设定哪些数据库表和记录需要检测。在测试运行时, 测试程序就会自动会收集一套数据指标、核对数据库内的实际数值和预期的数值。WinRunner 自动显示检测结果, 在有更新/删除/插入的记录上突出显示以引起注意检查, 从而确认应用程序是否运行正常。

(3) 增强测试。WinRunner 针对相当数量的企业应用中的非标准对象, 提供了 Virtual Object Wizard 来识别以前未知的对象。其数据驱动向导 (Data Driver Wizard) 可以把一个业务流程测试转换为数据驱动测试, 从而反映多个用户各自独特且真实的行为。还可以通过 Function Generator 增加测试的功能, 从目录列表中选择一个功能增加到测试中以提高测试能力。例如可以选择 calendar, 然后从日历功能的下属目录中选择, 如 Calendar\_select\_date(), 就可以直观地输入参数, 把这个功能插入到测试脚本中。

(4) 分析结果。WinRunner 通过交互式的报告工具来提供详尽的、易读的报告, 会列出测试中发现的错误内容、位置、检查点和其他重要事件, 帮助对测试结果进行分析。这些测试结果还可以通过 MI 自己的测试管理工具 TestDirector 来查阅。

(5) 维护测试。每次记录测试时, WinRunner 会自动创建一个 GUI Map 文件以保存应用对象, 这些对象分层次组织, 既可以总览所有的对象, 也可以查询某个对象的详细信息。通过修改一个 GUI Map 文件而非无数个测试, 方便地实现测试重用。

## 3. TestDirector

TestDirector 是一套测试管理软件。可以使用它来规范科学的测试管理流程, 建立起针对项目的测试方案和计划, 消除组织机构间、地域间的障碍, 让测试人员、开发人员或其他 IT 人员通过一个中央数据仓库, 在不同地方就能交互测试信息。TestDirector 将测试过程流水化——从测试需求管理, 到测试计划, 测试日程安排, 测试执行到出错后的错误跟踪——仅在一个基于浏览器的应用中便可完成, 而不需要每个客户端都安装一套客户端程序。

- 需求管理。程序的需求驱动整个测试过程。TestDirector 的 Web 界面简化了这些需求管理过程, 以此可以验证应用程序的每一个特性或功能是否正常。通过提供一

个比较直观的机制将需求和测试用例、测试结果和报告的错误联系起来,从而确保能达到最高的测试覆盖率。即使频繁的更新,仍能简单地将应用需求与相关的测试对应起来。

- 测试计划的制定。其 Test Plan Manager 指导测试人员如何将应用需求转换为具体的测试计划,组织起明确的任务和责任,并在测试计划期间为测试小组提供关键要点和 Web 界面来协调团队间的沟通。它提供了多种方式来建立完整的测试计划:
  - ◆ 可以从草图上建立一份计划。
  - ◆ 根据用 Requirements Manager 所定义下的应用需求,通过 Test Plan Wizard 快捷地生成一份测试计划。
  - ◆ 如果已经将计划信息以文字处理文件形式,如 Microsoft Word 方式存储,可以再利用这些信息,并将它导入到 Test Plan Manager。
  - ◆ 把各种类型的测试汇总在一个可折叠式目录树内,可以在一个目录下查询到所有的测试计划。
  - ◆ Test Plan Manager 还能进一步完善测试设计和以文件形式描述每一个测试步骤,包括对每一项测试、操作顺序、检查点和预期的结果,为每一项测试附属文件,如 Word、Excel、HTML,用于更详尽的记录每次测试计划。
- 人工与自动测试的结合。多数的测试项目需要人工与自动测试结合,包括健全、还原和系统测试。即使符合自动测试要求的工具,在大部分情况下也需要人工操作。启用一个演变性的而非革新性的自动化切换机制,能让测试人员决定哪些重复的人工测试可转变为自动脚本以提高测试速度。TestDirector 还能简化将人工测试切换到自动测试脚本的转换,并可立即启动测试设计过程。
- 安排和执行测试。一旦测试计划建立后,TestDirector 的测试实验室管理为测试日程制订提供一个基于 Web 的框架。其 Smart Scheduler 能根据测试计划中创立的指标对运行着的测试执行监控,能自动分辨是系统还是应用错误,然后将测试切换到网络的其他机器。或当网络上任何一台主机空闲,测试任务会安排到这台主机上,也就是能充分利用时间、机器、网络资源等。使用 Graphic Designer 图表设计,可以很快地将测试分类以满足不同的测试目的,如功能性测试、负载测试、完整性测试等。它的拖动功能可简化设计和排列在多个机器上运行的测试,最终根据设定好的时间、路径或其他测试的成功与否,为序列测试制订执行日程。
- 缺陷管理。TestDirector 的出错管理直接贯穿作用于测试的全过程,从最初发现问题,到修改错误,再到验证修改结果。由于同一项目组中的成员经常分布于不同的地方,TestDirector 基于浏览器的特征,使这些用户可以随时随地查询出错跟踪情况。利用出错管理,测试人员只需进入一个 URL,就可汇报和更新错误,过滤整理错误列表并作趋势分析。
- 图形化和报表输出。TestDirector 常规化的图表和报告帮助对数据信息进行分析,还以标准的 HTML 或 Word 形式提供生成和发送正式测试报告。测试分析数据还可简便地输入到标准化的报告工具,如 Excel、ReportSmith、CrystalReports 和其他类型的第三方工具。

- 和其他工具的集成。TestDirector 可以与 LoadRunner、WinRunner 进行有效的集成, 来统一管理各种测试用例、测试脚本、使用情景与测试结果, 并且可以面向发生问题的部分进行错误跟踪, 达到与开发部门实时交互。用 WinRunner、QuickTest、LoadTest 或 LoadRunner 来自动运行功能性或负载测试, 无论成功与否, 测试信息都会被自动汇集传送到 TestDirector 的数据存储中心。

## 11.6 Compuware 产品的整体解决方案

美国康博 (www.compuware.com) 公司是全球第四大软件公司, 主要从事应用软件开发周期的流程和技术的研究, 开发相应的工具软件, 为全球计算机用户的应用系统提供从开发、集成、测试、运行、管理到维护的全方位保障和服务。

康博的产品中, 除了服务级管理的企业解决方案 Vantage 系列产品、数据管理工具 File-AID 之外, 主要提供以 QACenter (QARun、QALoad 等) 和 SoftICE 为代表的各种软件测试工具产品。

### 11.6.1 Compuware 产品结构

#### 1. 自动化的黑盒测试工具 QACenter

- QARun: 自动的捕获和回放, 为关键的客户/服务器、电子商务到企业资源规划 (ERP) 应用提供企业级的功能测试。
- QALoad: 系统应用性能、负载测试工具。
- TrackRecord: 应用生命周期的变更请求和缺陷跟踪、管理。
- QADirector: 用于分布式应用的高级测试管理。
- WebCheck Web: 站点链接检查、质量分析等测试工具。
- Reconcile: 测试需求管理。
- TestPartner: Web 应用功能测试。
- File-AID/CS
- AHiperstation

#### 2. 自动化的白盒测试工具——Numega 系列产品

- DevPartner Studio: 目前支持 Visual C++、Visual Basic、Java、ASP。
- DriverStudio/SoftICE Driver Suite: 驱动程序开发, 测试工具。
- DBPartner: 数据库开发及测试工具, 目前支持 Sybase、Microsoft SQL Server、Oracle Suite。
- Error Detection: 自动的运行期错误检测和诊断。
- Coverage Analysis: 自动代码覆盖率分析。
- Performance Analysis: 自动性能分析和优化。



- SmartCheck: 自动 Visual Basic 运行期错误检测和诊断。
- FailSafe: 自动 Visual Basic 错误处理和恢复。
- CodeReview: 自动 Visual Basic 源代码分析。
- JCheck: 自动 Java 可视线程和事件分析。
- NuMega DriverStudio: 设备驱动开发。
- DBPartner Debugger: 交互式的存储过程开发、调试和优化。

### 3. Vantage 应用级网络性能监控管理软件

- Application Expert (网络应用性能分析)。
- Server Vantage (服务器数据库性能监控)。
- Network Vantage (网络应用性能监控)。

## 11.6.2 Compuware 分阶段解决方案

在这里,按照软件工程中涉及的需求分析与设计阶段、编码实现阶段、应用测试阶段、应用软件的网路性能测试这 4 个主要阶段来讨论 Compuware 全面解决方案。

### 1. 需求分析与设计阶段

需求分析与设计阶段以测试需求管理工具 Reconcile 为代表,它允许项目团队创建、变更、追踪和报告项目需求。

Reconcile 能与 Microsoft Word、关系型数据库、简单的项目浏览界面结合起来使用,并与该公司的 TrackRecord 和 QADirector 实现无缝集成,以提供需求管理的全面解决方案。它使测试人员可以对项目进行全程监控,从计划、到调试直至开发测试,帮助确保每一个人与项目有联系的工作人员能够及时了解项目的业务、功能、测试需求以及他们之间的关系,以避免引起严重的进度失误和应用失败。

### 2. 编码实现阶段

在这一阶段,Compuware 公司的产品较多,主要有运行期错误检测和诊断工具 Error Detection、代码覆盖率分析工具 Coverage Analysis、性能分析和优化工具 Performance Analysis、设备驱动开发套件 NuMega DriverStudio、Java 可视线程和事件分析工具 JCheck、交互式的存储过程调试和优化工具 DBPartner Debugger 以及有关 Visual Basic 源代码分析、错误诊断、处理和恢复工具 (CodeReview、SmartCheck、FailSafe 等)。

- 运行期错误检测和诊断工具 Error Detection,检测那些由于逻辑错误引起的内存溢出或资源泄露、API、OLE 错误等问题,这些错误一般是不容易被程序员自己发现出来的。通过对被测应用程序的操作,Error Detection 提供清晰的、详细的程序错误分析,自动查明静态的堆栈错误及内存/资源泄露,并能够迅速地定位出错的源代码,即使在没有源代码的情况下也可检查第三方组件的错误。
- 代码覆盖率分析工具 Coverage Analysis,帮助解决在手工测试中总会有一部分代

码功能没有被检测到而影响系统可靠性的问题,因为程序的失败往往可能是由这部分未被检测到的代码造成的。在测试程序时,每完成一次应用会话(session),Coverage Analysis 就能够列出在这次对话中所有函数被调用次数、所占比率等,并可以直接定位到源代码,当然也可以合并多个应用会话来进行检测。Coverage Analysis 能通过衡量和跟踪代码执行及代码稳定性,节省开发时间和改善代码的可靠性。

- 性能分析和优化工具 Performance Analysis, 帮助快速查找性能瓶颈,从而提高整个代码性能。在测试程序时,每完成一次应用 session, Performance 能提供这次对话中函数的调用时间,提供详细的应用程序和组件性能的分析,并自动定位到运行缓慢的代码。这样就能帮助程序员尽快地调整应用程序的性能。
- 设备驱动开发套件 NuMega DriverStudio, 是构成设备驱动程序的开发、调试、测试、调谐以及部署等一整套工具软件,包括久经考验的工具 SoftICE、DriverWorks、VtoolsD 和 DriverAgent 以及基于应用层技术的新的设备驱动程序工具,如新推出的对网络驱动程序开发的工具 DriverNetworks。
- Java 可视线程和事件分析工具 JCheck, 对于 Java 开发人员来说是一个功能强大的图形化的线程和事件分析工具,它提供了一个生动的图形化的方法来表现程序的线程的状态信息以及和 Windows 线程、同步对象、线程组等的交互作用信息,使开发人员能够直观地分析 Java Applet 或 Application。通过这些形象化的图形显示,可以确定 runtime 错误,对执行和逻辑错误进行分析,立刻发现线程问题如死锁、活锁、资源缺乏和系统失败,诊断线程同步和时间选择问题,分析程序执行流程。而后 JCheck 对于那些错误可以定位和显示详细的信息并能定位到源代码。
- 交互式的存储过程调试和优化工具 DBPartner Debugger, 是集成的开发与调试平台,可以创建、测试和管理基于服务器的 Sybase 存储过程和微软 SQL Server 的存储过程,有效地消除 Transact SQL 错误,快速而精确地编写 Transact SQL 应用,浏览和管理数据对象,测试从外部应用中调用的存储过程。
- Visual Basic 源代码分析、错误诊断、处理和恢复工具 (CodeReview、SmartCheck、FailSafe 等)。CodeReview 对应用程序的组件、逻辑、Windows 和 Visual Basic 自身潜在的数百个问题进行严格地源代码检查,包括 Y2K 问题、逻辑错误、应用程序性能和可用性问题、Windows API 调用和标准一致性问题等。CodeReview 系统还提供了两个子模块,一个是 Metrics,通过对 Visual Basic 工程 (vbp) 的执行,计算出代码的长度、复杂度、理解度、语言的使用等级、出错的可能性等数据。另一个是 Namer,它调用一个 Visual Basic 工程,自动并规则地对其中的对象重新命名,并备份原来没有规则命名的工程文件,使开发人员对程序能够有条理地管理。SmartCheck 能很快地查找到隐藏的运行时错误、Windows API 函数错误调用、内存分配以及其他一些重要的程序错误,能够将检测到的错误快速地定位到源代码。使用了 FailSafe,将插入额外的代码对程序进行插装。当程序执行时,FailSafe 通过这些插装的代码捕获、记录执行时程序和系统的重要信息,直接指出错误发生时程序和系统的状态,这些丰富的信息使开发人员能够快速且正确地解决问题。

### 3. 应用测试阶段

在应用测试阶段,主要有自动捕获和回放工具 QARun、应用性能加载工具 QALoad Web 站点质量分析工具 WebCheck、Web 应用功能测试工具 TestPartner、测试管理和设计系统 QADirector 和缺陷跟踪系统 TrackRecord。

- QARun 是功能测试工具,将耗时的测试脚本开发和执行任务变为自动化的过程。其实现的方式是通过鼠标移动、键盘点击操作被测应用,既而得到相应的测试脚本,对该脚本可以进行编辑和调试。在记录的过程中可针对被测应用中所包含的功能点进行基线值的建立,换句话说就是在插入检查点的同时建立期望值。在这里检查点是目标系统的一个特殊方面在一特定点的期望状态。通常,检查点在 QARun 提示目标系统执行一系列事件之后被执行。检查点用于确定实际结果与期望结果是否相同。
- 负载、性能测试工具 QA Load,对于分布式的应用执行有效的负载测试,通过模拟成百或上千的用户执行关键业务,以确定问题所在、优化系统性能,确保应用的成功部署。可以完成下列工作:
  - ◆ 预测系统性能。
  - ◆ 通过重复测试寻找瓶颈问题。
  - ◆ 从控制中心点管理全局负载测试。
  - ◆ 快速创建仿真的负载测试。

QALoad 的测试脚本开发是由捕获会话、转换捕获会话到脚本以及修改和编译脚本一系列的过程组成。一旦脚本编译通过后,使用 QALoad 的组织把脚本分配至测试环境中相应的机器上,驱动多个 play agent 模拟大量用户的并发操作,实施应用的负载测试,减轻了大量的人工工作,提高了效率。

- Web 站点质量分析工具 WebCheck 和功能测试工具 TestPartner。WebCheck 能自动扫描 Web 站点不止 50 个的潜在问题类型并且提供 19 个 HTML 报告。TestPartner 侧重基于 Windows、浏览器应用的一个 Web 自动测试,可以记录应用用户会话,添加适当的确认功能,并且在会话之后重放,确保应用能按期望继续运行。TestPartner 脚本语言是 Visual Basic for Applications(VBA)。它提供 Java、Visual Basic、Visual C++ 开发的应用测试支持,包括 COM 组件、ActiveX 和自动的对象。
- 测试管理和设计系统 QADirector,其分布式的测试能力和多平台支持,能够使开发和测试团队从一个单点跨混合环境控制测试活动,共享测试资源、测试信息和测试结果。QADirector 协调整个测试过程,包括计划和组织测试需求、自动测试和手工测试的管理、浏览和分析测试结果、需求管理、缺陷跟踪系统的集成。
- 缺陷跟踪系统 TrackRecord,能帮助跨整个企业的自动化缺陷跟踪、项目管理和可靠性保证。TrackRecord 使耗时的任务自动化,如文档和报告故障、通信状态和优先权、定位瓶颈。

#### 4. 应用软件的网路性能测试

- 网络应用性能分析工具 **Application Expert**，帮助用户快速发现和解决应用的性能问题。除此之外，能预测网络带宽、延迟、负载和 TCP 端口对用户响应时间的影响和变化。它能发现应用的瓶颈，明确展示应用在网络上各个阶段的运行状态，在线程级分析应用的问题。每个网络线程活动都能以一种很容易理解的模式独立地展示出来，如以甘特图的形式展示每个请求/回应对的变化（包括帧数量、字节数、驻留时间、线程的序列号、间距）。可视化的线程分析让用户看到不同应用请求和响应时间的内部关联结果，它和 **Bounce Diagram** 混合使用让用户快速寻找和分析复杂流量行为，快速定位应用的瓶颈。
- 网络应用性能监控工具 **Network Vantage**，自动发现 1500 多种应用和承载应用的网络的硬件设备、网络应用的流量和流量的拓扑结构，对于用户自己开发的应用提供 15 种定义模式。许多网络的主要问题产生是由于应用性能的降低、流量的增加或网络层设计的错误。**Network Vantage** 可以发现网络的性能，以及运行何种应用、平均反应时间、应用的负载和流量。通过这些信息，可以做到性能优化、服务水平管理、容量设计、资源的计算再利用。
- 服务器数据库性能监控工具 **Server Vantage**，对主机和操作系统的监控，包括磁盘管理、网络、安全控制、文件系统、用户、内存、交换区（SWAP）、打印机、CPU、进程等。更重要的功能是智能化的 Agent，监控数据库系统中关键的资源，一旦资源使用超过设定值，Agent 就会向 **Server Vantage** 主控台发出警报。Agent 可以自动地管理 Oracle、其他数据库及数据库应用，以帮助管理员将关系数据库的运行调谐在最佳可靠性、性能表现的平衡点。**Server Vantage** 也可以采用定制的方式实现对用户自己开发应用的进程、进程的服务端口、应用程序性能的监控。

### 小结

由于手工测试具有一定的局限性，如无法模拟系统运行几十年、无法模拟大量用户等，而引入软件测试自动化。软件测试自动化主要是通过所开发的软件测试工具、脚本（Script）等来实现，具有良好的可操作性、可重复性和高效率等特点，是软件测试中提高测试效率、覆盖率和可靠性等的重要测试手段。

测试工具可以根据不同的测试方法、对象和目的进行分类，如白盒测试工具、黑盒测试工具、单元测试工具、功能测试工具、负载测试工具等。选择测试工具不仅要遵守一定的程序和步骤，而且要注重测试工具的特性，结合自己的实际应用特点，来做出决定。

## 思考题

1. 手工测试和自动化测试有什么主要区别？
2. 手工测试和自动化测试如何进行有效的结合？试举出适当的例子。
3. 作为 Web 应用系统，其主要测试可以通过哪几个工具解决问题？
4. 如果是一家开发 J2EE 企业级系统的软件公司，在 IBM-Rational、Compuware 和 MI 公司中，选择哪一家公司？



## **第 3 部分**

# **软件测试的实践**

# 第 12 章 组织和管理测试团队

建立、组织和管理一支优秀的测试团队是做好软件测试工作的基础，也是最重要工作之一。做测试团队的组织和管理工作，会碰到下列问题：

- 软件测试团队的任务是什么？
- 测试团队在开发中所占的比重有多大？
- 测试团队由哪些角色构成？
- 每个角色的要求、工作范围和责任有什么区别？
- 如何组建一支新的测试团队？怎样确定测试团队的规模？
- 优秀软件测试工程师应具备什么样的素质？
- 怎样发掘、面试和聘用优秀的测试人员？
- 测试人员的职业发展方向在哪里？如何进行培训和引导？
- 什么能够产生对测试人员的激励作用？
- 怎样根据现有人员合理安排工作？

学完本章内容，就可以得到这些问题的答案。

## 12.1 测试团队的地位和责任

在组建测试团队前，首先要清楚测试团队的责任，软件测试是软件测试团队的任务，而不是测试团队的责任。软件测试只是测试团队的一个中心工作，除此之外，围绕这个中心做许多其他的事情，比如网络环境的建立和维护、软件代码的管理等，都是测试团队的事情。一般情况下，要根据软件测试团队的规模和在公司的定位来决定它的工作任务、责任。

### 12.1.1 软件测试团队的任务

软件测试团队的任务是建立测试计划、设计测试用例、执行测试、评估测试结果和递交测试报告等，这些测试任务是最基本的，是测试团队在各个阶段的、自身的主要工作。除了这些主要的任务，测试团队还要完成其他任务，如阅读和审查软件功能说明书、设计文档，审查程序，和开发人员、项目经理等进行充分交流。所有的任务都是为了履行测试团队的责任。那么什么是软件测试团队的责任？

#### 1. 软件测试团队的责任

如果单从软件测试来看，软件测试团队是不能保证产品质量的，也就是说，产品的质



量不是靠测出来的，而是靠产品开发的所有人员（需求分析人员、系统设计人员、程序员、测试人员等）共同努力来获得的。测试人员的基本责任应该是：

- 发现软件程序、系统或产品中所有的问题。
- 尽早地发现问题。
- 督促开发人员尽快地解决程序中的缺陷。

除了这些基本责任，软件测试团队的责任还包括。

- 帮助项目管理人员制订合理的开发计划。
- 对问题进行分析、分类总结和跟踪，以便让项目的管理者和相关的负责人能够对产品当前的质量情况一目了然。
- 帮助改善开发流程，提高产品开发效率。
- 提高程序编写的规范性、易读性、可维护性等。

## 2. 软件测试和质量保证合二为一

对于不少公司，包括微软公司，将软件测试团队和质量保证（QA）团队合在一起，或成为测试小组，或称 QA 部门等，不管名称如何，但它们的责任从测试团队的责任扩张到整个质量保证的领域。作为这样一个团队，具有两个基本职能：软件测试和质量保证。它拥有更多的责任：

- 在产品的整个生命周期，要与项目中相关的部门（市场、设计、开发、产品配置等）合作，负责跟踪和分析产品中的问题，并站在用户的角度，对产品进行全面测试，对不足之处提出质疑。
- 更重要的是对产品开发过程进行跟踪、审查，定义流程并推广流程，及时纠正在执行新的流程所出现的问题，不断改进流程。
- 分析竞争对手的产品，了解自己产品设计的不足，提出改进的意见。

把软件测试和质量保证两项职能结合起来做，工作会更有效。软件测试为质量保证提供数据和质量评判的依据，质量保证可以指导软件测试的进行，质量保证和软件测试相辅相成，质量保证主要审查开发过程、流程，强调质量以预防为主。而测试主要审查产品，包括需求文档、设计文档等，以事后检查为主，保证每个阶段的产品输出是正确的，符合产品质量要求的。

## 3. 测试团队的地位和其他团队的关系

从软件测试团队的任务和责任看，我们就可以知道软件测试的重要性，具有应有的地位。软件的实现有时会遇到一些难题，需要技术攻关解决。从技术上看，软件设计人员和程序员要求高些，但质量始终是产品和企业的生命力，质量是第一的，所以为了保证质量，不要受到过紧的时间日程表和预算的严重影响，软件测试或质量保证人员应具有权威性，也就是具有很高的地位。

也可以通过看开发团队的构成，了解测试团队的地位。在不同的公司中，开发团队的模式存在一些差别，甚至有比较大的区别。如果进行分类，可以概括为三类：

- 以开发为核心，测试只是开发队伍的一部分，也就是开发团队中有测试人员，但

没有形成独立的团队，见图 12-1。

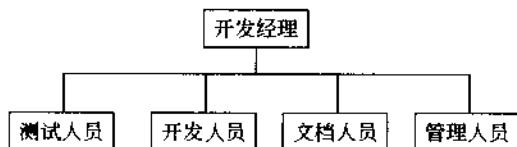


图 12-1 以开发为核心的组织模型

- 以项目经理为核心，开发小组和测试小组并存，隶属于项目经理领导，见图 12-2。

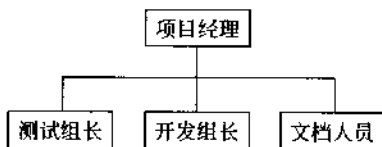


图 12-2 以项目经理为核心的组织模型

- 项目经理、开发经理和测试经理“三国鼎立”，测试团队具有独立的、权威性的地位，见图 12-3。

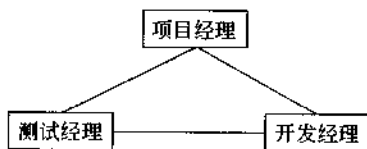


图 12-3 项目经理、测试经理和开发经理三国鼎立的组织模型

其次，测试小组管理具有三个方向：向内、向上和向外。这三个方向也是描述测试团队和其他开发团队之间的关系。

- 向内管理，就是确定测试团队和测试不同岗位的工作职责，招聘团队成员，组织团队的结构，监督和激励员工。
- 向上管理，就是总结测试过程的近况，向上级提交紧急问题以引起项目管理团队关注。设定预期目标，快速而谨慎地应对方向性变化，参加公司级的管理会议，介绍测试团队的业绩和计划等。
- 向外管理，就是分析测试结果，对问题报告进行分类，与同级管理人员讨论测试需求和服务。

### 12.1.2 测试团队的规模

上面分析了软件测试团队在软件开发中的作用和地位，现在来看看测试团队在开发中所占的比重，然后去确定软件测试小组或部门的规模。如果是针对一个项目建立测试小组，规模相对比较容易确定，可以根据测试的范围来评估测试的工作量，然后就可以确定测试小组的人数。对于长期存在的一个测试部门，其规模的确定相对比较困难，要考虑研发部门或工程部门的预算、产品路线图（product roadmap）、项目交叉重叠、项目延迟等各种情

况。一般在考虑各种因素的情况下，还要加上 10%到 20%的富裕量 (buffer)。

测试团队的规模还可以从另一个角度去考虑，即在整个软件开发部门所占的比重，或相对开发人员 (developers, 包括系统设计、程序设计、编写人员。虽然这里的做法是不科学的，开发人员本来应该指所有参加系统或产品开发的技术人员，也就是包括产品规格需求分析、设计人员和测试人员，但目前业界普遍认可了这种错误的用法) 所占的比例。从经验看，不同的应用，软件测试和软件开发人员的比例也是不同的。大致可以分为三类：

- 像操作系统一类的产品，对测试要求最高，测试人员与开发人员的比例为 2:1。因为操作系统功能多，应用复杂，其用户的水平层次千差万别，但同时要求稳定性很强，支持各类硬件，提供各种应用接口，所以测试的工作量非常大。如微软公司参与 Windows 2000 的开发人员是 900 人，而测试人员达 1800 人。
- 像应用平台、支撑系统一类的产品，对测试要求比较高，不仅系统本身要运行在不同的操作系统平台上，还要支持不同的应用接口和应用需求，其比例要低一些，测试人员与开发人员的比例在 1:1 的水平就可以了。
- 对于特定的应用系统一类产品，由于用户对象清楚、范围小，甚至对应用平台或应用环境加以限制，所以测试人员可以再减少些，但测试人员与开发人员的比例至少要保证在 1:2 的水平之上。

软件测试人员的规模主要看产品质量的需求，这个比例应该在上述范围之内，即测试人员与开发人员的比例在 1:2 到 2:1 之间。如果超过这个范围就不合理了。国内不少软件公司都不够规范，测试人员寥寥无几，或者就根本没有全职的专业测试人员，所开发出的产品根本没有质量保证。

## 12.2 测试团队的构成

当组建一个开发团队时，要决定这个团队应该有多少人参加，需要什么技术的人员参加，项目经理是谁？多少个开发人员？多少个测试人员？多少个程序经理？这些问题搞清楚了，开发团队就基本建立起来了。同样，组建一个测试团队，也要了解测试团队的基本构成。

### 12.2.1 测试团队的基本构成

测试团队的构成，从理论上说，和其规模没有多大关系，也就是人们常说的“麻雀虽小，五脏俱全”。如果项目很小，测试小组就一个人，那么这个人就要扮演不同的角色。一般来看，一个比较健全的测试部门应该具有下面这些角色。

- QA/测试经理：人员招聘、培训、管理，资源调配，测试方法改进等。
- 实验室管理人员：设置、配置和维护实验室的测试环境，主要是服务器和网络环境等。
- 内审员：审查流程，并提出改进流程的建议；建立测试文档所需的各种模板，检

查软件缺陷描述及其他测试报告的质量等。

- 测试组长：业务专家，负责项目的管理、测试计划的制订、项目文档的审查、测试用例的设计和审查、任务的安排、和项目经理、开发组长的沟通等。
- 测试设计人员/资深测试工程师：负责产品设计规格说明书的审查、测试用例的设计、技术难题的解决、新人和一般测试人员的培训和指导、实际测试任务的执行。
- 一般（初级）测试工程师：执行测试用例和相关的测试任务。

对于比较大规模的测试团队，测试工程师分为三个层次：初级测试工程师、测试工程师、资深（高级）测试工程师等，同时还设立自动化测试工程师、系统测试工程师和架构（architecture）工程师。

对于规模很小的测试小组，可能没有设置测试经理，只有测试组长，这时测试组长承担测试经理的部分责任，如参加面试工作、资源管理、团队发展等，并且要做内审员的工作，检查软件缺陷描述及其他测试报告的质量等。资深测试工程师不仅要负责设计规格说明书的审查、测试用例的设计等，还要设置测试环境，即承担实验室管理人员的责任。

## 12.2.2 测试人员的责任

在上一节介绍了团队的基本构成，为了更好地理解团队中的每一位成员所起的作用，就需要清楚不同的角色所应该承担的责任。

主要角色的责任，先从一般（初级）测试工程师开始，再介绍资深测试工程师，最后到测试经理。这个过程有利于读者理解他们的责任，测试工程师虽然和初级测试工程师责任不一样，但肯定的是测试工程师能做好所有要求初级测试工程师做好的工作。

不同层次的测试工程师责任有一定的区别，但都是技术工作，主要任务是设计和执行各种测试任务，是测试工作的基础。

下面详细定义了初级测试工程师、测试工程师、资深（高级）测试工程师、测试实验室管理员、软件包构建或发布工程师、测试组长、测试经理等不同测试岗位的责任。

### 1. 初级测试工程师

初级测试工程师的责任比较简单，还不具备完全独立的工作能力，需要测试工程师或资深测试工程师的指导，要求比较低，主要有下列7项责任。

- 了解和熟悉产品的功能、特性等。
- 验证产品在功能、界面上是否和产品规格说明书一致。
- 按照要求，执行测试用例，进行功能测试、验收测试等，并能发现所暴露的问题。
- 清楚地描述所出现的软件问题。
- 使用简单的测试工具。
- 努力学习新技术和软件工程方法，不断提高自己的专业水平。
- 接受测试工程师的指导，执行主管所交待的其他工作。

### 2. 测试工程师

测试工程师的责任相对多些，熟悉测试流程、测试方法和技术，参与自动化测试，具

有独立的工作能力，但基本已执行测试为主，主要责任如下。

- 熟悉产品的功能、特性，审查产品规格说明书。
- 验证产品是否满足了规格说明书所描述的需求。
- 根据需求文档或设计文档，可以设计功能方面的测试用例。
- 根据测试用例，执行各种测试，发现所暴露的问题。
- 全面使用测试工具，包括测试脚本的编写。
- 安装、设置简单的系统测试环境。
- 报告所发现的软件缺陷，审查软件缺陷，跟踪缺陷修改的情况，直到缺陷关闭。
- 写测试报告。
- 负责对初级测试工程师的指导，执行主管所交待的其他工作。

### 3. 资深测试工程师

资深测试工程师不仅具有良好的技术、产品分析能力、解决问题能力、丰富的测试工作经验，而且有较好的编程、自动化测试经验，熟悉测试流程、测试方法和技术，解决 QA 工作中可能遇到的各种技术问题。主要责任如下。

- 负责系统一个或多个模块的测试工作。
- 制订某个模块或某个阶段的测试计划、测试策略。
- 设计自动化测试框架或结构，开发测试脚本、必要的测试工具。
- 设计测试环境所需的系统或网络结构，安装、设置复杂的系统测试环境。
- 熟悉产品的功能、特性，审查产品规格说明书，并提出改进意见。
- 审查系统、程序设计说明书，提出对系统模块设计的改进要求。
- 审查代码。
- 验证产品是否满足了规格说明书所描述的需求。
- 根据需求文档或设计文档，设计复杂的测试用例。
- 执行那些边界的、特殊的、技术难度高的测试用例。
- 负责对测试工程师的指导，执行主管所交待的其他工作。

### 4. 测试实验室管理员

测试实验室管理员主要负责建立、设置和维护测试环境，保证测试环境的稳定运行。其主要责任如下。

- 负责测试环境所需的网络规划和建设，维护网络的正常运行。
- 建立、设置和维护测试环境所需的应用服务器或软件平台。
- 对实验室的硬件、软件资源进行登记、分配和管理。
- 申请所需求的、新的硬件资源、软件资源，协助有关部门进行采购、验收。
- 对使用实验室的硬件、软件资源的权限进行设计、设置，保证其安全性。
- 安装新的测试平台、被测试的系统等。
- 优化测试环境，提高测试环境中网络、服务器和其他设备运行的性能。

### 5. 软件包构建或发布工程师 (release engineer)

发布工程师在 QA 工作中起着很重要的角色, 负责测试产品的上传、打包和发布。其主要责任是。

- 负责源程序代码管理系统 (如 CVS、SourceSafe 等) 的建立、管理和维护。
- 制订 Check in/Check out 等相关的源代码控制规则。
- 文件名定义规范, 建立合理的程序文件结构和存储目录结构。
- 为程序的编译、连接等软件包构造建立自动处理文件。
- 保证测试最新的产品包 (Code、Image 等) 上传到相应的服务器上, 并确认各模块或组件之间相互匹配。
- 每天为各项目新的或修改的代码重新构造新的软件包。确保不含病毒, 不缺图片和各种文件。
- 软件包的接收、发送、存储和备份等。

### 6. 测试组长

测试组长一般具备资深测试工程师的能力和经历, 可能在技术上相对弱些, 不是小组内最强的, 其责任偏重测试项目的计划、跟踪和管理, 同时负责测试小组的团队管理和开展。其主要责任如下。

- 测试小组的管理或参与测试团队的管理。
- 负责一个独立的测试项目。
- 制订整个项目的测试计划、测试策略, 包括风险评估、日程表安排等。
- 熟悉产品的功能、特性, 审查产品规格说明书, 并提出改进意见。
- 审查系统、程序设计说明书。
- 验证产品是否满足了规格说明书所描述的需求。
- 实施软件测试, 并对软件问题进行跟踪分析和报告, 推动测试中发现的问题及时合理地解决。
- 编写项目的整体测试报告, 保证产品质量。
- 对竞争者的产品进行分析, 提出对软件的进一步改进的要求, 并且评估改进方案是否合理。
- 负责测试项目内部的资源、任务安排。
- 监督测试流程的执行, 并将执行过程中所发现的问题反馈给测试经理或项目经理。
- 为团队成员提供技术指导, 协助主管或测试经理工作。

### 7. 测试或 QA 经理

测试或 QA 经理的主要工作在团队、资源和项目等的管理上, 不同于测试组长。测试组长主要集中在项目管理上, 一般不负责测试人员的招聘、流程定义等管理工作, 而且偏重技术。测试或 QA 经理对产品的质量负全面责任, 有责任向公司最高管理层反映软件开发过程中管理问题或产品中的质量问题, 使公司能全面掌握生产和质量状况。其主要责任

如下。

- 负责整个测试团队或部门的管理，包括测试岗位责任的定义，组织团队结构的建立和优化，团队的建设和发展，培训活动的组织，员工的激励等。
- 负责一个完整产品的软件测试和质量保证等工作，包括项目组长的指定、项目资源的安排、项目进度的跟踪、项目审查和总结等。
- 测试部门年度/季度计划、预算的编写、实施和评估。
- 促进质量文化的普及，使整个开发团队的每位成员都有一个正确的客户和质量观念。
- 在测试人员的招聘、考核等方面的工作，协助人力资源部门。
- 定义、实施软件测试流程或整个开发周期流程，并收集、处理流程实施中所存在问题，最终不断改进流程。
- 审查项目的测试计划、测试策略等，包括资源调度和平衡、风险评估等。
- 和其他部门协调，参加多方会议解决产品规格说明、设计等问题。
- 审查系统、程序设计说明书。
- 实施软件测试，并对软件问题进行跟踪分析和报告，推动测试中发现问题及时合理地解决。
- 审查项目的测试报告，进行产品质量的分析，提交质量分析报告。
- 对竞争者的产品进行分析，提出对软件的进一步改进的要求并且评估改进方案是否合理。

这是一个微软测试工程师的一天工作，作为例子来说明测试工程师的责任。

- (1) 产品编译必须在此之前完成。
- (2) 每日凌晨 3 时，测试编译自动开始。
- (3) 如果测试编译成功，BVT (basic verification test) 测试自动开始。
- (4) 每天测试工程师先检查报告 Test Build 与 BVT 结果的 E-mail。
- (5) 如果有 BVT 错误，在第一时间分析原因，隔离错误代码并汇报 0 级缺陷 (Priority 0 bug, 最高级别的缺陷)。
- (6) 开发团队对于 0 级缺陷应于当日之内修改完毕。
- (7) 测试工程师接着用 Product Studio 检查 bug 情况，验证分配给自己的、已修改合格的 bug。
- (8) 关闭 bug 并增加针对此 bug 的 Regression Test。
- (9) 验证最近的 Lab Run 结果。
- (10) 如果其中有新的错误，隔离并汇报新 bug。
- (11) 开发新的测试 Spec 与新的测试代码。
- (12) 使用个人 Private Run 来验证新开发的测试程序。
- (13) 使用个人 Private Run 来验证开发伙伴新开发的产品程序没有重大错误。
- (14) 改进与提高自动化测试系统的功能。
- (15) 参与 Spec, Test Spec Review 会议。
- (16) 做测试同伴测试代码 Review, UE 帮助文件 Review。

(17) 回答内外 Newsgroup 的问题。

### 12.2.3 测试团队的组织模型

测试团队的组织直接关系到测试团队的工作效率和生产力，其组织的方式由测试团队规模和具体任务、技术等决定。对于小的测试团队所要求的结构很简单，一般以项目组织就可以了。对于大型测试团队的组织，一层结构就难以满足管理的要求，有必要构造 2~3 层的组织结构。对于这种多层结构，可以归纳为两种基本类型：

- 从测试所采用的技术角度去组织，把整个测试团队划分为不同技术的下层结构组织。当拿到一个项目后，将其分解为不同技术的模块，由不同的第二层结构组织去执行，如图 12-4 所示。这种结构的优势在技术上可以进行充分交流，即技术共享性比较好，有利于技术的发展和深入，对于那些技术深、产品单一的软件公司比较合适。

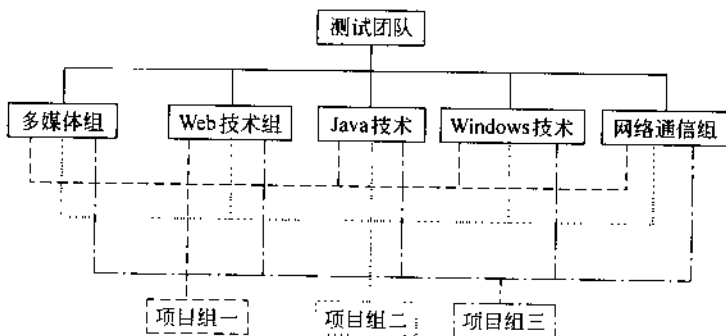


图 12-4 按技术领域来组建团队模型示意图

- 从产品线去组织，将整个测试团队按照公司的不同产品进行划分，形成下层结构组织。任何一个产品的开发工作，都是在第二层结构组织内部进行，而一个产品往往是由多个项目，这些项目再一次被分配到第二层下面结构组织内的第三层组织，如图 12-5 所示。这种结构的优势有利于产品各个模块的协调、集成，接口测试会比较充分，产品功能特性理解比较深，有利于产品的开发，但不利于技术的深入。对于那些产品比较多、公司规模比较大的软件测试团队比较合适。

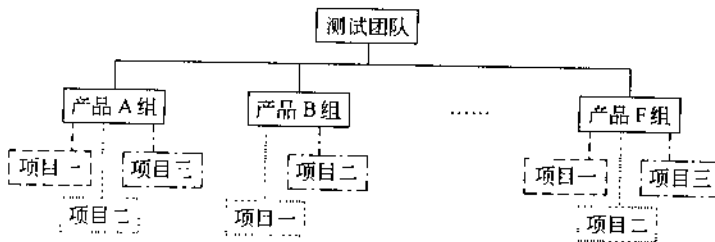


图 12-5 按产品线来组建团队模型示意图



## 12.3 如何从零开始

前面已经介绍了软件测试团队的任务和责任、测试团队的规模、测试团队的组织模型、测试团队的基本构成等，还定义了测试团队中不同岗位人员（初级测试工程师、测试工程师、资深/高级测试工程师、测试实验室管理员、软件包构建或发布工程师、测试组长、测试经理）的责任。有了这些定义和描述，就可以开始组建测试队伍。这一节就假定从零开始组建一支新的测试团队，主要从以下五个方面进行讨论：

- 建立测试团队的正确观念。
- 测试人员的招聘。
- 测试人员的培训。
- 组建测试团队的三个阶段。
- 测试流程的建立和完善。

### 12.3.1 建立测试团队的正确观念

在组建测试团队之前，要建立正确的观点，这些观点会对测试人员招聘、激励和测试的效率、测试工作的质量、结果的稳定性等都会有积极的影响。

#### 1. 测试人员的要求

不少计算机软件业界人士对软件测试人员要求比较低，认为他们只要会操作计算机，有一定的软件使用经验就可以了。这些人认为，软件测试人员只要一步一步操作所要测试的软件，就能发现程序中的问题；或者对照软件产品规格设计说明书，通过比较就容易发现两者不一致的地方，这些都不需要什么技术。

这种想法是错误的，软件测试的方法不只是功能测试，还有难度相对大得多的系统测试，包括系统的安全性、可靠性、稳定性和兼容性测试等；其方法也不只是一种黑盒测试方法，还有白盒测试方法，测试人员需要编程经验，对多数编程语言（C/C++、Java、ASP、PHP等）有一定的了解，分析系统构造、实现的原理。

对软件测试人员的要求和对程序员的要求是不一样的，测试人员的技术要求总体上说会低些，但测试人员在沟通能力、理解能力、分析问题能力等方面要求会高些。

对不同层次的测试人员的要求也不相同，对数据库测试工程师需要数据库设计、数据库管理方面的经验，对自动化测试工程师需要良好的编程经验，对测试组长除了需要良好的编程经验、测试经验之外，还需要良好的项目管理能力和组织能力等。

#### 2. 测试职业的地位

国内社会普遍存在一种错误的倾向，认为软件测试人员的地位低。在软件公司，开发人员是一等公民，测试人员是二等公民。这种错误偏见产生的根源有两个方面，一方面就是上面提到的，错误地认为对软件测试人员要求比较低、容易招；另一方面，由于历史的

原因,国内绝大多数软件公司是最终客户开发应用系统,是做软件项目,而不是做产品,所以强调市场开拓和公关能力所带来的合同,以及如何把合同中的系统功能开发出来。即强调实现的能力,所以质量的问题没有被认为是影响企业生存和发展的关键因素。这种偏见同样对薪水体系的建立也会产生影响,测试人员所拿到工资的平均水平可能是开发人员的三分之二,内资企业可能会更低些。

可喜的是这种情况正在发生变化,人们开始越来越认识到软件质量的重要性,越来越多的软件测试职位等待着对软件测试有兴趣的工程师、大学毕业生的加入。这种变化主要有以下几个方面的原因:

- 中国加入 WTO 后,国际软件公司纷纷在我国建立软件研发基地,由于受到设计能力、知识产权保护等影响,这些基地主要的任务以测试为主。
- 国内越来越多的软件公司开始做产品,因为做项目很累,企业发展很慢。一旦做软件产品,自然软件质量就变得很重要了,对软件测试的投入就会加大。
- 软件产业从一个高科技领域,开始慢慢走向成熟行业。对于一个成熟的行业,企业最终的竞争不在技术上,而是在质量和服务上。

通过举几个例子,可以更好地说明这种变化:

(1) IBM 在 1994 年就开始在中国建立开发中心,刚开始聚焦在软件开发项目上,结果到 1998 年才发展到 100 多人。1999 年,IBM 开发中心开始转型,将软件开发分拆出去成立 IBM 研发中心(CRL),其他部分成立开发中心(CDL),业务中心同时转移到软件测试上来,从国外来的外包项目 90% 以上都是软件测试项目,结果不到三年就发展到近 1000 人。CDL 计划要发展到 5000 人。

(2) Microsoft 中国开发中心,为了兑现与中国政府的软件合作协议,成立了“战略合作部”来推动中国软件外包产业,一开始就很明确地将方向只定位在软件测试上,Microsoft 在美国本土有 15 000 人的软件测试人员,希望要将一半的工作移出美国,其中 40% 会移到中国,预计增加软件测试工程师的位置多达 3000 个。

(3) CitiBank 旗下的软件公司 CitiSoft 在中国成立开发中心,短短一年发展到 200 人,三年就发展到 1600 人,其核心业务也是软件测试。

### 3. 测试职业的发展空间

有些人错误认为软件测试人员的发展空间比较小,如果做了测试,就学不到技术,结果可能一辈子都只好做软件测试。实际情况不是这样的,软件测试更强调流程,对整个软件开发过程的每一步都要进行跟踪、审查,软件测试更强调交流和沟通,要及时解决问题、处理冲突。所有这些,对软件测试工程师向 QA 经理、项目经理、软件企业高层管理人员等方向发展都会有帮助。

### 4. 测试职业的教育和培训

在传统的理工科大学中,一般都设置了程序设计、高级语言(C/C++、Java)编写程序、算法设计等课程,但几乎没有开设与软件测试相关的课程,所以软件测试人员的资源相对程序员要少得多。不过,这种情况也在变化,目前全国 35 所著名大学的软件学院都相继开

设软件测试相关课程或相关专业，有些大学也开始设置了软件测试的硕士生点、博士生点等。

## 12.3.2 测试人员的招聘

如果测试实验室新来了一个不合适的测试人员，那么其他员工将不得不为这位新人忍受痛苦，除了培训之外，这个小组中的人不得不弥补这位新人马虎所带来的过多错误。此外，如果一个人的工作表现较差，会使整个测试小组给人一种无能和粗心的感觉，结果导致经理们轻视整个小组的测试结果、报告和其他测试数据。更大问题是，没有做出贡献的人通常自己也知道这种状况，并且采用各种防御性手段来掩饰这些失败，例如诋毁其他测试人员的工作成果。在这种情况下，那些积极做出贡献的人将会忍受委屈，其主动性、积极性将受到严重影响。在这两种情况下，有能力的测试人员宁可辞职，也不愿意和那些令人讨厌的、无能的人一起工作。所有这些结果，对那个测试小组都是不公平的，结果也和我们的愿望相反，原来招一个新人是补充资源、增加团队的能力，结果是新人进来了，老员工却离开了，人力资源没有增加，反而减少。

与此相反，如果招聘一个正确的人，那么在工作业绩、效率和信誉等方面，就能够给测试小组带来巨大的收获，用仔细挑选的人员来组建测试小组，这个小组作为一个整体，比人员的简单累加具有更强的能力。

人力资源部可能只关注公司的人事程序，通过对一个人进行批评、沟通，甚至劝退、辞退，来完成本职工作。如果作为一个团队的优秀经理，则会关心自己的团队和在这个团队工作的每个成员，工作不仅仅停留在程序上，而是使他们工作快乐，包括新雇用的人。

所以说，人员招聘很重要。要做好测试人员的招聘，就要对测试团队的任务和责任、测试团队的构成、测试人员的责任有一个很好的了解，并帮助各层管理人员、人事部门和应聘者建立对软件测试职业的正确观念等。

### 1. 软件测试人才市场

总体上说，国内软件测试人才非常缺乏，有经验的、专业的测试工程师很难招到。由于国内软件测试人才市场的现状，软件测试工程师的招聘对象，就不得不做些调整，目标锁定在是有一定技术基础、个人素质和个性方面有潜力，能够培养成为测试工程师的人员，有测试工作经验只能作为优选项，而不能作为必要条件。为此，在招聘时着重考察应聘者是否具有过硬的技术基础、良好的个人素质，包括是否具有团队协作精神、自我学习能力、逻辑思维能力和表达沟通能力等。作为优秀的测试工程师，必须具备的素质主要包括以下五个方面：

- 出色的沟通协调能力。由于测试工作本身的一个重要任务就是找出程序、系统中的缺陷，可以说是挑毛病，不会让开发者快乐，所以交流、沟通能力就很重要，这样才能将与相关人员之间可能发生的冲突和对抗减少到最低程度。在与软件产品打交道的过程中，测试工程师比其他部门的人参与的项目更多，必须能够和技术开发人员（设计、编程人员）以及非技术人员（包括客户、用户、管理和培训

人员等)进行良好沟通和交流。

- 扎实的技术功底。虽然功能测试这样的工作技术要求低,但测试工作仍然是一项技术性很强的工作,在进行单元测试、集成测试、系统测试时,测试人员必须明白被测软件系统的实现原理、方法以及涉及的各种系统平台、技术等,同时还要进行测试脚本的编写、测试工具软件的开发等。而且拥有编程或开发经验的测试人员会对软件开发过程有更深入的理解,对提高软件质量是很有帮助的。
- 必要的个人特性。除了沟通能力和技术水平之外,耐心、细心、执著、怀疑一切的态度、敏锐的洞察能力、否定性的创造力等基本素质,对做好软件测试能起到不可低估的作用。测试人员以高度的责任感,用客户的眼光进行评估。本着对质量一丝不苟的追求,不放过任何一个可能存在的疑点,能够对整个产品的操作与使用保持一种“系统”的眼光,并充分关注细节,能经受得住坏消息而保持质量目标,只有这样,才能保证测试工作的有效性和可靠性。
- 经验和技能。做软件测试工作,需要对产品有着很深的理解,能知用户所想、用户所需,站在用户的角度去审视软件、测试软件,提出产品改进的建议,所有这一切都建立在已有的工作经验之上。同时,需要一些非技术的良好技能,包括分析能力、写作能力、组织能力等。测试过程中需要考虑可能的变化因素、处理所出现的问题、控制这个测试流程的有序进行,保持有效性、灵活性和条理性,随时要把产品测试过程中的计划、质量问题等写成文件、报告。
- 态度。除此之外,需要理解和采取适当的态度去做软件测试,进行软件测试需要很多人的眼光要进行一百八十度的转变,因为测试的目标是要让被测软件失败,边界意味着被超越而不是被遵从,一个成功的测试揭示一个缺陷。软件测试依赖于已建立的软件工程的思想、方法和技术,所以做软件测试人员需要对软件工程有一个科学的、公正的态度。

## 2. 明确各类测试人员的要求

根据测试人员不同岗位的责任,就比较容易确定对不同岗位的测试人员的要求。在此,列出对初级测试工程师、测试工程师、资深(高级)测试工程师的一般要求,有助于找到所需的、合适的人才。

- 初级测试工程师
  - ◆ 计算机、数学、物理及相关专业,大专以上学历。
  - ◆ 有责任感、诚恳、工作认真。
  - ◆ 可以胜任重复性工作,工作细致,有耐心。
  - ◆ 思路清晰,良好的思维能力。
  - ◆ 1年以上工作经验。
  - ◆ 熟悉计算机操作与软件配置。
  - ◆ 具有良好的人际沟通及语言表达能力。
  - ◆ 注重团队精神,有良好的合作意识。
  - ◆ 英语读写熟练,英语达到四级以上。

- ◆ 具备软件工程的基本知识。
- ◆ 有志于软件测试工作。
- 测试工程师
  - ◆ 计算机、数学、物理及相关专业，本科以上学历。
  - ◆ 高度的责任感，积极的工作态度。
  - ◆ 熟悉软件测试流程，具有 1 年以上软件测试的工作经验。
  - ◆ 熟悉某一种操作系统平台（Windows/UNIX/Mac）和应用平台。
  - ◆ 掌握 HTML/XML 和 JavaScript，并且有初步的 Visual Basic、C/C++ 或 Java 编程经验。
  - ◆ 能够使用一种或两种以上的、目前流行的软件测试工具。
  - ◆ 拥有较好的沟通技巧，良好的表达能力和应变能力。
  - ◆ 具有良好的反向思维和发散思维能力。
  - ◆ 热爱软件测试工作，具有较强的团队合作精神。
  - ◆ 英语读写熟练，并具有一定的英语口语表达能力，英语达到四级以上。
  - ◆ 对软件工程、软件开发流程等有很好的认识。
  - ◆ 有较强的逻辑分析能力和学习能力，良好的文档撰写能力。
  - ◆ 心理素质良好。
- 资深（高级）测试工程师
  - ◆ 计算机或相关专业大学本科或以上学历。
  - ◆ 熟悉软件测试。
  - ◆ 具有 3 年以上软件测试的工作经验。
  - ◆ 具有一年的编程经验，熟悉 C/C++ 或 Java 编程语言。
  - ◆ 具有在数据库（Oracle、DB2 或 Microsoft SQL）方面良好的应用经验。
  - ◆ 具有某一个应用领域的知识和应用背景。
  - ◆ 精通软件测试理论和方法，能够熟练应用多种测试工具。
  - ◆ 具有测试项目管理经验，能独立承担软件测试设计、项目组织等能力。
  - ◆ 工作作风严谨，具有很强的责任心和敬业心，积极的进取精神。
  - ◆ 善于团队合作，具有很强的沟通、协调能力。
  - ◆ 富有创新、怀疑精神和质量意识。
  - ◆ 独立分析问题、解决问题的能力，具备较强的总结能力。
  - ◆ 英语熟练，具有良好的英语表达能力，英语达到六级以上。
  - ◆ 对软件工程、软件开发流程等有很好的认识。
  - ◆ 熟悉软件质量管理流程，熟悉 ISO9001 及 CMM。

### 3. 面试

测试人员岗位责任和要求明确了，就可以进军 IT 人才市场，向市场推荐公司，树立公司的良好品牌和形象，宣传公司的企业文化，宣传软件测试的职业发展空间、待遇和机遇等，寻找测试团队所需要的人才。

在招聘过程中,要收集和筛选应聘人员的简历,要进行现场的沟通、笔试等,但最重要的是面试。面试是招聘人员的最重要环节,主要理由有:

- 挑选员工的第一标准是应聘者是否具有良好的品德,良好的品质是职业道德的基础。判断应聘者是否具有良好的品德,只有通过面试。
- 简历可以告知应聘者的基本情况,有些重要的细节不能从中得到。简历中还常含有夸大其词的内容,如刚毕业或工作不到一年的应聘者,在其简历中写有类似于“精通C语言编程、精通数据库设计”的内容,这里的“精通”对不同层次人员的理解可能是完全不一样的。其次,少数简历还存在欺骗性或隐藏性。
- 通过笔试只能了解应聘人员目前掌握的理论知识和基本技能,难考察其实际工作经验、动手能力和思维能力。
- 在面试的过程中,可以通过与应聘者讨论一些问题,来判断应聘者有没有团队精神、责任心及工作热情、创新精神。
- 只有通过面试,才可以了解应聘者独立工作能力如何,解决问题的能力如何,在面对困难时是退缩还是勇往直前,因为实际工作中常有新问题需要解决。
- 要测试应聘者有没有快速学习的能力,日新月异的技术发展要求有这种能力。

根据有关分析,国内人才较普遍具有的优点,比如有雄心壮志、理论基础,编程能力强,能吃苦耐劳,讲纪律,讲服从等,但缺点也比较突出,如创新精神不足,做事缺乏主动,独立工作能力弱,缺乏诚实、不够直率,对很多事没有主见,不善于与人交流,缺乏合作精神等。通过面试,以确认应聘者在这些方面是否有比较明显的缺点,即使对优点也要认真审视。专家曾告诫,不要招聘那些有雄心壮志的人作为测试工程师,许多工作就是一种工作,并不是为了实现某些伟大目标的必经途径或过程。

面试过程,可以归纳为以下7个基本的步骤:

- (1) 收集简历,并按照工作岗位描述(责任、要求、薪水范围等)来筛选简历。
- (2) 参加基本的笔试,了解其基本技术水平、相关知识程度、专业能力以及外语水平。
- (3) 人力资源部经理或相当职位的人,向应聘者介绍公司的历史、现状和发展方向、薪水、福利待遇、培训机会和企业文化等,同时了解应聘者的职业道德、工作态度、简历的真实程度等。
- (4) 测试小组中成员,包括测试工程师和测试组长,通过面试,着重考察应聘者工作经验、技术深度、实际能力等,以了解应聘者的能力和特性是否适合目前的工作岗位。
- (5) 测试经理会做一个全面的测试,着重考察应聘者的职业观念、理解能力、思维能力、潜在的发展素质等,决定建议是否录用。
- (6) 其他部门经理或更高位置的经理,可能需要对应聘者进行面试,考察应聘者的合作态度、技术范围的广度、沟通表达能力等。
- (7) 人力资源部经理针对公司所能提供的待遇、人事关系处理等方面,和应聘者进一步沟通和确认。

面试过程有多种形式,如笔试、面谈、电话交谈、在咖啡馆轻松交谈等。面谈是一种主要形式,一对一、多对一都可以采用,经常是先高度概括地介绍一下项目,然后不断向应聘者提问,来了解各方面情况,而且要快速提问,以考察应聘者的反应速度、真诚度、

思维能力等。最后，可以留几分钟让应聘者提问，帮助应聘者澄清必要的疑问，同时也考察了应聘者的主动性、合作精神、创造性等。

面试提问的技巧很多，而且提哪些问题也很重要，这里简单列一下经常要提的问题：

- 为什么想离开目前的工作？
- 你曾听说过我们公司吗？你对于本公司的第一印象如何？
- 目前的工作上，你觉得比较困难的部分在哪里？
- 如果我们的竞争对手也有意录用你，你的态度如何？
- 你觉得自己最大的弱点（缺点）是什么？
- 你找工作时最在乎的是什么？请谈一下你心中理想的工作是什么？
- 请谈谈在工作时曾经令你感到十分沮丧的一次经历。
- 你最近找工作时曾面谈过哪些工作？应征什么职位？结果如何？
- 谈谈你最近阅读的一本书或杂志。
- 你通常从事什么样的休闲活动？
- 对于目前的工作，你觉得最不喜欢的地方是什么？
- 如果你有机会重新选择，你会选择不一样的工作领域吗？
- 你觉得作为软件测试工程师，你的优势有哪些？
- 如果你进入本公司，对于这项职务以及这个部门，你打算做什么样的改变？
- 在执行最后一次测试时，你期望它通过还是通不过？为什么？
- 讲一讲在与项目小组的其他人员讨论近来的错误报告时，你的方式和语气。
- 谈论一下，你提交并存在争议的错误报告。
- 你觉得在软件测试上获得成功需要什么样的特质及能力？
- 你如何规划未来，你认为5年后能达到什么样的成就？
- 你认为“成功”的定义是什么？
- 谈谈你对加班的看法。
- 在你过去的工作经验中，曾遇到什么样的难题？你如何克服它？
- 什么样的管理风格是你所欣赏的？

#### 4. 对应聘者可以进行量化的评估

图12-6是Speedywriter测试小组进行评估的量化指标，就像所看到的那样，最右面的那一列定义了重要的技能，可将它分为八个方面进行考察：教育、工作经验、职业水准、测试技能、计划能力、设计和开发能力、软件配置、测试执行。紧跟着的三列定义了测试小组中的工作岗位所期望和所需要的经历。中间的五列定义了测试小组成员在这些技术中所处于的等级，从不了解（“0”）到专家级（“3”）。最左面的两列是小组中每个技能和每类技能的最小等级和平均等级，通过它可以确定技能的相对强项和弱项。

如果到了筛选的最后阶段，应该能够确定这个人是否符合工作要求的基本条件。同时，对于是否喜欢这个应聘者，是否适合测试小组，以及是否应该聘用，也应该有一个基本的感觉。如果这些问题中，任何一个的回答是“否”，那么就应该把这份简历放在一旁，需要进一步讨论、考察。最后要提醒的是，在紧张的项目过程中，不要面试不可能对项目有积

软件自助测试小组技能评估和管理工作表

	Legend			0: 不了解		1: 有些了解		2: 了解		3: 专家级	
	R: 必须			D: 期望							
	T1=测试技术人员			TM 测试经理		MTE 人工测试工程师		ATE 自动测试工程师			
技能和资格	Minimum Ratings	ATE Minimum Ratings	MTE Minimum Ratings	TM Jack	MTE: Late-Isu	ATE Bob	MTE: Huesh	STU Maria	Team Minimum	STeam Average	
普通资格											
教育											
理学学士 (或以上)	D	D	D	BS(CSE)	Ph.D.(CS)	BS(Math)	MA(Psych)	BS(Bus)			
测试培训或认证	D	D	D	CISOE	ISEB						
其他							LFC	CPA			
工作经验 (年)											
测试角色	D	5R	5R	7	5	6	11	12			
非测试、计算机	D	D	D	3	2			4			
非计算机、域	D	D	D								
非计算机、非域							10	6			
所有/任何/其他	1D	5R	5R	10	7	6	21	22			
职业水准											
口头交流	1R	2R	2R	3	1	2	3	2	1	2.2	
非正式书面交流	1R	3R	3R	3	3	2	3	3	2	2.8	
正式书面交流	D	D	D	3	0	1	3	1	0	1.6	
继续教育	D	R	R	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
测试小组的构造/交叉培训	D	2R	2R	3	2	1	3	2	1	2.2	
交叉功能关系的构造	D	2R	2R	3	2	1	3	2	1	2.2	
阅读 (记忆力、推理和分析能力)	1R	2R	2R	3	2	2	3	2	2	2.4	
业务/技术趋势 (杂志的阅读)	D	1R	1R	3	1	3	1	1	1	1.8	
测试技能											
普通											
测试标准	D	2R	2R	3	3	3	3	3	3	3.0	
软件开发生命周期	D	2R	2R	3	3	3	3	2	2	2.8	
测试/开发过程/成熟度	D	1R	1R	2	2	3	1	1	1	1.8	
变更管理	D	1R	1R	2	2	3	1	1	1	1.8	
涉及业务的测试/同步数据线路控制	D	1R	1R	3	3	3	2	1	1	2.4	
计划											
估计		D	D	3	1	1	2	1	1	1.6	
文档		D	D	3	1	3	2	1	1	2.0	
质量成本		D	D	3	2	2	1	1	1	1.8	
质量风险/失效模式和效果分析		D	D	3	2	1	1	1	1	1.6	
质量风险分析和管理		D	D	3	2	1	1	1	1	1.6	
设计和开发											
行为 (黑盒)	D	2R	2R	2	3	3	3	2	2	2.6	
结构 (白盒)	D	D	1R	1	3	2	1	2	1	1.8	
静态 (需求、规格说明、文档)	D	D	2R	2	3	1	3	2	1	2.2	
可靠性 (统计)		2R	D	1	1	3	1	2	1	1.6	
执行 (建模/模拟/测试)		2R	D	1	2	3	1	3	1	2.0	
代码/数据流/覆盖率		2R	2R	2	3	1	3	3	1	2.4	
质量风险/需求覆盖率(跟踪能力)		1R	2R	3	2	3	1	1	1	2.0	
自动化 (开发)											
检测设备执行 (Silk 等)		3R	D	1	1	3	1	3	1	1.8	
检测设备测试管理		D	D	3	1	2	1	1	1	1.6	
客户工具管理		3R	D	3	1	3	1	3	1	2.2	
配置											
测试数据生成		1R	D	1	1	2	1	2	1	1.4	
版本控制		1R	1R	2	2	2	1	3	1	2.0	
配置管理		D	1R	1	2	2	1	1	1	1.4	
集成测试		D	1R	3	2	2	1	1	1	1.8	
执行											
人工脚本	D	D	3R	3	3	1	3	1	1	2.2	
人工探测	D	D	3R	3	3	1	3	1	1	2.2	
自动		3R	3R	1	1	3	1	3	1	1.8	
错误频率	D	3R	3R	3	3	3	3	3	3	3.0	
错误报告	D	3R	3R	3	3	3	3	3	3	3.0	
测试状态报告	D	2R	1R	3	2	3	3	2	2	2.6	
测试尺度 (监控板)	D	1R	1R	3	2	3	1	3	1	2.4	
平均测试技能				2.4	2.1	2.3	1.7	1.9	1.3	2.1	

图 12-6 技能评估工作表第一页



极贡献的人。否则,后果轻则是浪费了时间,重则是在用人方面犯下错误,由于资源紧张,可能在面试时,降低了用人标准,这可能会导致浪费掉整整数月的时间,损害测试小组的信誉,还降低测试小组的生产效率。

### 12.3.3 测试新人的培训

正如上一节所述,由于国内软件测试人才缺乏,在招聘软件测试工程师时不得不做些策略上的调整,目标不能锁定在有测试工作经验的工程师上,而锁定在那些有一定技术基础、个人素质和个性方面有潜力能够培养成为测试工程师的人员。这样在招聘之后,培训就成为一个重要的任务。

一般情况下,通常要为新到的测试人员分配指导人员。指导人员将为新人指明工作方向,并且在前几周的工作中,审查他们的工作。的确,这可能会降低生产率,但是,它也能够使测试小组的产品有统一的质量水平,并且这还向每个员工表明:这位新人是测试小组中重要的成员,任何一位成员的成功对整个小组的成功都将起到重要作用。

- 培训目标:新员工须达到 QA 人员上岗基本要求,基本了解公司产品,掌握 QA 的基本知识,熟悉工作流程及相应规则。
- 培训的方式:是由正式的课堂培训 (formal class training) 和工作实习 (on-job practice) 构成,而且一般为每一个新人配备一个有经验的 (资深) 测试工程师作为师傅,负责日常指导工作,了解新员工的进度和学习的难点,检查新人的测试结果、回答问题等。
- 培训的内容:给新测试人员提供培训,让他们提出问题,例如怎样设计和编写测试用例、怎样做探测性的测试、怎样编写错误报告、测试计划中应该有什么内容、如何建立测试环境等。一般包括如下内容。
  - ◆ 了解公司基本情况、人事制度、福利待遇、企业文化等。
  - ◆ 了解 QA 或测试部门的组成、职责、任务、工作现状及发展。
  - ◆ 了解公司的资源,阅读有关文档,包括员工手册、技术工作手册。
  - ◆ 工作报告格式,包括日报、周报、月报、测试报告和质量分析报告。
  - ◆ 理解和掌握公司产品的结构、主流产品的功能、产品发展方向。
  - ◆ 掌握软件测试、QA 的知识、方法、技术和流程。
  - ◆ 学会使用已有的测试工具和测试管理系统。
  - ◆ 公司所采用的其他技术。
  - ◆ 通过实践工作掌握具体的方法、提高独立工作能力。
- ◆ 培训的考核:这是必要的环节,培训不和考核结合起来,不能起到相应的效果,也很难检查培训的效果。考核可以分为中间检查和培训结束时的全面考核,表 12-1 作为一个例子,说明测试工程师一般要考核的项目。

表 12-1 测试工程师考核的项目

考核的项目	优秀 (5)	良好 (4)	通过 (3)	不通过 (1)	差 (-1)
公司产品	>90	>75	>60	<60	<40
测试知识	>90	>75	>60	<60	<40
ISO9001 & CMM 知识	>90	>75	>60	<60	<40
测试技术	优秀	良好	正常	不够好	差
工作态度	热情	积极	正常	消极	恶劣
测试用例设计	90%	80%	70%	<70%	<50%
测试用例执行	举一反三, 覆盖边界	严格执行	符合要求	1~2 个明显问题没被发现	多个明显问题没被发现
发现 bug 能力	95%	90%	80%	<80%	<60%
bug 描述	没问题, 清楚	没问题	只有小问题	描述不清楚, 缺少信息	不会描写
问题分析	积极去做, 有效果	有效果	去做	做得少	不做
报告	高质量, 及时	高质量, 80%	正常, 80%	质量不够好	没做
流程控制	严格遵守	遵守	基本遵守	忽略某些地方	没遵守

## 12.4 测试团队的管理和发展

当完成了人员招聘、建立测试团队之后, 测试团队的管理和发展就成为当前主要工作之一。要想成功管理好自己的团队, 使测试团队有一个良好的发展, 针对测试团队的特点, 管理任务集中在以下几个方面:

- 在整个团队内部树立良好的团队意识。
- 采用相对应的激励方法。
- 将 PSP 到 TSP 思想或全部内容引进到测试团队中。
- 做好知识共享和在岗培训。

当然, 那些团队管理的普遍方法同样有助于测试团队的建设, 包括组织结构的合理设置和优化、岗位责任明确、绩效和目标考核、类似于生日聚会和郊外旅游等团队活动、经理或领导的人格魅力等。在这一节, 主要是针对测试团队的特点, 包括技术特点和在整个软件团队中的地位/角色等来讨论如何做好测试团队管理和发展的工作。

### 12.4.1 树立良好的测试团队意识

要树立起五种良好的测试团队意识: 目标意识、团队意识、服务意识、竞争意识和危

机意识。

### 1. 目标意识

- 目标到人。团队管理会体现到项目管理，项目是否完成了预期的目标，所以团队中每个人必需有明确的目标。考核时，测试组长或经理要关注每一个人完成的结果，当结果不理想时，再返回去逆向考核过程。
- 个人目标与团队目标相结合。除完成项目任务外，每个人必须有明确的自身发展方向，并将自己的发展目标和团队的目标有效的结合起来，个人和团队才能达到“双赢”，进入良性的发展循环过程。
- 责任心。软件质量的目标需要每个团队成员的高度责任心做保障。要求每个成员按时兑现自己所作的承诺，每个人要认清自己工作的重要性和工作失误会带来的严重后果。
- 自信心。团队中每个人应该有足够的自信完成自己的任务，而且不受外界的因素影响。尤其是在项目出现困难和危机的时候，更需要坚定信心，各人完成自己的既定目标，以摆脱团队的危机。

### 2. 团队意识

- 集体成功观。团队中所有成员必需意识到，个人的成功融入集体的成功之中，只有项目成功、团队成功，才谈得上个人的成功。特别是对软件测试团队，团队成功是非常重要的，任何一个人的错误，都会降低团队信誉，测试团队若失去信誉，没有人相信测试结果，那将是最可怕的。
- 树正气，刹歪风，水桶原理更适合于衡量团队战斗力。团队中全体成员要认清即使是极少数人的工作进度拖延，也会造成项目的不可控，即使是个别模块的不稳定也会造成整个系统瘫痪。虽然在团队中不是所有人都很重要，但任何一个人的差错，就会使整个团队的工作功亏一篑。因此，团队中每一个成员要勇于和影响团队士气、同不遵守规则或流程的任何不良行为作斗争。
- 良好的沟通。团队中所有成员应该及时有效沟通，相互理解。团队中出现意见分歧很正常，通过交流、沟通达成一致，不要缺乏沟通使矛盾激化。

### 3. 服务意识

- 面向客户的服务。软件测试工作是提高质量的一个重要环节，目的就是为了提高客户的满意度，追求客户满意。所以，团队成员面向客户的态度是决定软件测试工作成败的关键因素之一。
- 面向团队内部的服务。正如本书前面定义的，客户分为外部客户和内部客户，外部客户是传统意义上的客户或软件产品的最终用户，而下一道工序的接收者是上一道工序生产者的内部客户，如编程人员是系统设计人员的客户，测试人员是设计、编程人员的客户。如果在整个团体内部或软件公司内贯彻这样一个概念，那么这个团队就有了成功的基础。

#### 4. 竞争意识

- 责权利均衡，奖惩分明，团队中也要引入竞争机制。责任定义清楚，并与权利、利益相结合，做好绩效评估、考核，对新的、空的位置，择优上岗，所有事情争取公正处理，形成一种良好的竞争机制，进行正确引导。
- 所有的角色都一样重要。每一个角色都责任不同、工作任务不一样，甚至薪水也不一样，但都同样重要。每一个角色也是相对的，只要做得好，谁都有发展的空间，谁都可以唱主角。

#### 5. 危机意识

在测试团队，危机来自于两个方面，一是从团队内部人员竞争所带给每个测试工程师的危机意识，如果个人不前进就意味着倒退，会被淘汰；另一种危机来自于公司外部的竞争对手，或者说市场优胜劣汰的残酷性，产品质量上出现的问题就是给竞争对手一个喘息的机会，在保证产品质量基础上，还要分析竞争对手的产品，对自己公司的产品不断提出改进意见，从而在功能上还要战胜竞争对手的产品。

### 12.4.2 测试团队的激励方法

由于软件测试工作需要每个成员都需要有高度的责任感、全身心投入，我们就必须通过良好的管理方法和一系列激励措施，在测试小组中保持高昂的士气和动力。比如，可采用一些管理形式去做那些代替组织团队不能做的事情：使测试小组的成员相信，测试部门经理和更上层的经理重视他们的工作和贡献，尊重他们的想法，并且当他们需要帮助时，你会支持他们。

#### 1. 表扬和奖励

表扬和奖励仍然是激励测试人员最主要的一种形式，甚至采用积极的方式去帮助测试人员改正缺点，而不用批评或责备方式。表扬的形式多种，一封 E-mail 是简单的方式，虽然没有一点物质刺激，但还是有作用的。当然也可以设置一系列奖励项目，如

- 发现 bug 最多的人员被授予 bug 王。
- 最有价值 bug (MVB)。
- 优秀测试计划。
- 最有价值测试用例。
- 季度优秀员工。
- 季度优秀新员工。

#### 2. 站在测试小组一边

我们曾经提过，目前国内还存在一些误区，认为测试工作地位要低一些，做编程更具有技术性和挑战性。我们或许不能改变这种不正确的观念，并且在相关政策上，这种错误

的看法从某种程度上也得到体现。然而,可以采取一些办法,把这种想法对测试小组的消极影响降低到最低。

首先要使测试人员坚信,他们的想法和建议能够得到足够的尊重,因为他们最了解产品,对产品的每一项功能都很清楚,所以对产品提出的建议一般都具有针对性、有极高价值。

其次,要保护测试人员。编程人员是可以犯错误的,因为程序中有错误是正常的,没错误是不正常的,程序中的错误有测试人员来把守,但测试人员不能犯错误,如果某个测试人员漏掉某一个 bug,可能会让用户碰到,问题会变得很严重。也就是说,某个测试人员发现 1000 个 bug,结果漏掉一个严重 bug,最后得到的评价就是不好的,可能是前功尽弃。而按照科学研究表明,发现的 bug 越多,潜在的、未发现的 bug 可能性也会越大。所以,测试人员往往受到不公正的对待,需要被保护。当面对别人对你测试小组成员进行愤怒的攻击时,不管这个倒霉的测试人员犯了什么错误,不要公开地批评和指责他。然而,作为测试部门的经理,却应该承认存在的问题,可以说:“是的,我或许能够把工作做得更好。我想是在评审报告时出现了失误,但是让我们再讨论一下这个错误。”这样会有助于测试人员的情绪稳定。

此外,不管测试人员犯了什么样的错误,都不应该干扰对他们从事的正确事情的注意力。如果在一个令人不快的错误报告中缺乏隔离步骤,但是这个错误是不可否认的,那么就不要让人们忽视这个事实。如果一个测试人员的性格,不适应严重性逐渐增强的问题,那么应该把注意力转回到问题本身上来。应该和在场的测试人员一起做这件事情。在公开的场合中,应该明确地支持你的组员,然后在这件事情过去和反应平静之后,找一个机会,私下同大家做必要的交流工作。通过展示在受攻击时表现的这种团结,或者在私下处理他们的失误,你将在你的员工中建立真正的忠诚信念。

### 3. 提高士气

提高士气可以从以下几个方面去做:

- 薪水。如果测试工程师的薪金标准低于公司中其他工程师的水平,就应该和人力资源部一起努力,争取使具有同等职责、经验、学历的人得到同等水平的薪水。如果存在不合理的情况,就设法做些薪水调查,帮助解决实际存在的问题。
- 职务。开发人员的职务可能比测试人员的职务有更高的等级和威望,这种不公平的现象也应该得到消除,即使不能消除,也要朝这个方向努力。也可以设置测试人员专业系列职务,建立一套认证体系,这些专业系列职务包括初级测试工程师、测试工程师、资深测试工程师、主任测试工程师、项目组长、项目经理、产品经理、质量经理、质量总监等。
- 工作时间。由于软件测试的特殊性,可能加班的时间会多些,这就要求测试经理多为测试人员争取额外补偿和其他的利益,或指出这样做的不公平性,设法制订一个更合理的项目日程表,改进工作方法,提高工作效率。
- 培训机会。由于测试任务重、或者认识上的误区(如觉得测试技术性低,没必要参加培训),测试人员很少有机会参加一些重要的培训。作为测试经理,就应该做

好培训预算，设法创造机会，至少让那些优秀的测试人员去参加非常有用、有吸引力的软件测试技术或项目管理等方面的会议。

#### 4. 支持合理的工作方式

由于软件测试的特殊性，不切实际的进度安排和软件开发前期工作没做好，结果所有被延误的时间会压向测试阶段，测试所需要的时间被挤压，从而造成测试人员加班比较常见的现象。所以，要帮助他们建立有效的、合理的工作方式，保缓解被挤压的时间所带来的压力，并且注意关心测试小组成员的成长。

在软件测试中，常用的方法有：

- 白天创建、编辑测试脚本，在下班前启动自动测试脚本，让系统晚上自动运行测试，第二天早上拿到结果。
- 调节各人休息时间表，保证测试每周 7 天都能执行测试，充分利用测试机器和其他资源，对每个测试人员依然每周工作 5 天。
- 将测试工作进行分解、细化，一部分人（2~3 个人比较好，至少包括测试项目组长）可以先进入某个测试项目，设计测试计划、测试用例、建立环境等。
- 项目计划或产品功能的变化，对测试影响要比开发大，事先要对这些风险进行充分估计，在估计测试时间时，要留有一定的余地（10% buffer）。

### 12.4.3 从 PSP 到 TSP

早在 20 世纪 70 年代中期，美国国防部就组织力量研究软件项目失败的原因，发现在失败的软件项目中，70%是由于管理不善所造成的，因而认为管理影响全周，并掀起了研究软件管理技术的热潮。20 年后，根据美国三份经典研究报告，这一状况并未得到转变。软件开发仍然很难预测，大约只有 10% 的项目能够在预定的费用和进度下交付符合需求的软件，管理仍然是软件项目成败的主要因素，并指出开发过程中的返工是软件过程不成熟的标志。例如在 Capers Jones 的报告中就指出，在 17 个影响软件项目成败的主要因素中，与项目管理直接有关的有 6 个（软件度量、工作量估计、项目规划、进展报告、需求变化、风险管理），间接有关的有 3 个（系统构架、开发方法、配置管理）。

为了系统地解决软件项目管理问题，美国国防部于 1984 年在 Carnegie-Mellon 大学建立了软件工程研究所，1986 年开始研究并于 1991 年提出能力成熟度模型 CMM，1989 年开始研究并于 1994 年提出个体软件过程 PSP，1994 年开始研究并于 1998 年由 CMU/SEI 召开的过程工程年会上第一次介绍了 TSP 草案，于 1999 年发表了有关 TSP 的一本书，使软件过程框架形成一个包含 CMM、PSP 和 TSP 三者的严密的整体。

为了全面提高团队的个人素质和团队整体的作战能力，可以实施 PSP 和 TSP，至少将 PSP 和 TSP 中的思想引入到团队的管理和发展中。

#### 1. PSP（personal software process，个体软件过程）

PSP 是一个过程描述、度量和方法的结构化集合，通过采用一些表格、脚本和标准，

可帮助软件工程师估算和计划其工作，能够帮助软件工程师改善其个人素质。

PSP 基本原则是每个人都是不同的，对于某个工程师有效的方法不一定适合另一个工程师，PSP 帮助工程师测量和跟踪自己的工作，使之能够找到最适合自己的方法。软件工程师在做项目的开发计划时，或是由经验而来，或是由用户需求而定，往往存在计划与实际相差比较大的情况，包括时间表和开发成本，或者是前松后紧，遗漏过多，造成维护量的增加。如何减少这种情况的发生？就需要把经验量化并做出分析。通过记录项目的估算情况与实际情况，并进行比较分析，则既利于有经验的软件工程师提高以后项目的预测率，也利于新软件开发人员参考其他工程师的经验，以利于以后工作的规划与开展。

通过记录软件工程师在项目设计及编写代码阶段出现的错误及解决办法，以及记录测试与维护阶段出现的错误、缺陷及解决办法，并产生报告，列出经常出现的错误及错误类型，可把错误尽量控制在交付用户使用前，并尽量减少错误的发生。当然，项目评估的准确度依赖于历史数据的积累，只有正确的历史数据越充分，在评估新项目时所采用的指标数才会越准确。在项目进展过程中，还需要根据影响因素的变化不断调整估算指标。

## 2. TSP (team software process, 团队软件过程)

TSP 对团队软件过程的定义、度量和改革提出了一整套原则、策略和方法，把 CMM 要求实施的管理与 PSP 要求开发人员具有的技巧结合起来，以按时交付高质量的软件，并把成本控制在预算的范围之内。在 TSP 中，讲述了如何创建高效且具有自我管理能力的工程小组，工程人员如何才能成为合格的项目组成员，管理人员如何对团队提供指导和支持，如何保持良好的工程环境使项目组能充分发挥自己的水平等软件工程管理问题。具体地说，TSP 的目标为创建具有自我管理能力的团队，管理人员要善于引导和激励团队的全体成员使他们能发挥自己的最高水平，采用 CMM 来进行软件过程的改革，为处于高成熟度的软件组织的过程改革提供指导。

- TSP 基于以下四条基本原理：应该遵循一个确定的、可重复的过程并迅速获得反馈，这样才能使学习和改革最有成效；一个团队是否高效，是由明确的目标、有效的工作环境、有能力的教练和积极的领导等四方面因素的综合作用所确定的；应注意及时总结经验教训，当组员在项目面临各种各样的实际问题并寻求有效的解决问题方案时，就会更深刻地体会到 TSP 的力量；应注意借鉴前人和他人的经验，来规定过程改进的指令。
- 在实施 TSP 的过程中，应该贯彻集体管理与自我管理相结合的原则，具体地说，有以下六项原则：
  - ◆ 计划工作的原则。在每一阶段开始时制订工作计划，规定明确的目标。
  - ◆ 实事求是的原则。目标不应过高也不应过低而应实事求是，在检查计划时如果发现未能完成或者已经超越规定的目标，应分析原因，并根据实际情况对原有计划作必要的修改。
  - ◆ 动态监控的原则。一方面应定期追踪项目进展状态并向有关人员汇报，另一方面应经常评审自己是否按 PSP 原理进行工作。

- ◆ 自我管理的原则。开发小组成员如发现过程不合适,应主动、及时地进行改进,以保证始终用高质量的过程来生产高质量的软件,任何消极埋怨或坐视等待的态度都是不对的。
- ◆ 集体管理的原则。项目开发小组的全体成员都要积极参加和关心小组的工作规划、进展追踪和决策制订等工作。
- ◆ 独立负责的原则。按 TSP 原理进行管理,每个成员都要担任一个角色。建议在一个软件开发小组内把管理的角色分成客户界面、设计方案、实现技术、工作规划、软件过程、产品质量、工程支持以及产品测试等八类。如果小组成员的数目较少,则可将其中的某些角色合并,如果小组成员的数目较多,则可将其中的某些角色拆分。总之,每个成员都要独立担当一个角色。
- TSP 流程使用 23 个过程指南、14 个数据表格和 3 个标准。在这些过程指南中定义了 173 个启动和开发步骤。每一个步骤都不复杂,但它们的描述都非常详细,以便开发人员能够清楚地知道下一步应该做什么,应该怎样去做。这些过程指南可用来指导项目组来完成启动过程和一步步地完成整个项目。
- ◆ TSP 过程质量度量元可用两组元素来表达,一组元素用以度量开发小组的素质,称之为开发小组素质度量元,另一组用以度量软件过程的质量,称之为软件过程质量度量元。开发小组素质的基本度量元有以下五项,所编文档的页数;所编代码的行数;花费在各个开发阶段/任务上的时间;在各个开发阶段中注入和改正的缺陷数目;在各个阶段对最终产品增加的价值。软件过程质量的基本度量元有以下五项,设计工作量应大于编码工作量;设计评审工作量至少应占一半以上的设计工作量;代码评审工作量应占一半以上的代码编制的工作量;每千行源程序在编译阶段发现的差错不应超过 10 个;每千行源程序在测试阶段发现的差错不应超过 5 个。

## 12.4.4 知识共享和在岗培训

应该安排合适的工作,同时也供正式和非正式的培训机会,来帮助人们实现技能增长的需要。对于非正式的培训机会,相当于是达到知识共享的一系列活动的一部分,包括:

- 组内技术交流和讨论。
- ◆ 让经验丰富或技术好的小组成员进行指导或举办讲座。
- ◆ 建立测试部门的小图书室或图书俱乐部,测试小组一起阅读测试方面的书籍,然后在就餐时间或其他时间进行讨论、交流。

正式的培训机会,是指参加外部的专业培训班、正式的测试技术研讨会、专家提供的培训课程等,也包括将专家请到公司内部所做的培训,以及公司内部组织的、类似 CMM 那样的辅导课。

为了做好在岗培训的工作,除了前面所述的,要采取灵活丰富的多种形式之外,主要抓好两点。

- ◆ 测试培训流程,包括组织、考核等。



- 测试内容及其模块化, 可以根据培训对象的不同要求, 进行裁剪。

### 1. 测试培训流程

为了建立一套测试培训流程, 首先要设定专业的测试培训专员, 来负责培训流程的制定, 协调测试培训模块的建立, 启动和跟踪测试流程的执行以及最后培训的反馈、考核等。一套完整的测试培训流程应包括的内容有:

- 测试培训对象的定义, 即使都是对在职的测试人员培训, 其层次也是不一样的, 有初级测试工程师也有资深测试工程师, 有组员也有组长。
- 测试目标的明确, 培训目标的四个方向是计算机软件技术、软件测试业务需求、项目和团队管理提升、相关领域的知识扩展等。测试培训就是要围绕着四个方向去定义有针对性的、可度量/考核的、具体目标。
- 如何创建、审查培训模块、课程, 相当于学校的课程大纲, 但在软件测试培训中模块化更强, 内容范围小、针对性强。
- 如何认证一个合格的培训教师, 来保证培训的效果。
- 如何审查培训的内容, 确保符合培训课程大纲的要求。
- 如何收集培训之后的反馈结果, 包括反馈表的限制。
- 如何对培训进行考核, 包括培训考试, 以及将参加培训的积极性、次数、成绩记入全年员工绩效考核中。
- 对培训结果/效果的分析, 以不断改进培训。

### 2. 测试内容及其模块化

CSDP 描述了一个资深测试工程师所要培训的内容, 如下所列:

- 商业活动和工程经济。包括工程经济、规范、专业实践、标准。
- 软件需求。包括需求工程过程、需求引出、需求分析、软件需求说明书、需求确认、需求管理。
- 软件设计。包括软件设计理念、软件体系结构、软件设计质量分析和评估、软件设计标记和文档、软件设计策略和方法、软件设计中人的因素、软件和系统安全。
- 软件实施。包括计划实施、代码设计、数据设计和管理、错误处理、源代码组织、代码文档、QA 实施、系统集成和配置、调整代码、实施工具。
- 软件测试。包括测试类型、测试水平、测试策略、测试设计、代码测试覆盖、说明书测试覆盖、测试执行、测试文档、测试管理。
- 软件维护。包括软件可维护性、软件维护过程、软件维护度量、软件维护计划、软件维护管理、软件维护文档。
- 软件结构管理。包括 SCM 过程管理、软件配置确认、软件配置控制、软件配置状态计算、软件配置审核、软件发布管理和递交。
- 软件工程管理。包括度量、组织管理和协调、初始化和范围定义、计划、软件获取、设定、风险管理、复审和评价、工程结束、后期终止活动。
- 软件工程过程。包括过程基础构造、过程度量、过程定义、定性的过程分析、过

程执行和变更。

- 软件工程工具和方法。包括管理工具和方法、开发工具和方法、维护工具和方法、支持工具和方法。
- 软件质量。包括软件质量理念、计划 SQA 和 V&V、SQA 和 V&V 的方法、应用到 SQA 和 V&V 上的度量法。

## 12.5 优秀软件测试工程师的必备素质

人是测试工作中最有价值也是最重要的资源，只有保证测试工程师良好的素质，才能保证测试、产品的质量。然而，在软件开发产业中有一种习惯非常普遍，就是让那些经验最少的新手、没有效率的开发者或不适合于其他工作的人去做测试工作。这绝对是一种目光短浅的行为，对一个系统进行有效的测试所需要的技能绝对不比进行软件开发需要的少，事实上，测试者需要获得极其广泛的经验，去解决所遇到许多开发者不可能遇到的问题。

为高质、高效地完成测试任务，优秀的软件测试工程师应具有很好的素质和能力，包括沟通能力、技术能力、自信心、交流能力和幽默感、耐心、很强的记忆力、怀疑一切的精神、勤奋精神、洞察力、适度的好奇心、反向思维和发散思维能力等等。沟通能力和技术能力在招聘测试人员时曾经介绍过，我们有必要再复习一下。

### 1. 沟通能力

优秀的测试工程师必须能够同测试涉及到的所有人进行沟通，具有与技术（开发者）和非技术人员（客户、管理人员）的交流能力。既要可以和用户谈得来，又能同开发人员说得上话，不幸的是这两类人没有共同语言。和用户谈话的重点必须放在系统可以正确地处理什么和不可以处理什么上，尽量不使用专业术语。而和开发者交流时，尽量要使用专业术语，这对用户反馈的相同信息，测试人员必须重新组织，以另一种方式表达出来，测试小组的成员必须能够同等地同用户和开发者沟通。

### 2. 技术能力

就总体而言，开发人员对那些不懂技术的人持一种轻视的态度。一旦测试小组的某个成员做出了一个比较明显的错误断定，可能会被夸张地到处传扬，那么测试小组的可信度就会受到影响，其他正确的测试结果也会受到质疑。再者，由于软件错误通常依赖于技术，或者至少受构造系统所使用的技术的影响，所以测试人员掌握编程语言、系统构架、操作系统的特性、网络、表示层、数据库的功能和操作等知识，应该了解系统是怎样构成的，明白被测软件系统的概念、技术，要建立测试环境、编写测试脚本，又要会使用软件工程工具。要做到这些，需要有几年以上的编程经验以及对技术和应用领域的深刻理解。

### 3. 自信心

开发人员指责测试人员出了错是常有的事，测试工程师必须对自己的观点有足够的自

信心,对自己所报的 bug 有信心。如果没有信心或受开发人员影响过大,测试工作就缺乏独立性,程序中的漏洞或缺陷容易被忽略过去,就谈不上保证软件产品质量。

还有一种情况也是常见的,软件产品设计规格说明书总是或多或少存在一些逻辑问题,编程人员和测试人员对那些有问题的功能存在争议,这时候信心会帮助测试人员发现产品设计中的问题。

#### 4. 外交能力和幽默感

优秀的测试人员必须能够同测试涉及到的所有人(编程、设计等技术人员,客户、管理等非技术人员)进行良好的沟通。机智老练和外交手法有助于维护与开发人员的协作关系,幽默感同样也是很有帮助的。

测试人员应该把精力集中在查找错误上面,而不是放在找出是开发小组中哪个成员引入的错误。这样可以保证测试的否定性结果只是针对产品,而不是针对编程人员,也就是说要使用一种公正和公平的方式指出具体错误,这对于测试工作是有益的。一般来说,武断地对产品进行攻击是错误的。如果采取的方法过于强硬,对测试者来说,在以后和开发部门的合作方面就相当于“赢了战争却输了战役”。在遇到狡辩的情况下,一个幽默的批评将是很有帮助的。

#### 5. 耐心

有些软件测试工作需要难以置信的耐心。有时需要花费惊人的时间去分离、识别一个错误,需要对其中一个测试用例运行几十遍、甚至几百遍,了解错误在什么情况、或什么平台下才发生。测试人员需要保持平静,尤其是在集中注意力解决困难问题的时候,特别是在测试执行阶段,面对成百上千个测试用例,要一个个去执行,还要在不同的测试环境上重复,耐心是必要的。当然,我们尽量让测试工具去完成那些重复性的任务。

#### 6. 很强的记忆力

一个优秀的测试工程师应该有能力将以前曾经遇到过的类似错误从记忆深处挖掘出来,这一能力在测试过程中的价值是无法衡量的。因为许多新出现的问题和已经发现的问题相差无几。

在测试一个产品的高版本时,对以前所发布的各种版本产品功能清楚,就很容易了解新版本的功能做了哪些改动、为什么那么改、改了之后会对其他特性有哪些影响等一系列问题。如果熟悉软件各种老版本所出现的缺陷,有助于对新版本的用例设计和测试执行。

#### 7. 怀疑精神

可以预料,开发人员会尽他们最大的努力将所有的错误解释过去,测试人员必须听每个人的说明,但必须保持高度警惕、怀疑一切,直到自己的分析结果或亲自测试之后,才做出决定。并具有自我督促能力,才能够保证每天的工作都能高质量完成。

#### 8. 洞察力

一个好的测试工程师具有一种先天的敏感性,并且还能尝试着通过一些巧妙的变化去

发现问题。同时,还具有“测试是为了破坏”的观点,捕获用户观点的能力,强烈的质量追求,对细节的关注能力。应用的高风险区的判断能力以便将有限的测试针对重点环节。

### 9. 适度的好奇心

优秀的测试工程师在开发测试用例时使用的方法,与勘探专家在一个山洞中摸索前进的方法一样。虽然周围可能存在大量的死路,但是测试工程师具有适度的好奇心,会促使他们向山洞中的深处探索,向一切没有去过的地方前进,最终可能会有一个大发现。

编写出导致错误出现的测试用例,这就需要好奇心。测试工程师必须阅读规格说明,与开发人员一起讨论“假设分析”的场景,并在大脑中反复思考被测试系统,还要从所有的角度加以检查。测试工程师如果没有好奇心并对要达到的目标缺乏强烈兴趣,那么他只能写出肤浅的测试用例。

如果测试人员在一个错误上花费很多时间,通过尝试很多变体去探索造成这种错误出现的根本原因,这样做也是不正确的。所以,好奇心需要适度。应该使用什么标准去区分“足够好奇”、“不够好奇”和“过分好奇”呢?在及时完成测试执行任务和编写灵活高效的测试用例之间,在进度的压力和探究错误发生根源之间,优秀的测试人员能够取得平衡。

### 10. 反向思维和发散思维能力

测试工程师应想尽办法来考虑产品可能出现失败的各种方式会最大限度地暴露其存在的问题、用严格的边界条件来检验它,让系统经受压力测试,或者是强迫它处理“不可能发生的”错误。因此,优秀的测试人员应具有是在一种安全的环境下发现错误,并且之后可以让项目小组来修复它们。测试需要通过悲观的思想倾向去追求崇高的目标,只有提高产品的质量才能使公司取得成功。

## 小结

在本章我们了解到软件测试团队的基本任务就是建立测试计划、设计测试用例、执行测试、评估测试结果和递交测试报告,最终提高产品质量,保证所发布的软件产品具有一定水平的质量。为了完成这个任务,必须组建一支专业技术强的测试团队。在这个专业测试团队中,要建立合理的团队结构,并从技术角度或产品线去构造团队的组织模型。了解到测试团队的基本构成和设置的不同岗位,包括初级测试工程师、测试工程师、资深(高级)测试工程师、测试实验室管理员、软件包构建或发布工程师、测试组长、测试经理等等,清楚地定义这些岗位的责任和要求,对人员招聘、团队的组建有很重要的指导意义。

如何发掘、面试和聘用优秀的测试人员,关键是了解一个优秀的测试人员应该具备哪些基本的素质和能力,我们除了强调沟通、技术、思维等基本能力之外,还谈到一些基本素质,包括幽默感、耐心、怀疑一切的精神、勤奋精神、适度的好奇心等。

对测试人员的激励和培训非常重要,这时由人才市场资源和测试工作本身的特点所决定,要坚持站在测试人员一边,通过不同的方式提高士气,培养测试人员的信心和精益求精的精神。同时,做好培训的流程和培训内容的模块化,引入PSP和TSP思想,整体地提

高测试团队的协调能力和测试效率。

## 思考题

1. 软件测试团队的任务是什么？
2. 开发工程师如何和测试团队合作？之间的关系又是怎样？
3. 在不成熟的软件测试团队中，最常见的问题是什么？
4. 如何建立、提高测试团队的地位？

# 第 13 章 测试环境的建立

早在设计测试计划时，测试环境与资源配备就是一项重要的内容。测试环境的建立是测试工作的基础。合理的计划安排还可以节约软件成本。一位优秀的测试人员应有能力规划测试环境，并根据需要对测试环境进行调整。

## 13.1 测试环境的重要性

配置测试环境是测试实施的一个重要阶段,测试环境适合与否会严重影响测试结果的真实性和正确性。

### 1. 测试环境的定义

软件测试环境包括设计环境、实施环境和管理环境。本章讲述的是通常意义上的测试环境即测试的实施环境。

- 软件测试设计环境：编制测试计划/说明/报告及与测试有关的文件所基于的软件/硬件设备和支持。在设计阶段根据客户的需求进行环境设计，当然期望测试环境无限接近于客户所需软件运行的真实环境，但实际上由于各种资源的限制，只能在近似的模拟环境中进行测试。
- 软件测试实施环境：对软件系统进行各级测试所基于的软件/硬件设备和支持。测试实施环境包括被测软件的运行平台和用于各级测试的工具。实施环境必须尽可能地模拟真实环境，以期能够测试出真实环境中的所有问题，同时也需要理想环境以便找出问题的真正原因。
- 软件测试管理环境：管理测试资源所基于的软件/硬件设备和支持。测试资源指测试活动所利用或产生的有形物质（如软件、硬件、文档）或无形财富（如人力、时间、测试操作等）。广义的测试管理环境包含测试设计环境、测试实施环境和专门的测试管理工具。例如，对 bug 的跟踪、分析管理，对 CASE 的分类管理，对测试任务的分派、资源管理等。

### 2. 测试环境是测试的基础

测试环境贯穿了测试的各个阶段，每个测试阶段中测试环境对测试影响是不一样的。

在测试的计划阶段，充分理解客户需求，掌握产品的基本特性有助于测试环境的设计，合理调度使用各种资源，申请获得未具备的资源，保证计划的顺利实施。如果在测试计划中规划了一个不正确的环境，直到实施的过程中才发现，浪费了大量的人力和物力取得一些无用的结果，即使只是遗漏了一些环境配置，如不能及时发现，及时申请购买或调用，

也会影响整个项目的进度。在计划阶段，考虑周全很重要。

在单元测试和集成测试阶段，有部分测试工作是由开发人员完成的。开发人员的测试环境通常为开发环境，近似于理想环境。理想环境有利于代码的调试和分析，但测试结果不能视为真实结果。有这样一个例子，测试人员报告的 bug 在开发环境中无法重现，开发人员就在测试人员的测试环境中研究，原来是环境系统的设置不同造成的，此时测试人员就应该分析修改系统设置是否合理。如果合理，这就是一个很棒的解决方案，但要求用户手工修改系统设置，或不能识别用户的系统设置通常都是不合理的，这应该是个严重的 bug。

在系统测试和验收测试阶段，测试环境必须模拟并最大限度地接近实际环境。测试人员在设计测试案例时就得写明测试环境，因为在不同的环境中预期的结果是不同的。测试中运行测试案例，报告 bug 时有一项基本的要求，就是写明测试环境，以便开发人员再现 bug，减少不必要的交流和讨论。大型的软件系统，特别是支持多平台的软件系统，往往测试环境比较复杂，而且在不同的环境下，软件的特性有差异，问题的解决方案也不同。

测试环境是软件测试的基础，使用错误的测试环境，可能会遇到以下的情形：

- 得出完全错误，甚至是相反的结果。
- 得出的结果与实际使用中的结果有很大误差。
- 忽略了实际使用可能会出现严重错误，将严重的 bug 遗留到客户的手中。
- 导致项目返工，造成巨大的资源浪费。
- 导致项目延期，信誉的损失。

所以，测试环境问题的重要性应该得到充分的重视。尽量将测试环境的因素降到最小，避免因测试环境出现的问题。

## 13.2 测试环境的各要素

测试环境包括硬件环境和软件环境，硬件环境指测试必需的服务器、客户端、网络连接设备，以及打印机、扫描仪等辅助硬件设备所构成的环境；软件环境指被测软件运行时的操作系统、数据库及其他应用软件构成的环境。细分测试环境的五个要素是：软件、硬件、网络环境、数据准备、测试工具。在阐述这五要素前，先了解使用环境、主测试环境及辅测试环境的概念。

### 13.2.1 产品的使用环境对测试环境的影响

产品的可用性不仅取决于产品本身，还受使用环境的影响。使用环境包括以下几方面：

- 用户特征：包括用户的知识背景、技能、经验、学历、年龄、体力等，有时还可按经验、职务或能力来分组。
- 使用产品的目标：产品的主要用途以及工作时间长短、强度等。
- 社会物理环境：如计算环境、温湿度环境、法律环境、社会文化习惯环境等。

在详细分析使用环境的基础上,设计出测试环境并在测试计划中明确定义,并使测试环境具有使用环境的典型特征,包括:

- 测试对象特征:知识背景、技能、学历、年龄和体力等。
- 测试任务:应符合产品的主要用途并覆盖其主要功能,详细规定任务要求和执行顺序。
- 测试的社会物理环境:所使用的物理设备、软件以及相关的物理和社会环境特征。

### 13.2.2 主测试环境与辅测试环境

在实际测试中,软件环境又可分为主测试环境和辅测试环境。主测试环境是测试软件功能、安全可靠、性能、易用性等大多数指标的主要环境。一般来说,配置主测试环境可遵循下列原则:

- 符合软件运行的最低要求,测试环境首先要保证能支撑软件正常运行。
- 选用比较普及的操作系统和软件平台。
- 营造相对简单、独立的测试环境。除了操作系统,测试机上只安装软件运行和测试必需的软件,以免不相关的软件影响测试实施。
- 无毒的环境。利用有效的正版杀毒软件检测软件环境,保证测试环境中没有病毒。

辅测试环境常常用来满足不同的测试需求或特殊测试项目,例如:

- 兼容性测试:在满足软件运行要求的范围内,可选择一些典型的操作系统和常用应用软件对其安装卸载和主要功能进行验证。
- 模拟真实环境测试:有些软件,特别是面向大众的商品化软件,在测试时常常需要考察在真实环境中的表现。如测试杀毒软件的扫描速度时,硬盘上布置的不同类型文件的比例要尽量接近真实环境,这样测试出来的数据才有实际意义。
- 横向对比测试:利用辅测试环境“克隆”出完全一致的测试环境,从而保证各个被测软件平等对比。

主测试环境与辅测试环境是相对的,不同性质的软件对其要求也不一样,利用好主测试环境与辅测试环境,不仅有利于软件产品的充分测试,也可减少测试的工作量。

### 13.2.3 测试环境的五要素

测试环境的基本要素是:软件、硬件。在基本要素的基础上派生出网络环境、数据准备、测试工具三要素。

#### 1. 硬件

软件测试最基本的硬件包括服务器和测试用机。测试 Audio、Video 等多媒体产品需要配备摄像头、麦克风、音箱等;测试产品的 USB 接口,并行、串行和红外接口的功能时需要配备移动硬盘、打印机、扫描仪等设备;测试基于手机、数码相机的二次开发的软件需要配备手机、数码相机等;测试网络产品,搭建网络环境时需要配备交换机、路由器等网



络设备；测试基于 HTTPS 的产品安全性需要配备 SSL-BOX。为保证测试工作的正常运行，有时还得配备 UPS（不间断电源）、稳压电源等附属设备。

硬件设备多种多样，完全根据产品的需求进行选择。但选择时有一个配置的标准。通常有标准配置、最佳配置和最低配置几个概念。

例如，一台服务器的主要性能指标由 CPU、主板、内存、硬盘决定。现设计要求将来的应用服务器（或者已经应用了的服务器）配置：Intel 架构，双 Xeon CPU 主频是 2.4GHz，内存为 1GB，硬盘为 36GB SCSI，网卡为 1000Mb/s 自适应，机箱为 2U。此配置为标准配置，完全符合设计要求。

如设计指定标准配置同时，建议使用性能更高配置的服务器也就是最佳配置。如已拥有性能更高配置的机器也可以作为一种选择，但需验证与标准配置的兼容性，没必要重新添置标准配置的机器。但相关的性能测试、容量测试等还是应该在标准配置的服务器上运行。

最低配置指的是能够满足系统运行所需的最低硬件配置，在某些情况下用低配置设备搭建部分测试环境是明智的。如预算经费不足时，或已经拥有部分低配置设备时。有些配置测试的任务本身就是要求能够保证软件系统能够在最低配置的设备上正常运行。

通常一个较完善的测试环境均包括标准配置、最佳配置和最低配置的设备，只是根据项目的需求和条件的限制所占的比例不同。如压力测试、性能测试、容量测试必须保证在标准配置及最佳配置的设备上运行，而功能性测试、用户界面测试等完全可以在低配置的机器上运行。

## 2. 软件

软件环境包括操作系统和应用程序。测试工具软件也是软件环境派生出来的一部分。建立软件测试环境的原则是选择具有广泛代表性的重要操作系统和大量应用程序。在兼容性测试中软件环境尤其重要，例如对 Web 的测试。

常见的操作系统如下。

- Windows 系列：DOS、Windows 98、Windows NT、Windows 2000、Windows XP。
- UNIX 系列：Solaris 2.6、Solaris 2.8、Red Hat 8.0。
- Mac 系列：OS X。
- 嵌入式操作系统：VxWorks、pSOS、QNX、Nucleus Plus、CMX、Windows CE、DeltaOS。

常见的数据库管理系统如下。

- Oracle 8i、Oracle 9i      PC 平台、Sun 平台
- Sybase      PC 平台
- MS SQL Server 2000      PC 平台

常见的应用程序如下。

- Microsoft Office 2000；
- Foxmail；
- Realplay。

### 3. 数据准备

测试的数据源非常重要，应尽可能地取得大量真实数据。无法取得真实数据时尽可能模拟出大量数据。数据准备包括数据量和真实性两个方面。

现实中越来越多的软件产品需要处理大量的信息，不可避免地使用到数据库系统。少量数据情况下，软件产品表现出色，一旦交付使用，数据急速增长，往往一个简单的数据查询操作就可能耗费掉大量宝贵的系统资源，使产品性能下降，失去可用性。这样的案例已经太多了。

数据的真实性通常表现为正确数据和错误数据，在容错测试中对错误数据的处理和系统恢复是测试的关键。对于更为复杂的嵌入式实时软件系统，例如惯性导航系统仅有惯性平台还不够，为了产生测试数据，还必须使惯性平台按要求运动起来，也可以用软件来仿真外部设备，但开发仿真程序又并非易事。

### 4. 网络环境

随着网络的普及，越来越多的软件产品离不开网络环境，网络环境是硬件因素和软件因素的综合。各种路由器、交换机、网线、网卡等是硬件基础，各种代理、网关、协议、防火墙等是软件基础。正确的网络环境离不开人的因素，搭建、维护、调整网络环境以适应测试的需要。人为的造成网络环境的错误，也将导致测试任务的失败。负责网络环境的测试人员应具备网络管理员的基本素质。

### 5. 测试工具

选择测试工具的描述包括两个方面：折中需求和实际条件来选择已有的测试工具；有重点地自行开发测试辅助工具。有时软件测试必须依托工具，以便测试过程的自动/半自动执行和测试结果的自动/半自动评审和报告。目前市场上测试工具分为三类：代码分析工具、自动/半自动测试过程管理工具和测试资源管理工具。

## 13.3 建立测试实验室

对于专业的测试公司或拥有专业测试队伍的公司，建立不同的测试实验室以适应不同软件的测试需求，然而在一个新公司中，对于第一次组建测试实验室的人来说，就会面临很多困难。本节将以 C/S 结构产品中如何建立服务器端压力测试实验室为例进行说明。

通常建立测试实验室需要经历四个步骤：

- (1) 论证建立测试实验室的必要性。
- (2) 选择和规划实验室场所。
- (3) 制定测试实验室配置清单。
- (4) 集成和配置测试设备。

### 13.3.1 建立测试实验室的必要性

并不是所有的软件公司都需要测试实验室。某些公司只是在特定的时期需要测试实验室,一些公司从来就没有过测试实验室,而是将测试工作交给专业测试公司去做。建立和维护一个测试实验室的成本是非常昂贵的,所以在决定是否建立实验室时需要进行仔细的评估。

评估时以下几点有助于做出决定。

- 是否需要长期使用测试设备? 软件产品开发周期不长,测试周期也不会长,试验室的利用率不高,相对成本就很高。也许交给专业测试公司去做会更合适。如果可以多个项目同时、顺序或两者兼而有之地进行,就会提高实验室的利用率,从而降低测试成本。

如开发 C/S 结构的产品,建立服务器端压力测试实验室通常考虑产品的测试周期一般较长,压力测试通常需要 24 小时以上的高强度运行来发现产品中的问题。同时,关于服务器端的其他功能性测试、容量测试、性能测试等均是比较耗时,而且可以在压力测试实验室环境中运行的。如公司会对该产品不断地升级换代,也就是需要不断地在新版本上测试,就有建立压力测试实验室的必要了。如果还能够配合其他项目或产品的测试则更好。

- 是否需要体积庞大的测试工具? 在测试工具中,是否存在需要放置在长期固定的位置且不便携带的工具,例如前面提到的惯性导航系统如果不依赖于仿真程序,就需要惯性平台,而且让被测对象运动起来。这样通常需要建立专门的测试试验室。

通常的软件产品较少用到体积庞大的硬件测试工具,综合其他因素如果没有成立专门的实验室的必要,尽可能地将测试分布到日常工作环境中去,例如仅仅对 Web 站点的功能测试,网络基础设施和服务器或许直接放在公司的服务器机房中,办公室内的少量工作用机就可以完成测试任务。

- 是否需要特殊的环境? 如果测试平台有严格的环境要求,例如对电压、温度、湿度、空间等,则有必要建立单独的测试实验室。

前面所说的服务器端压力测试,服务器可能是一组服务器,也可能是多组服务器,而且可能分布在不同的网段内,在进行不同的测试任务时结构可能需要调整。压力测试的数据量较大,也决定其不宜与公司服务器机房等一起放置造成网络堵塞,而应该建立独立的测试环境,否则会影响公司其他业务的正常运转。另一方面,压力测试和容量测试需要大量的测试用机,如果每一台服务器设计容量为 1000 个联机用户,多台服务器形成一种分布式结构用来支持各地的用户,即使在最小化的服务器结构中也需要支持近万个用户的测试。即使可以通过测试工具模拟大量真实用户,每台测试用机支持模拟用户的量也是有限的,而且还需要一些真实的客户端用于分析测试中出现的缺陷,测试平台由多组服务器与大量的测试用机组成决定需要一个独立的实验室,以便于操作和维护。

- 是否存在安全性问题? 测试的软件也许是公司最新的成果,各种数据和资料是否会被泄密,出于安全性考虑知道的人越少越好;测试使用的设备中,有些是体积

小但比较昂贵的设备是否会被其他人员“借走”；非测试人员可能会无意中改变测试环境中的一些设置，导致测试失败，造成损失，例如其他小组的一些人不能自觉地约束自己，他们总是使用补丁程序或试图在测试平台的一些设备上验证某些想法，但随后就忘记了取消所做的操作。建立测试实验室拥有独立的测试环境，严格管理制度或章程或许能够阻止这些对测试平台的破坏性活动。

综合上述因素进行分析，如果确实需要测试实验室，要先试着单独为测试实验室做一个预算，它的成本很是高的，而且维护成本也很高。如果预算成本让人无法承受，那么就需要寻找其他的方案。通过谨慎地使用诸如第三方测试实验室或供应商之类的外部资源，就可以利用他们在实验室和在实验室资源上的投资来降低或消除自建实验室的必要性。但有时非常昂贵的测试预算也最终被证明是精明的投资。

### 13.3.2 选择和规划实验室

一旦确定有建立测试实验室的必要，就需要为实验室选择场所并规划它的配置。应当考虑各种因素，例如空间尺寸、照明、布局、功能区、温度、湿度、防火和安全、电源、静电、设施，等等，尽可能描绘出实验室的量化层平面图进行规划，不断地完善调整规划。

**空间尺寸：**通常情况是在现有的建筑中选择合适的房间，而不是从打地基开始建一个建筑物。选择之前，先预计的实验室空间大小、形状、高度等。选择时房间的大小决定了实际的工作空间，房间的形状决定了布局方法，房间的高度可能会影响设备（过高的设备或设备架）的安置。再绘制实验室的图纸通道的位置，门的位置和开启方向等均应标识清楚。如果需要建设完整的建筑物，就简单多了，提供设计方案和具体要求，由专业设计人员完善该方案。

**照明：**自然照明通常指窗户的采光效果，实验室中的窗户应该安装有色玻璃和有效的遮阳物，同时还要注意窗户距离地面的高度。阳光的照射能够产生高温，还会让人很难看清楚计算机屏幕显示的内容，而且它还有可能破坏价格昂贵的设备。人工照明通常使用荧光灯，虽然白炽灯与荧光灯一样，不会使人感到头痛或眼痛，但是它只能照亮有限的区域。聚光灯和轨道灯虽然很亮，但是发出的光的类型决定了它们更适合安装在会议室而不是实验室。当测试人员阅读文档资料和屏幕上的内容或者评估显示的质量时，平和稳定的光照是非常重要的。在平面图上应该显示照明装置和窗户的位置。

**布局：**当选好实验室空间之后，一定要清楚将要安装哪些类型的操作台和设备架。哪些设备必须放在设备架上，哪些设备必须独立放置，选用什么样的操作台或者桌椅，留出足够的空间保证人和设备能够自由出入。

**功能区：**如果在同一实验室有多个工作区，明显的划分有利于设备的维护和管理，也避免不同测试间的相互干扰。例如服务器区、测试操作区，还可以按压力测试区、兼容性测试区等进行划分。在一些专业测试公司，还为一些特殊客户设置专门的测试实验室或工作区，只有实际测试人员和特殊客户的人员可以进出。

**防火和安全：**每一个工作区都应该装有烟火自动报警器。为了控制火灾，测试实验室必须备有可随时使用的手提灭火器，而且还是带电安全的和高效的。如果实验室装有大

计算机或电燃机，就需要配备能扑灭大型电力火灾的设备。图纸中应该标出自动火灾报警器和灭火设备，不能被其他物体挡住了这些灭火设备。

**电源：**实验室中必须提供电源，12V、120V、220V、360V 等根据需要提供。并且保证这些可用的电源安置在正确的位置。为保证输入电源的电压波动不影响测试，可能须配备稳压电源，为保证电源的不中断，可能要配备 UPS 不间断电源，为保证停电和电力不足时的用电还得配备备用电源。平面图中应显示所有的电源布置情况。

**静电：**实验室是容易产生静电的地方，应该配防静电的垫子和其他接地金属物，用以帮助测试人员释放身上积累的静电。否则静电也可能会造成精密设备的损坏，如果要在实验室中铺设地毯，就需要特别注意预防静电。

**设施：**为员工准备的配套设施是非常重要的，例如洗手间、楼梯、电梯、空调、外部电话线、网络电缆（宽带）等，因为这不仅仅是法律义务，而且还影响到小组的工作效率。在图纸中应该标明所有的设施和连接。

根据上述的各种因素，综合考虑规划一个较完美的实验室环境，让测试人员在一个舒适的环境中享受工作的乐趣，避免因环境问题带来的困扰。

### 13.3.3 集成和配置测试设备

运作测试实验室需要哪些设备呢？每个测试实验室都有不同的需求，而且任何特定设备的重要性依赖于它所支持的特定测试的重要性。在为公司建立一个测试实验室时，首先要理出一个设备清单，然后进行预算评估，接着采购设备，最后是集成安装。

#### 1. 测试实验室配置清单

运用 13.2 节的知识有助于给出合理的配置清单。第一，产品的使用环境决定了测试环境，要尽可能地模拟真实的用户使用环境。第二，将主测试环境与辅测试环境结合在一起考虑，要保证主测试环境的完整性和正确性，辅测试环境要充分利用主测试环境中的资源。第三，测试环境的各要素也就是实验室配置清单的主要内容。

对于软件和硬件项目来说，由于受市场营销和销售人员针对目标客户所做的计划，技术支持人员对于当前客户使用情况的汇报，测试小组怎样最有可能发现错误的想法，支持结构的设计和描述等的驱动，所以只有在考虑上述因素的基础上，才能决定购买哪些物品。如果基本上只关注功能测试（黑盒测试），试着整理出一个实验室所需东西的清单就比较简单，一个标准的清单模板应包括以下各项。

- 软件

- ◆ 操作系统：需要所有具有广泛代表性的重要操作系统。
- ◆ 应用程序：测试人员在测试平台上使用大量的应用软件来做兼容性测试。
- ◆ 测试工具和实用工具：引入自动化测试，选择适合的测试工具。用于诊断问题的实用工具（例如测量系统的性能、监控网络流量、跟踪记录出错信息、备份关键的数据的工具等）。
- ◆ 第三方软件：软件开发过程中，有时会直接购买使用第三方的软件产品或使

用许可。这类软件在开发和测试过程中通常可以通过签订协议的方式获得。

- 硬件
  - ◆ 计算机、服务器：详细到具体配置。
  - ◆ 输入、输出设备：监视器、打印机、扫描仪等，详细到具体型号或性能指标。
  - ◆ 数据备份、存储设备：外接硬盘、光盘刻录机等。
- 网络设备

为了避免在公司的网络上进行测试，需要在测试实验室建立一个独立的网络环境。集线器和交换机价格相对较便宜，但是应该保证能够得到可靠的带宽。电缆这样相对较小的物件花费却相对较高，很容易忽略它的重要性，由于电缆引起的故障却相当难以判断而且耗费大量的时间。

- 电源及特殊的工具

浪涌电压保护器、不间断电源、示波器，工具箱等。

- 其他

办公用品、耗材和易损备件等。笔记本、纸张、笔、胶带、墨盒等看似很小的东西往往给测试人员带来很大的麻烦，例如为寻找打印纸浪费时间会破坏了工作时的良好心情。

## 2. 集成和配置测试设备

有些情况下，测试实验室配置清单中物件数量是非常巨大的。如果测试一个像视频游戏这样的应用程序，的确需要相当多的支持配置。各种类型的声卡、视频卡、控制手柄等全部罗列在配置清单中好像不太合适，有可能会超出预算。而与配置相关的遗漏又往往造成风险，导致与性能、可靠性、数据性质、用户界面和操作等其他内容有关的错误出现。所以有必要对测试配置清单进行评估，保留主要的和必须购买的配置，有些配置可以通过其他途径获得，例如，从合作单位或部门借用，从生产厂家或经销商处借来试用等。

经过评估后可以提交申请，获得批准后开始采购。采购时首先按安装的先后顺序分批购买，当一大堆设备同时运到而又不能及时安装时，就得另外出一笔费用找临时仓库。其次，得按测试进度的需要分批购买，保证主要测试设备的经费，避免因配置变化或市场变化造成资金问题。最后考虑批量定货，使一些小的配置能以赠品的方式配置。

集成和配置测试设备的通常做法是能由测试人员完成的尽可能由测试人员自己完成，只有测试人员最清楚具体的配置方法和要求。测试人员无法独立完成的需要监督、指导专业人员完成。例如网络布线，是明线还是暗线，信息点具体位置等，如果测试中需要调整网络结构，应采取测试人员更容易接受的方案。

# 13.4 测试环境的维护和管理

实验室通常会设置管理员，管理员的职责就是维护和管理测试环境。

实验室设备分为消耗性和耐久性两种。消耗性物品不需要严格的管理，试图跟踪一支笔的消耗过程是非常愚蠢的，消耗性物品的管理主要表现在保障供给，杜绝浪费。耐久性

物品需要严格的管理，主要表现在使用标签，进行登记、跟踪，调度使用，特别是一些便携式设备（例如笔记本电脑）需落实到具体的使用人员，而有些设备必须限制移动或借用，防止丢失和损坏。

测试环境的维护不仅仅是管理员的职责，也是每个测试人员的职责。维护的概念不仅包括硬件设备的保养维修，更重要的是维护测试环境的正确性，何时需要更新操作系统，何时需要软件版本升级，何时需要调整网络结构，只有测试人员真正了解需求，环境的正确与否直接影响测试结果。

聪明的管理员都会制定严格的规章制度，并严格地执行。利用制度来约束测试人员，利用制度排除外界因素的干扰，保证测试环境的安全和稳定。例如某家测试公司，它的大多数客户都有自己的独立实验室，要求所有人都必须用卡片钥匙才能进入实验室。除了每个测试人员都需要配有一把卡片钥匙之外，在每个上锁的房间中都放一台打印机来记录持卡人进入测试实验室的情况。只有那些为客户项目工作的员工才能持有进入客户实验室的卡片钥匙。

## 小结

本章从测试环境对测试的影响论述了测试环境的重要性，介绍了使用环境、测试环境、主测试环境、辅测试环境等概念，重点阐述了测试环境的各要素，就如何建立测试实验室进行了系统、详细的说明。

## 思考题

1. 什么是测试环境？为什么测试环境是测试的基础？
2. 测试环境中有哪些基本要素？
3. 如何进行产品说明书的验证？
4. 建立测试实验室的基本步骤是什么？
5. 试写一份实验室管理制度。

# 第 14 章 软件测试用例的设计

测试用例是为了实现测试有效性的一种常用工具，好的测试用例可以在测试过程中重复利用。同时，在测试过程中可以通过对测试用例的组织 and 跟踪来完成对测试工作的量化和管理。本章将从软件测试实践中一些常用的测试用例组织和编写角度来阐述如何设计测试用例。

## 14.1 测试用例设计概述

测试用例是为了特定目的（如考察特定程序路径或验证是否符合特定的需求）而设计的测试数据及与之相关的测试规程的一个特定的集合，或称为有效地发现软件缺陷的最小测试执行单元。测试用例在测试中具有重要的作用，测试用例拥有特定的书写标准，在设计测试用例时需要考虑一系列的因素，并遵循一些基本的原则。

### 14.1.1 测试用例的重要性

前面的章节中，提到在测试计划和测试过程中需要使用测试用例。为什么需要测试用例呢？在测试过程中使用测试用例具有几个方面的作用。

- 有效性：测试用例是测试人员测试过程中的重要参考依据。不同的测试人员根据相同的测试用例所得到的输出应该是一致的，对于准确的测试用例的计划、执行和跟踪是测试的有效性的有力证明。
- 可复用性：良好的测试用例具有重复使用的功能，使得测试过程事半功倍，在前面的章节中，我们提到测试是不可能进行穷举测试的，因此，设计良好的测试用例将大大节约时间，提高测试效率。
- 易组织性：即使是很小的项目，也可能会有几千甚至更多的测试用例，测试用例可能在数月甚至几年的测试过程中被创建和使用，正确的测试计划会很好地组织这些测试用例并提供给测试人员或者其他项目的人参考和有效的使用。
- 可评估性：从测试的项目管理角度来说，测试用例的通过率是检验代码质量的保证。经常说代码的质量不高或者代码的质量很好，量化的标准应该是测试用例的通过率和软件错误（bug）的数目。
- 可管理性：测试用例也可以作为检验测试人员进度、工作量以及跟踪/管理测试人员的工作效率的因素，尤其是比较适用于对于新的测试人员的检验，从而更加合理做出测试安排和计划。

因此，测试用例将会使得测试的成本降低，并具有可重复使用功能，也是作为检测测



试效果的重要因素,设计良好的测试用例将事半功半。

测试用例不是每个人都可以编写的,它需要撰写者对产品的设计、功能规格说明书、用户场景以及程序/模块的结构都有比较透彻的了解。测试人员一开始只能执行别人写好的测试案例,随着项目的进度以及测试人员的成熟,测试人员很快能自己编写测试用例,并可以提供给别人使用。

### 14.1.2 测试用例设计书写标准

在编写测试用例过程中,需要参考和规范一些基本的测试用例编写标准,在 ANSI/IEEE 829-1983 标准中列出了和测试设计相关的测试用例编写规范和模板。标准模板中主要元素如下。

- 标识符 (identification): 每个测试用例应该有一个惟一的标识符,它将成为所有和测试用例相关的文档/表格引用和参考的基本元素,这些文档/表格包括设计规格说明书、测试日志表、测试报告等。
- 测试项 (test item): 测试用例应该准确地描述所需要测试的项及其特征,测试项应该比测试设计说明中所列出的特性描述更加具体,例如做 Windows 计算器应用程序的窗口测试,测试对象是整个的应用程序用户界面,这样测试项就应该是应用程序的界面的特性要求,例如窗口缩放测试、界面布局、菜单等。
- 测试环境要求 (test environment): 用来表征执行该测试用例需要的测试环境,一般来说,在整个的测试模块里面应该包含整个的测试环境的特殊需求,而单个测试用例的测试环境需要表征该测试用例所单独需要的特殊环境需求。
- 输入标准 (input criteria): 用来执行测试用例的输入需求。这些输入可能包括数据、文件,或者操作 (例如鼠标的左键单击,键盘的按键处理等),必要的时候,相关的数据库、文件也必须被罗列。
- 输出标准 (output criteria): 标识按照指定的环境和输入标准得到的期望输出结果。如果可能的话,尽量提供适当的系统规格说明来证明期望的结果。
- 测试用例之间的关联: 用来标识该测试用例与其他的测试 (或其他测试用例) 之间的依赖关系。在测试的实际过程中,很多的测试用例并不是单独存在的,它们之间可能有某种依赖关系,例如,用例 A 需要基于 B 的测试结果正确的基础上才能进行,此时需要在 A 的测试用例中表明对 B 的依赖性,从而保证测试用例的严谨性。

综上所述,如果使用一个数据库的表来表征测试用例的话,它应该有以下的格式:

这样的结构,可以在组织和跟踪测试用例中使用,在本章的最后一节中将对测试用例的组织和跟踪进行详细的讨论。

如果用数据词典的表示方法,测试用例可以简单地表示成:测试用例 = {输入数据 + 期望结果},其中 {} 表示重复。这个式子还表明,每一个完整的测试用例不仅包含有被测程序的输入数据,而且还包括用这组数据执行被测程序后预期的输出结果。

接下来,用一个具体的例子来描述测试用例的组成结构如表 14-1 所示。例如,要对 Windows 记事本程序进行测试,选取其中的一个测试项——文件菜单栏的测试。

表 14-1 测试用例的组成

字段名称	类型	是否必选	注 释
标识符	整型	是	惟一标识该测试用例的值
测试项	字符型	是	测试的对象
测试环境要求	字符型	否	可能在整个模块里面使用相同的测试环境需求
输入标准	字符型	是	
输出标准	字符型	是	
测试用例间的关联	字符型	否	并非所有的测试用例之间都需要关联

测试对象：记事本程序文件菜单栏（测试用例标识 1000，下同），所包含的子测试用例描述如下：

- |-----文件/新建（1001）
- |-----文件/打开（1002）
- |-----文件/保存（1003）
- |-----文件/另存（1004）
- |-----文件/页面设置（1005）
- |-----文件/打印（1006）
- |-----文件/退出（1007）
- |-----菜单布局（1008）
- |-----快捷键（1009）

选取其中的一个子测试用例——文件/退出（1007）作为例子，测试用例如表 14-2 所示：

表 14-2

字段名称	描 述
标识符	1001
测试项	记事本程序，“文件”菜单栏中的“文件” “退出”命令的功能测试
测试环境要求	Window 2000 Professional 中文版
输入标准	(1) 打开 Windows 记事本程序，不输入任何字符，单击“文件” “退出”命令 (2) 打开 Windows 记事本程序，输入一些字符，不保存文件，单击“文件” “退出”命令 (3) 打开 Windows 记事本程序，输入一些字符，保存文件，单击“文件” “退出”命令 (4) 打开一个 Windows 记事本文件（扩展名为.txt），不做任何修改，单击“文件” “退出”命令 (5) 打开一个 Windows 记事本文件，修改后不保存，单击“文件” “退出”命令
输出标准	(1) 记事本未做修改，单击“文件” “退出”命令，能正确地退出应用程序，无提示信息 (2) 记事本做修改未保存或者另存，单击“文件” “退出”命令，会提示“未定标题文件的文字已经改变，想保存文件吗？”单击“是”，Windows 将打开“保存”/“另存为”对话框，单击“否”，文件将不被保存并退出记事本程序，单击“取消”将返回记事本窗口
测试用例间的关联	1009（快捷键测试）

通过这个例子了解了测试用例的组成方法。要组织成一个完整的良好测试用例，还需要更多的技巧，并要考虑一些常见的因素。

### 14.1.3 测试用例设计考虑因素

测试是不可能实现穷举测试的，因此试图用所有的测试用例来覆盖所有测试可能遇到的情形是不可能的，所以，在测试用例的编写、组织过程中，尽量考虑有代表性的典型的测试用例，来实现以点带面的穷举测试。这要求在测试用例设计中考虑一些基本因素：

- 测试用例必须具有代表性、典型性。一个测试用例能基本涵盖一组或者多组情形，也是测试用例设计的初衷，这涉及到测试用例的设计方法，在后面的章节中，我们将对这个问题，以常见的白盒和黑盒测试用例为例子，进行详细的阐述。
- 测试用例设计时，要浓缩系统设计。测试用例需要很确切地反映功能设计，但同时最好不要完全地复制使用规格设计说明书。同时，测试用例还需要结合多个规格说明书要求进行设计，将所有的可能的情况结合起来考虑。下面的例子是一个常见的 Web 登录页面，通过这个例子来阐述从功能规格说明书到具体的测试用例编写的整个过程。

#### a) 用户登录的功能设计规格说明书（摘选）

##### 1. 用户登录

1.1 满足基本页面布局图示（登录页面图，此处略去）。

1.2 当用户没有输入用户名和密码时，不立即弹出错误对话框，而是在页面上使用红色字体来提示，见 2 描述。

1.3 用户密码使用掩码符号（\*）来标识。

1.4 \*代表必选字段，将出现在输入文本框的后面。

##### 2. 登录出现错误

当出现错误时，在页面的顶部会出现相应的错误提示。错误提示的内容见 3。错误提示是高亮的红色字体实现。

##### 3. 错误信息描述

###### 3.1 用户名输入为空

属性	值
编号	MSG0001
显示的页面	ErrorPage0001
出现条件	当用户输入的用户名为空而试图登录
提示信息	错误： 请输入用户名

###### 3.2 密码为空

属性	值
编号	MSG0002
显示的页面	ErrorPage0002
出现条件	当用户密码输入为空且没有出现 WMSG001 的提示信息
提示信息	错误：请输入密码

### 3.3 用户名/密码不匹配

属性	值
编号	MSG0003
显示的页面	ErrorPage0003
出现条件	当用户名和密码不匹配时
提示信息	错误：您输入的用户名或者密码不正确

(注：本例子中的页面图示，消息编号如 MSG001 的描述均未给出。)

#### b) 通用安全性设计规格说明书（摘选）

##### 1. 安全性描述

1.1 输入安全性：在用户登录或者信用卡验证过程中，如果三次输入不正确，页面将需要重新打开才能生效。

1.2 密码：在所有的用户密码中，都必须使用掩码符号（\*），数据在数据库中存储使用统一的加密和解密算法。

1.3 Cookie：在信用卡信息验证，用户名输入时，Cookie 都是被禁止的，当用户第一次输入后，浏览器将不再提供是否保存信息的提示信息，自动完成功能将被禁用。

1.4 SSL 校验：所有的站点访问时，必须经过 SSL 校验。

##### 2. 错误描述：（略）

#### c) 测试用例

结合相关规格说明书要求，理解和掌握测试用例设计的关键点，测试用例设计如表 14-3 所示。

- 测试用例需要考虑到正确的输入，也需要考虑错误的或者异常的输入，以及需要分析怎样使得这样的错误或者异常能够发生。

例如，在进行电子邮件地址校验的时候，不仅需要考虑到正确的电子邮件地址（如 pass@hf.webex.com）的输入，同时需要考虑错误的、不合法的（如没有@符号的输入）或者带有异常字符（单引号、斜杠、双引号等）的电子邮件地址输入，尤其是在做 Web 页面测试的时候，通常会出现一些字符转义问题而造成异常情况的发生。

在前面提到的用户登录页面的例子中，需要考虑特殊字符的输入，尤其是脚本语言敏感的字符输入。根据这个原则，将上面的测试用例进行完善（用粗体标出），如表 14-4 所示。

表 14-3 用户登录功能测试用例

字段名称	描 述
标识符	1100
测试项	站点用户登录功能测试
测试环境要求	(1) 用户 test/pass 为有效登录用户, 用户 test1 为无效登录用户 (2) 浏览器的 Cookie 未被禁用
输入标准	(1) 输入正确的用户名和密码, 单击“登录”按钮 (2) 输入错误的用户名和密码, 单击“登录”按钮 (3) 不输入用户名和密码, 单击“登录”按钮 (4) 输入正确的用户并不输入密码, 单击“登录”按钮 (5) 三次输入无效的用户名和密码尝试登录 (6) 第一次登录成功后, 重新打开浏览器登录, 输入上次成功登录的用户名的第一个字符
输出标准	(1) 数据库中存在的用户将能正确登录 (2) 错误的或者无效用户登录失败, 并在页面的顶部出现红色字体: “错误: 用户名或密码输入错误” (3) 用户名为空时, 页面顶部出现红色字体提示: “请输入用户名” (4) 密码为空且用户名不为空时, 页面顶部出现红色字体提示: “请输入密码” (5) 三次无效登录后, 第四次尝试登录会出现提示信息“您已经三次尝试登录失败, 请重新打开浏览器进行登录”, 此后的登录过程将被禁止 (6) 自动完成功能将被禁用, 查看浏览器的 Cookie 信息, 将不会出现上次登录的用户和密码信息, 第一次使用一个新账户登录时, 浏览器将不会提示“是否记住密码以便下次使用”对话框 (7) 所有的密码均以*方式输入
测试用例间的关联	1101 (有效密码测试)

表 14-4 完善的测试用例

字段名称	描 述
标识符	1100
测试项	站点用户登录功能测试
测试环境要求	(1) 用户 pass/pass 为有效登录用户, 用户 pass1/pass 为无效登录用户, 用户 pass'jean/password 为有效登录用户 (2) 浏览器的 Cookie 未被禁用
输入标准	(1) 输入正确的用户名和密码, 单击“登录”按钮 (2) 输入错误的用户名和密码, 单击“登录”按钮 (3) 不输入用户名和密码, 单击“登录”按钮 (4) 输入正确的用户并不输入密码, 单击“登录”按钮 (5) 输入带特殊字符 (/、\、"和#, 如 pass'jean) 的用户名和密码, 单击“登录”按钮 (6) 三次输入无效的用户名和密码尝试登录 (7) 第一次登录成功后, 重新打开浏览器登录, 输入上次成功登录的用户名的第一个字符

续表

字段名称	描 述
输出标准	(1) 数据库中存在的用户 (pass/pass, pass'jean/password) 将能正确登录 (2) 错误的或者无效用户登录失败, 并在页面的顶部出现红色字体: “错误: 用户名或密码输入错误” (3) 用户名为空时, 页面顶部出现红色字体提示: “请输入用户名” (4) 密码为空且用户名不为空时, 页面顶部出现红色字体提示: “请输入密码” (5) 含特殊字符 (‘、/、”、#) 的用户名, 如数据库中有该记录, 将能正确登录; 如无该用户记录, 将不能登录。校验过程和普通的字符相同, 不能出现空白页面或者脚本错误 (6) 三次无效登录后, 第四次尝试登录会出现提示信息 “您已经三次尝试登录失败, 请重新打开浏览器进行登录”, 此后的登录过程将被禁止 (7) 自动完成功能将被禁用, 查看浏览器的 Cookie 信息, 将不会出现上次登录的用户和密码信息, 第一次使用一个新账户登录时, 浏览器将不会提示 “是否记住密码以便下次使用” 对话框 (8) 所有的密码均以*方式输入
测试用例间的关联	1101 (有效密码测试)

- 用户测试用例设计, 要多考虑用户实际使用场景。

用户测试用例是基于用户实际的可能场景, 从用户的角度来模拟程序的输入, 从而针对程序来进行测试的用例。用户测试用例不仅需要考虑用户实际的环境因素, 例如在 Web 程序中需要对用户的连接速度、负载进行模拟, 需要考虑各种网络连接方式的速度。在本地化软件测试时, 需要尊重用户的所在国家、区域的风俗、语言以及习惯用法。关于本地化语言的测试, 详见本书第 10 章的描述。

14.1.4 测试用例设计的基本原则

在测试用例设计时, 除了需要遵守基本的测试用例编写规范外, 还需要遵循一些基本的原则。

1. 尽量避免含糊的测试用例

含糊的测试用例给测试过程带来困难, 甚至会影响测试的结果。在测试过程中, 测试用例的状态是惟一的, 通常情况下, 在执行测试过程中, 良好的测试用例一般会有三种状态: 通过 (PASS)、未通过 (Failed) 以及未进行测试 (Not Done), 如果测试未通过, 一般会有测试的错误 (bug) 报告进行关联; 如未进行测试, 则需要说明原因 (测试用例本身的错误、测试用例目前不适用、环境因素等), 因此, 清晰的测试用例使测试人员在测试过程中不会出现模棱两可的情况, 不能说这个测试用例部分通过, 部分未通过, 或者是从这个测试用例描述中不能找到问题, 但软件错误应该出现在这个测试用例中。这样的测试用例将会给测试人员的判断带来困难, 同时也不利于测试过程的跟踪。

例如, 还用上面的例子来说明, 对用户登录的页面校验测试进行测试用例设计:

- 输入正确的用户和密码, 所有程序工作正常。

- 输入错误的用户和密码，程序工作不正常，并弹出对话框。

在上面这样的测试用例设计，未能清楚地描述什么样是程序正常工作状态，什么样是程序不正常工作状态，这样含糊不清的测试用例必然会导致测试过程中问题的遗漏。

## 2. 尽量将具有相类似功能的测试用例抽象并归类

一直强调软件测试过程是无法进行穷举测试的，因此，对相类似的测试用例的抽象过程显得尤为重要，一个好测试用例应该是能代表一组或者一系列的测试过程。

## 3. 尽量避免冗长和复杂的测试用例

这样做的主要目的是保证验证结果的惟一性。这也是和第一条原则相一致的，为的是在测试过程执行过程中，确保测试用例的输出状态惟一性，从而便于跟踪和管理。在一些很长和复杂的测试用例设计过程中，需要将测试用例进行合理的分解，从而保证测试用例的准确性。在某些时候，当测试用例包含很多不同类型的输入或者输出，或者测试过程的逻辑复杂而不连续，此时需要对测试用例进行分解。

在实际的测试用例设计中，需要将前述的基本原则和考虑因素结合起来，遵循基本的测试用例编写规范，按照实际测试的需求灵活地组织设计测试用例。

在测试用例设计中，主要考虑白盒测试用例和黑盒测试用例。

# 14.2 白盒测试用例设计方法

在前面的章节中提到了白盒测试是软件测试中常用的一种测试方法，白盒测试是为了测试证明每种内部操作和过程是否符合设计规格和要求，因此白盒测试是基于对程序的基本输入输出已经了解的基础上进行的程序内部逻辑结构测试，白盒测试又称为结构测试或者逻辑驱动测试或者基于程序的测试。

相对于黑盒测试，白盒测试主要是针对程序内部逻辑和数据流程的测试，而黑盒测试主要是单元的输入输出功能测试，因此，白盒测试用例的设计需要了解程序的内部逻辑，而黑盒测试用例则不需要考虑程序的实现过程。

白盒测试主要对程序模块进行以下检查：

- 对程序模块的所有独立的执行路径至少要测试一次。
- 对所有的逻辑判定，取真或假的两种情况至少要测试一次。
- 对程序进行边界检查（常见的如数据越界检验）。
- 检验内部数据结构的有效性。

在白盒测试用例设计中，主要使用两种方法：逻辑覆盖法和基本路径测试法。

## 14.2.1 逻辑覆盖法

逻辑覆盖法主要以程序内部的逻辑结构为基础的测试用例设计方法。在逻辑覆盖法中，

又可以分为语句覆盖、判定覆盖、条件覆盖、判定-条件覆盖、条件组合覆盖和路径覆盖。

### 1. 语句覆盖

语句覆盖法的基本思想是：设计若干测试用例，运行被测程序，使程序中的每个可执行语句至少执行一次。

有这样一个例子程序，用来实现一个简单的数学运算。

程序源代码：

```
Dim a, b As Integer
Dim c As Double
If (a > 0 And b > 0) Then
    c = c / a
End If
If (a > 1 Or c > 1) Then
    c = c + 1
End If
c = b + c
```

程序流程图如图 14-1(a)所示。

由这个流程图可以看出，该程序模块有 4 条不同的路径：

P1: (1-2-4)

P2: (1-2-5)

P3: (1-3-4)

P4: (1-3-5)

将里面的判定条件和过程记录为图(b)：

条件 M={a>0 and b>0}

条件 N={a>1 or c>1}

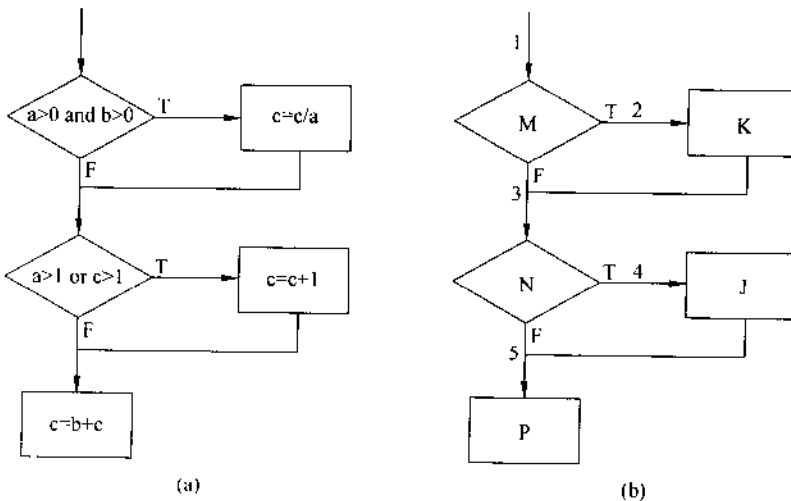


图 14-1 程序流程图

这样，程序的 4 条不同路径可以表示为：



P1 (1-2-4) =M and N

P2 (1-2-5) =/M and N (/M 表示 M 取反, 下同)

P3 (1-3-4) =M and /N

P4 (1-3-5) =/M and /N

P1 包含了所有可执行语句, 按照语句覆盖的测试用例设计原则, 可以使用 P1 来设计测试用例。令  $a=2$ ,  $b=1$ ,  $c=6$ , 此时满足条件  $M\{a>0 \text{ and } b>0\}$  和条件  $N\{a>1 \text{ or } c>1\}$  (注: 此时的  $c=c/a=3$ ), 这样, 测试用例的输入  $\{a=2, b=1, c=6\}$  和对应的输出  $\{a=2, b=1, c=10\}$  覆盖路径 P1。

在使用语句覆盖法进行测试用例设计时, 能够使所有的执行语句都能被测试, 但是不能准确地判断运算中的逻辑关系错误。在这个例子里面, 如果 M 的条件是  $a>0 \text{ or } b>0$ , 而不是 and 关系, 这时的测试用例仍然可以覆盖所有可执行语句, 但不能发现其中的逻辑错误。如下所示:

Dim a, b As Integer	Dim a, b As Integer
Dim c As Double	Dim c As Double
If (a > 0 And b > 0) Then	If (a>0 Or b>0) Then ' 错误! 但测试结果相同
c = c / a	c = c / a
End If	End If
If (a>1 and c>1) Then	If (a>1 and c>1) Then
c=c+1	c=c+1
End If	End If
c=b+c	c=b+c

## 2. 判定覆盖

判定覆盖法的基本思想是: 设计若干用例, 运行被测程序, 使得程序中每个判断的取真分支和取假分支至少经历一次, 即判断真假值均曾被满足。

按照判定覆盖的基本思路, 可以这样针对上面提到的测试的用例进行设计, P1 和 P2 可以作为测试用例, 其中 P1 作为取真的路径, P2 作为取反的路径, 这样, 输入  $\{a=2, b=1, c=6\}$  和输出  $\{a=2, b=1, c=10\}$  覆盖路径 P1, 输入  $\{a=1, b=2, c=3\}$  和输出  $\{a=1, b=2, c=5\}$  覆盖路径 P2。

判定覆盖设计测试用例时会忽略条件中取“或”(or)的情况, 例如, 在上面的例子中, 如果条件 N 中是  $c<1$  而不是  $c>1$ , 同样能得到相同的结果。

Dim a, b As Integer	Dim a, b As Integer
Dim c As Double	Dim c As Double
If (a > 0 And b > 0) Then	If (a > 0 And b > 0) Then
c = c / a	c = c / a
End If	End If
If (a>1 and c>1) Then	If (a>1 Or c>1) Then' 错误! 但测试结果相同
c=c+1	c=c+1
End If	End If

c=b+c

c=b+c

3. 条件覆盖

条件覆盖的基本思想是：设计若干测试用例，执行被测程序以后要使每个判断中每个条件的可能取值至少满足一次。

对于第一个判定条件 M，可以分割如下：

条件 a>0：取真（TRUE）时为 T1，取假（FALSE）时为 F1；

条件 b>0：取真（TRUE）时为 T2，取假（FALSE）时为 F2。

对于第二个判定条件 N，

条件 a>1：取真（TRUE）时为 T3，取假（FALSE）时为 F3；

条件 c>1：取真（TRUE）时为 T4，取假（FALSE）时为 F4。

根据条件覆盖的基本思想，和这 8 个条件取值，组合成测试用例，如表 14-5 所示。

表 14-5 采用条件覆盖设计的测试用例

测试用例	取值条件	具体取值条件	通过路径
输入：a=2，b=-1，c=-2 输出：a=2，b=-1，c=-3	T1，F2，T3，F4	a>0，b<=0，a>1，c<=1	P3（1-3-4）
输入：a=-1，b=2，c=3 输出：a=-1，b=2，c=6	F1，T2，F3，T4	a<=0，b>0，a<=1，c>1	P4（1-3-5）

在这个例子中，要涵盖所有的条件组合，保证每个条件的取真取假都能至少运行一次的测试用例设计还有好几种，这里不再描述。

在这种测试用例设计方法中，可以看到，测试用例不一定能满足前面提到的判定覆盖的要求，即判定条件 M 和 N 的真假没有至少被执行一次，同样，这样可能会造成程序逻辑错误被遗漏。

4. 判定-条件覆盖

判定-条件覆盖实际上是将前两种方法结合起来的一种设计方法，它是判定和条件覆盖设计方法的交集，即设计足够的测试用例，使得判断条件中的所有条件可能至少执行一次取值，同时，所有判断的可能结果至少执行一次。

按照这种思想，结合前面的方法思路，在前面的例子中，应该至少保证判定条件 M 和 N 各取真/假一次，同时要保证 8 个条件取值（T1，F1，T2，F2，…，F4）至少执行一次，如表 14-6 所示。

表 14-6 采用判定-条件覆盖设计的测试用例

测试用例	取值条件	具体取值条件	判定条件	通过路径
输入：a=2，b=1，c=6 输出：a=2，b=1，c=5	T1，T2，T3，T4	a>0，b>0，a>1，c>1	M，N	P1（1-2-4）
输入：a=-1，b=-2，c=-3 输出：a=-1，b=-2，c=-5	F1，F2，F3，F4	a<=0，b<=0，a<=1，c<=1	/M，/N	P3（1-3-5）

### 5. 条件组合覆盖

条件组合覆盖的基本思想是：设计足够的测试用例，使得判断中每个条件的所有可能至少出现一次，并且每个判断本身的判定结果也至少出现一次，与条件覆盖的差别是它不是简单地要求每个条件都出现“真”与“假”两种结果，而是要求让这些结果的所有可能组合都至少出现一次。

按照条件组合覆盖的基本思想，对于前面的例子，设计组合条件如表 14-7 所示。

表 14-7 采用条件组合覆盖设计的测试用例

组合编号	覆盖条件取值	判定条件取值	判定-条件组合
1	T1, T2	M	$a > 0$ , $b > 0$ , M 取真
2	T1, F2	/M	$a > 0$ , $b \leq 0$ , M 取假
3	F1, T2	/M	$a \leq 0$ , $b > 0$ , M 取假
4	F1, F2	/M	$a \leq 0$ , $b \leq 0$ , M 取假
5	T3, T4	N	$a > 1$ , $c > 1$ , N 取真
6	T3, F4	/N	$a > 1$ , $c \leq 1$ , N 取假
7	F3, T4	/N	$a \leq 1$ , $c > 1$ , N 取假
8	F3, F4	/N	$a \leq 1$ , $c \leq 1$ , N 取假

针对 8 种组合条件，来设计所有能覆盖这些组合的设计用例，如表 14-8 所示。

表 14-8 采用 8 种组合条件设计的测试用例

测试用例	覆盖条件	覆盖路径	覆盖组合
输入：a=2, b=1, c=6 输出：a=2, b=1, c=5	T1, T2, T3, T4	P1 (1-2-4)	1, 5
输入：a=2, b=-1, c=-2 输出：a=2, b=-1, c=-3	T1, F2, T3, F4	P3 (1-3-4)	2, 6
输入：a=-1, b=2, c=3 输出：a=-1, b=2, c=6	F1, T2, F3, T4	P3 (1-3-4)	3, 7
输入：a=-1, b=-2, c=-3 输出：a=-1, b=-2, c=-5	F1, F2, F3, F4	P4 (1-3-5)	4, 8

从上面的表格可以看出，条件组合覆盖事实上是将覆盖条件组合成满足条件的判定条件，同时保证判定条件的所有取值至少执行一次。

条件组合覆盖设计方法也有缺陷，从上面的测试用例中可以看到，所有的条件覆盖组合不能保证所有的路径被执行（P2 (1-2-5) 没有被执行）。

### 6. 路径覆盖

顾名思义，路径覆盖就是设计所有的测试用例，来覆盖程序中的所有可能的执行路径。

这样可以采用下面的测试用例来完全覆盖路径 P1、P2、P3、P4，如表 14-9 所示。

表 14-9 采用路径覆盖设计的测试用例

测试用例	覆盖路径	覆盖条件	覆盖组合
输入：a=2，b=1，c=6 输出：a=2，b=1，c=5	P1 (1-2-4)	T1，T2，T3，T4	1，5
输入：a=1，b=1，c=-3 输出：a=1，b=1，c=-2	P2 (1-2-5)	T1，T2，F3，F4	1，8
输入：a=-1，b=2，c=3 输出：a=-1，b=2，c=6	P3 (1-3-4)	F1，F2，F3，T4	4，7
输入：a=-1，b=-2，c=-3 输出：a=-1，b=-2，c=-5	P4 (1-3-5)	F1，F2，F3，F4	4，8

同样，路径覆盖法没有涵盖所有的条件覆盖组合，如组合 2，3，6。

通过前面例子可以看到，采用其中任何一种方法都不能完全覆盖所有的测试用例，因此，在实际的测试用例设计过程中，可以根据需要和不同的测试用例设计特征，将不同的设计方法组合起来，交叉使用，以实现最佳的测试用例输出。

采用条件组合和路径覆盖两种方法的结合来重新设计测试用例如下，这个测试用例就能覆盖该例子程序的所有测试，如表 14-10 所示。

表 14-10 采用条件组合和路径覆盖设计的测试用例

测试用例	覆盖路径	覆盖条件	覆盖组合
输入：a=2，b=1，c=6 输出：a=2，b=1，c=5	P1 (1-2-4)	T1，T2，T3，T4	1，5
输入：a=1，b=1，c=-3 输出：a=1，b=1，c=-2	P2 (1-2-5)	T1，T2，F3，F4	1，8
输入：a=-1，b=2，c=3 输出：a=-1，b=2，c=6	P3 (1-3-4)	F1，F2，F3，T4	4，7
输入：a=-1，b=-2，c=-3 输出：a=-1，b=-2，c=-5	P4 (1-3-5)	F1，F2，F3，F4	4，8
输入：a=2，b=-1，c=-2 输出：a=2，b=-1，c=-3	P3 (1-3-4)	T1，F2，T3，F4	2，6
输入：a=-1，b=2，c=3 输出：a=-1，b=2，c=6	P4 (1-3-4)	F1，T2，F3，T4	3，7

14.2.2 基本路径测试法

基本路径测试法是在程序控制流图的基础上，通过分析控制构造的环路复杂性导出基本可执行路径集合，从而设计测试用例的方法。设计出的测试用例要保证在测试中程序的

每个可执行语句至少执行一次。基本路径测试法包括以下 5 个方面：

- 程序的流程图：程序流程控制图描述程序控制流的一种图示方法，可以使用流程图软件，如 Microsoft 的 Visio 来实现程序流程的描述。
- 计算程序环境复杂性：通过对程序的控制流程图的分析和判断来计算模块复杂性度量。从程序的环路复杂性可导出程序基本路径集中的独立路径条数，这是确定程序中每个可执行语句至少执行依次所必需的测试用例数目的上界。
- 导出测试用例：通过程序流程图的基本路径来导出基本的程序路径的集合，这个过程和前面提到的逻辑覆盖法设计相类似。
- 准备测试用例：确保基本路径集中的每一条路径的执行。
- 图形矩阵：是在基本路径测试中起辅助作用的软件工具，利用它可以实现自动地确定一个基本路径集。

## 14.3 黑盒测试用例设计方法

黑盒测试也叫功能测试或者数据驱动测试，它主要是基于功能规格说明进行的测试，相对于白盒测试而言，黑盒测试的目的主要是为了发现以下几种类型的错误：

- 模块中是否有功能遗漏或者逻辑错误。
- 模块接口是否存在问题。
- 是否有数据结构错误或者外部信息访问错误。
- 性能上是否满足要求。

黑盒测试用例设计主要有 5 种方法：

- 等价类划分法
- 边界值分析法
- 错误推测法
- 因果图法
- 功能图法

在接下来的小节中将逐个讨论各种设计方法的使用。

### 14.3.1 等价类划分法

前面提到测试是不能穷举的。等价类划分法是黑盒测试用例设计中一种重要的、常用的设计方法，它将不能穷举的测试过程进行合理分类，从而保证设计出来的测试用例具有完整性和代表性。

等价类划分法是把所有可能的输入数据，即程序的输入域划分成若干部分（子集），然后从每一个子集中选取少数具有代表性的数据作为测试用例。在等价类划分法设计测试用例的过程中，需要使用两个过程：分类和抽象。第一个过程是分类，即将输入域按照具有相同特性或者类似功能进行分类；第二个过程抽象，即在各个子类中抽象出相同特性并用

实例来表征这个特性。

等价类是指某个输入域的子集合，在该子集合中，各个输入数据对于揭露程序中的错误都是等效的，它们具有等价特性，这样，对于表征该类的数据输入将能代表整个子集合的输入。因此，可以合理地假定：测试某等价类的代表值就是等效于对于这一类其他值的测试。举个例子：设计这样的测试用例，来实现一个对所有的实数进行开方运算的程序的测试，这时候需要将所有的实数（输入域）进行划分，可以分成：正实数、负实数和 0，使用+1.4444 代表正实数，用-2.345 代表负实数，输入的等价类就可以使用+1.4444、-2.345 和 0 来表示。

在进行等价类划分的过程中，不但要考虑有效等价类划分，同时需要考虑无效等价类划分。有效等价类和无效等价类定义如下：

- 有效等价类：是指输入完全满足程序输入的规格说明，有效、有意义的输入数据所构成的集合。利用有效等价类可以检验程序是否满足规格说明所规定的功能和性能。
- 无效等价类：和有效等价类相反，即不满足程序输入要求或者无效的输入数据构成的集合。使用无效等价类，可以鉴别程序异常情况的处理。在程序设计中，不但要保证所有有效的数据输入能产生正确的输出，同时需要保障在输入错误或者空输入的时候能有异常保护，这样的测试才能保证软件的可靠性。

基于上面的描述，在等价类划分时，使用以下几个原则：

- 在输入条件规定了取值范围或者个数的前提下，可以确定一个有效等价类和两个无效等价类。例如：程序输入条件为满足小于 100 大于 10 的整数  $x$ ，则有效等价类为  $10 < x < 100$ ，两个无效等价类为  $x < 10$  和  $x > 100$ 。
- 在输入条件规定了输入值的集合或者规定了“必须如何”的条件下，可以确定一个有效等价类和一个无效等价类。例如：程序输入条件为  $x=10$ ，则有效等价类为  $x=10$ ，无效等价类为  $x \neq 10$ 。
- 在输入条件是一个布尔量的情况下，可确定一个有效等价类和一个无效等价类。例如：程序输入条件为 `BOOL x=true`，则有效等价类为  $x=true$ ，无效等价类为  $x=false$ 。
- 在给定了一组输入数据（包括  $n$  个输入值），并且程序要对每一个输入值分别进行处理的情况下，可确定  $n$  个有效等价类和一个无效等价类。例如：程序输入条件为  $x$  取值于一个固定的枚举类型  $\{1, 3, 7, 10, 15\}$ ，则有效等价类为  $x=1, x=3, x=7, x=10, x=15$ ，而程序中对这 5 个数值分别进行了处理，对于任何其他数值使用默认 Default 处理方式，此时无效等价类为  $x \neq 1, 3, 7, 10, 15$  的值的集合。
- 在规定了输入数据必须遵守的规则的情况下，可确定一个有效等价类和若干个无效等价类，例如输入是页面上用户输入有效 E-mail 地址的规则，必须满足几个条件，含有 @，@后面格式为  $x.x$ ，E-mail 地址不带有特殊符号”、#、‘、&。有效等价类就是满足所有条件的输入的集合，无效等价类就是不满足其中任何一个条件或者所有条件的输入的集合。
- 在确定已知的等价类中各元素在程序处理中的方式不同的情况下，则应再将该等

价类进一步划分为更小的等价类。

在使用等价类做测试用例设计时，按照划分好的等价类，

- 为每个等价类规定一个唯一的编号。
- 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这个过程，直至所有的有效等价类都被覆盖，即分割有效等价类直到最小。
- 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的无效等价类，重复这个过程，直至所有的无效等价类都被覆盖，即分割无效等价类直到最小。

### 14.3.2 边界值分析法

边界值分析法是对等价类划分法的补充。

顾名思义，边界值分析法是对输入的边界值进行测试，在测试过程中，可能会忽略边界值的条件，大量的错误是发生在输入或输出范围的边界上，而不是发生在输入输出范围的内部，如做一个除法运算的例子，如果测试者忽略被除数为 0 的情况就会导致问题的遗漏。因此，在测试用例设计中，需要对输入的条件进行分析并且汲取其中的边界值条件，通过对这些边界值的测试来查出更多的错误。

在实际的软件设计过程中，会涉及到大量的边界值条件和过程，这里有个简单的 VB 程序的小例子：

```
Dim data(10) as integer
Dim iIndex as integer
For iIndex=1 to 10
    data(iIndex)=-1
Next iIndex
```

在这个程序中，代码的目标是为了创建一个拥有 10 个元素的一维数组，看起来很合理，但是，在绝大多数 Basic 语言中，当一个有上限的数组被定义时，其第一个元素是 0 而不是 1，这样，程序中 data(10) 的元素应该是从 0 到 10 共 11 个数据，而不是 10 个。这样，当程序从 1 到 10 进行循环的时候，就忽略了数组下限为 0 的情况，而这种情况下的输入是合法的。

```
Data(0)=0, data(1)=-1, data(2)=-1,...,data(10)=-1
```

注意到 data(0) 的值是 0 而不是 -1，当别的程序员使用这个数组的时候，他可能不会想到这个数组是应该从 1 开始而选用默认的从 0 开始，这样就可能造成软件的缺陷或错误的产生。

上面提到的例子是常用的数组边界检查时遇到的。通常情况下，软件测试所包含的边界检验有几种类型：数字、字符、位置、质量、大小、速度、方位、尺寸、空间等，而相应地，这些类型的边界值应该在最大/最小，首位/末尾，上/下，最快/最慢，最高/最低，最短/最长，空/满等情况下。这时候对用户的输入和计算机软件本身的特性的边界值条件进行详细的分析和考虑就很有必要。

在边界值分析法中，最重要的工作是确定正确边界值域。一般情况下，输入和输出等

价类的边界，就是应该着重测试的边界情况。另外，一般需要选取正好等于、刚刚大于和刚刚小于边界值的数据作为测试数据，举个例子来说明，如表 14-11 所示。

表 14-11 利用边界值作为测试数据的例子

项	边 界 值	测试用例的设计思路
字符	起始-1 个字符/结束+1 个字符	假设一个文本输入区域要求允许输入 1 到 255 个字符，输入 1 个和 255 个字符作为有效等价类；输入 0 个和 256 个字符作为无效等价类，这几个数值都属于边界条件值
数值	开始位-1/结束位+1	例如软件要求数据的输入域需要输入 9 位的数据，可以使用最简单的 00000~00000 作为最小和 99999~99999 为最大值，然后刚好使用小于 9 位和大于 9 位的数值来作为边界条件
方向	刚刚超过/刚刚低于	
空间	小于空余空间一点/大于满空间一点	例如在做软盘的数据存储时候，使用比最小剩余磁盘空间大一点（几 KB）的文件作为最大值检验的边界条件

以上的例子都是基本的边界值条件，它是基于应用程序的功能设计而需要考虑的因素，从软件的规格说明书或者常识中得到，也是最终用户可以很容易发现问题的边界条件。然而，在软件测试用例设计过程中，某些边界值条件是不需要呈现给用户的，或者说用户是很难注意到的，但同时确实出现属于检验范畴内的边界值条件，称为子边界值条件或者内部边界值条件，主要有下面几种。

● 数值的边界值检验

计算机是基于二进制进行工作的，因此，软件的任何数值运算都有一定的范围限制，如表 14-12 所示。

表 14-12 计算机数值运算的范围

项	范 围 或 值
位 (bit)	0 或 1
字节 (byte)	0~255
字 (word)	0~65、535 (单字) 或 0~4、294、967、295 (双字)
千 (K)	1 024
兆 (M)	1 048 576
吉 (G)	1 073 741 824
太 (T)	1 099 511 627 776

这样，在数值的边界值条件检验中，例如对字节进行检验，边界值条件可以设置成 254、255 和 256。

● 字符的边界值检验

在计算机软件中，字符也是很重要的表示元素，其中 ASCII 和 Unicode 是常见的编码方式，在表 14-13 中列出了一些简单的 ASCII 码对应表。



表 14-13 字符的 ASCII 码对应表

字 符	ASCII 码值	字 符	ASCII 码值
空 (null)	0	A	65
空格 (space)	32	a	97
斜杠 (/)	47	左中括号 ([)	91
0	48	z	122
冒号 (:) )	58	Z	90
@	64	单引号 (')	96

在做文本输入或者文本转换的测试过程中，需要非常清晰地了解 ASCII 码的一些基本对应关系，例如小写字母 a 和大写字母 A 在表中的对应是不同的，这些也必须被考虑到数据区域划分的过程中，从而根据这些定义等价有效类来设计测试用例。

- 其他边界值检验

几种情况默认值/空值/空格/未输入值/零、无效数据/不正确数据和干扰（垃圾）数据等。

在实际的测试用例设计中，需要将基本的软件设计要求和程序定义的要求结合起来，即结合基本边界值条件和子边界值条件来设计有效的测试用例。

### 14.3.3 因果图法

前面介绍的等价分类法和边界分析法等都没有考虑到输入情况的组合。这样虽然各种输入条件可能出错的情况已经看到了，但多个输入情况组合起来可能出错的情况却被忽视了。检验各种输入条件的组合并非一件很容易的事情，因为即使将所有的输入条件划分成等价类，它们之间的组合情况也相当复杂，因此，必须需要考虑采用一种适合于多种条件的组合，相应能产生多个动作的形式来进行测试用例的设计，这就需要采用因果图法。

因果图法就是一种利用图解法分析输入的各种组合情况，从而设计测试用例的方法，它适合于检查程序输入条件的各种情况的组合。因果图法最终生成的是判定表。

由因果图法怎样生成测试用例呢？如图 14-2 所示。

(1) 分析软件规格说明书中的输入输出条件并分析出等价类，将每个输入输出赋予一个标志符。分析规格说明中的语义，通过这些语义来找出相对应的输入与输入之间，输入与输出之间关系。

(2) 将对应的输入输出之间，输入与输出之间的关系关联起来，并将其中不可能的组合情况标注成约束或者限制条件，形成因果图。

(3) 由因果图转化成判定表。

(4) 将判定表的每一列拿出来作为依据，设计测试用例。

### 14.3.4 错误推测法

推测是一种思维方式，在软件的测试用例设计中，它主要是依赖经验、直觉和简单的判断，来推测程序中可能存在的各种错误，从而有针对性地设计测试用例。

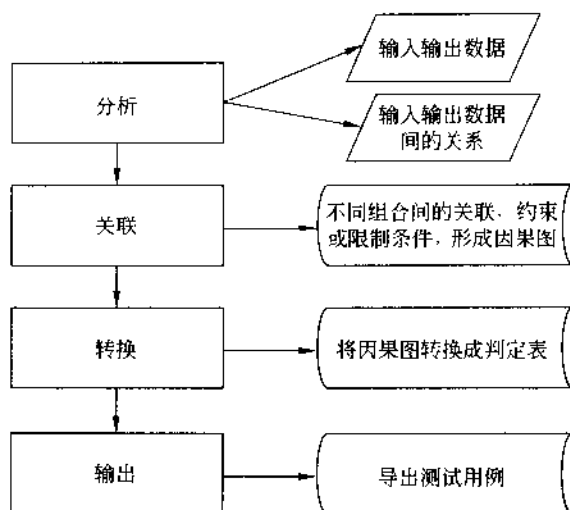


图 14-2 因果图法示例

错误推测方法的基本思想是：列举程序中所有可能出现的错误和容易发现错误的地方，根据它们来选择和设计测试用例。可以利用不同测试阶段的经验和对软件系统的认识来进行测试用例的设计，例如，在单元测试中程序模块已经遇到的错误，可以在系统测试中为这些可能出现问题的地方组织测试用例。在前一个版本中发现的常见的错误在下一个版本测试中有针对性地设计测试用例。在应用软件中可能出错的环节，例如 C++ 软件的内存分配、内存泄漏、Web 程序的 session 失效问题、JavaScript 字符转义等一些常见的普遍的问题，选择性地设计测试用例。

综上所述，在错误推测法中，通常依据下列因素来进行判断和设计测试用例。

- 客观因素：产品先前版本的问题，衰减测试。
- 已知因素：语言、操作系统、浏览器的限制可能带来的问题。
- 经验：由模块之间的关联所联想到的测试，由修复软件的错误可能会带来的问题。

### 14.3.5 功能图法

前面的章节中提到，一个程序的功能通常由静态说明和动态说明组成，动态说明描述了输入数据的次序或者转移的次序；静态说明描述了输入条件和输出条件之间的对应关系。对于比较复杂的程序，由于大量的组合情况的存在，仅仅使用静态说明来组织测试往往是不够的，必须还要动态说明来补充。功能图法就是因此而产生的一种测试用例设计方法。

功能图法就是使用功能图形式化地表示程序的功能说明，并机械地生成功能图的测试用例。功能图模型由状态迁移图和逻辑功能模型组成。其中，状态迁移图用于表示输入数据序列以及相应的输出数据，由输入和当前的状态决定输出数据和后续状态；逻辑功能模型用于表示在状态输入条件和输出条件之间的对应关系。逻辑功能模型只适合于描述静态说明，输出数据仅仅由输入数据决定。测试用例测试由测试中经过的一系列的状态以及在每个状态中必须依靠输入、输出数据满足的一对条件组成。

功能图法是一种黑盒和白盒混合用例设计方法，在功能图法中，要用到逻辑覆盖和路径测试的概念和方法，这属于白盒测试用例设计中的内容。

我们举个例子来说明，假设我们做 Windows 的屏幕保护程序测试（有密码保护功能），其程序流程图如图 14-3 所示，状态迁移图如图 14-4 所示，对应的逻辑功能表如表 14-14 所示。

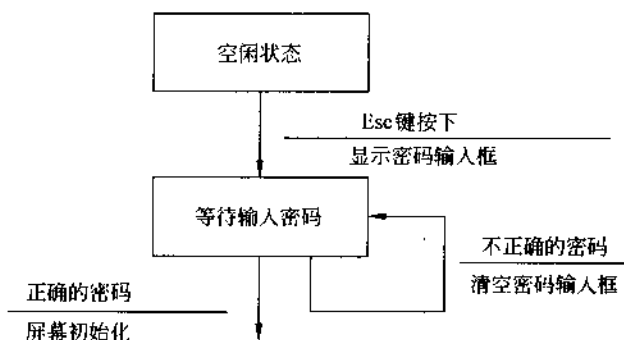


图 14-3 程序流程图

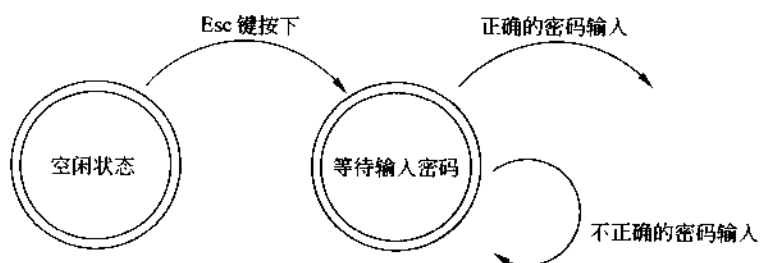


图 14-4 状态迁移图

表 14-14 逻辑功能表

输入	Esc 键按下	I1
	其他键按下	I2
	正确的密码输入	I3
	错误的密码输入	I4
输出	显示密码输入框	O1
	密码错误提示信息	O2
状态	空闲状态	S1
	等待输入密码	S2
	返回空闲状态	S3
	初始化屏幕	S4

接下来，需要利用功能图来生成测试用例，从逻辑功能表中，可以根据所有的输入、输出以及状态来生成所需要的节点和路径，形成实现功能图的基本路径组合。这样，就可

以使用前面白盒测试用例中提到的路径覆盖法来设计测试用例了。

## 14.4 测试用例的组织和跟踪

前面主要讲的是常用的测试用例设计方法。测试用例最终是为实现有效的测试服务的，那么怎样将这些测试用例完整地结合到测试过程中加以使用呢？这就涉及到测试用例的组织、跟踪和维护问题。

### 14.4.1 组织测试用例

在一个测试过程中，可能涉及到很多的、不同类型的测试用例，在测试中需要有效地对这些测试用例进行组织，这要求测试工程师必须了解测试用例在不同过程中的属性。

- 测试用例的属性

在整个测试计划和测试过程中，测试用例都起到了很重要的作用，不同的阶段，测试用例的属性也不同。可以利用这些属性来进行有效的测试用例组织如图 14-5 所示。

- ◆ 测试用例的编写过程的属性：标识符、测试环境、输入标准、输出标准、关联测试用例标识是构成测试用例的基本因素。
- ◆ 测试用例的组织过程的属性：所属的测试模块/测试组件/测试计划、优先级、类型。
- ◆ 测试用例的执行过程的属性：所属的测试过程/测试任务/测试执行、测试环境和平台、测试结果、关联的软件错误或注释。

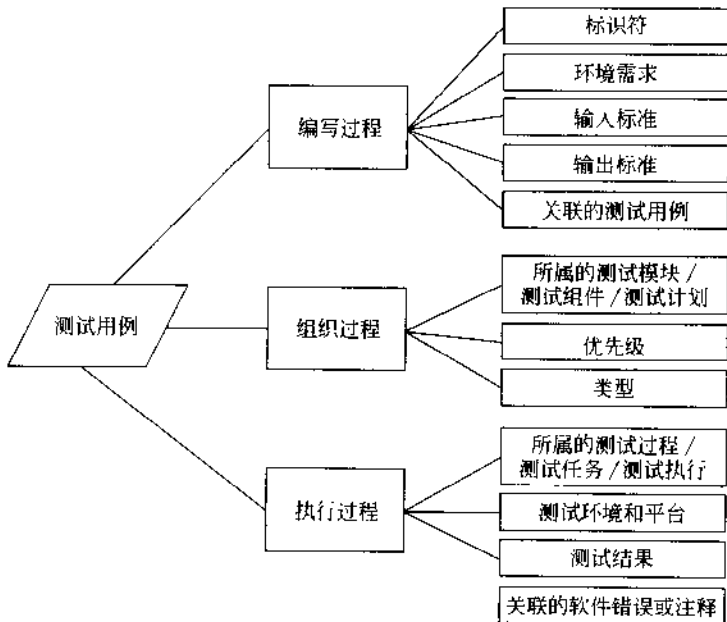


图 14-5 有效的测试用例组织

在后面的小节中，将针对测试用例的属性在组织过程和执行过程中的关系进行具体化的描述。

- 测试用例的组织流程

一般地，进行测试用例的组织需要使用自顶向下的设计方法，首先由测试计划实现测试设计说明书，再通过具体的测试设计说明书实现测试用例的规格说明书，由规格说明书来编写具体的测试用例，如图14-6所示。

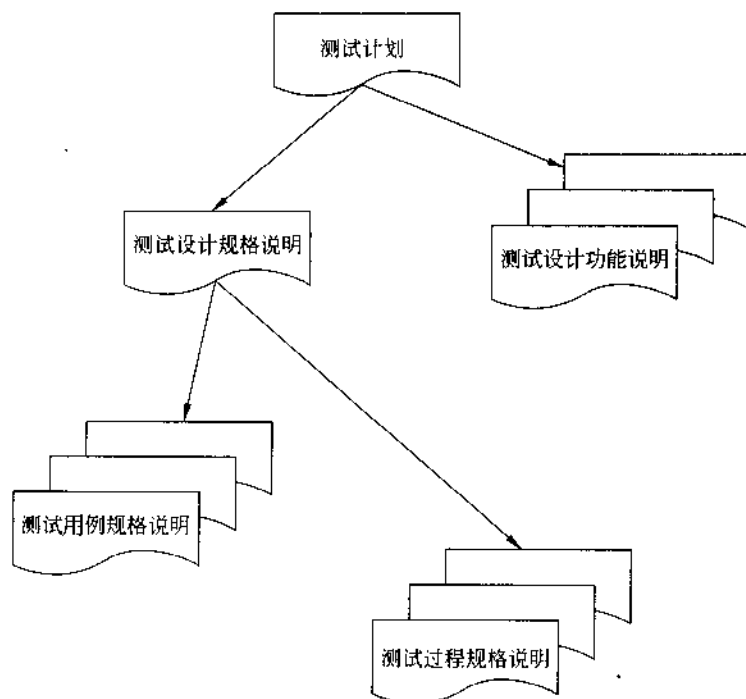


图 14-6 测试用例组织

按照这个组织流程来组织测试用例，这涉及到测试用例组织的方法。

- 测试用例的组织方法

测试用例必须有效地组织起来才能发挥效率。通常情况下，使用以下的几种方法来组织测试用例。

- ◆ 按照程序的功能块组织：应用程序的规格说明书一般是按照不同的功能块进行组织的，因此，按照程序的功能块进行测试用例的组织是一种很好的方法。将属于不同模块的测试用例组织在一起，能够很好地覆盖所测试的内容，准确地执行测试计划。
- ◆ 按照测试用例的类型组织：将不同类型的测试用例按照类型进行分类组织测试，也是一种常见的方法，例如，一个测试过程中可以将功能/逻辑测试、压力/负载测试、异常测试、兼容性测试等具有相同类型的用例组织成单独的测试单元或模块来测试。

- ◆ 按照测试用例的优先级组织：和软件错误相类似，测试用例拥有不同优先级，可以按照实际测试过程的需要，自己定义测试用例的优先级，从而使得测试过程有层次、有主次的进行。

以上各种方式中，根据程序的功能块进行组织是最常用的方法，同时可以将三种方式混合起来灵活运用，例如可以先按照不同的程序功能块将测试用例分成若干个模块，再在不同的模块中划分出不同类型的测试用例，按照优先级顺序进行排列，这样就能形成一个完整而清晰的，基于测试用例进行组织的测试计划。

#### ● 测试用例组织与测试过程

组织好测试用例后，需要使用测试用例。按照前面的测试用例的属性分析和组织方法，可以通过以下的过程来实现测试用例的组织 and 测试过程的组织。在图 14-7 中：

- (1) 测试模块是由单个的测试用例组织起来的。
- (2) 多个测试模块组成测试套件（测试单元）。
- (3) 测试套件加上所需要的测试环境和测试平台需求组成测试计划。
- (4) 测试计划确定后，形成测试执行。
- (5) 测试执行划分成多个测试任务。
- (6) 将测试任务分配给测试人员实现测试过程，测试过程的分配按照测试模块来划分，测试过程中参考的是单个测试用例。
- (7) 由测试人员的测试过程形成测试结果。

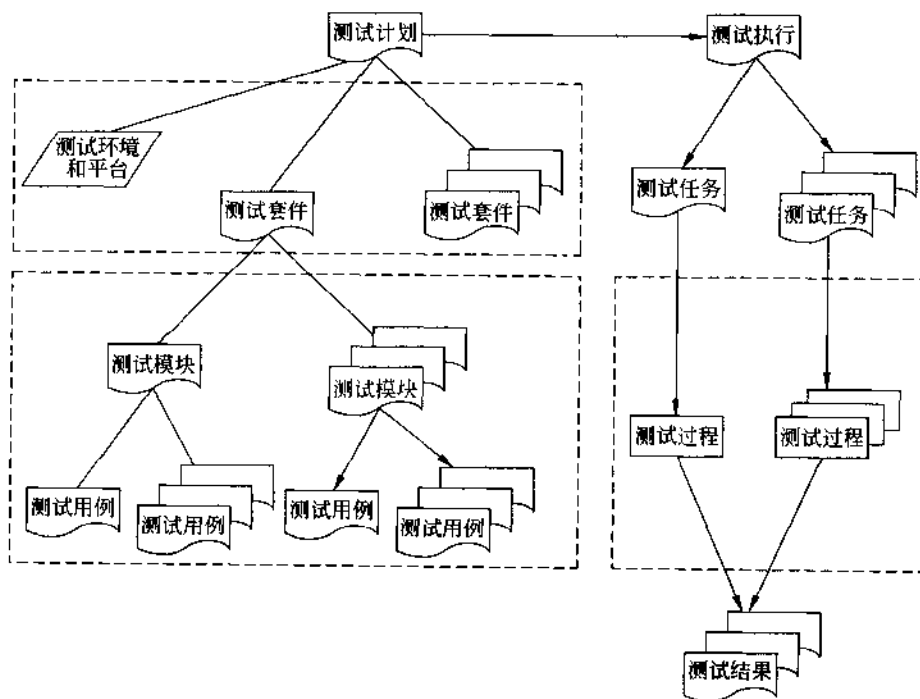


图 14-7 测试用例的组织 and 测试过程的组织

最终形成的测试结果就是所需要跟踪和分析的测试输出。

### 14.4.2 跟踪测试用例

在测试过程计划确定后测试执行开始之前，测试组长应该能够回答下面的几个问题：

- 测试计划中需要执行哪些测试组件？
- 测试计划中有多少测试用例？
- 在执行测试过程中，使用什么方法来记录测试用例的状态？
- 如何挑选出有效的测试组件和测试用例来着重测试某些模块？
- 上次使用的测试用例的通过率是多少？
- 在未通过的测试用例中，有多少是上次执行的时候也未通过的？

准确地回答这些问题，需要对测试过程中测试用例进行跟踪。

前面提到，测试过程中，测试用例有三种状态：通过、未通过和未测试。根据在测试执行过程中测试用例的状态，实现测试用例的跟踪，从而进行测试有效性的检验。因此，测试用例的跟踪主要是针对测试过程中测试用例的执行和输出而进行的跟踪，从而达到测试过程的可管理性和进行测试有效性评估。

跟踪测试用例包括两个方面的内容。

- 测试用例执行的跟踪：本章的开头，就提到测试用例具有易组织性、可评估性和管理性，在测试用例执行过程中，实现测试用例执行过程的跟踪可以有效地将测试过程量化。例如，执行一轮测试中，需要跟踪总共执行了多少测试用例，每个测试人员平均每天使用多少测试用例，测试用例中通过、未通过以及未使用的占多少，未使用的原因是什么，当然，这是个相对的过程，测试人员工作量的跟踪不应该仅仅凭借测试用例的执行情况和发现的程序缺陷多少来判定，但至少，通过测试执行情况的跟踪可以大致判定当前的项目/软件 and 测试的质量与进度，并对测试的时间做出大致的推断，如图 14-8 所示。

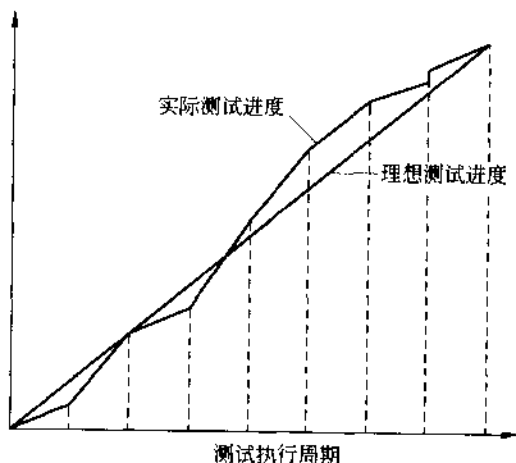


图 14-8 测试进度跟踪图

- 测试用例覆盖率的跟踪：测试用例的覆盖率指的是根据测试用例进行测试的执行结果与实际的软件存在的问题的比较，从而实现对测试有效性的评估如图 14-9。

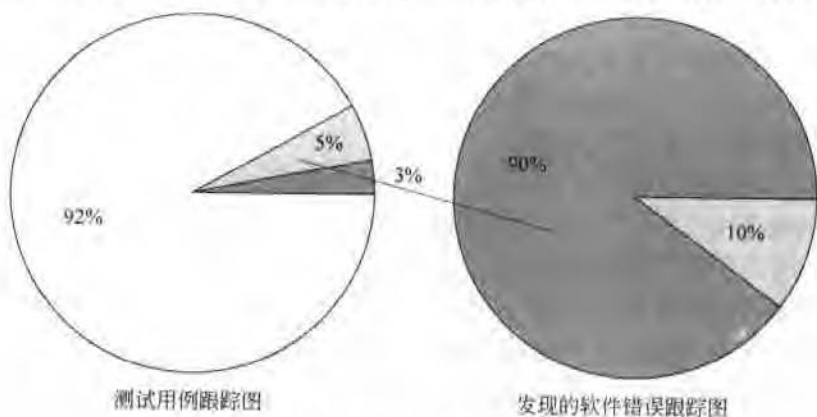


图 14-9 测试用例覆盖率的跟踪

如图 14-9 所示，在一个测试执行中，92%的测试用例通过，5%的测试用例未通过，3%的测试用例未使用，在发现的软件缺陷和错误中，有 90%通过测试用例检测出来的，而有 10%是未通过测试用例检验出来，此时，需要对这些软件错误进行分类和数据分析，完善测试用例，从而使得测试结果更准确，让遗漏问题的可能性最小化。

图 14-10 是针对每个测试模块的测试用例的跟踪。

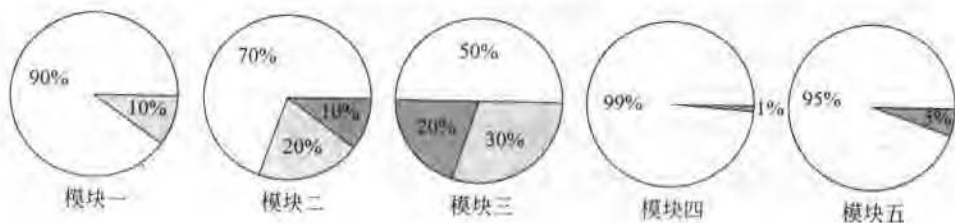


图 14-10 模块测试用例跟踪图

通过以上的分析，不难发现，模块三和模块四的通过率和未使用率都很低，此时测试组长需要对这两个模块的测试用例以及测试过程进行分析，是这个模块的测试用例设计不合理？还是测试过程中模块中的软件错误太多？根据实际的数据分析，可以对这两个模块进行单独再测试，通过纵向的数据比较，来实现软件质量的管理和追踪。

跟踪测试用例的形式一般有几种。

- 记忆：顾名思义，凭借个人的记忆来跟踪测试用例，这是一种非常不可取的方法，除非是测试只是基于个人开发的小型软件上。
- 书面文档：在比较小规模测试项目中，使用书面文档记录和跟踪测试用例也是可行的一种方法。测试用例清单的列表和图例也可以被有效地使用，但作为组织和搜索数据进行分析时，这种方法是很有局限的。
- 电子表格：一种流行而高效的方法是使用电子表格来跟踪和记录测试的过程，如图 14-11 所示，通过表格中列出的测试用例的跟踪细节，可以直观地看到测试的状



态以及分析和统计测试用例的通过,与软件缺陷的关联等,这为测试中有效管理和分析测试过程以及软件的质量提供了有效的量化依据。

	A	B	C	D	E	F
1	测试跟踪表					
2	测试组件/测试用例	测试结果(2004/01/10)	测试结果(2004/01/20)	测试结果(2004/01/30)	软件缺陷ID列表	
3	模块一					
4	测试用例1001	PASS	PASS	PASS		
5	测试用例1002	PASS	PASS	PASS		
6	测试用例1003	FAIL	PASS	PASS	10	
7	测试用例1004	PASS	PASS	PASS		
8	测试用例1005	PASS	PASS	PASS		
9	测试用例1006	FAIL	FAIL	PASS	13,15	
10	测试用例1007	PASS	PASS	PASS		
11	测试用例1008	PASS	PASS	PASS		
12	测试用例1009	PASS	PASS	PASS		
13	测试用例1010	PASS	PASS	PASS		
14						
15	模块二					
16	测试用例2001	FAIL	PASS	PASS	19	
17	测试用例2002	PASS	PASS	PASS		
18	测试用例2003	PASS	PASS	PASS		
19	测试用例2004	PASS	PASS	PASS		
20	测试用例2005	FAIL	FAIL	FAIL	11,20,24	
21	测试用例2006	PASS	PASS	PASS		
22	测试用例2007	PASS	PASS	PASS		
23	测试用例2008	PASS	PASS	PASS		
24	测试用例2009	PASS	PASS	PASS		
25	测试用例2010	PASS	PASS	PASS		
26						

图 14-11 跟踪和记录测试的过程

- 自定义数据库:最理想的方式是通过自定义的数据库来跟踪测试用例的执行和覆盖率,例如,测试人员通过特定的自定义程序如 Web 页面将测试的结果提交,通过自定义的数据库(Access、SQL Server、MySQL、Oracle 等用户习惯的数据库系统)来存储这些测试结果,并通过自己编写的工具生成报表、分析图等,这样将更加有效地管理和跟踪整个的测试过程,当然,所花费的成本将也是最高的。

### 14.4.3 维护测试用例

组织和编写良好的测试用例具有很强的可复用性,因此,在重复使用的过程中,需要对测试用例进行维护或者更新,测试用例不是一成不变的,当一个阶段测试过程结束后,或多或少地会发现一些测试用例编写的不够合理,或者是在下一个版本中使用前一个版本的测试用例,这时,部分功能描述已经发生了更改,也需要修改测试用例,使之具有良好的延续性,如表 14-15 所示。

通常情况下,测试用例需要更新可能有以下几种原因。

- 先前的测试用例设计不全面或者不够准确。随着测试的深入和对产品规格说明书的深入研究,某些地方的逻辑描述不确切,不清楚。
- 部分很严重的软件错误未在测试用例中涵盖。
- 新的版本有新功能的需求或者改动。
- 编写的测试用例不规范或者语句错误。
- 旧的测试用例已经不再适用,需要删除。

表 14-15 测试用例

原 因	更新时间	优 先 级
测试过程的深入发现的逻辑错误	测试过程中	高，需要及时更新
部分有严重的软件错误未在测试用例中涵盖	测试过程中	高，需要及时更新
新的软件版本需求或改动	测试过程前	高，需要在测试进行前进行更新
编写的测试用例不规范或者语句错误	测试过程中	中，尽快修复，以免混淆测试人员
旧的测试用例已经不再使用，需要删除	测试过程后	中，删除或注释掉不再使用的测试用例

● 测试用例维护和修改流程

维护测试用例的过程是实时的、长期的，和编写测试用例不同，维护测试用例一般不涉及到大的组织结构的改动，例如在某个模块里面，如果先前的测试用例已经不能覆盖目前的测试内容，可能需要重新定义一个独立的测试模块单元来重新组织新的测试用例。和测试用例编写过程相同的是，测试用例的维护需要几个特定的流程。

(1) 测试工程师、测试用例编写者或者其他使用、查看测试用例的人发现测试用例有错误或者不合理，向编写者提出测试用例修改建议，并提供足够的理由（功能规格说明书、用户使用场景等）。

(2) 测试用例编写者（或修改者）根据测试用例的关联性和修改意见，进行测试用例的修改。

(3) 向开发项目组长递交修改后的测试用例。

(4) 项目组长、开发人员以及测试用例编写者（或修改者）进行复核后提出意见，通过后，由测试用例编写者（或修改者）提出最后的修改，并提供修改后的文档和修改日志。

基本的流程图如图 14-12 所示。

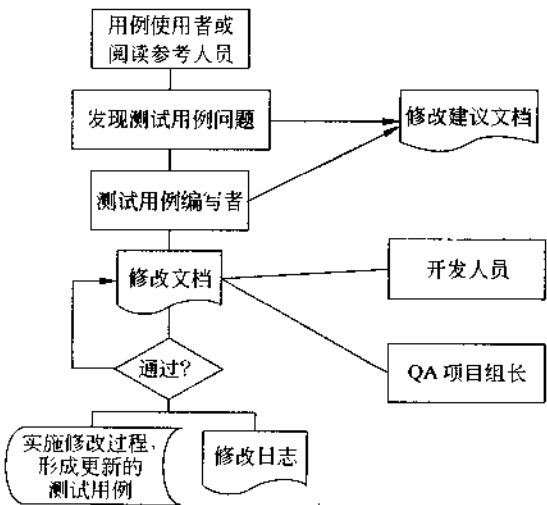


图 14-12 测试用例的维护基本的流程图

### 14.4.4 测试用例的覆盖率

在前面的测试用例跟踪一节中提到, 测试用例的覆盖率指的是根据测试用例进行测试的执行结果与实际的软件存在的问题的比较, 从而实现对测试有效性的评估。

测试用例的覆盖率是评估测试过程和测试计划的一个参考依据, 需要对低覆盖率的测试用例进行数据分析, 根据跟踪和分析的结果, 更有针对性、更有效地组织测试过程。

当然, 测试用例的覆盖率并非一个绝对的判定因素, 它对整个测试过程起到分析、参考作用, 因此, 将测试用例的覆盖率作为检验测试过程和代码质量的依据是不够准确的。

### 小结

测试用例的设计是测试过程中一个很重要的组成部分, 围绕测试用例而形成的测试过程和组织方法是一个比较复杂的软件过程, 测试用例的设计也是循序渐进的过程, 随着测试过程的进行和完善而逐渐成熟起来。

通过本章的学习, 应该了解和掌握以下内容:

1. 为什么要使用测试用例。
2. 测试用例由哪些基本元素组成。
3. 测试用例编写和设计时需要遵循的基本原则。
4. 白盒测试用例和黑盒测试用例设计的基本方法。
5. 测试用例设计、组织和测试过程组织之间的关系和实践过程。
6. 跟踪和维护测试用例。

### 思考题

1. 利用白盒测试用例设计的方法设计如下的测试用例, 测试用例要求。
2. 利用黑盒测试用例设计方法。
3. 试编写测试方案, 来实现。

# 第 15 章 报告所发现的软件缺陷

在这一章主要介绍有关软件缺陷描述、处理和跟踪整个生命周期中所积累的实践经验，包括软件缺陷的属性，处理、跟踪、建立基于数据库的软件缺陷跟踪系统等信息，使用软件缺陷的捕捉工具 Soft\_ICE 创建记录软件缺陷的日志文件，分离和再现软件缺陷。

## 15.1 软件缺陷的描述

软件缺陷是什么？在本书 2.2.1 节已给出定义：软件缺陷指的是系统或系统部件中那些导致系统或部件不能实现其功能的缺陷。如果在执行中遇到一个缺陷，可能引起系统的失效。那么准确、有效地定义和描述软件缺陷，可以使软件缺陷得以快速修复，节约软件测试项目的成本和资源，提高产品质量。

### 15.1.1 软件缺陷的基本描述

软件缺陷的描述是软件缺陷报告中测试人员对问题陈述的一部分，并且是软件缺陷报告的基础部分。同时，软件缺陷的描述也是测试人员就一个软件问题与开发小组交流的最初且最好的机会。一个好的描述需要使用简单、准确、专业的语言来抓住缺陷的本质。否则，它就会使信息含糊不清，可能会误导开发人员。以下是软件缺陷的有效描述规则：

- 单一准确。每个报告只针对一个软件缺陷，在一个报告中报告多个软件缺陷的弊端是常常会导致只有其中一个软件缺陷得到注意和修复。
- 可以再现。提供这个缺陷的精确步骤，使开发人员容易看懂，可以再现并修复缺陷。
- 完整统一。提供完整、前后统一的软件缺陷的修复步骤和信息，例如图片信息、Log 文件等。
- 短小简练。通过使用关键词，可以使软件缺陷的标题描述短小简练，又能准确解释产生缺陷的现象。如“主页的导航栏在低分辨率下显示不整齐”中“主页”、“导航栏”、“分辨率”等是关键词。
- 特定条件。许多软件功能在通常情况下没有问题，而是在某种特定条件下会存在缺陷，所以软件缺陷描述不要忽视这些看似细节但又必要的特定条件（如特定的操作系统、浏览器或某种设置等），能够提供帮助开发人员找到原因的线索。如“搜索功能在没有找到结果返回时跳转页面不对”。
- 补充完善。从发现 bug 那一刻起，测试人员的责任就是保证它被正确的报告，并且得到应有的重视，继续监视其修复的全过程。

- 不做评价。软件缺陷描述不要带有个人观点，不要对开发人员进行评价。软件缺陷报告是针对产品的。

遵循软件缺陷有效描述的规则会有下列益处：

- 清晰、准确的软件缺陷描述可以减少软件缺陷从开发人员返回的数量。
- 提高软件缺陷修复的速度，使每一个小组都能够有效地工作。
- 提高测试人员的信任度，可以得到开发人员对清晰的软件缺陷描述有效的响应。
- 加强开发人员、测试人员和管理人员之间的协同工作能力，让他们可以更好地工作。

### 15.1.2 软件缺陷属性

开发人员需要去修复每一个软件缺陷，但是不是每个软件缺陷都需要开发人员紧急修复呢？这需要定义软件缺陷属性，以提供开发人员作为参考，按照优先等级、严重程度去修复软件缺陷，不至于遗漏严重的软件缺陷。对于测试人员，利用软件缺陷属性可以跟踪软件缺陷，保证产品的质量。下面列出软件缺陷属性：

软件缺陷属性包括缺陷标识、缺陷类型、缺陷严重程度、缺陷产生可能性、缺陷优先级、缺陷状态、缺陷起源、缺陷来源、缺陷原因。

- 缺陷标识：是标记某个缺陷惟一的标识，可以使用数字序号表示。
- 缺陷类型：是根据缺陷的自然属性划分缺陷种类，如表 15-1 所示。

表 15-1 软件缺陷类型列表

缺陷类型	描 述
功能	影响了各种系统功能、逻辑的缺陷
用户界面	影响了用户界面、人机交互特性，包括屏幕格式、用户输入灵活性、结果输出格式等方面的缺陷
文档	影响发布和维护，包括注释、用户手册、设计文档
软件包	由于软件配置库、变更管理或版本控制引起的错误
性能	不满足系统可测量的属性值，如执行时间、事务处理速率等
系统/模块接口	与其他组件、模块或设备驱动程序、调用参数、控制块或参数列表等不匹配、冲突

- 缺陷严重程度：是指因缺陷引起的故障对软件产品的影响程度，所谓“严重性”指的是在测试条件下，一个错误在系统中的绝对影响，如表 15-2 所示。
- 缺陷产生的可能性：指缺陷在产品中发生的可能性，通常可以用频率来表示，如表 15-3 所示。
- 缺陷优先级：指缺陷必须被修复的紧急程度。“优先级”的衡量抓住了在严重性中没有考虑的重要程度因素，如表 15-4 所示。

一般来讲，缺陷严重等级和缺陷优先级相关性很强，但是，具有低优先级和高严重性的错误是可能的，反之亦然。例如，产品徽标是重要的，一旦它丢失了，这种缺陷是用户界面的产品缺陷，但是它阻碍产品的形象。那么它是优先级很高的软件缺陷。

表 15-2 软件缺陷严重等级列表

缺陷严重等级	描 述
致命 (Fatal)	系统任何一个主要功能完全丧失, 用户数据受到破坏, 系统崩溃、悬挂、死机, 或者危及人身安全
严重 (Critical)	系统的主要功能部分丧失, 数据不能保存, 系统的次要功能完全丧失, 系统所提供的功能或服务受到明显的影响
一般 (Major)	系统的次要功能没有完全实现, 但不影响用户的正常使用。例如: 提示信息不太准确或用户界面差、操作时间长等一些问题
较小 (Minor)	使操作者不方便或遇到麻烦, 但它不影响功能的操作和执行, 如个别不影响产品理解的错别字、文字排列不整齐等一些小问题

表 15-3 软件缺陷产生可能性列表

缺陷产生可能性	描 述
总是 (Always)	总是产生这个软件缺陷, 其产生的频率是 100%
通常 (Often)	按照测试用例, 通常情况下会产生这个软件缺陷, 其产生的频率大概是 80%~90%
有时 (Occasionally)	按照测试用例, 有时候产生这个软件缺陷, 其产生的频率大概是 30%~50%
很少 (rarely)	按照测试用例, 很少产生这个软件缺陷, 其产生的频率大概是 1%~5%

表 15-4 软件缺陷优先级列表

缺陷优先级	描 述
立即解决 (P1 级)	缺陷导致系统几乎不能使用或测试不能继续, 需立即修复
高优先级 (P2 级)	缺陷严重, 影响测试, 需要优先考虑
正常排队 (P3 级)	缺陷需要正常排队等待修复
低优先级 (P4 级)	缺陷可以在开发人员有时间的时候被纠正

- 缺陷状态: 指缺陷通过一个跟踪修复过程的进展情况, 也就是在软件生命周期中的状态基本定义, 如表 15-5 所示。

表 15-5 软件缺陷状态列表

缺陷状态	描 述
激活或打开 (active or open)	问题还没有解决, 存在源代码中, 确认“提交的缺陷”, 等待处理, 如新报的缺陷
已修正或修复 (fixed or resolved)	已被开发人员检查、修复过的缺陷, 通过单元测试, 认为已解决但还没有被测试人员验证
关闭或非激活 (close or inactive)	测试人员验证后, 确认缺陷不存在之后的状态
重新打开	测试人员验证后还依然存在的缺陷, 等待开发人员进一步修复
推迟	这个软件缺陷可以在下一个版本中解决
保留	由于技术原因或第三者软件的缺陷, 开发人员不能修复的缺陷

续表

缺陷状态	描 述
不能重现	开发不能再现这个软件缺陷, 需要测试人员检查缺陷再现的步骤
需要更多信息	开发能再现这个软件缺陷, 但开发人员需要一些信息, 例如缺陷的日志文件、图片等

- 缺陷起源: 缺陷引起的故障或事件第一次被检测到的阶段, 如表 15-6 所示。

表 15-6 软件缺陷起源列表

缺陷起源	描 述
需求	在需求阶段发现的缺陷
构架	在系统构架设计阶段发现的缺陷
设计	在程序设计阶段发现的缺陷
编码	在编码阶段发现的缺陷
测试	在测试阶段发现的缺陷
用户	在用户使用阶段发现的缺陷

第 2 章介绍了在软件生命周期中软件缺陷占的比例: 需求和构架设计阶段占 54%, 设计阶段占 25%, 编码阶段占 15%, 其他占 6%。

- 缺陷来源: 指缺陷所在的地方, 如文档、代码等, 如表 15-7 所示。

表 15-7 软件缺陷来源列表

缺陷来源	描 述
需求说明书	需求说明书的错误或不清楚引起的问题
设计文档	设计文档描述不准确, 和需求说明书不一致的问题
系统集成接口	系统各模块参数不匹配、开发组之间缺乏协调引起的缺陷
数据流 (库)	由于数据字典、数据库中的错误引起的缺陷
程序代码	纯粹在编码中的问题所引起的缺陷

- 缺陷根源: 指造成上述错误的根本因素, 以寻求软件开发流程的改进、管理水平的提高, 如表 15-8 所示。

表 15-8 软件缺陷根源列表

缺陷根源	描 述
测试策略	错误的测试范围, 误解了测试目标, 超越测试能力等
过程, 工具和方法	无效的需求收集过程, 过时的风险管理过程, 不适用的项目管理方法, 没有估算规程, 无效的变更控制过程等
团队/人	项目团队职责交叉, 缺乏培训。没有经验的项目团队, 缺乏士气和动机不纯等
缺乏组织和通讯	缺乏用户参与, 职责不明确, 管理失败等
硬件	硬件配置不对、缺乏, 或处理器缺陷导致算术精度丢失, 内存溢出等

续表

缺陷根源	描 述
软件	软件设置不对、缺乏，或操作系统错误导致无法释放资源，工具软件的错误，编译器的错误，千年虫问题等
工作环境	组织机构调整，预算改变，工作环境恶劣，如噪音过大

## 15.2 软件缺陷相关的信息

上一节所叙述的软件缺陷属性是其基本信息，为了更好地处理软件缺陷，我们需要了解其他相关的信息。

软件缺陷相关的信息包括软件缺陷图片、记录信息和如何再现和分离软件缺陷；对于某一个软件缺陷报告，测试人员应该给予相关的信息，例如捕捉到软件缺陷日志文件和图片，保证开发人员和其他的测试人员可以分离和再现它。在这一章中重点介绍可以捕捉 bug 的工具，在什么情况下需要添加图片文件和如何分离及再现软件缺陷的建议。

### 15.2.1 软件缺陷的图片、记录信息

软件缺陷的图片、记录信息是软件缺陷报告中重要的组成部分，下面将介绍常用的捕捉软件缺陷工具，如何使用这些工具，添加软件缺陷图片的作用和为什么要添加图片信息等内容。

#### 1. 记录软件缺陷的相关图片

一些涉及用户界面（UI）的软件缺陷很难用文字清楚地描述，因此软件测试人员通过附上图片比较直观地表示缺陷发生在产品界面什么位置、有什么问题等。

- 采用图片的格式

测试人员一般采用 JPG、GIF 的图片格式，因为这类文件占用的空间小，打开的速度快。

- 什么情况下需要附上图片

通常情况下，出现在用户界面（user interface），并且影响用户使用或者影响产品美观的软件缺陷，附上图片比较直观，例如：

- 当产品中有一段文字没有显示完全，为了明确标识这段文字的位置，测试人员必须贴上图片。
- 在测试外国语言版本的时候，当发现产品中有一段文字没有翻译，测试人员需要贴上图片标识没有翻译的文字。
- 在测试外国语言版本的时候，当发现产品中有一段外国文字显示乱字符，测试人员必须贴上图片标识那些乱字符的外国文字。
- 产品中的语法错误、标点符号使用不当等软件缺陷，测试人员贴上图片告诉开发



人员缺陷在什么地方。

- 在产品中运用错误公司标志、图片没有显示完全等软件缺陷，也需要贴上图片。

测试人员需要注意，有必要在图片上用颜色标注缺陷的位置，给开发人员一目了然的效果，使得软件缺陷尽快修复。

## 2. 使用 Soft-ICE 记录软件缺陷信息

在第 11 章已经知道，Soft-ICE 是 Compuware 公司产品 NuMega DriverStudio 中一个具有代表性的工具，用于跟踪软件运行时的变量、内存等状态，而且可以捕捉系统崩溃时的状态。使用它可以记录产品发生缺陷的地方，同时生成日志文件。一般情况下，测试人员需要在软件缺陷报告上附上日志文件，便于开发人员修复软件缺陷。

当遭遇软件崩溃时，如何使用 Soft-ICE？在开始测试之前，已经安装了 Soft-ICE 并启动了“faults on”命令。当软件发生崩溃现象时，可以使用下面命令去捕捉必要的信息：

- stack
- u eip-80

如果数据窗口是开启的状态，可以输入“wd”来关闭该窗口，然后再输入“dd esp-20”命令。stack、dd esp-20 是为了标注跟踪信息。

- 通过输入“x”，退出 Soft-ICE 的窗口；如果还是无法退出 Soft-ICE，需要输入“faults off”，然后输入“x”。
- 打开 Soft-ICE 应用程序，立即保存日志文件。一旦再次打开 Soft-ICE，请输入“faults on”。

以下是一些常用命令：

- 在任何时候，按 Esc 键删除命令行；
- 输入“Help”列出所有命令行及其帮助说明；
- 按 Ctrl+D 组合键，在 Soft-ICE 窗口和 Windows 系统之间转换。

当测试人员报告崩溃的软件缺陷的时候，如下几点是重要的：

- 任何造成软件崩溃的缺陷都需要附上 Soft-ICE 的跟踪 Log 文件，使开发人员能够分析这个缺陷的错误信息。
- 附上 Soft-ICE 的跟踪 Log 文件之后，重新打开检查 Log 文件信息是无误的。
- 如果不能附上 Soft-ICE 的跟踪 Log 文件，应该说明为什么不能附上 Soft-ICE 的跟踪 Log 文件。

## 15.2.2 分离和再现软件缺陷

要想有效地分离软件缺陷，需要清楚地、准确地描述产生软件缺陷的具体步骤和条件。在某些情况下只要具备特定的测试用例，软件缺陷就会再次出现，但某些情况下，再现、验证软件缺陷的条件、环境、技术等要求都非常高，而且非常浪费资源。

## 1. 分离和再现软件缺陷的步骤

为了有效地再现软件缺陷，除了按照软件缺陷的有效描述规则来描述软件缺陷，还要遵循软件缺陷分离和再现的方法，虽然有时少数几个缺陷很难再现、或者根本无法再现。以下介绍如何分离和再现缺陷的一些常用方法和技巧。

- 确保所有的步骤都被记录。记录下所做的每一件事、每一个步骤、每一个停顿。无意间丢失一个步骤或者增加一个多余步骤，可能导致无法再现软件缺陷。在尝试运行测试用例时，可以利用录制工具确切地记录执行步骤。所有的目标是确保导致软件缺陷所需的全部细节是可见的。
- 特定条件和时间。软件缺陷仅在特定时刻出现吗？软件缺陷在特定条件下产生吗？产生软件缺陷是网络忙吗？在较差和较好的硬件设备上运行测试用例会有不同的结果吗？
- 压力和负荷、内存和数据溢出相关的边界条件。执行某个测试可能导致产生缺陷的数据被覆盖，而只有在试图使用该数据时才会再现。在重启计算机后软件缺陷消失，当执行其他测试之后又出现这类软件缺陷，需要注意某些软件缺陷可能是在无意中产生的。
- 考虑资源依赖性包括内存、网络和硬件共享的相互作用等。软件缺陷是否仅在运行其他软件并与其他硬件通信的“繁忙”系统上出现？软件缺陷可能最终证实跟硬件资源、网络资源有相互的作用，审视这些影响有利于分离和再现软件缺陷。
- 不能忽视硬件。与软件不同，硬件不按预定方式工作。板卡松动、内存条损坏或者 CPU 过热都可能导致像是软件缺陷的失败。设法在不同硬件上再现软件缺陷。在执行配置或者兼容性测试时特别重要。判定软件缺陷是在一个系统上还是在多个系统上产生。

开发人员有时根据相对简单的错误信息就能找出问题所在。因为开发人员熟悉代码，因此看到症状、测试用例步骤和分离问题的过程时，可能得到查找软件缺陷的线索。一个软件缺陷分离问题有时需要小组的共同努力。如果尽最大努力分离软件缺陷，也无法表达再现步骤，那么仍然需要记录软件缺陷，因为软件缺陷这个问题还是存在的。

## 2. 分离和调试软件缺陷之间的区别

讨论分离和调试软件缺陷之间的区别，是为了分清测试人员与开发人员的责任，增加界限的清晰度与测试资源的控制能力。面对一个软件缺陷时，开发人员或测试人员为了修复它，会提出一系列处理缺陷的疑问：

- (1) 再现软件缺陷现象所需的最少步骤有哪些？这些步骤成功再现的可能性多大？
- (2) 软件缺陷是否成立？换句话说，测试结果是否可能起源于测试因素或者测试人员自身的错误，还是影响顾客需求的、系统真正的故障？
- (3) 哪些外部因素产生软件缺陷？
- (4) 哪些内部因素，是代码、网络还是环境引起的软件缺陷？
- (5) 怎样才能在不产生新缺陷的条件下使这个软件缺陷得到修复？

(6) 这种修复是否经过调试，单元是否经过测试？

(7) 问题解决了么？它是否通过了确认和回归测试，确定系统的其余部分仍能工作正常？

第(1)步证明了一个软件缺陷不是一个意外，同时精练操作步骤；第(2)、(3)步分离了这个软件缺陷；第(4)~(6)步是调试任务；第(7)步涉及确认和回归测试。在整个过程中，缺陷从测试阶段(第(1)~(3)步)、进入开发阶段(第(4)~(6)步)，然后再回到测试阶段(第(7)步)。虽然这个责任流程似乎简单而明显，但其边界不是很清晰(特别是第(3)、(4)步之间的边界)，会产生一些资源重叠而且浪费大量的精力。

如果软件缺陷描述清楚，包含了第(1)、(2)、(3)步中问题的答案，意味着在隔离与调试之间清楚地划上一条界限，测试人员就能专注于测试过程，而不受开发人员的影响。如果测试人员不能完全表现缺陷的特征，导致再现和错误种类的不确定性，因此无法将它分离，测试人员和开发人员就可能会一起调试过程。实际上，测试人员在其职责范围内有许多其他的工作，不应该被卷入调试工作中。开发人员询问测试人员是调试工作的一部分，这是开发人员的职责所在，而测试人员只要在软件缺陷描述的基础上回答问题就可以了。否则，测试人员可能会花费大量的时间去解答开发人员所提出的问题。

## 15.3 软件缺陷的处理和跟踪

软件缺陷被报出之后，接下来就是要对它进行处理和跟踪，包括软件缺陷生命周期、软件缺陷处理技巧、软件缺陷跟踪的方法和图表、软件缺陷跟踪系统。

软件缺陷跟踪管理是测试工作的一个重要部分，测试的目的是为了尽早发现软件系统中的缺陷，而对软件缺陷进行跟踪管理的目的是确保每个被发现的缺陷都能够及时得到处理。软件测试过程简单说就是围绕缺陷进行的，对缺陷的跟踪管理一般而言需要达到以下目标：

(1) 确保每个被发现的缺陷都能够被解决，“解决”的意思不一定是被修正，也可能是其他处理方式(例如，延迟到下一个版本中修正或者由于技术原因不能被修正)，总之，对每个被发现的 bug 的处理方式必须能够在开发组织中达成一致。

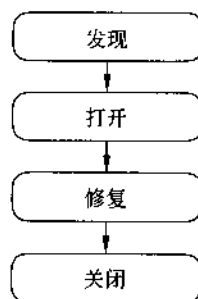
(2) 收集缺陷数据并根据缺陷趋势曲线识别测试过程的阶段。决定测试过程是否结束有很多种方式，通过缺陷趋势曲线来确定测试过程是否结束是常用并且较为有效的一种方式。

(3) 收集缺陷数据并在其上进行分析，作为组织的过程财富。

上述的第(1)条容易受到重视，在谈到缺陷跟踪管理时，一般人都会马上想到这一条，然而对第(2)和第(3)条目标却很容易忽视。其实，在一个运行良好的组织中，缺陷数据的收集和分析是很重要的，从缺陷数据中可以得到很多与软件质量相关的数据。

### 15.3.1 软件缺陷生命周期

生命周期的概念是一个物种从诞生到消亡经历的不同生命阶段，那么软件缺陷生命周期应该指的是一个软件缺陷被发现、报告到这个缺陷被修复、验证直至最后关闭的完整过程。在整个软件缺陷生命周期中，通常是以改变软件缺陷的状态(见表 15-5)来体现不同的生命阶段。因此，对于一个软件测试人员来讲，需要关注软件缺陷在生命周期中的状态变化，来跟踪软件质量和项目进度。一个简单的软件缺陷生命周期如图 15-1 所示。



(1) 发现—打开：测试人员找到软件缺陷并将软件缺陷提交给开发人员。

(2) 打开—修复：开发人员再现、修复缺陷，然后提交给测试人员去验证。

(3) 修复—关闭：测试人员验证修复过的软件，关闭已不存在的缺陷。

在实际工作中，软件缺陷的生命周期不可能像图 15-1 那么简单，需要考虑其他各种情况。图 15-2 给出了一个复杂的软件缺陷生命周期例子。

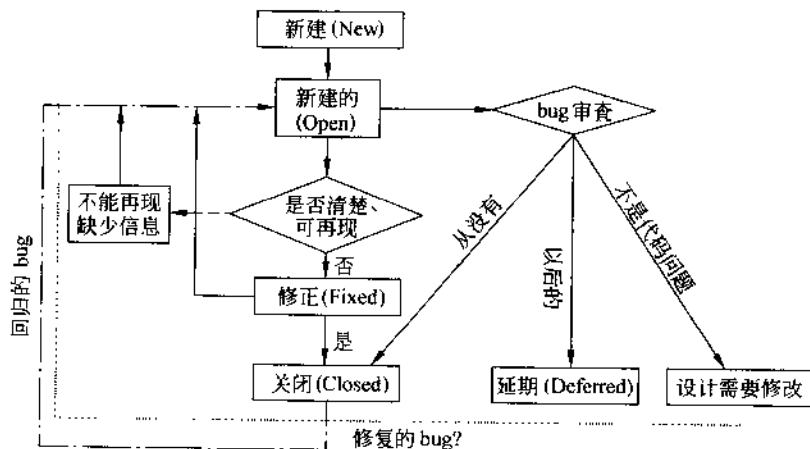


图 15-2 复杂的软件缺陷生命周期

综上所述，软件缺陷在生命周期中经历了数次的审阅和状态变化，最终测试人员关闭软件缺陷来结束软件缺陷的生命周期。软件缺陷生命周期中的不同阶段是测试人员、开发人员和管理人员一起参与、协同测试的过程。软件缺陷一旦发现，便进入测试人员、开发人员、管理人员的严密监控之中，直至软件缺陷生命周期终结，这样即可保证在较短的时间内高效率地关闭所有的缺陷，缩短软件测试的进程，提高软件质量，同时减少开发和维护成本。

### 15.3.2 软件缺陷处理技巧

管理人员、测试人员和开发人员需要掌握在软件缺陷生命周期的不同阶段处理软件缺陷技巧，从而尽快处理软件缺陷，缩短软件缺陷生命周期。以下列出处理软件缺陷的基本技巧：

- 审阅。当测试人员在缺陷跟踪数据库中输入了一个新的缺陷时，测试员应该提交它，以便在它能够起作用之前进行审阅。这种审阅可以由测试管理员、项目经理或其他人来进行，主要审阅缺陷报告的质量水平。
- 拒绝。如果审阅者决定需要对一份缺陷报告进行重大修改，例如需要添加更多的信息或者需要改变缺陷的严重等级，应该和测试人员一起讨论，由测试人员纠正缺陷报告，然后再次提交。
- 完善。如果测试员已经完整地描述了问题的特征并将其分离，那么审查者就会肯定这个报告。
- 分配。当开发组接受完整描述特征并被分离的问题时，测试员会将它分配给适当的开发人员，如果不知道具体开发人员，应分配给项目开发组长，由开发组长再分配给对应的开发人员。
- 测试。一旦开发人员修复一个缺陷，它就将进入测试阶段。缺陷的修复需要得到测试人员的验证，同时还要进行回归测试，检查这个缺陷的修复是否会引入新的问题。
- 重新打开。如果这个修复没有通过确认测试，那么测试人员将重新打开这个缺陷报告。重新打开一个缺陷，需要加注说明，否则会引起“打开—修复”多个来回，造成测试人员和开发人员不必要的矛盾。
- 关闭。如果修复通过验证测试，那么测试人员将关闭这个缺陷。只有测试人员有关闭缺陷的权限，开发人员没有这个权限。
- 暂缓。如果每个人都同意将确实存在的缺陷移到以后处理，应该指定下一个版本号或修改的日期。一旦新的版本开始时，这些暂缓的缺陷应该重新被打开。

测试人员、开发人员和管理人员只有紧密地合作，掌握软件缺陷处理技巧，在项目的不同阶段，及时地审查、处理和跟踪每个软件缺陷，加速软件缺陷状态的变换，不仅提高软件质量，而且促进项目的发展。

### 15.3.3 软件缺陷跟踪系统

在实际运用中还需要软件缺陷跟踪系统，以便描述报告所发现的缺陷，处理软件缺陷属性，跟踪软件缺陷的整个生命周期和生成软件缺陷跟踪图表等。为什么需要建立一套软件缺陷跟踪系统呢？因为它会让我们受益无穷，概括起来有：

- 软件缺陷跟踪系统拥有软件缺陷跟踪数据库，它不仅有利于软件缺陷地清楚描述，还提供统一的、标准化报告，使所有人的理解一致。

- 缺陷跟踪数据库允许自动连续的软件缺陷编号，还提供了大量供分析和统计的选项，这是手工方法无法实现的。
- 基于缺陷跟踪数据库，可快速生成满足各种查询条件的、所必要的缺陷报表、曲线图等，开发小组乃至公司的每一个人都可以随时掌握软件产品质量的整体状况或测试/开发的进度。
- 缺陷跟踪数据库提供了软件缺陷属性并允许开发小组根据对项目的相对和绝对重要性来修复缺陷。
- 可以在软件缺陷的生命期中管理缺陷，从最初的报告到最后的解决，确保了每一个缺陷不会被忽略。同时，它还可以使注意力保持在那些必须尽快修复的重要缺陷上。
- 当缺陷在它的生命周期中变化时，开发人员、测试人员以及管理人员将熟悉新的软件缺陷信息。一个设计良好的软件缺陷跟踪系统可以获取历史记录，并在检查缺陷的状态时参考历史记录。
- 在软件缺陷跟踪数据库中关闭每一份缺陷报告，它都可以被记录下来。当产品送出去时，每一份未关闭的缺陷报告都提供了预先警告的有效技术支持，并且证明测试人员找到特殊领域突然出现的事件中的软件缺陷。

接下来就介绍一下软件缺陷跟踪系统（它遵守 IEEE829-1983 标准）。

### 1. 软件缺陷报告

任何一个缺陷跟踪系统的核心都是“软件缺陷报告”，一份软件缺陷报告详细信息如表 15-9 所示。

表 15-9 软件缺陷项目列表

分类	项目	描述
可跟踪信息	缺陷 ID	惟一的、自动产生的缺陷 ID，用于识别、跟踪、查询
软件缺陷基本信息	缺陷状态	可分为“打开或激活的”、“已修正”、“关闭”等
	缺陷标题	描述缺陷的最主要信息
	缺陷的严重程度	一般分为“致命”、“严重”、“一般”、“较小”四种程度
	缺陷的优先级	描述处理缺陷的紧急程度，1 是优先级最高的等级，2 是正常的，3 是优先级最低的
	缺陷的产生频率	描述缺陷发生的可能性 1%~100%
	缺陷提交人	缺陷提交人的名字（会和邮件地址联系起来），一般就是发现缺陷的测试人员或其他人员
	缺陷提交时间	缺陷提交的时间
	缺陷所属项目/模块	缺陷所属的项目和模块，最好能较精确地定位至模块
	缺陷指定解决人	估计修复这个缺陷的开发人员，在缺陷状态下由开发组长指定相关的开发人员；也会自动和该开发人员的邮件地址联系起来，并自动发出邮件

续表

分类	项目	描 述
软件缺陷基本信息	缺陷指定解决时间	开发管理员指定的开发人员修改此缺陷的时间
	缺陷验证人	验证缺陷是否真正被修复的测试人员；也会和邮件地址联系起来
	缺陷验证结果描述	对验证结果的描述（通过、不通过）
	缺陷验证时间	对缺陷验证的时间
缺陷的详细描述	步骤	对缺陷的操作过程，按照步骤，一步一步地描述
	期望的结果	按照设计规格说明书或用户需求，在上述操作之后，所期望的结果，即正确的结果
	实际发生的结果	程序或系统实际发生的结果，即错误的结果
测试环境说明	测试环境	对测试环境描述，包括操作系统、浏览器、网络带宽、通信协议等
必要的附件	图片、Log 文件	对于某些文字很难表达清楚的缺陷，使用图片等附件是必要的；对于软件崩溃现象，需要使用 Soft_ICE 工具去捕捉日志文件作为附件提供给开发人员

软件缺陷的详细描述，如上所述，由三部分组成：操作/再现步骤、期望结果、实际结果。有必要再做进一步讨论。

- “步骤”提供了如何重复当前缺陷的准确描述，应简明而完备、清楚而准确。这些信息对开发人员是关键的，视为修复缺陷的向导，开发人员有时抱怨糟糕的缺陷报告，问题往往集中在这里。
- “期望结果”与测试用例标准或设计规格说明书及用户需求相一致，达到软件预期的功能。测试人员站在用户的角度要对它进行描述，它提供了验证缺陷的依据。
- “实际结果”测试人员收集的结果和信息，以确认缺陷确实是一个问题，并标识那些影响到缺陷表现的要素。

## 2. 缺陷报告的示例

一份优秀的缺陷报告记录下最少的重复步骤，不仅包括了期望结果、实际结果和必要的附件，还提供必要的的数据、测试环境或条件，以及简单的分析。下面是一个优秀的缺陷报告记录。

重现步骤：

- 打开一个编辑文字的软件并且创建一个新的文档（这个文件可以录入文字）。
- 在这个文件里随意录入一两行文字。
- 选中一两行文字，选择 Font 菜单，然后选择 Arial 字体格式。
- 一两行文字变成了无意义的乱字符。

期望结果：

当用户选择已录入的文字并改变文字格式的时候，文本应该显示正确的文字格式不会

显示成乱字符。

实际结果:

它是字体格式的问题,如果在把文字格式改变成 Arial 之前保存文件,缺陷不会出现。缺陷仅仅发生在 Windows 98,并且改变文字格式成其他的字体格式时,文字是正常显示的。见所附的图片<有一个链接,单击即可看到>

而一份含糊而不完整的缺陷报告缺少重建步骤,并且没有期望结果、实际结果和必要的图片,如下描述。

重现步骤:

- 打开一个编辑文字的软件。
- 录入一些文字。
- 选择 Arial 字体格式。
- 文字变成了乱字符。

期望结果:

实际结果:

一份散漫的缺陷报告(无关的重建步骤,以及对开发人员理解这个错误毫无帮助的结果信息)如下描述:

重现步骤:

- 在 Windows 98 上打开一个编辑文字的软件并且编辑存在的文件。
- 文件字体显示正常。
- 我添加了图片,这些图片显示正常。
- 在此之后,我创建了一个新的文档。
- 在这个文档中我随意录入了大量的文字。
- 在我录入这些文字之后,选择几行文字,并且选择 Font 菜单,然后选择 Arial 字体格式改变文字的字体。
- 有三次我重现了这个缺陷。
- 我在 Solaris 操作系统运行这些步骤,没有任何问题。
- 我在 Mac 操作系统运行这些步骤,没有任何问题。

期望结果:

当用户选择已录入的文字并改变文字格式的时候,文本应该显示正确的文字格式不会显示成乱字符。

实际结果:

我试着选择其他不同的字体格式,但是只有 Arial 字体格式有软件缺陷,不论如何,它可能会出现在我没有测试的其他的字体格式中。



写软件缺陷报告的关键是遵循软件缺陷的有效描述规则及分离和再现软件缺陷的步骤, 仔细做笔记, 这样才能写出简明、清晰的缺陷报告。测试人员还需注意应该在完成测试之后写缺陷报告, 写完报告后, 花一点额外的时间仔细检查一遍。

### 3. 缺陷跟踪数据库信息

项目中使用 Microsoft Excel 电子表格或 Word 文档来记录和跟踪软件缺陷, 但一般只限于最后的分析报告、文档的打印。为了灵活地存储、操作、搜索、分析以及报告大量数据, 我们需要建一个数据库。

可以使用 Microsoft Access 或 SQL Server, 也可以使用 Oracle、DB2 等关系数据库管理系统。一个缺陷跟踪数据库的基本表将要包括多达几十项的数据项, 如 bug 的 ID 号、标题 (title)、状态、严重程度、优先级、再现步骤、期望结果、实际结果、项目名称、模块、报告作者、日期等。

所有缺陷的数据不仅要存储在共享数据库中, 还要有相关的数据连接, 如产品特性数据库、产品配置数据库、测试用例数据库等的集成。因为某个缺陷是和某个产品特性、某个软件版本、某个测试用例等相关联的, 有必要建立起这些关联。同时为了提高缺陷处理的效率, 还有和邮件服务器集成, 通过邮件传递, 测试和开发人员随时可以获得由系统自动发出有关缺陷状态变化的邮件。

## 15.3.4 缺陷跟踪的方法和图表

缺陷数据是生成各种各样测试分析、质量控制图表的基础, 从这些缺陷分析图表中可以清楚地看到缺陷的修复过程, 分析缺陷发生的根本原因, 跟踪管理缺陷的效率。

### 1. 软件项目如何发展: 软件缺陷打开/关闭图表

打开/关闭图表是最基本的缺陷分析图表, 它能提供许多有关软件缺陷状态、项目进度、产品质量、开发人员的工作等信息。

- 项目目前的质量情况取决于累积打开曲线和累积关闭曲线的趋势。
- 项目目前的进度取决于累积关闭曲线和累积打开曲线起点的时间差。
- 开发人员已经修复软件缺陷了吗? 累积关闭曲线是否快速地上升。
- 测试人员是否积极地去验证软件缺陷, 也就是说, 是否累积关闭曲线紧跟在累积打开曲线后面。

管理者可以知道项目在哪一个时间点出现问题, 同时协调开发和测试之间的关系, 积极推动项目的发展, 从而达到项目里程碑的要求, 提高项目发布的质量。以下将通过打开/关闭的累积缺陷图分析项目的进展情况, 如图 15-3 所示。

- 当累积的打开曲线 (图 15-3 的顶部曲线) 在一条渐近线限制下稳定下来, 通常就认为该测试完成了。
- 修正日期在关闭日期之前, 可以看到关闭曲线大约落后了一个星期。这种滞后起源于将修复的软件缺陷引入到产品, 并将该产品发送到测试小组, 以及测试配置

和回归测试所引起的延迟。这种延迟集中到测试的最后一天。

- 当前测试阶段找到软件缺陷的能力在减弱。发现软件缺陷的极限在 8 月 23 号左右。接下来系统测试第二个周期发现少数几个软件缺陷，在最后的周期中没有发现缺陷。
- 开发人员完成了修复软件缺陷了吗？在测试和修复的过程中，发现这两条曲线在不断收敛，当这两条曲线收敛成一个点时，开发人员基本上完成了修复软件缺陷的任务了。并且注意到关闭曲线紧跟在打开曲线的后面，这表明项目小组正在快速地推进问题的解决。
- 当测试人员从一个测试阶段到另一个测试阶段时，发现累积打开曲线有一个凸起。这样的凸起是非常可怕的，说明开发人员修复软件缺陷引入了新的缺陷或者有些软件缺陷被遗漏到下一个阶段发现了。项目管理人员需要召开紧急会议分析当前项目情况，找到解决办法。

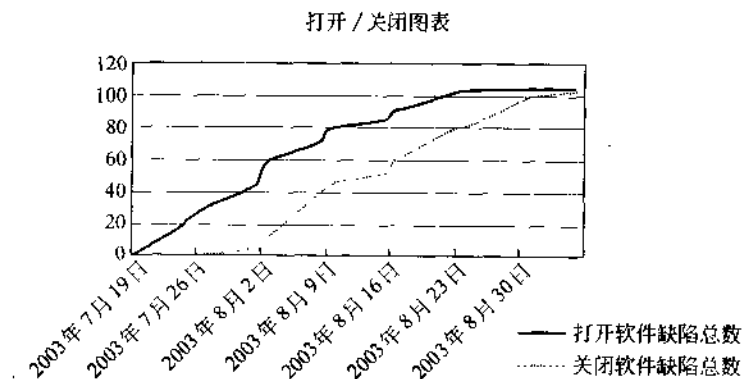


图 15-3 打开/关闭的累积缺陷图

## 2. 软件缺陷为何发生：根本原因图表

分析软件缺陷根本原因不仅有助于测试人员决定哪些功能领域需要增强测试，而且可以使开发人员的注意力集中到那些引起最严重、最频繁问题的领域。图 15-4 显示了软件缺陷产生的 3 个主要来源区域——用户界面显示、逻辑和规格说明书，这些方面的问题占据发现软件缺陷总数的 75%。如果从测试风险角度看，这些区域可能是隐藏缺陷比较多的地方，需要更细、更深的测试。从开发角度来说，这些就是代码质量提高的主要区域，假定某个产品前后发现 10 000 个 bug，代码在这三个区域减少 10 个百分点，则总 bug 数能减少 750 (7.5%)，代码质量改善效果就很显著。

## 3. 开发人员如何响应：关闭软件缺陷周期图表

“关闭周期”有一个简单直观的意义：关闭周期将开发人员对软件缺陷的响应量化到软件缺陷报告中，如图 15-5 所示。

一个稳定的关闭周期图表显示了从一天到另一天相对较少的变化，这是一个理想的例子。如果软件缺陷报告推迟了它们打开的日期，这就将日常关闭曲线拉向 0，此外，一个

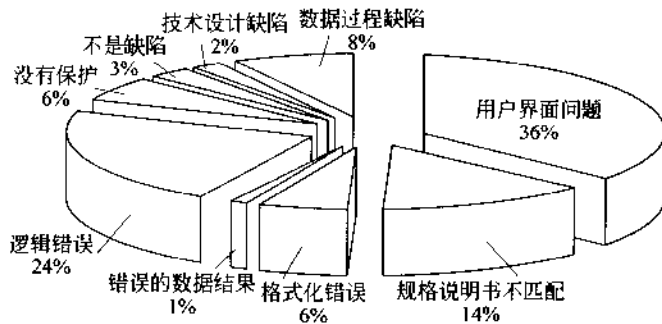


图 15-4 根本原因图表

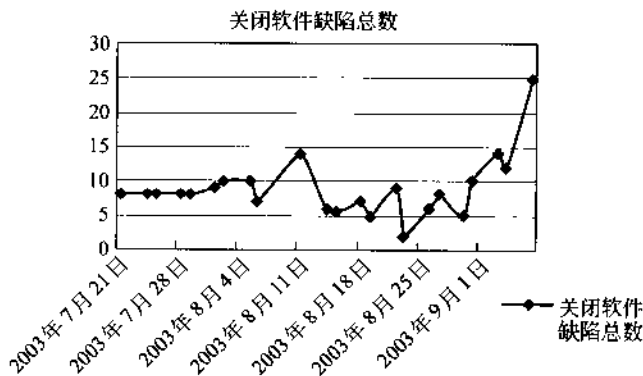


图 15-5 关闭软件缺陷周期图表

可接受的日常关闭曲线不显示朝向任何边界的明显倾斜。

一个稳定可接受的关闭周期图表指出了对一个理解良好、运行平稳的缺陷管理过程。理想情况是一个向下或水平趋向的关闭软件周期曲线。图 15-5 关闭软件缺陷周期图表中的关闭周期是稳定而可接受的。缺陷一般大约一个半星期内得到修正，这是一个良好的速度。

## 小结

本章讲解了应该遵循正规过程正确地描述、分离、分类、记录和跟踪软件缺陷，以保证它们最终得到解决修复。当项目处于测试执行阶段时，测试管理员管理缺陷主要由分析数据和收集数据组成，因此需要建立软件缺陷跟踪数据库存储、搜索和分析软件缺陷，从而生成一系列的图表，分析项目的发展趋势，找到薄弱的领域，合理地修复它。

## 思考题

1. 说出软件缺陷可能得不到修复的几个原因。
2. 怎么去描述软件缺陷？软件缺陷报告包括哪些组成部分？
3. 描述分离和再现软件缺陷的技术。

4. 软件缺陷生命周期的基本状态包括哪些?
5. 列举怎么组成软件缺陷跟踪数据库?
6. 如果某项目中软件缺陷发现速度下降,测试人员对项目即将关闭准备发布表示兴奋。请问可能有哪些原因会造成这种假象?

# 第 16 章 软件测试和质量分析报告

作为一个程序员如何写好代码是需要常常思考的问题，而作为一个测试人员如何写好测试报告同样是一项关键的工作。写好测试报告除了最基本的写作技能、规范的测试报告模板之外，更重要的是对测试评估、质量分析所进行的深入研究。在对系统或产品做出质量分析之前，首先要了解测试所执行的情况，要对测试人员或自己提出许多问题。

- 单元测试用什么方法？是否覆盖了程序的所有关键路径，满足程序中各种多分支条件？
- 集成测试是否对所有接口、参数进行了全面的测试？
- 系统测试是否包含了兼容性、安全性、恢复性等测试？如果做了，又是怎么进行的？
- 测试用例是怎么设计的？是否覆盖了用户特别的使用场景？
- 测试计划所要求的各项测试内容都完成了吗？
- 测试用例被 100% 执行了吗？
- 所有严重的 bug 都被修复了吗？

在这些问题得到肯定的答案之后，最难却最关键的问题是评估系统被测试的覆盖率。只有确认系统得到了充分的测试之后，针对测试结果所做出的分析才有很好的可靠性和准确性，测试人员对自己所得出的结论才有足够的信心。所以最后问题就归纳为：如何对测试过程和结果进行科学的、恰当的评估？

## 16.1 软件产品的质量度量

在谈软件产品质量度量之前，先了解软件度量的一些概念、内容和方法。软件度量是根据一定的规则，将数字或符号赋予系统、构件、过程等实体的特定属性，从而能清晰地理解该软件实体及其属性的量化表示。简而言之，软件度量就是对软件所包含的各种属性的量化表示。

软件度量可以提供对软件过程和软件产品深入了解的衡量指标，使组织能够更好地做出决策以达到目标。软件度量具有如下作用：

- 用数据指标表明验收标准。
- 监控项目进度和预见风险。
- 分配资源时进行量化均衡。
- 预计和控制产品的过程、成本和质量。

所以说软件度量是用来衡量软件过程质量和进行软件过程改进的重要手段。但是，为了保持数据的可靠性、客观性和准确性，必须保证度量结果不用于评价数据提供者的个人

工作绩效或素质。

### 16.1.1 软件度量的内容和分类

软件度量的根本目的是通过量化的分析和总结提高软件开发效率，降低软件缺陷和开发成本，提高软件产品质量。具体地说，包括以下四个方面。

- 收集信息：通过分析获得过程、产品、资源、环境的信息，确定以后预测的基线和模型。这是评估、预测、改进活动的基础。
- 预测：通过理解过程、产品各要素之间的关系建立模型，由已知的要素推算、估计其他要素，以便合理分配资源，合理制订计划。
- 评估：分析活动与计划的符合程度，确定是否有偏差，以便控制其执行。可以评估最终产品的质量，评估新技术的影响，评估过程改进对过程和产品的影响。
- 改进：根据得到的量化信息，可以帮助我们识别障碍物，查找问题的根源，以及能提高产品质量和过程效率的其他方法。与以前的量化信息比较，可以证实这些方法是否有效。

#### 1. 软件度量的相关概念

- 测量（Measurement）是对产品过程的某个属性的范围、数量、维度、容量或大小提供一个定量的指示。
- 度量（Metric）是对软件产品进行范围广泛的测度。它给出一个系统、构件或过程某个给定属性的度的定量测量。
- 指标（Indicator）是一个度量或一组度量的组合，即采用易于理解的形式，对软件过程、项目或产品质量提供更全面、深入的评价和了解，以利于过程和质量的分析。

#### 2. 有效软件度量的属性

- 简单的和可计算的。学习如何导出度量值应该是相对简单的，并且其计算不应该要求过多的工作量和时间。
- 经验和直觉上有说服力。度量应该符合软件工程师对于软件过程和产品的直觉概念，如测度模块内聚性的度量值应该随着内聚度的提高而提高。
- 一致的和客观的。度量不会产生二义性的结果，任何独立的第三方使用该软件的相同信息能够得到相同的度量值。
- 在其单位和维度的使用上是一致的。度量的数学计算应该使用不会导致奇异单位组合的测度。例如，把项目队伍的人员乘以程序中的编程语言的变量会引起一个直觉上没有说服力的单位组合。
- 编程语言独立。度量应该基于分析模型、设计模型或程序本身的结构，而不依赖于编程语言的句法和语法。
- 质量反馈的有效机制。度量会为软件开发效率、产品质量等提供积极的信息。

### 3. 软件度量的分类

软件度量一般可主要分为 3 类：

- 软件过程度量：用于过程的优化和改进。对于软件开发过程本身的度量，目的是形成组织的各种模型，作为对项目、产品的度量基础，以及对软件开发过程进行持续改进，提高软件生产力。过程度量往往不是直接进行，而是通过大量的项目度量分析、总结得出来的。典型的过程度量如 CMM 中 KPA 的度量。
- 软件项目度量：用于生产率评估和项目控制。对于软件开发项目的特定度量，目的是评估项目开发过程的质量，预测项目进度、工作量等，辅助管理者进行质量控制和项目控制。
- 产品质量度量：用于产品评估和决策，主要针对项目开发结果——最终产品的度量。一般来说，提到产品度量，指的是对产品的质量度量。

过程度量与项目度量的区别是：过程度量是战略性的，针对组织范围内进行，是对大量项目实践的总结和模型化，对于项目度量具有指导意义；项目度量是战术性的，针对具体的项目预测、评估、改进项目工作。产品度量是对产品质量的度量，用于对产品质量的评估和预测，在 16.1.3 节有详细介绍。过程度量的一个范例就是 CMM，将过程能力成熟度分为 5 个等级，详见第 4 章的介绍。

### 4. 项目度量的内容列表

- 规模度量：代码行数，以千行源代码（KLOC）为基准。它是工作量度量、进度度量的基础。
- 复杂度度量：有结构复杂度指标。预测软件产品各部分的复杂性，以便选择最可靠的程序设计方法，确定测试策略。
- 缺陷度量：帮助确定产品缺陷变化的状态，并标示修复缺陷活动所需的工作量。分析产品缺陷分布的情况，并指示需要加强何种研发活动，需要何种技术培训，预测产品的遗留缺陷情况，预测产品发布后缺陷的影响情况。
- 工作量度量：把任务分解并结合人力资源水平来度量，合理地分配研发资源和人力，获得最高的效率。
- 进度度量：通过任务分解、工作量度量、有效资源分配等做出计划，然后将实际结果和计划值进行对比来度量。
- 生产率度量：产生代码行数 / （人·月），测试用例数 / （人·日）。
- 风险度量：一般通过两个参数“风险发生的概率”和“风险发生后所带来的损失”来评估风险。
- 其他的项目动态度量（如需求更改、代码动态增长等）。

目前各个方面的软件度量技术都开始走向成熟，包括规模度量、工作量度量、缺陷度量、复杂度度量、可靠性度量等方面都有比较多的模型和方法可以利用。

## 16.1.2 软件度量的分工和过程

根据度量目标、内容和要求的不同,度量活动可能涉及一个项目的所有人员,也可能包括各种活动的数据收集与分析。一个完整的度量活动涉及的角色包括度量工作小组、数据提供者、IT支持者。

- 度量工作小组由专职的度量研究人员和项目协调人员组成。度量研究人员的主要职责是定义度量过程和指导进行度量活动,并对数据进行分析、反馈;项目协调人员是度量小组和项目组之间的联系人,其职责是为定义度量过程提供详细的需求信息,并负责度量过程在项目组的推行。
- 数据提供者一般是项目中的研发人员,有时还会包括用户服务人员和最终用户。数据提供者的职责主要是按照规定的格式向度量小组或IT支持者提供数据。
- IT支持者主要根据度量工作小组的需要,确定数据提供的格式与数据存储方式,提供数据收集工具与数据存储设备。

根据数据统计,度量活动所占用工作量总体上来说比较低,主要工作量由度量工作小组承担,IT支持者为5%~10%,而软件工程师作为数据提供者,其工作量仅占2%~4%,相当于每天只要花10~20分钟就可以完成数据提供任务。随着度量过程体系、IT支持工具的逐步完善,软件研发人员在度量活动上所花的时间将越来越少。以度量活动的分析结果为基础,可以提高劳动生产率和产品质量,其收益将远大于度量活动的成本。

为了说明软件度量的过程,这里以目标驱动的程度活动为例。目标驱动的软件度量活动主要包括5个阶段:

(1) 识别目标。根据管理者的不同要求,分析出度量的工作目标,并根据其优先级和可行性,得到度量活动的工作目标列表,并由管理者审核确认。

(2) 定义度量过程。根据各个度量目标,分别定义其收集要素、收集过程、分析、反馈过程、IT支持体系,为具体的收集活动、分析、反馈活动和IT设备、工具开发提供指导。具体的定义内容如下。

- 收集要素:定义收集活动和分析活动所需要的数据要素与收集表格。
- 收集过程:定义数据收集活动的形式、角色及数据的存储。
- 分析反馈过程:定义对数据分析方法和分析报告的反馈形式。
- IT支持体系:定义IT支持设备和工具,以协助数据收集和存储、分析。

(3) 搜集数据。根据度量过程的定义,数据提供者提供数据,IT支持者应用IT支持工具进行数据收集工作,并按指定的方式审查和存储。在规定的度量活动完成(或阶段性的度量活动完成)后,IT支持者输出数据收集结果给度量小组。

(4) 数据分析与反馈。度量小组根据数据收集结果,按照已定义的分析方法进行数据分析,完成规定格式的图表,向相关的管理者和数据提供者进行反馈。

(5) 过程改进。对于软件开发过程而言,根据度量的分析报告,管理者基于度量数据做出决策。这些决策可能包括滚动计划、纠正活动或不做改变就通过。

其中,“识别目标”和“定义度量过程”是保证成功搜集数据和分析数据的先决条件,



是度量过程最重要的阶段；“过程改进”是度量的最终目的。

对于软件度量过程而言，在改进过程中也评估度量过程自身的完备性。度量核心小组根据本次度量活动所发现的问题，将对度量过程做出改造，以提高度量活动的效率或者更加符合组织的商业目标。

先进的公司在软件开发的各个领域内广泛开展了软件度量活动，其对工作量的估计可以精确到一个人/天，对缺陷的预测可精确到各个模块的缺陷密度。通过采用包括软件度量在内的各种软件工程技术，这些公司在生产力水平和产品质量水平上得到了极大的提高。

### 16.1.3 软件质量模型

软件质量是指与软件产品满足规定、隐含的需求能力和有关特征的全体，即所有描述软件优秀程度的特性组合。应用软件的质量依赖于问题需求的描述、解决方案的建模设计、可执行程序的编码以及测试。为了更好地评价软件产品的质量，要将软件质量的特性组合转化为物理或数学模型。比较著名的质量模型有：

- 1976 年，Boehm 第一次提出了软件质量度量的层次模型，如图 16-1 所示。

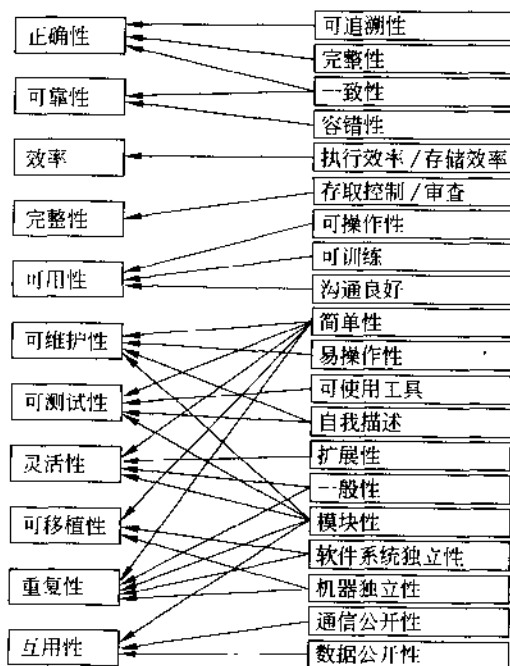


图 16-1 Boehm 软件质量度量模型

- 1978 年，McCall 等人提出了从软件质量要素、准则到度量的三层模型，如第 2 章的图 2-1 所示。
- 国际标准化组织（ISO）于 1985 年建议软件质量模型（ISO9126）由三层组成，其中第三层有用户定义，如图 16-2 所示。

- ◆ 高层：软件质量需求评价准则（SQRC）。
- ◆ 中层：软件质量设计评价准则（SQDC）。
- ◆ 低层：软件质量度量评价准则（SQMC）。

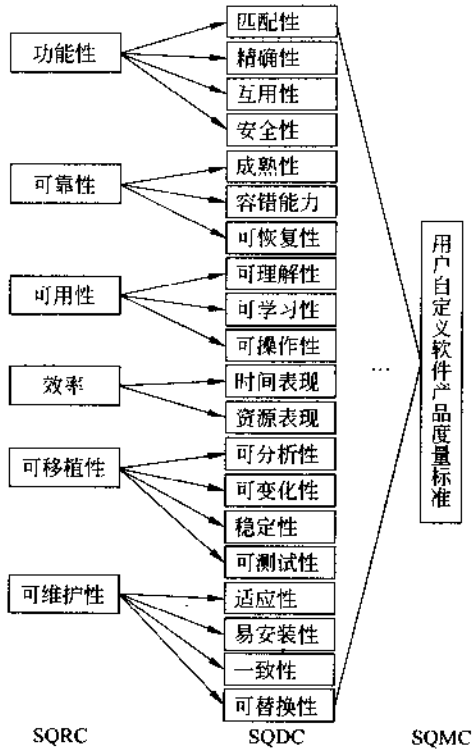


图 16-2 ISO9126 软件质量三层模型

在国家标准 GB/T 17544—1998（附录 C），GB / T 16260—1996（信息技术 软件产品评价 质量特性及其使用指南）中，对 ISO9126 软件质量模型中定义的质量标准有详细描述，概括如下。

- 功能性（functionality）：与现有的一组功能及其规定的性质有关的一组属性。这里的功能是指满足明确的或隐含需求的那些功能。
- 可靠性（reliability）：与在规定的一段时间和条件下，软件维持其性质水平的能力有关的一组属性。
- 易用性（usability）：与一组规定或潜在的用户为使用软件所需作的努力并且对这样使用所做评价有关的一组属性。
- 效率（efficiency）：与在规定的条件下，软件的性质水平和所使用资源量之间的关系有关的一组属性。
- 可维护性（maintainability）：与进行规定的修改所需的努力有关的一组属性。
- 可移植性（portability）：软件从某一环境转移到另一环境能力有关的一组属性。

## 16.1.4 软件质量的度量

软件需求是度量软件质量的基础,不符合需求的软件就不具备质量。定量的软件评估就是基于这样的原则来通过数学模型来实现,也就是尺度度量(metrics measurement)方法。这种定量度量适用于一些能够直接度量的特性,包括软件可靠性度量、复杂度度量、缺陷度量和规模度量,如程序出错率可定义为每千行代码(KLOC, kilometer lines of code)所含有的 bug 数。为了进行软件测试、质量的度量,需要根据质量模型(McCall 质量模型, Boehm 质量模型或 ISO9126)来准备足够的数,然后进行测试覆盖率、产品质量的量化评估分析。

- 明确性、正确性、可理解性、完全性、可验证性、一致性、简洁性、可追踪性、可修改性、精确性和可复用性的数据。这些数据可以用来评价分析模型和相应的质量表现特征。
- 公开的可能缺陷数与报告总缺陷数的对比则可以用来评价测试精确度和测试覆盖度,同时也可以预测项目发布时间。
- 产品发布前清除的缺陷数在总缺陷数中所占的百分比,有助于评估产品的质量。
- 按严重缺陷、子系统缺陷来划分,分类统计出平均修复时间,这样将有助于规划纠正缺陷的工作。
- 利用测试的统计数据,估算可维护性、可靠性、可用性和原有故障总数等数据。这些数据将有助于评估应用软件的稳定程度和可能产生的失败。

根据质量模型和上述观点,就可以用下列带加权因子的回归公式来度量质量:

$$M_i = c_1 \times f_1 + c_2 \times f_2 + \cdots + c_n \times f_n$$

$M_i$ 是一个软件质量因素(如 SQRC 层各项待计算值), $f_n$ 是影响质量因素的度量值(如 SQDC 层各项估计值), $c_n$ 是加权因子。部分度量值可以获得统计数据结果,还有部分的度量值难以得到准确的量化值,要靠主观测度,如通过检查表的形式,给这些特定属性进行评分。

## 16.1.5 质量度量的统计方法

质量度量的统计方法,是对质量评估量化的一种比较常用的方法,并且有不断增长的趋势。质量度量的统计方法包含以下步骤:

- (1) 收集和分类软件缺陷信息。
- (2) 找出导致每个缺陷的原因(例如不符合规格说明书、设计错误、代码错误、数据处理不对、对客户需求误解、违背标准、界面不友好等)。
- (3) 使用 Pareto 规则(80%缺陷主要是由 20%的主要因素造成的,20%缺陷是由另外 80%的次要因素造成的),要将这 20%的主要因素分离出来。
- (4) 一旦标出少数的主要因素,就比较容易纠正引起缺陷的问题。

为了说明这一过程,假定软件开发组织收集了为期一年的缺陷信息。有些错误是在软

件开发过程中发现的,其他缺陷则是在软件交付给最终用户之后发现的。尽管发现了数以百计的不同类型的错误,但是所有错误都可以追溯到下述原因中的一个或几个:

- 说明不完整或说明错误 (IES)。
- 与客户交流不够所产生的误解 (MCC)。
- 故意与说明偏离 (IDS)。
- 违反编程标准 (VPS)。
- 数据表示有错 (EDR)。
- 模块接口不一致 (IMI)。
- 设计逻辑有错 (EDL)。
- 不完整或错误的测试 (IET)。
- 不准确或不完整的文档 (IID)。
- 将设计翻译成程序设计语言中的错误 (PLT)。
- 不清晰或不一致的人机界面 (HCI)。
- 杂项 (MIS)。

为了使用质量度量的统计方法,需要收集上述各项数据,如表 16-1 所示,表中显示 IES、MCC、EDR 和 IET 占有所有错误的近 62%,是影响质量的主要原因。但如果只考虑那些严重影响产品质量的因素,少数的主要原因就变为 IES、EDR、PLT 和 EDL。一旦确定了什么是少数的主要原因 (IES、EDR 等),软件开发组织就可以集中到这些领域采取改进措施,质量改善的效果会非常明显。例如,为了减少与客户交流不够所产生的误解 (改正 MCC),在产品规格设计说明书中尽量不用专业术语,即使用了专业术语,也要定义清楚,以提高与客户的通信及说明的质量。为了改正 EDR (数据表示有错),不仅采用 CASE 工具进行数据建模,而且对数据字典、数据设计要实施严格的复审制度。

表 16-1 质量度量的统计数据收集

总计 (E <sub>i</sub> )			严重 (S <sub>i</sub> )		一般 (M <sub>i</sub> )		微小 (T <sub>i</sub> )	
错误	数量	百分比	数量	百分比	数量	百分比	数量	百分比
IES	296	22.3%	55	28.2%	95	18.6%	146	23.4%
MCC	204	15.3%	18	9.2%	87	17.0%	99	15.9%
IDS	64	4.8%	2	1.0%	31	6.1%	31	5.0%
VPS	34	2.6%	1	0.5%	19	3.7%	14	2.2%
EDR	182	13.7%	38	19.5%	90	17.6%	54	8.7%
IMI	82	6.2%	14	7.2%	21	4.1%	47	7.5%
EDL	64	4.8%	20	10.3%	17	3.3%	27	4.3%
IET	140	10.5%	17	8.7%	51	10.0%	72	11.6%
IID	54	4.1%	3	1.5%	28	5.5%	23	3.7%
PLT	87	6.5%	22	11.3%	26	5.1%	39	6.3%
HCI	42	3.2%	4	2.1%	27	5.3%	11	1.8%
MIS	81	6.1%	1	0.5%	20	3.9%	60	9.6%
总计	1330	100%	195	100%	512	100%	623	100%

当与缺陷跟踪数据库结合使用时，我们可以为软件开发周期的每个阶段计算“错误指标”。针对需求分析、设计、编码、测试和发布各个阶段，收集到以下数据：

- $E_i$  = 在软件工程过程中的第  $i$  步中发现的错误总数
- $S_i$  = 严重错误数
- $M_i$  = 一般错误数
- $T_i$  = 微小错误数
- $P_i$  = 第  $i$  步的产品规模（LOC、设计说明、文档页数）

$W_s$ 、 $W_m$ 、 $W_t$  分别是严重、一般、微小错误的加权因子，一般建议取值为  $W_s=10$ 、 $W_m=3$  和  $W_t=1$ ，作者建议取值为  $W_s=0.6$ 、 $W_m=0.3$  和  $W_t=0.1$ （构成 100%）。所以每个阶段的错误度量值可以表示为  $PI_i$ ：

$$PI_i = W_s (S_i / E_i) + W_m (M_i / E_i) + W_t (T_i / E_i)$$

最终的错误指标 EP 通过计算各个  $PI_i$  的加权效果得到。考虑到软件测试过程中越到后而发现的错误其权值越高，简单用 1,2,3,... 序列表示，则 EP 为：

$$EP = \sum (i \times PI_i) / PS, \quad \text{其中 } PS = \sum PI_i$$

错误指标与表 16-1 中收集的信息相结合，可以得出软件质量的整体改进指标。

质量度量的统计方法告诉我们：将时间集中用于解决主要的问题之上，首先就必须知道哪些是主要因素，而这些主要因素可以通过数据收集、统计方法等分离出来，从而可以真正有效地提高产品质量。实际上，大多数严重的缺陷都可以追溯到少数的根本原因之上，常常和我们的直觉比较接近，但是很少有人花时间收集数据来验证他们的感觉。

## 16.2 评估系统测试的覆盖程度

为什么要做软件测试评估呢？如果没有测试评估，就没有测试覆盖率的结果，就没有报告测试进程的根据。软件测试评估主要有两个目的，一是量化测试进程，判断测试进行的状态，决定什么时候测试可以结束；二是为最后的测试或质量分析报告生成所需的量化数据，如缺陷清除率、测试覆盖率等。

测试评估是软件测试的一个阶段性结论，是用所生成的测试评估报告来确定测试是否达到完全和成功的标准。测试评估可以说贯穿整个软件测试过程，可以在测试每个阶段结束前进行，也可以在测试过程中某一个时间进行。

系统的测试活动建立在至少一个测试覆盖策略基础上，而覆盖策略试图描述测试的一般目的，指导测试用例的设计。如果测试需求已经完全分类，则基于需求的覆盖策略可能足以生成测试完全程度评测的量化指标。例如，如果已经确定了所有性能测试需求，则可以引用测试结果来得到评测，如已经核对了 90% 的性能测试需求。测试评估工作主要是对测试覆盖率的评估，测试覆盖率是用来衡量测试完成多少的一种量化标准。

测试的覆盖率，可以用测试项目的数量和内容进行度量。除此之外，如果测试软件的数量较大，还要考虑数据量。测试的覆盖率，可以根据表 16-2 所示对测试指标进行评价。通过检查这些指标达到的程度，就可以度量出测试内容的覆盖程度。

表 16-2 测试覆盖程度表

测试覆盖项	测试覆盖率指标测试描述	测试结果
界面覆盖	符合需求（所有界面图标、信息区、状态区）	
静态功能覆盖	功能满足需求	
动态功能覆盖	所有功能的转换功能正确	
正常测试覆盖	所有硬件软件正常时处理	
异常测试覆盖	硬件或软件异常时处理（不允许的操作）	测试结束判断

对测试覆盖率的评估就是要确定测试执行的完全程度，其基本方法有基于需求的测试覆盖评估和基于代码的测试覆盖评估。

16.2.1 对软件需求的估算

在讨论基于需求的测试覆盖评估之前，先讨论对软件需求的估算。软件需求的估算有利于对测试需求的把握，有利于进一步测试覆盖率的估算。

假设在一个规格设计说明书中有 R 个需求，其功能需求的数目为 F，非功能需求数（如性能）为 N 则： $R = F + N$  为了确定需求的确定性，一种基于复审者对每个需求解释的一致性的度量方法为：

$$Q1 = M/R$$

其中 Q1 表示需求的确定性，M 是所有复审者都有相同解释的需求数目。当需求的模糊性越低时，Q1 的值越接近 1。而功能需求的完整性 Q2 则可以通过计算以下比率获得：

$$Q2 = Fu / (Ni \times Ns)$$

其中 Fu 是惟一功能需求的数目，Ni 是由规格设计说明书定义的输入个数，Ns 是被表示的状态个数。Q2 只是测度了一个系统所表示的必需功能百分比，但是它并没有考虑非功能需求。为了把这些非功能需求结合到整体度量中以求完整，必须考虑已有需求已经被确认的程度。这可以用 Q3 来表示：

$$Q3 = Fc / (Fc + Fnv)$$

其中 Fc 是已经确认为正确的需求个数，Fnv 是尚未被确认的需求个数。

16.2.2 基于需求的测试覆盖评估

基于需求的测试覆盖评估依赖于对已执行/运行的测试用例的核实和分析，所以基于需求的测试覆盖评测就转换为评估测试用例覆盖率，测试的目标是确保 100% 的测试用例全部成功地执行。如果这个目标不可行或不可能达到，则要根据不同的情况制订不同的测试覆盖标准。主要考虑风险和严重性、可接受的覆盖百分比。

在执行测试活动中，评估测试用例覆盖率又可以分为两类测试用例覆盖率估算：

(1) 确定已经执行的测试用例覆盖率，即在所有测试用例中有多少测试用例已被执行。假定 Tx 是已执行的测试过程数或测试用例数，R<sub>ft</sub> 是测试需求的总数：

$$\text{已执行的测试覆盖} = T_x / R_{ft}$$

(2) 确定成功的测试覆盖, 即执行时未出现失败的测试, 如没有出现缺陷或意外结果的测试, 假定  $T_s$  是已执行的完全成功、没有缺陷的测试过程数或测试用例数:

$$\text{成功的测试覆盖} = T_s / R_{ft}$$

在实际过程中, 很难确定  $R_{ft}$  的值, 可以根据需求点、以往经验来估算。

### 16.2.3 基于代码的测试覆盖评估

基于代码的测试覆盖评测是对被测试的程序代码语句、路径或条件的覆盖率分析。如果应用基于代码的覆盖, 则测试策略是根据测试已经执行源代码的多少来表示的。这种测试覆盖策略类型对于安全至上的系统来说非常重要。

评估代码覆盖率, 需要断定测试目标期望的、总的测试代码行数, 在测试中真正执行的代码行数及其百分比, 将此结果记录在测试评估报告中。

测试过程中已经执行的代码的多少, 与之相对的是要执行的剩余代码有多少。代码覆盖可以建立在控制流(语句、分支或路径)或数据流的基础上。控制流覆盖的目的是测试代码行、分支条件、代码中的路径或软件控制流的其他元素。数据流覆盖的目的是通过软件操作测试数据状态是否有效, 例如, 数据元素在使用之前是否已经定义。

基于代码的测试覆盖通过以下公式计算:

$$\text{已执行的测试覆盖} = T_c / T_{nc}$$

其中  $T_c$  是用代码语句、条件分支、代码路径、数据状态判定点或数据元素名表示的已执行项目数,  $T_{nc}$  (total number of items in the code) 是代码中的项目总数。

## 16.3 软件缺陷分析方法

质量是反映软件与需求相符程度的指标, 而缺陷被认为是软件与需求不一致的某种表现, 所以通过对测试过程中所有已发现的缺陷进行评估, 可以了解软件的质量状况。也就是说, 软件缺陷评估是评估软件质量的重要途径之一, 软件缺陷评估指标可以看做是度量软件质量的重要指标, 而且缺陷分析也可以用来评估当前软件的可靠性, 并且预测软件产品的可靠性变化, 缺陷分析在软件可靠性评估中占有相当大的作用。

软件缺陷评估的方法相对比较多样, 从简单的缺陷计数到严格的统计建模, 16.1.4 节质量度量的统计方法就是一个例子。通常软件缺陷评估模型假设缺陷的发现是呈泊松分布的, 则有关缺陷率的实际数据可以适用于这一模型, 但更严格的缺陷评估要考查在测试过程中发现缺陷的实际间隔时间。

对缺陷进行分析, 确定测试是否达到结束的标准, 也就是判定测试是否已达到用户可接受的状态。在评估缺陷时应遵照缺陷分析策略中制订的分析标准。最常用的缺陷分析方法有四种。

- 缺陷分布报告: 允许将缺陷计数作为一个或多个缺陷参数的函数来显示, 生成缺

陷数量与缺陷属性的函数。如测试需求和缺陷状态、严重性的分布情况等。

- 缺陷趋势报告：按各种状态将缺陷计数作为时间的函数显示。趋势报告可以是累计的，也可以是非累计的，可以看出缺陷增长和减少的趋势。
- 缺陷年龄报告：是一种特殊类型的缺陷分布报告，显示缺陷处于活动（active, open）状态的时间，展示一个缺陷处于某种状态的时间长短，从而了解处理这些缺陷的进度情况。
- 测试结果进度报告：展示测试过程在被测应用的几个版本中的执行结果以及测试周期，显示对应用程序进行若干次迭代和测试生命周期后的测试过程执行结果。

这些类型的分析为软件质量、可靠性的特性、变化趋势等提供了判断依据。例如，预期缺陷发现率将随着测试进度和修复进度而最终减少，这样可以设定一个阈值，在缺陷发现率低于该阈值时才能部署该软件。

对于缺陷分析，常用的主要缺陷参数有4个。

- 状态：缺陷的当前状态，如激活的（active）、修复的（fixed）或关闭的（closed）。
- 优先级：必须处理和解决缺陷的相对重要性。
- 严重性：缺陷对产品功能使用、用户数据等相关影响等。
- 起源：导致缺陷产生的主要因素，或排除该缺陷需要修复的构件。

下面就结合这些属性来进行缺陷分析、生成缺陷分析报告。

### 16.3.1 缺陷分布报告

如第15章所给定的缺陷优先级（priority），每个软件缺陷的优先级被设定为其中一种：

- 立即解决（P1级）
- 高优先级（P2级）
- 正常排队（P3级）
- 低优先级（P4级）

在测试报告中，主要分析前三种高优先级在总体上或功能上的分布情况。总体上各级别的缺陷数量应该遵守这样一个规律： $P1 < P2 < P3$ 。如图16-3所示，说明代码质量不好，不符合正常缺陷分布，需要进一步分析，找出其根本原因。

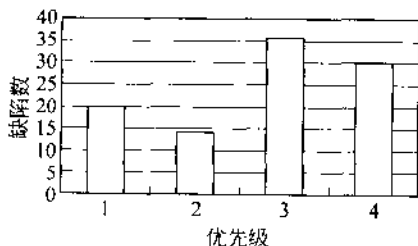


图 16-3 缺陷的四种优先级分布

对功能上的分布情况分析，可以了解哪些功能模块处理比较难、哪些功能模块程序质量比较差，有利于程序质量的改进和提高。进一步分析，可以计算出 P1 优先级缺陷从报告



到关闭所需要的平均时间,从而可以知道开发人员是否按照要求去做,一般来说 P1 优先级缺陷规定必须在 8 小时之内被解决。

缺陷起源分布报告(如 15 章图 15-4 所示),显示导致缺陷的根本原因(实施模型元素)上的分布情况,这种分析会帮助程序代码质量的改进。

### 16.3.2 缺陷趋势报告

一般测试标准要求:在测试结束前 P1、P2 两类优先级缺陷必须被全部处理完,所以有必要生成 P1、P2 两类优先级缺陷在时间上的分布图,以决定整个产品开发是否按预期进度进行,测试是否可以结束。在测试报告中,主要分析前三种高优先级的趋势变化。

在一个成熟的软件开发过程中,缺陷趋势会遵循着一种和预测比较接近的模式向前发展。在生命周期的初期,缺陷率增长很快。在达到顶峰后,就随时间以较慢的速率下降,如图 16-4 所示。可以根据这一趋势复审项目时间表,如果在 4 个星期的测试周期中,缺陷率在第三个星期仍然增长,则很明显项目有问题,需要审视代码质量、调整时间表。

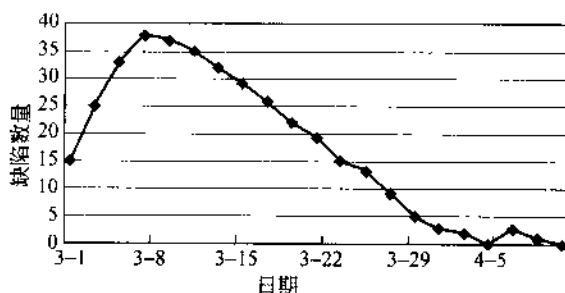


图 16-4 软件测试过程中发现缺陷数理想趋势图

实际测试过程中,可能出现一些波动现象,而且测试过程要经过单元测试、集成测试、功能测试、系统测试、验收测试等不同的阶段,其波动趋势会表现为周期性。

这种趋势分析还可以延伸到用来评估开发所做出的努力,就是对已修复的、已关闭的软件缺陷跟踪。理想情况下,已修复的、已关闭的缺陷数和所发现的缺陷数发展趋势相同或相近,只有滞后效应。特别是 P1、P2 优先级的缺陷,要及时被修复、关闭,这种关闭缺陷的速率应该维持在与打开缺陷的速率相同的水准上,滞后时间不宜超过 3 天,才能保证项目顺利进行。这种趋势图,以“激活的、修复的、关闭的”不同状态的缺陷累计数量来分析比较好,如图 15-3 就是一个例子,在项目开始时,新发现缺陷的速率快,关闭的缺陷速度也快,但随着时间推移,该速率不断降低,关闭的趋势与新缺陷的趋势相似,但滞后一周。

图 16-5 给出了一个理想趋势图。如果情况并非如此,这些趋势之间差别显著,则表明存在问题,缺陷处理流程有问题或修复缺陷所需的资源不足或回归测试策略不对等。当与测试覆盖评测结合起来时,缺陷分析可提供出色的评估,测试完成的标准也可以建立在此评估基础上。

缺陷龄期、缺陷发现率等分析,也能提供有关测试有效性和缺陷排除活动的良好反馈。

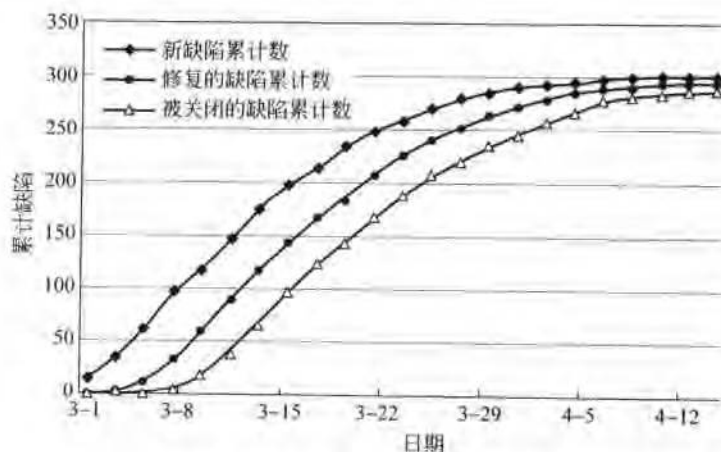


图 16-5 新发现的、修复的、关闭的累计缺陷数的理想趋势图

例如，如果大部分龄期较长的、未解决的缺陷处于有待确认的状态，则可能表明没有充足的资源应用于再次测试工作。

微软公司就利用发现的缺陷数和关闭的缺陷数趋势图，找出缺陷的收敛点，来预测产品的下一个阶段计划如图 16-6 所示。当出现没有激活状态缺陷的第一个时间，被定义为零 bug 反弹点 (ZBB, zero bug bounce)，从这一时刻开始，产品进入稳定期。

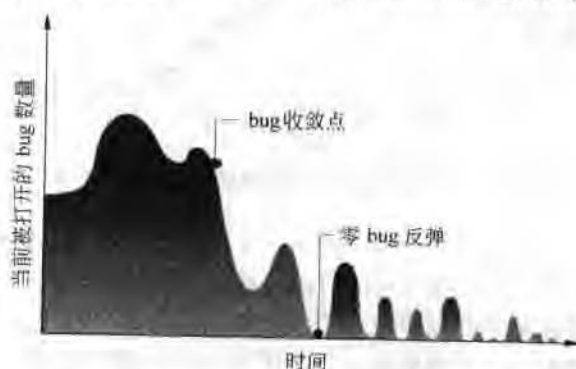


图 16-6 微软公司基于缺陷趋势图的里程碑定义

## 16.4 基于缺陷分析的产品质量评估

软件评估首先是建立基线，为软件产品的质量、软件测试评估设置起点，这个基准线上再设置测试的目标，作为对系统评估是否通过的标准。

缺陷评测的基线是对某一类或某一组织结果的一种度量，这种结果可能是常见的或典型的，如 10 000 行源程序 (LOC) 是程序规模的一个基准，每一千行代码有 3 个错误是测试中错误发现率的基准。基准对期望值的管理有很大帮助，目标就是相对基准而存在，也就是定义可接受行为的基准，如表 16-3 所示。

表 16-3 某个软件项目基准和目标

条 目	目 标	低 水 平
缺陷清除效率	>95%	<70%
原有缺陷密度	每个功能点 <4	每个功能点 >7
超出风险之外的成本	0%	>=10%
全部需求功能点	<1% 每个月平均值	>=50%
全部程序文档	每个功能点页数 <3	每个功能点页数 >6
员工离职率	每年 1%~3%	每年>5%

基于缺陷分析的产品质量评估方法有以下几种：

- 经典的种子公式
- 基于缺陷清除率的估算方法
- 软件产品性能评估技术
- 借助工具的方法

### 16.4.1 经典的种子公式

Mills 研究出通过已知缺陷（称为种子 bug）来估计程序当中潜在的、未知的缺陷数量。其基本前提是将测试队伍分为两个小组，一个小组事先将已知的共  $S$  个 bug（种子）安插在程序里，然后，让另一个测试小组尽可能发现程序的 bug，假如他们发现了  $s$  个种子 bug，则认为存在这样一个等式：

$$\frac{\text{已测试出的种子bug}(s)}{\text{所有的种子bug}(S)} = \frac{\text{已测试出的非种子bug}(n)}{\text{全部的非种子bug}(N)}$$

则可以推出程序的总 bug 数为：

$$N = S \times n/s$$

其中  $n$  是进行实际测试时所发现的 bug 总数。如果  $n = N$ ，说明所有的 bug 已找出来，说明做的测试足够充分。这种测试是否充分，可以用一个信心指数来表示，即用一个百分比表示，值越大，说明对产品质量的信心越高，最大值为 1。

$$C = \begin{cases} 1, & \text{if } n > N \\ S/(S - N + 1) & \text{if } n \leq N \end{cases}$$

但是这种假定本身的可能性就比较小，因为种子 bug 很难具有完全的代表性，根据相似系统确定的 bug 其结果可能差别很大。另外，人为设置程序的 bug，这工作本身就比较困难，要将正确的程序改为错误的程序，会引起其他的一些问题，即插入一个缺陷可能会引起 2~3 个缺陷，而且缺陷相互之间可能存在相互影响或有关联关系，虽然事先设定插入 20 个种子 bug，但结果可能是在程序中插入了 26、27 个种子 bug，所以按照上述计算的公式就不准确。

16.4.2 基于缺陷清除率的估算方法

让我们先引入几个变量，F 为描述软件规模用的功能点，D1 为在软件开发过程中发现的所有缺陷数，D2 为软件发布后发现的缺陷数，D 为发现的总缺陷数。因此， $D=D1+D2$ 。

对于一个应用软件项目，则有如下计算方程式（从不同的角度估算软件的质量）：

- 质量= $D2/F$ ；
- 缺陷注入率= $D/F$ ；
- 整体缺陷清除率= $D1/D$ ；

假如有 100 个功能点，即  $F=100$ ，而在开发过程中发现了 20 个错误，提交后又发现了 3 个错误，则： $D1=20$ ， $D2=3$ ， $D=D1+D2=23$ 。

- 质量（每功能点的缺陷数）= $D2/F=3/100=0.03$ （3%）。
- 缺陷注入率= $D/F=20/100=0.20$ （20%）
- 整体缺陷清除率= $D1/D=20/23=0.8696$ （86.96%）

有资料统计，美国的平均整体缺陷清除率目前只达到大约 85%，而对一些具有良好的管理和流程的著名软件公司，其主流软件产品的缺陷清除率可以超过 98%。

众所周知，清除软件缺陷的难易程度在各个阶段也是不同的。需求错误、规格说明、设计问题及错误修改是最难清除的，如表 16-4 所示。

表 16-4 不同缺陷源的清除效率

缺陷源	潜在缺陷	清除效率（%）	被交付的缺陷
需求报告	1.00	77	0.23
设计	1.25	85	0.19
编码	1.75	95	0.09
文档	0.60	80	0.12
错误修改	0.40	70	0.12
合计	5.00	85	0.75

表 16-5 反映的是 CMM 五个等级如何影响软件质量的，其数据来源于美国空军 1994 年委托 SPR（美国一家著名的调查公司）进行的一项研究。从表中可以看出，CMM 级别越高，缺陷清除率也越高。

表 16-5 SEI CMM 级别潜在缺陷与清除

SEI CMM 级别	潜在缺陷	清除效率（%）	被交付的缺陷
1	5.00	85	0.75
2	4.00	89	0.44
3	3.00	91	0.27
4	2.00	93	0.14
5	1.00	95	0.05

### 16.4.3 软件产品性能评估

软件产品性能评估其技术性相对较强，方法的基础是获取与性能表现相关的数据，如响应时间、数据吞吐量、数据流速率、操作可靠性等。性能评测一般和测试的执行结合起来做，或者是在执行测试时记录、保存各种数据，然后在评估测试活动中进行计算结果。主要的性能评测包括：

- 动态监测：在测试执行过程中，实时获取并显示正在监测指标的状态数据，通常以柱状图或曲线图的形式提供实时显示，来监测或评估性能测试执行情况。
- 响应时间 / 吞吐量：测试对象针对特定条件下某个要测量特性的相关性能行为，用响应时间或吞吐量来进行量化评测。这些报告通常用曲线图、统计图来表示。
- 百分比报告：对已收集数据的百分比评测和计算。
- 比较报告：比较不同性能测试的结果，以评估测试执行过程之间所做的变更对性能行为的影响，从而进一步分析不同测试执行情况的多个数据集之间的差异或趋势。
- 追踪报告：当性能行为可以接受时或性能监测表明存在可能的瓶颈时，追踪报告可能是最有价值的报告。追踪和配置文件报告显示低级信息。该信息包括主角与测试对象之间的消息、执行流、数据访问以及函数和系统调用等。

### 16.4.4 借助工具的方法

不使用专门的软件工具进行数据输入任务和相应的评估活动，要进行软件测试评估工作是很难进行的，至少使得这项工作变得繁重。许多自动化测试工具可以根据测试运行时所走过的程序路径来计算测试的覆盖率，程序能识别哪些程序路径、逻辑路径或输入条件被测试过，而整个程序的相应值也能确定。如借助软件测试工具来获得测试的覆盖率：

- 程序语句行的覆盖率。
- 程序分支、条件的覆盖率。
- 程序路径的覆盖率。

如使用 Rational PureCoverage 能自动找出代码中未经测试的代码，保证代码测试覆盖率，还可针对每次测试生成全面的覆盖率报告，可以归并程序多次运行所生成的覆盖数据，并自动比较测试结果，以评估测试进度。当然，还可以使用 Rational 的其他一些工具，如：

- TestManager，复审和评估基于需求的测试覆盖。
- TestFactory，评估测试覆盖，包括评估脚本的、基于代码的测试覆盖。
- LogViewer，评估测试的执行情况。

## 16.5 测试报告的模板、实例

在国家标准 GB/T 17544—1998（附录 C）对测试报告有了具体要求，也就是对测试对象（软件程序、系统、产品等）有一个清楚地描述，对测试记录、测试结果如实汇总分析，

报告出来。测试报告应具有如下结构：

- 产品标识。
- 用于测试的计算机系统。
- 使用的文档及其标识。
- 产品描述、用户文档、程序和数据的测试结果。
- 与要求不符的清单。
- 针对与建议的要求不符的清单，产品未作符合性测试的说明。
- 测试结束日期。

主要内容集中在第四项内容，即产品描述、用户文档、程序和数据的测试结果。在产品描述中提供关于用户文档、程序以及数据（如果有的话）的信息，其信息描述应该是正确的、清楚的、前后一致的、容易理解的、完整的并且易于浏览的。更重要的是，在测试报告中，产品的描述是和测试的内容有着相对应的关系，也就是说产品描述还要包含功能说明、可靠性说明、易用性说明和效率、可维护性、可移植性说明，特别在功能说明中，不仅需要概述产品的用户可调用功能、需要的数据等，而且将系统相应的边界值、安全性要求描述清楚。对易用性说明中要包括对用户界面、所要求的知识、适应用户的需要、防止侵权行为、使用效率和用户满意度等方面的要求。

对于用户文档，测试的标准比较清楚，就是完整性、正确性、一致性、易理解性和易浏览性。对于程序和数据的测试，需要从功能、可靠性、易用性和效率、可维护性、可移植性等方面进行测试，并在报告中反映出来。对前三项测试结果，要求更高些，即：

- 功能性，包括安装、功能表现，以及功能使用的正确性、一致性。
- 可靠性，系统不应陷入用户无法控制的状态，既不应崩溃也不应丢失数据。即使在下列情况下也应满足可靠性要求：
  - ◆ 使用的容量到达规定的极限。
  - ◆ 企图使用的容量超出规定的极限。
  - ◆ 由产品描述中列出的其他程序或用户造成的错误输入。
  - ◆ 用户文档中明确规定的非法指令。
- ◆ 易用性包括易理解性，易浏览性，可操作性三个方面。

这里给出一个测试分析报告的模板

---

## 1 引言（概述）

### 1.1 编写目的

说明这份测试分析报告的具体编写目的，指出预期的阅读范围。如：

- 通过对测试结果的分析得到对软件的评价。
- ◆ 为纠正软件缺陷提供依据。
- 使用户对系统运行建立信心。

### 1.2 背景

对被测试对象进行简单介绍、说明，如：

- a. 被测试软件系统的名称。

b. 该软件的任务提出者、开发者、用户, 指出测试环境与实际运行环境之间可能存在的差异以及这些差异对测试结果的影响。

### 1.3 定义

列出本文件中用到的或所涉及到的专业术语、缩写词的定义。

### 1.4 参考资料

说明软件测试所需的资料(需求分析、设计规范等), 列出要用到的参考资料, 如:

- a. 本项目经核准的测试计划书、测试需求分析报告。
- b. 属于本项目其他已批准的文件, 如需求文档规格说明、系统设计等文档。
- c. 本文件中各处引用的文件、资料, 包括所要用到的软件开发、测试标准。

## 2 测试对象和概要

包括测试项目、测试类型、测试阶段、测试方法、测试时间等。

用表格的形式列出每一项测试的标识符及其测试内容, 并指明实际进行的测试工作内容与测试计划中预先设计的内容之间的差别, 说明做出这种改变的原因。

## 3 测试结果及发现

### 3.1 测试 1 (标识符)

把本项测试中实际得到的动态输出(包括内部生成数据输出)结果同对于动态输出的要求进行比较, 陈述其中的各项发现。

### 3.2 测试 2 (标识符)

用类似 3.1 条的方式给出第 2 项及其以后各项测试内容的测试结表和发现。

## 4 对软件功能的结论

### 4.1 功能 1 (标识符)

#### 4.1.1 能力

简述该项功能, 说明为满足此项功能而设计的软件能力以及经过一系列测试已证实的能力。

#### 4.1.2 限制

说明测试数据值的范围(包括动态数据和静态数据), 列出就这项功能而言, 测试期间在该软件中查出的缺陷、局限性。

### 4.2 功能 2 (标识符)

用类似 4.1 的方式给出第 2 项及其后各项功能的测试结论。

.....

## 5 分析摘要

### 5.1 测试结果分析

列出测试结果分析记录, 并按所定义的模板产生 BUG 分布表和 BUG 分布图。从软件测试中发现的并最终确认的错误点等级数量来评估。

#### 5.1.1 对比分析

若非首次测试时, 将本次测试结果与首次测试、前一次测试的结果进行对比分析比较。

#### 5.1.2 测试评估

通过对测试结表的分析提出一个对软件能力的全面分析, 需标明遗留缺陷、局限性和

软件的约束限制等，并提出改进建议。

#### 5.2 能力

陈述经测试证实了的软件能力。如果所进行的测试是为了验证一项或几项特定性能要求的实现，应提供这方面的测试结果与要求之间的比较，并确定测试环境与实际运行环境之间可能存在的差异对能力的测试所带来的影响。

#### 5.3 缺陷和限制

陈述经测试证实的软件缺陷和限制，说明每项缺陷和限制对软件性能的影响，并说明全部测得的性能缺陷的累积影响和总影响。

#### 5.4 建议

对每项缺陷提出改进建议，如：

- a. 各项修改可采用的修改方法。
- b. 各项修改的紧迫程度。
- c. 各项修改预计的工作量。
- d. 各项修改的负责人。

#### 5.5 评价

说明该项软件的开发是否已达到预定目标，能否交付使用。

### 6 测试资源消耗

总结测试工作的资源消耗数据，如工作人员的水平级别数量、机时消耗等。

---

## 小结

代码和相应的文档是开发人员的主要工作成果，作为软件测试人员，其主要成果就是提交一份有足够信息的、分析透彻的测试报告或质量报告。为了提交这样一份信息准确而全面的报告，需要对测试进行全面的评价，就是要对已做过的测试和测试结果等前后两方面进行评估。概括起来，就是软件测试覆盖评估和测试结果的质量分析：

- 测试覆盖是对测试完全程度的评测，它建立在测试覆盖基础上。测试覆盖是由测试需求和测试用例的覆盖或已执行代码的覆盖表示的。
- 质量分析是对测试对象（被测系统或软件产品）的可靠性、稳定性以及性能的评测。质量建立在对测试结果的评估和对测试过程中变更请求分析的基础上。

从广义上说，测试覆盖不仅对测试量化分析，而且对已经完成的测试过程进行审查、评估，包含审查测试计划、测试用例、测试环境和测试用例的执行情况，看是否有漏洞、疏忽的地方，如果有，需要及时补救。

软件测试和质量分析报告的依据就是产品规格说明书、系统设计文档、测试计划书和对实际系统的测试数据，重点集中在缺陷历史数据的分析之上。通过缺陷分析，了解测试的进程、产品质量的当前状况，可以判断软件主要问题在哪些模块或什么主要原因引起相应的问题。软件质量度量和分析，不仅可以帮助写出高质量的测试报告，同时可以帮助开发人员解决问题，帮助产品管理人员做出是否发布产品的决定。



## 思考题

1. McCall 的质量因素是在 20 世纪 70 年代提出的，自从被提出来后几乎计算的每个方面改动都很大，但是，McCall 的因素继续应用到现代软件。根据这个事实能得出什么结论吗？

2. 为什么没有一个单一的、全包容的对程序复杂度或程序质量的度量？

3. 软件缺陷分析还有哪些其他的方法？

4. 开发一个小型的软件工具用来对软件缺陷进行趋势分析，计算出测试效率和修复 bug 的效率。

5. 基于需求的测试覆盖评估和基于代码的测试覆盖评估，哪一种是更有效的方法？

6. 如何设计一套基于经验的产品质量评估方法？

# 第 17 章 软件测试项目管理

《现代汉语词典》对管理一词的解释是：“负责某项工作使顺利进行”。由此可见，可以把管理看成是一个保障体系，它深入到我们所着手的工作的各个方面以确保该项工作的顺利和成功。这里我们要讨论的是对软件测试项目的管理。对于项目管理，又有什么不同的定义呢？PMI（Project Management Institute）给了一个比较准确的定义：“Project management is the applications of knowledge, skills, tools, techniques to project activities in order to meet or exceed stakeholder needs and expectations from the project”，即项目管理是在项目活动中运用一系列的知识、技能、工具和技术，以满足并超过相关利益者对项目的要求和期望。这里也指出了项目管理涉及的范畴和需要达到的目标。

使项目顺利进行并达到预期的效果，这就是项目管理所应该实现的基本目标。当然，能够在管理的过程中不断地提升目标，超越预定的目标，这是更高层次的项目管理。那么软件项目管理的目标就是为了使软件项目能够按照预定的成本、进度、质量顺利完成，而对成本、资源、进度、质量、风险等进行分析和控制的活动。对于软件测试项目管理，在概念上和软件项目管理没有区别，只不过侧重点不一样，而且其主导思想不同。对于一般的软件项目管理，对成本和进度控制比较严，而从软件测试的角度看，质量第一是基本点，所有项目管理工作都围绕提高产品质量而展开，最终保证在合理的成本、进度控制下，开发出满足用户要求和期望的、可维护的、高质量的软件产品。

软件测试项目管理的内容集中在以下几个方面：

- 软件项目的测试过程管理，包括软件项目的测试计划、测试用例设计、测试执行、测试结果的审查和分析，以及如何开发或使用测试过程管理工具。
- 软件项目的测试工作和产品质量的风险评估和控制。
- 软件项目的测试资源分配和进度控制。
- 软件项目的版本定义、变化控制和配置管理。
- 软件项目的软件构建、打包和发布等管理。

本章将分别介绍、讨论和阐述清楚测试项目管理的各个内容，希望读者通过本章的学习，掌握测试项目管理的思想、特点、方法和技巧。

## 17.1 软件测试项目管理的概述

软件项目管理活动包含测度和度量、估算、风险分析、进度安排、跟踪和控制等，这些内容在下面各节中都会有不同程度的涉及。其中估算和度量是很重要的一个环节，它包括对工作范围、时间、资源、质量、成本等的估算和度量，但同时这些因素本身又相互制约，项目管理者要能够对他们做出权衡。相对精确的估算有助于制订比较可行的计划，在

项目实际运行中不至于因为估算的不准确而不断地修改计划，导致整个过程不断地变动，使得项目进行的过程总是在处理意外情况。

## 17.1.1 软件项目管理的共性

软件测试项目管理的基本内容是计划、组织和监控，或者说有五项基本内容：度量或标准、预估或评估、风险分析、日程安排、跟踪和控制。再细分，软件测试项目管理有八个工作领域：

- 测试范围管理
- 时间管理
- 成本管理
- 质量管理
- 人力资源管理
- 沟通管理
- 风险管理
- 过程管理

而作为一个成功的项目经理需要具备四个方面的能力和素质：

- 问题解决和风险控制能力。
- 协调和沟通能力，具有良好的亲和力。
- 团队组织和激励能力，包括团队的影响力。
- 相应的专业技术能力。

### 1. 软件项目管理的 3P

有效的项目管理集中在 3P：人员（People）、问题（Problem）和过程（Process）。其中，人是决定的因素，对于软件开发，这一点更为明显，因为软件开发是人的智力密集型劳动。3P 对软件项目管理具有本质的影响：

(1) 人员必须被组织成有效率的小组，激发他们进行高质量的测试工作，并为这个团队的人员建立有效的沟通途径和方法，以实现小组之间、人员之间、管理者和被管理者之间有效的沟通；有效的团队应建立合适的组织结构和工作文化，不断促进团队整体表现的方法，通过一系列活动提高团队的凝聚能力、工作态度、积极性，共享团队的目标和文化，并最终在组织、管理和文化上得到和谐的、有机的结合。

(2) 问题在测试项目管理中表现为流程不清楚或控制不严、应用领域知识不足、需求不断变化和不一致性、沟通不流畅等，其解决办法是确定问题在什么地方，然后进行分解，逐个解决。在解决问题时需要良好的沟通、协调技巧。关于测试过程中可能产生的问题，项目管理者必须对此有一个前瞻性的考虑，任何管理者在测试早期对问题没有准备，又不能支持整个团队的有效沟通，就不可能为问题提供正确的解决方案。对于所发现的问题必须同用户和开发人员做细致的交流，尽可能地把任务分解成更小的组成部分或单元，以分配给对应的测试项目小组。

(3) 过程必须适应人员的需求和问题的解决, 人员的需求主要体现在能力、沟通、协调等之上, 问题能在整个软件项目开发的过程中得到跟踪和控制, 也就是说, 一套规范且有效的流程是保证项目运行过程平稳的基础。

对于 3P 的考虑要在计划中充分地反映出来, 计划是用来建立一个总体方向的, 是用来帮助开始进行项目的工作, 保证这些项目是在朝着一个目标前进的。反过来说, 计划要围绕人员、问题和过程来展开, 虽然所有的行动都是围绕项目的目标进行的。

所有软件项目中最关键的因素是人员。软件工程师可以按照不同的小组结构来组织, 从传统的控制层到“开放式范型”的小组。可以采用多种协调方式和通信技术来支持项目组的工作。优秀的测试工程师按照良好的流程进行项目测试, 才能最大限度地保证项目的成功。一个好的流程可以保证差的人做出来的东西不至于太差, 但不能确保做出精品。通过流程可以实现规范化、工业化、专业化的软件测试。流程是基础。

## 2. 软件项目度量在管理上的作用

项目度量的内容在 16.1.1 节中做了简单介绍, 在这里从管理的角度, 强调项目度量对软件项目管理者的作用, 即:

- 评估正在进行的项目的状态, 评估正在开发的软件产品的质量。
- 跟踪潜在的风险, 辅助软件项目的计划、跟踪及控制。
- 在问题造成不良影响之前发现问题。
- 调整工作流程或任务, 能够改善软件过程。
- 评估项目组控制软件工作产品的质量的能力。
- 分析这些度量可产生指导管理及技术行为的指标。

项目组收集到的并被转换成项目度量的测量数据, 也可以传送给负责软件过程改进的人们。因此, 多数度量既用于过程领域又用于项目领域。过程度量使得一个组织能够从战略级洞悉一个软件过程的功效。项目度量是战术的, 使得项目管理者能够以实时的方式改进项目的工作流程及技术方法。

## 3. 软件项目监控的过程步骤

软件项目监控的目的是通过建立对软件项目过程的可视性, 使项目管理者在软件项目性能与软件计划出现偏差时采取有效的纠正措施, 以确保软件过程的质量满足要求。一般软件项目的监控, 以获得真实、实时的项目的一手数据为基础, 按照“获取项目过程信息、分析判断、采取纠偏措施、验证”步骤建立过程的可视性, 通过过程可视实施项目目标管理与过程管理的统一。

在组织实施软件项目的过程中, 对项目的监控可从四个方面着手实施:

(1) 建立符合软件工程和软件项目管理流程要求的、实用的软件项目运行环境, 包括: 明确的过程流程、项目策划、组织支撑环境。

(2) 采用软件项目管理监控平台, 使项目目标管理和过程管理相结合, 提供项目的透明度, 建立过程可视性。

(3) 优秀的项目经理和质量保证经理构成项目的第一责任人, 采用双过程经理制, 项

目经理和软件质量保证经理构成软件项目的灵魂人物。

(4) 项目沟通。项目计划、进度和项目范围必须能够被项目成员方便地得到，以确保大家是在统一的交流平台上朝着共同目标前进。采用适当的图表和模板增强项目组内沟通效果和沟通的一致性；采用良好的邮件系统、日历系统、即时消息系统等构成一个完整的、协同的内部统一消息平台。

#### 4. 软件项目管理的三角关系

在软件项目管理中，最终变为产品、时间和成本这三者之间的平衡，如图 17-1 所示。在一个项目中，一般说某项是确定的，其他两项是可变的，这样，我们就可控制不变项，对可变项采取措施，保证项目达到预期效果。例如，产品质量是不变的，要有足够的时间和成本投入去保证产品。但同时市场决定产品，时间受到严格限制，这时如果要保证产品的功能得到完整的实现，就必须有足够的成本投入（人力资源、硬件资源等）；如果成本也受到限制，就不得不减少功能，实现产品的主要功能。

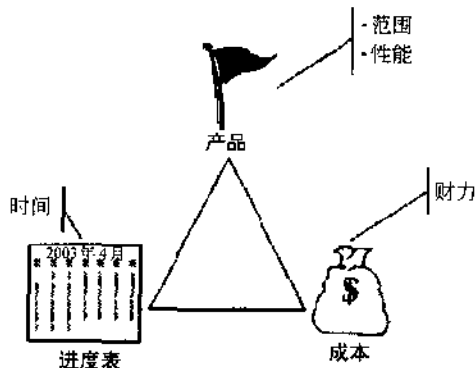


图 17-1 项目管理三角关系——产品、时间和成本

### 17.1.2 软件测试项目管理的特点

软件测试项目管理，一方面继承了一般软件项目管理的共性，另一方面也具有软件测试自身的管理特点。下面就来分析一下哪些是软件测试项目管理的特点。

软件测试项目管理是软件工程的保护性活动。它先于任何测试活动之前而开始，且持续贯穿于整个测试项目的定义、计划和测试之中。

(1) 软件质量标准定义不准确、任务边界模糊，如何确定什么时候软件测试可以结束，找不到严重的缺陷并不代表软件不存在严重的缺陷。软件测试项目的各个里程碑标准和度量的定义、管理要求更高。

(2) 软件测试项目的变化控制和预警分析要求高。随着系统分析、设计和实施的进展，客户的需求不断地被激发，需求不断变化，导致项目进度、系统设计、程序代码和相关文档的变化和修改，而且在修改过程中又可能产生新的问题，而结果受影响最大的是软件测试，因为程序设计和实现被拖延，而常常是最后的时间期限又被控制很严，结果由于测试执行阶段靠后容易造成测试时间被严重挤压。这种情况下，要么和项目经理沟通、谈判，

以争取更多的时间，要么让测试人员加班加点干，总之，保证产品的质量是一个更大的问题。

(3) 软件测试项目具有智力密集、劳动密集的特点，受人力资源影响最大，项目成员的结构、责任心、能力和稳定性对测试执行、产品质量有很大的影响。对于程序设计、编码等出现问题，由测试人员去把关。但如果测试人员的责任心不高，有些严重问题没能被及时发现而漏掉，最终会遗留到客户那里，后果不堪设想。所以软件测试项目的管理更需要细致，风险更大，流程跟踪要求更高。

(4) 测试任务的分配难，比如单元测试和集成测试、系统测试和验收测试等关联比较大，但要求的技术不同，不容易分离。如果将它们分离，边界条件又由谁来负责？

(5) 测试要求人力资源十分稳定。由于软件测试项目不仅是一个技术工作，而且要求对产品的功能、特性需要非常了解。其测试的对象——软件系统会变成一个不可见的逻辑实体，如果人员发生流动，对于没有深入了解产品的功能、特性又缺乏软件测试实践经验的人来说，很难在短时间里做到无缝承接项目的测试工作。

(6) 软件测试人员在待遇、地位可能受到一些不公正的待遇，但同时又要求测试人员需要丰富的工作经验、良好的心理素质和责任心。所以，在软件测试项目管理中，在人才激励和团队管理问题上应给予高度的重视。

由此可见，软件测试项目的管理的好坏对产品质量影响更直接，更富有挑战性，特别强调质量管理、人力资源管理、沟通管理、风险管理等，包括软件系统的配置管理，主要是版本管理。

软件测试项目的过程管理能否成功，通常受到三个核心层面的影响，即项目组内环境、项目所处的组织环境、整个开发流程所控制的全局环境。这三个环境要素直接关系到软件项目的可控性。项目组管理模型与项目过程模型、组织支撑环境和项目管理接口是上述三个环境中各自的核心要素，此外优秀的软件过程管理平台是实现整个项目生命周期项目过程监控的工具保证。

## 17.2 软件测试项目的组织

软件测试是软件质量保证的一个重要环节，也是软件开发最基本的组成部分，作为软件测试项目可以是一个独立的项目，并相应为这个项目建立一个独立的测试小组；也可以与产品开发、产品设计、客户培训和项目管理等组成一个完整的软件开发项目组，这时候测试小组是软件开发项目组的一部分，但也保证相对的独立性。

作为软件测试项目的组织，主要包括软件测试项目的人员结构和组织、制定规范的测试流程、建立客观的评价标准、畅通的交流渠道、完善的奖惩体系等。

### 1. 软件测试项目的人员组织模式

在第12章，我们就详细介绍了软件测试部门或团队的构成和组织模式，相对来说层次更高一些，但大部分思想可以用在这里，所以在此只针对测试小组这个层次来讨论一下人

员组织模式和实践方法。

对于一个测试小组，不需要设置 QA Lab administrator 和 Build Engineer 这样的角色，其小组基本构成如下。

- 测试组长：负责整个测试项目，包括测试计划的定制，和其他小组的沟通和协调。
- 内审员：审查测试流程、结果和报告，包括文档管理。
- 资深测试工程师：负责测试环境的建立、测试用例的设计，也兼做测试的具体工作。
- 测试工程师：负责测试的执行。

一个有效的软件测试项目管理者（测试组长、QA 经理或测试经理）要做到：

- 始终能够把质量放在第一位，测试工作的根本在于保证产品的质量，应该在测试小组中建立起有质量才能生存的观念，要把与测试有关的各项工作和组员的积极性结合起来，建立一套相适应的质量责任制度，形成一个严密的质量体系。
- 能够制订好测试策略、有计划地安排工作、系统的解决方案等。
- 注意合理分配任务，明确规定每一个人在测试工作中的具体任务、职责和权限，每个组员都明确自己该做什么、怎么做、负什么责任、做好的标准是什么。做到人人心中有数，为保证和提高产品质量（或服务质量）提供基本的保证。
- 遇到问题，能准确地判断是技术问题还是流程问题，更重视流程问题的解决。
- 有良好的意识去关心组员，关注项目组员的情绪，以鼓励为主，不断激励员工，鼓舞士气，发挥每一位员工的潜力，注重团队的工作效率。
- 将项目中已有的成功经验能灵活地应用到新的项目中，做好测试项目的风险管理和质量管理。
- 项目管理者具有良好的沟通能力，和其他部门不仅能进行有效沟通，而且可以施加自己的影响（说服别人），以促进项目的整体合作、理解和流程改进。

## 2. 测试项目的管理原则

- 可靠的需求。应当有一个经各方一致同意的、清楚的、完整的、详细的、整体的、可实现的、可测试的需求，并文档化。
- 合理的时间表。为计划、设计、测试、改错、再测试、变更，以及编制文档留出足够的时间，不应使用突击的办法来完成项目。
- 充分测试。尽早开始测试，每次改错或变更后，都应重新测试。项目计划中要为测试和改错留出足够时间。
- 尽可能坚持最初的需求。一旦开发工作开始，要准备防止修改需求和新增功能，要说明这样做的后果。如果必须进行变更，必须在时间表上有相应的反映。如果可能，在设计阶段使用快速的模型，以便使客户了解将会得到的东西。这将会使他们对他们的需求有较高的信心，减少以后的变更。
- 沟通。尽早使用模型，使客户的预想表达清楚；在适当时机进行预排和检查；充分利用团队通信手段，如电话会议、电子邮件、即时消息（MSN, Yahoo message）、变更管理工具等功能。

- 要确保文件是可用的和最新的。优选电子版文档，避免纸介质文档进行远距离联合作业及协作。

### 3. 测试计划先行

软件项目管理过程从项目计划的活动开始，软件测试项目也不例外，也是从测试计划开始。在测试计划活动中，首先要确认测试策略，并对工作范围、时间、资源、质量、成本等的估算。无论何时进行估算，我们都是在预测未来，并会接受某种程度的不确定性。

软件项目计划的目标是提供一个框架，不断收集信息，对不确定性进行分析，将不确定性的内容慢慢转换为确定性的内容，该过程最终使得管理者能够对资源、成本及进度进行合理的估算。这些估算是软件项目开始时在一个限定的时间框架内所做的，并且随着项目的进展不断更新。

计划不仅要反映测试的“真实”工作，而且还应该反映辅助活动，它包括：

- 休假和法定假日。
- 培训和教育。
- 项目管理活动，如规划和人员管理。
- 开销，如系统当机时间、会议和回复电子邮件。
- 体系结构定义。
- 测试之后的系统返工或系统交付。

### 4. 建立优先级

在项目的管理过程中，经常碰到的问题是：等待做的任务比较多，但人力资源和时间受到限制，要完成所有的任务几乎是不可能的。这时候要解决的就是为各项任务建立优先级，这样就可以根据优先级来先后处理各项任务，从繁重的软件开发活动中脱离出来。

有成效的领导知道自己首要的任务是为其他组员提供服务，这些服务包括训练和指导，解决问题和冲突，提供资源，建立项目目标和优先级，提供适当的技术指引。要使每个组员都能清楚知道这些内容，总是需要帮助他们。我们并不都能很幸运地工作在一个良好的环境里，但一定尽到最大的努力将任务单上排在最前面的工作做好，集中精力有效地、快乐地、尽可能地帮助组员。

项目开始之前的估算，只是给了我们一个大致的框架。在实际的操作中，我们还必须灵活调整，不能因为规定束缚了我们的手脚，从而可能耽搁项目的进程，甚至导致项目的失败。我们需要规则，但不能被规则所左右，当规则不适用于具体情况时，要有打破规则的勇气和魄力。

### 5. 建立客观的评价标准

为了建立客观的评价标准，首先必须将所有活动产生的有用的数据记录下来，要记录的内容包括会议纪要、审核记录、缺陷报告、测试报告，并保证所做的记录及时、充分、比较准确、客观。对所有的活动都有一个跟踪落实的过程，比如对所有的审核记录和更改请求都会有一个历史记录，并设置一个状态标识，标识其当前状态，通过跟踪其状态来监



督其落实,这样使整个项目过程具有良好的可测性、可跟踪性,强调以数据说话。当然,要善于利用各种工具,如 Microsoft Project,通过工具就比较容易做好记录,使纪录的数据更直观、数据化,便于评估。

## 17.3 软件测试项目的过程管理

从软件工程的角度讲,软件开发主要分为六个阶段:需求分析阶段、概要设计阶段、详细设计阶段、编码阶段、测试阶段、安装及维护阶段。软件测试项目的过程管理远不是锁定在测试阶段,因为软件测试不能等待代码全部完成后才开始对程序进行测试,而是在项目需求分析阶段就要开始参与进去,审查需求分析文档、产品规格说明书,然后在设计阶段,要审查系统设计文档、程序设计流程图、数据流图等,在代码测试阶段,需要审查代码,看是否遵守代码的变量定义规则、是否有足够的注释行等。如果从软件开发生命周期角度看,软件测试项目的过程管理在各个阶段的具体内容是不同的,但在每个阶段,测试任务的最终完成都要经过从计划、设计、执行到结果分析、总结等一系列相同步骤,这构成软件测试的一个基本过程。

所以软件测试项目的过程管理主要集中在软件测试项目的启动、测试计划、测试用例设计、测试执行、测试结果的审查和分析,以及如何开发或使用测试过程管理工具。但在本节主要是从管理的角度去讨论如何组织、跟踪和控制这些过程,而不是从测试技术的角度去讨论如何设计和实现,后者是本书前面各章讨论的内容。测试过程管理的基本内容如下。

(1) 测试项目启动:首先要确定项目组长,只有把项目组长确定下来,就可以组建整个测试小组,并可以和开发等部门开展工作。接着参加有关项目计划、分析和设计的会议,获得必要的需求分析、系统设计文档,以及相关产品/技术知识的培训和转移(knowledge transfer)。

(2) 测试计划阶段:确定测试范围、测试策略和方法,以及对风险、日程表、资源等进行分析和估计。如何组织和管理计划阶段,见 17.3.1 节。

(3) 测试设计阶段:制订测试的技术方案、设计测试用例、选择测试工具、写测试脚本等。测试用例设计要实现做好各项准备,才开始进行,最后还要让其他部门审查测试用例,详见 17.3.2 节。

(4) 测试执行阶段:建立或设置相关的测试环境,准备测试数据,执行测试用例,发现的软件缺陷进行报告、分析、跟踪等,测试执行没有很高的技术性,但是测试的基础,直接关系到测试的可靠性、客观性和准确性,详见 17.3.3 节。

(5) 测试结果的审查和分析:当测试执行结束后,对测试结果要进行整体的或综合分析,以确定软件产品质量的当前状态,为产品的改进或发布提供数据和依据。从管理来讲,要做好测试结果的审查和分析会议,以及做好测试报告或质量报告写作、审查,详见第 16 章。

## 17.3.1 测试计划阶段

测试计划的内容和模板、测试策略在第3章侧重理论上做了详细介绍。这里从测试项目实施和管理的角度,进一步讨论软件测试项目计划的实施目标、标准、计划阶段的细分、测试项目计划的要点和编制测试计划的一些技巧等。

测试项目计划的整体目标是为了确定测试的任务、所需的各种资源和投入、预见可能出现的问题和风险,以指导测试的执行,最终实现测试的目标,保证软件产品的质量。制订测试计划,要达到的目标有:

- (1) 为测试各项活动制定一个现实可行的、综合的计划,包括每项测试活动的对象、范围、方法、进度和预期结果。
- (2) 为项目实施建立一个组织模型,并定义测试项目中每个角色的责任和工作内容。
- (3) 开发有效的测试模型,能正确地验证正在开发的软件系统。
- (4) 确定测试所需要的时间和资源,以保证其可获得性、有效性。
- (5) 确立每个测试阶段测试完成以及测试成功的标准、要实现的目标。
- (6) 识别出测试活动中各种风险,并消除可能存在的风险,降低那些不可能消除的风险所带来的损失。

### 1. 软件测试项目的标准

为了保证测试的执行能按计划执行,必须确认测试在满足什么外部条件下才能开始,这就是要在测试计划中定义软件测试项目的输入标准,然后定义测试项目的输出标准。

#### (1) 测试的输入标准

- 整体项目计划框架:需要在框架清楚情况下,来定制测试计划。
- 需求规格说明书:只有了解到用户具体的、实际的需求,才能制订测试需求和测试范围。
- 技术知识或业务知识:技术的变化或新技术的引入,需要事先准备,包括人员的培训。
- 标准环境:符合用户使用环境或业务运行环境的需求。
- 设计文档:是设计软件测试用例的重要参考资料,帮助测试人员了解系统的薄弱环节、关键点等。
- 足够的资源,包括人力资源、硬件资源、软件资源和其他环境资源。
- 人员组织结构,项目经理、测试组长、成员等责任及其之间的关系已确定。

#### (2) 测试的输出标准

- 测试执行标准
- bug 描述和处理标准
- 文档标准和模板
- 测试分析、质量评估标准等

## 2. 测试实施策略的制定

测试策略描述当前测试项目的目标和所采用的测试方法。这个目标不是上述测试计划的目标,而是针对某个应用软件系统或程序。具体的测试项目要达到的预期结果,包括在规定的时间内哪些测试内容要完成,软件产品的特性或质量在哪些方面得到确认。

测试策略还要描述测试不同阶段(单元测试、集成测试、系统测试)的测试对象、范围和方法以及每个阶段内所要进行的测试类型(功能测试、性能测试、压力测试等)。在制订测试策略前,要确定测试策略项,测试策略包括:

- (1) 要使用的测试技术和工具,如 60%用 Rational Robot 自动测试,40%手工测试。
- (2) 测试完成标准,用以计划和实施测试,及通报测试结果。如 95%测试用例通过并且 P1、P2 级别的缺陷全部解决。
- (3) 影响资源分配的特殊考虑,例如有些测试必须在周末进行,有些测试必须通过远程环境执行,有些测试需考虑与外部接口、或硬件接口。

在确认测试方法时,要根据实际情况,结合测试方法的特点来选择合适的方法:

- 根据是否需要执行被测软件来划分,有静态测试和动态测试。静态测试,如规格说明书、程序代码的审查,在工作中容易被忽视,在测试策略上应说明如何加强这些环节。
- 根据是否针对系统的内部结构和具体实现算法来划分,有白盒测试和黑盒测试。如何将白盒测试和黑盒测试有机地结合起来测试,也是测试策略要处理的问题之一。尽管用户更倾向于基于程序规格说明的功能测试,但是白盒测试能发现潜在的逻辑错误,而这种错误往往是功能测试发现不了的。

综合起来,可能要在“基于测试技术的测试策略”和“基于测试方案的综合测试策略”之间做出一个选择。

## 3. 测试项目计划阶段的细分

测试项目的计划不可能一气呵成,而是要经过计划初期、起草、讨论、审查等不同阶段,才能将测试计划制订好。而且,不同的测试阶段(集成测试、系统测试、验收测试等)或不同的测试任务(安全性测试、性能测试、可靠性测试等)都可能要有具体的测试计划。

(1) 计划初期是收集整体项目计划、需求分析、功能设计、系统原型、用户用例(use case)等文档或信息,理解用户的真正需求,了解技术难点和弱点、或新的技术,和其他项目相关人员交流,在各个主要方面达到一致的理解。

(2) 测试计划最关键的一步就是确定测试需求、测试层次。将软件分解成单元,对各个单元写成测试需求,测试需求也是测试设计和开发测试用例的基础,并用来衡量测试覆盖率的重要指标。

(3) 计划起草。根据计划初期所掌握的各种信息、知识,确定测试策略,设计测试方法,完成测试计划的框架。

(4) 内部审查。在提供给其他部门讨论之前,先在测试小组/部门内部进行审查。

(5) 计划讨论和修改。召开有需求分析、设计、开发人员参加的计划讨论会议,测试

组长将测试计划设计的思想、策略做较详细的介绍，并听取大家对测试计划中各个部分的意见，进行讨论交流。

(6) 测试计划的多方审查。项目中的每个人都应当参与审查（即市场、开发、支持、技术写作及测试人）。计划的审查是必不可少的，尽管测试工程师努力地完成一个对产品的全面定义，但出自于一个测试工程师的定义不一定是完整或准确的。此外，就像开发者很难测试自己的代码那样，测试工程师也很难评估自己的测试计划。每一个计划审查者都可能根据其经验及专长提出修改建议，有时还能提供测试工程师在组织产品定义时不具备的信息。

(7) 测试计划的定稿和批准。在计划讨论、审查的基础上，综合各方面的意见，就可以完成测试计划书，然后报给测试经理或 QA 经理，得到批准，方可执行。

测试计划不仅是软件产品当前版本而且还是下一个版本的测试设计的主要信息来源，在进行新版本测试时，可以在原有的软件测试计划书上做修改，但要经过严格审查。

#### 4. 测试项目计划的要点

在第3章提到软件测试内容，主要包括：产品基本情况、测试需求说明、测试策略和记录、测试资源配置、计划表、问题跟踪报告、测试计划的评审、结果等。除了产品基本情况、测试需求说明、测试策略等，测试计划的焦点集中在：

(1) 计划的目的：项目的范围和目标，各阶段的测试范围、技术约束和管理特点。

(2) 项目估算：使用的历史数据，使用的评估技术，工作量、成本、时间估算依据。

(3) 风险计划：测试可能存在的风险分析、识别，以及风险的回避、监控、管理。

(4) 日程：项目工作分解结构，并采用时限图、甘特图等方法制定时间/资源表。

(5) 项目资源：人员、硬件和软件等资源的组织和分配，人力资源是重点，而且日程安排紧密联系。

(6) 跟踪和控制机制：质量保证和控制，变化管理和控制等。

测试计划书的内容也可以按集成测试、系统测试、验收测试等阶段去组织，为每一个阶段制订一个计划书，也可以为每个测试任务/目的（安全性测试、性能测试、可靠性测试等）制订特别的计划书，但其内容也不外乎是第3章所述的八项内容求以上所述的六个焦点。

同时，可以对上述测试计划书的每项内容，制订一个具体实施的计划，如每个阶段的测试重点、范围、所采用的方法、测试用例设计的思想、提交的内容等进行细化，供测试项目组的内部成员使用。对于一些重要的项目，会形成一系列的计划书，如测试范围/风险分析报告、测试标准工作计划、资源和培训计划、风险管理计划、测试实施计划、质量保证计划等。

#### 5. 编制测试项目计划的技巧

在计划书中，有些内容是介绍测试项目的背景、所采用的技术方法等，这些内容仅仅作为参考，但有些内容（如人员组成、日程安排）则可以看做是一种结论或是承诺，必须要实施或达到的目标，如测试小组结构和组成、测试项目的里程碑、面向解决方案的交付

内容、项目标准、质量标准、相关分析报告等。

要做好测试计划，测试设计人员要仔细阅读有关资料，包括用户需求规格说明书、设计文档、使用说明书等，全面熟悉系统，并对软件测试方法和项目管理技术有着深刻的理解，并建议注意以下方面：

- (1) 确定测试项目的任务，清楚测试范围和测试目标，如提交什么样的测试结果。
- (2) 让所有合适的相关人员参与测试项目的计划制订，特别是在测试计划早期。
- (3) 对测试的各阶段所需要的时间、人力及其他资源进行预估，尽量做到客观、准确、留有余地。
- (4) 制订测试项目的输入、输出和质量标准，并和有关方面达成一致。
- (5) 建立变化处理的流程规则，识别出在整个测试阶段中哪些是内在的、不可避免的变化因素，如何进行控制。

测试计划强调测试计划和执行的原则，在测试计划中描述进行测试所需的测试设计和步骤是另一层关于测试执行的原则。在测试设计和计划中的错误与欠缺，在设计转换成测试计划中特定的结构和测试步骤之后就难以补救。

### 17.3.2 软件测试设计和开发

当测试计划完成之后，测试过程就要进入软件测试设计和开发阶段。软件测试设计是建立在测试计划书的基础上，认真理解测试计划的测试大纲、测试内容及测试的通过准则，通过测试用例来完成测试内容的、典型的逻辑转换，作为测试的实施依据，以实现所确定的测试目标。软件设计是将软件需求转换成软件表示的过程，主要描绘出系统结构、详细的处理过程和数据库模式；软件测试设计则是将测试需求转换成测试用例，描述测试环境、测试执行的范围、层次和用户的使用场景。所以软件测试设计和开发是软件测试过程中一个技术深、要求高的一个关键阶段。

软件测试设计和开发主要内容有：

- (1) 制定测试的技术方案，确认各个测试阶段要采用的测试技术、测试环境和平台，以及选择什么样的测试工具。系统测试中的安全性、可靠性、稳定性、有效性等的测试技术方案是这部分工作内容的重点。
- (2) 设计测试用例，根据产品需求分析、系统技术设计等规格说明书，在测试的技术方案基础上，设计具体的测试用例。在第 14 章有详细介绍。
- (3) 设计测试用例特定的集合，满足一些特定的测试目的和任务，即根据测试目标、测试用例的特性和属性（优先级、层次、模块等），来选择不同的测试用例，构成执行某个特定测试任务的测试用例集合（组），如基本测试用例组、例外测试用例组、性能测试用例组、完全测试用例组等。
- (4) 测试开发：根据所选择的测试工具，将所有可以进行自动化测试的测试用例转换为测试脚本的过程。
- (5) 测试环境的设计，根据所选择的测试平台以及测试用例所要求的特定环境，进行服务器、网络等测试环境的设计。可以参考第 13 章相关内容。

软件测试设计中,要考虑的要点有:

- 所设计的测试技术方案是否可行、是否有效、是否能达到预期的测试目标。
- 所设计的测试用例是否完整、边界条件是否考虑、其覆盖率能达到多高。
- 所设计的测试环境是否和用户的实际使用环境比较接近。

其关键是做好测试设计前的知识传递,将设计/开发人员已经所掌握的技术、产品、设计等知识传递给测试人员;同时,要做好测试用例的审查工作,不仅要通过测试人员的审查,还要通过设计/开发人员的审查。

### 1. 测试用例设计的方法和管理

软件测试用例设计的方法,如第14章所述,有白盒测试和黑盒测试相对应的设计方法,黑盒测试的用例设计,采用等价类划分、因果图法、边值分析、用户界面测试、配置测试、安装选项验证等方法,适用于功能测试和验收测试。而白盒测试的用例设计,有以下方法:

- 采用逻辑覆盖(包括程序代码的语句覆盖、条件覆盖、分支覆盖)的结构测试用例的设计方法。
- 基于程序结构的域测试用例设计方法。“域”是指程序的输入空间,域测试正是在分析输入空间的基础上,完成域的分类、定义和验证,从而对各种不同的域选择适当的测试点(用例)进行测试。
- 数据流测试用例设计的方法,是通过程序的控制流,从建立的数据目标状态的序列中发现异常的结构测试方法。
- 根据对象状态或等待状态变化来设计测试用例,也是比较常见的方法。
- 基于程序错误的变异来设计测试用例,可以有效地发现程序中某些特定的错误。
- 基于代数运算符号的测试用例设计方法,受分支问题、二义性问题和大程序问题的困扰,使用较少。

测试用例要经过创建、修改和不断改善的过程,一个测试用例具有以下属性:

- 测试用例的优先次序(priority),优先级越高,被执行的时间越早、执行的频率越多。由最高优先级的测试用例组会构成基本验证测试(BVT, basic verification test),每次构建软件包时,都要被执行一遍。
- 测试用例的目标性,有的测试用例(major case)是为主要功能而设计,有的测试用例(minor case)是为次要功能而设计,有的(stress case)则为系统的负载而设计,有的(exception case)则为一些特殊场合而设计。
- 测试用例所属的范围,属于哪一个组件或模块,这种属性被用来管理测试用例。
- 测试用例的关联性,测试用例一般和软件产品特性相联系的,多数情况下验证某个产品的功能。这种属性可以被用于验证被修改的软件缺陷,或对软件产品紧急补丁包的测试。
- 测试用例的阶段性的,属于单元测试、集成测试、系统测试、验收测试中的某一个阶段。这样对每个阶段,构造一个测试用例的集合被执行,并容易计算出该阶段的测试覆盖率。

- 测试用例的状态性, 当前是否有效, 如果无效, 被置于 Inactive 状态, 不会被运行, 只有被激活的 (active) 测试用例才被运行。
- 测试用例的时效性, 针对同样功能, 可能所用的测试用例不同, 是因为不同的产品版本在产品功能、特性等方面的要求不同。
- 所有者、日期等特性, 测试用例还包括由谁、在什么时间创建的, 又由谁、在什么时间修改的。

根据上述特性, 再结合测试用例的编号 (ID)、标题、描述 (条件、步骤、期望结果) 等, 就可以对测试用例进行基于数据库方式的良好管理。

测试用例设计完之后, 要经过非正式和正式的审查:

(1) 非正式的审查: 一般在 QA 或测试小组 (部门) 内部进行, 包括同测试组人员互相检查 (peer review), 或让资深人员、测试组长帮助审查。

(2) 正式的审查: 一般通过正式 E-mail 将已设计好的测试用例发给相应的系统分析、设计人员和程序员, 让他们先通读看一遍, 将发现的问题记下来。然后由测试组长或项目经理召开一个测试用例审查会, 由测试设计人员先对测试用例的设计思想、方法、思路等进行说明, 然后系统分析、设计人员和程序员把问题提出来, 测试人员回答, 必要时做些讨论。

审查完的测试用例, 经修改后, 就可以直接用于手工测试或用于测试脚本的开发。

## 2. 测试开发

根据所选择的测试工具脚步语言, 如 Rational SQA Basic, 编写测试脚本, 将所有可以进行自动化测试的测试用例转化为测试脚本。其输入就是基于测试需求的测试用例, 输出是测试脚本和与之相对应的期望结果, 这种期望结果一般存储在数据库中或特定的格式化文件中。

(1) 测试开发的步骤, 首先要设立测试脚本开发环境, 安装测试工具软件, 设置管理服务器和具有代理的客户端池, 建立项目的共享路径、目录, 并能连接到脚本存储库和被测软件等。然后执行录制测试初始化过程、独立模块过程、导航过程和其他操作过程, 结合已经建立的测试用例, 将录制的测试脚本进行组织、调试和修改, 构造成一个有效的测试脚本体系, 并建立外部数据集合。

(2) 由于被测系统处在不完善阶段, 在运行测试脚本的过程中, 容易中断, 所以在测试脚本开发时, 要处理好这种错误, 及时记录当时的状态, 又能继续执行下去。处理这个问题, 有一些解决办法, 如跳转到别的测试过程、调用一个能够清除错误的过程等。

(3) 测试开发常见的问题。测试开发很乱, 与测试需求或测试策略没有对应性; 测试过程不可重用; 测试过程被作为一个编程任务来执行, 导致脚本可移植性差。这些问题应该避免, 在脚本的结构、模块化、参数传递、基础函数 (库) 等方面设计好。

## 17.3.3 测试执行阶段

当测试用例的设计和测试脚本的开发完成之后, 就开始执行测试。

- 手工测试: 在合适的测试环境上, 按照测试用例的条件、步骤要求, 准备测试数

据,对系统进行操作,比较实际结果和测试用例的所描述的期望结果,以确定系统是否正常运行或正常表现。

- 自动化测试:通过测试工具,运行测试脚本,得到测试结果。

自动化测试的管理相对比较容易,测试工具不会做小动作,会不打折扣地执行测试脚本,并能自动记录下测试结果。而对手工测试的管理相对要复杂得多,在整个测试执行阶段中,管理上会碰到一系列问题,主要有:

- 如何确保测试环境满足测试用例所描述的要求?
- 如何保证每个测试人员清楚自己的测试任务?
- 如何保证每个测试用例得到百分之百的执行?
- 如何保证所报告的 bug 正确、描述清楚、没有漏掉信息?
- 如何在验证 bug 和对新功能的测试上寻找平衡?
- 如何跟踪 bug 处理的进度,严重的 bug 及时得到解决?
- 如何确保正确的测试环境?需要专人(QA 试验室管理人员、测试组长等)进行检查,详见第 13 章。bug 的报告、跟踪和分析,详见第 15 章。

#### 1. 测试阶段目标的检查

要对每个测试阶段(代码审查、单元测试、集成测试、功能测试、系统测试和验收测试、安装测试等)的结果进行分析,保证每个阶段的测试任务得到执行,达到阶段性的目标。

(1) 代码审查:不是指编程人员互查,而是指测试人员参与的代码会审。一方面,要促使会审小组成员充分阅读待审的程序设计流程图、程序代码等;另一方面要求程序员讲解程序的逻辑,并对关键程序段一起通读,从而比较容易发现程序代码中的错误,而进一步讨论有可能暴露程序的结构问题。

(2) 单元测试:目的在于发现各模块内部可能存在的各种差错,一般由程序员自己做,但必须提交单元测试用例和测试报告,测试人员需要审查单元测试用例和测试报告。

(3) 集成测试:主要目标是发现与接口有关的问题,不管是外部接口还是内部参数的传递,要抓住关键模块,关键模块应尽早测试,并将自顶向下、自底向上两种测试策略结合起来,对各个模块严格执行。由于设计系统不同的模块、不同的层次或不同的部门,容易造成一些漏洞、疏忽,要根据设计文档多提问题、集体审查。

(4) 功能测试:目的是向未来的用户表明系统能够按预定要求的功能那样工作,这时的测试是直接操作完整的软件系统,需要站在用户的角度上,尽量模拟用户使用的各种情景,甚至让用户参与测试。

(5) 系统测试:目标是保证系统在实际的环境中能够稳定、可靠地运行下去,包括恢复性测试、安全性测试、强度测试和性能测试等。系统测试技术要求高,占用资源比较多,所以应充分设计好、准备充分。

(6) 验收测试:验收测试既可以是非正式的测试,也可以是正式的、有计划的测试。一个软件产品可能拥有众多用户,不可能由每个用户验收,此时多采用称为 $\alpha$ 、 $\beta$ 测试的过程。 $\alpha$ 测试是指软件开发公司组织内部人员模拟各类用户对即将发布的软件产品(称为 $\alpha$



版本)进行测试,试图发现错误并修正。 $\alpha$ 测试的关键在于尽可能逼真地模拟实际运行环境和可能的用户操作方式。经过 $\alpha$ 测试调整的软件产品称为 $\beta$ 版本。紧随其后的 $\beta$ 测试是指组织公司外部的典型用户试用 $\beta$ 版本,并要求用户报告异常情况、提出批评意见,然后再对 $\beta$ 版本进行改错和完善。

## 2. 测试用例执行的跟踪

测试用例执行直接关系到测试的效率、结果,不仅要做到测试效率高,而且要保证结果正确、准确、完整等,其管理关键是提高测试人员素质和责任心,树立良好的质量文化意识,其次要通过一定的跟踪手段从某些方面保证测试执行的质量。

- 测试效率的跟踪比较容易,按照测试任务和测试周期,可以得到期望的曲线,然后每天检查测试结果,了解是否按预期进度进行,见图 17-2 测试执行情况的跟踪曲线。
- 测试结果的跟踪相对存在一些风险,但如果记录好每个人的执行测试情况,即知道哪个测试用例是谁执行的。一旦某个 bug 被漏掉,可以追溯到具体责任人。

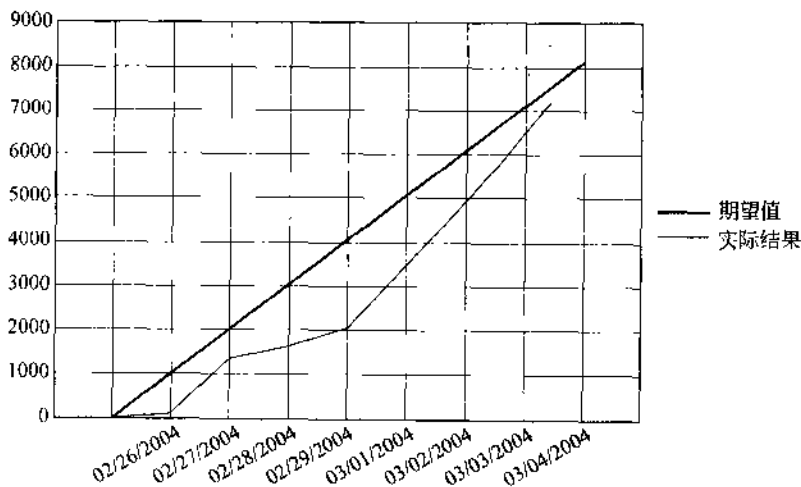


图 17-2 测试执行情况的跟踪曲线

## 3. bug 的跟踪和管理

bug 的跟踪和管理一般由数据库系统来执行,但数据库系统也是依赖于一定的规则和流程进行的,主要的思路有:

- (1) 设计好每个 bug 应该包含的信息条目、状态分类等。
- (2) 通过系统自动发出邮件给相应的开发人员和测试人员,使得任何 bug 都不会错过,并能得到及时处理。
- (3) 通过日报、周报等各类项目报告来跟踪目前 bug 状态。
- (4) 在各个大小里程碑之前,召开有关人员的会议,对 bug 进行会审。
- (5) 通过一些历史曲线、统计曲线等进行分析,预测未来情况。

#### 4. 和项目组外部人员的沟通

为了使测试进展顺利，必要的和项目组外部人员的良好沟通是必要的，这样使测试碰到的问题比较容易解决，测试中发现的 bug 的处理效率也会提高。有一些方法值得推荐：

- 通过一种合适的、可接受的方式指出对方的问题，尽量做到针对事，而不针对人。
- 每周要有一次不同部门参加的会议。
- 建立大项目的邮件组，包含各部门主要人员的邮件地址。
- 在同一个大项目组的开发、测试人员的日报、周报等要互相抄送。
- 适当搞些类似于 Party 的活动，改善关系，增加了解。

人员间的基本通信方式，主要有以下五种类型：

- 正式非个人方式，如正式审查会议。
- 正式个人之间交流，如成员之间的正式讨论，有电子邮件跟踪或文字记载，并伴随所做出的结论，以后可以跟踪、审查。
- 非正式个人之间交流，如个人之间通过电话、即时消息系统的自由交流。
- 内部公共论坛，大家就某个主题发表自己看法，或提问题，或回答问题。
- 成员网络，如成员与小组之外或公司之外有经验的相关人员进行交流。

#### 5. 测试执行结束

测试执行全部完成，并不意味着测试项目的结束，测试项目结束的阶段性标志是将测试报告或质量报告发出去后，并得到测试经理或项目经理的认可。除了测试报告或质量报告的写作之外，还要对测试计划、设计和执行等进行检查、分析，完成项目的总结。

关于测试报告或质量报告的写作，在第 16 章已经做了详细介绍。这里主要简单介绍一下对测试执行结束前后进行管理：

(1) 审查测试全过程：在原来跟踪的基础上，要对测试项目进行全过程、全方位的审视一遍，检查测试计划、测试用例是否得到执行，检查测试是否有漏洞。

(2) 对当前状态的审查：包括产品 bug 和过程中没解决的各类问题。对产品目前存在的缺陷进行逐个的分析，了解对产品质量影响的程度，从而决定产品的测试能否告一段落。

(3) 结束标志：根据上述两项的审查进行评估，如果所有测试内容完成、测试的覆盖率达到要求以及产品质量达到已定义的标准，就可以定稿测试报告，并发送出去。

(4) 项目总结：通过对项目中的问题分析，找出流程、技术或管理中所存在的根源，避免今后发生，并获得项目成功经验。

## 17.4 软件测试项目的资源管理

软件测试项目的资源管理是一项最基本的内容，项目的完成依赖于必要的、充分的资源，“巧妇难作无米之炊”，没有资源就无法去做事情；有了资源，如果不够充分，项目能进行下去，但不能及时完成任务。资源的管理目的不仅要保证测试项目要有足够的资源，

同时,能充分、有效地利用现有资源,进行资源的优化组合,避免任何的资源浪费。

测试项目的资源,主要分为人力资源、系统资源(硬件和软件资源)以及环境资源,或简单分为两类,即人力资源、环境资源。系统资源可以并入环境资源。每一类资源都由四个特征来说明:资源描述、可用性说明、需要该资源的时间,以及该资源被使用的持续时间。后两个特征可以看成是时间窗口,对于一个特定的窗口而言,资源的可用性必须在开发的最初期就建立起来。

资源的估算技术主要分为两大类:分解和经验建模。

- 分解技术需要划分出主要的软件功能,接着估算测试每一个功能所需的程序规模或人·月数。
- 经验技术使用根据经验导出的公式来预测工作量和时间。可以使用自动工具实现某一特定的经验模型。

通过比较和调和使用不同技术导出的估算值,计划者更有可能得到精确的估算。软件项目估算永远不会是一门精确的科学,但将良好的历史数据与系统化的技术结合起来能够提高估算的精确度。

### 1. 人力资源管理

一个软件测试项目所需的人员数目在完成了测试工作量的估算之后就能够基本确定,一般在测试计划中做了描述。但是软件测试项目所需的人员和要求在各个阶段是不同的:

(1) 在初期需要项目经理或测试组长介入进去,为测试项目提供总体方向、制订测试策略、测试计划,申请系统资源。

(2) 在测试前期,需要一些比较资深的测试设计、开发人员,对被测软件的详细了解、测试评估、测试需求的分解,设计测试用例、开发测试脚本。

(3) 在测试中期,主要是测试执行,要看测试自动化实现的程度,如果测试自动化程度高,人力的投入没有明显增加;如果测试自动化程度低,测试执行的人员要求多,需要比较早的计划,保证足够的资源。

(4) 在测试后期,资深的测试人员可以抽出部分时间去做新项目的准备工作。从经验看,人力资源的管理难度主要有三个方面:

- 资源需求的估计,依赖于工作量的估计和每个工程师的能力评估。
- 资源的应急处理,要有10%的资源余量(buffer)作为人力储备。
- 资源的阶段间或项目间的平衡艺术。

### 2. 测试环境资源

把建立所有必要测试环境所需要的计算机软件资源和硬件资源合称为测试环境资源。硬件提供了一个支持操作系统、应用系统和测试工具等运行的基本平台。软件资源包括操作系统、第三方软件产品、测试工具软件等。

- 硬件:测试存储库、网络/子网、客户测试机、测试开发的PC机。
- 软件:Rational Robot、Microsoft Office、数据库系统、配置需求列表等。
- 网络环境建立和协议配置。

见第13章。

### 3. 工作量的估计

工作量的估计是比较复杂的，针对不同的应用领域、程序设计技术、编程语言等，其估算方法是不同的。其估算可能要基于一些假定或定义。

- 效率假设：即测试队伍的工作效率。对于功能测试，这主要依赖于应用的复杂度、窗口的个数、每个窗口中的动作数目。对容量测试，主要依赖于建立测试所需数据的工作量大小。
- 测试假设：为了验证一个测试需求所需测试动作数目，包括每个测试用例的估计所用的时间。
- 所处测试周期的阶段：有些阶段主要工作都在设计，有些阶段主要是测试执行。
- 应用的复杂度指标和需求变化的影响程度，决定了测试需求的维数。测试需求的维数越多，工作量就越大。

工作量的估算也主要是通过分解技术、经验来实现。

## 17.5 测试项目的进度管理

项目的进度管理是一门艺术，是一个动态的过程，需要不断调度、协调，保证项目的均衡发展，实现项目整体的动态平衡。在项目进度的管理过程中，项目的实施与项目的计划是互动的。

项目开始前的计划，对任务的测试需求有一个大体的认识，但深度不够，进度表可能只是一个时间上的框架，其中一定程度上是靠计划制定者的经验来把握的。随着时间的推移、测试的不断深入，对任务会有进一步的认识，对很多问题都不再停留在比较粗的估算上，项目进度表会变得越来越详细、越准确。

项目的进度管理主要通过里程碑、关键路径的控制并借助工具来实现，同时要把握好进度与质量、成本的关系，以及充分了解进度的数量和质量的三重特性。

### 17.5.1 测试项目的里程碑和关键路径

在软件测试项目的计划书中，都会制订一个明确的日程进度表。如何对项目进行阶段划分、如何控制进度、如何控制风险等等，有一系列方法，但最成熟的技术是里程碑管理和关键路径的控制。

#### 1. 里程碑（Milestone）的定义和控制

里程碑一般是项目中完成阶段性工作的标志，即将一个过程性的任务用一个结论性的标志来描述任务结束的、明确的起止点。一系列的起止点就构成引导整个项目进展的里程碑（milestone）。一个里程碑标志着上一个阶段结束、下一个阶段开始，也就是定义当前

阶段完成的标准 (entry criteria) 和下一个新阶段启动的条件或前提 (entry criteria)。里程碑还有下列特征:

- 里程碑也是有层次的, 在一个父里程碑下一个层次中定义子里程碑。
- 不同类型的项目, 里程碑可能不同。
- 不同规模项目的里程碑, 其数量多少不一样, 里程碑可以合并或分解。

在软件测试周期中, 建议定义六个父里程碑、十几个子里程碑。

M1: 需求分析和设计的审查

M11: 市场/产品需求审查

M12: 产品规格说明书的审查

M13: 产品和技术知识传递

M14: 系统/程序设计的审查

M2: 测试计划和设计

M21: 测试计划的制定

M22: 测试计划的审查

M23: 测试用例的设计

M24: 测试用例的审查

M25: 测试工具的设计和选择

M26: 测试脚本的开发

M3: 代码 (包括单元测试) 完成

M4: 测试执行

M41: 集成测试完成

M42: 功能测试完成

M43: 系统测试完成

M44: 验收测试完成

M45: 安装测试完成

M5: 代码冻结

M6: 测试结束

M61: 为产品发布进行最后一轮测试

M62: 写测试和质量报告

对每个子里程碑, 还可以定义更小的里程碑——孙里程碑, 见表 17-1。

在一个里程碑到来之前, 要进行检查, 了解状态以确定是否能在预期的时间达到里程碑 exit criteria (阶段完成的标准), 如果存在较大差距, 就要采取措施, 争取达到里程碑的标准, 即使不能, 也要尽量减少这种差距。而每到一个里程碑, 必须严格检查实际完成的情况是否符合已定义的标准, 应及时对前一阶段的测试工作进行小结; 如果需要, 可以对后续测试工作计划进行调整, 如增加资源、延长下一个里程碑的时间, 以实现下一个里程碑的目标。

在项目管理进度跟踪的过程中, 给予里程碑事件足够的重视, 往往可以起到事半功倍的效用, 只要能保证里程碑事件的按时完成, 整个项目的进度也就有了保障。根据里程碑

就比较容易确定软件测试进度表。如表 17-1 所示是一个测试项目进度表的例子。

表 17-1 软件测试进度表示例

任 务	天	任 务	天	任 务	天	任 务	天
M21: 测试计划制定	11	M23: 测试设计	12	开发测试过程	5	验证测试结果	2
确定项目	1	测试用例的设计	7	测试和调试测试过程	2	调查突发结果	1
定义测试策略	2	测试用例的审查	2	修改测试过程	2	生成缺陷日记	1
分析测试需求	3	测试工具的选择	1	建立外部数据集	1	M62: 测试评估	3
估算测试工作量	1	测试环境的设计	2	重新测试, 并调试测试过程	2	评估测试需求的覆盖率	1
确定测试资源	1	M26: 测试开发	15	M42: 功能测试	9	评估缺陷	0.5
建立测试结构组织	1	建立测试开发环境	1	设置测试系统	1	决定是否达到测试完成的标准	0.5
生成测试计划文档	2	录制和回放原型过程	2	执行测试	4	测试报告	1

## 2. 项目的关键路径

每个项目可以事先根据各项任务的工作量估计、资源条件限制和日程安排, 确定一条关键路径。关键路径是一系列能够确定计算出项目完成日期的、任务构成的日程安排线索, 也就是说, 当关键路径上的最后一个任务完成时, 整个项目也就随之完成了; 或者说, 关键路径上的任何一项任务延迟, 整个项目就会延期。为了确保项目如期完成, 应该密切关注关键路径上的任务和为其分配的资源, 这些要素将决定项目能否准时完成。

关键路径法 (CPM) 是国际上公认的项目进度管理办法, 其计算方法简单, 许多项目管理工具, 如 Microsoft Project, 可以自动计算关键路径。随着项目的实施、进行的过程中, 关键路径可能由于某些当前关键路径上的任务的变化 (延迟) 时而变化, 产生新的关键路径, 所以关键路径也是动态的, 但这种动态性要控制在最小范围内。关键路径的这种变化可能导致原来不在关键路径上的任务成为关键路径的必经之路, 因此, 作为测试组长或项目经理需要随时关注项目进展, 跟踪项目的最新计划, 确保及时完成关键路径上的任务。

## 17.5.2 测试项目进度的特性及外在关系

任何一项工作, 最开始总是很容易看到进度, 就比如盖房子, 从无到有, 变化是很明显的。可是越到后来, 它的进度越来越不明显。软件测试也是如此, 开始测试之初, bug 比较容易发现, 但测试的进展并不是按 bug 的数量来计算的, 越到后面, bug 越来越难发现。要提高测试进度的质量, 将严重的、关键的问题在第一时间发现出来, 这样才不至于在最后阶段使得开发人员要对代码做大规模的变动, 无法保证测试的时间, 从而影响软件

的质量。这就是测试项目进度的数量和质量的双重特性，我们在关注进度的同时要把把握好这两个特性，在注重进度速度的同时，还要看进度前期的质量可以参考图 17-3 进度的特性图。

假设有一个 10 天的测试项目，在这个项目中测试工程师实际发现了 60 个 bug，其 bug 数量的日期分布应该呈渐渐回落之势，即一开始发现的 bug 最多，随着时间的推移，发现的 bug 应该越来越少，到最后 2~3 天应该不再报告新的 bug，而只是验证已被改正的 bug。理想的话，问题的难度也应该是呈降序排列的，尽早发现最不易解决的问题，因为在解决这些问题的时候，往往会产生新的 bug，及早报告，有利于我们更好的解决问题。当然，在做具体项目的时候，有一定的难度，不容易保证完全做到这样，这与测试工程师本身的技能和经验也是很有关系的。

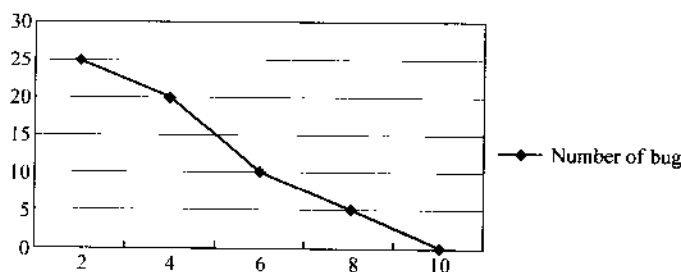


图 17-3 测试项目进度的特性图

### 1. 进度与质量关系

测试项目管理的基本原则是保证在预算内、满足质量的前提下，按进度完成项目。因此，可以看到，进度与质量存在一定程度上的矛盾关系，有时要保证质量，进度就必须放慢，有足够的时间进行测试；有时要保证进度，质量就受到一定影响或存在比较大的风险。如何处理质量与进度的关系？作者认为，保证质量是前提，然后考虑资源的调度和进度的调整。

首先，尽量利用历史数据，从以前完成过的项目来进行类比分析，以确定质量和进度所存在的某种数量关系，来控制进度和管理质量。

其次，可以采用对进度管理计划添加质量参数的方法，也就是通过参数调整进度和质量的关系。这一做法的前提是要有一定的历史数据。比如，从历史数据中得知，完成子项目的时间是 5 天，测试后有 15 个问题；完成同样子项目的时间是 7 天，测试后有 10 个问题；完成同样子项目的时间是 8 天，测试后有 5 个问题，……依此类推。

随着数据的不断增多的，采用二维坐标图，就会得到一些离散的点（不考虑资源的差异），并形成一条曲线，见图 17-4。考虑项目允许的质量范围，对照图中的数据，找出相应的参数。根据得到的参数，确定一个合适的进度计划。

### 2. 进度与成本的关系

在项目管理水平不够高的情况下，经常出现进度一直拖后，成本（人力资源  $\times$  时间）却居高不下。这里需要提高测试项目中进度与成本关系的控制水平。由于软件测试项目受规格说明书修改、设计修改、代码修改等影响比较大，只要某几个地方改几下，可能开发、

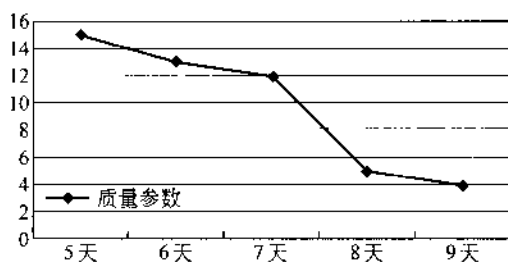


图 17-4 通过参数调整进度和质量的关系

设计只要花很少的时间人力资源，但测试由于得不到足够信息或过多采用黑盒测试方法，其测试成本很大。

还要指出的一点是，要对学习曲线有深刻地认识。在软件开发过程中，学习曲线（learning curve）有很大的用途。通常情况，在开发与上一个相似或同类型的新软件项目后，会比上一次节省 15%~20% 的时间。

### 17.5.3 测试项目进度的管理方法和工具

在软件测试管理中最重要、最基本的就是测试进度跟踪。众所周知，在进度压力之下，被压缩的时间通常是测试时间，这导致实际的进度随着时间的推移，与最初制定的计划相差越来越远。而如果有正式的度量方法，这种情况就很难出现，因为在其出现之前就有可能采取了行动。下面就介绍两测试项目进度的管理方法：测试进度 S 曲线法、缺陷跟踪曲线法。缺陷跟踪又可以分为新发现缺陷跟踪法和累计缺陷跟踪法，而以累计缺陷跟踪法比较好。关于缺陷跟踪趋势图，在前面两章已有介绍，这里将主要讨论它在进度管理的应用。

#### 1. 测试进度 S 曲线法

进度 S 曲线法通过对计划中的进度、尝试的进度与实际的进度三者对比来实现的，其采用的基本数据主要是测试用例或测试点的数量；同时，这些数据需按周统计，每周统计一次，反映在图表中。“S”意思是，随着时间的发展，积累的数据的形状越来越像 S 形。可以看到一般的测试过程中包含三个阶段，初始阶段、紧张阶段和成熟阶段，第一和第三个阶段所执行的测试数量（强度）远小于中间的第二个阶段，由此导致曲线的形状像一个扁扁的 S。

X 轴代表时间单位（推荐以“周”为单位），Y 轴代表当前累计的测试用例或者测试点数量，如图 17-5 所示，可以看到：

(1) 用趋势曲线（上方实线）代表计划中的测试用例数量，该曲线是在形成了测试计划之后，在实际测试执行之前事先画上的。

(2) 测试开始时，图上只有计划曲线。此后，每周添加两条柱状数据，浅色柱状数据代表当前周为止累计尝试执行的测试用例数，深色柱状数据为当前周为止累计实际执行的测试用例数。



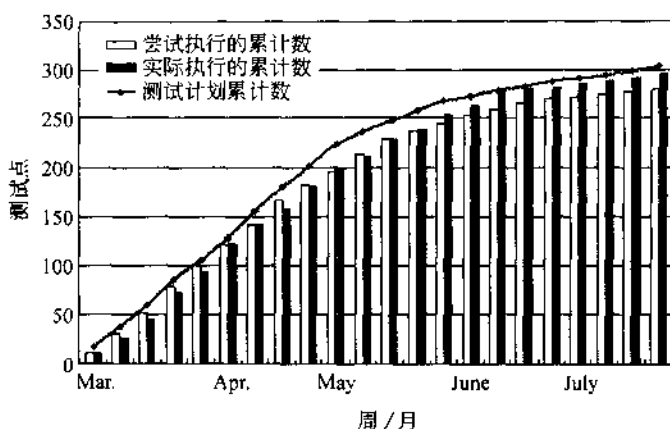


图 17-5 计划中的、尝试的与实际进度曲线图

(3) 在测试快速增长期（紧张阶段），尝试执行的测试用例数略高于原计划，而成功执行的则略低于原计划，这种情况是经常出现的。

由于测试用例的重要程度有所不同，因此，在实际测试中经常会给测试用例加上权重（test scores）。使用加权归一化（normalized）使得 S 曲线更为准确地反映测试进度（这样 Y 轴数据就是测试用例的加权数量），加权后的测试用例数通常称为测试点（test-point）。

一旦一个严格的计划曲线放在项目组前，它将成为奋斗的动力，整个小组的视线都开始关注计划、尝试与执行之间的偏差。由此，严格的评估是 S 曲线的成功的基本保证，例如，人力是否足够、测试用例之间是否存在相关性等。一般而言，在计划或者尝试数与实际执行数之间存在 15%~20% 的偏差就需要用于启动应急行动来进行弥补了。

一旦计划曲线被设定，任何对计划的变更都必须经过审查（review）。自然，需要严格的程序规范，否则计划成了变化，如同儿戏。同时，一般而言，最初的计划应作为基准（baseline），即使计划作了变更，也留作参考。该曲线与后来的计划曲线的对比显现的不同之处需要给出详尽的理由作为说明，同时也是此后制定计划的经验来源之一。

## 2. 测试进度 NOB 曲线法

测试所发现的软件缺陷数量，一定程度上代表了软件的质量，通过对它的跟踪来控制进度也是一种比较现实的方法，受到测试过程管理的高度重视。在第 16 章，我们分析在整个测试期间三个重要数据：新发现的累计缺陷数、已处理的累计缺陷数和已关闭的累计缺陷数。在这里主要收集当前所有打开的（激活的）缺陷数（NOB, number of open bug），也可以将严重级别的缺陷分离出来进行控制，从而形成 NOB 曲线，在很大程度上反应了软件的质量和测试的进度。

在 NOB 曲线法中，最重要的是确定基线数据或者典型数据，即实现为测试进度设计一套计划曲线或理想曲线。至少在跟踪开始的时候，需将项目进度关键点（里程碑）预期的 NOB 限制等级设置好，以及确定什么时间 NOB 达到高峰，NOB 在测试产品发布前能否降到足够的低。比较理想的模式是，相对于之前发布的版本或者基线，NOB 达到的高峰期出现的更早，在发布前降到足够的低并且稳定起来。值得注意的是，在这种度量方法里仅仅

注意数量是不够的,为了尽早达到系统的稳定,什么类型的缺陷以及具体哪些缺陷应该首先修正是非常重要的。

尽管 NOB 应该一直都被控制在合理的级别上,但是当功能测试的进展是最主要的开发事件时,应该关注的是测试的有效性和测试的执行,并在最大程度上鼓励缺陷的发现。过早地关注于 NOB 减少,将导致目标冲突,导致潜在的缺陷逃逸或者缺陷发现的延迟。因此,在测试紧张阶段,主要应该关注的是那些阻止测试进展的关键缺陷的纠正。当然,在测试接近完成时,就应该强烈关注 NOB 的减少,因为 NOB 曲线的后半部分尤为重要,因为它与质量问题密切相关如图 17-6 所示。

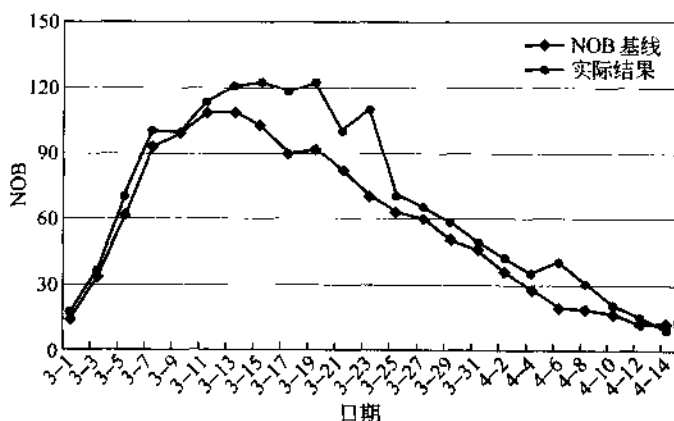


图 17-6 NOB 进度曲线示意图

Myers 有一个关于软件测试的著名的反直觉原则:在测试中发现缺陷多的地方,还有更多的缺陷将会被发现。这个原则背后的原因在于:如果测试效率没有被显著的改善,那么在纠正缺陷时,将引入越多的错误。因此,这种度量方法可以用下面的方式来诠释:

- 如果缺陷发生率跟以前发布的一个版本(或模板)相同或者更低,就应该考虑当前版本的测试是不是低效?如果不是,那么质量的前景是乐观的;如果是,那么就需要额外的测试。除了要对当前的项目采取措施,还需要对开发和测试的过程进行改善。
- 如果缺陷发生率比以前发布的一个版本(或模板)更高,那么就应该考虑我们是否为显著提高测试效率做了计划,并实际上做到了这一点?如果没有,那么质量将得不到保证;如果是这样,那么质量将得到保证或者可以说是乐观的。

对于这种度量法来说,其经常用于特定测试中缺陷达到的度量,比如功能测试、产品级测试、系统集成测试。

### 3. 测试进度管理工具

“工欲善其事,必先利其器”。要做好项目管理,首先得要有一个可以规划、跟踪、控制和改进项目管理的工具。微软公司推出的 Project,就是一个常用的、专业的项目管理工具,提供了项目管理所需的功能,可以细致地反映项目进行的整个过程,便于跟踪项目的进展、项目的分工等。它把一个任务划分为以下四个阶段进行管理,即:比较基准计划(原

始计划)、当前计划、实际计划和待执行计划(剩余计划或未完成计划)。

(1) 比较基准计划(原始计划):这里的计划数据记录了最初制订项目计划时项目的状态情况。由于项目一旦开始运作执行,项目计划总是处于动态变化中,如何评价计划的实施情况或计划本身设计的问题,需要随时可以获得原始计划数据,Project 把最初编制的计划作为“比较基准”存储起来,在项目调整过程中始终保持不变。

(2) 当前计划(正在进行):项目启动以后,由于主观或客观的原因,计划总是处在变化中,所以需要反映项目实际执行的计划。由于当前计划是根据实际已经发生的计划和任务间的制约关系而计算出来的,因此对于项目计划的管理和预测都具有现实的指导意义。

(3) 实际计划(已完成的):指那些已经开始实施但未完成,或已经全部完成的任务计划。已经实际执行的计划在项目管理中的重要性有两点,即它是计算项目产值的依据,也是规划和预测当前和未来计划的基本信息。

(4) 待执行计划(未完成的):不仅要考虑已经完成了多少任务,还必须知道有多少剩余的工作量需要完成,即待执行计划。如果一个任务已经开始但还没有做完,系统会根据完成情况自动计算剩余工作量,并重新测算需要的工期和成本。

对于时刻处在变化之中的项目计划来说,由人工来统计资源在整个项目中的安排日程是一件非常繁琐的事情,Microsoft Project 提供的“资源使用状态”视图,逐个列出资源承担的任务、在每个任务上工作的日期、人数、工作量、费用、累计工作量和累计费用等,通过从多角度、丰富的图表来描述。

- 网络图:以描述任务关系为重点的信息,它可以选择任意五种任务信息进行显示。通过选择不同类别的信息,可以建立基本信息网络图、基线网络图、跟踪网络图、费用网络图等。
- 横道图:是以表述任务时间关系为重点的信息,显示出工序的关系线,即每个任务的开始、结束时间,而且能够知道任务的紧前和紧后工序,就像在网络图上工作一样。横道条上可以把每个任务的基线计划、当前计划、实际计划、完成百分比、时差同时显示出来,便于进行综合分析。
- 资源图:反映资源使用状况为重点的信息,为资源分析和跟踪提供了八种图形,即资源需求曲线图、资源工作量图、资源累计工作量图、超分配工作量图、资源已经分配的百分数图、资源当前可用工作量、成本图、累计费用图等。

Microsoft Project 内置了多种筛选器,帮助建立各种文字报告,内容覆盖面广,可以直接使用。

- 项目摘要报告:项目汇总报告。
- 任务报告:未开始、正在进行的、已经完成的、推迟开始的、马上开始的、进度拖后的等各种任务报告,还包括关键任务、使用某种资源的、超出预算的、本周/月/季度的等各种任务报告,任务基本信息报告。
- 资源报告:预算报告,超强度分配、超出预算等资源报告,资源工作安排报告,工作量报告,周(月、季、年)现金流报告,资源基本信息报告。

## 17.6 测试项目的风险管理

软件测试项目存在着风险，但如果在项目管理中，预先重视风险的评估，并对要出现的风险有所防范，就可以最大限度地减少风险的发生或降低风险所带来的损失。

### 1. 风险的分类

根据风险出现的情况可分为可避免的风险和不可避免的风险两种，前者可以采取预防措施预防其发生，后者是不管采取什么措施都不可避免，只能降低风险所带来的损失。或者也可以分为已知风险（如有一两个员工不稳定等）、可预报风险（根据以往经验能预测可能出现的风险）和不可预报的风险。

根据风险内容，可以将风险划分为：

- 项目风险，如用户需求变化、成本提高，时间延长等。
- 技术风险，技术不成熟、技术转换、技术培训弱等造成的风险。
- 商业风险，市场不清楚、市场不稳定等。
- 战略风险，公司的经营战略需要调整时所产生的风险。
- 管理风险，管理人员的能力和素质、组织机构与新的流程适应性等带来的风险。

### 2. 风险管理的内容

风险的管理基本的内容有两项：风险评估和风险控制。

#### (1) 风险评估

- 识别风险
- 对已经识别出来的风险进行分析
- 确定这些风险的特点或可能带来的危害

#### (2) 风险控制

- 降低风险
- 风险管理计划
- 风险应急处理方法

### 3. 风险评估

对风险的评估主要依据三个因素：风险描述、风险概率和风险影响。从成本、进度及性能三个方面对风险进行评估。风险的评估是建立风险识别和分析的基础上。

在风险管理中，首先要将风险识别出来，特别是确定哪些是可避免的风险，哪些是不可避免的，对可避免的风险要尽量采取措施去避免，所以风险识别是第一步，也是很重要的一步。风险识别的有效方法是建立风险项目检查表，按风险内容进行分项检查，逐项检查。然后，对识别出来的风险进行分析，主要从下列四个方面进行分析：

- (1) 发生的可能性（风险概率）分析，建立一个尺度表示风险可能性（如，极罕见、

罕见、普通、可能、极可能)。

(2) 分析和描述发生的结果或风险带来的后果, 即估计风险发生后对产品和测试结果的影响、或造成的损失等。

(3) 确定风险评估的正确性, 要对每个风险的表现、范围、时间做出尽量准确的判断。

(4) 根据损失(影响)和风险概率的乘积, 来排定风险的优先队列。

方法可以采用 FMEA 法(failure mode and effects analysis, 失效模型和效果分析)。

#### 4. 风险的控制

风险的控制是建立在上述风险评估的结果上, 主要工作有:

(1) 采取措施避免那些可以避免的风险, 如测试环境不对可以通过事先列出要检查的所有条目, 在测试环境设置好后, 由其他人员按已列出条目逐条检查。

(2) 风险转移, 有些风险可能带来的后果非常严重, 能否通过一些方法, 将它转换为其他一些不会引起严重后果的低风险。如产品发布前夕发现某个不是很重要的新功能, 给原有的功能带来一个严重 bug, 这时处理这个 bug 所带来的风险就很大, 对策是去掉那个新功能, 转移这种风险。

(3) 有些风险不可避免, 就设法降低风险, 如“程序中未发现的缺陷”这种风险总是存在, 我们就要通过提高测试用例的覆盖率(如达到 99.9%)来降低这种风险。

(4) 为了避免、转移或降低风险, 事先要做好风险管理计划, 包括单个风险的处理和所有风险综合处理的管理计划。

(5) 对风险的处理还要制定一些应急的、有效的处理方案。

风险管理的完整内容和对策, 如图 17-7 所示。

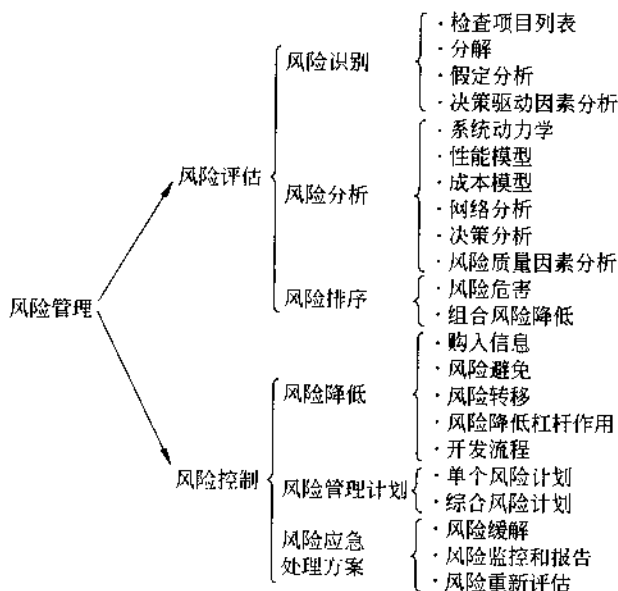


图 17-7 风险管理的内容和对策

控制风险还有一些其他策略，如：

- 在做计划时，资源、时间、预算等估算时，要留有余地，不要用到100%。
- 在项目开始前，把一些环节或边界上的有变化、难以控制的因素列入风险管理计划中。
- 对每个关键性技术人员培养后备人员，作好人员流动的准备，采取一些措施确保人员一旦离开公司，项目不会受到严重影响，仍能可以继续下去。
- 制订文档标准，并建立一种机制，保证文档及时产生。
- 对所有工作多进行互相审查，及时发现问题。

对所有过程进行日常跟踪，及时发现风险出现的征兆，避免风险。

## 17.7 测试项目的质量和配置管理

随着软件开发的规模越来越大，软件的质量问题显得越来越突出。软件质量的控制不单单是一个软件测试问题，在软件开发的所有阶段都应该引入软件质量管理和配置管理。

### 1. 质量管理的基本原则

- 控制所有过程的质量。
- 过程控制的出发点是预防不合格。
- 质量管理的中心任务是建立并实施文件化的质量体系。
- 持续的质量改进。
- 有效的质量体系应满足顾客和组织内部双方的需要和利益。
- 定期评价质量体系。
- 搞好质量管理关键在于领导和全员质量意识/文化。

### 2. 软件评审

软件评审并不是在软件开发完毕后进行评审，而是在软件开发的各个阶段都要进行评审，评审也可以看做是软件测试的一部分工作。因为在软件开发的各个阶段都可能产生错误，如果这些错误不及时发现并纠正，会不断地扩大，最后可能导致开发的失败。

(1) 评审目标是更早地发现任何形式表现的软件功能、逻辑或实现方面的错误，审验证软件的需求，保证软件按预先定义的标准表示。

(2) 评审过程，要经过准备（拟定主体和讨论项目）、反馈收集、会议并达成一致的结论、结论执行跟踪等各个阶段。

(3) 评审准则主要有以下内容：

- 评审产品，而不是评审设计者，不能使设计者有任何压力。
- 会场要有良好的气氛。
- 建立议事日程并维持它，会议不能脱离主题。
- 限制争论与反驳，评审会不是为了解决问题，而是为了发现问题。

- 指明问题范围，而不是解决提到的问题。
- 展示记录（最好有黑板，将问题随时写在黑板上）。
- 限制会议人数和坚持会前准备工作。
- 对每个被评审的产品要尽力评审清单（帮助评审人员思考）。
- 对每个正式技术评审分配资源和时间进度表。
- 对全部评审人员进行必要的培训。
- 及早地对自己的评审做评审（对评审准则的评审）。

### 3. 配置管理

软件配置管理简称 SCM (Software Configuration Management 的缩写)，是在团队开发中，标识、控制和管理软件变更的一种管理，所以配置管理对软件测试和质量保证影响比较大，其影响程度取决于项目规模和复杂性、人员素质、流程、管理水平等。

软件配置管理分为版本管理、问题跟踪和建立管理三个部分，其中版本管理是基础。版本管理应完成以下主要任务：

- 建立项目。
- 重构任何修订版的某一项或某一文件。
- 利用加锁技术防止覆盖。
- 当增加一个修订版时要求输入变更描述。
- 提供比较任意两个修订版的使用工具。
- 采用增量存储方式。
- 提供对修订版历史和锁定状态的报告功能。
- 提供归并功能。
- 允许在任何时候重构任何版本。
- 权限的设置。
- 晋升模型的建立。
- 提供各种报告。

## 17.8 软件测试文档的管理

软件测试项目其实是一个交互的过程，包括客户所提交的需求文档、开发人员所提交的设计文档，这些都是测试工程师做测试的指导性文件，测试工程师应该就这些文档和客户以及开发工程师进行深入、广泛的交流，以期对某些有争议的问题达成共识。这些交流的过程也应该以某种形式记录下来，这对于后期解决某些不明确的问题，也是很好的证明资料。测试工程师的测试报告、bug 报告、开发工程师对相关 bug 所给出的解释以及双方就某些情况所做的交流都是很宝贵的信息，应该尽可能地保存下来，以便给其后的测试提供借鉴。在特定项目过程中，解决问题的成功模式和方法可以系统地保留下来。

每一个测试项目过程中都会产生很多文档，从项目启动前的计划书到项目结束后的总

结报告，其间还有产品需求、测试计划、测试用例和各种重要会议的会议记录等。软件测试文件就为了实现这些目的，对测试中的要求、过程及测试结果以正式的文件形式写出，所以说测试文件的编写是测试工作规范化的一个重要组成部分，有必要将文档管理融入到项目管理中去，成为项目管理很重要的一个环节。文档管理所包含的主要内容为以下几个方面：

- 文档的分类管理
- 文档的格式和模板管理
- 文档的一致性管理
- 文档的存储管理

### 1. 测试文档的分类管理

测试文件简单地分为两类，测试文档模板和测试过程中生成的文档。测试文档模板是对相应要生成的文档所定义的格式、内容做出严格要求的示范文档。基本的测试文档模板有：

- 测试计划文档模板
- 测试需求分析模板
- 测试用例模板
- 测试评审模板
- 测试报告模板

同时，可以按照输入媒介来分为电子文档、纸质文档和其他一些特殊文档，对于电子文档和、纸质文档存储和管理的办法都是不一样的，应该分别对待。多数情况下，按照文档的用途来划分，可以分为以下几种：

- 测试日常工作文档（流程定义、工作手册等）
- 测试培训文档和相关技术文档
- 测试计划、设计文档
- 测试跟踪、审查资料
- 测试结果分析报告或产品发布质量报告

实际上，不论是作为测试小组还是作为测试部门，除了要管理测试本身的文档，还要管理外部输入的文档和软件产品文档。外部输入的文档主要包括系统需求分析报告、设计规格说明书、项目计划书等；软件产品文档包括发布说明、用户手册、技术手册、安装说明、帮助文档等。

### 2. 测试文档的存储和共享

我们知道，要管理的测试文档很多，一方面要能很好、可靠地进行存储，另一方面又能有效地、充分地利用这些文档，这两个方面是相辅相成的，需要统一考虑。

要做好测试文档的存储，事先要做好各种准备，从文档的分类、文件名的格式、文件的模板等严格要求测试文件的编制。对于文件名，虽然是一个小问题，做得不好引起麻烦也挺多的，所以要有明确的规定，要求文件名必须用英文，并包含测试组名、项目名、文



件类型、日期等，如：

T-Team Name-Project Name-Weekly Report-2004-3-08.doc

——T 代表测试类文档。

接下来就是要求按照完整的模板逐条写出所需要的计划书、报告等。

文档存储要和怎么使用这些文档结合起来做，也就是说，从测试文档的使用目的来进行文档存储的规划和设计。测试文档的使用可以分为个人使用、项目组内部使用和所有测试人员都需要使用，其存储也就服务这三个对象，并考虑具体的使用方法。概括起来，文档存储的规划、设计要考虑以下因素：

(1) 共享方式：共享目录、FTP 方式、HTTP 方式。

(2) 手段：自己开发文档管理系统，或借助第三方的商品化软件，如 Microsoft Sharepoint。

(3) 安全性：测试文档一般比较多地涉及公司内部机密信息，需要保证其安全性，严格设置相关的用户权限体系。

(4) 目录结构：目录可以按照团队、项目、文件类型的多层次关系设置。

(5) 操作要灵活，包括存取、上载、修改 (Update)、阅读等各项操作。

### 3. 文档模板

在做软件测试项目的时候，有些文档是每个项目都必备的，如测试计划书、测试案例、测试项目报告、质量分析报告等，对于这些经常使用的文档类型，就可以把格式和内容统一起来，为每一种类型的文档建立相对固定的模板。模板建立之后，便于文档的管理和分类，也为测试工程师提供便利，比较容易编制、写成所需要的测试文档。整个开发团体的其他成员对同类文档的格式非常熟悉，可以直接去查找自己最关心的部分，比较清晰，一目了然。

对于特定的项目，文档模板可以酌情增删其中的条目。制定模板的初衷是为了方便工作，而不是禁锢我们的思维，在做具体工作的时候，一定要把原则性和灵活性掌握好。

## 小结

软件测试项目的管理，在共性上继承了软件工程和管理学的项目管理理念、方法、技术和工具，这其中自然包括过程管理、进度管理、资源管理、风险管理和文档管理等领域的成果继承。

但在软件测试项目的管理中，会遇到一系列特有的问题，如软件质量标准定义不准确、任务边界模糊、软件测试项目的变化控制和预警分析要求高、项目成员的责任心和稳定性对测试项目的质量有很大的影响等。由于这些问题，软件测试项目的管理更需要细致，风险管理、流程跟踪和改进更为重要，需设法在项目组织、过程监控和质量管理等方面去处理。

软件测试项目管理另一个焦点集中在过程管理，从测试项目准备、测试计划阶段、测试设计阶段一直到执行阶段，都要从测试本身的特点出发，围绕产品质量线索展开，借助

测试管理系统，最终保障测试项目的顺利进行。

## 思考题

1. 设想您被指定为某软件测试项目的组长，您认为自己该着手准备哪些事情？
2. 有人说没有学过项目管理的人，根本就做不好项目管理。谈谈您对这种说法的理解。
3. 只要搭建正确的测试环境，并且拥有足够的资源，就能够保证测试项目的成功。您认同这个观点吗？
4. 既然我们事先制定了项目计划和规则，就要始终严格按照计划和规则办事。您觉得呢？
5. 如何把握项目管理中的资源分配？
6. 如何在项目的进程中实现对测试工程师的培训？
7. 测试是对产品质量的监督，所以测试工程师最大的任务就是保证产品按照客户的要求实现其功能，至于和开发工程师的沟通，这些不属于测试工程师的工作范畴。
8. 请结合具体的项目说说，在软件项目管理中最重要的因素是什么？

# 附录 A 软件测试的英文术语 及中文解释

## - A -

### Acceptance testing 验收测试

软件或硬件开发的一个测试阶段，以检验所测试的系统是否正确实现了所有的用户需求，以保证其达到可以交付使用的状态。通常是产品发布之前最后一个测试阶段。

### Accessibility 可接近性

使组成软件的各部分便于选择使用或维护的程度。

### Active or open 激活状态

未被解决的缺陷的一种状态，如新报告的 bug，或验证后 bug 仍然存在的状态。

### Adaptability 适应性

使不同的系统约束条件和用户需求得到满足的容易程度。

### Analytical model 分析模型

用一组可解方程来表示一个过程或一个现象。

### Architecture 体系结构

系统各部件之间的结构和关系。

### Automated test case generator 自动测试用例生成器

### Audit 审核

为获得审核证据并对其进行客观的评价，以确定满足经协商的准则的程度所进行的系统的、独立的并形成文件的过程。

### Audit scope 审核范围

审核的广度和界限。

### Auditor 审核员

### Auditor qualifications 审核员资格

### Availability 可用性

软件在投入使用时能实现其指定的系统功能的概率，可用系统正常工作时间和总的运行时间之比计算。

## - B -

### Behavioral test 行为测试

基本计算机系统、硬件或软件假定完成的用途和功能测试，根据产品特征、操作描述

和用户方案进行的。也被称为黑盒测试或功能测试。

**Baseline 基线**

在配置项目生存周期的某一特定时间内,正式指定或固定下来的配置标识文件和一组这样的文件或数据,可用做下一步开发的基础。

**Black-box test 黑盒测试**

不管程序内部结构是什么样的,把系统或软件看成一个黑盒,只是根据输入输出条件、边界条件和限制,从用户观点/数据驱动的方式,来验证产品所应该具有的功能是否实现,是否满足用户的要求。参见行为测试。

**Bottom-up Integration 自底向上系统集成方法****Boundary condition 边界条件**

描述极值的测试需求。如果子系统取范围[0...12]内的数,则 0 和 12 就是边界。边界条件比内部值更容易发现程序缺陷,因为程序员常常在边界犯错误。

**Bug 软件缺陷**

系统或程序中隐藏的功能缺陷、错误或瑕疵,而导致软件产品在某种程度上不能满足用户的需要。

**Bug crawl 错误评审会议**

集中对测试系统中每个被报告的现存错误进行逐项评审的会议或讨论。在评审过程中,可以指定错误修改日期、优先级、推迟处理不严重的错误。也被称为错误剔除(bug scrub)。

**Build 构件**

软件产品的一个工作版本,其中包含最终产品将拥有的能力的一个规定的子集。

**B/S, Browser/Server 浏览器/服务器(结构)**

## - C -

**Capability 能力**

组织、体系或过程实现产品并使其满足要求的本领。

**Capacity test 容量测试**

预先分析出反映软件系统应用特征的某项指标的极限值,如某个 Web 站点可以支持的多少个并发用户的访问量。

**Certification 认证**

证实软件系统或程序在其运行环境中能满足规定的需求的过程,认证使验证和确认的过程扩充到实际的或模拟的运行环境中。

**Change control 变更控制管理**

提议做一项改变,并对其进行估计、同意或拒绝、调度和跟踪的过程。

**CCB, Change Control Board 变更控制委员会**

控制需求变化、代码修改的一种内部组织。

**Characteristic 特性**

区分的特征。

**Close or inactive 关闭或非激活状态**

已经被解决的缺陷的一种状态,即被测试人员验证后,确认 bug 不存在之后的状态。

**Closure period 修复周期**

在最初的缺陷报告到修复确认之间的时间。修复时期是衡量开发部门对缺陷报告响应的速度。

**Code audit 代码审计**

由某人或小组对源代码进行的独立的审查,以验证其是否符合软件设计文件和程序设计标准,还可能对正确性和有效性进行估计。

**Code Completed 代码完成**

在某个版本所有新功能和改动的代码完成时间,是软件开发周期中的一个重要里程碑。

**Code Freeze 代码冻结**

所有的缺陷修改之后,不允许对代码做任何非授权的改动起始时间,是另一个重要的里程碑。

**Code inspection 代码审查****Code walk-through 代码走查****Cohesion 内聚度**

单个程序模块所执行的诸任务在功能上的互相关联的程度。

**Compatibility 兼容性**

软件从一个计算机系统或环境移植到另一个系统或环境的容易程度。

**Compatibility testing 兼容性测试**

测试在特殊的硬件/软件/操作系统/网络环境下的软件表现。

**Compile 编译**

将高级语言程序变换成与之等价的浮动的或绝对的机器代码。

**Complexity 复杂性**

系统或系统组成部分的复杂程度,如接口的数量和错综程度、条件转移的数量和错综程度、嵌套的深度、数据结构类型等。

**Component 组件,部件**

构成系统或程序的基本部分、具有独立功能的单元。

**Component testing 组件测试**

在系统集成之前,对构成系统的各个组件进行测试的阶段,类似于模块测试或单元测试。

**Confirmation tests 确认测试**

一套选定的测试,用于对报告中不能完全修复的缺陷进行测试的方法,包含对每个缺陷报告都重新执行测试过程和隔离步骤。

**Configuration 配置**

为确定系统或系统组成部分的特定版本而在技术文档中提出的需求和制定的产品硬件、软件的特性要求。

**Configuration management 配置管理**

标识和确定系统中配置项的过程，在系统整个生存周期内控制这些项的变化，记录并报告配置的状态和更动要求，验证配置项的完整性和正确性。

**Conformity 合格（符合）**

满足要求。

**Congruent 一致性**

对测试系统体系结构的描述，其中测试系统的所有组件彼此相关，并与测试系统的目标相一致。

**Continual improvement 持续改进**

增强满足要求的能力的循环活动。

**Contractually required audit 合同所要求的审计****Correction 纠正**

为消除已发现的不合格所采取的措施。

**Corrective action 纠正措施**

为消除已发现的不合格或其他不期望情况的原因所采取的措施。

**Coupling 耦合度**

计算机程序中模块之间相互依赖的量度。

**Coverage 覆盖率**

是检查对系统或子系统测试是否彻底的一种程度衡量，是测试质量的近似度量。

**Criteria 准则**

确定为依据的一组方针、程序或要求。

**Critical bug 严重的缺陷**

指导致主要功能或特性没有实现、丧失的严重缺陷。

**Customer 顾客**

接受产品的组织或个人。

**Customer satisfaction 顾客满意**

顾客对其要求已被满足的程度的感觉。

**C/S, Client/Server 客户端/服务器（结构）**

- D -

**Data dictionary 数据字典**

软件系统中的所有数据项的名字、含义及相关特性（如数据项长度、表示等）的集合。

**Data structure 数据结构**

数据项之间的次序安排和可访问性的一种形式表示，其中不涉及其实际存储排列方法。

**Data flow testing 数据流测试**

测试需求需要检验所有已定义的数据输入、处理、输出的一种测试类型。

**Debugging 调试**

开发人员确定引起缺陷的根本原因和确定可能的修复措施的过程,通过调试来修复缺陷。

**Defect 缺陷**

未满足与预期或规定用途有关的要求。

**Delivery 交付**

软件研制周期中的一个阶段,将产品提交给计划中的用户供其使用。

**Dependability 可信性**

关于可用性及其影响因素:可靠性、维修性和维修保障性的全部特性。

**Design and development 设计与开发**

将要求转换为规定的特性或产品、过程或体系的规范的一组过程。

**Design specification 设计规格说明**

描述设计要求的正式文档,对系统或系统组成部分(如软件配置项、算法、控制逻辑、数据结构、输入输出格式和接口等)进行设计。

**Development life cycle 开发生存周期**

**Deviation permit 偏离许可**

产品实现前,偏离原规定要求的许可。

**Distributed testing 分布式测试**

在多个位置、涉及到多个开发小组或两者兼备的情况下进行的测试。

**Document 文件**

信息及其承载媒体。

**Driver 驱动模块**

对底层或子层模块进行集成测试中,所编制的调用这些模块的程序。

**- E -**

**Effectiveness 有效性**

完成策划的活动并达到策划的结果的程度。

**Efficiency 效率**

得到的结果与所使用的资源之间的关系。

**Encapsulation 封装**

将系统功能隔离在一个模块中,并为该模块提供精确的规格说明的面向对象技术。

**Entry criteria 进入标准**

一套决策的指导方针或数据指标,用于决定项目是否准备好进入特定的测试阶段。

**Error seeding 错误播种**

一种有效性的理论方法。在测试时向被测试系统中加入已知的错误,然后检查这些已知错误被发现的比例,通过这个比例来衡量被测试系统中残留的错误并测量测试系统自身。一般不采用这个方法,人们普遍怀疑该方法的精确性。

**Error, faults and failures 错误、缺陷与失效**

根据[IEEE83], 错误是人为的失误, 产生一个或多个缺陷, 这些缺陷被嵌入在程序的文本中。执行有缺陷的代码时, 会产生零个或多个失效。

**Escalate** 向上呈交

将问题递交到更高级管理层寻求解决方案。

**Evaluation** 评价, 评估

决定某产品、项目、活动或服务是否符合它的规定的准则的过程。

**Exit criteria** 退出标准

与 **Entry criteria** 相反, 这里是决定项目是否可以退出或结束当前的测试阶段的标准或指标。

**Experiencia of quality** 质量体验

客户和用户对于系统是否实现他们的期望和需求的看法。

**Exploratory testing** 探索性测试

并行地进行测试的设计、开发和执行, 通常伴随着学习被测试的系统和不重要的测试文档。

## - F -

**Failure** 失效

系统或系统部件丧失了(在规定的限度内)执行所要求功能的能力。

**Fatal bug** 致命的缺陷

造成系统或程序崩溃(Crash)、死机、悬挂、数据丢失或主要功能完全丧失等缺陷。

**Fault injection** 错误注入

参见 **Error seeding** (错误播种)。

**Fault of omission** 遗漏缺陷

通过补充文本可以改正的缺陷。

**Feasible coverage** 可行覆盖率

已满足覆盖率条件百分比的一种度量, 刨去不可能或太难满足的部分。

**Feature** 产品特性

系统中满足用户需求的某种功能、表现和特征。

**Fidelity** 逼真度

对测试系统, 是指准确构造客户的硬件、软件和网络环境的模型和模拟客户行为的程度。

**Field-reported bug** 现场报告缺陷

通常由客户或销售人员报告在发布、部署或交付产品时的缺陷。

**First customer ship** 首位客户送货

对于大宗市场系统, 指它的系统完成彻底的测试、形成产品、发送给首位付款客户的阶段。也被称为发布或通用有效性。

**Fixed or Resolved** 已修正状态



缺陷被开发人员处理过并通过开发人员单元测试的状态，还需要测试人员的验证。

**Flexibility 灵活性**

测试组件能处理在被测试系统的细小改变，而不报告不存在的或确实存在的缺陷的程度。

**FMEA 失效模型和效果分析**

Failure Mode and Effects Analysis 的缩写，一种用于识别和定义潜在质量风险的方法，该方法按风险的优先级别排序，并对每一个风险取相应措施预防和/或发现相关问题。

**Functional Specification 产品功能规格说明（书）**

产品或系统要实现的、满足用户需求的各种功能、特性和界面的描述（文档）。

**Functional tests 功能测试**

参见行为测试或黑盒测试，但它还意味着集中于功能正确性方面的测试。功能测试必须和其他测试方法一起处理潜在的重要的质量风险，比如性能、负荷、容量等。

**Functionality 功能性**

软件所实现的功能达到它的设计规范和满足用户需求的程度。

## - G -

**GA 通用有效性**

General Availability 的简称，参见首位客户送货。

**Grade 等级**

对功能用处相同但质量要求不同的产品、过程或体系所作的分类或分级。

**Granularity 粒度**

聚集的精确度或粗糙度。高粒度测试让测试者检查低级细节，如结构测试。而行为测试不是低粒度测试，它给测试者提供整体系统行为的信息，而不是细节。

## - I -

**Ideal fault condition 理想缺陷条件**

在测试理论中，指可达性、必要性和传播性条件。

**Identifier 标识符**

用以命名、指示或定位的符号。标识符可以和数据结构、数据项或程序位置相关联。

**Information 信息**

有意义的资料。

**Infrastructure 基础设施**

组织运行所必需的一组设施、设备和服务。

**Implementation requirement 实现需求**

对软件设计的实现产生影响或限制的任何需求。例如，设计描述、软件开发标准、程序设计语言需求、软件质量保证标准等。

**Input 输入**

一般术语，表示子系统所操作的任何数据，包括文件输入、函数参数传入值、全局变量值等。

**Inspection 检验**

通过观察和判断，必要时结合测量、试验所进行的符合性评价。

**Integration testing 集成测试**

在单元测试的基础上，按照设计要求，将所有单元（模块/组件）组装成为系统而进行的测试，通过测试，可以发现单元之间关系和接口中的错误。

**Interoperability 互操作性**

两个或多个系统交换信息并相互使用已交换的信息的能力、或可互相操作的能力。

**Isolation 隔离**

对缺陷进行分析而找出导致问题的根本原因。如通过重复那些用于再现缺陷的步骤，并改变系统配置、权限、负载等环境因素或条件，从而识别影响缺陷的因素。

- L -

**Log file 记录文件**

包含测试运行时的实际输出的文件。

- M -

**Maintainable 可维护性**

由于用户新的需求、功能增强或所发现的问题，对已开始使用的系统修改的难易程度。

**Major bug 一般的缺陷**

这样的软件缺陷虽然不影响系统的基本使用，但没有很好地实现功能、没有达到预期效果。如次要功能丧失、提示信息不太准确、用户界面不友好、操作时间长等问题。

**Management 管理**

指导和控制组织的相互协调的活动。

**Management system 管理体系**

建立方针和目标并实现这些目标的体系。

**Measurement control system 测量控制体系**

为完成计量确认并持续控制测量过程所必需的一组相互关联或相互作用的要素。

**Measurement process 测量过程**

确定量值的一组操作。

**Metrological characteristic 计量特性**

可影响测量结果的区分的特征。

**Metrological confirmation 计量确认**

为了确保测量设备处在符合其预期使用要求的状态所要求的一组操作。

**Metrological function 计量职能**

组织建立并实施测量控制体系的职责。

**Milestone 里程碑**

项目中完成阶段性工作的标志,即将一个过程性的任务用一个结论性的标志来描述任务结束的、明确的起止点。

**Minor bug 微小的缺陷**

对功能几乎没有影响,产品及属性仍可使用,如有个错别字、文字排列不对齐等一些小问题。

**Modified Top-down Integration 改进的自顶向下集成**

集成测试中的一种混合策略。

**Modularity 模块性**

软件由若干部分组成的离散程度,即表明改变一个组成部分时对另外的组成部分影响程度。

**Module 模块**

是离散的程序单元,且对于编译、与其他单元相结合是可识别的。

**MTBF 失效平均时间**

Mean Time Between Failure 的缩写,其数据预测系统的现场错误率,暗示了稳定性、可靠性。

**MTTR 平均维修时间**

Mean Time to Repair 的缩写。这个数字表明了系统的可恢复性。

- N -

**Necessity condition 必要性条件**

在测试理论中,采用导致程序出现错误内部状态的值来检验缺陷的需求。

**Nest 嵌套**

把某一类的一个结构或多个结构合并到同一类结构中去,如程序中的循环嵌套,形成自我迭代。

- O -

**Objective evidence 客观证据**

支持事物存在或其真实性的资料。

**Organization 组织**

职责、权限和相互关系得到安排的一组人员及设施。

**Organizational structure 组织结构**

人员的职责、权限和相互关系的安排。

**Orthogonal 正交**

两个或多个变量之间的关系描述，或集合中元素值互不影响的一组集合元素之间关系的描述。

**Output 输出**

作为子系统执行结果的任何内容，包括文件输出、函数返回值等。

**- P -**

**Peer review 同级评审**

在一个项目组内，组员之间相互阅读和审查对方的代码、缺陷报告、计划书等。

**Performance 性能**

在指定条件下，用软件实现某种功能所需的计算机资源（包括时间）的有效程度。

**Performance Test 性能测试**

确定系统运行时的性能表现，如得到运行速度、响应时间、占有系统资源等方面的系统数据。

**Pilot testing 引导测试**

验证系统在真实硬件和客户基础上处理典型操作的能力。其测试通过后可以开始系统的部署。

**Preventive action 预防措施**

为消除潜在不合格或其他潜在不期望情况的原因所采取的措施。

**Priority 优先权**

缺陷要被修复的重要性，从用户的角度看，是指缺陷对系统的可行性和可接受性的影响。

**Procedure 程序，过程**

为进行某项活动或过程所规定的途径。

**Program Specification 概要说明**

**Process 过程**

将输入转换为输出的相互关联或相互作用的、有目标的、有组织的、有次序的一系列活动。

**Product 产品**

过程的结果。

**Product testing 产品测试**

参见集成测试。

**Project 项目**

由一组有起止时间的、相互协调的受控活动所组成的特定过程，该过程要达到符合规定要求的目标，包括时间、成本和资源的约束条件。

## - Q -

**Qualification process 鉴定过程**

证实满足规定要求的能力的过程。

**Qualified auditor 合格审核员**

已成功通过了一个审核员鉴定过程的人员。

**Quality 质量**

是产品或服务满足固有特性（明示或暗示的）要求的程或其能力的特性和特征的集合。

**Quality assurance 质量保证**

通过对软件产品和活动有计划的进行评审和审计来确保任何经过认可的标准和步骤都被遵循、并且保证问题被及时发现和处理。质量管理的一部分，致力于提供能满足质量要求的信任。

**Quality characteristic 质量特性**

有关要求的产品、过程或体系的固有特性。

**Quality control 质量控制**

质量管理的一部分，致力于满足质量要求。

**Quality improvement 质量改进**

质量管理的一部分，致力于增强满足质量要求的能力。

**Quality management 质量管理**

指导和控制组织的关于质量的相互协调的活动。

**Quality management system 质量管理体系**

指导和控制组织的关于质量的管理体系。

**Quality manual 质量手册**

规定组织质量管理体系的文件。

**Quality metric 质量度量**

对软件所具有的，影响其质量的给定属性所进行的定量测量。

**Quality objective 质量目标**

关于质量的所追求的目的。

**Quality plan 质量计划**

对特定的项目、产品、过程或合同，规定由谁及何时应用程序和相关资源的文件。

**Quality planning 质量策划**

质量管理的一部分，致力于制定质量目标并规定必要作业过程和相关资源以实现质量目标。

**Quality policy 质量方针**

由组织的最高管理者正式发布的该组织总的质量意图和质量方向。

**Quality risk management 质量风险管理**

以预防或发现并消除风险为目的，识别、优化、管理被测试系统质量风险的过程。

## - R -

### **railroading 铁路运输方式**

一种在新的测试循环开始时, 不是重新开始测试, 而是按测试包的顺序或从所处位置继续执行测试的技术。这种技术的目标是当范围不是很明确的时候, 测试范围达到可接受程度, 并使得回归测试间距达到最小。

### **Record 记录**

阐明所取得的结果或提供所完成活动的证据的文件。

### **Recovery testing 恢复测试**

对系统崩溃、或失效之后的系统和数据重新恢复的能力和效率。

### **Regression 回归**

作为被测试系统改变的结果, 当系统新的修改版本  $S_{n+1}$  包含了从  $S_1$  次修改到  $S_n$  次修改都没有描述的缺陷时所出现的问题。

### **Regression tests 回归测试**

用于验证改变了的系统或其组件仍然保持应有的特性, 即保证不会因为处理存在的缺陷、添加产品新功能等进行的程序修改而导致原有正常功能的实效。

### **Regression test gap 回归测试间距**

对于被测试系统任何给定的改变或修改, 整个测试系统提供的测试范围和实际重新执行的部分测试系统提供的测试范围之间的差别。

### **Release 产品发布**

对进入一个过程的下一阶段的许可。

### **Reliability 可靠性**

质量的一种特性, 在规定的的时间和条件下, 软件所能维持其性能水平的程度。

### **Reporting logs 报告日志**

测试工具产生的原始测试输出, 如通过 SoftICE 所捕获的程序运行状态的文本文件。

### **Requirement 要求**

明示的、通常隐含的或必须履行的需求或期望。

### **Reusability 可重用性, 复用率**

一个模块可在多种应用中加以利用的程度。

### **Review 评审**

为确定主题事项达到规定目标的适宜性、充分性和有效性所进行的活动。

### **Root cause 根本原因**

缺陷发生的潜在原因, 可能与观察到的缺陷表象不一致。

## - S -

### **Scalability 可扩展性**

软件系统可以在不同规模、不同档次的硬件平台上运行的能力。

**script 脚本**

自动测试工具的程序指令，程序指令由解释性语言（被称为脚本语言）写成。

**Security 安全性、保密性**

对软件系统的保护能力，以防止其受到意外的或蓄意的存取、使用、修改、毁坏或泄密。

**Security testing 安全测试**

测试系统在应付非授权的内部/外部访问、故意的损坏时的防护情况。

**Service manageability 可维护性**

在一个运行软件中，当环境改变或软件发生缺陷时，进行相应修改所做努力的程度。

**Severity 严重性**

缺陷对被测试系统的影响，在终端客户条件下发生的可能性或失败妨碍系统使用的程度。

**SDLC, Software Development Life Cycle 软件开发生命周期**

从需求分析、设计、编程、测试到维护的整个软件开发过程。

**Software development process 软件开发过程****Software engineering 软件工程**

为解决软件开发中各种问题所采用的系统方法和工程技术、工具等的一门学科。

**Specification 规范**

阐明要求的文件。

**Stability 稳定性**

在有干扰或破坏事件影响下仍能保持不变的能力或返回到原始状态的能力。

**Standard combining rules 标准组合规则**

组合测试需求来构成测试规格说明的方式。每个测试需求都至少使用一次，每个测试规格说明要满足尽可能多的合理的测试需求。

**Stress testing 压力测试**

测试是用来检查系统在大负荷条件下系统运行的情况。

**Structured Programming 结构化程序设计**

一种定义良好的软件开发技术，采用自顶向下设计和实现方法，并具有结构化程序的控制构造。

**Structural test 结构测试**

基于代码的或程序结构内部的测试技术和方法，又称为白盒测试。

**Stub 桩模块**

对顶层或上层模块进行集成测试中，所编制的替代下层模块的程序。

**Supplier 供方**

提供产品的组织或个人。

**System testing 系统测试**

软件测试的一个阶段，将软件放在整个计算机环境下，包括软硬件平台、某些支持软件、数据和人员等，在实际运行环境下进行一系列的可用性测试。

## - T -

### TBD 待定

To Be Determined 的简称。在测试文档中表示一项进行中的工作的有用标记。

### Test automation 测试自动化

通过软件工具自动执行软件测试的方法。

### Test case 测试用例

为了特定目的（如考查特定程序路径或验证是否符合特定的需求）而设计的测试数据及与之相关的测试规程的一个特定的集合，或称为有效地发现软件缺陷的最小测试执行单元。在 IEEE829，被称为测试规格说明和测试过程。

### Test case library 测试用例库

独立的、可重用的测试用例集合。

### Test casually 随机测试

模拟客户操作的随意性，进行大量的、自动化的随机测试，来发现今后用户可能会碰到的问题。

### Test coverage 测试覆盖

测试系统覆盖被测试系统的程度。这种度量可以表示为结构化元素（如代码行）或功能点被覆盖的百分比。

### Test environment 测试环境

进行测试的环境，包括测试平台、测试基础设施、测试实验室和其他设施。

### Test escape 测试遗漏

在测试过程中任何应该被捕捉却没有被捕捉到的现场报告缺陷。

### Test phase 测试阶段

指定特殊的质量风险集合并由一个或更多测试通过组成的测试期。

### Test platform 测试平台

被测试系统的任何硬件或软件支持环境，不是测试对象。

### Test specification 测试规格说明

测试规格说明是用来测试子系统的一个特定输入集。测试规格说明满足一个或多个测试需求。例如：单个测试规格说明  $A = -1$  和  $B = 0$  可以满足  $A < B$  和  $A < 0$  两个测试需求。测试规格说明还包含根据这些值执行子系统后，预期应得到结果的精确描述。

### Test suite 测试包，测试套件

一组测试用例的集合、执行框架，是组织测试用例的方法，通过测试用例组合可以创造新的测试条件或满足某个特定的测试目标。

### Test system 测试系统

集成的和可维护项的集合，用于在被测试的软件或硬件中发现、再生产、隔离、描述和管理缺陷。这些项由测试环境、测试过程和测试组件组成。

### Test to fail 基于失效的测试



以尽可能多地发现系统功能失效、问题为目的，在设计、开发和执行测试时所涉及的想法。这个态度代表了测试的正确思维方法。

**Test to pass 基于通过的测试**

以证明符合需求和操作的正确性为目的，在设计、开发和执行测试时所涉及的想法。主要用于验收测试。

**Test tool 测试工具**

应用于测试用例执行、安装或撤销测试环境、创造测试条件或者度量测试结果的过程中的软件系统或程序。

**Testability 可测试性**

软件的一种性质，表明既便于测试准则的建立又便于就这些准则对软件进行评价的程度。

**Tolerance Test 容错测试**

对系统在各种异常条件下提供继续操作的能力的测试。

**Traceability 可追溯性**

追溯所考虑对象的历史、应用情况或所处场所的能力。

**- U -**

**Unit testing 单元测试**

指一段代码、一个函数或子程序、模块或组件的基本测试，一般由开发者执行，采用白盒测试方法，可从程序的内部结构出发设计测试用例。

**Usability 可用性**

软件在用户学习、操作和理解等方面所做努力的程度，如安装性、使用性、界面友好性等，并能否适用于不同特点的用户，包括对残疾人、有缺陷的人能提供产品使用的有效途径或手段。

**- V -**

**Validation 确认**

通过提供客观证据对特定的预期使用或应用要求已得到满足的认定，要能保证所生产的软件可追溯到用户需求的一系列活动。

**Verification 验证**

即检验软件是否已正确地实现了产品规格书所定义的系统功能和特性。验证过程提供证据表明软件相关产品与所有生命周期活动的要求（如正确性、完整性、一致性、准确性等）相一致。

**Version 版本**

某一配置项的一个可标识的实例。

## - W -

### **White-box tests 白盒测试**

在已知产品的内部工作过程（如计算机程序的结构和语句），检验程序中的变量状态、语句、路径、条件、逻辑结构等是否符合设计规格要求、或达到预定要求的结果。参见结构测试。

### **Walk-through 走查**

评审的一种方式，指由某个设计/开发者通读已书写的设计或编码，其他成员负责提出问题并对有关技术、风格、可能的错误、是否违背开发标准的地方等进行评论。

# 附录 B 质量管理体系——要求

(国家标准 GB/T19001-2000, Idt ISO9001:2000)

---

## 1 范围

### 1.1 总则

本标准有下列需求的组织规定了质量管理体系要求：

- a) 需要证实其有能力稳定地提供满足顾客和适用的法律法规要求的产品；
- b) 通过体系的有效应用，包括体系持续改进的过程以及保证符合顾客与适用的法律法规要求，旨在增进顾客满意。

注：在本标准中，术语“产品”仅适用于预期提供给顾客或顾客所要求的产品。

### 1.2 应用

本标准规定的所有要求是通用的，旨在适用于各种类型、不同规模和提供不同产品的组织。

当本标准的任何要求因组织及其产品的特点不适用时，可以考虑对其进行删减。

除非删减仅限于本标准第 7 章中那些不影响组织提供满足顾客和适用法律法规要求的产品的能力或责任的要求，否则不能声称符合本标准。

## 2 引用标准

下列标准所包含的条文，通过在本标准中引用而构成本标准的条文。本标准出版时，所示版本均为有效。所有标准都会被修订，使用本标准的各方应探讨使用下列标准最新版本的可能性。

GB/T19000-2000 质量管理体系 基础和术语 (idt ISO9000:2000)

## 3 术语和定义

本标准采用 GB/T19000 中的术语和定义。

本标准描述供应链所使用的以下术语经过了更改，以反映当前的使用情况：

供方→组织→顾客

本标准中的术语“组织”用以取代 GB/T19001-1994 所使用的术语“供方”，术语“供方”用以取代术语“分承包方”。

本标准中所出现的术语“产品”，也可指“服务”。

## 4 质量管理体系

### 4.1 总要求

组织应按本标准的要求建立质量管理体系，形成文件，加以实施和保持，并持续改进

其有效性。

组织应：

- a) 识别质量管理体系所需的过程及其在组织中的应用（见 1.2）；
- b) 确定这些过程的顺序和相互作用；
- c) 确定为确保这些过程的有效运作和控制所需的准则和方法；
- d) 确保可以获得必要的资源和信息，以支持这些过程的运作和监视；
- e) 监视、测量和分析这些过程；
- f) 实施必要的措施，以实现对这些过程所策划的结果和对这些过程的持续改进。

组织应按本标准的要求管理这些过程。

针对组织所选择的任何影响产品符合要求的外包过程，组织应确保对其实施控制。对此类外包过程的控制应在质量管理体系中加以识别。

注：上述质量管理体系所需的过程应当包括与管理活动、资源提供、产品实现和测量有关的过程。

## 4.2 文件要求

### 4.2.1 总则

质量管理体系文件应包括：

- a) 形成文件的质量方针和质量目标；
- b) 质量手册；
- c) 本标准所要求的形成文件的程序；
- d) 组织为确保其过程有效策划、运作和控制所需的文件；
- e) 本标准所要求的记录（见 4.2.4）。

注：

本标准出现“形成文件的程序”之处，即要求建立该程序，形成文件，并加以实施和保持。

不同组织的质量管理体系文件的多少与详略程度取决于：

- a) 组织的规模和活动的类型；
- b) 过程及其相互作用的复杂程度；
- c) 人员的能力。

文件可采用任何形式或类型的媒体。

### 4.2.2 质量手册

组织应编制和保持质量手册，质量手册包括：

- a) 质量管理体系的范围，包括任何删减的细节与合理性（见 1.2）；
- b) 为质量管理体系编制的形成文件的程序或对其引用；
- c) 质量管理体系过程之间的相互作用的表述。

### 4.2.3 文件控制

质量管理体系所要求的文件应予以控制。记录是一种特殊类型的文件，应依据 4.2.4 的要求进行控制。

应编制形成文件的程序，以规定以下方面所需的控制：

- a) 文件发布前得到批准，以确保文件是充分与适宜的；

- b) 必要时对文件进行评审与更新，并再次批准；
- c) 确保文件的更改和现行修订状态得到识别；
- d) 确保在使用处可获得适用文件的有关版本；
- e) 确保文件保持清晰、易于识别；
- f) 确保外来文件得到识别，并控制其分发；
- g) 防止作废文件的非预期使用，若因任何原因而保留作废文件时，对这些文件进行适当的标识。

#### 4.2.4 记录控制

应建立并保持记录，以提供符合要求和质量管理体系有效运行的证据。记录应保持清晰、易于识别和检索。应编制形成文件的程序，以规定记录的标识、储存、保护、检索、保存期限和处置所需的控制。

### 5 管理职责

#### 5.1 管理承诺

最高管理者应通过以下活动，对其建立、实施质量管理体系并持续改进其有效性的承诺提供证据：

- a) 向组织传达满足顾客和法律法规要求的重要性；
- b) 制定质量方针；
- c) 确保质量目标的制定；
- d) 进行管理评审；
- e) 确保资源的获得。

#### 5.2 以顾客为关注焦点

最高管理者应以增进顾客满意为目的，确保顾客的要求得到确定并予以满足（见 7.2.1 和 8.2.1）。

#### 5.3 质量方针

最高管理者应确保质量方针：

- a) 与组织的宗旨相适应；
- b) 包括对满足要求和持续改进质量管理体系有效性的承诺；
- c) 提供制定和评审质量目标的框架；
- d) 在组织内得到沟通和理解；
- e) 在持续适宜性方面得到评审。

#### 5.4 策划

##### 5.4.1 质量目标

最高管理者应确保在组织的相关职能和层次上建立质量目标，质量目标包括满足产品要求所需的内容（见 7.1 a）。质量目标应是可测量的，并与质量方针保持一致。

##### 5.4.2 质量管理体系策划

最高管理者应确保：

- a) 对质量管理体系进行策划，以满足质量目标以及 4.1 的要求。

b) 在对质量管理体系的变更进行策划和实施时，保持质量管理体系的完整性。

## 5.5 职责、权限和沟通

### 5.5.1 职责和权限

最高管理者应确保组织内的职责、权限得到规定和沟通。

### 5.5.2 管理者代表

最高管理者应指定一名管理者，无论该成员在其他方面的职责如何，应具有以下方面的职责和权限：

- a) 确保质量管理体系所需的过程得到建立、实施和保持；
- b) 向最高管理者报告质量管理体系的业绩和任何改进的需求；
- c) 确保在整个组织内提高满足顾客要求的意识。

注：管理者代表的职责可包括与质量管理体系有关事宜的外部联络。

### 5.5.3 内部沟通

最高管理者应确保在组织内建立适当的沟通过程，并确保对质量管理体系的有效性进行沟通。

## 5.6 管理评审

### 5.6.1 总则

最高管理者应按策划的时间间隔评审质量管理体系，以确保其持续的适宜性、充分性和有效性。评审应包括评价质量管理体系改进的机会和变更的需要，包括质量方针和质量目标。

应保持管理评审的记录（见 4.2.4）。

### 5.6.2 评审输入

管理评审的输入应包括以下方面的信息：

- a) 审核结果；
- b) 顾客反馈；
- c) 过程的业绩和产品的符合性；
- d) 预防和纠正措施的状况；
- e) 以往管理评审的跟踪措施；
- f) 可能影响质量管理体系的变更；
- g) 改进的建议。

### 5.6.3 评审输出

管理评审的输出应包括与以下方面有关的任何决定和措施：

- a) 质量管理体系及其过程有效性的改进；
- b) 与顾客要求有关的产品的改进；
- c) 资源需求。

## 6 资源管理

### 6.1 资源的提供

组织应确定并提供以下方面所需的资源：

- a) 实施、保持质量管理体系并持续改进其有效性;
- b) 通过满足顾客要求, 增进顾客满意。

## 6.2 人力资源

### 6.2.1 总则

基于适当的教育、培训、技能和经验, 从事影响产品质量工作的人员应是能够胜任的。

### 6.2.2 能力、意识和培训

组织应:

- a) 确定从事影响产品质量工作的人员所必要的能力;
- b) 提供培训或采取其他措施以满足这些需求;
- c) 评价所采取措施的有效性;
- d) 确保员工认识到所从事活动的相关性和重要性, 以及如何为实现质量目标作出

贡献;

- e) 保持教育、培训、技能和经验的适当记录 (见 4.2.4)。

## 6.3 基础设施

组织应确定、提供并维护为达到产品符合要求所需的基础设施。适用时, 基础设施包括:

- a) 建筑物、工作场所和相关的设施;
- b) 过程设备 (硬件和软件);
- c) 支持性服务 (如运输或通讯)。

## 6.4 工作环境

组织应确定和管理为达到产品符合要求所需的工作环境。

# 7 产品实现

## 7.1 产品实现的策划

组织应策划和开发产品实现所需的过程。产品实现的策划应与质量管理体系其他过程的要求相一致 (见 4.1)。

在对产品实现进行策划时, 组织应确定以下方面的适当内容:

- a) 产品的质量目标和要求;
- b) 针对产品确定过程、文件和资源的需求;
- c) 产品所要求的验证、确认、监视、检验和试验活动, 以及产品接收准则;
- d) 为实现过程及其产品满足要求提供证据所需的记录。

策划的输出形式应适于组织的运作方式。

注:

对应用于特定产品、项目或合同的质量管理体系的过程 (包括产品实现过程) 和资源作出规定的文件可称之为质量计划。

组织也可将 7.3 的要求应用于产品实现过程的开发。

## 7.2 与顾客有关的过程

### 7.2.1 与产品有关的要求的确定

组织应确定:

- a) 顾客规定的要求, 包括对交付及交付后活动的要求;
- b) 顾客虽然没有明示, 但规定的用途或已知的预期用途所必需的要求;
- c) 与产品有关的法律法规要求;
- d) 组织确定的任何附加要求。

#### 7.2.2 与产品有关的要求的评审

组织应评审与产品有关的要求。评审应在组织向顾客作出提供产品的承诺之前进行(如提交标书、接受合同或订单及接受合同或订单的更改), 并应确保:

- a) 产品要求得到规定;
- b) 与以前表述不一致的合同或订单的要求已予解决;
- c) 组织有能力满足规定的要求。

评审结果及评审所引起的措施的记录应予保持(见 4.2.4)。

若顾客提供的要求没有形成文件, 组织在接收顾客要求前应对顾客要求进行确认。

若产品要求发生变更, 组织应确保相关文件得到修改, 并确保相关人员知道已变更的要求。

注: 在某些情况中, 如网上销售, 对每一个订单进行正式的评审可能是不实际的。而代之对有关的产品信息, 如产品目录、产品广告内容等进行评审。

#### 7.2.3 顾客沟通

组织应对以下有关方面确定并实施与顾客沟通的有效安排:

- a) 产品信息;
- b) 问询、合同或订单的处理, 包括对其的修改;
- c) 顾客反馈, 包括顾客抱怨。

### 7.3 设计和开发

#### 7.3.1 设计和开发策划

组织应对产品的设计和开发进行策划和控制。

在进行设计和开发策划时, 组织应确定:

- a) 设计和开发阶段;
- b) 适于每个设计和开发阶段的评审、验证和确认活动;
- c) 设计和开发的职责和权限。

组织应对参与设计和开发的不同小组之间的接口实施管理, 以确保有效的沟通, 并明确职责分工。

随设计和开发的进展, 在适当时, 策划的输出应予以更新。

#### 7.3.2 设计和开发输入

应确定与产品要求有关的输入, 并保持记录(见 4.2.4)。这些输入应包括:

- a) 功能和性能要求;
- b) 适用的法律法规要求;
- c) 适用时, 以前类似设计提供的信息;
- d) 设计和开发所必需的其他要求。



应对这些输入进行评审,以确保其充分性与适宜性。要求应完整、清楚,并且不能自相矛盾。

### 7.3.3 设计和开发输出

设计和开发的输出应以能够针对设计和开发的输入进行验证的方式提出,并应在放行前得到批准。

设计和开发输出应:

- a) 满足设计和开发输入的要求;
- b) 给出采购、生产和服务提供的适当信息;
- c) 包含或引用产品接收准则;
- d) 规定对产品的安全和正常使用所必需的产品特性。

### 7.3.4 设计和开发评审

在适宜的阶段,应依据所策划的安排(见 7.3.1)对设计和开发进行系统的评审,以使:

- a) 评价设计和开发的结果满足要求的能力;
- b) 识别任何问题并提出必要的措施。

评审的参加者应包括与所评审的设计和开发阶段有关的职能的代表。评审结果及任何必要措施的记录应予以保持(见 4.2.4)。

### 7.3.5 设计和开发验证

为确保设计和开发输出满足输入的要求,应依据所策划的安排(见 7.3.1)对设计和开发进行验证。验证结果及任何必要措施的记录应予以保持(见 4.2.4)。

### 7.3.6 设计和开发确认

为确保产品能够满足规定的使用要求或已知的预期用途的要求,应依据所策划的安排(见 7.3.1)对设计和开发进行确认。只要可行,确认应在产品交付或实施之前完成。确认结果及任何必要措施的记录应予以保持(见 4.2.4)。

### 7.3.7 设计和开发更改的控制

应识别设计和开发的更改,并保持记录。在适当时,应对设计和开发的更改进行评审、验证和确认,并在实施前得到批准。设计和开发更改的评审应包括评价更改对产品组成部分和已交付产品的影响。

更改评审结果及任何必要措施的记录应予以保持(见 4.2.4)。

## 7.4 采购

### 7.4.1 采购过程

组织应确保采购的产品符合规定的采购要求。对供方及采购的产品控制的类型和程度应取决于采购的产品对随后的产品实现或最终产品的影响。

组织应根据供方按组织的要求提供产品的能力评价和选择供方。应制定选择、评价和重新评价的准则。评价结果及评价所引起的任何必要措施的记录应予以保持(见 4.2.4)。

### 7.4.2 采购信息

采购信息应表述拟采购的产品,适当时包括:

- a) 产品、程序、过程 and 设备的批准要求;
- b) 人员资格的要求;

c) 质量管理体系的要求。

在与供方沟通前，组织应确保规定的采购要求是充分与适宜的。

#### 7.4.3 采购产品的验证

组织应确定并实施检验或其他必要的活动，以确保采购的产品满足规定的采购要求。

当组织或其顾客拟在供方的现场实施验证时，组织应在采购信息中对拟验证的安排和产品放行的方法做出规定。

### 7.5 生产和服务提供

#### 7.5.1 生产和服务提供的控制

组织应策划并在受控条件下进行生产和服务提供。适用时，受控条件应包括：

- a) 获得表述产品特性的信息；
- b) 必要时，获得作业指导书；
- c) 使用适宜的设备；
- d) 获得和使用监视和测量装置；
- e) 实施监视和测量；
- f) 放行、交付和交付后活动的实施。

#### 7.5.2 生产和服务提供过程的确认

当生产和服务提供过程的输出不能由后续的监视或测量加以验证时，组织应对任何这样的过程实施确认。这包括仅在产品使用或服务已交付之后问题才显现的过程。

确认应证实这些过程实现所策划的结果的能力。

组织应规定确认这些过程的安排，适用时包括：

- a) 为过程的评审和批准所规定的准则；
- b) 设备的认可和人员资格的鉴定；
- c) 使用特定的方法和程序；
- d) 记录的要求（见 4.2.4）；
- e) 再确认。

#### 7.5.3 标识和可追溯性

适当时，组织应在产品实现的全过程中使用适宜的方法识别产品。

组织应针对监视和测量要求识别产品的状态。

在有可追溯性要求的场合，组织应控制并记录产品的惟一性标识（见 4.2.4）。

注：在某些行业，技术状态管理是保持标识和可追溯性的一种方法。

#### 7.5.4 顾客财产

组织应爱护在组织控制下或组织使用的顾客财产。组织应识别、验证、保护和维持供其使用或构成产品一部分的顾客财产。若顾客财产发生丢失、损坏或发现不适用的情况时，应报告顾客，并保持记录（见 4.2.4）。

注：顾客财产可包括知识产权。

#### 7.5.5 产品防护

在内部处理和交付到预定的地点期间，组织应针对产品的符合性提供防护，这种防护应包括标识、搬运、包装、贮存和保护。防护也应适用于产品的组成部分。

## 7.6 监视和测量装置的控制

组织应确定需实施的监视和测量以及所需的监视和测量装置，为产品符合确定的要求（见 7.2.1）提供证据。

组织应建立过程，以确保监视和测量活动可行并以与监视和测量的要求相一致的方式实施。

当有必要确保结果有效的场合时，测量设备应：

a) 对照能溯源到国际或国家标准的测量标准，按照规定的时间间隔或使用前进行校准或检定。当不存在上述标准时，应记录校准或检定的依据；

b) 必要时进行调整或再调整；

c) 得到识别，以确定其校准状态；

d) 防止可能使测量结果失效的调整；

e) 在搬运、维护和储存期间防止损坏或失效；

此外，当发现设备不符合要求时，组织应对以往测量结果的有效性进行评价和记录。组织应对该设备和任何受影响的产品采取适当的措施。校准和验证结果的记录应予以保持（见 4.2.4）。

当计算机软件用于规定要求的监视和测量时，应确认其满足预期用途的能力。确认应在初次使用前进行，并在必要时予以重新确认。

注：作为指南，参见 GB/T19022.1 和 GB/T19022.2。

## 8 测量、分析和改进

### 8.1 总则

组织应策划并实施以下方面所需的监视、测量、分析和改进过程：

a) 证实产品的符合性；

b) 确保质量管理体系的符合性；

c) 持续改进质量管理体系的有效性。

这应包括对统计技术在内的适用方法及其应用程度的确定。

### 8.2 监视和测量

#### 8.2.1 顾客满意

作为对质量管理体系业绩的一种测量，组织应监视顾客关于组织是否满足其要求的感受的相关信息，并确定获取和利用这种信息的方法。

#### 8.2.2 内部审核

组织应按策划的时间间隔进行内部审核，以确定质量管理体系是否：

a) 符合策划的安排（见 7.1）、本标准的要求以及组织所确定的质量管理体系的要求；

b) 得到有效实施与保持。

考虑拟审核的过程和区域的状况和重要性以及以往审核的结果，组织应对审核方案进行策划。应规定审核的准则、范围、频次和方法。审核员的选择和审核的实施应确保审核过程的客观性和公正性。审核员不应审核自己的工作。

策划和实施审核以及报告结果和保持记录（见 4.2.4）的职责和要求应在形成文件的程

序中做出规定。

负责受审区域的管理者应确保及时采取措施，以消除所发现的不合格及其原因。跟踪活动应包括对所采取措施的验证和验证结果的报告（见 8.5.2）。

注：作为指南，参见 GB/T19021.1、GB/T19021.2 及 GB/T19021.3。

### 8.2.3 过程的监视和测量

组织应采用适宜的方法对质量管理体系过程进行监视，并在适用时进行测量。这些方法应证实过程实现所策划的结果的能力。当未能达到所策划的结果时，应采取适当的纠正和纠正措施，以确保产品的符合性。

### 8.2.4 产品的监视和测量

组织应对产品的特性进行监视和测量，以验证产品要求已得到满足。这种监视和测量应依据所策划的安排（见 7.1），在产品实现过程的适当阶段进行。

应保持符合接收准则的证据。记录应指明有权放行产品的人员（见 4.2.4）。

除非得到有关授权人员的批准，适用时得到顾客的批准，否则在策划的安排（见 7.1）已圆满完成之前，不能放行产品和交付服务。

## 8.3 不合格品控制

组织应确保不符合产品要求的产品得到识别和控制，以防止其非预期的使用或交付。不合格品控制以及不合格品处置的有关职责和权限应在形成文件的程序中作出规定。

组织应通过下列一种或几种途径，处置不合格品：

- a) 采取措施，消除发现的不合格；
- b) 经有关授权人员批准，适用时经顾客批准，让步使用、放行或接收不合格品；
- c) 采取措施，防止其原预期的使用或应用。

应保持不合格的性质以及随后所采取的任何措施的记录，包括所批准的让步的记录（4.2.4）。

应对纠正后的产品再次进行验证，以证实符合要求。

当在交付或开始使用后发现产品不合格时，组织应采取与不合格的影响或潜在影响的程度相适应的措施。

## 8.4 数据分析

组织应确定、收集和分析适当的数据，以证实质量管理体系的适宜性和有效性，并评价在何处可以持续改进质量管理体系的有效性。这应包括来自监视和测量的结果以及其他有关来源的数据。

数据分析应提供有关以下方面的信息：

- a) 顾客满意（见 8.2.1）；
- b) 与产品要求的符合性（见 7.2.1）；
- c) 过程和产品的特性及趋势，包括采取预防措施的机会；
- d) 供方。

## 8.5 改进

### 8.5.1 持续改进

组织应利用质量方针、质量目标、审核结果、数据分析、纠正和预防措施以及管理评

审，持续改进质量管理体系的有效性。

#### 8.5.2 纠正措施

组织应采取措施，以消除不合格的原因，防止不合格的再发生。纠正措施应与所遇到不合格的影响程度相适应。

应编制形成文件的程序，以规定以下方面的要求：

- a) 评审不合格（包括顾客抱怨）；
- b) 确定不合格的原因；
- c) 评价确保不合格不再发生的措施的需求；
- d) 确定和实施所需的措施；
- e) 记录所采取措施的结果（见 4.2.4）；
- f) 评审所采取的纠正措施。

#### 8.5.3 预防措施

组织应确定措施，以消除潜在不合格的原因，防止不合格的发生。预防措施应与潜在问题的影响程度相适应。

应编制形成文件的程序，以规定以下方面的要求：

- a) 确定潜在不合格及其原因；
- b) 评价防止不合格发生的措施的需求；
- c) 确定并实施所需的措施；
- d) 记录所采取措施的结果（见 4.2.4）；
- e) 评审所采取的预防措施。

# 附录 C 信息技术——软件包质量 要求和测试

Information technology—Software packages Quality requirements testing  
(国家标准 GB/T17544—1998, Idt ISO/IEC12119:1994)

---

## 1 范围

本标准适用于软件包。例如文本处理程序、电子表格、数据库程序、图形软件包、技术或科学函数计算程序以及实用程序。

它规定了：

——软件包要求（质量要求）；

——针对这些要求，如何对软件包进行测试的细则（测试细则，特别是第三方测试）。

它只涉及要提供的或要交付的软件包，不涉及它们的产生过程（包括活动和中间产品，如规格说明）。供方的质量体系超出了本标准的范围。

注：某特种软件需要附加的要求，如安全要求高的软件。

本标准期望的用户如下：

a) 在下述情况下使用本标准的供方：

- 1) 规定软件包的要求时；
- 2) 设计描述产品的格式时；
- 3) 评价他们自己的产品时；
- 4) 发布符合[ISO / IEC 第 22 号导则]的声明时；
- 5) 申请合格[ISO / IEC 第 23 号导则]证书或标志时。

b) 希望建立第三方认证模式（国际的、地区的及国家的）[ISO / IEC 第 16、28 和 44 号导则]的认证机构；

c) 为合格证书或标志而进行测试的测试实验室，测试实验室必须遵循测试指令 [ISO / IEC 第 25 号导则]；

d) 认可认证机构和测试实验室的认可机构 [ISO / IEC 第 40 和 58 号导则]；

e) 评价测试实验室能力的实验室审核员[ISO / IEC 第 58 号导则]；

f) 购买者：

- 1) 用本标准规定的内容来比较他们的要求；
- 2) 用现有产品的产品描述中的信息来比较期望的工作任务的要求；
- 3) 寻求已认证的产品；

- 4) 此外, 检验要求是否被满足。
- g) 用户: 可以从更好的产品获益。

## 2 定义

本标准采用下列定义。源自其他标准的定义列于附录 A 以便于引用。

### 2.1 功能 (function)

程序中的一个算法的实现。利用该实现, 用户或程序可以执行某一工作任务的全部或部分内容。

注:

- ① 对于用户人说, 功能不一定是能访问的 (如数据的自动备份或存储)。
- ② 这里功能的概念比 GB / T 5271.14 (失效、故障、维护和可靠性的描述中) 使用的功能概念要窄, 但比 GB / T 5271.2 (算术和逻辑运算) 和 GB / T 5272.15 (程序设计语言) 中定义的要宽。

### 2.2 需求文档 (requirements document)

包含由软件包满足的建议、要求或规则的任何组合的文档。

注: 例子有技术或人类工效标准, 来自某一组织 (如市场部、技术或用户协会) 的需求列表 (或模型的需求规格说明), 法律或法令。

### 2.3 产品描述 (product description)

陈述软件包性质的文档, 其主要目的是帮助潜在的购买者在购买前对产品进行适用性评价。

注: 该术语比 GB / T 5271.20 中的术语“系统描述”更具体。产品描述的目的包括 ISO9127 中“覆盖信息”的目的。

产品描述不是规格说明, 但它可用于不同的用途。

### 2.4 用户文档 (user documentation)

以打印的或非打印形式得到的文档的完整集合, 用户文档的提供有利于产品的应用并且是产品的必备部分。

### 2.5 包文档 (package documentation)

产品描述和用户文档。

### 2.6 测试用例 (test case)

测试者使用的文档化的细则, 其规定如何对某项功能或功能组合进行测试。测试用例包括下列内容的详细信息:

- 测试目标;
- 要测试的功能;
- 测试环境和其他条件 (配置细节和准备工作);
- 测试数据;
- 过程;
- 系统的预期行为。

### 2.7 维护 (maintenance)

是系统维护的一部分 (见 A5.2), 其涉及软件包的修改。

### 3 质量要求

#### 3.1~3.3 包含

——每个软件包要有产品描述和用户文档的要求；

——产品描述的要求，尤其应包含规定信息，并且其所有要求的内容是可测试的、正确的；

——用户文档的要求；

——包含在软件包中的程序要求和数据要求。

注：

① 关于用户文档、程序和数据的要求包含许多一般要求（独立于产品描述中的约定），但不包括用户希望的所有性质。

② 某些性质，例如，用户文档和程序消息的“时期解件”和“易于浏览”，按用户的观点这些性质是被公认的。然而，由于这些性质难于清晰地测试且结果难于再现，使得这些性质目前仅作为建议来规定。

③ 3.1~3.3 中的要求按 GB/T16260 中出现的特性次序来安排。

如果一个软件包遵循 3.1~3.3 中所有的要求，则该软件包符合本标准。建议是可选的（通过使用“宜”这个词来表示）。

注：要证明一个产品对 3.1~3.3 中的要求的符合性可能是困难的或不可能的。但是，根据 ISO / IEC 第 2 号导则，为得合格证书，按照第 4 章来测试（包括文档的评审）被认为是足够了，并且不需要形式证明。

#### 3.1 产品描述

每个软件包应有一个产品描述。

产品描述定义产品。产品描述是产品软件包文档的一部分，它提供关于用户文档、程序以及数据（如果有的话）的信息。

产品描述的主要目的是：

——帮助用户或潜在的购买者作出产品是否适用于他们的评价。从这一意义说，产品描述也是销售信息；

——作为测试的基础（见第 4 章）。

对产品感兴趣的人们可获得产品描述。

##### 3.1.1 内容的一般要求

产品描述宜是充分可理解的、完整的并且易于浏览，以帮助潜在的购买者在购买该产品前评价产品对他们自己的适用性。

产品描述应避免不一致，每个术语在任何地方都应有相同的意义。

产品描述的说明应是可测试的并且是正确的。

注：如果有外部需求文档（见 3.1.2e），本条要求引用外部需求文档中的说明。

下列 3.1.2~3.1.8 规定产品描述应包含或它包含的内容，它也可包含产品的附加说明。

##### 3.1.2 标识和批示

###### a) 产品描述的标识

产品描述应且有惟一的文档标识。它可以有不同于产品描述的命名，例如，“功能描



述”、“产品信息”“产品清单”。

b) 产品的标识

产品描述应标识产品。产品标识应至少有产品名字和版本号或日期。如果在产品描述中提及两个或多个派生版本，则每个版本应至少有产品名、派生版本名和版本号或日期。

c) 供方

产品描述应至少包含一个供方的名字和地址。

注：名字和地址不必打印；有供方的公章即可。

d) 工作任务

产品描述应标识期望的产品能完成工作任务。

e) 符合需求文档

产品描述可以引用产品应符合的需求文档的内容。在这种情况下应标识相关的编辑版本。

f) 要求的系统

应标识将产品投入使用所要求的系统（硬件、软件及其配置）包括制造厂商名和所有部件的类型标识符，例如：

- 包括协处理器的处理单元；
- 主存规格；
- 外存的类型和规格；
- 扩展卡；
- 输入和输出设备；
- 网络环境；
- 系统软件和其他软件。

对于不同的工作任务、不同的边界值或不同的效率要求，可以规定不同的要求系统。

如果先前特定的硬件或软件产品已经标识，则语句“(如果兼容，或任何其他……)”可以出现在产品描述中。如果产品的先前版本已经标识，则语句“如果兼容，或升级的版本”可以出现在产品描述中。语句“自版本 X 至少到版本 Y”可以出现在产品描述中，而“自版本 X”不能出现在产品描述中。

注：由于版本 X+3 的出现，语句“自版本 X”将变得不正确，因为对于版本 X+3，软件包操作将会失败。

g) 与其他产品的接口

如果产品描述引用了其他产品接口，则应对所引用的接口或产品进行标识。

h) 要交付项

应对要提供的产品的每个物理部件进行标识，特别是所有打印的文档和所有的数据媒体。

应说明提供的程序形式如源程序、目标模块，或加载模块。

注：媒体格式（如磁盘格式）不必指明，因为可能的格式集合是由要求的系统决定的（见 3.1.2f）。

i) 安装

应说明产品安装是否能由用户来完成。

j) 支持

应说明是否提供对产品操作的支持。

k) 维护

应说明是否提供维护。如果提供维护，应说明具体包括什么。

### 3.1.3 功能说明

a) 功能概述

产品描述应概述产品的用户可调用功能、需要的数据、所提供的设施。

对每个所论及的功能（尤其是选项和变量）是否是下列内容的一部分应清晰地说明：

- 产品功能的；
- 在产品描述中完整描述的产品扩展功能的；
- 在产品描述中所引用的产品扩展功能的；
- 无保证的补充功能的。

注：不必论及每个用户可调用功能，不必给出功能如何调用的每个细节。

b) 边界值

如果由于产品特定的边界值致使产品的使用受限，则应提供这些边界值。例如：

- 最小或最大值；
- 键的长度；
- 文卷中的记录的最大数目；
- 检索准则的最大数目；
- 最小样本大小。

当不可能提供固定的边界值时（例如，边界值取决于应用问题的类型或输入数据时），则应说明这些限制，可以提供允许的值组合，更具体的信息写入用户文档。

c) 安全

如果提供的话，产品描述中应包含有关防止对程序或数据非授权的无意访问或蓄意访问的手段。

### 3.1.4 可靠性说明

产品描述应包含数据存储规程的信息。

注：例如，只要说明使用操作系统进行备份就可以了。

应描述保证产品的功能能力的附加性质。例如：

- 检验输入的合理性；
- 防止由于用户的错误而产生的严重后果；
- 出错恢复。

### 3.1.5 易用性说明

a) 用户界面

应命名用户界面的类型，例如：命令行、菜单、窗口、功能键及帮助功能。

b) 要求的知识

应规定应用该产品所要求的专门知识。例如：

- 技术领域的知识；

- 操作系统的知识;
- 经过专门培训可获得的知识;
- 除了已写入产品描述中以外的其他语言知识。

应说明用户文档和用户界面（括出错信息和可视数据）所使用的所有自然语言，软件包本身和该产品描述中所涉及的所有其他产品的有关内容都应加以说明。

注：这种要求超出了 ISO9127: 1988 的 6.1.7 的规定，在那里，关于所用的语言规定是可选的。

#### c) 适应用户的需要

如果产品能被用户作适应性修改，则应标识这种修改的工具和修改工具的使用的条件。

例如：

- 参数的改变;
- 计算的算法改变;
- 功能键的分配。

#### d) 防止侵权行为

如果防止侵权的技术保护可能有碍于软件的使用，则应说明这种保护，例如：

- 防止复制的技术保护;
- 程序设置的使用截止日期;
- 相互约定的付费复制。

#### e) 使用效率和用户满意度

产品描述可以包括关于使用效率和用户满意度的数据。

注：这样的数据时遵循 ISO9241—11 的指南。

### 3.1.6 效率说明

产品描述可以包含产品的时间行为的数据，诸如在指定条件下（例如系统配置和负载分布）关于给定功能的响应时间和吞吐率。

#### 3.1.7 可维护性说明

产品描述可包含可维护性说明。

#### 3.1.8 可移植性说明

产品描述可包含可移植性说明。

## 3.2 用户文档

### 3.2.1 完整性

用户文档应包含产品使用所需信息。在产品描述中说明的所有功能以及在程序中用户可调用的所有功能，都应在用户文档中加以完整地描述。

用户文档中应再次说明产品描述中给出的所有边界值。

如果安装能由用户来完成，则用户文档应包括安装手册，该手册应包含所有必要的信息（见 3.3.1a）。安装手册宜说明一次安装的最小文卷和最大文卷。

如果维护能由用户来完成，则用户文档应包括程序维护手册，该手册应包含各种有关该软件维护所需要的信息。

### 3.2.2 正确性

用户文档中所有信息应是正确的，不能有歧义和错误的表达。

### 3.2.3 一致性

则用户文档自身内容或相互之间以及与产品描述之间都不应相互矛盾。每个术语的含义宜处处保持一致。

注：程序和数据的一致性在 3.3.1 d 中论及。

### 3.2.4 易理解性

用户文档对于正常执行其工作任务的一般用户宜是易理解的，例如，通过使用适当的术语、图形表示，详细的解释以及引出有用的信息源来表现。

### 3.2.5 易浏览性

用户文档宜易于浏览，以使相互关系明确。

每个文档应有目录表和索引表。

如果文档未提供印刷本，则应指明打印过程。

## 3.3 程序和数据

### 3.3.1 功能性

#### a) 安装

如用安装能由用户来完成，则按照安装手册中的信息应能成功安装。产品描述中指出的每种所要求的系统对于程序的安装应是充分的。

安装之后，程序能否运行应是可鉴别的。例如，使用提供的测试用例或通过相应信息的自检。

#### b) 功能表现

用户文档中提到的所有功能应是可执行的。程序应按照用户文档中的给定形式，在规定的边界值范围内使用相应的设施、性质和数据执行其功能。

注：由于在产品描述中涉及的所有功能也应出现在用户文档中，这些功能更应是可执行的。

#### c) 正确性

程序和数据应与产品描述及用户文档中的全部说明相对应。为完成工作任务，程序功能应以正确的方式执行。特别是，程序和数据应符合产品描述所引用的任一需求文档中的全部需求。

#### d) 一致性

程序和数据其本身不能自相矛盾，并且同产品描述和用户文档不能相互矛盾。每个术语应处处具有相同的含义。

由用户先例的程序操作控制和程序行为（例如，消息，屏幕输入格式和打印报表）宜有一致的结构。

### 3.3.2 可靠性

系统（包括硬件、要求的软件及属于该产品的程序）不应陷入用户无法控制的状态，既不应崩溃也不应丢失数据。

即使在下列情况下也应满足上述要求：

- 使用的容量到达规定的极限；
- 企图使用的容量超出规定的极限；
- 由产品描述中列出的其他程序或用户造成的错误输入；

——用户文档中明确规定的非法指令。

只是那些不能用任何程序捕获的硬中断和操作系统中断（例如，系统操作复位用的键或组合键）不在此范围之内。

当程序认为输入错误或输入未定义时，应视为不允许的输入，不加处理。

### 3.3.3 易用性

关于易用性，根据本标准的规定，鼓励研究 ISO9241 系列标准最新版本应用的可能性。

注：特别是宜考虑 ISO9241 系列的第 10 部分和第 13 部分。

#### a) 易理解性

程序的问题、消息和结果应是易理解的，例如：

- 通过选择适当的术语；
- 通过图形表示；
- 通过提供背景信息；
- 通过帮助功能的解释。

出错消息应提供解释相应差错产生原因和纠正的详细信息（例如通过引用用户文档的条文）。

#### b) 易浏览性

如果有多种媒体，则每种数据媒体应具有产品标识、可辨别编号或文本。

对于使用程序进行工作的用户，总能找到哪个功能正在被执行是可能的。

程序宜以易观察易读的形式向用户提供信息。通过对信息的适当编码和分组对用户提供指导，必要时，程序可向用户发出警报。

源程序的消息应如此设计，即用户通过类型容易区分它们。例如：

- 确认；
- 程序询问；
- 警告；
- 出错消息。

屏幕输入格式，报表和其他输入、输出宜设计清晰和易于浏览。一般包括：

- 字母数字字段左对齐；
- 数字字段右对齐；
- 在表中，小数点或逗号要排在同一垂直线上；
- 字段界限是可识别的；
- 哪些字段的使用是受限的，哪些字段是可识别的；
- 标识输入失败后要立即在屏幕输入格式中加亮；
- 通过一个可视或可听的信号来引起用户注意屏幕内容的改变。

#### c) 可操作性

具有严惩后果的功能执行应是可逆的，或者程序应给出该后果的明显警告并且在执行该命令前要求确认。特别是数据的删除和重写，以及中断一个过长的处理操作，这种动作往往有严重后果。

如果文档文本编制是以对话形式提供。用户应直接访问该文本的子条文，例如通过目

录表显示的选择和按关键字检索功能来实现。

#### 3.3.4 效率

应遵循产品描述中的效率说明。

#### 3.3.5 可维护性

应遵循产品描述中的可维护性说明。

#### 3.3.6 可移植性

应遵循产品描述中的可移植性说明。

### 4 测试细则

4.1~4.5 的细则规定如何按照质量要求来测试产品。包括根据所有符合性产品要求的性质测试和按照产品描述约定的性质测试。包括通过文档的检查测试和程序及数据的黑盒测试。

这些细则描述了功能测试（黑盒测试），不包括结构测试，因为结构测试需要得到源代码。

产品仅在它要求的系统中被测试。对于计算机工作时的人类工效评价，本标准不作考虑。

注：

① 这些细则上要是根据某些认证模式，针对第三方测试（见第 1 章 C 项）。在生产过程中，这种测试比使用结构测试可能更经济且更有效。

② 第 4 章不包含关于软件包的要求（所有这些要求包含在第 3 章中）。一个软件包不按第 4 章进行测试可能是符合的，但是这样的测试无法发现个符合性的存在。

③ 当产品描述确定了要求的系统时，基于该要求的系统上的产品的任何不符合性被作为该产品的个符合性处理。

④ 认证模式可对照建议选项来进行测试。

⑤ 关于人类工效评价的指南包含在 ISO9241-11 中。

#### 4.1 测试预要求

##### 4.1.1 产品项的现场要求

对于要测试的软件包所有要交付的项（见 3.1.2h）以及产品描述（见 3.1.2e）中已标识的需求文档都应提供到测试现场。

##### 4.1.2 对系统组成部分的现场要求

对于软件包的测试，在产品描述中已指明要求的所有计算机系统的组成部分应提供到测试现场。

##### 4.1.3 培训

如果在产品描述中提到培训，则测试者应有机会使用培训材料和培训大纲。

#### 4.2 测试活动

产品描述、用户文档、程序和任何要交付的数据都作为软件包的组成部分，并且——应按第 3 章中的要求进行符合性测试，且——它按第 3 章中的建议进行符合性测试。

测试对象应源于并包括第 3 章中所有要求（完整性、一致性等）。

如果在产品描述中涉及到其他产品，只需针对该产品的产品描述中提出的要求对这些产品进行测试。

如果测试者做出下述判断时，则对产品描述中的细节，用户文档中的细节，功能中的细节和产品的数据中的细节不需要测试：

——这些细节对已指明的工作任务的影响可忽略。

这些不做测试的细节应在测试记录和测试报告中说明。对它们不做测试的理由应在测试记录中记录。

#### 4.2.1 产品描述

第 3 章中的要求的实现应被测试，并且第 3 章中的建议的实现宜被测试。

#### 4.2.2 用户文档

第 3 章中的要求的实现应被测试，并且第 3 章中的建议的实现宜被测试。

#### 4.2.3 程序和数据

第 3 章中的要求的实现应被测试，并且第 3 章中建议的实现宜被测试。

程序应在产品描述中提及的所有的计算机系统中进行测试。

如果存在若干不同的程序变量，每个都应测试。函数亦是如此，按照产品描述和用户文档进行测试，一组变量标识的函数按每个变量进行测试。应使用以产品描述和用户文档为基础构造的测试用例来测试提供的程序和数据。进一步的材料（例如源程序）不必考虑，除非在产品描述或用户文档中作了说明，才需要测试它们。

测试用例应有规则地系统地来构造。

注：依确定的方法进行随机测试是许可的。

如果例子是在用户文档中给出的，则它们应作为测试用例，但测试不应局限于这些例子。可以使出软件包供应方提供的测试用例，但测试不应局限于这些测试用例。

##### a) 安装

如果按照产品描述，用户能完成安装，则应测试这种安装，即程序是否像安装手册中描述的那样能成功地安装和测试。

否则应保证被安装程序的硬、软件环境符合于产品描述中说明的计算机系统。

##### b) 程序执行

测试用例应覆盖软件描述和用户文档中描述的所有功能，并且考虑有代表性的工作任务的功能组合。应针对所有的边界值（按照产品描述和用户文档）来测试程序，而这些边界值在要求的系统中提供。

在用户文档中明显地不赞成或声明禁用的输入或命令序列应属于测试范围。

#### 4.3 测试记录

每个测试记录应包含足够的信息以方便重复测试[ISO / IEC25 号导则]。测试记录应包括：

- 测试计划或包含测试用例（每个测试用例说明它的目标，见 2.6）的测试规格说明；
- 与测试用例相关的所有结果，包括在测试期间出现的所有失败；
- 测试中涉及的人员身份。

#### 4.4 测试报告

测试的对象和结果（如测试记录中记录的）应在测试报告中汇总。测试报告应具有如下结构：

1 产品标识；

2 用于测试的计算机系统（硬件、软件以及它们的配置）；

3 使用的文档（及其标识）；

4 产品描述、用户文档、程序和数据的测试结果；

5 与要求不符的清单；

6 针对建议的要求不符的清单，或者是不循建议要求的清单，或者针对建议要求产品未作符合性测试的说明；

7 测试结束日期。

测试报告的第 4 章（测试结果）应包括相应于 3.1~4.2 每个标题的说明。

另外，针对建议要求的符合性产品未作测试的说明，测试报告的第 6 章可提供观察到的不符合建议要求的清单。

测试报告的标识（测试实验室、产品标识、测试报告的日期）和面页总数应出现在测试报告的每页上。

测试报告应包括：

——仅与测试项相关的测试结果有效性的说明；

——未经测试实验室书面批准，不得复制报告（完整复制除外）的说明 [ISO / IEC25 号导则]。

测试报告宜遵循 ISO / IEC25 号导则有关测试报告的规定。

#### 4.5 跟踪测试

当某一产品已经测试过，再测试时（注意考虑先前的测试）：

——文档功能和数据中所有的改变部分都应测试，就像该产品是一新产品一样；

——受改变部分影响的或受要求的系统中的改变影响（根据测试者的专门知识）的所有来改变部分都应测试，就像该产品是一新产品一样；

——所有的其他部分应至少按样本进行测试。



# 附录 D 测试计划模板

<项目名称>

## 测试计划

### 修订历史记录

版本	日期	AMD	修订者	说明
1.0	XXXX 年 XX 月 XX			

( A-添加, M-修改, D-删除 )

# 目 录

1. 简介 .....	415
1.1 目的 .....	415
1.2 背景 .....	415
1.3 范围 .....	415
2. 测试参考文档和测试提交文档 .....	415
2.1 测试参考文档 .....	415
2.2 测试提交文档 .....	416
3. 测试进度 .....	416
4. 测试资源 .....	416
4.1 人力资源 .....	416
4.2 测试环境 .....	416
4.3 测试工具 .....	417
5. 系统风险、优先级 .....	417
6. 测试策略 .....	417
6.1 数据和数据库完整性测试 .....	417
6.2 接口测试 .....	418
6.3 集成测试 .....	418
6.4 功能测试 .....	418
6.5 用户界面测试 .....	419
6.6 性能评测 .....	419
6.7 容量测试 .....	420
6.8 安全性和访问控制测试 .....	421
6.9 故障转移和恢复测试 .....	422
6.10 配置测试 .....	423
6.11 安装测试 .....	423
7. 问题严重度描述 .....	424
8. 与测试有关的任务 .....	424

## 1. 简介

### 1.1 目的

<项目名称>的这一“测试计划”文档有助于实现以下目标:

[确定现有项目的信息和应测试的软件构件。

列出推荐的测试需求(高级需求)。

推荐可采用的测试策略,并对这些策略加以说明。

确定所需的资源,并对测试的工作量进行估计。

列出测试项目的可交付元素]

### 1.2 背景

[对测试对象(构件、应用程序、系统等)及其目标进行简要说明。需要包括的信息有:主要的功能和性能、测试对象的构架以及项目的简史。]

### 1.3 范围

[描述测试的各个阶段(例如,单元测试、集成测试或系统测试),并说明本计划所针对的测试类型(如功能测试或性能测试)。

简要地列出测试对象中将接受测试或将不接受测试的那些性能和功能。

如果在编写此文档的过程中做出的某些假设可能会影响测试设计、开发或实施,则列出所有这些假设。

列出可能会影响测试设计、开发或实施的所有约束、风险或意外事件]

## 2. 测试参考文档和测试提交文档

### 2.1 测试参考文档

下表列出了制定测试计划时所使用的文档,并标明了各文档的可用性:

[注:可适当地删除或添加文档项。]

文档(版本/日期)	已创建或可用	已被接收或已经过复审	作者或来源	备注
软件需求定义	是□ 否□	是□ 否□		
收件系统分析	是□ 否□	是□ 否□		
软件概要设计	是□ 否□	是□ 否□		
软件详细设计	是□ 否□	是□ 否□		
软件测试需求	是□ 否□	是□ 否□		
硬件需求定义	是□ 否□	是□ 否□		
硬件结构设计(包含PCB)	是□ 否□	是□ 否□		
硬件测试需求	是□ 否□	是□ 否□		

续表

文档（版本/日期）	已创建或可用	已被接收或已经过复审	作者或来源	备注
USB 驱动设计	是 <input type="checkbox"/> 否 <input type="checkbox"/>	是 <input type="checkbox"/> 否 <input type="checkbox"/>		
模块开发手册	是 <input type="checkbox"/> 否 <input type="checkbox"/>	是 <input type="checkbox"/> 否 <input type="checkbox"/>		
系统集成方案	是 <input type="checkbox"/> 否 <input type="checkbox"/>	是 <input type="checkbox"/> 否 <input type="checkbox"/>		
测试方案	是 <input type="checkbox"/> 否 <input type="checkbox"/>	是 <input type="checkbox"/> 否 <input type="checkbox"/>		
用户操作手册	是 <input type="checkbox"/> 否 <input type="checkbox"/>	是 <input type="checkbox"/> 否 <input type="checkbox"/>		
安装指南	是 <input type="checkbox"/> 否 <input type="checkbox"/>	是 <input type="checkbox"/> 否 <input type="checkbox"/>		

2.2 测试提交文档

[下面应当列出在测试阶段结束后，所有可提交的文档]

3. 测试进度

测试活动	计划开始日期	实际开始日期	结束日期
制定测试计划			
设计测试			
集成测试			
系统测试			
性能测试			
安装测试			
用户验收测试			
对测试进行评估			
产品发布			

4. 测试资源

4.1 人力资源

下表列出了在此项目的人员配备方面所作的各种假定。

[注：可适当地删除或添加角色项。]

角色	所推荐的最少资源（所分配的专职角色数量）	具体职责或注释

4.2 测试环境

下表列出了测试的系统环境。

软件环境（相关软件，操作系统等）

硬件环境（网络、设备等）

### 4.3 测试工具

此项目将列出测试使用的工具：

用途	工具	生产厂商/自产	版本

## 5. 系统风险、优先级

[简要描述测试阶段的风险和处理的优先级]

## 6. 测试策略

[测试策略提供了对测试对象进行测试的推荐方法。对于每种测试，都应提供测试说明，并解释其实施的原因。]

制定测试策略时所考虑的主要事项有：将要使用的技术以及判断测试何时完成的标准。

下面列出了在进行每项测试时需考虑的事项，除此之外，测试还只应在安全的环境中  
使用已知的、有控制的数据库来执行。]

注意：不实施某种测试，则应该用一句话加以说明，并陈述这样的理由。例如，“将不  
实施该测试。该测试本项目不适用”。

### 6.1 数据和数据库完整性测试

[在<项目名称>中，数据库和数据库进程应作为一个子系统来进行测试。在测试这些  
子系统时，不应将测试对象的用户界面用作数据的接口。对于数据库管理系统（DBMS），  
还需要进行深入的研究，以确定可以支持以下测试的工具和技术。]

测试目标	[确保数据库访问方法和进程正常运行，数据不会遭到损坏]
测试范围	
技术	[调用各个数据库访问方法和进程，并在其中填充有效的和无效的数据（或对数 据的请求）。 检查数据库，确保数据已按预期的方式填充，并且所有的数据库事件已正常发 生；或者检查所返回的数据、确保正当的理由检索到了正确的数据]

续表

开始标准	
完成标准	[所有的数据库访问方法和进程都按照设计的方式运行，数据没有遭到损坏]
测试重点和优先级	
需考虑的特殊事项	[测试可能需要 DBMS 开发环境或驱动程序在数据库中直接输入或修改数据。 进程应该以手工方式调用。 应使用小型或最小的数据库（记录的数量有限）来使所有无法接受的事件具有更大的可视度]

6.2 接口测试

测试目标	确保接口调用的正确性
测试范围	所有软件、硬件接口，记录输入输出数据
技术	
开始标准	
完成标准	
测试重点和优先级	
需考虑的特殊事项	接口的限制条件

6.3 集成测试

[集成测试——主要目的检测系统是否达到需求，对业务流程及数据流的处理是否符合标准，检测系统对业务流处理是否存在逻辑不严谨及错误，检测需求是否存在不合理的标准及要求。此阶段测试基于功能完成的测试。]

测试目标	检测需求中业务流程，数据流的正确性
测试范围	需求中明确的业务流程，或组合不同功能模块而形成一个大的功能
技术	[利用有效的和无效的数据来执行各个用例，用例流或功能，以核实以下内容： 在使用有效数据时得到预期的结果。 在使用无效数据时显示相应的错误消息或警告消息。 各业务规则都得到了正确的应用]
开始标准	在完或某个集成测试时必须达到标准
完成标准	[所计划的测试已全部执行。 所发现的缺陷已全部解决]
测试重点和优先级	测试重点指在测试过程中需着重测试的地方，优先级可以根据需求及严重来定
需考虑的特殊事项	[确定或说明那些将对功能测试的实施和执行造成影响的事项或因素（内部的或外部的）]

6.4 功能测试

[对测试对象的功能测试应侧重于所有可直接追踪到用例或业务功能和业务规则的测

试需求。这种测试的目标是核实数据的接受、处理和检索是否正确，以及业务规则的实施是否恰当。此类测试基于黑盒技术，该技术通过图形用户界面（GUI）与应用程序进行交互，并对交互的输出或结果进行分析，以此来核实应用程序及其内部进程。以下为各种应用程序列出了推荐使用的测试概要：]

测试目标	[确保测试的功能正常，其中包括导航，数据输入，处理和检索等功能]
测试范围	
技术	[利用有效的和无效的数据来执行各个用例、用例流或功能，以核实以下内容： 在使用有效数据时得到预期的结果。 在使用无效数据时显示相应的错误消息或警告消息。 各业务规则都得到了正确的应用]
开始标准	
完成标准	
测试重点和优先级	
需考虑的特殊事项	[确定或说明那些将对功能测试的实施和执行造成影响的事项或因素（内部或外部的）]

## 6.5 用户界面测试

[用户界面（UI）测试用于核实用户与软件之间的交互。UI 测试的目标是确保用户界面会通过测试对象的功能来为用户提供相应的访问或浏览功能。另外，UI 测试还可确保 UI 中的对象按照预期的方式运行，并符合公司或行业的标准。]

测试目标	[核实以下内容： 通过测试进行的浏览可正确反映业务的功能和需求，这种浏览包括窗口与窗口之间、字段与字段之间的浏览，以及各种访问方法（Tab 键、鼠标移动和快捷键）的使用 窗口的对象和特征（例如，菜单、大小、位置、状态和中心）都符合标准]
测试范围	
技术	[为每个窗口创建或修改测试，以核实各个应用程序窗口和对象都可正确地进行浏览，并处于正常的对象状态]
开始标准	
完成标准	[成功地核实出各个窗口都与基准版本保持一致，或符合可接受标准]
测试重点和优先级	
需考虑的特殊事项	[并不是所有定制或第三方对象的特征都可访问]

## 6.6 性能评测

[性能评测是一种性能测试，它对响应时间、事务处理速率和其他与时间相关的需求进行评测和评估。性能评测的目标是核实性能需求是否都已满足。实施和执行性能评测的目的是将测试对象的性能行为当作条件（例如工作量或硬件配置）的一种函数来进行评测和

微调。

注：以下所说的事务是指“逻辑业务事务”。这种事务被定义为将由系统的某个 Actor 通过使用测试对象来执行的特定用例，添加或修改给定的合同。]

测试目标	[核实所指定的事务或业务功能在以下情况下的性能行为： 正常的预期工作量、预期的最繁重工作量]
测试范围	
技术	[使用为功能或业务周期测试制定的测试过程。 通过修改数据文件来增加事务数量，或通过修改脚本来增加每项事务的迭代数量。 脚本应该在一台计算机上运行（最好是以单个用户、单个事务为基准），并在多个客户机（虚拟的或实际的客户机，请参见下面的“需要考虑的特殊事项”）上重复]
开始标准	
完成标准	[单个事务或单个用户：在每个事务所预期时间范围内成功地完成测试脚本，没有发生任何故障] [多个事务或多个用户：在可接受的时间范围内成功地完成测试脚本，没有发生任何故障]
测试重点和优先级	
需考虑的特殊事项	[综合的性能测试还包括在服务器上添加后台工作量。 可采用多种方法来执行此操作，其中包括： 直接将“事务强行分配到”服务器上，这通常以“结构化语言”（SQL）调用的形式来实现。 通过创建“虚拟的”用户负载来模拟数百上千个客户机。此负载可通过“远程终端仿真（Remote Terminal Emulation）”工具来实现。此技术还可用于在网络中加载“流量”。 使用多台实际客户机（每台客户机都运行测试脚本）在系统上添加负载。 性能测试所用的数据库应该是实际大小或相同缩放比例的数据库]

## 6.7 容量测试

[容量测试使测试对象处理大量的数据，以确定是否达到了将使软件发生故障的极限。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。例如，如果测试对象正在为生成一份报表而处理一组数据库记录，那么容量测试就会使用一个大型的测试数据库。检验该软件是否正常运行并生成了正确的报表。]

测试目标	[核实测试对象在以下高容量条件下能否正常运行： 连接或模拟了最大（实际或实际允许）数量的客户机，所有客户机在长时间内执行相同的、且情况（性能）最坏的业务功能。 已达到最大的数据库大小（实际的或按比例缩放），而且同时对执行多个查询或报表事务]
------	--------------------------------------------------------------------------------------------------------------------------------



续表

测试范围	
技术	[使用为性能评测或负载测试制订的测试。 应该使用多台客户机来运行相同的测试或互补的测试,以便在长时间内产生最繁重的事务量或最差的事务组合(请参见上面的“强度测试”) 创建最大的数据库大小(实际的、按比例缩放的、或填充了代表性数据的数据库),并使用多台客户机在长时间内同时运行查询和报表事务]
开始标准	
完成标准	[所计划的测试已全部执行,而且达到或超出指定的系统限制时没有出现任何软件故障]
测试重点和优先级	
需考虑的特殊事项	[对于上述的高容量条件,哪个时间段是可以接受的时间]

## 6.8 安全性和访问控制测试

[安全性和访问控制测试侧重于安全性的两个关键方面:

应用程序级别的安全性,包括对数据或业务功能的访问。

系统级别的安全性,包括对系统的登录或远程访问。

应用程序级别的安全性可确保:在预期的安全性情况下,Actor 只能访问特定的功能或用例,或者只能访问有限的数据库。例如,可能会允许所有人输入数据,创建新账户,但只有管理员才能删除这些数据或账户。如果具有数据级别的安全性,测试就可确保“用户类型一”能够看到所有客户消息(包括财务数据),而“用户二”看见同一客户的统计数据。

系统级别的安全性可确保只有具备系统访问权限的用户才能访问应用程序,而且只能通过相应的网关来访问。]

测试目标	应用程序级别的安全性: [核实 Actor 只能访问其所属用户类型已被授权访问的那些功能或数据。] 系统级别的安全性: [核实只有具备系统和应用程序访问权限的 Actor 才能访问系统和应用程序]
测试范围	
技术	应用程序级别的安全性: [确定并列出各用户类型及其被授权访问的功能或数据] [为各用户类型创建测试,并通过创建各用户类型所特有的事务来核实其权限] [修改用户类型并为相同的用户重新运行测试。对于每种用户类型,确保正确地提供或拒绝了这些附加的功能或数据] 系统级别的访问: [请参见以下的“需考虑的特殊事项”]
开始标准	
完成标准	[各种已知的 Actor 类型都可访问相应的功能或数据,而且所有事务都按照预期的方式运行,并在先前的应用程序功能测试中运行了所有的事务]
测试重点和优先级	
需考虑的特殊事项	[必须与相应的网络或系统管理员一直对系统访问权进行检查和讨论。由于此测试可能是网络管理可系统管理的职能,可能会不需要执行此测试]

## 6.9 故障转移和恢复测试

[故障转移和恢复测试可确保测试对象能成功完成转移,并能从导致意外数据损失或数据完整性破坏的各种硬件、软件可网络故障中恢复。

故障转移测试可确保:对于必须持续运行的系统,一旦发生故障,备用系统就将不失时机地“顶替”发生故障的系统,以避免丢失任何数据或事务。

恢复测试是一种对抗性的测试过程。在这种测试中,将把应用程序或系统置于极端的条件下(或者是模拟的极端条件下),以产生故障(例如设备输入/输出(I/O)故障或无效的数据数据库指针和关键字)。然后调用恢复进程并监测和检查应用程序和系统,核实应用程序或系统和数据已得到了正确的恢复。]

测试目标	[确保恢复进程(手工或自动)将数据库、应用程序和系统正确地恢复到预期的已知状态。测试中将包括以下各种情况: 客户机断电、服务器断电、通过网络服务器产生的通信中断 数据库指针或关键字无效、数据库中的数据元素无效或遭到破坏]
测试范围	
技术	[应该使用为功能和业务周期测试创建的测试来创建一系列的事务。一旦达到预期的测试起点,就应该分别执行或模拟以下操作: <ul style="list-style-type: none"> <li>● 客户机断电:关闭PC机的电源。</li> <li>● 服务器断电:模拟或启动服务器的断电过程。</li> <li>● 通过网络服务器产生的中断:模拟或启动网络的通信中断(实际断开通信线路的连接或关闭网络服务器或路由器的电源)。</li> <li>● 一旦实现了上述情况(或模拟情况),就应该执行其他事务。而且一旦达到第二个测试点状态,就应调用恢复过程。</li> <li>● 在测试不完整的周期时,所使用的技术与上述技术相同,只不过应异常终止或提前终止数据库进程本身。</li> <li>● 对以下情况的测试需要达到一个已知的数据库状态。当破坏若干个数据库字段、指针和关键字时,应该以手工方式在数据库中(通过数据库工具)直接进行。其他事务应该通过使用“应用程序功能测试”和“业务周期测试”中的测试来执行,并且应执行完整的周期]</li> </ul>
开始标准	
完成标准	[在所有上述情况中,应用程序、数据库和系统应该在恢复过程完成时立即返回到一个已知的预期状态。此状态包括仅限于已知损坏的字段、指针或关键字范围内的数据损坏,以及表明进程或事务因中断面而被完或的报表]
测试重点和优先级	
需考虑的特殊事项	[恢复测试会给其他操作带来许多的麻烦。断开缆线连接的方法(模拟断电或通信中断)可能并不可取或不可行。所以,可能会需要采用其他方法,例如诊断性软件工具。  需要系统(或计算机操作)、数据库和网络组中的资源。  这些测试应该在工作时间之外或在一台独立的计算机上运行]

## 6.10 配置测试

[配置测试核实测试对象在不同的软件和硬件配置中的运行情况。在大多数生产环境中，客户机工作站、网络连接和数据库服务器的具体硬件规格会有所不同。客户机工作站可能会安装不同的软件，例如，应用程序、驱动程序等。而且在任何时候，都可能运行许多不同的软件组合，从而占用不同的资源。]

测试目标	[核实测试可在所需的硬件和软件配置中正常运行]
测试范围	
技术	<ul style="list-style-type: none"> <li>• [使用功能测试脚本。</li> <li>• 在测试过程中或在测试开始之前，打开各种与非测试对象相关的软件（例如 Microsoft 应用程序，Excel 和 Word），然后将其关闭。</li> <li>• 执行所选的事务，以模拟 Actor 与测试对象软件和非测试对象软件之间的交互。</li> <li>• 重复上述步骤，尽量减少客户机工作站上的常规可用内存]</li> </ul>
开始标准	
完成标准	[对于测试对象软件和非测试对象软件的各种组合，所有事务都成功完成，没有出现任何故障]
测试重点和优先级	
需考虑的特殊事项	<ul style="list-style-type: none"> <li>• [需要、可以使用并可以通过桌面访问哪种非测试对象软件？</li> <li>• 通常使用的是哪些应用程序？</li> <li>• 应用程序正在运行什么数据？例如，在 Excel 中打开的大型电子表格，或是在 Word 中打开的 1000 页文档。</li> <li>• 作为此测试的一部分，应将整修系统、网络服务器、数据库等都记录下来]</li> </ul>

## 6.11 安装测试

[安装测试有两个目的。第一个目的是确保该软件在正常情况和异常情况的不同条件下，例如，进行首次安装、升级、完整的或自定义的安装都能进行安装。异常情况包括磁盘空间不足、缺少目录创建权限等。第二个目的是核实软件在安装后可立即正常运行。这通常是指运行大量为功能测试制定的测试。]

测试目标	核实在以下情况下，测试对象可正确地安装到各种所需的硬件配置中： <ul style="list-style-type: none"> <li>• 首次安装。以前从未安装过&lt;项目名称&gt;的新计算机</li> <li>• 更新。以前安装过相同版本的&lt;项目名称&gt;的计算机</li> <li>• 更新。以前安装过&lt;Project Name&gt;的较早版本的计算机</li> </ul>
测试范围	
技术	<ul style="list-style-type: none"> <li>• [手工开发脚本或开发自动脚本，以验证目标计算机的状态</li> <li>• 启动或执行安装。</li> <li>• 使用预先确定的功能测试脚本子集表运行事务]</li> </ul>

续表

开始标准	
完成标准	<项目名称>事务成功执行，没有出现任何故障。
测试重点和优先级	
需考虑的特殊事项	[应该选择<项目名称>的哪些事务才能准确地测试出<项目名称>应用程序已经成功安装，而且没有遗漏主要的软件构件]

7. 问题严重度描述

问题严重度	描述	响应时间
高	例如使系统崩溃	程序员在多长时间内改正此问题
中		
低		

8. 与测试有关的任务

- 制订测试计划
  - ◆ 确定测试需求、评估风险、制订测试策略
  - ◆ 确定测试资源、创建时间表、生成测试计划
- 设计测试
  - ◆ 确定并说明测试用例
  - ◆ 确定测试过程，并建立测试过程的结构
- 复审和评估测试覆盖
- 实施测试
  - ◆ 记录或通过编程创建测试脚本
  - ◆ 确定设计与实施模型中的测试专用功能
  - ◆ 建立外部数据集
- ◆ 执行测试
  - 执行测试过程、评估测试的执行情况、评估测试用例覆盖、评估代码覆盖
  - 核实结果、调查意外结果
  - 记录缺陷、分析缺陷
  - 确定是否达到了测试完成标准与成功标准

# 附录 E C++ Inspection Checklist

Copyright© 1999 by Christopher Fox.

---

## 1. Variable and Constant Declaration Defects (VC)

- ☐ Are descriptive variable and constant names used in accord with naming conventions?
- ☐ Are there variables with confusingly similar names?
- ☐ Is every variable correctly typed?
- ☐ Is every variable properly initialized?
- ☐ Could any non-local variables be made local?
- ☐ Are there literal constants that should be named constants?
- ☐ Are there macros that should be constants?
- ☐ Are there variables that should be constants?

## 2. Function Definition Defects (FD)

- ☐ Are descriptive function names used in accord with naming conventions?
- ☐ For every parameter of every function: Is the parameter passing mechanism (value or reference) appropriate?
- ☐ Are function parameters constant where appropriate?
- ☐ Is every function parameter value checked before being used?
- ☐ For every function: Does it return the correct value at every function return point?
- ☐ Can default arguments be used in instead of function overloading?
- ☐ Can any function or operator overloading be avoided?

## 3. Class Definition Defects (CD)

- ☐ Does each class have an appropriate constructor and destructor?
- ☐ Does every public base class have a virtual destructor function?
- ☐ Does every class that allocates space in a constructor have a copy constructor?
- ☐ For each member of every class: Could access to the member be further restricted?
- ☐ Do any derived classes have common members that should be in the base class?
- ☐ Can the class inheritance hierarchy be simplified?

## 4. Data Reference Defects (DR)

- ☐ For every array reference: Is each subscript value within the defined bounds?
- ☐ For every pointer reference: Is the correct level of indirection used?
- ☐ For every reference through pointer: Is the referenced storage currently allocated to the proper data?

## 5. Computation/Numeric Defects (CN)

- ☐ Are there any computations with mixed data types?
- ☐ Is overflow or underflow possible during a computation?
- ☐ For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?
- ☐ Are parentheses used to avoid ambiguity?

#### 6. Comparison/Relational Defects (CR)

- ☐ Is the "=" expression used instead of "=="?
- ☐ For every boolean test: Is the correct condition checked?
- ☐ Are there any comparisons between variables of inconsistent types?
- ☐ Are the comparison operators correct?
- ☐ Is each boolean expression correct?
- ☐ Are there improper and unnoticed side-effects of a comparison?
- ☐ Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?

#### 7. Control Flow Defects (CF)

- ☐ For each loop: Is the best choice of looping constructs used?
- ☐ Will all loops terminate?
- ☐ When there are multiple exits from a loop, is each exit necessary and handled properly?
- ☐ Does each switch statement have a default case?
- ☐ Are missing switch case break statements correct and marked with a comment?
- ☐ Is the nesting of loops and branches too deep, and is it correct?
- ☐ Can any nested if statements be converted into a switch statement?
- ☐ Are null bodied control structures correct and marked with braces or comments?
- ☐ Does every function terminate?
- ☐ Are goto statements avoided?

#### 8. Input-Output Defects (IO)

- ☐ Have all files been opened before use?
- ☐ Are the attributes of the open statement consistent with the use of the file?
- ☐ Have all files been closed after use?
- ☐ Is buffered data flushed?
- ☐ Are there spelling or grammatical errors in any text printed or displayed?
- ☐ Are error conditions checked?

#### 9. Module Interface Defects (MI)

- ☐ Are the number, order, types, and values of parameters in every function call in agreement with the called function's declaration?
- ☐ Do the values in units agree (e.g., inches versus yards)?
- ☐ Are reference and value parameters used properly?
- ☐ If a parameter is passed by reference, does its value get changed, and changed correctly by the called function?

**10. Comment Defects (CM)**

- ☐ Does every function, class, and file have an appropriate header comment?
- ☐ Does every variable or constant declaration have a comment?
- ☐ Is the underlying behavior of each function and class expressed in plain language?
- ☐ Is the header comment for each function and class consistent with the behavior of the function or class?
- ☐ Do the comments and code agree?
- ☐ Do the comments help in understanding the code?
- ☐ Are there enough comments in the code?
- ☐ Are there too many comments in the code?

**11. Layout and Packaging Defects (LP)**

- ☐ Is a standard indentation and layout format used consistently?
- ☐ For each function: Is it no more than about 60 lines long?
- ☐ For each compile module: Is no more than about 600 lines long?

**12. Modularity Defects (MD)**

- ☐ Is there a low level of coupling between modules (functions and classes)?
- ☐ Is there a high level of cohesion within each module (functions or class)?
- ☐ Is there repetitive code that could be replaced by a call to a function that provides the behavior of the repetitive code?
- ☐ Are library functions or pre-defined classes used where and when appropriate?

**13. Storage Usage Defects (SU)**

- ☐ Is statically allocated memory large enough?
- ☐ Is dynamically allocated memory large enough?
- ☐ Is all dynamically allocated memory freed, and freed when appropriate? Is there a free for every malloc and a delete for every new?

**14. Performance Defects (PE) [Optional]**

- ☐ Can better data structures or more efficient algorithms be used?
- ☐ Are logical tests arranged such that the often successful and inexpensive tests precede the more pensive and less frequently successful tests?
- ☐ Can the cost of recomputing a value be reduced by computing it once and storing the results?
- ☐ Is every result that is computed and stored actually used?
- ☐ Can a computation be moved outside a loop?
- ☐ Are there tests within a loop that do not need to be done?
- ☐ Can a short loop be unrolled?
- ☐ Are there two loops operating on the same data that can be combined into one?
- ☐ Are frequently used variables declared register?
- ☐ Are short and commonly called functions declared inline?

# 附录 F Java Code Inspection Checklist

## 1. Variable and Constant Declaration Defects (VC)

1. Are descriptive variable and constant names used in accord with naming conventions?
2. Are there variables with confusingly similar names?
3. Is every variable properly initialized?
4. Could any non-local variables be made local?
5. Are there literal constants that should be named constants?
6. Are there macros that should be constants?
7. Are there variables that should be constants?

## 2. Function Definition Defects (FD)

8. Are descriptive function names used in accord with naming conventions?
9. Is every function parameter value checked before being used?
10. For every function: Does it return the correct value at every function return point?

## 3. Class Definition Defects (CD)

11. Does each class have an appropriate constructor and destructor?
12. For each member of every class: Could access to the member be further restricted?
13. Do any derived classes have common members that should be in the base class?
14. Can the class inheritance hierarchy be simplified?

## 4. Computation/Numeric Defects (CN)

15. Is overflow or underflow possible during a computation?
16. For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?
17. Are parentheses used to avoid ambiguity?

## 5. Comparison/Relational Defects (CR)

18. Are the comparison operators correct?
19. Is each boolean expression correct?
20. Are there improper and unnoticed side-effects of a comparison?



## 6. Control Flow Defects (CF)

- 21. For each loop: Is the best choice of looping constructs used?
- 22. Will all loops terminate?
- 23. When there are multiple exits from a loop, is each exit necessary and handled properly?

### Praktikum Software Engineering 99: Code Inspection Checklist?

- 24. Does each switch statement have a default case?
- 25. Are missing switch case break statements correct and marked with a comment?
- 26. Is the nesting of loops and branches too deep, and is it correct?
- 27. Can any nested if statements be converted into a switch statement?
- 28. Are null bodied control structures correct and marked with braces or comments?
- 29. Does every function terminate?
- 30. Are goto statements avoided?

## 7. Input-Output Defects (IO)

- 31. Have all files been opened before use?
- 32. Are the attributes of the open statement consistent with the use of the file?
- 33. Have all files been closed after use?
- 34. Is buffered data flushed?
- 35. Are there spelling or grammatical errors in any text printed or displayed?
- 36. Are error conditions checked?

## 8. Module Interface Defects (MI)

- 37. Are the number, order, types, and values of parameters in every function call in agreement with the called function's declaration?
- 38. Do the values in units agree (e.g., inches versus yards)?

## 9. Comment Defects (CM)

- 39. Does every function, class, and file have an appropriate header comment?
- 40. Does every variable or constant declaration have a comment?
- 41. Is the underlying behavior of each function and class expressed in plain language?
- 42. Is the header comment for each function and class consistent with the behavior of the function or class?
- 43. Do the comments and code agree?
- 44. Do the comments help in understanding the code?
- 45. Are there enough comments in the code?
- 46. Are there too many comments in the code?

**10. Packaging Defects (LP)**

- 47. For each file: Does it contain only one class?
- 48. For each function: Is it no more than about 60 lines long?
- 49. For each class: Is no more than 2000 lines long (Sun Coding Standard) ?

Praktikum Software Engineering 99: Code Inspection Checklist

**11. Modularity Defects (MO)**

- 50. Is there a low level of coupling between packages (classes)?
- 51. Is there a high level of cohesion within each package?
- 52. Is there duplicate code that could be replaced by a call to a function that provides the behavior of the duplicate code?
- 53. Are framework classes used where and when appropriate?

**12. Performance Defects (PE) [Optional]**

- 54. Can better data structures or more efficient algorithms be used?
- 55. Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?
- 56. Can the cost of recomputing a value be reduced by computing it once and storing the results?
- 57. Is every result that is computed and stored actually used?
- 58. Can a computation be moved outside a loop?
- 59. Are there tests within a loop that do not need to be done?
- 60. Can a short loop be unrolled?
- 61. Are there two loops operating on the same data that can be combined into one?

## 参 考 文 献

- 1 Ron Patton. Software Testing. SAMS Publishing, 2001
- 2 Gordon Schelmeyer & James McManus. Handbook of Software Quality Assurance. Third Edition.
- 3 Mark Fewster & Dorothy Graham. Software Test Automation. Addison-Wesley, 1999
- 4 Rex Black. Managing the Testing Process (2nd Edition). Jihn Wiley & Sons, Inc., 2002
- 5 Roger Pressman. Software Engineering: A Practitioner's Approach. 5<sup>th</sup> Edition. McGraw-Hill Companies, 2001
- 6 飞思科技产品研发中心. 实用软件测试方法与应用. 北京: 电子工业出版社, 2003
- 7 Matt Teles 等著, 邓劲生等译. 程序调试思想与实践. 北京: 中国水利水电出版社, 2002
- 8 黄锡滋编著. 软件可靠性、安全性与质量保证. 北京: 电子工业出版社
- 9 面向对象的软件测试. 杨文宏等译. 北京: 机械工业出版社, 中信出版社, 2003
- 10 新浪科技. <http://tech.sina.com.cn/i/w/2003-08-20/0658223003.shtml>
- 11 计算机世界. [http://www.ccw.com.cn/htm/news1/net/safe/01\\_6\\_14\\_2.asp](http://www.ccw.com.cn/htm/news1/net/safe/01_6_14_2.asp)
- 12 天极网络. <http://www.yesky.com/20030321/1658575.shtml>
- 13 搜狐 IT 频道. <http://it.sohu.com/29/69/article210466929.shtml>
- 14 Subrahmanyam Allamaraju 等著, J2EE 服务器端高级编程. 闻道工作室译
- 15 Lingo Systems American translation association *The Guide to Translation and Localization Preparing Products for the Global Marketplace*, 2002
- 16 [STSC 99] *A Gentle Introduction To Software Engineering*, V3.1, United States Air Force, Software Technology Support Center, 1999
- 17 [SEL-81-305] *Recommended Approach to Software Development*, V3.1, National Aeronautics and Space Administration, 1992
- 18 Software Process Improvement With CMM》[美] Joseph Raynus
- 19 <http://www.51cmm.com/>
- 20 <http://www.opentest.net/softtest/mbug.htm>