目录

引子

在本书的开头，有必要介绍软件测试在软件开发中的位置、地位和作用，也就是帮助读者建立起有关软件测试的基本轮廓（big picture），这样对阅读和理解以后各章的内容会有很大帮助。要建立起软件测试的基本轮廓，也就需要回答下列几个问题：

- 软件测试的作用是什么？
- 软件测试在软件开发生命周期（SDLC）中的位置在哪里？
- 软件测试过程是怎样的？
- 软件测试团队的地位如何？
- 软件测试和软件质量保证（SQA）有何区别？

下面我们先简单回答这些问题。即使您不能完全理解也不要急，后面会逐步帮助您解开软件测试之谜。但有一点是明确的，在看完这段“引子”后，您对软件测试会有一个整体的认识，从而不至于陷入“盲人摸象”的困境。

1. 软件测试的作用

在购买商品时，会发现商品上贴有一个“QC”标签，这就是产品经过质量检验（Quality Control）的标志。软件测试就好比制造工厂的质量检验工作，是对软件产品和阶段性工作成果进行质量检验，力求发现其中的各种缺陷，并督促修正缺陷，从而控制和保证软件产品的质量。所以，软件测试是软件公司致力于提高软件产品质量的重要手段之一。

2. 软件测试在 SDLC 中的位置

在著名的软件瀑布模型中，软件测试处在“编程”的下游，在“软件维护”的上游，

先有编程后有测试，测试的位置很清楚，但瀑布模型没有反映 SDLC 的本质，没能准确无误地反映测试的位置。

实际上，测试贯穿整个 SDLC，从需求评审、设计评审开始，就介入到软件产品的开发活动或软件项目实施中了。测试人员借助于需求定义的阅读、讨论和审查，不仅能发现需求定义的问题，而且可以了解产品的设计特性、用户的真正需求，进而确定测试目标，准备用例（Use Case）并策划测试活动。同理，在软件设计阶段，测试人员可以了解系统是如何实现的，以及构建在什么样的平台之上等各类问题，这样可以提前准备系统的测试环境，包括硬件和第三方软件的采购，并着手研究如何测试系统，完成系统测试用例设计、测试工具的选型或启动测试工具的开发，进一步完善测试计划等。所有这些准备工作，都要花去很多时间，应尽早开展起来。

当设计人员在做详细设计时，测试人员就应直接参与具体的设计、参与设计的评审，找出设计的缺陷。同时，完成功能特性测试的用例，并基于这些测试用例开发测试脚本。

在编程阶段就进行单元测试，是一种很有效的办法，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，减少开发成本。

软件测试在 SDLC 中的位置，可以通过图 0-1 充分地体现出来。软件测试和软件开发构成一个全过程的交互、协作的关系，两者自始至终一起工作，共同致力于同一个目标——按时、高质量地完成项目。

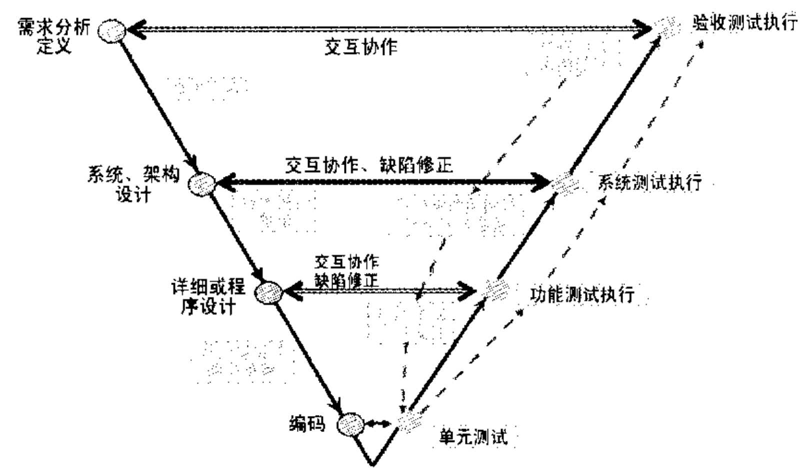


图 0-1　软件测试和 SDLC 的关系

3. 软件测试过程

软件测试的全过程，要经历如图 0-2 所示的很多个里程碑，其中主要的里程碑有：

- 产品需求文档（PRD）或市场需求文档（MRD）的评审和签发；
- 产品规格说明书（Spec）的评审和签发；
- 测试计划、测试计划书的评审和签发；
- 测试用例的设计、评审和签发；
- 功能测试；
- 系统测试；
- 验收测试。

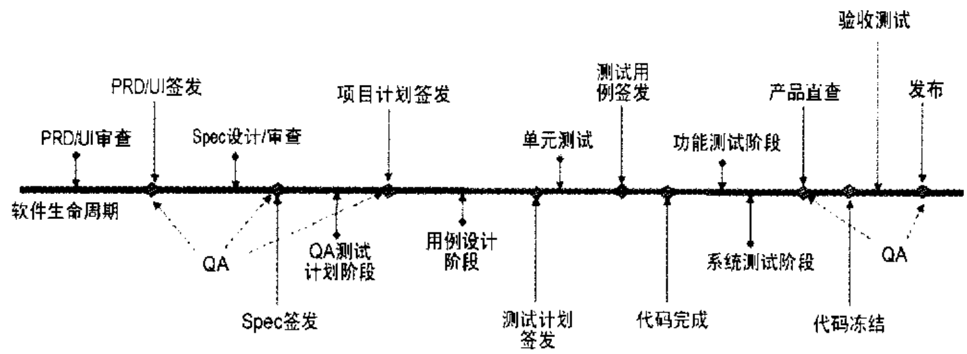


图 0-2 测试全过程的跟踪示意图

4. 软件测试团队的地位

在软件开发中，虽然有很多角色，有各种各样的人员参与，包括项目经理、产品经理、UI（用户界面）设计人员、文档人员等，但最大的两个团队就是测试团队和开发团队（由设计人员/程序员组成），也就是说，在一个软件公司，测试人员和程序员，占有最大比重。当然，有些软件公司，销售人员或客户支持人员也比较多。由此可见，软件测试团队的地位应该是举足轻重的。

5. 软件测试和 SQA 的区别

如前面所说，软件测试属于软件控制，它和 SQA（质量保证）的区别概括起来有 4 点，见表 0-1。

表 0-1

项目	软件测试	SQA
工作性质	技术性工作	管理性工作
对象	软件产品（包括阶段性产品）	软件过程
焦点	事后检验	强调预防
范围	在研发（R&D）部门或技术部门	在公司层次，跨所有部门，包括市场、销售、客户服务、行政、后勤、人事等部门

1

第1章 测试项目启动

良好的开端是成功的一半，要做好软件项目的测试工作，就不能忽视测试项目的启动。

软件测试，贯穿整个软件生命周期的始终。从项目启动的第一天开始，软件测试就介入其中，从软件测试计划、测试用例设计、自动化脚本开发、测试执行、结果分析直至完成测试报告，经过一个又一个不断循环上升的软件过程，履行其使命。

良好的开端是成功的一半，要做好软件项目的测试工作，就不能忽视测试项目的启动。测试项目启动时，要抓住一些要点。

- 弄清项目背景，抓住项目的要素。
- 深刻理解项目的需求。对于软件测试，理解质量需求显得尤为重要。

- 人是决定因素，项目测试组组长（经理）的人选至关重要，他/她将帮助建立整个测试组，并带领大家前进直至项目成功。
- 针对项目的不同类型、不同特点、不同规模，区别对待。

首先介绍软件质量需求和软件项目的特点，并以两个典型的项目为背景，展开项目的实施过程。然后，通过启动过程的展示，使大家认识到软件测试和软件开发是同时开始的，决不是在代码写完之后才开始，从而消除过去可能存在的误区。更重要的是告诉大家，在项目启动过程中应做好软件测试所需的各项准备工作，从而为接下来的软件测试计划、设计和执行打下坚实的基础。

1.1 了解软件的质量需求

软件测试是软件质量保证的一种诉求，是质量保证过程中所依赖的主要活动之一。质量保证的结果，在很大程度上依赖于软件测试的开展以及执行的结果。所以要做好测试工作，必须清楚地了解软件的质量需求。或者说，只有正确地理解了软件质量的含义、质量需求等之后，我们才能深刻地认识到软件测试的意义，才能准确地定义测试的目标、把握测试的要点，不会误入歧途，才有可能将软件测试工作做得尽善尽美。

1.1.1 软件产品的质量需求

软件的质量需求，从根本上说是为了引导和满足客户的需求，而软件质量具体表现在软件产品（或服务）固有的特性之上，如适用性、功能性、有效性、可靠性和性能等。在软件项目开发过程中，为了更好地进行软件质量管理，常常将软件质量分为功能需求和非功能需求。软件质量不仅要满足客户需求，还要考虑企业自身的利益，如缩短开发周期、降低开发成本等。这时，软件质量被分为用户的质量需求和软件组织自身的质量需求，除了用户和企业自身需求之外，我们还必须考虑一些特殊的社会需求和宗教需求。

- 游戏软件不应该含有暴力、色情内容，而且须不断提醒用户，长时间玩游戏有害健康；但如果从用户个体角度看，至少有部分用户喜欢那些暴力、色情的画面。
- 产品颜色的设置要和国家或地区的喜好、禁忌一致。
- 在有特定宗教信仰（如基督教、伊斯兰教、佛教、东正教等）的国家或地区，要尽量尊重宗教的意愿或旨意。

概念

软件质量定义 1983年，ANSI/IEEE STD729给出了软件质量的定义：软件产品满足规定的和隐含的与需求能力有关的全部特征和特性，它包括：

（1）软件产品质量满足用户要求的程度；

（2）软件各种属性的组合程度；

（3）用户对软件产品的综合反映程度；

（4）软件在使用过程中满足用户要求的程度。

关于软件质量，还有其他一些定义，体现了软件质量属性的不同视点。例如：

- **客户满意度** 指使最终的软件产品能最大限度地满足客户需求的程度。
- **一致性准则** 指在生命周期的每个阶段中，其工作产品总能与上一阶段的工作产品保持一致，最终可追溯到分配需求。
- **软件质量度量** 指设立软件质量度量指标体系（例如：ISO－9126），并以此来度量软件产品的质量。
- **过程质量观** 指软件的质量就是其开发过程的质量。因此，对软件质量的度量转化成了对软件过程的度量。要定义一套良好的软件“过程”，并严格控制软件的开发照此过程进行。

一些软件质量专家对软件质量有自己的定义。

- SEI的Watts Humphrey倾向于把质量说成“在实用性、需求、可靠性和可维护性一致上，达到优秀的水准”，以及“软件系统的质量取决于开发和维护它的过程的质量”。
- James Martin声称软件质量意味着在预算内按时发布产品，并满足用户需求。
- 软件复杂性领域内的专家Tom McCabe定义质量是“用户满意度的高水准、低缺陷率，而且伴随着低复杂性”。
- 贝尔实验室的John Musa认为质量是“低缺陷率、软件功能忠实于用户需求、高可靠性”的组合。
- 质量保证研究所（QAI）的Bill Perry定义质量是用户满意度的高水准、忠实于用户需求。

1．质量的功能需求

软件质量的功能需求比较容易理解，就是通过人机交互界面来完成用户所需要的各项

操作，包括数据的输入和结果输出。软件功能，一般会在有关的产品文档中进行定义，例如：

- 市场需求文档（Marketing Requirement Document，MRD）。
- 产品需求文档（Production Requirement Document，PRD）。
- 用户界面模拟（模型）文档（User Interface Mock-up，UI Mock-up）。
- 产品规格说明书（Production Specification，Spec）。

功能需求，也可被称为可说明性（accountability），用户可以基于产品或服务的描述和定义进行使用，即产品可以满足上述文档对功能的说明。

易用性（usability）可以视为非功能性需求，但更倾向于功能性需求。对于一个软件来说，易用性被视为用户学习、操作和理解软件所做出努力的程度，它和功能联系比较多。易用性好，常表现为“安装简单方便、容易使用、界面友好”等，并能适用于不同特点的用户，包括能给残疾人、有缺陷的人提供产品使用的有效途径或手段。

2．质量的非功能需求

软件的非功能需求内涵更为丰富，虽然建议在MRD和PRD中将其描述清楚，但在多数情况下，做到这点会比较困难。它往往需要根据软件设计阶段的工作和测试验证的结果才能最终确定。

软件的非功能需求主要体现在性能、有效性、可靠性等方面。

（1）**性能**（performance）：指在指定条件下，用软件实现某种功能所需计算机资源（包括内存大小、CPU占用时间等）的有效程度，以及系统响应、表现的状态。如果系统用完了所有可用的资源，那么系统性能就会下降。性能的操作特征包括与作业负载相关的特征，如响应时间、负载容量等。

（2）**有效性**（availablity）：指产品或服务对于客户能否保持有效，即在预定的启动时间中，系统真正可用并且完全运行的时间所占的百分比。可以用“系统平均无故障时间（MTTF，Mean Time To Failure）除以总的运行时间（MTTF与故障修复时间之和）”来计算有效性。例如，银行系统对时间要求的要求更严格——有效性要高，大于99.99%的有效性才能满足质量要求。一个有效性需求可能会这样说明：“工作日期间，从上午8点到下午8点，系统的有效性至少要达到99.95%。在当地时间，从下午8点到第2天早上8点，系统的有效性至少应达到99.5%。”

（3）**可靠性**（reliability）：指在规定的时间和条件下，软件维持其正常的功能操作、性能水平的程度，如软件坚固性和可靠性（防故障能力，即防止崩溃、内存泄漏等能力）、

资源利用率、代码完整性及技术兼容性等。健壮性和有效性有时可看成是可靠性的一部分。衡量软件可靠性的方法，包括正确执行操作所占的比例，在发现新缺陷之前系统运行的时间长度和缺陷出现的密度。可靠性对一些软件系统要求特别高，要求有很强的容错能力，并能保证长时间稳定运行，比如航空、铁路交通管制系统，全国联合售票系统等。

概念

软件质量的非功能性需求中的其他概念

（1）**可维护性**：指在一个运行软件中，当环境改变或软件发生错误时，进行相应修改所做努力的难易程度。可维护性取决于理解软件、更改软件和测试软件的难易程度，可维护性与灵活性密切相关。高可维护性对于那些经历周期性更改的产品或快速开发的产品很重要。

（2）**兼容性**：指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难易程度，或者是一个系统和外部条件共同工作的难易程度。兼容性表现在多个方面，如系统的软件和硬件的兼容性、软件不同版本的系统、数据的兼容性。兼容性，一定程度上也决定了或包含了可移植性。

（3）**可扩展性**：指将来功能增加、系统扩充的难易程度或能力。

（4）**可移植性**：指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难易程度。可维护性和可扩展性，对提高可移植性有很大的帮助。

3. 质量的用户需求

质量的用户需求是和用户直接相关的质量需求，包含功能性需求和部分非功能性需求。非功能性需求则集中在产品带给用户的体验上，包括性能、可用性、安全性等，但用户一般不关心软件开发成本和市场行为，除非是用户自己开发自己用，或者是用户自己投资的项目。所以质量的用户需求一般不包括软件可维护性、可移植性和可扩充性，而主要表现在：

- 能正常使用全部所需的功能，符合需求规格。
- 功能强大，而且界面美观，易用、好用。
- 内容健康，有益于生活和工作。
- 用户的数据安全、受保护和兼容。
- 能及时得到新的产品或得到更完美的软件服务。

- 软件可靠性很高，使用软件服务没有时间障碍。

4. 质量的企业需求

对于软件开发企业，软件的质量需求更多体现在市场竞争、市场国际化、成本控制等方面。企业知道，质量好的软件是留住客户的最关键手段，软件企业也必须依靠质量，才能立于不败之地。软件产业已是国际化的市场，质量是进入国际市场的一个关键门坎，而且高质量的软件可以大大降低“质量问题产生的劣质成本或不良成本”，增加公司的盈利。所以质量的企业需求，主要是一些非功能性需求，如软件的可维护性、兼容性、可移植性和可扩展性等，表现在下列内容上。

- 系统能否在异常情况下继续运行？故障期间的系统数据可以保存下来吗？
- 软件发生故障的频率如何？能否回避异常状况？
- 发现软件差错后是否容易修改？变更使用中的软件是否容易？
- 是否容易与其他系统连接？可否在其他系统里再利用？
- 功能扩充是否简单？移植到其他平台上是否也能正常运行？
- 检验性能是否简单？
- 软件管理是否容易？
- 需要多少资源？

软件的质量属性对于不同的软件种类来说，所具有的重要性是不同的，如“高效性和可靠性”对于嵌入式系统是很重要的，“有效性和可维护性”对于服务器、主机系统程序是非常重要的，就像“互操作性和可用性”对于桌面系统那样重要。

1.1.2 软件质量的对立面——软件缺陷

软件质量存在着对立面，即存在着不符合质量需求或违背软件用户、客户、企业意愿的问题，这就是常说的软件缺陷（Defect），常常又被叫做“Bug（臭虫）”。在现实中，软件很难做得完美无缺，总会存在这样那样的问题，不能很好地满足客户和企业的需求，这些问题也被称为软件缺陷。换句话说，任何降低软件质量的因素或问题，都可以被看作软件缺陷。而由于软件属于无形产品且软件系统具有复杂性，软件缺陷的产生，在一定程度上又是很难避免的。

通过 “软件缺陷第一次被叫做臭虫”的有趣故事——Grace Hopper 曾在计算机的继电器中发现一只“飞蛾”，它导致计算机死机，我们就更容易理解什么是软件缺陷，以及软件缺陷的危害了。

故事

故事发生在1945年9月的一天，那是一个炎热的下午。机房是一间在第一次世界大战时建造的老建筑，没有空调，所有窗户都敞开着。Hopper正领着她的研究小组夜以继日地工作，研制一台称为“MARK II”的计算机，它使用了大量的继电器（电子机械装置，那时还没有使用晶体管），这是一台不是纯粹的电子计算机。突然，MARK II死机了。研究人员试了很多次还是无法启动，然后就开始用各种方法来查找问题究竟出在哪里，最后定位到板子F第70号继电器出错。Hopper观察这个出错的继电器，惊奇地发现一只飞蛾躺在中间，已经被继电器打死。她小心地用摄子将蛾子夹出来，用透明胶布粘贴到“事件记录本”中，并注明“第一个发现虫子的实例”，然后计算机又恢复了正常。从此以后，人们将计算机错误戏称为臭虫（Bug），而把找寻错误的工作称为“找臭虫”（Debug）。Grace Hopper的事件记录本，连同那个飞蛾，现在都陈列在美国历史博物馆中。

实际上，在日常使用软件的经历中，也有类似的一些体验。如当使用一个新软件时，弹出错误提示窗口；在使用浏览器时，打开了几个网页之后，结果浏览器崩溃了；访问某个网站时，速度很慢，但同时访问其他网站时，速度却还正常，这说明不是网络连接问题，而是那个网站性能有问题。所有这些，都是软件缺陷的例子。

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。软件缺陷的表现形式主要有：

- 功能、特性没有实现或部分实现。
- 设计不合理，功能特性不明确，逻辑不清楚或存在矛盾。
- 产品实际结果和所期望的结果不一致。
- 没有达到产品规格说明书所规定的特性、性能指标等。
- 运行出错，包括运行中断、系统崩溃、界面混乱等。
- 数据结果不正确、精度不够、不完整或格式不统一。
- 用户不能接受的其他问题，如存取时间过长、界面不美观。
- 硬件或系统软件上存在的其他问题。

概念

软件缺陷的定义

在IEEE 1983 of IEEE Standard 729中给软件缺陷下了一个标准的定义：

（1）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；

（2）从外部看，软件缺陷是系统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存在的问题，是计算机系统或程序中存在的任何一种破坏正常运行能力的问题或错误，或者隐藏的功能缺陷或瑕疵。缺陷会导致软件产品在某种程度上不能满足用户的需要，或者表现为用户所需要的功能没有得到实现。

不要将“软件缺陷”和“软件错误（error）”两个概念混淆起来。软件缺陷范围更广，它涵盖了软件错误，还涵盖不一致性问题、功能需求定义缺陷和产品设计缺陷等。软件错误属于软件缺陷的一种类型——程序或系统的内部缺陷，往往是软件代码本身的问题，如程序的算法错误、语法错误或数据计算不正确、数据溢出等。软件错误表现如下。

- 数组和变量初始化错误或赋值错误。
- 算法错误，指在给定条件下没能给出正确或准确的结果。
- 语法错误，在一般情况下，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，往往在运行时才会发现。
- 计算和精度问题，指计算的结果没有达到所需要的精度，如小数点后的位数。
- 程序结构不合理、算法没有被优化，造成系统性能低下、系统资源浪费。
- 接口参数传递不匹配，导致模块不能成功地集成在一起。
- 文字显示内容不正确或拼写错误。
- 输出格式不对或不美观等。

但软件错误往往会导致系统的某项功能失效，或者成为系统使用的故障，即软件内部缺陷最终表现为外部缺陷。外部缺陷主要表现为软件故障或功能失效，即软件提供给用户的功能或服务，不能达到用户的要求或没有达到事先设计的指标。如在使用时突然中断，导致不能获得最终结果，或者获得的结果是不正确的。

1.1.3 软件缺陷产生的原因

软件缺陷的产生主要是由软件产品的特点和开发过程决定的，如软件的需求经常不够明确，而且需求变化频繁，开发人员不太了解软件需求，不清楚应该“做什么”和“不做什么”，常常做一些不合需求的事情，这时产生的问题最多。同时，软件竞争非常厉害，技术日新月异，使用新的技术，也容易产生问题。对于不少软件企业来说，“争取在时间上取胜”常常是其主要的市场竞争策略之一；实现新功能、很酷的功能，被认为比质量更为重要，这会导致日程安排很紧；需求分析、设计等投入的时间和精力远远不够，这也是

产生软件错误的主要原因之一。

产生软件错误可能还有其他一些原因，例如，软件设计文档描述不清楚，文档本身存在错误，这会导致使用者产生更多的错误；还有沟通上的问题、开发人员的态度问题及项目管理问题等。《微软开发者成功之路（之一）》概括了以下 7 项主要原因：

- 项目期限的压力。
- 产品的复杂度。
- 沟通不良。
- 开发人员疲劳、压力过大或受到干扰。
- 缺乏足够的知识、技能和经验。
- 不了解客户的需求。
- 缺乏动力。

我们可以从软件自身特点、团队工作和项目管理等多个方面进行分析，找出导致软件缺陷更多的一些原因，这可以归纳为如下 3 个方面。

1. 软件开发过程自身的特点造成的问题

- 软件需求定义难以做到清清楚楚，导致设计目标偏离客户的需求，从而引起功能或产品特性上的缺陷。
- 软件系统结构非常复杂，而又无法构造成一个完美的层次结构或组件结构，结果将导致意想不到的问题。例如系统中对象、类太多，在众多的对象、类之间相互调用、引用时容易出现问题。
- 新技术的采用，可能涉及技术或系统兼容性问题，而事先没有考虑到。
- 对程序逻辑路径或数据范围的边界考虑不够周全，容易在边界条件上出错，或者超过边界条件后又缺少保护导致出错。
- 系统运行环境复杂，用户使用的计算机环境千变万化，测试环境很难完全模拟用户的各种实际环境和操作习惯，不可避免在一些特定的用户环境下系统出问题。
- 系统实际运行中的数据量要比开发、测试环境的数据量大得多，而且系统实际运行中由于各种意外可能会产生一些垃圾数据，从而引起系统负载和数据不兼容等问题。
- 对于一些实时应用系统来说，如果没有精心的设计和技术处理，容易造成时间上无法保持精确同步，使系统行为在时间上出现不协调、不一致从而出错。
- 没有考虑或处理好系统崩溃后的自我恢复、故障转移或数据的异地备份等情况，

从而存在系统安全性、可靠性的隐患。

- 通讯端口多、存取和加密手段矛盾等会造成系统的安全、性能等存在问题。

2. 软件项目管理的问题

- 受质量文化的影响，不重视质量计划，对质量、资源、任务、成本等的平衡性把握不好，容易挤掉需求分析、评审、测试等的时间，于是遗留的缺陷也会比较多。
- 开发周期短，需求分析、设计、编程、测试等各项工作不能完全按照定义好的流程来进行，工作不够充分，结果也就不完整、不准确，错误较多；周期短，还给各类开发人员造成太大压力，从而引起一些人为的错误。
- 开发流程不够完善，存在较多的随机性和缺乏严谨的内审或评审机制，容易产生问题。
- 文档不完善、风险估计不足等。

3. 团队工作的问题

- 沟通不够、不流畅，导致不同阶段、不同团队的开发人员对问题的理解不一致
- 项目组成员技术水平参差不齐，或者新员工较多或培训不够等，也容易引起问题。

1.1.4 究竟什么是软件测试

简单地说，软件测试就是发现系统需求定义、系统设计、程序代码和文档中的缺陷。但仅仅这样简单认识软件测试还是不够的，那么究竟什么是软件测试呢？

G.J.Myers 给出的定义：“程序测试是为了发现错误而执行程序的过程”。这个定义被软件测试业界所认可，并经常被引用。但实际上，这样的定义还不能完全反映软件测试的内涵，它仍局限于“程序测试”。随后，G.J.Myers 进一步提出了有关程序测试的 3 个重要观点，那就是：

（1）测试是为了证明程序有错，而不是证明程序无错误。

（2）一个好的测试用例在于它能发现至今未发现的错误。

（3）一个成功的测试是发现了至今未发现的错误的测试。

这些观点使程序测试的含义显得更加完善和丰富，但是还没有跳出“程序测试”的圈子。作为软件测试，相对于程序测试，其内涵和范畴有很大不同。

要完整地理解软件测试，就要从不同方面和视角去辨证地审视软件测试。概括起来，

软件测试就是贯穿整个软件开发生命周期、对软件产品（包括阶段性产品）进行验证和确认的活动过程，其目的是尽快尽早地发现在软件产品中存在的各种问题——与用户需求、预先的定义不一致的地方。

概念

软件测试的定义

在 IEEE 1983 of IEEE Standard 729 中给软件测试下了一个标准的定义：使用人工或自动的手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。这里明确地提出软件测试的目标是检验是否满足需求。

对软件测试还有一些不同的定义。

- G.J.Myers 在其经典著作《软件测试之艺术》(The Art of Software Testing[1]) 中给出的定义是："程序测试是为了发现错误而执行程序的过程"。
- 测试被定义为"对软件系统中潜在的各种风险进行评估的活动"。
- 软件测试就是"验证 (Verification)"和"有效性确认 (Validation)"活动构成的整体，即软件测试 = V&V。

1. 软件测试的狭义观点和广义观点

G.J.Myers 给出了测试定义——"程序测试是为了发现错误而执行程序的过程"，实际上这是一个狭义的概念，因为他认为测试是执行程序的过程，也就是传统意义上的测试——在代码完成后，通过运行程序来发现程序代码或软件系统中错误。但是，这种意义上的测试是不能在代码完成之前发现软件系统需求及设计上的问题的，把需求、发现设计上的问题遗留到后期——最终在代码中体现出来，这样就可能会造成设计、编程的部分或全部返工。需求阶段和设计阶段的缺陷在开发过程中会产生扩大效应，缺陷的严重性随时间发展越来越严重，其结果会大大增加软件开发的成本、延长开发的周期等。这种狭义的观点主要是受软件开发瀑布模型的影响，而且非常不利于保证软件质量。

正是为了尽早地发现问题，才将测试延伸到需求评审、设计审查活动中去，也就是将"软件质量保证"的部分活动归属于测试活动。实际上，在软件开发的实际操作中，常常会将"软件测试"和"质量保证"这两种努力（efforts）合并起来。

延伸后的软件测试，被认为是软件测试的一种广义概念。这就引出了软件测试的两个

注 1：G.J.Myers, New York: John Wiley & Sons, 1979.

概念"静态测试"和"动态测试"。这样，静态测试和动态测试就构成了一个全过程的、完整的软件测试，而且静态测试显得更为重要。

概念

静态测试的主要活动是评审，即通过对需求、设计、配置、程序和其他各类文档的审查来检验相应的内容是否满足用户的需求。由于静态测试不需要运行程序，所以测试对象是属于静态的。

动态测试通过运行程序来发现软件系统中的问题，在程序运行过程中发现缺陷，它具有动态性。

2. 软件测试的辨证观点

G.J.Myers 的第 2 个观点是"测试是为了证明程序有错，而不是证明程序无错误"，引出了软件测试的另外一个争论：

软件测试究竟是证明所有软件的功能特性是正确的，还是相反——对软件系统进行各种试探和攻击，找出软件系统中不正常或不工作的地方？

就我个人理解，这两个方面都有一定道理，前者（证明或验证所有软件的功能特性是正确的）是从质量保证的角度来思考软件测试，后者（证明程序有错）从软件测试的直接目标和测试效率来思考，两者应该相辅相成。在后者的思想背景下，可以认为，测试不是为了证明所有的功能都能正常工作，恰恰相反，测试就是为了找出那些不能正常工作、不一致性的问题，也就是说，测试的工作就是发现缺陷（detect bug），即在软件开发过程中，分析、设计与编码等工作都是建设性的，唯独测试带有"破坏性"，它想方设法发现软件所存在的问题。

软件测试就是在这两者之间获得平衡，但对于不同的应用领域，两者的比重是不一样的。例如，国防、航天、银行等软件系统，承受不了系统的任何一次失效，因为任何失效都完全有可能导致灾难性的损失，所以强调前者，以保证非常高的软件质量。而一般的软件应用或服务，则可以强调后者，质量目标设置在"用户可接受水平"，以降低软件开发成本，加快软件发布速度，有利于市场的扩张。

下面就是这两种观点的基本描述。

（1）验证软件是"工作的"，以正向思维方式，针对软件系统的所有功能点，逐个验证其正确性。其代表人物是软件测试领域的先驱 Dr. Bill Hetzel （代表论著《全面指导

软件测试》，*The Complete Guide to Software Testing*[2]）。

（2）证明软件是“不工作的”，以反向思维方式，不断思考开发人员理解的误区、不良的习惯、程序代码的边界、无效数据的输入及系统的弱点，试图破坏系统、摧毁系统，目标就是发现系统中各种各样的问题。其代表人物就是上面多次提到的 G.J.Myers。他强调，一个成功的测试必须是发现缺陷的测试，不然就没有价值。

3. 软件测试的风险观点

测试被定义为“对软件系统中潜在的各种风险进行评估的活动”，这就引出软件测试的风险观点。软件测试自身的风险性是大家公认的，测试的覆盖率不能做到 100%；另外一方面，软件测试的标准有时不清楚，软件规格说明书是测试中的一个标准，但也不是唯一的标准。因为规格说明书本身的内容完全有可能是错误的，它所定义的国内特性不是用户所需要的。所以，我们常常强调软件测试人员应该站在客户的角度去进行测试，除了发现程序中的错误，还要发现需求定义的错误、设计规格说明书的缺陷。但是，测试在大多数时间/情况下是由工程师完成的，而不是客户自己来做，所以又怎么能保证工程师和客户想得一样呢？

有人把开发比作打靶，目标明确，就是按照设计规格说明书去实现系统的功能。而把测试比作捞鱼，目标不明确，自己判断哪些地方鱼多，就去哪些地方捞；如果只捞大鱼（严重缺陷），网眼就可以大些、撒网区域相对比较集中（测试点集中在主要功能上）。如果想把大大小小的鱼都捞上来，网眼就要小，普遍撒网，不放过任何一块区域（测试点遍及所有功能）。

在“风险”观点的框架下，软件测试可以被看作是一个动态的监控过程，对软件开发全过程进行检测，随时发现不健康的征兆，发现问题、报告问题，并重新评估新的风险，设置新的监控基准，不断地持续下去，包括回归测试。这时，软件测试完全可以看作是软件质量控制的过程。

对应这种观点，产生基于风险的测试策略，首先评估测试的风险，每个功能出问题的概率有多大？根据 Pareto 原则（也叫 80/20 原则），哪些功能是用户最常用的功能（约占 20%）？如果某个功能出问题，其对用户的影响又有多大？然后根据风险大小确定测试的优先级。优先级高的功能特性，测试优先得到执行。一般来讲，针对用户最常用的这 20%功能（优先级高）的测试会得到完全地、充分地执行，而低优先级功能的测试（用户不常用的功能，约占 80%）就可能由于时间或经费的限制，降低测试的要求、减少测

注 2：Bill Hetzel, New York: John Wiley & Son, 1993.ISBN 0471565679.

试工作量，这样做风险性并不是很大。

4. 软件测试的经济学观点

"一个好的测试用例在于它能发现至今未发现的错误"，这体现了软件测试的经济学观点。实际上，软件测试经济学问题至今仍是业界关注的问题之一。经济学的核心就是要盈利，盈利的基础就是要有一个清楚的商业性目标。同样，商业性目标是否正确，直接决定了企业是否盈利。在多数情况下，软件测试是在公司内执行的。正是公司的行为目的，决定了软件测试含义或定义经济性的一面。正如对软件质量的定义不仅仅局限于"和客户需求的一致性、适用性"，而且要增加其他的要求——"开发成本控制在预算内、按时发布软件、系统易于维护"等。

软件测试也一样，要尽快尽早地发现更多的缺陷，并督促和帮助开发人员修正缺陷。原因很简单，缺陷发现得越早，所付出的代价就越低，例如在编程阶段发现一个需求定义上的错误，其代价将 10 倍于在需求阶段就发现该缺陷的代价。这就是从经济学的观点来说明测试进行得越早越好这样一个道理。Boehm 在 1981 年写了一本书——《软件工程经济》（*Software Engineering Economics*[3]），那里面谈到了同样的问题。

5. 软件测试的标准观点

如果从标准观点来看软件测试，可以定义为"验证（Verification）"和"有效性确认（Validation）"活动构成的整体，即软件测试 = V&V。

"验证"是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）一致。相当于以软件产品设计规格说明书为标准进行软件测试的活动。

"有效性确认"是确认所开发的软件是否满足用户真正需求的活动。一切从客户出发，理解客户的需求，并对软件需求定义和设计存疑，以发现需求定义和产品设计中的问题。这主要通过各种软件评审活动来实现，包括让客户参加评审和测试活动。

当然，软件测试的对象是产品（包括阶段性产品，如市场需求说明书、产品规格说明书、技术设计文档、数据字典、程序包、用户文档等），而质量保证和管理的对象集中于软件开发的标准、流程和方法等上。

1.1.5 软件测试的目标

软件测试的目标，概括地说，就是为了更快、更早地将软件产品或软件系统中所存在

注 3：B.W.Boehm, NJ: Prentice-Hall, 1981.ISBN 0-13-822122-7.

的问题找出来，并促进系统分析人员、设计人员和程序员尽快地解决这些问题。软件测试是质量保证过程中不可缺少的一部分，测试人员和整个软件开发团队共同努力，确保及时地向客户或用户提供一个高质量的软件产品，包括在正确性、效率、适用性、可维护性、可扩充性、可伸缩性、安全性、可靠性、系统性能、系统容量、服务可管理性、兼容性等各方面满足事先设计或定义的要求，使软件系统最终能满足用户的需求和软件企业自身的需求。

测试人员没有发现缺陷并不能说软件中就不存在缺陷，因为要证明没有缺陷不是一件容易的事。不管已做测试工作的量多大，缺陷仍可能就存在于那些没有被测到的地方，而这个地方可能很小。实际上，测试人员能说的就是，所发现的缺陷肯定是缺陷，随着经验的积累，测试人员肯定的信心就越强。所以说，软件测试标目标是找到至今还未发现的缺陷，而不是确保没有缺陷。为什么这样说呢？主要是有以下几个原因。

（1）测试的覆盖率几乎不可能达到 100%。也就是说，软件测试不能穷举所有的测试用例，不能将程序中所有的路径都测试一遍，因为对于多数软件系统，由于其复杂性和规模，测试用例数或程序路径数会是一个非常大的数据。不能完成 100%的测试，也就不可能将所有的缺陷发现出来，因此测试总是存在风险的。如果有充足的时间不断地进行测试，总是可以找到更多的缺陷。

（2）发现缺陷越多的地方，往往漏掉缺陷的可能性越大。缺陷多的地方，意味着这一区域的代码质量差，但并不意味着测试效率和能力得到了很大地提高，因为测试效率和能力的提高不是一朝一夕的事。缺陷越多的地方，其风险越大。

（3）修正过去的缺陷会产生新的缺陷，而且需求总是在变化着的，系统不是一成不变的，“变”是永恒的。这种“变”会带来新的缺陷、新的风险。

（4）测试人员对产品的理解不能完全代表实际用户的理解，在两者之间肯定存在差异。这种差异的存在，也就意味着会存在开发人员和测试人员没有发现的缺陷，因为他们认为这些不是缺陷，但对用户来说，这些是缺陷。

（5）软件测试的环境难以和实际运行或用户的环境完全吻合。他们存在着差异，这种差异也蕴含着风险。只在用户环境下才存在的一些缺陷，或者是在运行一段时间以后，由过多的数据或一些脏数据（特殊原因产生的无效数据）而引起的一些缺陷，不会或不容易被测试人员发现。

（6）没有缺陷不是靠测试来保证的，而是靠软件过程的各个环节保证的。任何一个缺陷都是在软件构建的过程中产生的，即在需求定义、软件设计和编程过程中会引入各种各样的缺陷，在测试活动中并没有引入任何一个缺陷，不是吗？

全程软件测试

要点

软件测试是质量保证的重要活动之一。其间接目的是验证所有功能可以按照事先设计或定义的那样实现，但其直接目的并不是验证每个功能都能正常实现，而是设法找到每个功能不能正常实现的地方，即尽量去促使软件故障的产生。这就是软件测试的反向思维，它和软件开发构建系统的正向思维相呼应。这和上面所说的“软件测试的目的是找到至今还未发现的缺陷，而不是确保没有缺陷”是一致的，也和软件测试的风险观点归为静态测试的范畴是一致的，这样才可以保证测试的效率。

基于测试的反向思维，我们可以说，一个成功的测试就是测试用例没有通过——测试的实际结果没有达到所期望的正确结果。通过测试使软件系统出现故障，说明这次测试是成功的。所以在测试时，应尽量发挥自己的创造力，采用一些看似“不正当”的手法，去发现软件系统薄弱的环节或漏洞，从而迫使软件缺陷暴露出来。如果将软件测试目标定义为“验证应用程序能够正常工作”，测试人员的思路就会受到限制，测试可能会过于循规蹈矩，测试数据、配置和测试用例的设计就往往处在一些正常范围内，测试通过率会很高，测试看上去很顺利，但不容易发现软件缺陷，潜在的风险（遗漏缺陷的可能性）会很大。

当测试人员采用一些“不正当”的方法、数据或配置等进行测试时，开发人员可能会反对，于是对测试人员说，“没有人会那样做！”但作为测试人员还是要坚持这样去做，因为用户的操作或使用也完全可能会是各种各样的、有意或无意的行为。有时用户的确不会有意那么去做，但无意中会有许多不规范的操作：

- 可能无意中会做一些非常规的操作，如快速地用鼠标将某个对象来回拖拽几次、对着某个按钮连击好几次等；
- 用户也会犯错误，如不小心做了删除操作、不小心碰了电源开关或给出一个错误的配置等；
- 用户的计算机资源达到或接近其上限值，如内存、磁盘空间、数据库记录等；或者同时有很多并发用户在访问系统；
- 用户的运行环境不匹配，如没有光驱、软驱，没有安装打印机，代理服务器设置的特定方式等。

即使在这样不规范的操作下，我们的程序、系统也应该经得起考验，具有应有的抗冲击、抗异常的能力。

1.2 选定测试组长

为什么这么早就谈“选定测试组长”的问题，而不是介绍更多的软件测试的知识呢？也不是先讨论如何建立测试团队或测试组呢？

首先，本书是根据实际软件项目测试活动展开的先后次序来安排内容的，测试组长在项目一启动就要确定。其次，在大多数场合中会先选定项目测试负责人——暂且称测试组长（也可能是测试经理）。在测试组长确定以后，由测试组长来决定测试组人选，或者是测试组长和测试团队的上层管理者商量如何组建项目测试组，包括测试组的具体人选。更为重要的原因是测试资源和开发资源的分配关系，虽然两者需要同时介入一个项目，但在前期，对测试人员的需求会少些，测试组长可以先于测试组其他人员进入项目，了解项目背景、审查项目有关文档并制定测试计划。这时并不需要很多的测试人员，此时项目组的测试人员还在其他项目的收尾工作上，虽然大部分开发人员已从那个项目抽出来了。开发人员和测试人员在项目上的投入存在一定的时间差。动态地平衡项目人力资源，能更充分地利用资源，提高生产力，创造更好的效益。

如果还是不能理解这样的管理思路，那让我们一起先看看测试过程和开发过程的关系，相信你就会明白了。

1.2.1 测试过程和开发过程的关系

测试过程和开发过程都贯穿软件过程的整个生命周期，它们是相辅相成、相互依赖的关系，概括起来有 3 个关键点。

（1）测试过程和开发过程是同时开始的，同时结束的，两者保持同步的关系。

（2）测试过程是对开发过程中阶段性成果和最终的产品进行验证的过程，所以两者相互依赖。前期，测试过程更多地依赖于开发过程；后期，开发过程更多地依赖于测试过程。

（3）测试工作的重点和开发工作的重点可能是不一样的，两者有各自的特点。不论是在资源管理，还是在风险管理上，两者都存在着差异。

1. 同步的关系

改进的 V 模型和 W 模型（也可视为 V 模型的延伸），是对测试过程和开发过程同步关系的最好描述。V 模型，在参考书《软件测试方法和技术》和《单元测试之道》（请见

附录 H）中已有描述，这里以 W 模型为例，说明两者的同步关系。

从图 1-1 可以看出，软件分析、设计和实现的过程，同时伴随着软件测试——验证和确认的过程，而且包括软件测试目标设定、测试计划和用例设计、测试环境建立等一系列测试活动的过程。也就是说，项目一启动，软件测试的工作也就启动了，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的。对软件开发过程和测试过程两者之间的同步关系，做进一步阐述如下。

- 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，确定测试目标，可以准备用例（Use Case）并策划测试活动。
- 在做系统设计时，测试人员可以了解系统是如何实现的，基于什么样的运行平台，这样可以设计系统的测试方案和测试计划，并尽早准备系统的测试环境，包括硬件和第三方软件的采购。这些准备工作往往需要较长的时间来处理。
- 在做详细设计时，测试人员可以参与设计，并对设计进行评审，找出设计的缺陷，同时设计功能、新特性等各方面的测试用例，完善测试计划，并基于这些测试用例开发测试脚本。
- 在编程的同时进行单元测试是一种很有效的办法，此法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。

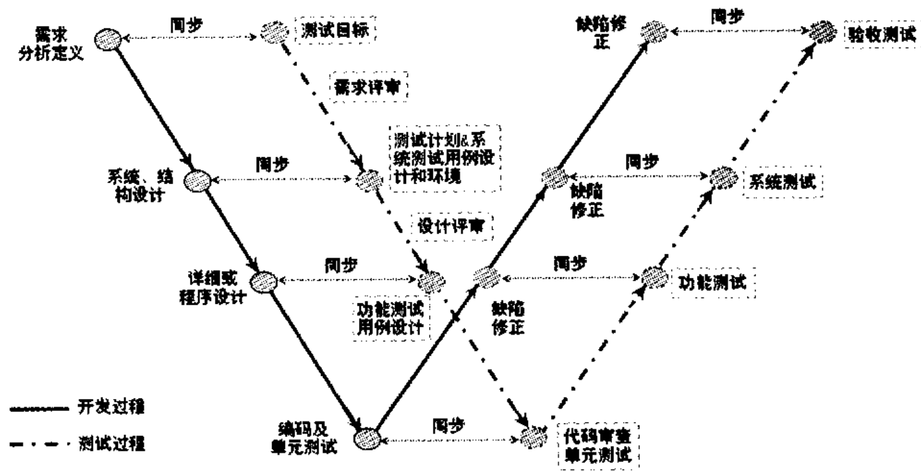


图 1-1 测试过程和开发过程的同步关系

2. 依赖的关系

测试过程和开发过程的依赖关系很明显，两者的同步关系从侧面也说明了这一点。再

者，测试活动是建立在软件开发的成果之上的，即测试的对象就是软件开发的阶段性成果，如设计文档、程序代码和可执行的程序。而软件开发的进一步活动又依赖于测试的结果，如果测试结果反映开发成果正确、良好，开发活动就可以快速地进入下一个阶段，否则，开发活动则需要修正当前阶段所存在的缺陷，不能进入下一个阶段。

如果将软件产品研发周期分为两半，前一半称为前期，主要是设计、实现，相当于 V 模型的左边；后一半称为后期，主要是验证和缺陷修正阶段，相当于 V 模型的右边。

在前期，软件研发过程主要集中在设计、实现阶段，设计和实现的能力决定了整个软件研发过程的进展状态，自然，测试过程前期更多地依赖于开发过程。在后期，测试策略和能力会直接影响测试效率，测试效率高就能更快地发现缺陷，缺陷也能更快地得到修正，所以开发过程后期更多地依赖于测试过程。

3. 两者的差异

开发过程的中心任务是构建软件，而测试过程的中心任务就是发现构建过程中存在的问题。由于关键任务不同，思维方式、方法和策略都会不同，资源、风险等过程管理也不相同。在后面的章节，我们会不断触及这些主题，并进行深入讨论，这里着重讨论资源分配。

从图 1-2 可以看出，开发前期投入的资源较多，在编程时到达高峰，然后逐渐减少。而测试人员在后期投入的资源较多，前期投入的资源相对少。当然，如果自动化程度很高的话，等所有测试脚本都完成之后，测试执行的工作量会大大降低。所以，测试资源的分配依赖于测试自动化的程度。另外，测试资源的分配还依赖于软件构建（设计、编程等）的质量、应用系统的规模和复杂度等。

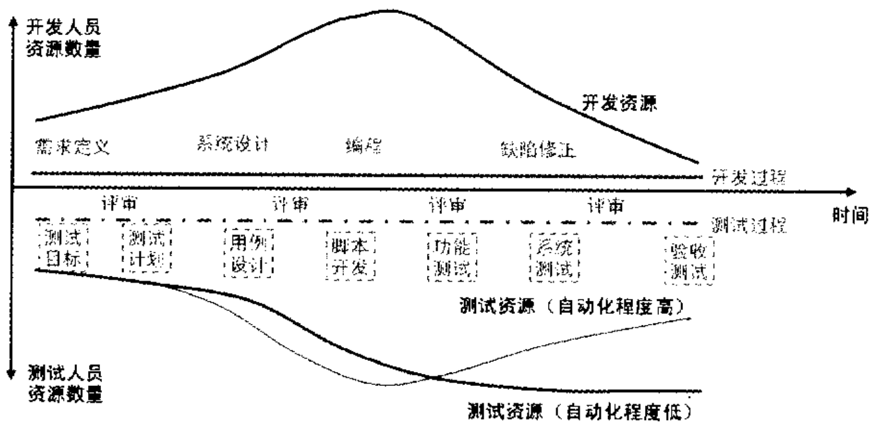


图 1-2　测试过程和开发过程资源分配的差异

1.2.2 测试组长的人选

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，这些测试任务都是最基本的，是测试团队在各个阶段自身的主要工作。除了这些主要的任务，测试团队还要完成其他任务：阅读和审查软件功能说明书、设计文档，审查程序，与开发人员、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。

测试组长将来要负责测试组的日常管理，更重要的是确保软件测试计划、测试用例和执行的质量，使项目获得成功。测试组长不仅要有良好的技术、产品分析能力和解决问题能力，包括对开发平台和系统运行平台的了解，在相关的数据库、网络、编程语言等领域的经验，而且要有丰富的测试工作经验，熟悉测试流程、测试方法和自动化技术，能解决测试工作中可能遇到的各种问题。

测试组长的能力要相对全面，但着重强调在项目管理、测试流程控制、沟通等方面的能力，在技术上也不能弱，但不需要过高的要求。测试组长的责任偏重于测试项目的计划和跟踪、有效测试策略的制定和测试小组的团队管理等，其主要责任如下：

- 负责一个独立的测试项目及其测试组的管理工作。
- 制定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。
- 负责工作量的预估和测试项目内部的资源、任务安排。
- 熟悉产品的功能、特性，审查产品需求规格说明书，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。
- 对项目组成员进行培训和指导。
- 作为测试组代表，参与项目组的有关会议，如缺陷复审或清理会议。
- 根据需求设计文档和代码，负责和参与测试用例的设计和评审。
- 参与代码的审查、检查单元测试的状态。
- 选择测试工具或拟定测试脚本的结构，指导或参与脚本的开发。
- 确定和审查测试环境的配置，协助测试实验室管理员在本项目测试环境的工作。
- 实施软件测试，控制测试进度，并对所发现的缺陷进行跟踪分析和报告。
- 解决测试中的项目问题，或者将这些问题反馈给项目经理或测试经理，推动这些问题得到及时合理的解决。

- 编写项目的整体测试报告，保证产品质量。
- 负责项目的总结，总结经验、吸取教训。

概念

测试用例是为了特定的测试目的（如考察特定程序路径或验证某个产品特性）而设计的测试条件、测试数据及与之相关的测试规程的一个特定的使用实例或场景。测试用例也可以被称为有效地发现软件缺陷的最小测试执行单元。

测试脚本是测试工具执行的一组指令集合，使计算机能自动完成测试用例的执行，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，也可以直接用脚本语言编写脚本。测试工具脚本中包含的数据和指令，可以根据指令组合形式分为线性脚本、结构化脚本和共享脚本，也可以根据指令和数据的分离形式，分为数据驱动脚本和关键字驱动脚本。

1.2.3 测试团队在项目中的位置

单靠软件测试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。测试团队的基本责任应该是：

- 发现软件程序、系统或产品中所有的问题。
- 尽早地发现问题。
- 督促和协助开发人员尽快地解决程序中的缺陷。
- 帮助项目管理人员制定合理的开发计划。
- 对缺陷进行跟踪、分析和分类总结，以便让项目的管理人员和相关的负责人能够及时、清楚地了解产品当前的质量状态。
- 帮助改善开发流程、提高产品开发效率。
- 促进程序编写的规范性、易读性、可维护性等。

从软件测试团队的上述责任就能认识到软件测试的重要性，它具有很高的地位。软件的实现有时会遇到一些难题，需要技术攻关，软件设计人员和程序员在纯计算机技术上的要求较高。但质量始终是产品和企业的生命，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，质量保证人员应具有权威性和与之相称的地位。

不少公司（包括微软公司），都将软件测试团队和质量保证（QA）团队合在一起，

组成测试（或 QA）部门。不管名称如何，但其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，拥有更多的责任，具有两个基本职能：软件测试和质量保证。

（1）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。

（2）更重要的是对产品的开发过程进行跟踪、审查，定义流程并推广流程，及时纠正在执行新的流程中所出现的问题，不断改进流程。

（3）分析竞争对手的产品，了解自己产品设计不足之处，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，旨在保证每个阶段的产品输出是正确的、符合产品质量要求的。

在不同的公司中，开发团队的模式可能会存在一些差别，甚至有比较大的区别，从而软件测试团队的地位也是不一样的。如果进行分类，可以概括为 3 类。

（1）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 1-3。

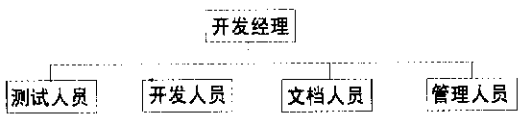


图 1-3　以开发为核心的组织模型

（2）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 1-4。

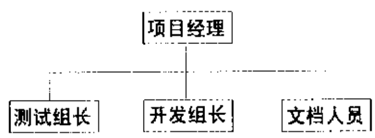


图 1-4　以项目经理为核心的组织模型

（3）项目经理、开发组长和测试组长“三足鼎立”，测试团队具有独立的、权威的地位，见图 1-5。

图 1-5 三足鼎立的组织模型

1.3 掌控项目背景

无论是做什么项目，要想获得成功，必须清楚项目的背景，包括项目的来源、客户和需求等。软件测试也不例外，要做好软件测试计划和随后的工作，也需要掌控软件项目的背景。如何更好地掌控项目背景？就是要抓住与软件测试相关的项目要素，获得、了解与这些要素相关的详细信息。通过了解软件项目的要素信息，就可以掌握项目的不同特点，并针对这些特点采取相适应的测试方法和策略，更好地开展测试活动。

1.3.1 软件测试的项目要素

拿到一个软件项目，首先要了解哪些内容？或者说，项目测试计划前，需要获取哪些项目数据，知道项目有哪些特点？这就要求我们在项目启动时，一定要抓住软件项目的要素，根据这些要素，逐个地去搜集信息、获得数据，并进行分析。

项目的基本要素包括项目范围、工期、质量和成本，如果进一步展开项目要素，它将包括项目的目标、客户和用户、范围、工期、预算和成本、质量、人员、材料设备、技术和合同管理等。即使在软件特定领域，软件项目的要素也离不开这些内容，而且这些内容也都是很重要的内容，只不过许多技术人员不关心预算、成本、合同等，甚至不关心客户。在这里，讨论的焦点虽然集中在技术方面，但也绝对不能忽视客户或用户的需求，不能忽视软件开发成本和合同中的有关条款内容。

要点

做产品，不是写程序。如果技术人员把要做的事情仅仅看作是写代码、编程序和开发系统，而不是看作开发为客户服务的产品，那么问题可能会很严重。技术人员往往过多地从技术方面思考，而很少真正从客户的角度去想，有时也可以开发出比较精妙的、甚至让人惊叹的东西，但在大多数情况下，他们开发出来的产品将不是客户真正需要的、高质量的产品。在这种不正确的意识下开发出来的软件产品，缺陷会很多，离客户需求的距离也会很远，并且开发效率低，会给企业带来很高的开发成本。

要掌控软件测试项目的背景，就要了解下列软件项目的要素及其相关的信息。

（1）**用户**：软件的用户是谁？是为个人服务，还是为中小企业服务，或者为大企业服务？用户是专业的技术人员，还是普通的教师、学生、公务员？如果是面向公务员，可进一步分析，是面向政府机关各层次的工作人员（包括高层领导、中层领导、办事员），还是只面向办事人员？面向不同层次的人员，对软件系统的界面、权限等有什么不同的特殊要求？

（2）**目标**：就是软件要真正解决的问题，或要实现的业务功能，如处理个人所得税网上申报的所有流程、网上购物、信用卡认证、英语词汇学习等，非常具体、清楚。

（3）**范围**：软件产品功能的范围和服务对象的范围。如个人所得税网上申报业务项目，只是完成“申报、查询”工作，还是否包括“税率调整通告、咨询、申诉、个税管理”等功能？是为某个城市居民服务，还是包括农村居民？是为北京市服务，还是为全国服务？

（4）**工期**：项目的阶段划分、各个里程碑的日期等。项目什么时候启动？系统设计和编程又分别在什么时候完成？产品发布和市场宣传等日期是否已确定？

（5）**项目类型**：是长期性产品，还是一次性项目？是独立项目，还是多方合作的、综合性集成项目？是本地项目，还是外包项目？是企业应用系统，还是一般软件工具开发？如果是 Web 应用系统，也可以进一步分为“以页面内容为中心、以数据库为中心、以应用逻辑（工作流）为中心”等不同类型。

（6）**软件平台**：是单机，还是网络？是客户端软件，还是服务器端软件？是 Windows，还是 Mac、Linux、Solaris？是企业内部网络（Intranet），还是互联网（Internet）、无线移动网络？是微软公司.Net 平台，还是 J2EE/J2SE/J2ME 等？

（7）**开发工具和语言**：是 Visual Studio，还是 Eclipse、PowerBuilder？是 C++、C#、Java/JSP、ASP、PHP、VB 等语言中的一种，还是混合编程语言？

概念

客户和用户是两个不同的概念。软件产品的使用者或软件服务接受者是用户（User），而客户（Customer）是更为广义的概念，是公司为实现目标所需的产品和过程而影响到的人，不仅包括接受产品或服务的组织或个人，而且包括潜在的客户，例如消费者、委托人、最终使用者、零售商、受益者和采购方等。当一个软件组织直接为用户服务，为特定的用户完成特定的项目时，这时客户和用户是同一个对象。但如果某个用户的产品是

通过第 3 方——软件承包商等委托给软件组织进行开发的，这时客户和用户则是分离的。

里程碑是软件生命周期各阶段末尾的特定点，在这些特定点上，阶段工作已结束，并且已经取得了正式的阶段性产品。里程碑的设定伴随着阶段性目标、进入/进出标准或指标等的确定，设定里程碑是为了更好的管理项目。如微软软件开发框架（MSF）中有愿景、计划、开发、稳定和发布等主要里的程碑。在开发阶段，还可进一步设定 3~4 个小里程碑，如设计、编程、集成和代码冻结等。对于软件测试，常见的里程碑有：测试计划签发（Sign-off）、测试用例签发、自动测试脚本完成、功能测试完成、性能测试完成等，可以参考附录 A 的软件测试全景图。

1.3.2 两个典型项目的介绍

在实际工作中，经常碰到的软件项目主要有如下类型。

（1）纯客户端软件，如字处理软件、下载软件、媒体（音频/视频）播放软件等。

（2）纯 Web（B/S）应用系统，如门户网站、个人博客网站、网络信息服务等。

（3）客户端/服务器（C/S）应用系统，如邮件系统、群件或工作流系统、即时消息系统等。

（4）大型复杂企业级系统，涉及面广、集成性强，包括了 B/S、C/S、数据库、目录服务、服务器集群、XML 接口等各个子系统。

这里以两个典型项目作为全程测试活动过程的项目背景，旨在帮助大家更好地掌握测试的方法、技术和时间。其中 Google Talk 是一个客户端/服务器（C/S）软件，主要项目因素可简单描述如下。

- 用户：想拥有即时消息（Instance Message，IM）的网络用户，年轻人偏多。
- 目标：操作方便、沟通及时并能和邮件等集成。
- 范围：见后面分析。
- 工期：第一个版本需要 6 个月左右，发布周期相对长些，适合应用敏捷方法。
- 类型：长期性产品，不断升级。
- 运行在 Windows、Mac 和 Linux 等各种操作系统之上。
- 开发语言主要是 C++。

而雅虎日历（Yahoo! Calendar）是一个 B/S 应用系统，包括页面、Web 服务器、数据库和邮件服务器等，项目特点如下。

- 用户：网络用户、商务人员使用较多，特别是销售、管理人员居多。
- 目标：管理日常活动、事件，及时获得提醒等。
- 范围：见后面功能分析和质量需求分析。
- 工期：第一个版本需要 3~6 个月，发布周期比较独立，以后每 1~2 个月发布一次更新产品。
- 类型：长期性产品，更新和部署相对容易。
- 服务器端运行在 Linux/Apache/WebLogic 之上，客户端主要是通过浏览器访问，可以运行在各种操作系统上。
- 开发语言主要是 Java。

1.3.3 Google Talk 客户端功能简介

Google Talk 是由 Google 推出的一个直观易用、免费的即时通信工具，它可从 *http://www.google.com/talk/intl/zh-CN/index.html* 下载。与其他同类软件相比，Google Talk 简洁、使用方便，而且结合 Google 的其他组件，功能也很强大。Google Talk 有如下主要的功能。

- 使用 Gmail 用户名和密码登录。
- 单击姓名打开聊天窗口，可同时打开任意多个聊天窗口。在框中直接输入，发送即时消息（如图 1-6 所示）。
- 通过单击“电子邮件”按钮，或者从好友列表中直接单击姓名旁边的邮件图标，使用 Gmail 账户发送电子邮件。
- 要呼叫好友，在打开的聊天窗口中单击呼叫按钮。还可从好友列表中直接单击姓名旁边的呼叫图标进行呼叫。
- 从 Gmail 联系人列表中选择或直接输入电子邮件地址，可以邀请好友。
- 如果好友也安装了 Google Talk，其姓名旁边将显示彩球图标，可立即呼叫或向其发送即时消息。
- Gmail 联系人会预载到 Google Talk。最常发送电子邮件的联系人列在好友列表的顶部。

- 不同颜色的小球图标及不同的符号表示好友的在线状态，如有空通话或聊天、闲置、正忙和离线等。
- 可设置在线状态或对其他人将鼠标放在 Google Talk 窗口中好友姓名上时所显示的信息进行自定义。
- 如果其他 Google Talk 用户想将某个人添加到其好友列表中，对方会在屏幕顶部看到提醒。
- 可邀请任意多个好友下载 Google Talk，并将其添加到好友列表中。
- 如果在聊天时取消记录，所有的聊天内容均不会保存在双方的 Gmail 账户中。

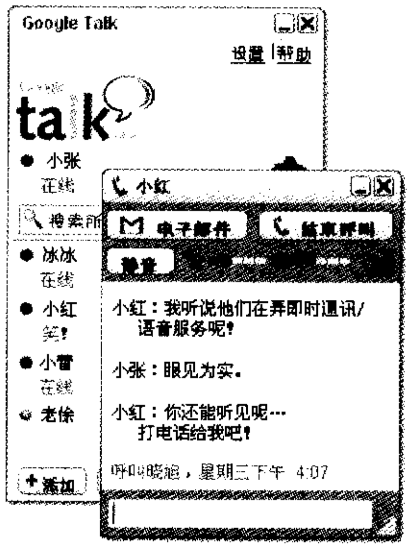


图 1-6 Google Talk 主界面和聊天窗口

1.3.4 Web 应用——雅虎日历功能介绍

随着互联网的应用越来越普遍，人们已经开始摆脱墙上的传统日历、办公桌上的台历，而使用在线日历。在线日历，可以随时随地访问，并且可以进行更好地协作，形成群组日历，可以共享在线日历。雅虎日历（*http://cn.calendar.yahoo.com/*）服务于 1998 年推出，是一个相对比较成熟的产品，如图 1-7 所示。其主要功能有如下。

（1）**建立活动**。提供了用户添加活动事项的方法，用户可以输入活动事项的名称和选择有限的级别、到期日、状态，以及是否共享和为活动事项添加便笺的功能，另外，雅虎日历还提供了搜索输入的功能，用户可以设置搜索的各种选项。

（2）**日历显示功能**。图 1-7 的左侧是时间活动浓缩的月历，可以点击月历中的日期直接添加、编辑和删除项目，还可以添加周期性（每周、每月等）的活动。而在主要界面——屏幕右边提供了 4 种日历浏览模式：日期、星期、月份及年份，用户可以根据日历跳转到特定的日期或在不同的时间分类里面列出用户的活动事项。

- 日：按照用户所选择的日期的时间顺序排列，从上午 8:00 到下午 6:00 显示用户添加的活动事项，默认是当前日期。
- 周：显示用户一周内每天的活动事项，默认是当前周。
- 月：显示所选择的月内每天的用户活动事项，按周排序，默认是当前月。
- 年：显示所选择的年的所有月和日期，并且有活动事项的日期以粗体高亮度来显示，默认是当前年。
- 活动列表：列出用户有活动事项的日期，按照时间先后的顺序排列。
- 待办事项：列出用户的待办事项，按照时间顺序排列，并提供添加/删除/标记功能。

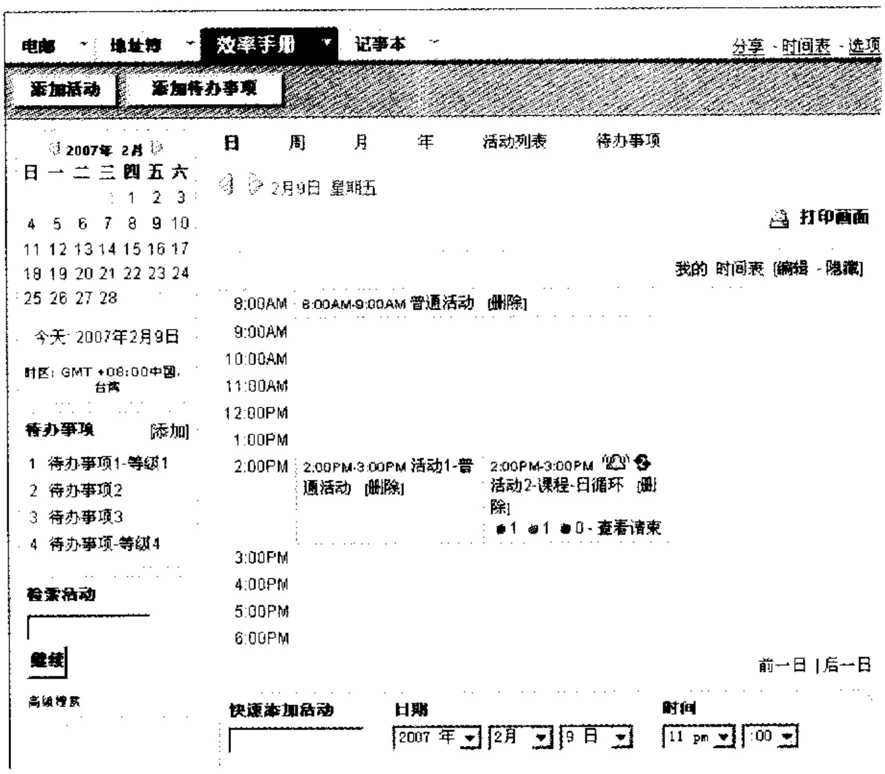


图 1-7 雅虎日历的主界面

（3）**共享效率手册** 提供完善的日历工具，还可以发送电子邮件邀请其他人、当收到新邮件时会有相应的提示等。针对日历共享，有多种不同的权限设定。

（4）**个性化定制** 可以选择所在的时区、工作时间、提醒方式、个人偏好等。

（5）**搜索功能** 可以根据名称、内容等进行精确或模糊的查询。

（6）**便捷的时间表** 能将所注意或关心的事件时间和个人行事整合起来，如选择有兴趣的事件种类；在显示设定中选择显示或隐藏事件；将事件的时间及日期和个人效率手册进行对照；自动显示更新及新事件等。

（7）**提醒功能** 用于设置效率手册中将发生活动的时间，使用户能提前（从 5 分钟到 14 天）得到及时的邮件通知、雅虎即时通、手机短消息提醒等。

1.4 制定测试规范

为了确保后续的测试工作质量，需要制定软件测试规范。当然，对于一个软件组织来说，并不是等到每个测试项目到来时，才开始进行测试规范的制定。实际上，一个规范的软件组织，事先已有一套软件开发过程规范，这其中包括软件测试规范。但对于一个具体项目，可能要对组织定义的测试规范进行剪裁，从而获得适合本项目的软件测试规范。不管是哪种情况，软件测试规范都是重要的，它伴随着整个测试过程，规范着测试活动的行为，能确保测试工作的质量，进而确保软件产品的质量。

概念

软件测试规范就是对软件测试的流程过程化，并对每一个过程元素进行明确的界定，形成完整的规范体系。规范一般形成在标准之后，与标准相比，规范显得更微观，往往是标准在某个领域的具体应用中逐步形成的，它更具领域特点、更易于操作。

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的属于自己企业的规范，特别是专门提供测试服务的企业，这种操作规范的内容与实施情况往往是其取得软件开发商信任的法宝。

软件测试规范，可以参见国家标准 GB-8566《信息技术软件生存期过程》和国际标准 ISO/IEC TR 15504。

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。这里主要讨论软件测试规范中实质性的内容，而不讨论一些附属性的内容（如规范目的、范围、文档结构、词汇表、参考信息、可追溯性等）。

在软件开发实践中，一般会从以下几个方面入手来规范测试过程，并在每个子过程中明确角色、职责、活动内容及所需文档。

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

1. 角色的确定

任何项目的实施，首先要考虑的是人的因素，对人进行识别与确认，软件测试尤其不能例外。在软件测试中，通常会把所有涉及人员进行分类以确立角色，并按角色进行职责划分，如表1-1所示。

表1-1　软件测试中最基本的角色定义

测试组长	业务专家，负责项目的管理、测试计划的制定、项目文档的审查、测试用例的设计和审查、任务的安排、与项目经理和开发组长的沟通等
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
资深测试工程师	负责产品设计规格说明书的审查、测试用例的设计和技术难题的解决，主要参与数据库、系统性能和安全性等技术难度较高的测试
自动化测试工程师	负责测试工具的开发、测试脚本的开发等
初级测试工程师	执行测试用例和相关的测试任务，侧重功能测试用例的设计和执行

2. 进入准则

进入准则也就是对软件测试切入点的确立。根据软件测试的广义观点，软件测试伴随着整个软件开发生命周期中的活动，在软件过程的各个阶段都离不开验证和确认活动，因此软件项目立项并得到批准，就意味着软件测试的开始。对于各个阶段，也需要定义测试

进入准则，见下面“4. 活动过程”的描述。

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中判断是否符合要求的依据和标准，包括需求描述、系统设计、程序代码及软件配置计划等文档资料。这些文档都是测试的输入项，如表 1-2 所示。

表 1-2　软件测试输入项

软件项目计划	软件项目计划是一个综合的项目信息载体，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，如市场需求文档（MRD）或产品需求文档（PRD），以及利用 CASE、UML 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模块或组件、类、对象等中的元素，如数据字典、对象属性、类的方法和全局变量等，可以据此实现编码	《详细设计说明书》
软件程序单元	包括所有已完成的程序单元源代码、数据库脚本、系统配置文件等	
软件集成计划	软件工作版本的定义、工作版本的内容、集成的策略及实施的先后顺序等	《系统集成计划》
软件工作版本	按照集成计划创建的各个集成工作版本	《软件配置计划》

4. 活动过程

a. 制定测试计划

角色：测试设计人员，一般由测试组长、资深测试工程师担任。

活动描述：

- 制定测试计划——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——根据需求规格说明书、质量计划等收集和整理测试需求信息，确定质量需求和测试目标。

- 制定测试策略——针对测试需求，定义测试阶段、测试类型、测试方法、测试风险回避措施及所需的测试工具等。
- 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或每一个测试阶段建立通过准则。
- 确定资源和进度——确定测试所需的软硬件资源、人力资源及测试进度。
- 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。

b. 测试设计

角色：测试设计人员（测试工程师）。

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。

（1）设计测试用例。

- 为每一个测试需求，确定其需要的测试用例。
- 为每一个测试用例，确定其输入及预期结果。
- 确定测试用例的测试环境配置、需要的驱动程序或桩程序。
- 编写测试用例文档。
- 对测试用例进行同行评审。

（2）设计测试过程。

- 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。

（3）设计单元测试和集成测试需要的驱动程序和桩程序。

c. 开发测试工具和脚本

角色：自动化测试工程师、测试工程师和程序员。

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桩程序。

- 根据测试过程，创建、开发测试脚本，并且调试测试脚本。
- 根据设计编写测试需要的测试驱动程序和桩程序。

d. **执行单元测试**

角色：以程序员为主，测试工程师为辅。

活动描述：执行单元测试的目的是验证单元的内部结构及单元实现的功能。

- 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；
- 详细记录单元测试结果，并将测试结果提交给相关人员（项目经理、开发组长、产品经理等）；
- 回归测试——对修改后的单元执行回归测试。

e. **执行集成测试**

角色：程序员和测试工程师。

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化脚本执行集成测试；
- 详细记录集成测试的结果，并将测试结果提交给相关人员。

f. **执行系统测试**

角色：（资深）测试工程师、测试实验室管理员。

活动描述：执行系统测试的目的是确认软件系统工作版本不仅满足功能性需求，还满足非功能性需求，如性能、安全性、兼容性等。

- 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；
- 详细记录系统测试结果，并对测试结果进行分析，提交测试结果和分析报告给相关人员；
- 回归测试——对修改后的软件系统版本执行回归测试。

g. **评估测试**

角色：测试人员和相关人员。

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——由相关组对每一次测试结果进行分析，并提出变更请求或其他处理意见。
- 评估阶段测试状态和产品质量状态，如对每一个阶段的测试覆盖率进行评估；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和提供测试分析报告并进行审查。

概念

（1）**单元测试**是对软件基本组成单元进行的测试，其测试对象是软件设计的最小单位——模块或组件，也可以包括类或函数，软件单元具有独立性，可以将它与系统/程序的其他部分隔离出来，从而完成测试。单元测试也是软件测试过程中最早期的测试活动。

（2）**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。集成测试一般是一个逐渐加入单元进行测试的持续过程，直至所有单元被组合在一起，成功地构成完整的软件系统，从而完成集成测试的使命。

（3）**系统测试**就是充分运行或模拟运行软件系统，以验证系统是否满足产品的质量需求，特别是非功能性的质量需求。与功能测试的区别在于系统测试主要指压力测试、容量测试、性能测试、安全性测试和可靠性测试等。系统测试可以和功能测试并行进行，但基本功能已能正常运行后，才能进行系统测试。

（4）**验收测试**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，也称为交付测试。验收测试一般会根据产品规格说明书严格地检查产品，逐字逐句地对照说明书检查对软件产品做出的各方面要求，确保所开发的软件产品符合用户预期的各项要求，即验收测试是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环境、数据和习惯等。验收测试的重要特征就是用户参与。

（5）**回归测试**是由于软件修改或变更，对修改后的工作版本所有可能影响的范围进行测试。回归测试的目的是发现原来正常的功能特性出现新的问题——回归缺陷，从而确保原来正常的或符合要求的特性不受其他区域修改的影响。回归测试伴随着整个测试过程，在功能测试和系统测试、单元测试和集成测试中，一旦有变更或修正，都要进行相应的回归测试。

5. 输出项

软件输出项较多，包括软件测试计划、测试用例，还包括测试缺陷记录、测试结果分析报告等，如表 1-3 所示。

表 1-3 软件测试输出项

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	测试过程对给定测试用例(或测试用例集)的设置、执行和结果评估的详细说明的集合	软件测试过程模板
测试缺陷记录	测试结果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	测试缺陷记录模板
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）的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

验证和确认是一个过程，在这过程中，依据需求定义和产品规范，确定软件活动和产品是否满足所给定的要求和条件，判断产品中所实现的功能、特性是否满足客户的实际需求，如表 1-4 所示。

表 1-4 软件测试验证与确认项

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、其他相关测试计划进行评审
软件测试用例评审	由测试组、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、其他相关组对测试过程进行评审
测试结果评估	由测试组、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、其他相关组对测试分析报告进行评审
质量保证（SQA）评审	由 SQA 人员对软件测试活动进行审计

7. 退出准则

退出准则满足组织/项目的测试结束的标准。

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活动就结束了。软件质量度量工作，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，提供给客户及组织内部的管理层。

1.5 小结

从软件质量概念及其需求出发，了解什么是软件缺陷及其产生的原因，从而真正理解究竟什么是软件测试。

软件质量的需求不仅分为功能性需求和非功能性需求，而且还分为用户的需求和软件企业自身的需求，从而构成一个完整的质量需求。相对于功能性需求，非功能性需求容易被忽视，但非功能性需求对软件质量同样重要。

无论是对测试人员，还是测试组长，都需要理解软件测试的内涵：

- 由静态测试和动态测试构成一个全过程的、完整的软件测试，而且静态测试显得更为重要。
- 测试是为了证明程序有错，而不是证明程序无错误，即测试的工作就是为了发现缺陷，为了找出那些不能正常工作、不一致性的问题。这里强调反向思维，以提高测试的效率。
- 测试是对软件系统中潜在的各种风险进行评估的活动，强调软件测试自身的风险性，测试的覆盖率很难达到 100%。所以，软件测试被看作是一个动态的、全过程的质量监控过程，它会随时发现、报告软件缺陷和问题，不断评估质量风险，持续进行回归测试，以保证质量。
- 一个好的测试用例是在于它能发现至今未发现的错误，要尽快尽早地发现更多的缺陷。缺陷发现得越早，所付出的代价就越低。
- 根据 Pareto 原则（80/20 原则），用户最常用的 20%功能（优先级高）应得到完全地、充分地测试。
- 软件测试就是“验证（Verification）”和“有效性确认（Validation）”活动构成的整体。“验证”是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，即它是针对软件开发活动一致性的检验。“有效性确认”是一切从客户出发，理解客户的需求，确认所开发的软件是否满足用户真正需求的活动。

测试过程和开发过程都贯穿软件过程的整个生命周期，它们相辅相成、相互依赖，具有同步的关系，而在资源、风险管理上，两者的差异较大。前期，测试过程更多地依赖于开发过程，后期，开发过程更多地依赖于测试过程。

为了做好软件测试计划和随后的工作，促进项目获得成功，必须清楚项目的背景，包括项目的来源、客户和需求等，抓住和软件测试相关的项目要素，并制定软件测试规范。

2

第2章 测试计划

测试计划是一个过程，而不仅仅是一个文档。测试计划有助于测试范围的确定、测试策略的优化和测试风险的规避。

在项目启动之后，就要着手软件项目的计划，包括软件测试计划。软件测试计划是整个开发计划的组成部分，同时，它又依赖于软件组织过程、项目的总体计划、质量文化和方针。在测试计划活动中，首先要确认测试目标、范围和需求，然后制定测试策略，并对测试任务、时间、资源、成本和风险等进行估算或评估。

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化为确定性的内容，该过程最终使得

项目测试负责人能够对资源、成本及进度进行越来越合理、准确的估算。这些估算是软件项目开始时在一个限定的时间框架内做出的，并且随着项目的进展而不断更新。所以，测试计划强调的是一个过程，计划（Planning）的过程，而不仅仅是为了一个文档——“测试计划书”（Test Plan）。

测试计划活动过程伴随着需求文档的审查，而需求文档的评审反过来也有利于测试计划的制定。而且，测试计划必须建立在软件需求定义之上，为软件的质量需求验证和确认活动的开展进行规划和指导。

2.1 产品需求文档审查和评审

产品需求文档的审查和评审，是软件开发中重要的环节之一，也是测试活动之一——静态测试中的需求验证。通过需求评审来保证系统需求在市场需求文档（MRD）或产品需求文档（PRD）及相关的文档中无歧义的描述。不一致、遗漏和错误将会被审查出来并得到改正，而且需求描述文档应该符合已定义的软件过程和软件产品的标准。需求文档的评审，是做好软件测试计划和设计等工作的基础。

2.1.1 需求评审的重要性

软件缺陷并不只是在编程阶段才产生的，需求和设计阶段同样会产生问题，各种阶段性成果中都有可能存在缺陷，包括需求定义文档、设计文档、程序代码等。如果按阶段性工作成果——市场需求文档、规格说明书、系统设计文档、程序代码等存在的缺陷所占的比例来看（即进行缺陷分布的对比分析），结果显示，软件缺陷出现最多的地方是软件需求规格说明书（即软件需求定义），而不是程序代码，如图 2-1 所示。

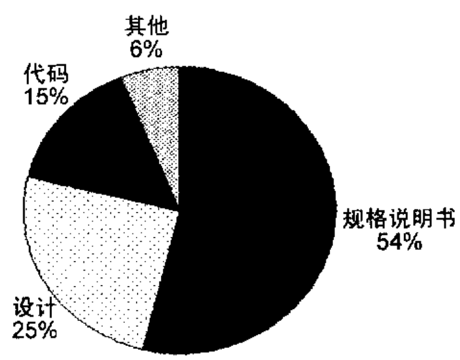


图 2-1 软件缺陷构成示意图

这可能和我们平常的印象不同，人们习惯上认为写代码时容易产生更多的缺陷。为什么软件产品规格说明书是软件缺陷存在最多的地方？主要原因有：

- 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，所以有些特性在当时不可能很清晰。
- 需求变化的不一致性，用户的需求总是在不断地变化，这些变化应该在与需求相关的各类文档中得到描述，但往往被忽视，并容易引起前后文、上下文的矛盾。
- 没有得到开发团队或管理层的足够重视，在规格说明书设计和写作上投入的人力、时间不足。
- 在整个开发队伍中没有进行充分沟通，不同角色之间对需求的理解不一致。

除了横向比较分析之外，我们再来从时间轴进行纵向分析。也许一开始——在需求定义阶段，只是一个很小范围内的潜在错误，但随着产品开发工作的不断推进，小错误会扩散成大错误。就像那种多人传话游戏，第一个人传话给第二个人，从第二个人就开始失真，越往后传其失真就越厉害，最后会牛头不对马嘴。这说明错误发生得越早，造成的危害就越大。

换个角度看，如果错误发现得越迟（如编程阶段），修正这个错误时，可能会要返回到很早的时间（需求定义），这样越到后来，往前返工要重做的事情就越多。所以，修改后期的错误所做的工作要比修改前期的错误大得多。如果错误不能及早发现，那只可能带来越来越严重的后果。缺陷发现或解决得越迟，其带来的成本就越高。

Boehm 在《软件工程经济》一书中断言：平均而言，如果在需求阶段修正一个错误的代价是 1，那么，在设计阶段发现并修正该错误的代价将是 3～6 倍，在编程阶段才发现该错误的代价是 10 倍，在内部测试阶段则会变为 20～40 倍，在外部测试阶段则是它的 30～70 倍，而到了产品发布出去时，这个数字就会高达 40～1000 倍。修正错误的代价不是随时间线性增长，而几乎是呈指数级增长的。图 2-2 就说明了这样一个道理。

仅这一点，就非常容易理解为什么测试人员在需求分析一开始时就介入进去？为什么要尽早发现问题，而且发现得越早越好？道理很简单，缺陷发现得越早、修正得越早，成本就越低。

概括起来，需求评审对软件测试和质量的作用表现在以下几个方面，显示了其重要性。

（1）对软件需求进行正确性的检查，以发现需求定义中的问题，尽早地将缺陷发现出来，降低成本，并使后续过程的变更减少，降低风险。

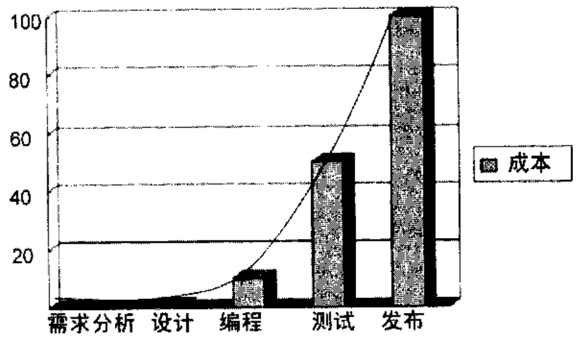


图 2-2 软件缺陷随着时间带来越来越大的成本

（2）保证软件需求的可测试性，即确认任何客户需求或产品质量需求都是明确的、可预见的并被描述在文档中，将来可以用某种方法来判断、验证这种需求或特性是否已得到实现。

（3）通过产品需求文档的评审，与市场、产品、开发等各部门相关人员沟通，使得大家认识一致（on the same page），避免在后期产生不同的理解，引起争吵。

（4）通过产品需求文档的评审，更好地理解产品的功能性和非功能性需求，为制定测试计划和测试方法打下基础，特别是为测试范围、工作量等方面的分析、评估工作积累充足的信息。

（5）在需求文档评审通过后，测试的目标和范围就确定了。虽然此后会有需求的变更，但可以得到有效的控制，这样可降低测试的风险。

概念

评审。IEEE Std 1028-1988给出的定义：评审是对软件元素或者项目状态的一种评估手段，以确定其是否与计划的结果保持一致，并使其得到改进。检验工作产品是否正确地满足了以往工作产品中建立的规范，如需求或设计文档。

评审可以分为管理评审、技术评审、文档评审和流程评审。

（1）**管理评审**就是质量体系评审。通常由最高管理者策划和组织，按规定的时间间隔，以实施质量方针和目标的适应性和有效性为评价基准，对体系文件的适应性和质量活动的有效性进行评价。

（2）**技术评审**对产品以及各阶段的输出内容（阶段性成果、半产品）进行评估，以确保需求规格说明书、设计文档、测试计划和用例等之间保持一致，并得到严格地执行，

正确地开发出软件产品。

（3）**文档评审**对软件过程中所存在的各类文档进行格式、内容评审，检查它们是否符合已有的模板，其文档格式是否符合标准，在内容上则主要检查其一致性、可测试性等。

（4）**流程评审**对软件开发过程的评审。它是针对流程而不是产品的评审，通过对流程的监控，保证项目遵循组织定义的软件过程，严格执行质量保证方针。

2.1.2 测试人员在需求评审中的角色

需求评审，归为静态测试的范畴，包含了文档评审和技术评审双重内容，通常通过正式的评审会议来进行。在正式评审小组中，一般存在多种角色，包括协调人、作者、评审员、用户代表等。为了保证评审的质量和效率，需要精心挑选评审员。首先要保证不同类型的人员都要参与进来，否则很可能会漏掉很重要的需求；其次在不同类型的人员中要选择那些真正与系统相关的、对系统有足够了解的人员参与进来，否则很可能使评审的效率降低或者最终不切实际地修改了系统的范围。而测试人员，主要起着评审员的角色，检查需求文档是否按照相应的模板要求严格进行，需求定义是否合理和清楚等。

测试人员作为评审员，在评审过程中，要注意下列事项：

- 明确自己的角色和责任。
- 熟悉评审内容，为评审做好准备，做细做到位。
- 在评审会上，针对问题阐述观点，而不是针对个人。
- 可以分别讨论主要的问题和次要的问题。
- 在会议前或者会议后可以就存在的问题提出自己建设性的意见。
- 提高自己的沟通能力，采取适当的、灵活的表述方式。
- 对发现的问题跟踪下去。

在软件需求评审过程中，一般来说，作者事先主动地向相关利益者介绍需求分析、定义的背景和说明相关的细节，以征集大家的意见。在介绍之前，应先将文档发给相关利益者。测试人员作为评审员，在获得需求文档（MRD，PRD）后应仔细地阅读，将阅读中发现的问题、不明白的地方一一记下来，通过邮件发给文档的作者，或通过其他形式（面对面会谈、电话、远程互联网会议等）进行交流。其中重要的一点就是要善于提问，要向自己问问题。

- 这些需求都是用户提出来的？有没有画蛇添足的需求？

- 有没有漏掉什么需求？有没有忽视竞争对手的产品特性？
- 需求文档中，是否正确地描述了需求？
- 我的理解和他们（MRD、PRD的作者）的理解一致吗？

通过交流，大家达成一致的认识和理解，并修改不正确、不清楚的地方。在各种沟通形式中，自然、面对面的沟通形式效率最好，但是在口头交流达成统一意见后，最后须通过文档、邮件或工作流系统等记录下来，作为备忘录。

如果通过一些非正式的形式（临时评审、走查等）不能很好地完成需求分析的评审，就必须通过正式形式（集体/小组评审会议）来完成这一工作。对于比较大型、需求改动较大等的项目来说，一次评审会议也许还不够，还要通过2~3次甚至更多次的评审会议才能最后达成一致。

评审是否完成是以需求文档获得多方签发（sign-off）或签字通过为标志的。这不应该只体现在“签字”形式上，更重要的是达到下面的结果：

- 所有参与方达成一致。
- 已发现的问题被阐述清楚、被修正。

概念

评审的形式

（1）相互评审、交叉评审（Pear-to-pear Review），甲和乙在一个项目组，处在一个领域，但工作内容不同，甲的工作成果交给乙审查，乙的工作成果交给甲审查。相互评审是最不正式的一种评审形式，但应用方便、有效，被普遍使用。

（2）轮查（Pass-round），又称分配审查方法，是一种异步评审方式。作者将需要评审的内容发送给各位评审员，并收集他们的反馈意见。

（3）走查（Walkthrough），产品的作者将产品在现场向一组同事介绍，描述产品有怎样的功能、结构，从头到尾走一遍，以收集大家的意见。希望参与评审的其他同事可以发现产品中的错误，并能进行现场讨论。这种形式介于正式和非正式之间，其应用普遍。

（4）小组评审（Group Review），通过正式的小组会议完成评审工作，是有计划的和结构化的评审形式。评审定义了评审会议中的各种角色和相应的责任，所有参与者在评审会议的前几天就拿到了评审材料，并对该材料进行了独立研究。

（5）审查（Inspection）。审查和小组评审很相似，但更为严格，是最系统化、最严密的评审形式，包含了制定计划、准备和组织会议、跟踪和分析审查结果等。

2.1.3 需求评审的标准

测试、评审依据的标准是非常重要的。如果没有依据，就无法判断对与错，很难更准确地、更快速地发现问题。所以在进行软件评审、测试时，制定一个很好的质量评判标准是必要的。

对系统需求的评审着重于审查对用户需求描述的解释是否完整、准确。根据 IEEE 建议的需求说明标准，对于系统需求进行审查的质量因素有如下内容。

（1）**正确性**：检查在任意条件下软件系统需求定义及其说明的正确性。如需求定义是否符合软件标准、规范的要求？是否所有的功能都有明确的目的，是否存在对用户无意义的功能？每个需求定义是否都合理，经得起推敲？所采用的算法和规则是否科学、成熟和可靠？有哪些证据说明用户提供的规则是正确的？是否正确地定义了各种故障模式和错误类型的处理方式？对设计和实现的限制是否都有论证？

（2）**完备性**：涵盖系统需求的功能、性能、输入/输出、条件限制、应用范围等方面，覆盖率越高，完备性越好。通过增强创造力的方法避免思维的局限性，能全面地考虑各种各样的应用场景或操作模式，以提高完备性。如是否有漏掉的功能？是否有漏掉的输入、输出或条件？是否考虑了不同需求的人机界面？需求定义是否包含了有关文件（如质量手册、配置计划等）中所规定的特定需求？功能性需求是否覆盖了所有非正常情况的处理，出现异常情况时系统如何响应？是否识别出了所有与时间因素有关的功能？是否识别并定义了在未来潜在变化的需求？是否定义和说明了系统输入和输出的来源、类型、值域、精度、单位和格式等？

（3）**易理解性**：需求文档的描述性被理解的难易程度，包括清晰性。如需求描述是否足够清楚和明确，使其已能作为开发设计说明书和功能性测试数据的基础？是否将系统的实际需求内容和所附带的背景信息分离开来？每一个需求是否只有一种解释，语言是否有歧义性？功能性需求的描述结构化、流程化是否良好？是否使用了形式化或半形式化的语言？需求定义是否包含了实现的细节，是否过分细致了？

（4）**一致性**：包含了兼容性，如所定义的需求之间是否一致，是否有冲突和矛盾？是否使用了标准术语和统一形式？使用的术语是否是唯一的？同义词、缩略语等的使用在全文中是否一致并事先已予以说明？所规定的操作模式、算法和数据格式等是否相互兼容？

是否说明了系统中软件、硬件和其他环境之间的相互影响？

（5）**可行性**：需求中定义的功能是否具有可执行性、可操作性等。如需求定义的功能是否能通过现有的技术实现？所规定的模式、数值方法是否能解决需求中存在的问题？所有的功能是否都能够在某些非常规条件下实现？是否能够达到特定的性能要求？

（6）**易修改性**：对需求定义的描述易于修改的程度，如是否有统一的索引、交叉引用表？是否采用了良好的文档结构？是否有冗余的信息？

（7）**易测试性**（可验证性）：所定义的功能正确性是否能被判断？系统的非功能需求（如性能、可用性等）是否有验证的标准和方法？输入、数出的数据是否有清楚地定义从而容易验证其精确性？

（8）**易追溯性**：每一项需求定义是否可以确定其来源？是否可以根据上下文找到所需要的依据或支持数据？后续的功能变更是否都能找到其最初定义的功能？功能的限制条件是否可以找到其存在的理由？

对用户的需求进行评审时，上述标准就是评审的依据，逐字逐句地审查需求规格说明书的各项描述。需求规格的描述，不仅包括功能性需求，而且包括非功能性需求。例如，系统的性能指标描述应该清楚、明确，而不是给一个简单的描述——“每一个页面访问的响应时间不超过 3 秒”，业务要求通常用指定响应时间的非技术术语表示性能，如下所示：

系统能够每秒接受 50 个安全登录，在正常情况下或平均情况下（如按一定的时间间隔采样）Web 页面刷新的响应时间不超过 3 秒。在定义的高峰期间，响应时间也不得超过 12 秒。年平均或每百万事务的错误数须少于 3.4 个。

业务要求通常用指定响应时间的非技术术语表示性能。有了更专业、更明确的性能指标，就可以对一些关键的使用案例进行研究，以确定在系统层次如何保证该要求得到实现的结构、技术或方式。在多数情况下，将容量测试的结果作为用户负载的条件，即研究在用户负载较大或最不利情况下保证系统的性能。如在这种情况下，系统的性能有保证，在其他情况下就不会有问题。

要点

需求评审的方法

（1）**分层次评审**。用户的需求是分层次的，对不同层次的需求，其描述形式是有区别的，参与评审的人员也是不同的。一般，可以分成如下的层次：

- **目标性需求**指整个系统需要达到的业务目标，是最高层次的、基本的需求，是企业的高层管理人员所关注的。如果让具体的操作人员去评审目标性需求，容易导致“捡了芝麻，丢了西瓜”的现象。
- **功能性需求**指整个系统需要实现的功能和任务，是目标之下的第二层需求，是企业的中层管理人员所关注的。
- **操作性需求**指完成每个任务具体的人机交互（UI）需求，是企业的具体操作人员所关注的。如果让高层的管理人员也去评审那些操作性需求，无疑是一种资源的浪费。

（2）**分阶段评审**。应该在需求形成的过程中进行分阶段的多次评审，而不是在需求最终形成后才进行仅有的一次评审。分阶段评审可以将原本需要进行的大规模评审拆分成各个小规模的评审，这样就降低了需求分析返工的风险，提高了评审的质量。比如可以在形成目标性需求时完成第一次评审，在形成系统功能框架时再进行一次评审。当功能细化成几个部分后，可以对每个部分分别进行评审，并对关键的非功能特性进行单独的评审。最后对整体的需求进行全面的评审。

2.2 项目的测试需求和任务

在掌控了软件项目的背景，又参与了软件系统的需求评审后，进一步了解软件特定的功能性和非功能性需求，从而可以分析软件系统的质量需求，确定项目的测试需求和基本任务。从另一方面讲，在制定测试计划前，也必须清楚测试的具体需求和任务，知道哪些测试目标优先级高，哪些目标优先级低，要完成哪些相应的测试任务才能确保目标的实现，然后才能分析测试范围和估算测试的工作量。测试需求是测试设计和开发测试用例的基础，测试需求分解得越细，对测试用例的设计质量的帮助越大，详细的测试需求还是衡量测试覆盖率的重要指标。只有在做好测试需求的基础上，才能规划具体项目所需的资源、时间以及所存在的风险等。

测试目标取决于软件质量的需求，所以测试目标可以被分为功能性测试（包括 UI）需求、非功能性的系统测试需求。功能测试需求比较容易确定，而非功能性的测试需求相对难以确定，需要更多的讨论和分析。

2.2.1 确定软件功能测试需求

我们知道功能测试需求比较容易确定，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软件各方面功能的使用要求，包括用户界面的友好性。功能测试的基本目标是

从用户需求出发，尽可能尽早地发现不满足用户需求、与产品规格说明书不一致的所有问题。通过测试，以最小的代价和最低的风险确保下列功能性质量需求得到很好的满足（虽然不是100%的满足，至少达到99.5%以上的覆盖率），这些需求主要有：

- 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。
- 每一项功能能正常运行、输出结果正确。
- 能处理各种不正常的操作，对异常数据的输入可以进行提示、容错处理等。
- 系统的界面清晰、美观。
- 菜单、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，如测试最大输入的文字数、单双字节、特殊符号等。
- 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。

在进行功能测试需求时，针对项目的不同类型，功能测试的侧重点是有差异的，但这种差异在功能测试需求定义上差别不大，而在非功能性测试需求上这种差异很明显。对于纯客户端软件，要增加对安装卸载、显示特性和操作方便性等方面的要求，如：

- 可以完成第一次安装，也可以进行多次安装，并有安装提示。
- 安装过程中，可以倒退一步或几步，临时取消安装。
- 安装后可进行配置，能解决有冲突的端口地址。
- 能卸载干净，但不应该删除使用本软件后的用户个人数据。

用户界面（User Interface， UI）是和用户进行交互的窗口。仅从这一点，就可以清楚地知道用户界面友好程度的重要性。用户界面是否友好直接影响用户对软件产品或软件服务的满意度，即我们经常提到的用户体验，用户界面设计就是给用户一个良好的体验，不仅使用软件简单、方便和明了，而且心情舒畅、愉悦。对于 Web 应用，更强调网页内容和文字表述，但这些往往是开发人员容易忽视的地方。对于开发人员来说，注意力常常集中在功能的实现上。文字不仅误导用户的操作或影响用户的体验，而且有时可能会引起法

律方面的问题。测试人员应确保内容表达符合习惯，更专业、流畅，有时需要招聘1~2个语言学（文学、中文、英文、日文等）专业的人员参加测试队伍。在UI上，显示功能的要求有：

- 通用框架、浮动窗口和文字等整体上布局合理、位置恰当。
- 文字没有乱码、换行正常，而且内容格式、顺序正确。
- 文字标记和超链接可以打开和跳转成功。
- 色彩搭配要协调，要形成对比强烈的色彩效果，也要恰到好处。

2.2.2 非功能性的系统测试需求

对于非功能性的系统测试，涉及到非功能性的质量需求，包括系统性能、安全性、兼容性、扩充性等，其测试需求会因不同的项目类型差异比较大，下面就做一个简单的分析。

- 纯客户端软件，如字处理软件、下载软件、媒体（音频/视频）播放软件等在系统测试要求上是最低的，对性能、容错性、稳定性等有一定的要求，如占用较少的系统资源（CPU和内存），而且能运行在不同的操作系统上，一般分为Windows版、Linux版和Mac版等。在Windows上要支持Windows NT、Windows 2000、Windows XP和Windows Vista等。
- 纯Web（B/S）应用系统，如门户网站、个人博客网站、网络信息服务等在系统测试上要求较高，特别强调性能和可用性，对安全性有一定的要求，主要是保证数据的备份和登录权限。性能要求好，可以允许大量并发用户的访问，而且用户在任何时刻可以访问，即每周7天，每天24小时（7×24）运行。
- 客户端/服务器（C/S）应用系统，如邮件系统、群件或工作流系统、即时消息系统等在系统测试需求上与Web应用系统接近，也可能出现大量并发访问的用户，但安全性相对好写，客户端是特定的开发软件，相对于浏览器来说，对端口、协议等的限制比较容易做到。
- 大型复杂企业级系统涉及面广、集成性强，包括了B/S、C/S、数据库、目录服务、服务器集群、XML接口等各个子系统。在系统测试需求上，这类系统要求最高，不论是在性能、可用性方面，还是在安全性、可靠性等方面，都有很高的要求。

对于企业级应用系统来说，存在着不同的应用模式，其系统的架构也不一样，可以分为“以功能为中心、以数据库为中心和以业务逻辑（工作流）为中心”等，在进行系统测试时，所设定的目标也有一定的区别。

- 以功能为中心的系统，强调模块化的低耦合性和高内聚性，这类系统的可扩充性、维护性要求很高。
- 以数据库为中心的系统，强调数据处理的性能、正确性和有效性，使数据具有良好的一致性和兼容性，同时，确保数据的安全性，包括数据的存储、访问控制、加密、备份和恢复等。
- 以应用逻辑（工作流）为中心的系统，强调灵活、流畅和时间性，系统的可配置性强，接口规范，如采用 XML 统一各工作流构件的输入和输出。

除此之外，还有其他一些因素的影响，如项目的周期性和依赖性等。如果项目是一次性的，对可扩充性、可移植性等要求低，而长期性的项目（如产品开发）对可扩充性、可移植性要求就很高。

例如，像雅虎日历这样的 Web 服务软件的非功能性系统测试需求有：

（1）**性能要求**，如在正常使用情况下，系统的响应时间为 3～5 秒，即一个页面能在 3～5 秒内显示出来，即使在使用高峰期（如上下班时间）系统的响应时间也不应超过 15 秒。还有其他的一些性能要求。

- 多人同时登录（并发用户活动）、设置活动时，页面的响应速度要在 3 秒之内。
- 通过页面进行搜索时，查询时间应控制在 5 秒以内。
- 设置共享时、用户更新活动信息时，是否能快速同步，即在另一共享好友处即刻显示更新过的信息。
- 当活动/事件达到一定数量（200～1000）时，页面响应速度要在 5 秒之内。
- 当循环会议较多时，页面的处理速度正常（5 秒之内）。

（2）**容错处理**，如业务处理过程中中断事务时，系统能保存当前状态，在某个时刻可以恢复操作。又如在用户访问页面、提交表单过程中，中断了网络连接，系统能诊断出故障并给予正确的提示。

（3）**兼容性要求**，指客户端可以在不同的操作系统、浏览器上运行。操作系统包括 Windows、Mac、Solaris、Linux 等，浏览器包括 IE、FireFox、NetScape、Mozilla 和 Safari 等。

（4）**配置要求**，要支持不同的 Internet 连接速度（宽带网、ADSL、ISDN 等）、屏幕分辨率（800×600、1027×768 等）、打印等。例如页面居中，就比较容易适合不同的分辨率。如果页面靠左或靠右对齐，当使用过高分辨率时就不好看。直接打印所显示的页面一般不好看，理想情况是提供一个专供打印的页面（干净、整齐）。

（5）**安全性要求**，包括各种登录模式的安全性验证、Cookie 和 Session 的有效期验证和日志文件等。系统能否有很好的机制抵御外部的攻击，如黑客试图通过发送大量数据包来攻击服务器，系统能自动检测出不正常，提示系统管理员采取措施，甚至自动屏蔽相应的 IP 地址或端口。安全性要求还包括数据访问权限设置、数据备份和恢复等。再者，服务器上的目录设置要正确并禁止用文件列表，严格把关某些可执行文件的存取权限。这样，用户就不能根据 URL 来推测相应的路径，进而浏览该目录下的文件，从而执行或查看某些文件了。如果不是这样，系统和数据是很不安全的。

（6）**可靠性**，指系统能长期使用，保持功能正常、性能稳定。如服务器内存没有泄漏，有足够的硬盘空间，硬盘空间使用率达到 70%或 80%时有提示等。

2.2.3 软件即服务的测试需求

软件即服务（Software as a Service，SaaS）的应用模式越来越多，软件运行的服务质量（QoS）要求也越来越高，这对软件测试提出了新的要求。

QoS 要求是指定某些功能（如性能、可用性、可伸缩性和可维护性）系统特性的技术规范。例如，如果服务必须全年每天 24 小时可用，则可用性要求必须解决或满足此业务要求。此前，已对软件性能和可靠性做了较多的介绍，所以下面将主要讨论可用性、安全性、可伸缩性和可维护性等。

1. 可用性

可用性是指系统正常运行的能力或程度，在一定程度上也是系统可靠性的表现。可用性一般用正常向用户提供软件服务的时间占总时间的百分比来表示，即：

$$\text{可用性} = \text{正常运行时间} \,/\, (\text{正常运行时间} + \text{非正常运行时间}) \times 100\%$$

系统非正常运行时间可能是因硬件、软件、网络故障或任何其他因素（如断电）所致，这些因素能让系统停止工作或连接中断不能被访问或性能急剧降低不能使用软件现有的服务等。

可用性指标一般要求达到 4 个或 5 个“9”，即 99.99% 或 99.999%。

- 如果可用性达到 99.99%，对于一个全年不间断（7x24 的方式）运行的系统，意味着全年（525 600 分钟）不能正常工作的时间只有 52 分钟，不到一个小时。
- 如果可用性达到 99.999%，意味着全年不能正常工作的时间只有 5 分钟。

所以一个系统的可用性达到 99.999%，基本能满足用户的需求。当然，不同的应用系统，可用性要求是不一样，非实时性的信息系统或一般网站要求都很低，可能在 99%和

99.5%之间，而对一些军事系统，则要求很高，如美国防空雷达系统全年失效时间不超过两秒，可用性高达7个“9”之上，达99.999994%.

2. 可伸缩性

可伸缩性是增加系统容量的能力，从而使系统可以支持来自现有用户或扩大的用户群体的额外负载。可伸缩性通常要求在对部署体系结构的设计不做修改的情况下增加资源以满足系统增加的容量。

不必在QoS要求中指定可伸缩性要求，除非业务要求中对预测的部署增长做了明确的说明。不过，在软件部署设计阶段，即使未指定可用性 QoS 要求，部署体系结构也应添加一定余量，以应付可能的系统规模扩展。

3. 安全性要求

安全性是一个复杂的主题，涉及到部署系统的各个级别。安全性要求分析，包括确定可能的或潜在的各类安全威胁和找到处理这些威胁的策略，即：

- 确定关键（有形的和无形的）资产，并找到对这些资产的威胁。
- 确定使组织暴露于可能带来风险威胁的薄弱环节。
- 开发减轻组织风险的安全规划。

分析安全要求应由软件组织的各方面风险承担者参与，包括管理员、业务分析师和信息技术人员等。通常，组织会指定一个安全结构设计师来领导安全措施的设计和实现。

4. 可维护性要求

可维护性是指对部署系统进行维护的难易程度，其中包括系统监视、系统出现故障的修复、系统功能的增强、系统数据的更新和备份、系统软硬件的升级等任务执行的难易程度。可维护性和可用性之间的关系密切，存在一定的依赖性。只有具有良好的可维护性，才可能保证系统运行的高可用性。

服务级别协议（SLA）指定了最低性能要求，以及未能满足此要求时必须提供的客户支持级别和程度。与 QoS 要求一样，服务级别要求源自业务要求，对要求的测试条件及不符合要求的构成条件均有明确规定，并代表着对部署系统必须达到的整体系统特性的担保。服务级别协议被视为合同，所以必须明确规定服务级别要求。如表2-1所示。

表2-1 影响 QoS 要求的系统特性

系统质量	说 明
性能	指按用户负载条件对响应时间和吞吐量所作的度量

续表

系统质量	说 明
可用性	指对系统资源和服务可供最终用户使用的程度度量，通常以系统的正常运行时间来表示
可伸缩性	指随时间推移为部署系统增加容量（和用户）的能力。可伸缩性通常涉及向系统添加资源，但不应要求对部署体系结构进行更改
安全性	指对系统及其用户的完整性进行说明的复杂因素组合。安全性包括用户的验证和授权、数据的安全以及对已部署系统的安全访问
潜在容量	指在不增加资源的情况下，系统处理异常峰值负载的能力。潜在容量是可用性、性能和可伸缩性特性中的一个因素
可维护性	指对已部署系统进行维护的难易度，其中包括监视系统、修复出现的故障以及升级硬件和软件组件等任务

2.2.4 各个阶段的测试任务

软件测试贯穿软件产品开发的整个生命周期，从产品的需求分析审查到最后的验收测试，直至软件发布。从测试实际的前后过程来看，软件测试的过程是由一系列不同的测试阶段组成的，这些软件测试的步骤分为：需求分析审查、设计审查、单元测试、集成测试（组装测试）、功能测试、系统测试、验收测试、回归测试（维护）等，各阶段的输入和输出标准如表 2-2 所示。

表 2-2 各测试阶段输入和输出标准

阶 段	输入和要求	输 出
需求分析审查（Requirements Review）	市场/产品需求定义、分析文档和相关技术文档。 要求：需求定义要准确、完整和一致，真正理解客户的需求	需求定义中的问题列表，批准的需求分析文档。 测试计划书的起草
设计审查（Design Review）	产品规格设计说明、系统架构和技术设计文档、测试计划和测试用例。 要求：系统结构的合理性、处理过程的正确性、数据库的规范化、模块的独立性等。清楚定义测试计划的策略、范围、资源和风险，测试用例的有效性和完备性	设计问题列表、批准的各类设计文档、系统和功能的测试计划和测试用例。 测试环境的准备

续表

阶 段	输入和要求	输 出
单元测试 （Unit Testing）	源程序、编程规范、产品规格设计说明书和详细的程序设计文档。 要求：遵守规范、模块的高内聚性、功能实现的一致性和正确性	缺陷报告、跟踪报告；完善的测试用例、测试计划。 对系统功能及其实现等了解清楚
集成测试 （Integration Testing）	通过单元测试的模块或组件、编程规范、集成测试规格说明和程序设计文档、系统设计文档。 要求：接口定义清楚且正确，模块或组件一起工作正常，能集成为完整的系统	缺陷报告、跟踪报告；完善的测试用例、测试计划；集成测试分析报告。 集成后的系统
功能验证 （Functionality Testing）	代码软件包（含文档），功能详细设计说明书；测试计划和用例。 要求：模块集成功能的正确性、适用性	缺陷报告、代码完成状态报告、功能验证测试报告
系统测试 （System Testing）	修改后的软件包、测试环境、系统测试用例和测试计划。 要求：系统能正常地、有效地运行，包括性能、可靠性、安全性、兼容性等	缺陷报告、系统性能分析报告、缺陷状态报告、阶段性测试报告
验收测试 （Acceptance Testing）	产品规格设计说明、预发布的软件包、确认测试用例。 要求：向用户表明系统能够按照预定的要求那样工作，使系统最终可以正式发布或向用户提供服务。用户要参与验收测试，包括 α 测试（内部用户测试）、β 测试（外部用户测试）	用户验收报告、缺陷报告审查、版本审查、最终测试报告
版本发布 （Release）	软件发布包、软件发布检查表（清单）	当前版本已知问题的清单、版本发布报告
维护 （Maintance）	变更的需求、修改的软件包、测试用例和计划 要求：新的或增强的功能正常、原有的功能正常，不能出现回归缺陷	缺陷报告、更改跟踪报告、测试报告

2.3 测试范围分析和工作量估计

在确定了测试需求之后，为了达到这些目标，就要执行相应的测试任务。在执行测试任务的过程中，需要分析测试的范围，根据测试的范围、任务和其他条件，决定测试的工作量和测试环境。基于质量需求和测试的工作量、测试环境、产品发布的设想时间等要求，就可以确定测试进度和所需的测试资源，或者基于现有的测试资源，来决定测试的日程表。

在测试总量不变、质量需求不变的情况下，人力资源和进度是相互制约的，增加人力资源，可以缩短测试周期；人力资源不足，可以通过延长测试周期来完成所需的测试工作。资源和进度的关系也不是简单的线性关系，后面会有更仔细的分析。

2.3.1 功能测试范围的分析

在分析测试范围时，一般先进行功能测试的范围分析，然后再进行非功能性测试的范围分析。对于功能测试，可以借助流程图和框图等进行简单、有效的分析。在面向对象的软件开发中，也可借助 UML 用例图、活动图、协作图和状态图来进行功能测试范围分析。这里通过前述两个实例来说明如何做简单功能测试的范围分析。

1. Google Talk 客户端软件

基本功能测试需要根据具体功能的逻辑、黑盒测试方法等进行测试用例的设计，并考虑用户的习惯思维，把功能划分成若干个模块：

- 登录与注销。
- 主面板功能设置。
- 文字聊天。
- 语音聊天。
- 用户设置。
- 消息/呼叫的提示。
- 文件与语音邮件发送。

分别进行测试，然后进行集成测试。让我们简单地分析一下功能测试的范围。

- 安装与卸载。要求安装方便，卸载后无遗留文件，以及对用户电脑无任何副作用。
- 注册，登录与注销。用户注册 Google 账号，在客户端登录，其他好友能看到其登录后上线和注销后下线等相关状态信息。

- 好友管理。可以方便地添加、删除好友，Google Talk 会自动将最常联系的好友排在列表的最上面，也可以用查找框快速地查找好友。
- 文字聊天。聊天窗口分菜单、消息显示、消息输入几个部分，Talk 的功能相对简单，重点要求是方便地在用户之间传递消息，以及不同输入语言的正确显示。
- 语音通话。分为呼叫、通话、结束三个阶段。要求控制反应灵敏，语音清晰连续。
- 邮件管理。Talk 和 Gmail 做到了很好的整合，聊天记录也是可以保存在 Gmail 邮件服务器上的，在客户端我们重点只看邮件提醒的功能。

2. 雅虎日历的测试范围

雅虎日历的功能可以分为 4 部分——日历页面显示、事件和待办事项添加功能、搜索功能和选项设置，如图 2-3 所示。测试关键点在于确保雅虎日历的组件能够准确、安全、无错误地实现事件定制及浏览、预约提醒、日历共享、个性化设定等功能。而对具体的一个功能的测试，其测试范围一般包括输入、编辑、查询、显示等。如在数据输入方面，要测试最大输入的文字数、双字节文字、特殊符号等各种情况，还要测试输入区域支持拷贝、粘贴的情况。

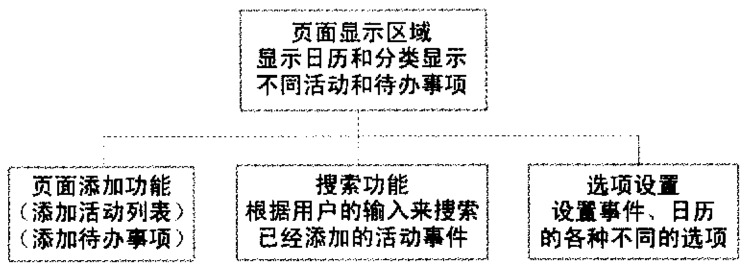


图 2-3　雅虎日历的功能框图

对于 Web 的功能分析，也可以从页面帧结构、布局来进行分析。例如雅虎日历的显示区域设定为 4 个子区域：

- 顶部是导航栏，调用主要的功能菜单。
- 左边区域，显示日历、功能按钮和搜索输入框等。
- 右边主区域，按日、周、月、年等显示事件和活动的时间和内容。
- 右边底部区域是快速活动添加区。

如图 2-4 所示。在显示上，要测试日历和各个分类框架显示格式是否正确，排序是否正确，文字标记和超链接是否可以打开和跳转成功。重点要测试右边的显示区域，要求在输入很多且很长的待办事项的情况下，显示内容可以自动换行，字符没有乱码和截断，相

互之间的日、周、月、年表单之间相互跳转没有问题。

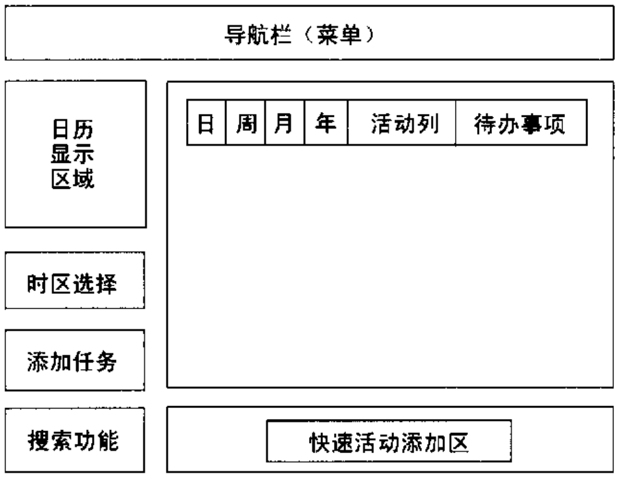


图 2-4 雅虎日历的显示区域分析

在进一步进行功能分析后，可以了解更具体的功能测试范围。

（1）登录，是最基本的功能需求，对用户身份进行合法性验证，对各种登录模式的安全性验证，包括 Cookie 和 Session 的有效期验证。

（2）活动定制功能。

- 定制会议后，能正确显示。
- 循环会议可以成功指定与显示。
- 会议可以被成功修改。
- 跨日会议的准确性，以及夏令时的准确显示。
- 与其他日历的兼容，如导入/导出数据是否正常。

（3）提醒功能。根据制定活动时的设置能够在活动之前发送提醒到雅虎通、电子邮件地址、移动设备。

（4）共享功能。可以根据用户的权限设置来决定是否让他人看到自己的日历。

（5）时间表能将用户所注意或关心事件的时间和用户的个人行事整合起来，直接了解效率手册中的重要活动，并可查看朋友的效率手册、俱乐部的效率手册、运动队的比赛日程以及其他更多内容。按照合并视图或栏视图方式显示。

3. 一般性的Web测试项目

（1）用户登录，登录的用户名、口令能否保存？口令忘了，能否找回来？允许登录失败的次数是否有限制？口令字符有没有严格要求（如长度、大小写、特殊字符）？是否硬性规定经过一段时间后必须改变口令？

（2）站点地图和导航条。每个网站都需要站点地图，让用户一看就能了解网络内容，而且当新用户在网站中迷失方向时，站点地图可以引导用户进行浏览，找到所想访问的内容。需要验证站点地图每一个链接是否存在而且正确，有没有涵盖站点上所有内容的链接。是否每个页面都有导航条？导航条是否一致、直观？

（3）链接，去正确地方，即链接地址正确，并能显示正常、自然，不要给人突然的感觉。

（4）表单，各项输入是必需的、合理的，各项操作正常，对于错误的输入有准确、适当的提示，并完成最后的提交。提交后，返回提交内容的显示，使用户放心。如用户通过表单进行注册，能输入用户名、口令、地址、电话、爱好等各种信息，当格式、内容不对或不符时，及时给予提示，在用户提交信息后，进一步检查各项内容的正确性，然后写入数据库、返回注册成功的消息。

（5）数据校验，根据业务规则和流程对用户输入数据进行校验，是许多系统不可缺少的。通过列表选择、规则提示或在线帮助，能很好解决这问题。

（6）Cookie，在Web应用中到处可见，用来保存用户注册、访问和其他本地客户端信息，所保存的信息要加密，并能及时更新。Cookie被删除了，能被重建。

（7）Session，是否安全、稳定，而且占用较少的资源。

（8）SSL、防火墙等的测试。使用了SSL，浏览器地址栏中URL的“http:”就变为“https:”了，服务器的连接端口号则由80变为433，应用程序接口API也要和页面保持一致。防火墙支持更多的设置，包括代理、验证方式、超时等。

（9）接口测试，与数据库服务器、第3方产品接口（如电子商务网站信用卡验证）的测试，包括接口错误代号和列表。

4. UI测试的需求

- 过分地使用粗体字、大字体和下画线可能会让用户感到不舒服。
- 背景颜色使用不当，虽然美观，但不易阅读，内容阅读才是最重要的。通常来说，文字和背景对比较大比较适宜，背景浅淡则文字采用深颜色；背景深黑则文字采

用亮色。对比要采用适当，和谐往往更容易被人接受。

- 每一张图片都是必要的，位置和大小合适，采用了 JPG 和 GIF 格式，而且和文字吻合。
- 不要因为使用图片而使窗口和段落排列古怪或者出现孤行。
- 表格显示是否清晰，必要的数据能否在一个页面显示？翻页、水平移动是否流畅等？表格里的文字是否能折行且保持内容完整，或者使用表格栏宽度设置协调？

2.3.2 系统测试范围的分析

系统测试是对已经集成好的软件系统进行彻底的测试，以验证软件系统的整体性能等是否满足其产品设计规格所指定的要求。

系统测试分析包括性能测试、兼容性测试、适用性测试和安全性测试等，还要进行第3方产品集成测试、可靠性测试等。

1. 一般系统测试的需求

（1）**容错处理**，如当业务处理过程中中断事务时，系统能保存当前状态，在某个时刻可以恢复操作。又如在用户访问页面、提交表单的过程中，中断了网络连接，系统能诊断出故障并给予正确的提示。

（2）**兼容性要求**，如在不同的操作系统（如 Windows、Mac、Solaris、Linux 等）、浏览器（如 IE、FireFox、NetScape、Mozilla 和 Safari 等）上运行。

表 2-3 兼容性测试的环境组合

OS / Browser	Windows 2000	Windows XP	Windows 2003 Server	Windows Vista
IE6.0	√		√	
IE.7.0		√		√
Firefox 1.5	√		√	
FireFox 2.0		√		√
Mozilla 1.7	√	√	√	

（1）**配置要求**，支持不同的 Internet 连接速度（宽带网、ADSL、ISDN 等）、屏幕分辨率（800×600、1027×768 等）、打印等。例如页面居中，就比较容易适合不同的分

辨率。如果页面靠左或靠右对齐，当使用过高分辨率时就不好看。直接打印所显示的页面一般不好看，理想情况下是，提供一个专供打印的页面（干净、整齐）。

（2）**性能要求**，在正常使用情况下，系统的响应时间为3～5秒，即一个页面能在3～5秒内显示出来，即使在使用高峰期（如上下班时间）系统的响应时间不应超过15秒。

（3）**安全性要求**，系统能否有很好的机制抵御外部的攻击，如黑客试图通过发送大量数据包来攻击服务器，系统能自动检测出不正常，提示系统管理员采取措施，甚至自动屏蔽相应的IP地址或端口。安全性要求还包括数据访问权限设置、数据备份和恢复等。再者，服务器上目录设置要正确并禁止文件列表，严格某些可执行文件的存取权限。这样，用户就不能根据URL来推测相应的路径，进而浏览该目录下的文件，从而执行或查看某些文件了。如果不是这样，系统和数据是很不安全的。

（4）**可靠性**，系统能长期使用，保持功能正常、性能稳定。如服务器内存没有泄漏、有足够的硬盘空间，硬盘空间使用率达到70%或80%时有提示。

（5）**日志文件**，服务器的日志文件能正常生成、追加内容，当日志文件很大时，可以创建新的日志文件。日志可记录下所有的事务处理、用户操作进出的时间、用户访问的IP地址、系统管理员所做的任何改动等。

2. Google Talk 客户端软件性能测试需求

对于客户端软件的性能，主要从资源（CPU、内存等）占有率、操作过程流畅等考虑，从而给出具体、准确的性能指标。

- 所占计算机资源：Google Talk 大约只占系统1.5MB的内存，在好友数量以及对话窗口非常多的情况下，仍能使计算机CPU、内存的占用处在一个合理的范围内。例如，如果内存在5MB之内，它占用CPU最大不超过20%（正常运行时间，不是软件启动的短暂时间）。
- 网络资源占用。Google Talk 的语音占带宽大约24~32 KB，文字部分就更少。
- 尤其是在较差的网络条件下，语音与文字聊天应该能保持流畅。
- 发生故障后能恢复，在出现网络中断时，程序应该能自动或提示重连。
- 在长时间运行（7×24小时）下，不出现程序崩溃、内存泄漏等现象。

3. 雅虎日历的 Web 性能测试

目的是准确展示网络带宽、延迟、负载和TCP端口的变化是如何影响用户的响应时

间的。客户端的响应时间测试，我们可以检测在添加大批量活动数据后的页面响应速度。比如，往雅虎日历中添加大批量的待办事项，然后查看页面浏览的响应时间。

压力测试是测试系统的限制和故障恢复能力，也就是测试 Web 应用系统会不会崩溃，在什么情况下会崩溃，测试出系统能承受多少用户的并发操作。雅虎日历压力测试的区域主要包括建立雅虎日历新事项时提交的表单、登录雅虎日历和其他信息时的传输页面等。

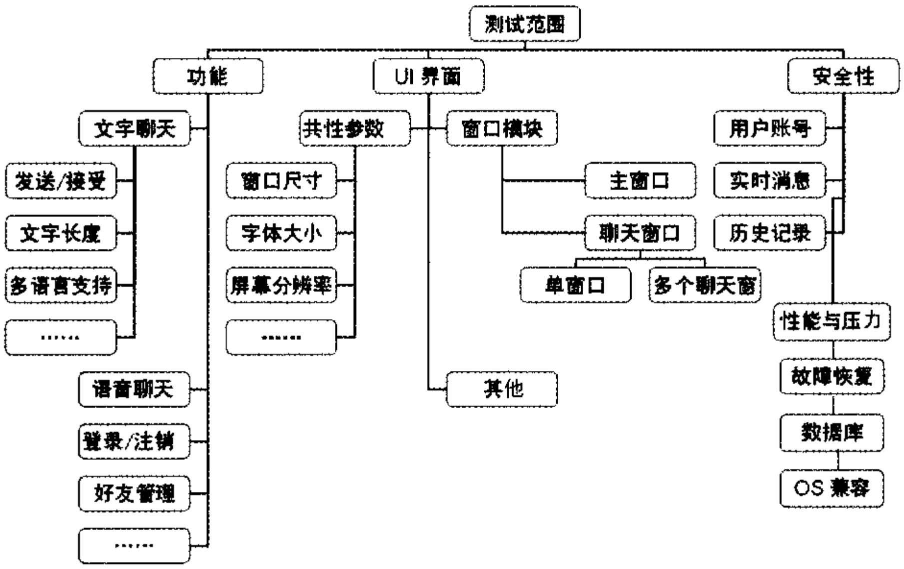


图 2-5 测试范围的整体分析示意图

2.3.3 工作量的估计

测试的工作量是根据测试范围、测试任务和开发阶段来确定的。测试范围和测试任务是测试工作量估算的主要依据。如何确定测试范围已在上一节做了充分的讨论，可以根据产品需求规格说明来决定。测试任务是由质量需求、测试目标来决定的，质量要求越高，越要进行更深、更充分的测试，回归测试的次数和频率也要加大，自然，测试的工作量要增大。处在不同的开发阶段，测试工作量的差异也挺大。新产品第一个版本的开发过程，相对于以后的版本来说，测试的工作量要大一些。但也不是绝对的，例如，第一个版本的功能较少，在第 2、3 个版本中，增加了较多的新功能，虽然新加的功能没有第一个版本的功能多，但是在第 2、3 个版本的测试中，不仅要完成新功能的测试，还要完成第一个

版本的功能回归测试，以确保原有的功能正常。

在一般情况下，一个项目要进行 2~3 次回归测试。所以，假定一轮（Round）功能测试需要 100 个人日（man-day），则完成一个项目所有的功能测试肯定就不止 100 个人日，往往需要 200～300 多个人日。可以采用以下公式计算：

$$W = Wo + Wo \times R1 + Wo \times R2 + Wo \times R3$$

- W 为总工作量，Wo 为一轮测试的工作量。
- R1，R2，R3 为每轮的递减系数。受不同的代码质量、开发流程和测试周期等影响，R1、R2、R3 的值是不同的。对于每一个公司来说，可以通过历史积累的数据获得经验值。

测试的工作量，还受自动化测试程度、编程质量、开发模式等多种因素影响。在这些影响的因素中，编程质量是主要的。编程质量越低，测试的重复次数（回归测试）就越多。回归测试的范围，在这 3 次中可能各不相同，这取决于测试结果，即测试缺陷的分布情况。缺陷多且分布很广的话，所有的测试用例都要被再执行一遍。缺陷少且分布比较集中，可以选择部分或少数的测试用例作为回归测试所要执行的范围。

代码质量相对较低的情况下，假定 R1、R2、R3 的值分别为 80%、60%、40%，若一轮功能测试的工作量是 100 个人日，则总的测试工作量为 280 个人日。如果代码质量高，一般只需要进行两轮的回归测试，R1、R2 值也降为 60%、30%，则总的测试工作量为 190 个人日，工作量减少了 32%以上。

自动化程度越高，测试工作量就越低。由计算机运行的自动化脚本效率很高，能使执行实际测试的工作量大大降低。但是在很多情况下，测试自动化并不能大幅度降低工作量，因为测试脚本开发的工作量很大。也就是说，将总体的测试工作量前移了，从测试执行阶段移到测试脚本设计和开发的阶段，总体工作量没有明显降低。同时，由于自动化脚本可以重复使用，而且机器可以没日没夜地运行，回归测试就可以频繁进行，如每天可以执行一次，这样任何回归缺陷都可以即时发现，提高软件产品的质量。

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。

（1）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。对于容量测试，主要依赖于建立测试所需数据的工作量大小。

（2）测试假设，目的是验证一个测试需求所需测试的动作数目，包括估计的每个测试用例所用的时间。

（3）阶段假定，指所处测试周期不同阶段（测试设计、脚本开发、测试执行等）的划分，包括时间的长短。

（4）复杂度假定，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。

（5）风险假定。一般考虑各种因素影响下所存在的风险，将这些风险带来的工作量设定为估算工作量之外的 10%~20%。

2.3.4 工作分解结构表方法

要做好测试工作量的估算，需要对测试任务进行细化，对每项测试任务进行分解，然后根据分解后的子任务进行估算。通常来说，分解的粒度越小，估算精度越高。可以再加上 10%~15%的浮动幅度，来确定实际所需的测试工作量。比较专业的方法是工作分解结构表（WBS），它按 3 个步骤来完成。

（1）列出本项目需要完成的各项任务，如测试计划、需求和设计评审、测试设计、脚本开发、测试执行等。

（2）对每个任务进一步细分，可进行多层次的细分，直到不能细分为止。如针对测试计划，首先可细分为：

- 确定测试目标。
- 确定测试范围。
- 确定测试资源和进度。
- 测试计划写作。
- 测试计划评审。

“确定测试范围”还可再细分为功能性测试范围和非功能性测试范围的分析。**“测试计划评审”**可以再分为测试组内评审、项目组评审、公司质量保证小组评审和最终批准。

（3）列出需要完成的所有任务之后，根据任务的层次给任务进行编号，就形成了完整的工作分解结构表（如表 2-4 所示）。

表 2-4 测试工作分解结构表

1 测试计划	4 测试执行
1.1 确定测试目标	4.1 第1轮新功能测试
1.2 确定测试范围	4.2 性能测试
1.3 确定测试资源和进度	4.3 安全性测试
1.4 测试计划写作	4.4 安装测试
1.5 测试计划评审	4.5 第2轮回归新测试
2 需求和设计评审	4.6 升级和迁移测试
2.1 阅读文档以了解系统需求	4.7 最后一轮回归测试
2.2 需求规格说明书评审	5 测试环境建立和维护
2.3 编写/修改测试需求	5.1 软硬件购买
2.4 设计讨论	5.2 测试环境建立
2.5 设计文档评审	5.3 日常维护
3 测试设计和脚本开发	6 测试结果分析和报告
3.1 确定测试点	6.1 缺陷跟踪和分析
3.2 设计测试用例	6.2 性能测试结果分析
3.3 评审和修改测试用例	6.3 编写测试报告
3.4 设计测试脚本结构	7 测试管理工作
3.5 编写测试脚本基础函数	7.1 测试人员培训
3.6 录制测试脚本	7.2 项目会议
3.7 调试和修改测试脚本	7.3 日常管理
3.8 测试数据准备	……

WBS 除了用表格的方式表达之外，还可以采用结构图的方式，那样会更直观、方便，如图 2-6 所示。

当 WBS 完成之后，就拥有了制定日程安排、资源分配和预算编制的基础信息，这样不仅可获得总体的测试工作量，还包括了各个阶段或各个任务的工作量，有利于资源分配和日程安排。所以，WBS 方法不仅适合工作量的估算，还适合日程安排、资源分配等计划工作。

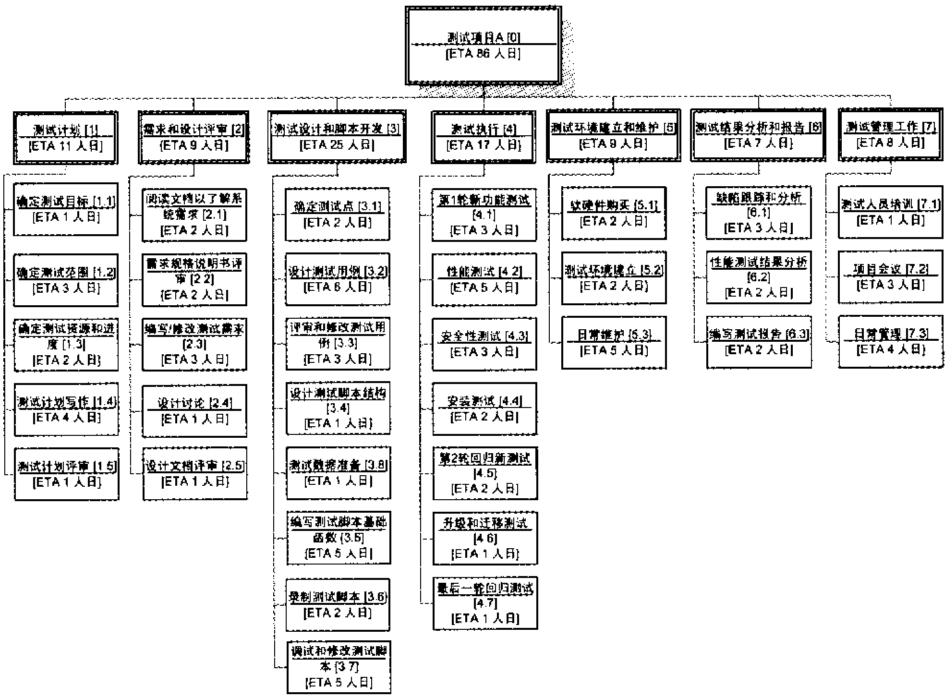


图 2-6 工作分解结构图

2.3.5 工作量估计的实例

结合雅虎日历的功能点可以看出，测试工作量与测试用例的数量成比例。根据全面且细化的测试用例，可以更准确地估计测试周期各阶段的时间安排。根据雅虎日历的功能计算，测试用例数为 6×60 = 360 例（以平均每个大模块 60 个用例来算）。除了测试用例数，还要考虑以下因素：

- 根据测试团队和项目的具体情况来算，如 2.3.3 节中的几个假定：效率假设、测试假设和应用的维数等。
- 测试平台、环境的不同组合，包括操作系统、浏览器、通讯协议、防火墙、代理服务器等的组合。
- 回归测试频率和重复次数。
- 自动化测试的水平。
- 其他特定的因素，增加 10%～20%的余量。

在雅虎日历的测试中，作如下假定和分析：

- 所有人员为中级软件测试工程师的水平。
- 每个测试用例设计时间为 20 分钟，包括评审、输入到用例管理数据库中等所用的时间。所以测试用例设计的时间为 120 小时，即 15 个人日。
- 70%的测试用例可以进行自动化测试，30%为手工测试。即自动化测试用例数为 252 例，手工测试用例数为 108 例。
- 每位工程师每天可开发 10 个测试用例的测试脚本，包括调试。所以测试脚本开发的工作量为 26 个人日。
- 要进行两次的回归测试，R1、R2 值为 70%、40%，则单平台下手工运行的测试用例数为 108×(1＋70%＋40%) = 227 例。
- 对操作系统没有影响，而且不考虑 SSL 的支持，只考虑浏览器 IE6.0、IE7.0、Firefox1.5、Firefox2.0 和代理服务器的影响。作为交叉组合，共设为 4 种。
- 也没有必要在 4 种组合上运行所有的测试用例，两种主要组合运行 100%的手工测试用例，另两种组合运行 50%的手工测试用例，即测试用例数为原来的 3 倍，所以手工运行的测试用例数为 227×3 = 681 例。
- 假定每个测试工程师每天可以运行 60 个测试用例，即每个测试用例的执行要用 5 分钟，运行测试用例要用 5 个小时，另外 3 个小时用于处理缺陷报告和邮件、与开发人员沟通等。所以手工测试用例执行的时间为 12 个人日。
- 自动化测试的运行都在晚上进行，工程师需要时间分析测试结果、修改脚本适应新的变化、做缺陷报告等，估计要 5 个人日。

这样就估算出了功能测试的基本工作量，即 15＋26+12+5=58 个人日。

对系统测试的工作量，可以按照同样的方法进行，所不同的是系统测试几乎是由测试工具完成的，工作量主要集中在环境构建、测试数据准备和结果分析等上面。表 2-5 给出了雅虎日历所要的测试工作量。

表 2-5 测试工作量估算示例

任　务	工作量（人日）	说　明
整个测试过程	88	下列各项的合计
需求评审与测试计划	8	是其子项的合计
评审需求文档和定义测试需求	3	
估计工作量	1	

续表

任　　务	工作量（人日）	说　　明
确定资源	1	
生成和评审测试计划文档	3	
设计测试用例	23	
分析测试用例结构	1	
测试用例设计方法讨论	2	
编写功能测试用例	12	包括集成测试
编写系统测试用例	4	
查看测试需求的覆盖率	2	
测试用例的评审	2	
开发自动化测试脚本	30	
录制和调试测试脚本	4	涵盖了系统测试的基础脚本
改为关键字、数据驱动结构	5	
基础函数的编写	6	
系统测试脚本的开发	4	
建立外部数据集合	3	包括系统测试的数据准备
重新调试、修改脚本	8	
测试执行	26	
设置测试环境	1	
功能测试（手工测试部分）	12	包括结果分析、缺陷报告
功能测试（自动化测试部分）	5	包括缺陷报告
系统测试	6	主要是性能、安全性测试等
评估、跟踪和验证缺陷	2	和项目经理、开发人员一起评审缺陷
测试评估	2	
评估测试需求的覆盖率	0.5	
编写和审查测试报告	1	
项目分析总结	0.5	

2.4 测试资源需求和团队组建

分析测试范围之后，所需要的测试资源就比较清楚了。测试的资源需求，包括人力资源和软、硬件资源。人力资源，侧重如何组建测试团队——项目测试组，而软、硬件资源，对于不同的项目差异很大。这里只讨论一般的操作方法，设计测试环境的建立，将在第7、9章会作详细介绍。

2.4.1 测试资源需求

如果将测试资源进行较为详细的分类，可以归纳为如图2-7所示。

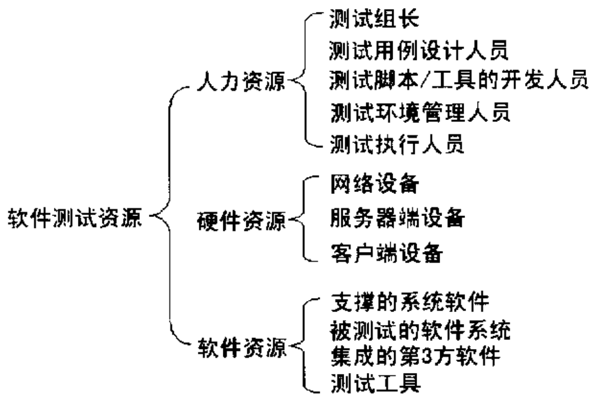


图2-7 软件测试资源的分类

1. 人力资源需求

在完成了测试工作量的估算之后，软件测试项目所需的人员数目就能够基本确定了。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，测试组长首先要介入进去，参与需求评审、确定测试需求和测试范围、制定测试策略和测试计划等。

（2）在测试前期，需要一些比较资深的测试设计人员、测试脚本或测试工具开发人员参与或负责软件测试需求的制定和分解、设计测试用例、开发测试脚本等工作。

（3）在测试中期，主要是测试的执行，测试需求的数量取决于测试自动化实现的程度。如果测试自动化程度高，人力的投入则不需要明显的增加；如果测试自动化程度低，对执行测试的人员要求就比较多了。

（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。

2. 测试环境资源

把建立所有必要的测试环境所需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3方软件产品、测试工具软件等，如：

- 硬件：交换机、路由器、负载均衡器（Load balance）、服务器、客户端PC机、摄像头、特殊的显示卡和声卡、耳机、麦克风等。
- 支撑的系统软件：Linux操作系统、Web服务器（如Apache）、中间件（如Tomcat、WebLogic）、数据库系统软件MySQL/Oracle等。
- 测试工具：JUnit、Jmeter、Selenium、IBM-Rational Robot等。

2.4.2 团队组建

团队组建是动态的，可针对一个项目建立测试小组，这个测试小组因项目存在而存在，因项目结束而撤销。根据所评估的测试工作量以及产品开发的时间表，就能确定测试小组的人数。从经验看，不同的软件应用领域，软件测试人员和开发人员的比例往往是不同的，大致可以分为3类：

（1）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。

（2）像应用平台、支撑系统软件一类的产品，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。测试人员与开发人员的比例要低一些，在1:1的水平就可以了。

（3）对于特定的应用系统一类的产品，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了限制，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。

除此之外，这个比例还依赖于交到测试人员手中的代码质量，代码质量越高，所要进行的回归测试工作量会越低，所要求的测试人员自然会越少。如果代码质量很高，加之开发人员做了足够的单元测试，所要进行的功能测试很有限，这时测试人员主要集中在系统

性能、安全性和可靠性等测试上，这时测试人员的需求量就可以很低，可以突破一般情况下的比例（1:2 到 2:1 之间）。而代码质量取决于软件过程、流程和开发人员的水平、能力等方面，更重要的是企业文化，包括质量文化。

一个比较健全的测试组，所具有的角色（如第 1 章表 1-1 所示）包括测试组长、实验室管理人员、自动化测试工程师、资深测试工程师和初级测试工程师。对于规模很小的测试小组，就可能没有设置独立的实验室管理人员，而由其中资深的测试工程师或测试组长兼任。作为测试组长，有时要承担测试经理的部分责任，如人员培训、资源管理、团队发展等，并负责内部审查、评审的工作，检查软件缺陷描述及其他测试报告的质量等。

2.4.3 培训

项目测试组的内部培训不容忽视，特别是当项目组有新人或初级测试工程师时，培训的作用更大。除了整个项目组所做的有关产品、业务领域的培训外，测试组还要做些有关测试用例设计方法、测试自动化原理、测试脚本开发技术、环境设置等方面的培训。

培训内容实际可以分为纵向和横向两部分。纵向培训实际是知识传递的过程，参与需求和设计评审的测试人员，要对未参与这两方面工作的人进行培训；参与测试用例设计、测试脚本开发的人，也要对其他人做些说明和讲解。横向培训是在不同团队之间进行的，即请开发团队、软件工程过程组等对测试人员进行有关技术、流程管理等方面的培训。无论是纵向培训还是横向培训，都要讲究培训的有效性、及时性、正确性和完整性。培训的主要内容有：

- 标准的测试流程有哪些？本项目中有什么具体的或特定的测试流程？
- 自己的职责和任务是什么？
- 本项目的用户有何特殊需求，如何更好地理解测试目标？
- 怎样设计和编写测试用例，有哪些具体方法？哪些方法可以用于本项目？
- 如何使用当前的测试工具？如何针对当前测试工具开发测试脚本？
- 怎样做探测性（Ad-hoc）的测试？
- 怎样报告软件缺陷，哪些信息是必须附上的？
- 工作报告内容和格式有什么要求，日报和周报有什么区别？
- 如何建立测试环境等。

通过培训，可以达到下面一些目标：

- 帮助整个项目组建立对所有问题的共识，包括对测试需求的一致理解。
- 每一个测试人员达到合格的水平和能力，保证测试工作的质量，包括牢记缺陷报告的具体要求。
- 提高部分测试人员的测试设计和开发能力，保证测试用例、脚本的质量。
- 培训就是一个知识传递、知识共享的过程，可及时更新大家的知识。

培训可以有正式的和非正式的两种形式，而且通常可以采用非正式的形式，如：

- 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。
- 建立一帮一的伙伴关系或者师傅带徒弟的关系。

培训的考核是必要的环节，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。

2.5 测试里程碑和进度安排

在软件测试项目的计划书中，都会制定一个明确的日程进度表。项目进度安排，实际上取决于测试的目标和现有的人力资源。测试目标的定义，决定了测试里程碑的定义。测试工作量是一定的，所以测试进度和测试资源之间存在很强的依赖性。当人力资源充足时，测试周期短；当人力资源较少时，测试周期就会长。

里程碑一般是项目中完成阶段性工作的标志，即用一个结论性的标志来描述一个过程性任务明确的起止点。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）和下个新阶段启动的条件或前提（Entry Criteria）。里程碑具有很强的时序性，还具有下列特征：

- 里程碑也是有层次的，在一个父里程碑的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，其数量的多少不一样，里程碑可以合并或分解。

在软件测试周期中，建议定义下列 6 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，以利于更有效的控制，如表 2-6 所示。

表 2-6 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

2.6 测试风险分析

在 1.1.4 节讨论了测试的风险观点，测试被定义为“对软件系统中潜在的各种风险进行评估的活动”，这意味着软件测试潜在着较高的风险，所以软件测试的风险分析非常重要。软件测试风险，就是要将测试范围、测试过程中的风险识别出来，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。

风险识别的有效方法是建立风险项目检查表，按风险内容进行逐项检查、逐个确认。对于测试的风险，可以给出如下一个风险项目检查表，如表 2-7 所示。

表 2-7 软件测试风险项目检查表

类别	内容	示例
人员风险	测试人员的状态、责任感、行为规范等	因个人工作疏忽而漏掉缺陷；某个人生病、离职等，造成资源不足，使测试不够充分或缩小了测试范围

续表

类别	内容	示例
环境风险	在多数情况下，测试环境是一个模拟环境，很难和实际运行环境一致	用户数据量、运行环境的垃圾数据、十几年的运行时间等
测试范围(广度)	很难完成100%的测试覆盖率，有些边界范围容易被忽视	测试很难覆盖模块之间接口参数传递、成千上万的操作组合等
测试深度	对系统容量、可靠性等测试深度不够	对互联网上的应用、操作行为和习俗等研究不够，测试时达不到实际的用户数
回归测试	回归测试，一般都不会运行所有的测试用例，而是运行部分的测试用例。越到测试后期，回归测试执行的用例数越少	修正一个缺陷后，除了验证这个缺陷之外，测试人员往往根据自己的经验来确定回归测试范围，而且这个范围很小
需求变更	软件需求变化相对较多，有时在后期还发生需求变更，从而影响设计、代码，最后反映到测试中来	需求变更后，文档不一致，测试用例没有及时更新、回归测试不足等
用户期望	测试人员不是用户，很难百分之百地把握用户的期望，这种差异也会带来风险	适用性测试就是一个典型的例子，不同的用户对界面的喜好和操作习惯，都会存在或多或少的差异
测试技术	借助一些测试技术完成测试任务，可能有些测试技术不够完善，有些测试技术存在一定的假定，这都会带来风险	如正交实验法在软件测试中应用时，很难达到其规定的条件
测试工具	测试工具经常是模拟手工操作、模拟软件运行的状态变化、数据传递，但可能存在和实际的操作、状态和数据传递等的差异	如性能测试工具模拟1000个并发用户同时向服务器发送请求，这些请求都从一个客户端发出。而实际运行环境中，1000个用户从世界不同的地方、不同的机器发出请求，请求的数据也不同

在测试风险分析中，逐项检查，确认风险之后，要找出对策，以避免风险产生或降低风险所带来的影响。表2-8给出了软件开发中常见的风险，说明这些风险发生的可能性、对测试的影响和影响程度以及如何进行预防和控制。

表 2-8 软件测试风险识别和控制措施

风 险	可能性	潜在的影响	严重性	预防/处理措施
软件需求不清楚、变更导致测试需求及范围发生了变化	高	导致测试计划、工作量等发生变化	较严重	和用户充分沟通，做好调研、需求获取和分析，调整测试策略和计划
开发进度延长，包括项目计划的变更、各个环节的进度拖延	中	推迟系统测试执行的时间和进度	严重	设定更多的子里程碑，控制整体进度，做好沟通和协调
由于设计时间不足、代码互审和单元测试不够，导致开发代码质量低	中	Bug 太多、严重，反复测试的次数和工作量大	严重	做好软件设计、提高编码人员的编码水平、进行单元测试。严格控制提交测试的版本、调整测试策略和计划
对需求的理解偏差太大。原因是缺乏原型、与客户沟通不足、需求评审不到位	中	对 Bug、设计的合理性等确认困难	严重	与用户、产品经理多沟通，并借助一些原型和演示版本来改进
测试工程师对业务不熟悉，主要原因是业务领域新、测试人员是新人或介入项目太迟	低	测试数据准备不足、不充分，测不到关键点，同时测试效率难以提高	一般	测试人员及早介入项目，与产品经理，市场、设计等各类开发人员沟通，加强培训，建立伙伴、师傅带徒弟的关系
由于项目提交日期的变更导致测试周期变更，一般是由客户提出的	低	系统测试总时间缩短，难以保证测试的质量	非常严重	严格控制项目的时间变更，多与客户沟通并得到客户的理解。调整测试策略、测试资源及计划

下面会针对本书的两个典型案例给出简单的风险分析。客户端软件的测试风险相对来说会低一些，所以不作讨论。而且本书对软件测试中普遍存在的风险也不作讨论。

1. Google Talk 的几个测试风险

- 如果基本功能方面的问题太多，将严重影响性能压力测试的进度。
- 在第二轮测试后，产品应该基本无大的问题。开发要在第三轮测试开始前保证修复好所有已发现的主要问题。
- 多语言版的测试要求测试人员前期准备充分，测试人员对所测语言有可能不太了

解，对该语言国家的使用习惯也不是很熟悉。

- 开发拿出产品第一版本时，要提供或与测试人员共同完成压力测试工具的开发。

2．雅虎日历的测试风险

（1）项目复杂度。由于开始对项目的复杂度估计不足，导致项目后期的产品代码不能按时完成，这样势必会影响到测试的环节。

（2）需求的变化。用户的反馈、市场需求的变化会导致项目后期增加一些新的产品功能，这样就会使产品不能按照原定的测试计划完成，以致测试人员处于等待状态中。

（3）服务器的升级。基于 Web 方式的产品，随着不断推出的新服务器产品（如 Linux 和 Apache 的版本升级），其兼容性需要验证，而且其安全性、稳定性和性能等方面所受到的影响需要分析，在测试过程中更换第 3 方产品的新版本，就要面临这样一个问题——是否要重新测试已测试的范围。往往不会从头再来，而是根据已定义的策略进行选择性的测试。

（4）数据库的升级。基于 Web 方式的产品，后台一般都离不开数据库的支持。如果测试过程中遇到数据库表结构的变化、版本升级（例如 Oracle 9i 升级到 Oracle 10G），都会给测试过程带来风险。例如，升级后的数据库变得不稳定，有可能退回到原来的版本，影响测试甚至导致测试的失败。

（5）测试环境的不稳定性。例如被测试的服务器不能被访问，需要重新启动和配置，这会占用一定的时间。一旦不能访问测试环境，测试人员不仅无事可做，还常常会抱怨。这种情况影响了测试人员的情绪，最终也影响了测试结果。

（6）国际化和本地化的影响。如支持哪些语言版本？国际化版本的测试策略和方法，以及翻译公司是否能及时完成任务，翻译是否准确也会带来风险。

3．测试风险的控制方法

（1）根据风险发生的概率和带来的影响确定风险的优先级，然后采取措施避免那些可以避免的风险。如测试环境不对，可以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，有些风险带来的后果可能非常严重，能否通过一些方法，将它转化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了一个严重的 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，就要通过提高测试用例的覆盖率来降低这种风险；

（4）为了避免、转移或降低风险，事先要做好风险管理计划。例如，把一些环节或边界上有变化、难以控制的因素列入风险管理计划中。

（5）对风险的处理还要制定一些应急的、有效的处理方案。例如，为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，仍可以继续下去。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。

（6）在做计划时，估算资源、时间、预算等要留有余地，不要用到100%。

（7）制定文档标准，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。

知识点

风险管理的基本内容有两项：风险评估和风险控制（如图2-8所示）。

（1）风险评估，主要依据三个因素：风险描述、风险概率和风险影响。可以从成本、进度及性能三个方面对风险进行评估，它是在建立在风险识别和分析的基础上的。通过评估可以确定这些风险的特点或可能带来的危害。

（2）风险控制，制定风险管理计划和风险应急处理方法，降低风险、消除风险。

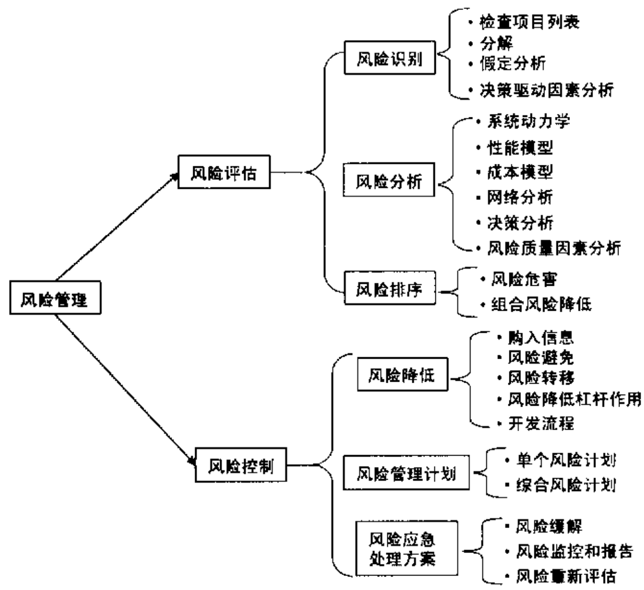


图 2-8 风险管理的基本内容

2.7 制定有效的测试策略

为了最大程度地降低测试风险，尽早地发现各种潜在的软件缺陷，在测试实施之前，测试组必须确定合适、有效的测试策略，并以此为依据选定测试方法、测试工具和测试用例设计思想。通过制定测试策略来指导测试用例的设计、测试工具的选择和执行，特别是能解决测试当中经常碰到的一些问题。良好的测试策略必将给软件测试带来事半功倍的效果，可以充分利用有限的人力和物力资源，高效地完成测试目标。

测试策略描述当前测试项目的目标和所采用的测试方法，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还包括，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）。其内容包括：

- 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，软件功能要满足整体需求，实现正确，并与用户文档的描述保持一致。如声明测试完成的标准——95%的测试用例通过并且两个最高级别的缺陷全部被修正，用以计划、实施及通报测试结果。
- 描述测试用例是什么样的，如何执行，用了什么样的数据，又如何执行后续的回归测试。
- 选定要使用的测试技术和工具。如采用了什么工具，工具的来源是什么，是否是60%用 Rational Robot 自动测试，40%用手工测试。
- 根据经验判断和头脑风暴，对以往的测试中经常出现的问题加以考虑。又如采取一些发散性的思维，往往能帮助找到新的测试途径。
- 考虑影响资源分配的特殊情况。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，有些测试须考虑与外部或硬件接口。

为了更好地确定软件测试策略，可以试着问如下一些问题，在寻找这些答案的过程中，也就找到了最佳的测试策略。

- 回归测试的范围如何确定？
- 如何利用可重复性的测试？
- 测试缺乏可预见性，如何收集衡量测试结果的指标？
- 如何建立稳定的、模拟系统实际运行的测试环境？

- 如何从无穷的输入数据中选择合理的、有效的测试数据集？
- 如何加强静态测试——规格说明书、设计文档和程序代码等的审查？
- 如何处理单元测试和集成测试的关系？
- 如何处理手工测试和自动化测试之间的平衡，使它们的互补性得到发挥，使测试的效率和质量达到最佳状态？
- 如何衡量这份测试策略的有效性？

1. 测试策略制定的 3 项基本要素

软件测试策略制定有 3 项基本要素：输入、输出和过程。

（1）输入，作为制定测试策略的依据，包括限制条件和已具有的资源：

> 所要求的软、硬件的详细说明，包括测试环境、测试工具等。
> 人力资源和测试进度的约束，包括测试组成员的角色和职责说明。
> 测试方法和衡量测试是否通过的标准。
> 被测软件组件或系统的功能性和技术性需求文档，以及其变更请求的控制流程。
> 软件系统所受的其他限制。

（2）输出，制定策略的成果，即最终对所制定策略的定义或说明：

> 通过/失败的准则和测试风险评估的结果。
> 已批准和签署的测试策略文档。
> 和测试策略相对应的测试计划、测试用例的设计思想和思路。

（3）制定策略的过程。测试组分析需求，参与设计的讨论，要求开发、编写针对所有测试级别的测试策略，并和项目组一起复审测试策略和测试计划。测试策略应该覆盖整个项目的生命周期，需要各类技术人员（系统架构师、数据库管理员、编程人员等）参与。各类技术人员相互之间应多交流、讨论，以保证制定正确的测试策略。

2. 基于测试技术的测试策略

著名的软件测试专家 Myers 指出了使用各种测试方法的综合策略。

- 在任何情况下都要使用边界值分析方法，因为为边界值分析方法所设计的测试用例能很有效地发现软件代码的缺陷。

- 等价类划分方法是对边界值分析方法的有效补充。
- 如果软件某些功能的输入数据/条件存在多种组合情况，则一开始就可选用因果图法。
- 错误推测法可以帮助追加一些比较特殊、不易直接推理出来的测试用例。
- 对照程序逻辑来审查已有测试用例的逻辑覆盖程度。如果没有达到要求的覆盖率，则应当再增加一些测试用例。
- 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。

3. 分阶段的测试策略

- 严格地执行代码复查，以保证在早期就发现问题，而不是在代码发布之后。
- 利用单元测试和集成测试，可以尽早地发现更多的问题，并准备好自动化测试的BVT（Build Verification Test，软件包验证测试）。
- 需要建立一个正规的且自动化的烟雾测试，只有100%通过烟雾测试，才能进入下一个阶段。
- 系统测试中，以每次发布用户基线为结束标志，用户基线会增长，同时也会逐渐地要求一些更为精确的性能测试。
- 不能忽略安全性测试、可用性测试、配置测试和数据完整性测试。
- 在功能性测试、安全性测试、配置测试中可进行一些探索性测试。
- 制定更为详细的UAT（用户验收测试）测试计划，将其与测试脚本和培训材料一起提供给用户，以帮助用户快速提高并完成任务。

4. 基于测试方案的综合测试策略

（1）根据软件产品或服务特性对客户的使用价值以及特性失效所造成的损失，来确定相应特性的测试优先级。产品特性的优先级越高，其被测试的时间越早，测试的力度也越大。

（2）要使用尽可能少的测试用例，发现尽可能多的程序错误。一次完整的软件测试过后，如果程序中遗漏的（较）严重错误过多，则表明本次测试是不足的或失败的，这意味着可能让用户承担较大的利益损失风险。反过来说，如果过度测试，则又会浪费软件企业自身的宝贵资源。所以，需要在以上两点——风险和效率上进行权衡，找到一个最佳平衡点。

（3）测试策略应该尽量地简单、清晰，例如可以在有限的白板上通过 2~3 行字和 1~2 图就描述出测试策略来，或者可以通过一个简短的会议（20～30 分钟），就能把测试策略解释清楚。

（4）基于缺陷分析的测试策略。通过缺陷分析，可以更好地了解开发人员的习惯，找到容易犯错误的地方，可以更好地设计测试用例，更快地发现缺陷。也可以从缺陷出发，反推回去，找到合适的测试策略。

5. 测试策略的示例

在雅虎日历的测试项目中，要考虑在浏览器和操作系统的不同组合下进行测试。最简单的办法就是完成所有组合的测试，如 7 个操作系统（Windows 2000、Windows XP、Windows Vista、Mac 10.2、Mac 10.4、Linux 和 Solaris）和 7 个浏览器（IE6、IE7、FireFox1.5、FireFox2.0、NS8、Mozilla 1.7、Safari2.0）的组合有近 40 种——因为有些组合是不存在的。以前面所说的 360 个测试用例为基础，在各种环境上共要执行 14 000 多个测试用例，工作量太大。这时就需要根据操作系统和浏览器的市场占有率、对测试用例的影响程度、缺陷分析的结论和经验等，简化或优化组合。从表 2-9 可以看出等价组合降到了 8，只要执行大约 2880 个测试用例，测试的工作量大大降低。

表 2-9 针对浏览器和操作系统的优化组合

操作系统/浏览器	IE6.0	IE7.0	FireFox 1.5	FireFox 2.0	NS 8.0	Mozilla 1.7	Safari 2.0
Windows 2000	50%	20%					
Windows XP	100%	50%			50%		
Windows Vista	20%	100%					
Mac 10.2			50%		20%		
Mac 10.4				100%			50%
Linux				50%		50%	
Solaris			50%			50%	

针对雅虎日历这样的产品，还可以制定针对功能性和用户界面的测试策略。如按照雅虎日历的功能划分来设计测试用例，保证一系列的逻辑被有效的验证。

- 输入时间或活动的组合（不同长度和单双字节的字符串长度）。

- 相同的时间或活动在不同日历里面的显示和跳转（用户界面）。
- 改变用户的不同设置。

在不同的分辨率、操作系统和浏览器的组合情况下完成上述的功能测试。

概念

（1）烟雾测试。烟雾测试（Smoke Testing）是在代码被检入（check in）代码库之前对代码修改进行验证的流程或活动。在代码复审（code reviews）之后，烟雾测试是识别和修正软件缺陷的最有效方法。它可以确认代码的修改符合要求和期望，且不会造成软件包构建的失败。烟雾测试和自动化回归测试的脚本集一起被用来测试那些高风险的功能或高容量的事务处理。

（2）BVT（Build Verification Test）是软件包构造之后由测试工具（脚本）完成的验证测试，用以确认基本功能正常，没有出现严重的缺陷。BVT通过后，才进行手工测试，或者进一步的自动化测试。

2.8 完整生成测试计划书

在软件测试需求和测试范围分析、工作量估计、测试资源和进度安排、测试风险评估、测试策略制定等工作做完之后，测试计划也就基本大功告成了。测试计划本身就是为了解决测试目标、任务、方法、资源、进度和风险等问题，所以当这些问题被解决或找到相应的对策和处理措施之后，测试计划剩下的工作就是写好这个文档，将上述内容描述清楚。有一点必须在这里说明的就是，测试计划是一个过程，不仅仅是“测试计划书”这样一个文档，测试计划会随着情况的变化不断进行调整，以便于优化资源和进度安排，减少风险，提高测试效率，并及时修改“测试计划书”。

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的，这些内容仅仅作为参考，但有些内容（如人员组成、日程安排）也可以看作是一种结论，或者承诺，是必须要实施或达到的目标，如测试小组的结构和组成、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。测试计划内容的焦点则集中在下列内容上：

（1）**目标和范围**：包括产品特性、质量目标，各阶段的测试对象、目标、范围和限制。

（2）**项目估算**：根据历史数据和采用恰当的评估技术，对测试工作量、所需资源（人力、时间、软硬件环境）做出合理的估算。

（3）**风险计划**：对测试可能存在的风险进行分析、识别，以及对风险的回避、监控和管理。

（4）**进度安排**：分解项目工作结构，并采用时限图、甘特图等方法制定时间/资源表。

（5）**资源配置**：人员、硬件和软件等资源的组织和分配包含每一个阶段和每一个任务所需要的资源。人力资源是重点，而且与日程安排联系密切。当发生类似到了使用期限或者资源共享的时候，要及时更新这个计划。

（6）**跟踪和控制机制**：包括质量保证和控制、变化管理和控制等，明确如何准备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，是用什么样的测试用例测出该问题的，以及明确问题产生时的测试环境。

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。为每一个阶段制定一个计划书，还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）制定特别的计划书。

同时，可以为上述测试计划书的每项内容制定一个具体实施的计划，如将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。对于更为详细的要求，可以参考国家标准《测试计划（GB8567—88）》中制定的测试计划通用模板。

2.9 小结

本章主要是介绍如何创建有效的测试计划。首先，强调需求评审的重要性，只有做好产品需求文档的审查和评审，才能清楚测试的需求。而且做好产品需求的评审工作，才能更早地发现需求上的问题，以避免引起后续设计、代码上更严重的缺陷。

在测试计划过程中，主要做好下列各项工作：

- 确定软件功能性、非功能性的测试需求，以及各个阶段的测试任务。
- 进行测试范围分析，从而对测试工作量进行估算。工作量估算方法主要介绍了工

作分解结构表方法，并给出了实例。

- 测试资源需求、团队组建，包括培训。
- 测试里程碑和进度的安排。
- 对测试风险进行分析。
- 制定有效的测试策略。

最后完整地生成测试计划书，进行计划书的评审、跟踪和及时修改，测试计划是一个过程，不仅仅是“测试计划书”这样一个文档，测试计划会随着情况的变化不断进行调整，用以优化资源和进度安排，降低风险，提高测试效率。

第3章 设计验证

成功的产品开发和演化依赖于体系结构恰当的选择。我们能承受由于不鉴别、不评估体系结构而带来的后果吗？

大量实践统计表明，在大规模软件开发中有 70%的错误来自需求和设计阶段。良好的需求分析和软件设计过程不仅可以提高软件开发效率，也是确保软件产品正确、可靠的基础。在完成了需求定义或需求文档审查以后，需要对软件设计进行验证。软件验证涵盖了系统架构设计、系统部署的逻辑和物理设计、系统详细设计的验证，其中很重要一点就是选择设计的评价标准。

在以往有关软件测试的著作中，很少触及软件设计验证，但这也不能否定软件设计验证是软件测试过程中重要的环节之一。软件设计是把软件需求转换为软件表示的过程，也是将用户需求准确转化为软件系统的唯一途径。

对软件设计进行验证，就是更好地保证这种转换的正确性和完整性，也是为了更好地设计系统测试的用例。再者，软件设计的验证，需要在编程之前进行，这样做可避免编程误入歧途，并为软件部署的准备获得更多的时间。这一切，都有助于缩短软件开发周期，使产品早日面向市场，提高软件企业的竞争力。

3.1 系统架构的审查

软件设计一般可以分为体系结构设计（Architecture Design）和详细设计（Detailed Design）两个阶段。体系结构设计，过去习惯称为总体设计或概要设计，指将软件需求转化为数据结构和软件的系统结构，并定义子系统（组件）和它们之间的通讯或接口。

体系结构设计是软件开发过程中决定软件产品质量的关键阶段，如 RUP 就强调以架构设计为核心，所以体系结构设计的评审是非常重要的。软件设计评审，主要是技术评审，审查软件在总体结构、外部接口、主要部件功能分配、全局数据结构以及各主要部件之间的接口等方面的合适性、完整性，从而保证软件系统可以满足系统功能性和非功能性的需求。

3.1.1 系统架构选型的确认

软件体系结构，一般可以分为客户机/服务器（C/S）结构、浏览器/服务器（B/S）结构和中间件多层结构等，也可以分为集中式系统、分布式系统和对等的 P2P 结构系统等，还可以分为实时同步系统、异步系统等。

系统平台软件和终端软件体系结构的划分是以高性能、高可靠性、高安全性、高扩展性和可管理性为原则的。系统软件采用分布式处理方式，可将不同功能的应用放在不同的计算机上处理，也可以将相同功能的应用分布在不同的计算机上处理，这样可以在不提高单个计算机处理能力的情况下有效提高整个系统的处理能力，以保护或降低硬件投资。

在多层体系设计中，按照以下方式进行明确地层次划分。针对多层次架构，设计的评审方法将采用分层评审和整体评审相结合的方式。经过整体评审到分层评审、再从分层评审到整体评审，这样的过程既能确保评审的深度，又能确保评审的一致性。分层评审，这里给出一个例子，就是从人机交互、业务逻辑和数据服务等 3 个层次来评审体系架构。

- 人机交互，提供简洁的人机交互界面，完成用户输入和输出的请求。
- 业务逻辑，包括业务流程和规则的设定，能灵活满足业务需求变化的需要，是客户与数据库对话的桥梁。在业务逻辑层，需要实现分布式管理、负载均衡、故障

转移和恢复、安全隔离等设计需求。

- 数据服务，提供数据的存储服务，确保在数据库设计上的一致性、完备性、安全性和高效性等。

在软件体系结构的设计中会采用一些建模常用的模型，如结构模型、框架模型、动态模型、过程模型和功能模型等。针对不同的软件应用或服务，应选用合适的模型。为了很好地完成系统架构的选型或评估，应综合各方面考虑，确保选择合适的模型对系统架构进行描述。例如，信息系统、管理系统的结构清楚、严谨，适合采用结构模型，而实时应用系统则选用动态模型、过程模型较好。

根据业务需求，更好地构造合理的、清晰的体系结构，需要从不同的角度去分析业务逻辑，这就是视图的应用。软件设计有三种基本视图——物理视图、逻辑视图和概念视图，分别对应实体空间、过程空间和形式空间。在理想情况下，每个视图只有一个模型，但实际上每个视图可能有多个模型，这是因为组织和技术在不断成长和改变。然而，能否得到这些模型合理的最小集合，是软件设计及其验证的关键。

为组织规划和构建不同级别的模型是一项相当费时费力的工作，然而，能否正确定义这些模型，使之为软件设计服务，在软件设计中也是至关重要的。结构模型的设计错误总是会导致严重的设计问题或运作问题（如可伸缩性、可靠性问题），严重时甚至会导致项目无法完成以及影响业务。作为资深的测试人员，有责任帮助系统设计人员寻找合适的设计框架、设计模式来创建和实现这些模型，从而将使用错误模型而带来的风险降到最低。所以，对于一个有一定规模的软件组织，有必要培养软件测试架构师。

概念

三种基本的视图

（1）**概念视图**，用于定义应用程序的业务需求和商业用户视图，以便生成业务模型。概念性建模技术（例如用例分析、活动图解、过程设计和业务实体建模等）有助于构建关键的业务过程及其使用的数据描述，可以强调业务目标和需求，并且不包含实现技术。它所对应的形式空间是抽象的逻辑空间。

（2）**逻辑视图**，就是结构设计师创建的应用程序模型，可以视为业务模型或应用程序结构的逻辑视图，用以决定数据管理和处理步骤的对应关系。它会根据逻辑信息和顺序设计模型部件之间的交互，来确定模型保留的数据类型和状态，从而实现系统的总体结构。逻辑视图对应于软件表示模型，是沟通源系统和目标系统的桥梁。表示模型的形成需要一个过程，所以称其为过程空间。

（3）**物理视图**。应用程序模型的每个元素均要求映射到真正的技术元素上。通过这种方法，应用程序模型以实现模型的方式得以实现。它所对应的实体空间——物理的、现实的空间是源系统所在的空间（源系统就是表示软件要实现自动化的系统）。

3.1.2 软件设计评审标准

有关软件设计的质量属性，主要包括可维护性、可移植性、可测试性和健壮性等。这些质量属性被认为是特别的静态测试——设计验证的需求，软件设计的验证就是从这些需求出发，来完成相应的任务的。软件设计验证的需求可以分为三类，如下所示。

（1）软件运行和服务的设计验证需求：性能、安全性、可用性、功能性、可使用性。

（2）软件部署和维护周期的设计验证需求：可修改性、可移植性、可复用性、可集成性、可测试性。

（3）与体系结构本质相关的验证需求：概念完整性、正确性、完备性和可构造性。

软件设计的评审，依赖于软件系统所采用的技术平台，还依赖于软件规模、结构、度量方法，包括复杂度、耦合性、内聚性等的度量。使用度量可以更准确地判断软件设计是否符合相关的设计标准。同时，软件设计的评价工具和技术比较多，可帮助人们确保软件设计的质量。

- **软件设计评审**：有正式的和半正式的，通常是以小组方式进行，用来验证和保证设计结果的质量。
- **静态分析**：正式或半正式的静态（不可执行的）分析技术，可以用于评价一个设计（例如故障树分析或自动交叉检查）。
- **模拟与原型**：软件设计通过软件系统设计模型来表示，软件设计评价可以转化为对软件系统设计模型的评价，这是评价设计的动态技术，如性能模拟或可行性原型。

软件设计的三种基本视图（物理、逻辑和概念视图）分别对应实体空间、过程空间和形式空间，所以软件设计评价定义了三类标准，即实体空间标准、过程空间标准和形式空间标准。

概念

三类设计评价标准

（1）**实体空间标准**，以源系统作为标准来度量系统设计模型，是一个软件设计最终

应该符合的标准。它依赖于对于源系统的认识程度，同时由于软件设计是思维的产物，它又很难直接应用于软件设计模型上。设计的合理性就是实体空间的标准，但没有一个具体的内容和形式。实体空间标准的执行，一般可通过业务领域专家组或用户代表根据经验进行评审来实现。

（2）**过程空间标准**，可以看作实体空间的间接标准，是基于分析模型和设计模型来定义的。由于设计模型的存在，过程空间标准在设计评价中就比较容易使用，如检查设计是否符合需求，就能检验设计模型和分析模型的一致性。软件开发一般采用迭代或增量的分阶段模型进行，设计活动也分多次进行，通过不同阶段设计结果（设计书）的对比，可以找到与设计不一致的地方（设计缺陷），并能检查设计对需求的覆盖情况。

（3）**形式空间标准**，以目标系统的角度（即软件产品质量属性）检验系统设计。实体空间标准和过程空间标准可以保证目标系统的功能满足源系统，但不能保证目标系统在运行状态下的质量属性。形式空间标准实际就是产品质量标准，可以使用质量模型进行评价，如围绕产品改进、产品运行、产品移交等3种使用情况来组织质量属性，并测试目标系统。

通过形式空间标准对软件设计进行检验时，往往并不存在一个唯一的检验标准。这是因为实际软件的质量要求不是唯一的，不同的软件有不同的质量属性要求。而特定软件的质量要求，是在需求分析、设计的过程中逐步形成的。这些质量要求最终会成为检验软件设计的标准之一。

在实际设计评审中，设计质量标准可以分为两类，一类是设计技术自身所要求的，另一类则是系统的非功能性质量特性所要求的。

1. 设计技术的评审标准

（1）设计结果的稳定性。以设计维护不变的时间来衡量，如果因为用户需求的变化或现有设计的错误或不足必须修改设计，那么修改范围的大小和次数就是影响软件设计质量的重要因素。

（2）设计的清晰性，指涉及目标描述是否明确，模块之间的关系阐述是否清楚，是否阐述了设计所依赖的运行环境，业务逻辑是否准确并且完备。清晰的设计也是重用性的基础。

（3）设计的合理性，主要指合理地划分了模块和模块结构的完整性，类的职责单一性、实体关联性和状态合理性等。可以进一步考察是否对不同的设计方案作了说明与比较，是否有方案选择的结论，是否清楚阐述了方案选择的理由。

（4）系统的模块结构所显示的宽度、深度、扇入值和扇出值，是衡量系统的复杂性的简单标准指标，如图3-1所示，所以每个模块应具有适当的深度、宽度、扇出和扇入，力求做到单入口单出口。

（5）系统模块间松耦合而模块内部又保持高度一致性、稳定性（强内聚力）是系统架构设计中侧重要考虑的内容，高耦合度或低内聚力的系统是很难维护的。

（6）给出的系统设计结构和数据处理流程是否能满足软件需求规格说明中所要求的全部功能性需求，模块的规格及大小划分是否与功能需求项以及约束性需求项保持一致。

（7）可测试性和可追溯性，所有的设计目标（性能、容量、兼容性等）是否可以通过测试结果来衡量；每一部分的设计是否都可以追溯到软件需求的定义，包括功能需求项和非功能需求项。

（8）所要设计的系统在整个项目软件（或在大系统中）中所处的地位和作用，以及与同级、上级系统之间的关系描述是否准确。

（9）是否对不完整、易变动或潜在的需求项都进行了相应的设计分析，对各种设计限制是否做了全面的考虑。

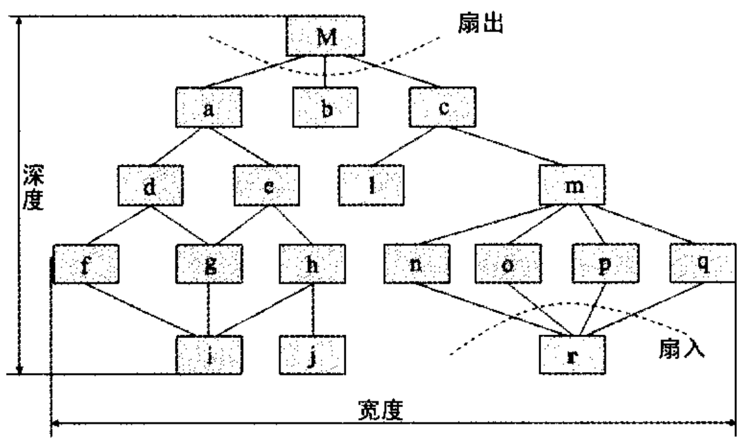


图3-1　系统模块结构的复杂性描述

2. 非功能性质量特性的设计评审要求

（1）安全性。数据和系统的分离、将系统权限和数据权限分别设置等都可以提高系统的安全性。例如，用设计中间层来隔离客户直接对数据服务器的访问，进一步保护数据库的安全性。系统的稳定性、可靠性都是对系统安全性的有力保障。

（2）性能。合理的体系结构、优秀的数据库管理系统（如 Oracle）、分布式应用体

系结构、服务器集群等设计都可以提高系统性能。又如，通过负载均衡以及中间层缓存数据的能力可以提高系统对客户端的响应速度。

（3）稳定性。采用多层分布式体系架构，可以提供更可靠的稳定性。例如，中间层缓冲客户端与数据库的实际连接，可以使数据库的实际连接数量远小于应用客户端的数量。通过分布式体系，将负载分布到多台服务器上，可以平衡负载、降低系统整体失效的风险，从根本上保证系统的稳定性。

（4）扩展性。软件设计的技术要求是系统扩展的重要保证，例如简单的模块结构、模块间低耦合性、多层分布体系架构等。又如，业务逻辑在中间服务器上，当业务规则变化后，客户端程序基本不做改动；而当业务猛增时，可以在中间层部署更多的应用服务器，以提高对客户端的响应，且所有变化对客户端都是透明的。

（5）可靠性。如系统设计保证不存在单点失效，任何系统的关键部位都有备份机制或故障转移处理机制。

3. 评审的输入

- 软件体系结构文档。
- 设计规范和设计指南。
- 风险列表。

4. 评审的输出

- 经认可的软件体系结构文档。
- 变更需求。
- 评审记录。

5. 评审的检查点

- 软件体系结构、设计模式。
- 部署视图、进程视图。
- 封装体、协议。

3.1.3 系统组件设计的审查

软件组件化（包括模块化、构件化）能使软件系统具备应付复杂问题，满足日后维护管理要求的某些重要特性，因此有必要验证组件及其内部设计是否合理。组件设计依赖于系统构建所采用的特定开发平台和技术，可以为组件设计建立其通用的准则。

- 组件的功能和接口定义正确，文档描述清楚，包含正确的硬件接口、通信接口设计、用户接口设计等。
- 为每个模块确定采用的算法，选择某种适当的工具表达算法的过程，写出模块的详细过程性描述。
- 数据结构、数据流和控制流的定义正确。能描述模块输入数据、输出数据及局部数据的全部细节。
- 功能、接口和数据设计具有可测试性、可预测性。如要为每一个模块设计出一组测试用例，以便在编码阶段对模块代码（即程序）进行预定的测试。

设计出软件的初步结构以后，应该审查分析该结构，通过模块分解或合并，力求降低耦合性，提高内聚力。例如，可以通过分解或合并模块以减少信息传递对全局数据的引用，并降低接口的复杂性。还可以通过以下方法降低程序的耦合度：隐藏实现细节、强制构件接口定义、不使用公用数据结构、不让应用程序直接操作数据库等。

图 3-2 显示 7 种耦合的表现形式，从非直接耦合到内容耦合，耦合性逐渐增强。内容耦合度很高，在设计中要尽量避免，可以借助数据库、XML 等手段，将公共环境耦合、外部耦合、控制耦合、特征耦合转化为数据耦合，以降低耦合性。

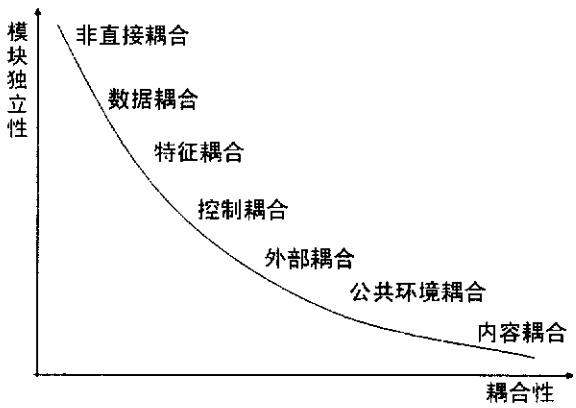


图 3-2 耦合的表现形式

图 3-3 描述了如何更好地提高系统组件内部的聚合力，从偶然内聚、逻辑内聚、时间内聚、过程内聚、通信内聚、信息内聚到功能内聚的内聚性和模块独立性是一个不断增强的过程。功能内聚是最高的，也就是为什么过去一直强调功能模块的划分。但软件设计模式也在不断地发展，它越来越多地强调多层次的划分，包括对数据层、逻辑层、业务层和表现层等的处理和实现。

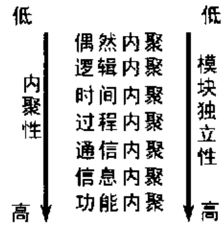


图 3-3 常见的内聚方式

概念

在面向对象设计时，应遵循下列设计原则：

（1）**开—闭原则**（Open-Closed Principle，OCP），一个软件实体应当对扩展开放、对修改关闭，即使这个模块可以在不被修改的前提下被扩展（改变这个模块的行为）。

（2）**单—职责原则**（Simple responsibility pinciple，SRP）。就一个类而言，应该仅有一个引起它变化的原因，如果有多于一个的动机去改变一个类，就应再创建一些类来完成每一个职责。

（3）**里氏代换原则**（Liskov Substitution Principle，常缩写为LSP）。如果一个软件实体使用的是一个基类，那么一定适用于其子类，而且根本不能察觉出基类对象和子类对象的区别，但反过来则不成立。也就是，应当尽量从抽象类继承，而不是从具体类继承。基于契约设计（Design By Constract，简称DBC）对LSP提供了支持。

（4）**依赖倒转原则**（Dependence Inversion Principle），要求客户端依赖于抽象耦合。抽象不应当依赖于细节，细节应当依赖于抽象。应当使用接口和抽象类进行变量的类型声明、方法的返还类型声明以及数据类型的转换等。

（5）**接口隔离原则**（Interface Segregation Principle，ISP）。根据客户需要的不同，为不同的客户端提供不同的服务是一种应当得到鼓励的做法。

（6）**合成/聚合复用原则**（Composite/Aggregate Reuse Principle，CARP）。在一个新的对象里面使用一些已有的对象，使之成为新对象的一部分；新的对象通过这些向对象的委派达到复用已有功能的目的。

（7）**迪米特法则**（Law of Demeter，LoD）**或称最少知识原则**（Least Knowledge Principle，LKP），一个对象应当对其他对象尽可能少了解。如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用，如果其中的一个类需要调用另一个类的某一个方法，可以通过第三者转发这个调用。

3.2 产品设计规格说明书的复审

作为需求定义和产品设计的重要输出，软件产品设计规格说明书是用户与开发人员双方在对软件需求取得共同理解的基础上达成的协议。它是实施开发工作的基础，把需求转化为行为，也就是通过所设计的功能来满足用户的需求。产品设计规格说明书对所开发软件的功能、性能、用户界面及运行环境等作出详细的描述，并通过复审确认文档中所有的描述是清晰、正确的。至少在各个层面上，测试人员和产品经理、设计人员、开发人员能达成一致。

3.2.1 重视设计规格说明书的审查

软件设计规格说明书对所开发软件的功能、性能、用户界面及运行环境等作出了详细的说明，它是用户与开发人员双方在对软件需求取得共同理解的基础上达成的协议，也是进行编程、测试工作的基础。通常使用下面的 3 种方法来编写软件设计规格说明书：

（1）采用良好的结构化和专业语言编写文本型文档。

（2）建立图形化模型，这些模型可以描绘系统状态及其之间的变化、转换的过程、数据字典及其关系、逻辑流或对象类及其之间的关系。

（3）编写形式化规格说明，通过使用数学上精确的形式化逻辑语言来定义系统设计。

产品规格说明书的审核应该发生在整个产品生命周期的初期，即在产品说明书生成后就进行，应该在产品设计前基本完成，以后在开发和测试过程中应尽量不作改动或少作改动。若确实需要变更的，需要通过严格的变更流程去控制。产品规格说明书很重要，是测试的标准，没有它，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，没有测试的标准容易陷入大量的争论之中。

因为软件开发周期较长，而整个行业变化又很快，种种原因会导致新的功能不断增加，产品规格说明书不断地更新。这是软件开发者痛恨而又不得不面对的难题。例如竞争对手推出了新的吸引人的功能，客户则提出新的要求以适应新的形势，致使原先的设计很难或无法实现。

软件测试人员必须要想到产品规格说明书可能改变，可能会有未曾计划的特性会加入，也可能会有经过测试并报告了的软件缺陷的特性发生变化甚至被删除。要针对产品规格说明书的变化，不断地验证并灵活地调整测试计划和测试方法。

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。所以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。

3.2.2 设计规格说明书的多层次审查

软件设计规格说明书的审查，首先是从客户的角度和立场出发并进行核实的一项工作。其次，它是一个技术评审的过程，也是文档测试和设计上技术审查的综合。从文档的质量需求看，要求设计规格说明书具有一致性、准确性、易理解性，并符合相应模板所要求的各项内容、格式等。从设计上看，不仅可以与同类产品进行比较，而且要与现有的各类标准保持一致，检验套用标准的正确性，不要和行业规范相抵触。

设计规格说明书的审查，可以从其设计的各类 UML 图、数据流图、工作流图等方面系统地进行。对设计规格说明书的审查，可以分两个层次来进行，即从高层次（high-level）审查开始，逐步向低层次（low-level）推进，全面保证设计规格说明书的质量。

1. 高层次审查

审查设计规格说明书，一开始不要陷入一些细节中，而是应该先从整体功能结构和操作逻辑上进行审查，可以从功能规模、复杂性、可测试性等方面逐项检查，力求发现设计上的严重缺陷。若发现任何不清楚的逻辑设计，应尽早地纠正。

功能结构的设计，不仅要正确反映产品需求和客户期望，使之符合 PRD/MRD 所定义的需求，而且应具有合理的层次性，功能前后一致，能满足客户各个方面的需求，即功能应具有良好的完备性。例如，在雅虎日历上，仅仅具有“建立、查询、修改、删除和显示各项事件”的基本功能是不够的，还应该具有下列功能：

- 添加用户的通讯录，更方便的功能是可以从 MS Outlook/Lotus Notes 等邮件系统中导入通讯录。
- 多种日历显示方式，如日、周、月和年。
- 自动提醒功能，有多种渠道提醒用户，如手机短信、邮件等。
- 还需要设置个人的一些爱好。
- 设置周期性（每周、每月、每个季度）重复的事件。
- 具有全文搜索功能。
- 可以显示不同事件的状态。
- 可以按不同方式（事件名、时间、状态等）显示事件。

在操作逻辑上，要符合用户的习惯。例如：

- 操作最多的是查询事件和添加事件，删除、修改的操作相对少些。
- 首先要看当天有哪些活动，然后会关心本周、本月的活动。
- 日历软件以日历为中心，所以可以考虑在屏幕左边设计一个小区域，用于显示日历。
- 最常用的功能在最方便、最显眼的地方。

除此之外，设计规格说明书应符合业界标准和惯例，所以高层次的审查还要集中检查是否符合已有的标准和规范，包括国际标准、国内标准、行业标准、地方标准和企业自身定义的规范。

2. 低层次审查

低层次的审查，就是从细节进行逐字逐句的检查，包括文字、图形化的描述是否准确、精确、完整和清楚。例如，对说明书中所有的术语（terminology）仔细检查，看是否事先对这些术语有清楚的定义，不能用同一个术语来描述意义不同的对象，同一个对象也不宜用两个以上的术语去描述，力求保证术语的准确性，不出现二义性。设计规格说明书中不要使用不确定性的词，如“有时、多数情况下、可能、差不多、容易、迅速”等，而应明确指出事件发生或结果出现所依赖的特定条件。

概括起来，低层次审查设计规格说明书的主要事项有：

- 保持语句和段落的简短。
- 采用主动语态，表达方式一致。
- 是无错别字、语法正确和标点正确的完整句子。
- 避免使用模糊、主观的术语。
- 避免使用比较性的词，如提高、最大化、最佳化等。正确地说明，应该是定量地说明需要提高的程度。

3.2.3 界面设计的评审

用户界面的设计是软件系统设计的重要组成部分，特别是对于交互式软件系统，用户界面设计的好坏直接影响到软件系统设计的质量。一个优秀的用户界面应该是一个直观的、对用户透明的界面，用户在首次接触这个软件时就觉得一目了然，不需要多少培训就能够很快上手使用。目前流行的界面风格比较多，但无论哪种风格，用户界面设计应服从以下这些最基本的原则。

（1）用户界面设计，应遵守惯例和通用法则，例如 Windows 应用程序，其界面设计

自然要按 Windows 界面的规范来设计，应是包含“菜单条、工具栏、工具厢、状态栏、滚动条、右键快捷菜单”的标准格式。

（2）独特性。在符合界面规范的前提下，设计具有自己独特风格的界面也很重要。在商业软件流通中，独特性能够很好地起到潜移默化的广告效应。

（3）用户界面必须保持一致性和规范性。系统和子系统各部分的菜单、对话框、按钮等应有相同或相近的风格、形式，包括色彩、尺寸、初始位置、操作路径、提示用词等。

（4）用户界面应有自助功能。用户界面应提供不同层次的帮助信息和一定的错误恢复能力，方便用户使用。提示信息必须规范、恰当、容易理解，例如，提供详尽而可靠的帮助文档、在线帮助、视频/Flash 指导等，用户在使用过程中可以随时查找到解决方法。

（5）用户界面易懂、易用。用户界面的形式和术语必须适应用户的能力和需求。界面上所有的按钮、菜单名称应该易懂，用词准确，能望文知意。理想的情况是用户不用查阅帮助就能知道该界面的功能并进行相关的正确操作。

（6）美观与协调性。界面布局、颜色和尺寸等应该适合美学观点，给人感觉协调舒适，能在有效的范围内吸引用户的注意力。

（7）快捷方式的组合。在菜单及按钮中使用快捷键可以让喜欢使用键盘的用户操作快捷，而且与 Windows 应用软件中快捷键的设计是一致的。

（8）错误保护：开发者应当尽量周全地考虑到各种可能发生的问题，使出错的可能性降至最小。如果因应用程序出错而退出，意味着用户操作中断，前功尽弃，不得不费时费力地重新登录、重新操作，很容易使用户对软件失去信心。

3.2.4 验证过程与确认过程

验证过程与确认过程（V&V）看起来类似，但是它们目标不同，处理的对象也不同，“验证”是检验软件工作产品是否符合规定的设计要求，而“确认”过程则要证明所开发的最终产品在其预定的环境中能发挥预定的作用，满足客户使用的需求。“验证”是以产品设计规格说明书作为依据的，而“确认”是以客户需求为目标的。

在验证活动圆满完成任务后，为什么还要进行确认活动？因为设计规格书本身就可能有问题或存在错误。即使软件产品中某个功能实现的结果和设计规格书完全一致，通过了验证，但设计的某些功能可能不是用户所需要的，这依然是严重的软件缺陷。因为设计规格书有可能一开始就对用户的某个需求理解错了，所以仅仅进行验证测试是不充分的，还需要进行确认测试。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。对产

品设计规格说明书的审查可以看作是“确认”活动的组成部分。

这就是勃姆（Boehm）对V&V的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过程规范。
- Validation： Are we building the right product? 是否构造了正确的软件？即是否正在做用户真正所需要的产品。

确认和验证活动往往同时进行，各项确认活动的作法均与验证活动类似（如测试、分析、仿真等），并尽可能地利用同一个环境的某些部分。所以在实际软件测试过程中，并不会将“验证”和“确认”完全隔离开来，分别开展活动，而是在综合的一个活动中去实现两者的目标。

验证和确认活动也应该是一个渐进的过程，因为它们要在产品和工作产品的整个开发过程中执行，即从需求阶段开始，在过程推进中不断对工作产品进行验证和确认，直至对最终完成的产品进行最后的验证和确认。在产品的每个层次上开展验证和确认活动，有助于提高产品（包括构件）满足顾客需求的能力和可能性。

概念

（1）**验证**（Verification，可译为“检验”）是指验证或检验软件是否已正确地实现了产品规格书上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新的基准。

（2）**确认**（Validation，更准确地翻译是“有效性确认”）是为了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，表明软件是否满足客户需求（指分配给软件的系统需求），并解决了相应问题。测试是实现验证活动和确认活动的最有效的手段和途径。测试的准则正是产品规格书所定义的系统功能和特性，而且包括用户在界面、适用性等方面的实际需求。

3.3 系统部署设计的审查

系统部署设计的审查，是基于软件服务的质量目标，用来审查软件部署的目标、策略

是否合理，是否得到彻底的执行。软件服务的质量目标会受到各种各样约束和因素的影响，对这些因素的分析，也是系统部署设计审查的任务之一，包括通过因果图分析方法确定影响因素和软件服务质量目标之间的关系，找出在软件部署上对软件服务的质量目标影响最关键的因素等。

系统部署设计的审查，着重是否服从和遵守部署设计的技术规范。该技术规范就是描述软件部署的体系结构所达到的服务质量（Quality of Service，QoS）技术指标，如性能指标、安全性等级、可靠性要求等，一般包含下列文档：

- 用户任务和使用模式分析，对不同的用户使用模式进行分析。
- 模拟用户与规划系统间交互的使用案例，如采用统一建模语言 UML 标准绘制使用案例图，以最终用户的视角说明操作的完整流程，为使用案例指定相对加权，加权最高的使用案例代表最常见的用户任务。
- 源自业务要求的服务质量要求，对不同使用模式的技术要求也不一样，应分别进行针对性分析。

技术规范就是软件部署规划的成果，将作为部署设计的输入。在技术要求阶段，可能还会指定作为随后创建服务级别协议（Service Level Agreement，SLA）的服务级别要求。SLA 规定是维护系统所必须提供的客户支持的条款，并且通常在部署设计阶段作为项目核准的一部分签署。

概念

（1）软件部署是通过整合的、虚拟化的或逻辑化的资源和进程集中管理的，以便对所要运行的程序提供技术和环境的支撑，从而保证软件系统在部署到合适的运行环境中时能具有最优的、最可靠的性能表现，并能对用户和系统的各种数据进行有效的存储、备份和恢复等。

（2）部署图（Deployment Diagram）展现了对运行时处理节点及其构件的部署。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。

3.3.1 系统部署逻辑设计的审查

软件部署的设计和软件的系统或架构设计同时进行，软件部署设计分为逻辑设计和物

理设计，两者相辅相成，构成一个完整的软件部署设计方案，从而最终保证实施的有效性。软件部署设计要完成下面的一系列任务：

- 构造软件系统的逻辑体系结构，包括软件系统的层次依赖关系。
- 将逻辑体系结构中指定的组件映射到物理环境，从而生成一个高级部署体系结构。
- 创建一个实现规范，该规范提供关于如何构建部署体系结构的初级详细信息。
- 创建一系列详细说明实现软件部署方案的计划，包括迁移计划、安装计划、用户管理计划、测试和验证计划等。

概念

（1）逻辑设计就是将系统的使用案例作为输入，确定实现解决方案所需的软件体系结构、组件及其之间的相互关系。

（2）软件部署的物理设计，是部署方案的具体设计，将逻辑体系结构的组件映射成物理环境的设备，包括服务器、存储器设备、网络设备和其他硬件，从而创建实际可部署的体系结构。例如，在物理设计中，指定硬件服务器硬件型号和配置、安装的物理目录和备份目录、网络连接设备品牌和型号等，还包括确定实际物理环境所必需的负载测试、功能性和操作性测试。

开发逻辑体系结构时，不仅需要确定向用户提供服务的组件，还要确定提供必要中间件和平台服务的其他组件。基础结构服务的依赖性和逻辑层提供了两种执行此分析的补充方法。例如，在 J2EE 体系结构中，正是由 QoS、基础结构服务依赖性和逻辑层构成了一个完整的三维体系。

（1）**系统服务质量**，如性能、可用性、可伸缩性及其他要素。

（2）**逻辑层**基于软件服务的特性，表示软件组件组成的逻辑层次关系及其业务关系；逻辑层表示可被客户层访问的对象、业务和数据服务。

（3）**基础结构服务依赖性**。软件组件需要一套允许分布式组件间相互通信和交互操作的底层基础结构服务。

无论是 J2EE 体系结构还是.Net 体系结构，都非常适合设计为多层体系结构。在多层体系结构中，服务根据其提供的功能放在不同层次中。每个服务都是逻辑独立的，并且可由同层或不同层的服务访问，如图 3-4 显示了企业应用程序的一个由“客户层、表示层、业务服务层和数据层”构成的多层体系结构模型。

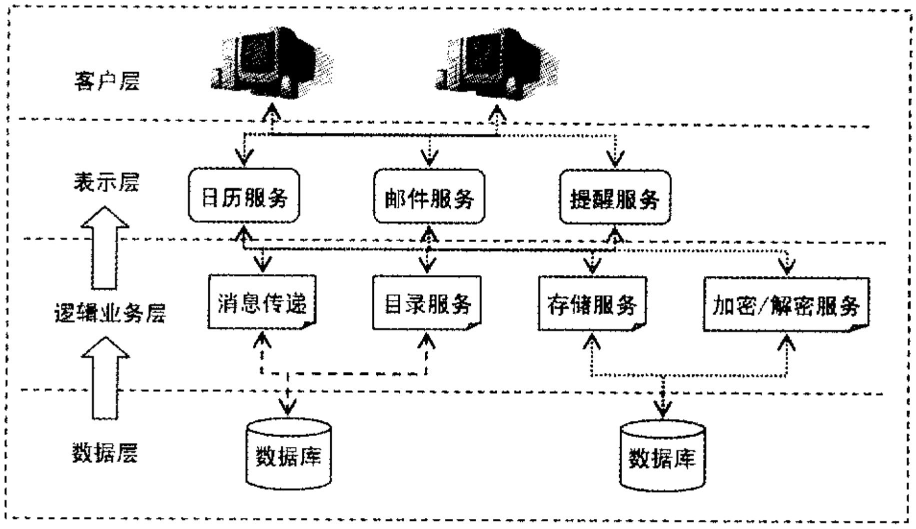


图 3-4　多层体系结构模型示意图

在软件部署设计阶段，多层体系结构中的功能或服务层次有助于确定在网络中分配服务的方式，有助于确定体系结构中的组件之间所存在的访问或支持服务。多层体系结构的直观性有助于规划软件部署解决方案的可用性、可伸缩性、安全性和其他质量特性。

3.3.2　软件部署物理设计的审查

软件部署的最终目标是实现业务目标，所以软件部署设计的质量好坏，最终取决于对满足业务目标的能力评估。逻辑体系结构是确定分配服务最佳方式的关键，物理设计是在逻辑体系结构框架下展开并受其约束的。逻辑设计的审查在上面已经讨论过，下面着重讨论软件部署物理设计的审查，其主要考察的质量因素有：

- 服务级别协议。指定了最低性能要求以及未能满足此要求时必须提供的客户支持级别和程度，相当于物理设计的底线。
- QoS 要求。物理体系结构和逻辑体系结构有清晰的映射关系，从而达到性能、可用性、可伸缩性、可维护性等 QoS 目标。如系统从意外故障中恢复的过程，包括大规模故障和小规模故障的恢复过程。
- 用量分析。有助于通过系统负载的使用模式来隔离性能瓶颈，开发出满足 QoS 要求的策略，以便用于物理设计中。尽管使用案例已包含在用量分析中，但评估部署设计时，应参考使用案例，确保任何案例中所揭示的问题在物理设计中得到了处理或解决。

- 成本。在物理设计中，满足 QoS 要求的同时，应尽量降低成本。所以，有必要设计 2~3 个软件部署的物理方案，通过分析、比较，对资源优化，采用平衡策略，能够使之在业务约束范围内达到业务要求，并获得成本最优化。

软件部署的物理设计是一个反复进行的过程，通常要复查 QoS 要求和初步设计，考虑不同 QoS 要求之间的相互关系，让 QoS 和成本平衡以获得最佳的部署方案，如表 3-1 所示。

表 3-1 软件部署物理设计的审查列表

系统性质	说　明
性能	对于将 CPU 集中分布在个别几台服务器上的性能解决方案，服务能否对计算能力加以高效利用？（例如，某些服务对可高效利用的 CPU 数量有上限）
潜在容量	设计策略是否处理超出性能估计的负载？ 对于过载，是采用垂直扩展的方式，还是以负载平衡到其他服务器的方式，或者以这两种方式兼用的方式进行处理？ 在达到下一部署扩展重大事件点前，潜在容量是否足以处理出现的异常峰值负载？
安全	是否对处理安全事务所需的性能开销给予了充分考虑？
可用性	对于水平冗余解决方案，是否对长期维护资源给予了充分估计？ 是否已将系统维护所需的计划停机考虑在内？ 是否在高端服务器和低端服务器间求得了成本平衡？
可伸缩性	是否对部署扩展的重大事件点进行了估计？ 是否制定了可在达到部署扩展重大事件点前提供足够的潜在容量来处理预测的负载增长的策略？
可维护性	是否在可用性设计中考虑了管理、监视和维护成本？ 是否考虑了采用委派管理解决方案来降低管理成本？

3.3.3 系统部署可用性设计的审查

可用性设计将考虑到系统整体的可用性降低或某个关键组件失效时所发生的情况，例如：

- 连接的用户是否必须重新启动一个新的会话。
- 一个区域内的故障对系统其他区域的影响程度。

在进行可用性设计验证之前，通过一个示例，可更好地帮助理解可用性的含义。

实例

Google Talk消息服务器的示例用于说明负载平衡的可用性策略。假定逻辑体系结构中每个组件（消息入站/出站、消息代理、消息存储、消息转发、语音处理等）都是2 CPU，4 GB 内存。其服务质量要求是：

- **可用性**：总体系统可用性应为 99.99%（不包括计划停机），单个计算机系统故障不会导致服务故障。
- **可伸缩性**：在日常峰值负载情况下，任何服务器的使用量都不应超过 80%，而且系统必须能够适应每年 10% 的长期增长速度。

为满足可用性要求，应为每个消息服务器组件提供两个实例，每一个都位于不同的硬件服务器上。如果一个组件的服务器发生故障，另一个组件可提供服务。两种可用性策略：

- 对所有关键的单个组件提供备份，分为主、从设备，在正常情况下，它们都可以正常工作，属于本地模式。
- 对一些提供 7×24 小时关键服务的系统，要设立异地备份，即系统之间互为主、从关系。两个独立的子系统互为备份。即使发生火灾、地震，这种策略还能保证系统的可用性。

在可用性方案中，配置比所需的资源要高，如CPU 数量为原估计数量的两倍，这样的结果是：

- 在一台服务器发生故障的情况下，另一台服务器提供处理负载的 CPU 能力。
- 对于在峰值负载下任何服务器的利用程度不超过 80% 的可伸缩性要求，添加的 CPU 能力可提供此保险余量。
- 对于适应年增长 10% 负载的可伸缩性要求，添加的 CPU 能力增加了潜在容量，在需要另外扩大规模之前可用于处理增长的负载。

软件部署的可用性设计主要通过下列 3 种方法——负载平衡、故障转移和备份机制来实现。

1. 负载平衡和故障转移

在多数情况下，负载平衡和故障转移可以一起考虑，例如，平行冗余的服务器可提供

负载平衡和故障转移两种功能来提高可用性。最简单的一种情形是双服务器系统，一台服务器即可满足性能要求，另一台服务器则作为备份服务器。其中一台服务器发生故障时，另一台服务器立刻接受请求，继续提供 100% 的服务，但这种成本比较高。

为了降低成本，可以通过在两台服务器间分配性能负载来实现负载平衡和故障转移。如果一台服务器发生故障，所有服务仍然可用，但是性能只能达到完全性能的某个百分比。例如，为满足性能要求的单个服务器需要配置 10 个 CPU，这时每个服务器配置为 6 个 CPU，两个服务器为 12 个 CPU，正常运行（它们同时运转）时能保证 100%~120%的性能。当其中一台服务器发生故障时，另一台服务器提供 6 个 CPU 的计算能力，即满足 60%的性能要求。

假如用 5 台双 CPU 服务器（5×2=10）来提供同样性能要求的软件服务，这时如果一台服务器发生故障，其余服务器可继续提供总计 8 个 CPU 的计算能力，达到 10 个 CPU 性能要求的 80%。如果在设计中增加一个具有 2 个 CPU 计算能力的服务器，实际得到的便是 N+1 设计。如果一台服务器发生故障，其余服务器仍可满足 100%的性能要求。N+1 设计具有下列优点：

- 单台服务器发生故障时的性能得到提升。
- 即使不止一台服务器停机，仍然具有可用性。
- 可轮换将服务器停机，以进行维护和升级。
- 多台低端服务器的价格通常低于单台高端服务器。

如图 3-5 所示，主应用域通过多台 Web 服务器、两台应用服务器、两台数据库服务器等构成内部负载平衡和故障转移机制，通过主应用域（Domain 或集群（cluster））和备份应用域构成系统级别的异地故障转移机制。不过，增加服务器数量会使管理和维护成本大幅增加，而且还应考虑在数据中心服务器维护的成本。

2. 复制机制

复制机制主要应用于数据的可用性设计，包括文件复制、目录复制和数据库复制等。复制机制也有多种策略，例如有单主复制和多主复制。采用单主复制，为主数据库提供一个中心源，然后将该中心源分配到使用者副本中，而采用多主复制，会在多个服务器间分配主数据库，然后为每个主数据库配置副本，如图 3-5 所示。

在多主复制中，一个或多个目录服务器实例管理主目录数据库。每个主数据库都有一个指定同步主数据库过程的复制协议，可被复制为任意数量的使用者数据库定期更新，而使用者的实例都按读取和搜索访问进行了优化，使用者接收的任何写操作都被引用回主数据库。多主复制策略提供了一个在更新主数据库时提供负载平衡、对目录操作提供本地控

制的可用性策略。

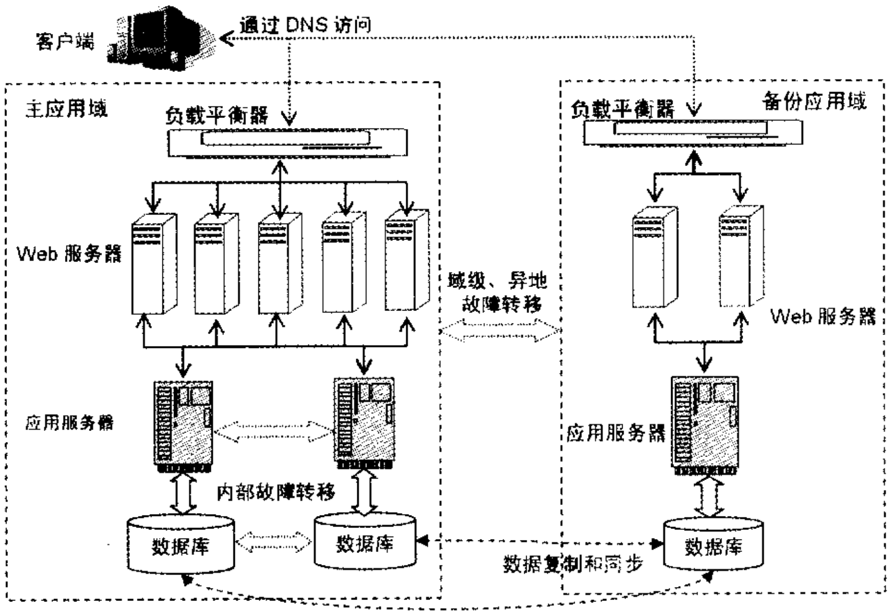


图 3-5 可用性设计（故障转移、负载平衡）的示意图

3. 可用性设计验证所需信息

为验证可用性策略和设计，测试人员需要收集有关可用性的信息，主要有如下内容。

- 指定的可用性中有多少个“9”，99.9%、99.99%或 99.999%？
- 故障转移情况下的性能要求是什么？如 1 分钟完成故障转移，或者故障转移的性能为原来的 60%。
- 用量分析是否区分高峰和非高峰使用时间？
- 地域考虑因素有哪些？
- 考虑可维护性、可伸缩性对可用性的要求？
- 是否存在单一故障点（Single-point failure）？

在用户看来，可用性更多牵涉单个服务，并不总是整个系统。例如，即时消息传送服务不可用，通常情况下对其他服务的可用性几乎没有影响。但是，许多其他服务（如目录服务器）所依存的服务的不可用，会对其他服务产生较大影响。较高的可用性规范应该明确引用关键任务的特定使用案例和用量分析。根据一组有序的优先级列出可用性需求，对软件部署的可用性验证也是有帮助的。在进行可用性验证时，还应研究组件交互和用量分

析，对组件进行逐个验证分析，以确定各个组件是否满足事先设计的要求。

不仅要验证是否满足 QoS 要求所需的资源，而且要对所有可用选项进行分析，通过分析每个设计决策中的平衡点，验证资源是否能被平衡利用，即确认是否为最佳解决方案——成本最低而又能满足 QoS 要求。例如，针对可用性进行水平扩展可能会提升总体可用性，但代价是需要增加维护和维修成本；针对性能进行垂直扩展可能会以经济的方式提高附加的计算能力，但所提供的软件服务对这些附加能力的使用效率不高。

概念

有关可用性的概念

- 单一故障点（Single-point failure）是指没有备用的冗余组件的硬件或软件组件，而这些组件是重要路径的组成部分，即该组件出现故障会使系统无法继续提供服务。设计容错系统时，必须确定并消除潜在的单一故障点。
- 关键任务，任何时候必须可用的服务。例如，应用程序的数据录入和 LDAP 目录服务功能、远程会议系统的会议服务（预定、开会/加会等）功能等。
- 负载平衡。使用冗余硬件（如服务器集群）和软件组件来分流处理负载。负载平衡器（如 NetScaler LoadBalance）把对某个服务的任意请求引导至该服务的服务器集群中当前负载最小的某个服务器上。如果任一实例发生故障，其他实例可以承担更大的负载。
- 故障转移。涉及对冗余硬件和软件的管理，在任何组件发生故障时提供对服务的不间断访问并保证关键数据的安全。如 Sun Cluster 软件为后端组件管理的关键数据提供了故障转移解决方案。
- 集群系统是冗余服务器、存储器及其他网络资源相结合的产物，在群集中的服务器彼此间不间断地通信时，如果其中一台服务器脱机，群集中的其余设备会将该服务器隔离，并将故障节点上的任何应用程序或数据故障转移到另一节点。这一故障转移过程所需时间较短，几乎不会中断为用户提供的服务。

3.3.4 系统部署可伸缩性设计的验证

可伸缩性是指增加系统容量的能力，而且要求在增加系统资源时不改变部署的体系结构。在系统需求分析、设计阶段，系统容量的预测往往只是估计值，可能与部署系统的实际情况存在较大差异，所以可伸缩性设计验证也是非常重要的。

在软件系统的可伸缩性设计上有不同的策略，如高性能设计策略和渐增式部署策略。高性能设计策略在性能要求的确定阶段加入潜在容量，可使系统具有一定的缓冲时间来应付增长的负载。而渐增式部署策略，则会基于负载的要求以及评估，事先明确系统扩展的条件以及条件可能达到的时间，对每一个重大的系统扩展特定日期/时间有一个估计和安排，从而建立部署的整个日程表。相对来说，高性能设计策略可使系统达到高可用性，从容地制订系统扩展的方案，但其一次性预算较高。

对用户设计模式的分析，可以预估系统的负载情况。系统的可伸缩性设计和用户、用量分析有直接的关系，更确切地说，系统的可伸缩性是建立在用户、用量分析的基础之上。如表 3-2 所示。

表 3-2 用户、用量分析项目列表

主 题	说 明
用户数量及类型	确定解决方案必须支持的用户数量，并在必要时对用户进行分类。例如: “企业对企业（B2B）”解决方案的访问用户数比较小，但每个用户访问时间长，对性能、安全性要求高。 “企业对消费者（B2C）”或“消费者对消费者（C2C）” 解决方案一般会有大量访问者，操作量大、数据大，而且区域分布也比较明显。
活动和非活动用户	确定活动和非活动用户的使用模式和使用比率。活动用户是指登录系统并与系统的服务进行交互的用户，系统的运行或操作性能主要关注这类用户。而非活动用户则对数据库、数据查询、存储需求影响较大。
管理用户	确定用户对系统访问的权限和范围，从而对软件部署进行监控、更新和支持的用户，包括安全性技术要求和特定的用户管理模式(例如，从防火墙外部管理部署)。
使用模式	确定各类用户如何访问系统，并提供预期用量目标。例如: 是否存在因用量高涨而产生的高峰期？持续时间是多少？ 正常业务时段和非正常业务时段的分布和区别或 7×24 不间断服务？ 用户是否呈明显的区域分布？
用户增长	确定用户群体规模是否固定，如果用户数量具有不断增长趋势，要进行预测并增加预测值。

续表

主 题	说 明
用户事务	确定必须支持的用户事务类型，可将这些用户事务转化为使用案例。例如： 用户登录后是否保持登录状态？是否频繁登录、注销？ 用户登录后，会执行哪些任务？有什么关键业务？ 用户间的重要协作是否通过公共电子日历、Web 页面或会议等来实现？
用户/历史数据	利用现有用户研究和其他资源来确定用户行为模式。应用程序过去记录下来的日志文件可能会包含一些有用的统计数据，对估量会有较大帮助。

3.3.5 系统部署安全性设计的验证

软件服务，多数都依赖了互联网。基于互联网的企业应用（尤其电子商务），其安全性设计和审查是至关重要的，包括安全政策和安全目标。软件系统的安全性涉及物理安全、网络安全、应用程序及数据安全、个人安全惯例等不同方面，但从安全性设计来看，主要对网络安全、应用程序及数据安全等进行审查和验证。

- 网络安全的审查，主要针对防火墙、安全访问区、访问控制列表和端口访问的设置进行，以确认是否已建立有效的策略，并防止未授权访问、篡改和拒绝服务（DoS）攻击。
- 应用程序及数据安全的验证，主要确认针对口令、加密、认证、访问权限和控制等的策略是否有效，是否存在安全漏洞。

对于特定的应用系统的安全要求，只有在明确了安全要求、目标的情况下，才能建立适当的、可靠的安全政策。例如，为 Web 环境建立安全目标时，应该考虑：

- 保护公司知识产权。
- 保护个人信息。
- 客户、合作伙伴、供应商等各方信息的分离。
- 财务交易的安全性。
- 建立对 Web 环境的信任。
- 致力于提供一种愉快而且安全的 Web 体验。

3.4 小结

软件设计的验证，就是从这些需求出发，来完成相应任务的。针对多层次架构，设计的评审方法将采用分层评审和整体评审相结合的方式，经过整体评审到分层评审、再从分层评审到整体评审的过程，既能确保评审的深度，又能确保评审的一致性。软件设计的验证需求，可以分为 3 类：

（1）软件运行和服务的设计验证需求：性能、安全性、可用性、功能性、可使用性。

（2）软件部署和维护周期的设计验证需求：可修改性、可移植性、可复用性、可集成性、可测试性。

（3）与体系结构本质相关的验证需求：概念完整性、正确性、完备性和可构造性。

系统部署设计的审查，是基于软件服务的质量目标，来审查软件部署的目标、策略是否合理、是否得到彻底的执行、是否服从和遵守部署设计的技术规范、是否满足 SLA 所规定的服务级别要求的。

系统部署设计的审查，涉及很多内容，本章主要讨论了下列 5 个方面：

- 部署逻辑设计的审查。
- 部署物理设计的审查。
- 部署可用性设计的审查。
- 部署可伸缩性设计的验证。
- 部署安全性设计的验证。

并对一些关键内容做了充分讨论，如多层体系结构、负载平衡、故障转移、复制机制等。

4

第4章

测试设计

如何灵活地运用各种基本方法来设计完整的测试用例，并最终实现暴露隐藏缺陷的目标，全凭测试设计人员的丰富经验和精心设计。

在测试过程中，测试计划完成、软件系统设计评审之后，就开始进入测试用例的设计过程。

测试需求和范围通过测试用例体现出来，并以更为有效的方式来执行测试，以便于更快地发现程序的缺陷。测试用例是测试脚本开发、测试执行的基础。只有设计好测试用例，才能保证测试的覆盖率。

本章将从软件测试实践中一些常用的测试用例设计思想、方法和组织角度，来阐述如何设计测试用例。通过本章的学习，可以了解和掌握以下的内容：

- 为什么需要测试用例?
- 如何根据项目的具体需求，构造测试用例结构?
- 测试用例有哪些基本元素组成?
- 设计测试用例时，需要遵循哪些基本的原则?
- 在测试用例设计上白盒测试方法和黑盒测试方法如何体现出来?
- 系统测试用例设计和功能测试用例设计有什么区别?
- 测试用例如何被组织起来完成不同阶段的测试任务?

4.1 测试用例框架的设计

测试用例是为某个特定测试目标而设计的，它是测试操作过程序列、条件、期望结果及相关数据的一个特定的集合，那么如何构造这个集合呢？测试用例的设计，类似于软件产品的设计，可以考虑面向对象、面向结构或面向方法来实现其框架，然后在这框架下细化测试用例——设计具体的测试用例。如果在结合第2章讨论的测试策略，软件测试用例的设计遵守下列4部曲：

（1）制定测试用例设计的策略和思想，在测试计划中描述出来。

（2）设计测试用例的框架，也就是测试用例的结构。

（3）细化结构，逐步设计出具体的测试用例。

（4）通过测试用例的评审，不断优化测试用例。

设计用例完成之后，有时需要开发相应的测试工具或测试脚本，有关这方面的内容，我们会在下一章介绍。

4.1.1 为什么需要测试用例

如何实现测试目标、完成任务，是借助测试用例来实现的。更具体地说，在进行软件测试时，总该需要一个类似“操作指导书”或“实验指导书”的文件来告诉我们如何操作，什么样的结果算是测试通过了吧，这就靠测试用例。测试用例是必不可少的、重要的测试件（Test ware）之一，测试用例是测试执行的基础，这是对测试用例的基本理解，再走近一点，我们还可以了解测试用例在多个方面的作用，更容易理解测试用例的重要性。

（1）测试用例是测试人员在测试过程中的重要参考依据。测试过程中，总要对测试结果有一个评判的依据，没有依据，就不可能知道测试结果是通过了还是没有，也不知道输入的数据正确与否，这一切需要定义，它在测试用例中得到了定义。

（2）测试用例可以帮助实施有效的测试，所有被执行的测试都是有意义的，不要执行毫无意义的测试操作。测试时是不可能进行穷举测试的，而应以最少的人力资源投入，在最短的时间内尽可能地发现所有的软件缺陷。完成测试任务，就依赖于良好设计的测试用例。测试用例将有助于节约测试时间，提高测试效率。

（3）良好的测试用例不断地被重复使用，使得测试过程事半功倍。在软件产品的开发过程中，要不断推出新的版本，并对原有功能进行多次的回归测试，即使在一个版本中，也要进行2~3次的回归测试。这些回归测试，就要求能重复使用测试用例。

（4）测试用例是一个知识积累的过程。在测试过程中，对产品特性的理解会越来越深，发现的缺陷也会越来越多。这些缺陷中，有些不是通过事先设计好的测试用例发现的，在对这些缺陷进行分析之后，需要加入新的测试用例，这就是知识积累的过程。即便最初的测试用例考虑不周全，随着测试的进行和软件版本的更新，也将日趋完善。

（5）测试用例是一个知识传递的过程，能保持一致的、稳定的测试质量。对于新的测试人员，测试用例的熟悉和执行，是学习产品特性和测试方法最有效的手段，这实际体现了知识传递的过程。有了测试用例，无论是谁来测试，参照测试用例实施，都能保障测试的质量，可以把人为因素的影响减至最小。

（6）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，正确的测试计划将会很好地组织这些测试用例并给测试人员或者其他项目的人提供参考和有效的使用方法。

（7）从项目管理的角度来说，测试用例的通过率是检验代码质量保证效果最主要的指标之一。我们经常说，代码的质量不高或者代码的质量很好，其依据往往就是测试用例的通过率，是以具体的量化结果作为依据的。

（8）测试用例也可以作为评估测试人员进度、工作量以及跟踪/管理测试人员的工作效率的主要因素，从而更加合理地做出测试安排或调整。不同的测试人员依据相同的测试用例所进行的测试，其结果往往是一致的，有利于对测试进行组织和管理。

因此，测试用例将会使得测试的成本降低，并可重复使用，也是检测测试效果的重要因素。设计良好的测试用例是测试的最重要工作。

4.1.2　测试用例设计考虑因素

在设计测试用例框架之前，需要了解各项影响测试用例设计的因素。或者说，在设计测试用例框架和具体单个测试用例时，需要掌握什么样的信息，才能做好这方面的工作。我们清楚，不可能穷举所有的测试场景或组合，所以在设计测试用例时，首先需要认识到这一点，要学会如何抓住测试的重点、要点或者关键点，然后以点带面，展开测试用例的设计。

测试用例的设计，就是围绕软件质量需求，分析质量需求的每一个剖面，使测试用例能覆盖各个剖面及测试点。另一方面，它会试图找出系统的薄弱环节、边界点等，因为这些特殊区域有必要得到更多的测试，尽力降低测试的风险，达到所设定的测试目标。

1. 测试用例设计的主要影响因素

（1）需求目标，是功能性的需求目标也是非功能性的需求目标。功能性的测试比较清楚，正确与否的判断能一目了然。而非功能性测试，其相对性比较强，需要从不同的侧面进行比照。

（2）用户实际使用的场景。从用户的角度来模拟程序的输入，包括用户的操作习惯，使产品更能贴近用户的需求。加强培养测试人员的用户体验意识，让他站在用户的角度去思考产品的每一个特性，确保为测试用例建立正确的判断依据。

（3）软件功能需求规格说明书、产品设计文档等，是测试用例设计的主要参考文档。这些文档对产品特性的描述方法、格式和详细程度，也会影响到测试用例的设计。

（4）测试的方法对测试用例的设计影响非常大。在测试用例的设计思路上，白盒测试方法和黑盒测试方法是从不同的哲学思想来解决问题的，前者从内部逻辑思路来考虑，后者从外部功能思路来考虑。

（5）测试的对象。客户端软件和服务器端系统、分布式系统和集中式系统、异步系统和同步系统等，其测试用例的侧重点或测试剖面是不同的，它们从不同的侧面去发现软件系统的弱点或薄弱环节。

（6）软件实现所采用的技术。不同的技术，要进行不同深度的挖掘，而且采用的测试工具也不同。

2. 测试用例设计的基本思想

（1）设计测试用例时，要寻求系统设计、功能设计的弱点。测试用例需要确切地反映功能设计中可能存在的各种问题，而不要简单拷贝产品规格设计说明书的内容。

（2）设计正面的测试用例，应该参照设计规格说明书，根据关联的功能、操作路径等设计。而对孤立的功能则直接按功能设计测试用例。基本事件的测试用例应包含所有需要实现的需求功能，覆盖率达 100%。

（3）设计负面的、异常的测试用例，如考虑错误的或者异常的输入，往往可以发现更多的软件缺陷，这显得更为重要。例如，在进行电子邮件地址校验的时候，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，通常会出现因一些字符转义问题而造成异常情况。

4.1.3 测试用例框架的构成

在测试用例设计过程中，首先将焦点集中在其框架的设计上，通过框架的建立，可以更有效地完成所需要的测试用例设计工作，包括：

- 可以根据产品特性建立起测试用例之间的关系。
- 容易创建为测试执行的任务。
- 提高测试执行的效率。
- 容易进行测试覆盖率的度量，提高测试的可视性。
- 更容易操作测试用例的评审。
- 更好地组织和维护测试用例。

根据产品线、产品特性的分类、质量需求或测试目标来建立测试用例的框架，如图 4-1 所示。测试用例框架，将功能性测试目标和非功能性测试目标分开，对于功能性测试用例，可以从产品特性的分类——功能模块来划分；而对于非功能性测试用例，则从具体的质量特性需求来划分。

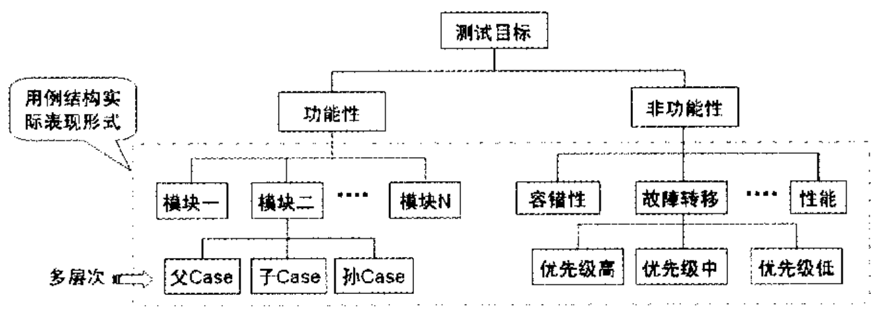


图 4-1 测试用例框架示意图

接着，通过测试用例的层次来表现模块中的子功能或子项的属性，也可以通过测试用例的优先级来设定测试执行的先后顺序。图 4-1 给出了一个测试项目的测试用例框架的完整结构，而图 4-2 则是两个例子，一是按测试类型设计的测试用例框架，另外一个是按程序功能设计的测试用例框架。

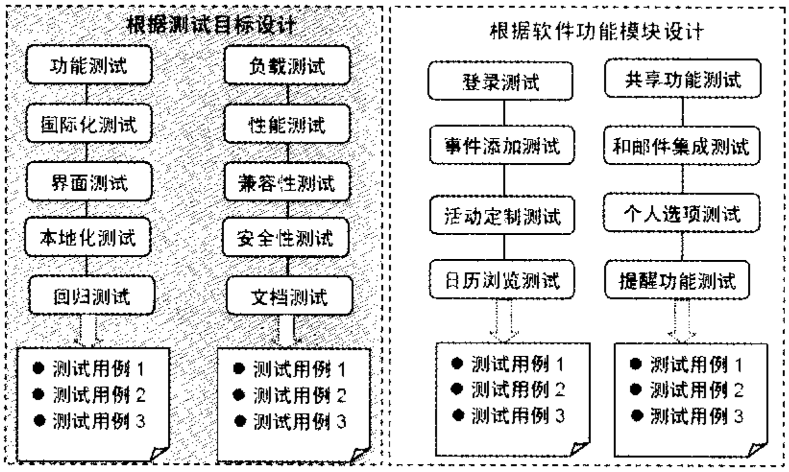


图 4-2 测试用例框架的两种方式示例

测试用例分为多个层次是很明显的，例如，在"登录"功能模块中，可以分为 3 个层次，如下例所示。

实例

用户身份合法性验证 （父case）

- **用户名验证** （子case）
  - 正常用户名的输入 （孙case）
  - 含有特殊字符的用户名
  - 字母大小写无关性测试
  - 非法用户名
- **口令验证**
  - 正常口令
  - 含有特殊字符的口令
  - 字母大小写敏感性测试
  - 口令有效期验证
  - 口令保存
  - 忘记口令后找回口令功能

优先级越高的测试用例，越应优先得到测试，并尽早地、更充分地被执行。测试用例的优先级是由下列 3 个方面决定的：

（1）从客户的角度来定义的产品特性优先级，那些客户最常用的特性或者是对客户使用或体验影响最大的产品特性，都是最重要的特性，其对应的测试用例优先级也最高。根据 80/20 原则，大约 20%的产品特性是用户经常接触的，其优先级高。

（2）从测试效率角度看，边界区域的测试用例相对正常区域的测试用例优先级高，因为更容易在边界区域发现软件的缺陷。

（3）从开发修正缺陷角度看，逻辑方面的测试用例比界面方面的测试用例的优先级高，因为开发人员修正一个逻辑方面的缺陷更难、时间更长或改动范围更大。这种修改，不仅仅是程序代码的修改，而且可能涉及到软件设计上的变更。

4.1.4 测试用例的元素

测试用例是对测试场景和操作的描述，所以必须给出测试目标、测试对象、测试环境要求、输入数据和操作步骤，概括为 5W1H。

- 测试目标：Why ——为什么而测？功能、性能、可用性、容错性、兼容性、安全性等。
- 测试对象：What ——测什么？被测试的项目，如对象、函数、类、菜单、按钮、表格、接口、整个系统等。
- 测试环境：Where ——在哪里测？测试用例运行时所处的环境，包括系统的配置和设定等要求，也包括操作系统、浏览器、通讯协议等单机或网络环境。
- 测试前提：When ——什么时候开始测？测试用例运行时所处的前提或条件限制。
- 输入数据：Which ——哪些数据？在操作时，系统所接受的各种可变化的数据，如数字、字符、文件等。
- 操作步骤：How ——如何测？执行软件和程序的先后次序步骤等。如打开对话框、点击按钮等。

这还不够，因为缺少一个评判的依据。如果没有评判标准，在执行完测试用例后，就不能根据测试结果来确定测试是否通过。所以，每个测试用例必须说明其输出标准——所期望的输出结果。所期望的输出结果，实际就是测试的验证点，每个测试用例最好只有一个验证点。

除了用例的基本描述信息之外，还需要上面讨论的用例框架所需的信息——所述模块、优先级、层次。为了今后管理方便，还要加上其他信息，如测试执行的预估时间、关联的测试用例，是否为自动化测试类别、关联的缺陷等。

综上所述，使用一个数据库的表结构来描述测试用例的元素，如表4-1所示。

表4-1 测试用例的元素列表

字段名称	类 型	注 释
标志符	整型	唯一标识该测试用例的值，自动生成
测试项	字符型	测试的对象，可以从软件配置库中选择
测试目标	字符型	从固定列表中选择一个
测试环境要求	字符型	可从列表中选择，如果没有，则直接输入新增内容
前提	字符型	事先设定、条件限制，如已登录、某个选项已选上
输入数据	字符型	输入要求说明或数据列举
操作步骤	字符型	按1, 2, … 操作步骤的顺序，准确详细地描述
期望输出	字符型	
所属模块	整型	模块标识符
优先级	整型	1，2，3 （1－优先级最高）
层次	整型	0，1，2，3（ 0－ 最高层）
关联的测试用例	整型	上层（父）用例的标识符
执行时间	实型	分钟
自动化标识	布尔型	T，F
关联的缺陷	枚举型	缺陷标识符列表

4.2 功能测试用例的设计

功能测试，依据产品设计规格说明书，主要采用黑盒测试方法，不需要看源代码，而是通过直接运行软件来验证每个功能是否都能正常使用，是否满足用户的要求。功能测试的过程，就是通过数据输入来驱动功能的运行并最终获得数据输出的过程，所以功能测试也被称为数据驱动测试方式。

在进行功能测试用例设计时，一般遵守下列操作的流程。

- 模块划分。
- 确定哪些模块适合手工测试，哪些模块适合自动化测试。
- 根据功能结构及其关系，进行模块层次划分，抓住测试点。
- 首先设计最上层的测试用例，然后再向下逐层推进。
- 最后是测试用例的评审。

概念

黑盒测试方法（Blake-box Testing），是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来考察数据的输入、条件限制和数据输出，进而完成测试。黑盒测试方法，指根据用户的需求和已经定义好的产品规格，针对程序接口和用户界面进行测试，检验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。

白盒测试方法（White-box Testing），也称结构测试或逻辑驱动测试，就是清楚软件产品内部的逻辑结构和工作过程，针对程序语句、路径、变量状态等来进行测试。例如，检验程序的各个分支条件是否得到满足，检查程序是否按照事先预定的路径执行。白盒测试的主要方法有逻辑覆盖、分支覆盖、条件组合覆盖、基本路径测试等。

黑盒测试方法和白盒测试方法的对比，如图4-3所示。

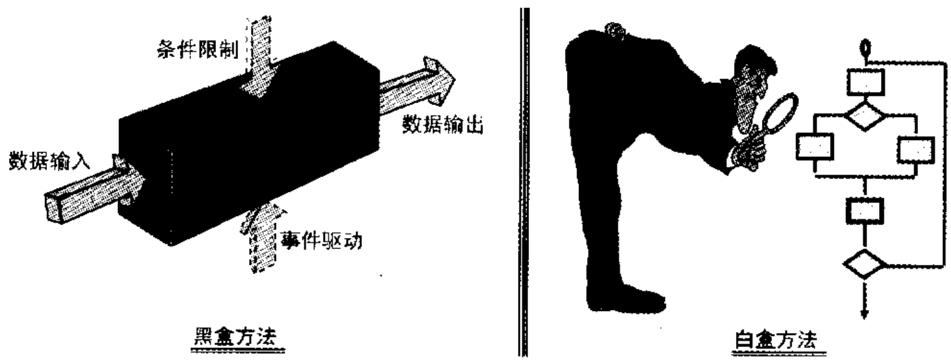


图 4-3 黑盒测试方法和白盒测试方法的对比示意图

4.2.1 功能测试用例的内容

功能测试用例，一般按照功能模块来组织，系统具有的所有功能都要得到测试，所以针对不同的应用系统，其功能测试的内容差异很大。如果把功能测试的内容抽象出来，功能测试的内容可以归为界面（UI）、数据、操作、逻辑、接口等几个方面的测试，通过这些测试，使其最终符合功能规格说明书的要求。

（1）界面测试，指测试系统界面整体布局的合理性，以及是否清晰、美观，包括颜色搭配、字体、文字是否对齐、图片大小与位置、弹出窗口的位置是否合适。其次，还会测试用户是否可以调整布局，是否可以自定义界面，包括文字、图片、颜色等。

（2）数据测试。指接受正确的数据输入，并对异常数据的输入有提示和容错处理。

- 数据输出结果正确，格式清晰和整齐。
- 数据的处理，要提供合适的保存和备份的功能。
- 能提供多种快速、方便的查询功能。
- 数据从输入到最终输出，符合数据流设计，正确地完成一个完整的数据处理过程（端到端的测试，end-to-end Testing）。
- 软件升级后，能继续支持旧版本的数据。

（3）操作测试。内容包括所有菜单、按钮的设计须符合操作习惯，能对操作有正确的响应，而且操作灵活、符合用户的习惯，例如程序安装、启动之后，有相应的提示框来告知用户此操作的状态——是成功还是失败。所有重要的操作，都应该允许用户后退到上一步骤，对不正确的、异常的操作要通过提示框及时提示或警示用户。危险的操作（如删除文件、修改重要数据等）需要进一步确认后，才被执行。经常性的操作应该有多个入口。按钮或菜单在不满足操作条件时，应变灰或暗淡显示，否则，就应该有提示。

（4）逻辑测试。这个测试的目的是使逻辑合理、清楚、流畅、不复杂。如果某个操作需要多个步骤实现，应该有清楚的提示，或者有一个向导来帮助用户完成。某项功能，其不同操作的路径也不一样，但逻辑上要保持一致。系统的各种状态应按照业务流程变化，并保持稳定。

（5）接口测试。这个测试是让接口能配合多种硬件周边设备，以及所需的第 3 方软件接口或公共接口的需要。不管是内部应用接口，还是外部应用接口，应保持其规范性、一致性和完备性。接口还须是可定义或可配置的，应具有良好的兼容性和扩充性。

4.2.2 功能测试用例的设计方法

在设计测试用例的过程中，测试人员应与产品人员、开发人员实时沟通，并使用产品的初级版本（Code drop），不断加深对产品功能的理解。沟通、试用产品可以说是测试用例设计的基本方法，因为只有对产品功能真正理解之后，才能对症下药，设计出有效的测试用例。

下一步，就是借助 UML 视图、逻辑结构图、数据流图等（如图 4-4 所示）进行软件的结构层次、数据流程或操作流程的分析。除了这些图，还可以借助决策表、功能点层次列表等更好地设计测试用例。

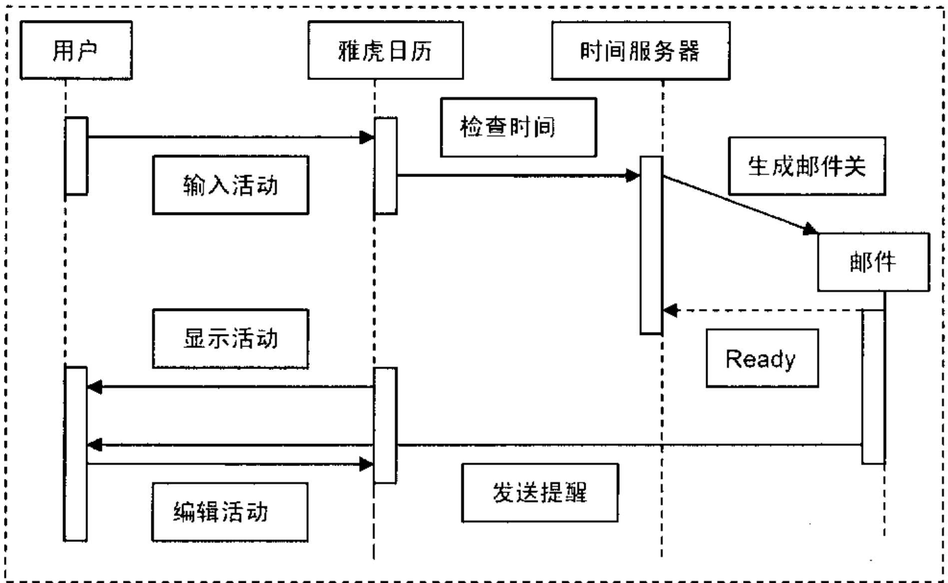


图 4-4 雅虎日历的提醒功能的序列图

UML 视图、逻辑结构图、数据流图等方法，是站在比较宏观（high-level）的高度，从不同的角度全面地理解产品特性，以帮助测试用例的设计。如果从微观或细节（Low Level）的角度看，软件测试用例的设计方法主要有：

- 等价类划分法。
- 边界值分析方法。
- 因果图。
- 功能图。
- 错误推测方法。

4.2.3 等价类划分法

等价类划分法是功能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。

等价类划分设计方法是把所有可能的输入数据，即程序的输入数据集合划分成若干个子集（即等价类），然后从每一个等价类中选取少数具有代表性的数据作为测试用例，如图 4-5 所示。

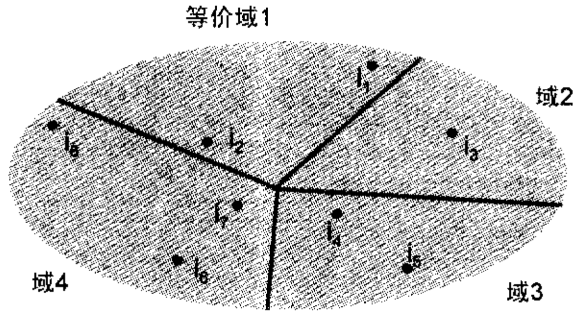


图 4-5 测试用例设计的等价类方法

在采用等价类划分法设计测试用例的过程中，一般会经过两个步骤：分类和抽象。

- 首先是分类，即将输入域按照相同特性或者类似功能进行分类。
- 然后进行抽象，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。

在等价类中，各个输入数据对于揭露程序中的错误是等效的，具有等价的特性，所以表征该类的数据输入将能代表整个数据子集——等价类的输入。因此可以合理假定：

测试某等价类的代表值就等效于对这一类其他值的测试

举个例子，需要为一个对所有实数进行开方运算的程序设计测试用例。这时，需要将所有的实数（输入域）进行划分，可以分成：

- 正实数，+3.1415926 代表正实数。
- 负实数，−1.44444 代表负实数。
- 0

则+3.1415926、−1.44444 和 0 就是 3 个等价类的特征值。

在进行等价类划分的过程中，不但要考虑有效等价类划分，同时还要考虑无效的等价类划分，如图 4-6 所示。使用无效等价类，可以测试程序的容错性——对异常情况的处理。

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可以进行容错处理，并得到保护，这样，软件运行才能稳定和可靠。

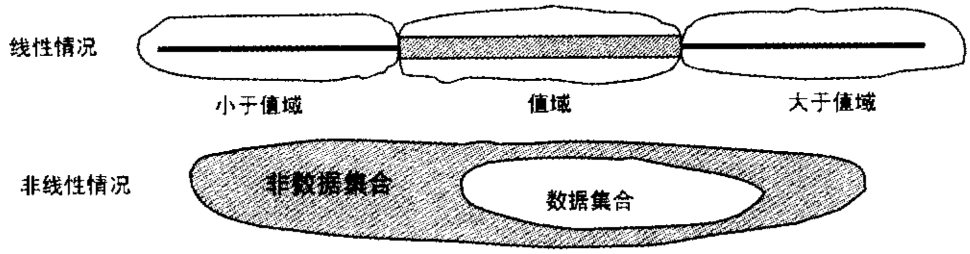


图 4-6 有效等价类划分和无效等价类

实例

Yahoo日历的浏览功能

- 每日浏览，工作时间（8:00—17:30）期间的时间是一个等价类，非工作时间是另外一个等价类。
- 每周浏览，周一到周五可看成一类，而周六和周日可以看成不同的一类。
- 每月浏览，每月有 31 日（1、3、5、7、8、10、12 月）可归为一类，每月只有 30 日（4、6、9、11 月）归为第 2 类，而 2 月份只有 28 天或 29 天（闰年）需单独归为一类。

还有，在创建事件或活动时，名称中输入的文字内容被归为有效数据的等价类，名称中什么都没输（空）就点击“添加”，应归为无效数据的等价类。

在进行等价类划分时，一般都可以归为下列几种情况。

（1）输入数据是布尔值，这是非常特殊的情况，有效等价类只有一个值——真（True），无效等价类也只有一个值——假（False）。

（2）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，程序输入数据要求是两位正整数 x，则有效等价类为 $10 \leqslant x \leqslant 99$，两个无效等价类为 $x<10$ 和 $x>99$。

（3）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，邮政编码则必须是由 6 位数字构成的有效值，其有效集合是清楚的，对应存在一个无效的集合。

（4）在规定了一组列表形式（n 个值）的输入数据，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，把我国的直辖市作为输入值，则等价类是一个固定的枚举类型{北京，上海，天津，重庆}，而且要针对各个城市分别取出相对应的数据，此时无效等价类为非直辖市的省、自治区等。

（5）更复杂的情况是，输入数据只是要求符合某几个规则，这时，可能存在多个有效等价类和若干个无效等价类。例如，邮件地址和用户名的输入。

- 要求输入 26 个英语字母和 10 个阿拉伯数字构成的、长度不超过 20 位的用户名。
- 有效的 E-mail 地址，必须含有"@"，@后面格式为 x.x，E-mail 地址不能带有一些特殊符号，如 "/\#'& 等。

在不满足上述情况时，可能需要进一步细分等价类，包括有效的等价类和无效的等价类。

在使用等价类划分方法设计测试用例时，应先划分有效的等价类和无效的等价类，然后：

（1）为每个等价类规定一个唯一的编号，一个等价类至少存在一个测试用例。

（2）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。

（3）对无效等价类，可以做相同处理。

概念

- **有效等价类**：*指输入完全满足程序输入的规格说明，是由有效且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由无效的输入数据构成的集合。*

4.2.4 边界值分析法

顾名思义，边界值分析法是指对输入的边界条件进行分析，设计出针对边界值的测试用例。因为在实际软件设计和编程中，开发人员往往容易忽视边界条件，这样大量的错误就出现在数据输入或输出范围的边界上。如除法运算中除数为 0 的数据溢出、数组变量中第一个元素和最后一个元素由于没有被赋值而出错。因此，在测试用例的设计中，对输入

的条件进行边界条件分析而且确定边界值，对提高测试效率是非常有帮助的。只有边界值确定下来了，才能划分出有效等价类和无效等价类。所以说，边界值分析方法是对等价类划分法的补充。在测试中，会将两者方法结合起来共同使用。

个人所得税处理的程序，就是边界值分析方法和等价类划分法结合起来使用的典型例子，如表 4-2 所示。

表 4-2 依赖于边界值的等价类划分

个人月收入（x）	税率
x < 0	无效输入
x <=1600	0%
1600< x < = 2100	5%
2100 < x < = 3600	10%
3600 < x < = 6600	15%
6600 < x < = 21600	20%
21600 < x < = 41600	25%
……	……
>101600	45%

边界条件首先体现在数值上，如表 4-2 所示，这些边界值比较容易理解，也比较容易确定。实际上，在软件中，边界条件无处不在，不仅存在于一些数据存储和运算中，还存在于“字符、位置、尺寸、操作、逻辑条件”等中，而这些类型的边界值存在于“最大/最小、首位/末尾、上/下、最高/最低、最短/最长、空/满” 之中。

对于测试用例的设计，不仅要取边界值作为其中的测试数据，而且要选取刚刚大于和刚刚小于边界值的数据作为测试数据，如图 4-7 所示。假如 0 可以看作绝对值函数 ABS(x) 的边界值，则测试数据是 3 个，即－1，0，1。边界值附近的数据确定方法如表 4-3 所示。

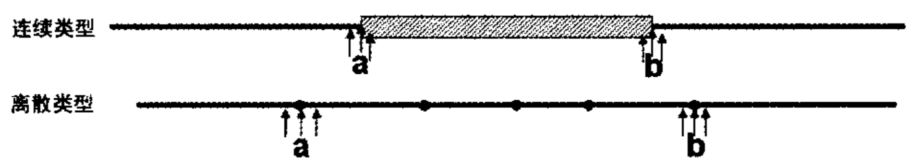


图 4-7 输入数据边界值的典型表现

表 4-3 边界值附近数据的几种确定方法

项	边界值附近数据	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；不输入字符（0 个）和输入 256 个字符就被作为无效等价类
数值	开始位-1/结束位+1	如数据的输入域为 1～999，其最小值为 1，而最大值为 999，则 0、1000 则刚刚好在边界值附近
空间	小于零空间一点/大于满空间一点	测试软盘的数据存储，使用比剩余磁盘空间大一点（几 KB）的文件作为测试的边界条件附近值

1. 数值的边界值检验

计算机内部数据是以二进制存储和计算的，因此，许多不同类型（整型、长整型、布尔型、数组、字符串长度等）的数据都受到一定的限制，具有很强的二进制特征，如表 4-4 所示。

表 4-4 各类二进制数值的边界

项	范围或值
位（Bit）	0 或 1
字节（Byte）	0～255 （2^8 个）
整型（Word）	0～65 535（2^{16} 个）
长整型	0～4 294 967 295（2^{32} 个）
千（K）/ 兆（M）	1 024/ 1 048 576（$2^{10}/2^{20}$）
千兆（G）	1 073 741 824（2^{30}）
万亿（T）	1 099 511 627 776（2^{40}）

在数值的边界值条件检验中，就可以参考表 4-3 进行。如对字节进行检验，边界值条件可以设置成 254、255 和 256。

2. 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 4-5 中列出了一些简单的 ASCII 码对应表：

表 4-5 字符和 ASCII 码值的对应关系

字 符	ASCII 码值	字 符	ASCII 码值
空（NULL）	0	A	65
空格（SPACE）	32	B	66
斜杠（/）	47	Y	89
0	48	Z	90
9	57	左中括号 [	91
冒号（：）	58	反斜杠（/）	92
分号 （；）	59	右中括号]	93
<	60	单引号（'）	96
=	61	a	97
>	62	b	98
?	63	y	121
@	64	z	122

在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，如小写字母 a 和大写字母 A、空和空格的 ASCII 码值是不同的，而且它们处在边界上，斜杠、冒号、@、左中括号和单引号恰好处在阿拉伯数字、英文字母的边界值附近。

3. 其他边界值检验

一些特殊的值，如默认值、空值、空格、未输入值、零，可以被认为是边界值。在字符编辑域、多选择项上，都存在这样的特殊边界值，或者可以看作是边界值的延伸。例如，如图 4-8 所示的"Justified"、"（none）"、"0 line"就是一些边界值，是测试的重点。

对于多选项，如图 4-9 所示，也是可以用边界值方法来进行测试用例设计，可以设置 3 个边界条件：

- 选上所有选项（最大值）。
- 不选上任何一项（空，零）。
- 只选一项（最小值）。

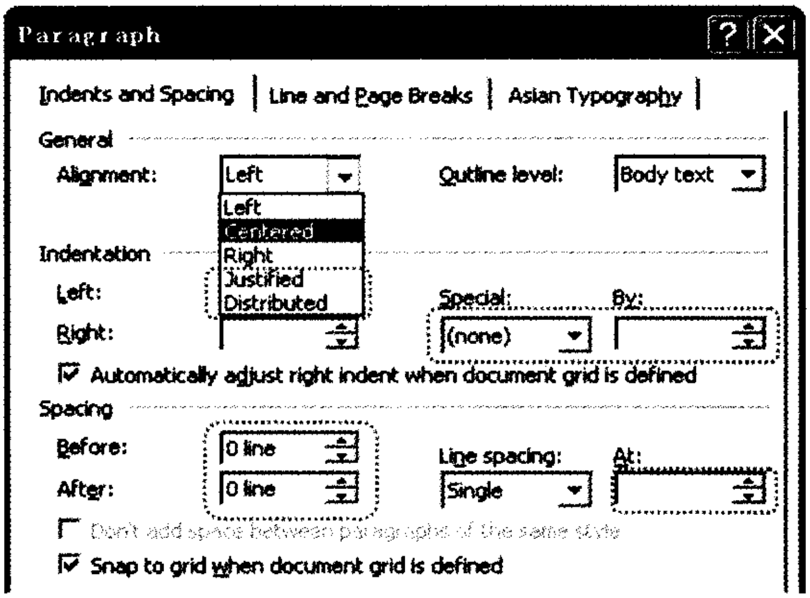


图 4-8 字符编辑域、多选择项的特殊边界值

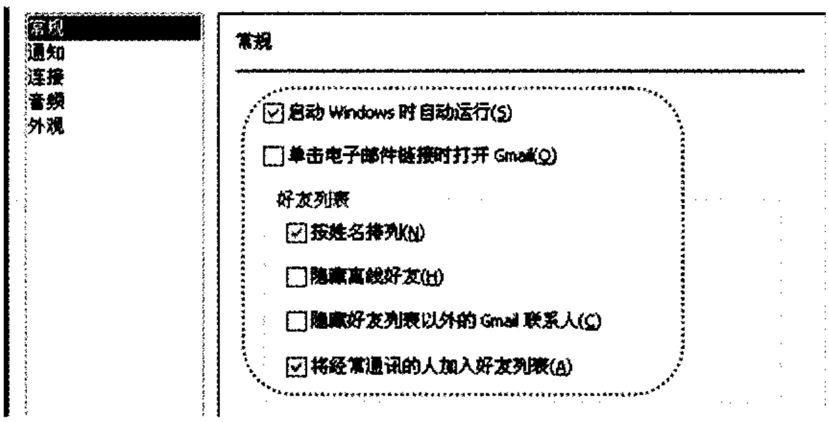


图 4-9 多选项的情况示意图

4.2.5 因果图法

等价分类法和边界分析法，主要是针对单个输入数据来设计测试用例的，没有考虑多

种输入数据的组合情况。组合情况是造成软件缺陷的主要方面之一，也是重要的测试点。检验各种输入条件的组合并非一件很容易的事情，即使能将所有数据输入的边界值确定下来并划分成等价类，数据输入的条件组合情况还是相当多。因果图法就是利用图解法分析软件输入（原因）和输出条件（结果）之间的关系，以设计测试用例的方法。因果图法适合于检查程序输入条件的多种情况的组合，并最终生成判定表，来获得对应的测试用例。

如何通过因果图法来生成测试用例呢？因果图法，一般需要经过下面4个步骤来完成，如图4-10所示。

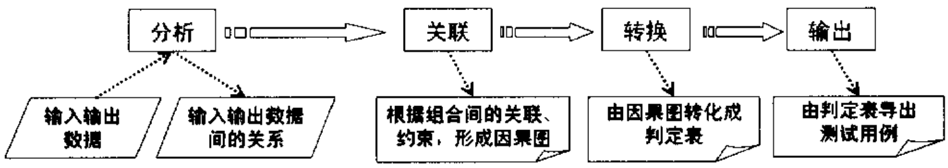


图4-10 因果图法示例

（1）分析软件规格说明书中的输入输出条件并划分出等价类，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多个输入因素之间的关系。

（2）找出输入因素与输出结果之间的关系，将对应的输入与输出关系关联起来，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）由因果图转化成决策表。任何由输入与输出之间关系构成的路径，都成为决策表的一列，也被视为决策表的一条规则。

（4）将决策表的每一列拿出来作为依据设计测试用例。一般来说，决策表的每一列对应一条测试用例。

概念

决策表（Decision Table）是分析和表达较为复杂逻辑条件下软件状态和行为的有效工具，一般和因果图法结合起来使用。利用决策表能够设计出完整的测试用例集合，能够将复杂的问题按照各种可能的情况全部列举出来，简明并避免遗漏。决策表通常由4个部分组成：

- 条件桩（Condition Stub），列出问题的所有条件，不强调条件的次序关系。
- 动作桩（Action Stub），列出问题规定可以采取的操作。
- 条件项（Condition Entry），列出针对条件桩的取值，即在所有可能情况下的真假值。

- 动作项（Action Entry），列出在条件项的各种取值情况下应该采取的动作。

任何一个条件组合的特定取值及其相应要执行的操作，可视为一条规则，最终表现在决策表中，即决策表的一列就是一条规则，贯穿条件项和动作项。显然，一条规则就是一条测试用例，决策表中列出多少组条件取值，也就有多少条规则和相应数量的测试用例。

Beizer 指出了适合使用因果图、决策表来设计测试用例的条件：

- 条件或影响因素没有严格的的排列顺序，即排列顺序对结果影响很小。
- 规则的排列顺序也不会影响测试的最后结果。
- 规则具有很强的独立性，即每当某一规则的条件已经满足并确定其动作（执行的操作）后，不必检验别的规则。

4.2.6 功能图法

功能图法，是和因果图相对应的一种测试用例设计方法。因果图将输入作为因素、将输出作为结果，以构造输入和输出之间的关系——因果图，处理软件功能的静态说明。功能图是描述程序状态变化、转移的过程，因为软件运行或操作的过程可以看作是其状态不断发生变化的过程。测试用例的设计就是如何覆盖所有软件所表现出来的状态，即在满足输入/输出数据的一组条件下，软件运行是一系列有次序的、受控制的状态变化过程。

使用功能图法设计测试用例，是借助功能图模型来实现的，而功能图模型由状态迁移图和逻辑功能模型组成：

- 在状态迁移图中，状态指出数据输入的位置（或时间），而迁移则指明状态的改变。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。
- 逻辑功能模型由布尔函数组成，它是要依靠决策表或因果图表示的逻辑功能。逻辑功能模型用于表示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静态说明，输出数据仅仅由输入数据决定。

以雅虎日历的登录页面为例，简单说明状态迁移图的应用。从等待状态开始，会达到成功状态或失败状态，并找出导致这种状态改变的输入条件，如图 4-11 所示。针对同样的例子，可以用决策表来描述逻辑功能模型，如表 4-6 所示。

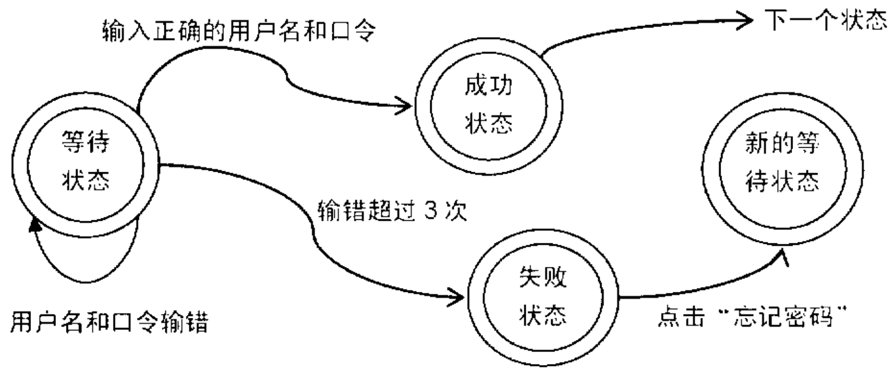


图 4-11 状态迁移图的示例

表 4-6 逻辑功能表的示例（1-成功，0-失败）

输入	正确的用户名	错误的用户名	错误的用户名	正确的用户名
	错误的密码	正确的密码	错误的密码	正确的密码
输出	0	0	0	1
	错误提示	错误提示	错误提示	
状态	等待重新输入	等待重新输入	等待重新输入	进入新的状态

在状态图中，如果用节点代替状态，用弧线代替状态迁移和变化，则功能图就可转化为流程图形式。有了流程图，测试用例设计的思路也就很明显了，就是要覆盖流程的分支和路径。在逻辑功能表中，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：

- 局部测试用例由原因组合（输入数据）与对应的结果值（输出数据）构成。
- 整体测试用例，由从初始状态到最后状态的测试路径构成。

4.2.7 错误推测法

在某些复杂的情况下，上述方法都不能奏效。在没有适当的方法时，就不得不采用这种推测法。推测法主要依赖经验、直觉来做出简单的判断甚至是猜测，给出可能存在缺陷的条件、场景等，在找到缺陷之后，设计出相应的测试用例。

例如，测试雅虎日历的“快速添加活动”功能，在输入活动名称后，设置完成期限早于当前日期，如果能成功地添加这个活动，就是一个缺陷，因为这种活动是不现实的或不合理的。另外，也可以猜测是否可以将完成日期设置为 2007 年 2 月 30 日，如果可以输入

这个日期，也是问题。

采用错误推测方法时，应尽量列举程序中所有可能出现的错误和值得怀疑的地方，从中作出选择，以设计测试用例。可以利用不同测试阶段的经验和对软件功能特性的理解来进行测试用例的设计，例如：

- 若在单元测试中程序模块测试曾经出现了错误，在后期功能测试中可以列出这些可能出现问题的地方，设计相应的测试用例。
- 根据前一个版本中发现的常见的错误，有针对性地为当前版本设计测试用例。
- 在应用软件中可能出错的环节，如C++程序的内存泄漏、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，需要特别对待处理。

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，回归测试中发现新的问题。
- 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；由修复软件的错误推测会带来的问题。

4.2.8 正交实验设计方法

在实际的软件项目中，作为输入条件的原因非常多，这时，如果用因果图分析，很难理出一个头绪，即使好不容易画出一个巨大的因果图，其组合数也是一个非常大的数字，测试的工作量非常之大，以至于时间和人力资源不允许而无法执行。为了有效地、合理地减少输入条件的组合数，降低工作量，可以利用正交实验设计方法进行测试用例的设计。

用正交实验设计方法来设计测试用例，其主要步骤是：

（1）对软件需求规格说明中的功能要求进行划分（层层分解与展开），分解成具体的、相对独立的基本功能。

（2）根据基本功能的质量需求，找出影响其功能实现的操作对象和外部因素，每个因素的取值可以看作水平，多个取值就存在多个水平。例如，有操作系统、浏览器和防火墙等3种因素影响，每种因素有4种取值，即：

- 操作系统：Windows XP、Linux、Mac X、Solaris
- 浏览器：IE6、IE7、FireFox、Mozilla
- 防火墙：ISA 2004、Blue Coast、Cisco PIX、Linux squid

（3）确定待测试软件中所有的因素及其权值，这是测试用例设计的关键，确保全面、

准确。权值是依据各因素的影响范围、发生的频率和质量的需求等来确定的。

（4）加权筛选，生成因素分析表。

（5）利用正交表构造测试数据集，正交表的每一行，就是一条测试用例。考虑交互作用不可忽略的处理因素和不可混杂的原则，有交互作用的组合优先安排。

利用正交实验设计方法设计测试用例，可控制生成的测试用例数量，覆盖率高且测试效率高。

概念

正交试验设计（Orthogonal experimental design）是研究多因素多水平的一种设计方法。它是基于Galois理论，根据正交性从全面试验中挑选出部分有代表性的点进行试验的。这些有代表性的点具备了“均匀分散，齐整可比”的特点。日本著名的统计学家田口玄一将正交试验选择的水平组合列成表格，称为正交表。例如，一个3因素3水平的实验，按全面实验要求，须进行3^3=27种组合的实验，但是按正交表设计实验，只需要进行9次，显然大大减少了工作量。

正交表具有以下两项性质，如表4-7所示：

- 每一列中，不同的数字出现的次数相等。在表 4-6 中，任何一列都有“1”、“2”、“3”，且在任一列的出现数均相等。
- 任意两列中数字的排列方式齐全而且均衡。任何两列（同一行内）有序对共有 9 种，1.1、1.2、1.3、2.1、2.2、2.3、3.1、3.2、3.3，且每对出现数也均相等。

表 4-7 3 水平 4 因素正交表 L_9（3^4）

因素 水平 试验号	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

伽罗华（Galois，1811—1832）是法国青年数学家，以其超越时代的天才思想提出了群的概念，并用群论彻底解决了根式求解代数方程的问题——即将全部代数方程可解性问题转化或归结为置换群及其子群结构分析的问题，由此发展了一整套关于群和域的理论。为了纪念他，人们称之为Galois理论。正是这套理论创立了抽象代数学，把代数学的研究推向了一个新的里程，而群论对数学分析、几何学、结晶学、理论物理学等领域产生了深远的影响，标志着现代数学的开始。

4.3 系统测试用例的设计

在第 3 章系统部署设计验证的过程中，实际上已经接触到系统测试用例的设计了，例如，用量分析的结果，可以帮助进行负载测试的测试用例设计；而故障转移（Fail-over）的全面理解，有助于帮助进行故障转移本身的测试，它是系统可用性、可靠性测试不可缺少的部分。不必强调的一点是，由于系统的基础平台、架构和构成等差异很大，所以测试用例设计方法也有较大的差别。

下面，我们可以讨论故障转移、Web 安全性的测试用例设计，了解了这两个方面的内容，对其他方面的系统测试用例设计也有较大的帮助。关于负载测试、性能测试等方面内容，详见第 9 章。

4.3.1 故障转移的测试用例设计

在服务器端，故障转移功能是不可缺少的，其测试的重要性随着 SaaS 应用的普遍增多而显得越来越突出。在第 3 章的 3.3.3 节中，进述了软件部署时，如何考虑故障转移的实现，包括应用域内服务器之间的本地故障转移，以及域与域之间、服务器集群之间的异地故障转移。

首先，针对应用域内服务器之间的本地故障转移，如何设计测试用例呢？不妨将第 3 章图 3-5 左边的主应用域的物理部署转换成逻辑示意图进行分析，如图 4-12 所示。

在正常情况下，Web 服务器 A 与应用服务器 A 相互协调工作，处理业务流程，他们两者都可以直接存取数据库 A。

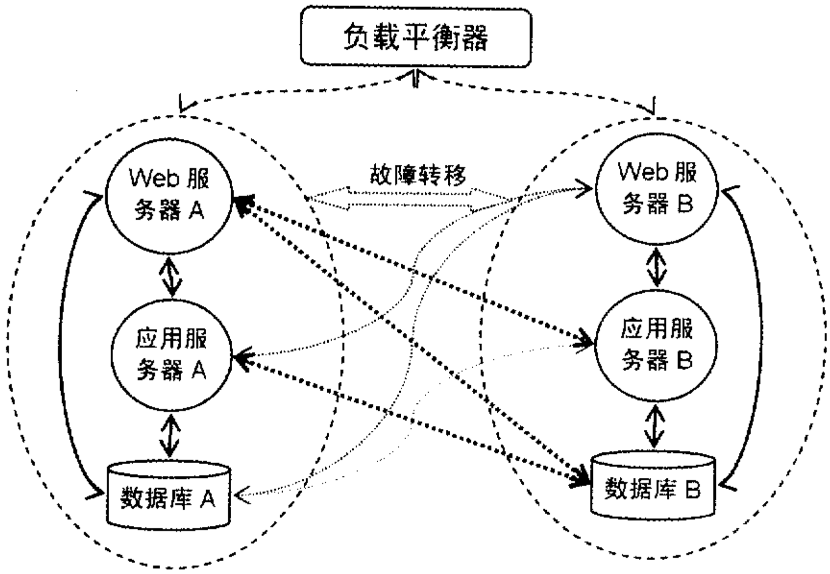


图 4-12 本地故障转移的逻辑示意图

- 当应用服务器 A 发生故障，不能正常工作时，Web 服务器 A 和应用服务器 B 进行通信，相互协作，共同处理业务。
- 如果是数据库 A 发生故障，不能正常工作时，Web 服务器 A 和应用服务器 A 只有存取数据库 B。为了给用户无任何变化的印象，数据库 A 和数据库 B 必须保持同步。
- 如果应用服务器 A 和数据库 A 一起发生故障，这时 Web 服务器 A 和应用服务器 B 通讯并直接存取数据库 B。
- 如果是 Web 服务器 A 发生故障，由负载平衡器处理，即用户只能访问 Web 服务器 B。系统的整体容量、性能等降低，但系统还是可以提供服务。

对 Web 服务器 B、应用服务器 B 和数据库 B，道理一样。

根据这样的分析，就比较容易设计故障转移的测试用例了。如何让服务器发生故障的最简单的方法是拔掉某个服务器的网络线。但有时也不可靠，网络通讯中断和服务器实际发生故障的判断可能是不同的，除非都以网络连接超时作为标准。最终，故障转移的测试用例设计如表 4-8 所示。

表 4-8 故障转移测试用例的典型示例

序 号	用例名称	前 提	步 骤	期望结果
1	应用服务器故障转移	应用服务器A不工作	启动服务; 进行操作使之满足前提; 观察结果; 一段时间后，消除故障; （重新）启动服务; 再观察结果	Web服务器A连接到应用服务器B，而且功能正常，但Web服务器A还是可以存取数据库A，应用服务器B没有改变，访问数据库B。故障消除后，Web服务器A又自动连接到应用服务器A
2	数据库故障转移	数据库A不工作		Web服务器A和应用服务器A都连接到数据库B，而且功能正常。故障消除后，又自动连接到数据库A
3	应用服务器和数据库故障转移	应用服务器A和数据库A都不工作		Web服务器A连接到应用服务器B和数据库B，而且功能正常。故障消除后，恢复原状，又自动连接到应用服务器A和数据库A
4	Web服务器故障转移	Web服务器A不工作		所有用户访问Web服务器B，而且功能正常，应用服务器B正常，没有通讯连接上的变化。故障消除后，又自动连接到Web服务器A

4.3.2 系统安全性的测试用例设计

现在系统越来越复杂，所提供的功能也越来越强大，同时，也就给系统带来更多的不

安全隐患，安全测试受到很大的挑战。安全测试的范围很广，涉及到各个层面，包括数据库和文件的存取、通讯端口、数据传输、缓冲区等。这里以 Web 为例，讨论 Web 安全性测试用例的设计。安全性测试目标可以归为下列两项：

- 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的系统和应用程序。

为了达到上述目标，就需要了解有关系统和数据两个方面可能存在的各种安全漏洞，如口令能否被截取，是否有别的入口（如 API）可以绕过“登录”关口，是否企图趁系统恢复之际非法进入系统等，然后针对这些漏洞去设计测试用例。对于 Web 的安全测试，可以列出其主要的测试范围，然后针对不同的测试范围设计测试用例：

（1）登录测试，所有需要登录的地方都应该受到限制，需要进行用户名和密码的验证和匹配等。登录测试时，需要考虑输入的密码是否有长度和条件限制，最多可以尝试多少次登录，密码失效周期，以及密码能否被欺骗或绕过等。Web 应用系统需要有是否超时的限制，当用户长时间不做任何操作的时候，需要重新登录才能使用其功能。

（2）用户权限测试。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。

（3）数据加密，包括各种场合下的数据加密（如 SSL、PKI 等），如用户将信用卡信息、用户登录密码信息等存储到数据库，或者从页面输入和浏览，以及用电子邮件发送等，都应该在操作前被加密。

（4）其他方面，包括跨站脚本攻击，直接使用服务器脚本语言，目录没有被保护，Session 和 Cookie 的不恰当处理等。

1. URL 和 API 的身份验证

有时在页面中点击按钮、菜单、链接等都很安全，但直接将一些地址输入到新打开的浏览器，可以绕过“登录”页面。这种情况，更多发生在提供给第 3 方的应用接口（API）软件包中。

其次，在 URL 中不能直接传递重要参数，更不能直接传递用户名和密码。即使要传递用户信息，必须设定另外一个独立的用户代号（屏幕名，非真实用户注册名）。在 Web

安全测试中，一定要注意这些。

2. 跨站脚本攻击

谈到了 Web 页面的安全性，不得不谈到跨站脚本攻击（Cross-site script，CSS），攻击者在页面某些输入域中使用跨站脚本来发送恶意代码给没有发觉的用户，以窃取某些资料和信息。所以，要保证页面的安全，所有页面上的输入域都需要验证，以防止 CSS 攻击。输入域的验证包括：

- 数据类型（字符串，整型，实数，等）。
- 允许的字符集。
- 最小和最大的长度。
- 是否允许空输入。
- 参数是否是必须的。
- 重复是否允许。
- 数值范围。
- 特定的值（枚举型）。
- 特定的模式（正则表达式）。

如果验证是否可以进行 CSS 攻击，就要检验下列内容是否不能被输入：

- HTML 标签：<···>···</···>
- 转义字符：`&amp`（&）；`&lt`（<）；`&gt`（>）；` `（空格）；
- 脚本语言，如 JavaScript `<script language= 'javascript' >……</script>`
- 特殊字符：' ' < > /

3. 其他 Web 安全漏洞

- 浏览器缓存，认证和会话数据不应该作为 GET 的一部分来发送，应该使用 POST。
- 程序在抛出异常的时候给出了比较详细的内部错误信息，暴露了不应该显示的执行细节，从而有潜在的危险。
- 攻击者使用缓冲区溢出来破坏 Web 应用程序的栈，通过发送特别编写的代码到 Web 程序中，让 Web 应用程序来执行恶意代码。
- 注入式漏洞。通过附加一个永远为"真"的条件，使系统中某个认证条件总是成立，从而侵入系统，如为 Select * from table A where username = '' +

username+'' and password= .... 加上'or 1=1'，就可以不输入任何 password 进行攻击。

概念

根据ISO 8402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。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：

（1）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。

（2）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。

（3）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。

4.4 测试用例的审查

测试用例的质量关系到测试执行的质量和测试工作本身的质量，那么，如何保证测试用例的质量？首先，要对用户需求、服务质量要求、产品特性有深刻且全面的理解；其次，采取正确、恰当的方法进行用例设计；再者，按照测试用例的标准格式或规范的模板来书写测试用例。除此之外，对测试用例的检查、评审，也是一种提高测试用例质量的主要且有效的手段。

4.4.1 测试用例书写标准

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，列出了与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罗列如下：

- **标志符**（Identification）：每个测试用例应该有一个唯一的标志符，它将成为所有与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 Items）：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列出的特性描述更加具体，例如，我们做 Windows 计算器应用程序的窗口测试，测试对象是整个应用程序的用户界面，其测试项将包括该应用程序的界面特性要求，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，在整个测试模块里面应该包含整个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单独所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期望的输出结果。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，他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，此时我们需要在 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。

4.4.2 测试用例评审要点

测试用例是软件测试的准则，但它并不是一经编制完成就成了准则。测试用例在设计编制过程中要组织同级互查。完成编制后应组织专家评审，需通过后才可以使用。评审委员会可由项目负责人和测试、编程、分析设计等有关人员组成，也可邀请客户代表参加。

在测试用例评审之前，要定义或明确评审的标准。在定义测试用例的评审标准时，首先要清楚什么样的测试用例是好的？好的测试用例应该是：

- 测试范围的覆盖率高，依据特定的测试目标的要求，覆盖所有的测试范围或内容。
- 测试用例设计能反向思维，有效地发现缺陷。测试是为了发现缺陷，能更快地发现缺陷或更有可能发现潜在缺陷的测试用例可提高测试效率。
- 易用性。设计思路容易被理解，测试用例的组织结构合理，测试用例的执行比较

顺畅，操作连贯性好。

- 易读性。前提条件、步骤和期望结果等描述清楚、准确。
- 易维护性。应该以很少的时间来完成测试用例的维护工作，包括添加、修改和删除测试用例。易用性和易读性，也有助于易维护性。

测试用例的评审，可以从测试用例的框架和结构开始，然后逐步向测试用例的局部或细节推进。

（1）为了把握测试用例的框架、结构，要分析其设计思路是否符合业务逻辑，是否符合技术设计的逻辑，是否可以和系统架构、组件等建立起完全的映射关系。

（2）在局部上，应有重有轻，评审时应抓住一些测试的难点和系统的关键点，从不同的角度向测试用例的设计者提问。

（3）在细节上，检查是否遵守测试用例编写的规范或模板，是否漏掉每一元素，每项元素是否描述清楚。

设计测试用例时，应寻求系统设计、功能设计的弱点。测试用例需要确切地反映功能设计中可能存在的各种问题，而不要简单拷贝产品规格设计说明书的内容。测试用例的评审，可以从正、反两方面进行检查。正面测试用例要求全面，反面测试用例要有创造性，思路要开阔。

（1）设计正面的测试用例，应该参照设计规格说明书，根据关联的功能、操作路径等设计测试用例。而对孤立的功能则直接按功能设计测试用例。基本事件的测试用例应包含所有需要实现的需求功能，覆盖率达 100%。

（2）设计负面的、异常的测试用例，往往可以发现更多的软件缺陷，显得更为重要。例如，考虑错误的或者异常的输入。在进行电子邮件地址校验的时候，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。

通过检查表来进行测试用例的评审，也是一种简单而有效的方法，例如，针对检查表中的各项问题，是否都能回答“是”。如果答案都为“是”，意味着测试用例通过了评审。

- 在设计测试用例前，是否先画好 UML 时序图、状态图或数据流程图等？
- 是否有常见错误表供编写测试用例使用？
- 测试用例的设计思路合理吗？与产品设计、技术设计吻合吗？
- 测试用例的结构层次清晰、合理吗？

- 软件需求的所有功能点是否都有正常功能用例对应？
- 是否每个正常用例都有对应的异常用例？
- 测试用例是否覆盖了所有已知的边界值，如特殊字符、最大值、最小值？
- 测试用例是否覆盖了已知的无效值，如空值、垃圾数据和错误错误操作等？
- 测试用例是否覆盖了输入条件的各种组合情况？
- 测试用例是否覆盖了各种安全性问题？
- 测试用例是否覆盖了负载平衡和故障转移等方面的可用性问题？
- 是否考虑了兼容性测试用例？如是否测试了新版本同以前版本的数据、接口的兼容性？
- 是否考虑了关联功能的测试用例？例如，用户修改了自己的邮件地址，那么提醒、报告等是否会发送到新的地址？
- 是否所有的接口数据都有对应的测试用例？
- 测试用例的前提条件、操作步骤描述是否明确、详尽？
- 当前测试是否最小程度地依赖于先前测试或步骤生成的数据和条件？
- 测试用例检查点（验证点）描述是否明确、完备？
- 是否重用了以前的测试用例？

概念

检查表（Check List）是用来帮助评审员找出被评审的对象中可能的缺陷的。检查表是一种常用的质量保证手段，也是正式技术评审的必要工具，评审过程往往由检查表驱动。一份精心设计的检查表，对于提高评审效率、改进评审质量具有很大帮助。

- 可靠性．人们借助检查表以确认被检查对象的所有质量特征均得到满足，避免遗漏任何项目。
- 效率。检查表归纳了所有检查要点，比起冗长的文档，使用检查表具有更高的工作效率．

如何制定合适的检查表呢？概括起来有以下几点：

- ➢ 不同类型的评审对象应该编制不同的检查表。
- ➢ 根据以往积累的经验收集同类评审对象的常见缺陷，按缺陷的（子）类型进行组织，并为每一个缺陷类型指定一个标识码。

> ➢ 基于以往的软件问题报告和个人经验，按照各种缺陷对软件影响的严重性和（或）发生的可能性从大至小排列缺陷类型。
>
> ➢ 以简单问句的形式（回答"是"或"否"）表达每一种缺陷。检查表不易过长。
>
> ➢ 根据评审对象的质量要求，对检查表中的问题做必要的增、删、修改和前后次序调整。

4.5 测试套件的创建

当遇上大量的测试用例时，如何更有效地使用、执行这些测试用例呢？这就需要创建测试套件（Test Suite）。通过测试套件，将服务于同一个测试目的或某一运行环境下的一系列测试用例有机地组合起来。测试套件是按照测试计划所定义的各个阶段的测试目标决定的，即先有测试计划，然后才有测试套件，如图 4-13 所示。

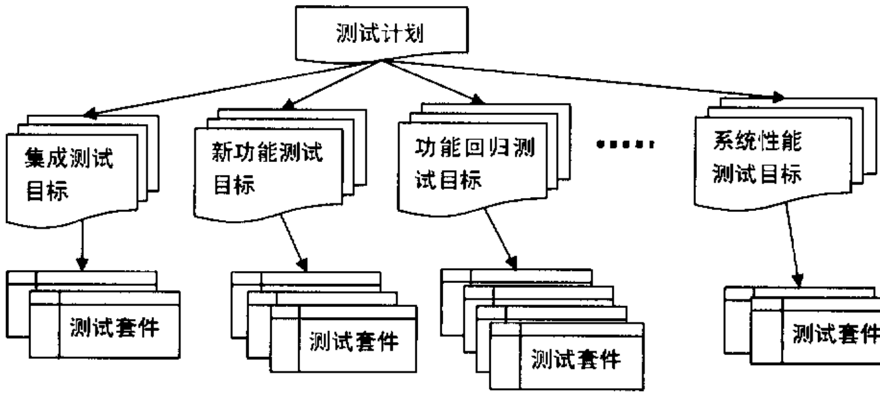


图 4-13 测试计划和测试套件的关系

为了说明测试套件的作用，以"功能测试"来进行讨论。在功能测试中，至少存在以下几种情况需要创建测试套件。

（1）在当前开发的版本中，并不是所有的功能模块都发生了改动，只有某些功能模块发生了变化，所以对于这些模块的测试要优先进行，这些模块要得到足够的测试。我们可以创建由这些改动模块的测试用例构成的测试套件。

（2）在修改的模块时，也不需要选择所有的测试用例，要根据测试用例的优先级来进行测试，优先级高的要先测试，所以针对不同的优先级（I, II, III），就创建不同的测试套件。

（3）由于多数情况下，一个平台（如操作系统或浏览器）上所存在的缺陷会占所有平台上缺陷的80%以上，甚至90%以上，所以开始时，可以创建一个平台上（如Windows Vista）的测试套件。

（4）多数情况下，自动化测试和手工测试是相互并存的，所以有必要为自动化测试、手工测试分别建立测试套件。

（5）在手工测试中，会受到硬件的限制，有的测试人员使用Windows平台，有的测试人员则使用Mac 平台，这时可以建立和测试人员相对应的测试套件。

（6）在回归测试中，可以先运行曾经发现缺陷的测试用例，然后再运行从来没有发现缺陷的测试用例。因为程序的缺陷还是潜在着一定规律的，或者开发人员会存在一定的编程习惯，所以曾经发现缺陷的测试用例更有价值，更容易帮助找到缺陷。这时，需要分别建立测试套件。

（7）有时，还要针对不同的用户群来定义测试目标，也就是建立相应的测试套件。

在把测试用例组织成测试套件时，通用的基本方法有下列3种。根据程序的功能模块进行组织是最常用的方法，同时可以将3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、有优先次序的有效套件，如图4-14所示。

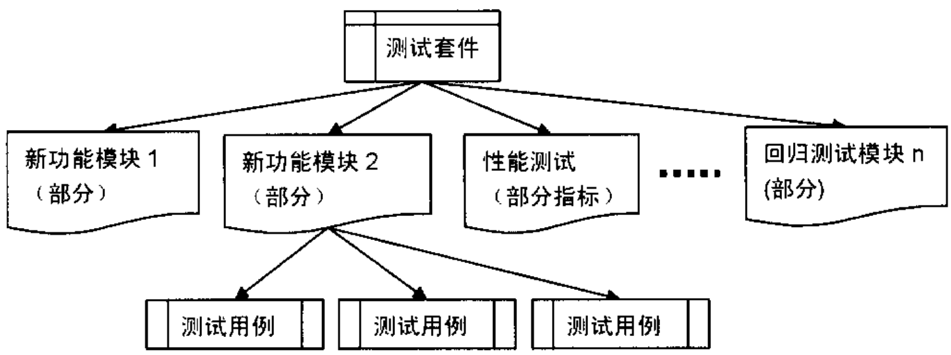


图4-14 测试套件的构成示意图

（1）按照程序的功能模块组织：应用程序的规格说明书一般是按照不同的功能模块进行组织的，因此，按照程序的功能模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，能够很好地检查测试所覆盖的内容，实现准确地执行测试的计划。

（2）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。在一个测试过程中，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起来，形成每个阶段或每个测试目标所需的测试用例组或集合。

（3）按照测试用例的优先级组织：与软件错误相类似，测试用例拥有不同优先级，我们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，从而使测试过程有层次、有主次地进行。

结合测试用例框架、具体测试用例和测试套件，我们就可以获得测试用例一个完整的面貌，如图 4-15 所示。

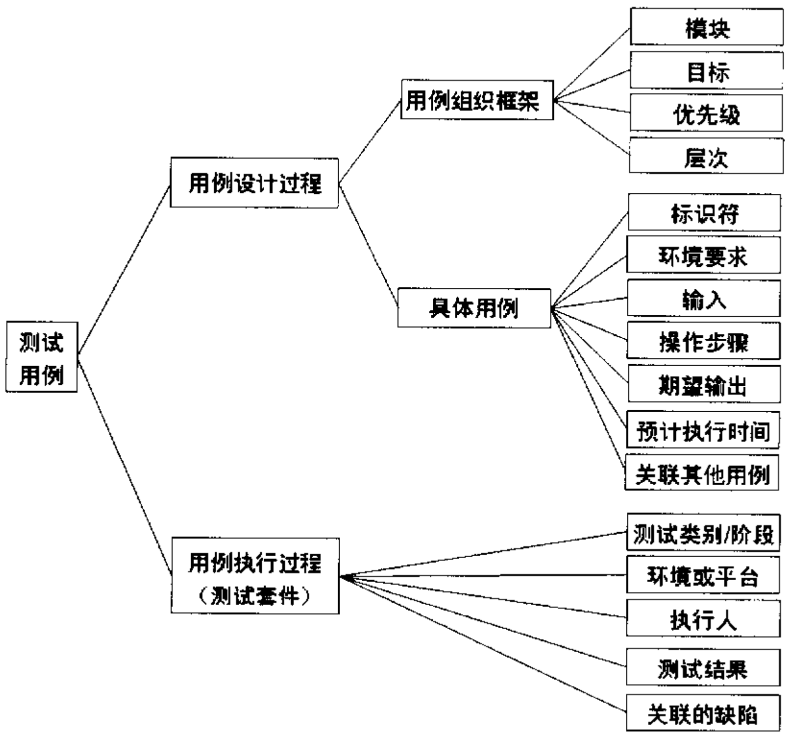


图 4-15　有效的测试用例组织

- 测试用例设计过程包括用例框架、具体测试用例描述。
- 用例框架，包含模块划分、测试目标、优先级和层次。
- 单个用例描述，包含“标识符、测试环境要求、数据输入或条件、操作步骤、所期望的输出”等基本信息，还包括本测试用例预计执行的时间、关联测试用例的

标识（父 Case 或子 Case）等其他信息。

- 用例执行过程，可以通过测试套件体现出来，但测试结果、缺陷等需要在执行后才能记录下来。

概念

测试套件（Test Suite）。根据测试计划的测试目标分类和阶段划分，为不同的阶段、不同的测试目标选用一系列测试用例，并针对特定的测试环境来进行测试，满足质量要求。由一系列测试用例和与之关联的测试环境组合而构成的集合，就是测试套件。

4.6 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，它随着测试过程的进行和完善而逐渐成熟起来的。

在测试用例设计过程中，首先将焦点集中在框架的设计上，通过框架的建立，可以更有效地完成所需要的测试用例设计工作。通过测试框架，可更好地体现测试设计的思路以及有效地组织测试用例。

在功能测试用例的设计方法中，主要介绍了等价类划分法、边界值分析法、错误推测法、因果图法、功能图法、正交实验设计法等，包括举例说明如何将等价类划分法和边界值分析法结合起来使用。

在系统测试用例设计中，通过讨论故障转移、Web 安全性等方面的测试用例设计，了解系统测试用例的设计思路。最后，着重讨论如何做好测试用例的评审工作，包括测试用例的书写标准、测试用例评审的要点和方法，以及比较完整的测试用例评审检查表。

5

第5章

测试工具选择和脚本开发

选择好工具、最流行的工具不一定适合自己，真正适合自己的工具才是最好的。也只有开发出数据驱动脚本或关键词驱动脚本，才能真正提高测试自动化的产出投入比。

完成了测试用例的设计后，接下来就是将其中可以实现自动化的那部分测试用例转换为测试脚本。虽然在某些情况下，测试人员会和开发人员一起合作，就所理解的程序直接开发脚本，然后从脚本中将注释转换为测试用例。在这种情况下，似乎是先有脚本，再有测试用例。但实际上并非如此，其实还是先有测试用例，再有测试脚本的，只不过是测试用例在心中，即先在思维中形成，然后融入测试脚本之中，在脚本注释中具体体现出来。虽然最后由脚本注释转换为测试用例，那只是文档形式的表现。如果事先没有测试用例，怎么能写测试脚本？测试自动化依旧依赖于测试流程和测试用例。没有好的测

试用例，测试工具会事倍功半。

在实现测试自动化的过程中，一般要先选定测试工具，然后根据测试工具的要求来开发相应的测试脚本。本章将介绍下列内容：

- 软件测试自动化的作用和内涵是什么？
- 测试工具有哪些分类？
- 如何选择适合自己的测试工具？
- 如何录制、开发测试脚本？
- 如何构建数据驱动、关键词驱动的脚本？

5.1 测试工具的需求分析

选择测试工具之前，肯定会问这样的问题——为什么要用测试工具？有了测试工具又能干什么？要回答这些问题，就要清楚测试对工具的需求。下面，就从3个方面来分析测试工具的需求，为如何选择测试工具打下基础。

- 讨论测试工具的优势，测试工具能帮助解决什么样的问题和困难。
- 测试工具的实现原理是什么，如何借助测试工具完成一项测试任务。
- 基于这样的原理，可以更清楚什么方面需要什么样的测试工具，不同的测试目标需要哪些不同的测试工具。

5.1.1 测试工具的优势

在测试过程中，测试工具是必需的，特别是在系统性能测试、负载测试（压力测试）和安全性测试等方面。在负载测试中，当你需要模拟大量数据（如每秒钟发送几十个数据包）或大量并发用户（如同时在线访问用户5000人）等各种应用场合时，如果不借助测试工具，几乎是不可能完成这种测试任务的。

其次，软件测试具有一定的重复性，主要体现在回归测试中。任何一个软件测试项目后期都要进行回归测试，可能是由于修正缺陷，也可能是由于软件不断升级而新加功能、增强功能。回归测试就是要验证已经实现的大部分功能，为了消除代码改动所造成的影响而要进行大量的测试。虽然回归测试找到软件缺陷的可能性小，效率比较低，但又是必要的、不可缺少的。如果由手工完成不断重复的回归测试，会在很大程度上降低测试工作的趣味性。如果由测试工具自动完成回归测试，将会解放测试人员，会是一件非常有意义的

事。

除此之外，手工测试还存在其他的限制，主要有：

（1）通过手工测试无法做到覆盖所有代码路径。

（2）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。

（3）在进行系统可靠性测试时，需要模拟系统运行十年或几十年的情景，以验证系统能否稳定运行，这也是手工测试无法模拟的。

（4）如果有大量（几千）的测试用例需要在短时间内（如 1 天）完成，又不可能投入大量人力，则无法采用手工测试。

（5）测试可以发现错误，但并不能表明程序的正确性。因为不论黑盒、白盒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。

由于手工测试的局限性，软件测试借助测试工具成为必要，并向全面自动化发展，以解决手工测试的局限性，带来最大收益。

（1）缩短软件开发测试周期。对上千个测试用例来说，测试工具可以在很短时间内完成测试，而且测试工具不知劳累，可 24 小时不停地运行同样的测试用例 10 遍、100 遍等。这些都说明了软件测试工具执行测试具有速度高、效率高的特点。

（2）脚本可以多次重复运行，降低成本。在回归测试中或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，测试工具可以多次运行同样的测试用例，而测试脚本只要开发一次。

（3）可增强测试的稳定性和可靠性，通过测试工具运行测试脚本，能保证百分之百被执行，所有的测试结果都能客观地记录下来。

（4）可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的，并且它能与所预期结果或规格说明书所规定的标准进行量化对比。

5.1.2 测试工具的实现原理

软件测试工具，就是一种特殊的程序，以模拟测试人员对计算机的操作过程和行为，包括数据请求和接收，或者类似于编译系统那样对计算机程序进行静态检查。测试工具应用很广，无处不在，可以根据其实现原理的不同分为白盒测试工具、黑盒测试工具。又可以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、测试管理工具等。单元测试工具多数属于白盒测试工具，而系统功能测试、负载测试和性能测试等一般属于黑盒测试工具，或者是一种介于两者之间的灰盒测试工具。

概念

（1）**白盒测试工具**是直接对代码进行分析，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、具体的某个变量。白盒测试工具根据是否运行代码，可进一步分为静态测试工具和动态测试工具。

- **使用静态测试工具**时，不需要对代码编译链接和生成可执行文件，不需要运行代码，可以在构建软件包、代码检入（Check-in）之前进行。
- **使用动态测试工具**时，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插入一些监测代码，掌握断点这一时刻程序运行的数据（对象属性、变量的值等）。

（2）**黑盒测试工具**采用测试过程的捕获和回放技术、脚本技术、虚拟用户模拟技术等模拟用户的操作，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。黑盒测试工具可以基于用户的需求，以及测试用例来完成测试，能发现更高层次的缺陷，如设计缺陷、与规格说明不一致等问题，这是白盒测试工具所不能解决的问题，白盒测试工具仅仅从编程语言自身的语法规则上来考察。

1．代码分析

代码分析类似于高级编译系统，一般是针对不同的高级语言去构造相应的分析工具。在工具中描述类/对象/函数/变量/运算等定义规则、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，再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，可在代码中设置一些“断点”，在这些断点和其他地方插入一些监测代码，存于构造的可执行文件中，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。首先将用户的每一步操作都将纪录下来，记录被操作的程序显示对象（窗口、按钮、滚动条等）或者实际物理位置，以及相对应的操作、状态变化、属性变化等。所有的记录转化为一种脚本语言所描述的过程，模拟用户的操作。

回放时，将脚本语言所描述的过程转化为屏幕上的操作，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，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。

在捕获和回放中，关键一点就是对象的识别，对浏览器页面、客户端窗口中各种对象的识别。对象的识别，就是获得这个对象的类别、名称、属性的值，如图 5-1 所示的文字编辑域“Agent Name”的各种属性值：

- 可编辑（Enabled）属性为真（True），可以修改该域的值。
- 焦点（Focused）属性为真（True），即当前操作点正落在这里。
- 高度（Height）属性的值为 20。
- 宽度（Width）属性的值为 119。
- 文字（Text）属性的值为“Harold”，也就是该文字编辑域的值。

常见的对象有弹出窗口、按钮、菜单、列表、编辑域等，如下面的脚本示例。

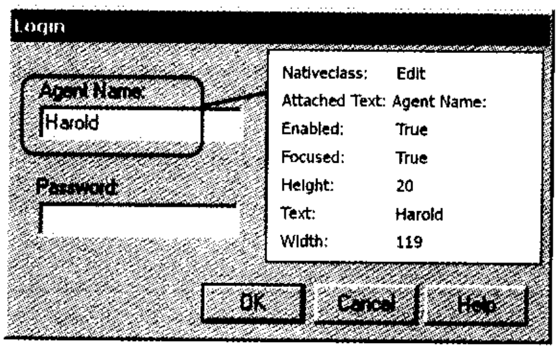


图 5-1 编辑域的对象识别示例

3. 脚本技术

脚本作为一种特殊的计算机程序，包含数据和指令。指令作为控制信息来操作软件中的对象，数据则主要是被操作对象属性的值。脚本技术，就是围绕着脚本程序结构而进行的设计，它可以实现测试用例所要求的输入、步骤和验证点，在创建脚本和维护脚本的两个不同成本中平衡，以获得测试自动化的最大收益。

脚本技术不仅能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，构造一定数量的虚拟用户同时进行操作，以给系统足够的负载，从而检验或获取系统的响应速度、数据吞吐能力等。

示例

```
Sub Main
    Dim Result As Integer

    Window SetContext, "Class=Shell_TrayWnd", ""
    PushButton Click, "Text=开始"

    Window SetContext, "Caption=「开始」菜单", ""
    ListView Click, "ObjectIndex=2;\;ItemText=控制面板(C)", "Coords=57,17"

    Window SetContext, "Caption=控制面板", ""
    ListView DblClick, "Text=FolderView;\;ItemText=显示", "Coords=37,30"

    Window SetContext, "Caption=显示 属性", ""
    TabControl Click, "ObjectIndex=1;\;ItemText=桌面", ""
    PushButton Click, "ObjectIndex=2"

    Window SetContext, "Class=#32770", ""
    GenericObject Click, "Class=Static;ClassIndex=1", "Coords=77,58"

    Window SetContext, "Caption=显示 属性", ""
    TabControl Click, "ObjectIndex=1;\;ItemText=屏幕保护程序", ""
    EditBox Click, "Label=等待(W):", "Coords=23,9"
    EditBox Left_Drag, "Label=等待(W):", "Coords=22,9,31,9"
    InputKeys "6"
    ComboBox Click, "ObjectIndex=1", "Coords=160,11"
    ComboListBox Click, "ObjectIndex=1", "Text=贝塞尔曲线"
    PushButton Click, "Text=设置(T)"

    Window SetContext, "Caption=贝塞尔曲线屏幕保护程序设置", ""
    SpinControl Click, "ObjectIndex=1", "Coords=8,5"
    SpinControl DblClick, "ObjectIndex=2", "Coords=9,7"
    ScrollBar HScrollTo, "ObjectIndex=1", "Position=13"
    PushButton Click, "Text=确定"

End Sub
```

4. 自动比较

自动测试时，预期输出是事先定义的，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。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。

概念

自动比较类型

（1）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。

（2）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匹配，而复杂比较是一种智能性比较，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，来排除与输出中预期会出现差异的部分进行比较以忽略特定的差异。

（3）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，都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，从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。

（4）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了，并执行了过滤任务之后，再进行比较。这样可以使比较标准化，测试结果可靠。

5.2 测试工具的选择

除了特殊应用的测试工具处，一般不建议自己开发测试工具，选用已有的、成熟的测试工具是一种比较明智的方法。选择测试工具，先是根据软件产品或项目的需要，确定要用哪一类的工具，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，也要从众多不同的产品中做出选择。

5.2.1 测试工具选择的标准

选择测试工具，需要针对测试的实际需求，了解适合这一需求的各种产品，并做进一步的调查。然后从中选出 3～4 种成熟的、优秀的工具软件作为候选工具，通过使用、评估和分析，来做最终的决定。在选择的过程中，关键是如何判断测试工具须具备哪些必要的功能特性，优势是否突出，是否真正适合自己等，也就是需要确定工具的选择标准。虽然购买商业化测试工具软件时，应具有沟通和商业谈判的能力和技巧，并认真考察其价格、服务等，但随着开源测试工具越来越多，我们有了更大的选择范围，不再局限于昂贵的商业化的软件。

测试工具选择的客观标准——特有的功能特性要求，是选择测试工具最关注的内容。但并不是说，测试功能越强大越好，因为解决问题是前提，适用才是根本。功能强大的工具，暂且不论其价格高低，往往多数功能没有用到，而且使用复杂，甚至不够稳定。为不需要的功能花钱是不明智的，同样，仅仅为了省几个钱，忽略了产品的关键功能或服务质量，也不能说是明智的行为。

根据多年积累的这方面经验，测试工具选择的客观标准主要有：

- 跨平台和环境的兼容性：测试工具应支持不同的运行平台，如各种操作系统和浏览器。这一点，对于互联网应用的测试更重要。如果开发的一套测试脚本，可以在 Windows/Mac/Solaris/Linux 或 IE/FireFox/Mozilla 上运行，这意味着自动化执行的测试用例翻了几倍，测试工具的产出明显。
- 易使用，例如界面简单、功能针对性强（而不是包罗万象）。
- 支持国际化版本和多种语言，这对软件国际化测试和本地化测试是非常重要的，也能保证测试工具有很高的覆盖面和产出。
- 支持脚本语言，可以构造类似编程语言的、简单的程序结构，如条件分支、循环结构等。并可以定义变量，实现参数的传递、多种组合数据的输入。这种脚本语言越接近熟悉的编程语言（如 C++、VB 等）越容易被接受。
- 脚本语言的简单性，可以推进测试自动化的普及。如脚本语言支持 HTML 或 XML 格式，就有可能使所有测试人员都投入到自动化测试中。
- 面向数据驱动（Data-Driven oriented）的脚本，包括支持流行的数据库（Oracle、SQL server、Access 等）、格式文件的数据存取操作。这有利于测试脚本的代码和数据输入分离，减少脚本维护的工作量，也有利于脚本的扩充和完善。
- 对程序界面中对象的识别能力。这种能力能够识别用户界面的各种对象，记录对象的 ID（而不是记录其位置的象素坐标），确保录制的测试脚本具有良好的可读性、修改的灵活性和维护的方便性。否则，只要程序界面稍做改变，在不同的屏幕分辨率下和不同的操作系统或测试环境下，原有的测试脚本就不能用了。
- 具有脚本开发良好的环境，能提供较强的脚本调试功能，而且支持脚本单步运行、设置断点等，可更有效地对测试脚本的执行进行跟踪、错误定位。
- 测试工具的集成能力包括和正在使用的开发工具、其他测试工具进行良好的集成，拥有良好的集成能力，可让它们互为受益。
- 图表功能。测试结果可以通过一些统计图表来表示，结果显示直观，一目了然，容易完成对测试结果的分析和解释。

脚本语言的功能性和简单性存在一个平衡的问题。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，例如类似于 C 语言。但越强大的脚本语言，就显得越复杂，掌握起来要更困难些。

具体问题具体对待，选择标准也不是一成不变的，它需要灵活的政策。由于客户端和服务端有不同的特点和测试需求，测试工具的选择标准也不同。如对 Web 客户端来说，由于 Web 的链接、快速变化的 UI 和复杂的逻辑，使得工具的录制功能更强、更稳定，而

且能适应不同的平台（Windows, Linux, Mac OS）和浏览器（IE, ForeFox, NS, ...）。而服务器一般不存在 UI 界面，主要是对不同协议的支持。

要点补充

测试工具的高级功能特性

（1）脚本语言是否支持外部函数库和函数的可重用性也是选择测试工具的标准之一。如果它支持函数调用，就可以建立一套比较通用的函数库，一旦被测试的软件做了修改，只要把原脚本中的相应函数进行更改即可，而不用进行大面积的脚本改动，可以省去大量工作。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、对数据库编程接口的调用等。举一个简单又实用的例子，在脚本中经常要执行“向数据库插入一条记录”的操作，这时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执行正常，这样就避免了人工检查，自动完成校验，提高了测试的自动化程度。

（2）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）抽象层，被称为测试映射图（Test Map）、测试帧（Test Frame）或取值映射图（Get Map）。抽象层介于被测应用程序和录制生成的测试脚本之间，它能解决针对程序界面直接进行脚本录制时所存在的问题。它将程序界面中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上元素的变化，这样可以减少测试脚本建立和维护的工作量。例如可以在脚本中将用户登录窗口的用户名和口令标识为逻辑名“NAME”和“PASSWORD”，映射为程序中对应的实际变量名，这样在很多脚本里就可以用相同的逻辑名去处理不同的用户登录操作。如果在下一版本的程序中，登录窗口的变量名发生了变化，只需要简单地将抽象层中这两个对象的映射关系进行修改，而不需要把所有脚本都修改一遍，脚本执行时能通过抽象层自动使用新的对象标识。如果工具软件支持程序界面对象自动搜索，并建立所发现对象的抽象层，当然就更好了。

（4）支持分布式测试，表现为可以定义测试任务或操作的依赖性和次序性，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。它对协同工作、通讯的软件应用模式（如网上会议系统、远程培训系统、聊天系统等）来说是必不可少的，目前在互联网上这种应用模式越来越多，它可以支持多用户、实时地共同操作同一个软件。例如，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。而且当两个测试任务要同时打开一个文件时，这类测试工具能保持协调或协同

处理，避免出现资源竞争问题。在测试执行中，这类测试工具自身传输的数据量很小，具有很强的独立性，这对被测软件的通讯影响很小。

5.2.2 测试工具评估报告实例

依据上面讨论的测试工具的评选标准对软件测试工具进行评估时，主要考察测试工具所具有的特性，特别是这类工具的关键特性。不同类型的测试工具，其关键特性是不同的：

- 功能测试工具对脚本功能、验证点、对象的识别等要比性能测试工具强。
- 性能测试工具也有其独特的功能特性，如：模拟虚拟用户、加载方式、分布式部署等。

对工具的评估，需要针对需要考察的特性论述各自的优缺点，甚至给出分数，最后提交报告。根据评估报告，特别是根据比较分析对照表，就比较容易做出决策。

表5-1是一个很好的例子（互联网上的一个实例，作者是Gary），稍做修改。从脚本录制、脚本调试、场景布置、监控设置、运行控制、结果获得等许多项目，来对两个著名的性能测试工具LoadRunner和Robot+TestManager进行比较。这个比较分析还是比较全面的，它可以考虑到技术支持、后续发展等因素，而且为每一项因素给出了分数。如果能为每一个项目给出加权因子，最后再给出加权后的总分，那就更好了，因为每个项目的重要性是不一样的。

这里主要借它来说明评估的方式或方法，这里对表5-1的结果正确与否不作结论。针对LoadRunner和OenSTA的更全面、更细致的比较分析请参考http://www.testingreflections.com/node/view/361。

表5-1 LoadRunner和Robot+TestManager的比较

比较内容		LoadRunner	得分	Robot+TestManager	得分
脚本录制	特点	按照不同的录制对象来确定录制脚本类型，不同的录制对象支持的方式不同	8	按照不同协议来录制VU脚本，不同的协议录制出的脚本不同	7
	优点	针对不同的录制对象时，支持的交互界面较好		可以录制不通过本机的其他机器之间的通讯内容	

续表

比较内容		LoadRunner	得 分	Robot+TestManager	得 分
脚本录制	缺点	只能录制通过本机的通讯内容		录制时人机交互界面支持不够好	
脚本调试	特点	脚本与其 WinRunner 脚本可以通用	8	VU 脚本与 GUI 脚本不同	6
	优点	易读性较强、调试方便		函数支持丰富、处理较灵活	
	缺点	函数支持相对弱一些		易读性差、调试难度大	
场景布置	特点	为用户内置了一些场景，可以较为方便的使用	7	用户可以灵活地定制测试场景，但是有些需要通过修改测试脚本来进行	8
	优点	图形化处理界面，操作简便		用户、场景、Suit 定义层次清楚	
	缺点	定制灵活性差一些		一般测试人员较难上手，需要较长时间熟悉	
监控设置	特点	可以监控多种资源情况	8	无	0
	优点	运行时获得的资源数据与脚本运行情况相匹配		无	
	缺点	采集数据含义不够明确和通用		无	
运行控制	特点	可以监控和设定不同虚拟用户的运行状态	8	针对脚本内容有全面的运行监控，用图形表示不同的运行状态	7
	优点	监控资源和虚拟用户同时进行		可细致地看到脚本运行情况，运行效率较高	
	缺点	运行脚本效率较差		无法与资源情况统一	

续表

比较内容		LoadRunner	得　分	Robot+TestManager	得　分
结果获得	特点	汇总信息采用单独工具，可以较为详细地对数据结果进行统计分析，可以自动生成报告	8	对脚本各种情况的监控支持较为全面	8
	优点	数据结果详细、图表丰富，可以灵活生成报表和报告		自动记录每一次的测试结果，包括每一步骤执行时间，以方便分析	
	缺点	结果不易保存，处理速度较慢		不能自动生成可读的测试报告，需要再加工	
操作性	特点	容易上手，适合于非技术人员操作	8	上手困难，需要有一定技术基础的人使用	6
界面友好性	特点	界面友好性较强	8	界面友好性较强，但是容易down掉，并且很难重启	5
技术支持性	特点	Mercury 技术人员较少，支持有困难	7	IBM技术人员较多，还有其他服务提供商给予技术支持	8
其他	特点	测试范围更广的专业工具	8	IBM 的产品重点不在 Robot 上，而是在新的 Eclipse 平台上	6
总分			78		61

5.2.3　测试工具选择的误区

不同的测试工具面向不同的测试目的，它们具有各自的特点和适用范围，所以不是任何一个优秀的测试工具都能适应不同公司的需求。某个公司怀着美好的愿望，花了不小的代价引入世界数一数二的软件测试工具。一年以后，这个测试工具却成了摆设。究其原因，就是没有充分考虑公司的现实情况，不切实际地期望通过测试工具来改变公司的现状，从而导致了失败。

测试工具毕竟是工具，不会改变一个公司的现状。影响软件开发效率和质量的关键因素是人，是人文环境（软件质量文化、流程），绝不是工具。测试工具在很大程度上依赖

于软件的可测试性、测试流程和软件产品/项目的自身特性，可测试性是在软件设计、编程中形成的，测试流程要根据测试自动化的需要进行改进，甚至变革。

过分强调测试工具的作用，极力追求各种软件测试工具，是软件测试本末倒置的表现。实际上，测试过程中 80%以上的缺陷是手工测试发现的，仅有不到 20%的缺陷是靠工具测试发现的，而且这还得要求测试人员合理地使用工具。

许多厂商在宣传自己的测试工具时，总是强调或放大工具的优势，而对工具所受到的限制、所存在的问题避而不谈，例如厂商在宣传时说测试工具只需通过简单的录制、回放，就可以完成测试。实际上，在绝大多数情况下，直接录制完的脚本往往并不能被顺利地回放，而且缺乏足够的验证点。测试绝不是演示，需要随着软件的修改对脚本不断维护，所以直接录制的脚本只能作为脚本开发的起点，还需要进行大量重构、修改脚本的工作。

选择测试工具，绝不能仅仅依据宣传和厂商自己事先准备好的演示来做出决定，需要亲自通过测试工具完成一个实际的软件测试任务，才能对工具的适用性做出正确的评估。

选择测试工具，不要求多，不要盲目地随大流，要有针对性，针对自己的软件产品，先解决实际测试工作中最突出的问题。如性能测试，必须依靠工具来实现。如果是特殊的客户端/服务器应用，可能需要自己开发这一类工具，不要奢求现成的测试工具。如果是 Web 应用，就比较容易找到合适的测试工具。有什么具体需求，就选择什么样的测试工具。

国内多数软件公司是针对最终用户进行项目开发而不是产品开发，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需求、用户界面变动都较大，这种情况下就不适合引入功能测试工具。因为对于变化大的需求和界面，录制或开发、修改脚本的工作量远远大于手工测试的工作量，又不能通过今后大量重复的回归测试收回成本，这种情况下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。这时，可以考虑引入白盒测试工具，特别是针对代码进行直接检查、分析的工具，用以提升代码质量。

5.3 测试工具完整方案之商业版

从功能性、系统性等方面进行比较，可以看出商业性软件测试工具还是强一些，这种软件测试工具有很好的基础，积累了很好的应用经验。这一章的重点放在功能测试工具、性能测试工具、Web 领域和其他的一些测试工具上，单元测试工具将放在第 6 章介绍。

在众多的测试工具厂商中，有几个著名的公司值得一提，它们就是 IBM-Rational、HP-Mercury Interactive（MI）、Compuware、Borland-Segue、Parasoft 和 Radview。特别是前面 3 家公司，它们都能提供一个较完整的测试自动化解决方案。这里就不一一详细介

绍了，读者可以通过访问它们各自的网站，获得全面的信息。

示例

1. Compuware（*www. Compuware.com*）

- 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。
- DBPartner：数据库开发及测试工具，有效地消除 Transact SQL 错误，快速而精确地编写 Transact SQL 应用。目前支持 Sybase、SQL Server、Oracle Suite。
- Error Detection：自动地运行期错误检测和诊断，检测那些由于逻辑错误引起的堆栈错误、内存溢出或资源泄露、API、OLE 错误等问题，并迅速地定位出错的源代码。
- Coverage Analysis：自动代码覆盖率分析，能够列出（手工）测试中每一次应用会话所有函数被调用的次数、所占比率等，并可以直接定位到源代码。
- Performance Analysis：自动性能分析和优化。能提供测试中每一次应用会话函数的调用时间，提供详细的应用程序和组件性能的分析，帮助快速查找性能瓶颈。
- SmartCheck：自动 VB 运行期错误检测和诊断，如隐藏的 run-time 错误、Windows API 函数错误调用、内存分配等错误。
- FailSafe：自动 VB 错误处理和恢复，通过插入额外的代码捕获，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，使开发人员能够快速且正确地解决问题。
- CodeReview：自动 VB 源代码分析，由 Metrics 和 Namer 组成，完成长度、复杂度、理解度的度量和自动并规则地对其中的对象重新命名，从而完成对应用程序的组件、逻辑等的源代码检查。

- JCheck：自动 Java 可视线程和事件分析，提供了一个生动的图形化的方法来表现程序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、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。

2. IBM-Rational（*www.rational.com*）

- Robot：对整个应用程序进行全面的功能测试。
- TestManager：管理所有测试活动和工件的核心平台。
- TestFactory：无需用户介入，自动查找运行时错误并生成优化脚本以用于回归测试。
- PureCoverage：提供代码覆盖分析，找出未经测试的程序代码。
- Quantify：快速识别应用程序性能瓶颈。
- Team Test：确实作好产品发布的准备。
- Purify：查找造成程序崩溃的那些难以发现的运行时错误。
- Suite TestStudio：测试各种工具套件产品，保证产品发布的质量。
- ClearQuest：集成的缺陷跟踪及变更请求管理。

3. HP- Mercury Interactive（*http://www.mercuryinteractive.com*）

- 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 应用能正确地工作。
- QuickTest：一个独创性的技术，能简化、加速功能测试。
- LoadRunner、Astra LoadTest：负载测试工作，预测系统的性能和表现。
- TOPAZ：性能监控和管理工具。

5.3.1 GUI 功能测试工具的比较

GUI 功能测试工具在软件测试自动化中占有重要的一席之地，它主要适合于回归测试阶段。比较流行的、专业的功能测试工具主要有 HP 公司的 WinRunner 和 QuickTest Pro（QTP）、Compuware 公司的 QARun 和 Test Partner、IBM 公司的 Rational Functional Tester 和 Robot、Borland 公司的 SilkTest 等。没有必要对这些工具做一一介绍，因为从互联网上都可以查到相关信息。这里只通过对 WinRunner、Robot 和 SilkTest 的比较（如表 5-2 所示）使读者领会 GUI 功能测试工具的重要特性是什么，以及要关注哪些内容，从而有助于这类工具的选择和应用。

表 5-2 WinRunner、Robot 和 SilkTest 的比较

	WinRunner	Robot	SilkTest
IDE 的支持	支持	多种	支持
记录与回放	支持	支持	支持
操作系统	Windows	Windows	Unix、Windows
浏览器	IE，FireFox	IE	IE
应用程序对象识别	支持	支持	支持
测试脚本	TSL	SQABasic （VB）	第四代面向对象的脚本语言 4Test，类似 C++
脚本调试功能	强大	支持，但不够强	类似 C++ Debugger
GUI 对象和脚本分离	是	不支持	对象声明在单独的 include 文件中，但是 case 脚本中也包含有
数据驱动	直接支持 Data Driver Wizard	间接支持	直接支持
自动数据生成	支持	不支持	支持
验证点	文本、位图和数值等	图像、文件、字符串、数字等	
允许通过数据库验证数据	通过 ODBC 及本地数据库连接	通过 ODBC 实现	自带 Database Tester
自动收集测试数据	是	是	是
图形化设计脚本执行集合	通过 Testdirector 设计	不支持	不支持
无人看管下的自动执行测试	通过 Testdirector	通过 DOS 批处理文件执行，或借助 TestManager	允许

续表

	WinRunner	Robot	SilkTest
分布式测试控制/同步/执行	借助 Testdirector 支持	借助 TestManager 支持	支持
非预期错误的恢复	允许	比较难	允许
自动创建测试结果日志	是	是	是
结果显示	直观	直观	
分析报告	详尽、易读		
toolbar/field 定制	可以	无法定制	无法定制 toolbar，可以定制 GUI 的 fields 内容
维护	重用性好，GUIMap 文件以保存应用对象，所以只要直接修改 GUImap	修改脚本	更新 include 文件
代码覆盖	不支持	借助 PureCoverage	不支持
性能测试	不支持，需用 Load Runner	结合 RationalQuantify	不支持
与测试管理工具接口	Testdirector	TestManager	自带 Case 管理功能

5.3.2 负载和性能测试工具

因为负载和性能测试是手工测试的弱项，是工具的强项，所以有不少优秀的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，主要有 MI 公司的 LoadRunner，Compuware 的 QALoad，Rational 的 SQA Load、Performance、Visual Qualitify。这里给出 3 个商业性能测试工具的比较分析，如表 5-3 所示，可加强对这类工具的认识。

表5-3 几个商业性能测试工具的比较

属 性	LoadRunner	QALoad	WebLoad
出品公司	HP-MI	Compuware	Radview
价格	昂贵	较贵	较贵
安装配置的复杂性	简单	简单	方便
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应用
支持平台	Windows、Unix或Linux	HP-UX、IBM AIX、Sun Solaris、Linux、NT/2k	Unix Windows
支持数据库	DB2、SQLserver、Orcale、Sybase	ADO、DB2、Oracle、Sybase、SQLserver、Odbc	ADO、DB2、Oracle、Sybase、SQLserver、Odbc
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VC++	Javascript
模拟/虚拟用户	是	是	是
自动数据生成	是	是	是
脚本调试	是	是	是
报表定制功能	是	是	是
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，同时运行太多影响结果的准确性

示例

LoadRunner

可以记录下客户端的操作，并以脚本的方式保存，然后在一台或几台 PC 机上模拟上百或上千虚拟用户同时操作的情景，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。

- 使用引擎 Virtual User Generator 能够自动生成虚拟用户，模拟真实用户的业务操作行为。通过记录下业务流程、转化为测试脚本，可以对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。
- Controller 能组织起多用户的测试方案，Rendezvous 功能则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。同时，使用 AutoLoad 技术，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，以及频率、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，在 Virtual users 运行时，利用 ContentCheck 检测应用程序的网络数据包内容，从而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，以便迅速查找到性能问题并追溯原由。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、网络和服务器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。

5.3.3 基于Web应用的测试工具

现在 Web 应用非常广泛，其相应的测试工具也比较多，无论是开源的、还是商业化的。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。

- Web 功能测试工具有 Parasoft 的 WebKing、MI 公司的 Astra QuickTest、RSW 公司的 E-Test Suite 等；
- Web 系统测试工具有 WorkBench、微软的 Web Application Stress Tool（WAS）和前面介绍的 WebLoad 等。
- Web 站点质量分析工具，如 Compuware 公司的 WebCheck，能自动扫描 Web 站点，发现多达 50 类的各种 Web 问题。
- 全面的 Web 测试工具 SOAPtest，能从 WSDL 文档中自动创建测试套件，用于服务功能的测试套件同样适用于负载测试。使用 XSLT 到 SOAP 消息实现客户端和服务器端正确的事务处理，彻底了解 SOAP 协议和 Web 服务方面的问题。

下面以工具 WebKing 为例，来了解 Web 测试工具的特点。WebKing 是一个独特的工具，可帮助开发人员防止和检测多层次 Web 应用中的错误，它具有下列技术和特点。

（1）执行白盒测试用例时，为表格对象自动生成一组用户输入并显示生成的目录和动态页，分析网站的结构从而找到测试的最佳途径。WebKing 能自动测试每一个静态和动态网页，检查所有的链接，以发现其中的构造错误，并借助其 CodeWizard 功能执行最重要的 HTML、CSS 和 JavaScript 编码标准检查。

（2）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，包括一组缺省的路径。它可以建立自己的测试用例以测试特定的路径。

（3）独有的 RuleWizard 特性——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，但 RuleWizard 能标识出不变的部分。CodeWizard 自动执行这些规则，以保证动态网页能够符合设计要求。

（4）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。每当修改网站时，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。网站发布时，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。

（5）WebKing 的特性：

- 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的统计信息。
- 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。

5.4 测试工具完整方案之开源版

开源软件运动已获得很大成功，正在改变软件业的开发思想、模式和方法等，也自然改变着软件测试的方法及其应用。借助开源软件测试工具，完全可以构造一个完整的自动化测试解决方案，在快速提高测试效率的同时，又能降低购买测试工具和获取相应售后服务的成本。

从单元测试、功能测试到性能测试，从 Web 页面测试到数据库、多媒体、通讯等应用领域的测试，直至软件测试的管理平台、缺陷跟踪系统，都基本可以获得合适的测试工具。可以说，目前较成熟的开源测试工具可以覆盖整个测试的工作领域。

5.4.1 开源测试工具总览

开源测试工具涵盖各个方面，先在这里进行整体的介绍，包括单元测试工具和测试管理工具，然后在相继的两个小节重点介绍功能测试工具 Selenium 和性能测试工具 Jmeter，而在第 6 章会较详细介绍 JUnit 的应用。

1. 测试模型

该模型是一个软件测试启发式模型，基本目标是能提醒测试人员在创建测试时应着重考虑的各种因素，进而来定制测试。该模型试图解决开源软件测试中的一系列问题，如下所示。

- 协商并理解项目的测试目标。
- 理解并协商与测试技术选择相关的各种因素，理解与测试工作有关的限制、要求和可用资源，从而使得测试更为高效。
- 在充分考虑和利用其他各种因素的前提下选择合适的测试技术以达到测试目标。
- 随时监控项目的状态，并在需要时调整测试计划，使得目标、测试技术的选择和各种因素保持统一。

2. 单元测试工具

- JUint，使用广泛的 Java 语言单元测试工具，详细内容见本书 6.5.1 节或 *http://www.junit.org/index.htm*。
- Mockrunner，是在 J2EE 环境中的单元测试工具，支持 Struts、Servlets、过滤器和标签类。
- 针对 C/C++语言的单元测试工具 Cgreen、Check、CppTest.
- 针对 C#语言的主要单元测试工具有 NUnit、NUnitForms。
- 针对 PHP、SQL 语言的单元测试工具有 PHPUnit、SQLUnit。
- 其他单元测试工具：HttpUnit、XMLUnit、DBUnit、ObjcUnit 和 SIPUnit 等。

3. 功能测试工具

- Selenium——强大的 Web 功能测试工具，由 Selenium IDE、Core 和 Selenium Remote Control 等 3 部分组成，可以和 Ant、EMMA 结合起来使用。
- MaxQ——功能测试工具，包括一个 HTTP 代理工具。它可以录制和回放测试脚本，并具有很强的统计图表功能，可用来处理测试结果。详见 *http://maxq.tigris.org/*。
- Abbot——Java 客户端功能测试工具，可基于 XML 的脚本或者 Java 代码实施用户界面测试的脚本。详细参考 *http://abbot.sourceforge.net/doc/overview.shtml*。
- 其他开源功能测试工具，如 WebInject、页面链接测试 Link Sleuth，更多请参考：*http://www.opensourcetesting.org/functional.php*。

4. 性能测试工具

- Jmeter——针对静态资源和动态资源（Servlets、Perl 脚本、Java 对象、数据查询和 FTP 服务等）的性能测试工具。Jmeter 可结合 Badboy 来使用，它能自动录制测试脚本。
- OpenSTA，可以模拟大量的虚拟用户来完成性能测试，可精确度量负载测试的结果，包括响应时间、资源使用情况、数据库服务器的使用情况。详见

http://portal.opensta.org/index.php。

- DBMonster 是一个生成随机数据、用来测试 SQL 数据库的压力测试工具，详见 *http://dbmonster.kernelpanic.pl/*。
- LoadSim——网络应用程序的负载模拟器。
- 更多的性能测试工具，可以访问 *http://www.opensourcetesting.org/performance.php*。

5. 数据库测试工具

- DBProbe。
- OSDL Database Test Suite，是基于 Linux 开发的测试框架的数据库测试工具套件，具有很好的实用价值，详见：*http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/*。
- 其他的数据库测试工具，如 OraRep、phpMyAdmin。更多的数据库测试工具见：*http://dbcommander.sourceforge.net/*。

6. 多媒体和网络测试工具

多媒体测试工具主要进行语音（VoIP）、视频（Vedio）和 IP 电话等应用系统的特殊测试，如多媒体数据交换、服务质量（QoS）等，主要有 AuthTool、SIPp、 Seagull、Sofia、Asterisk 和 X-Lite 等等。

通讯测试（包括多媒体通讯）一般离不开网络监控工具，网络工具很多，这里介绍几个比较流行、简单而强大的网络监控工具。

- Nessus——主要的 Unix/Linux 受攻击性的评估工具。
- Wireshark——前身是 Ethereal，网络协议检测程序，包括获取网络数据包的流向及其内容、监控 TCP session 动态，支持 Unix、Windows。
- Snort ——跨平台的网络信息探测工具，可以实施实时通讯分析和信息包记录、数据包有效性检查、协议分析和内容查询匹配，并能探测缓冲溢出、秘密端口扫描、CGI 攻击、操作系统侵入尝试等，从而进行实时报警。
- Netcat ——作为类似于 telent 的客户端，可以监听某个端口、扫描特定主机的端口，传输文件等，被称为网络中的瑞士军刀。

5.4.2 Web 功能测试工具——Selenium

Selenium 是 ThoughtWorks 专门为 Web 应用开发的自动化测试工具，适合进行功能

测试、验收测试，其最大的优势有几点：

- 可直接运行在浏览器之上，所见即所得，因为 Selenium 的核心是用 JavaScript 编写的。
- 支持多操作系统（Windows、Mac OS 和 Linux）和各种浏览器 Internet Explorer、Mozilla 和 Firefox。
- 支持两种开发脚本的模式 test runner （HTML 文件）和 driven（脚本语言编写），其语言包括 Java、.NET、Perl、Python 和 Ruby。使用 driven 脚本，有一部分测试在浏览器之外运行，而如果使用 test runner 脚本的话，测试完全在浏览器内运行。

1．组成

- Selenium IDE（集成开发环境）。它是一个 firefox 的插件（plug-in），可以录制、回放并编辑测试脚本。
- Selenium Core（核心）。它是符合断言（assertion）机制，由测试套件执行的平台。它是整个测试机制的核心部分，由纯 JavaScript 代码组成。
- Selenium Core HTA。它是针对 IE 设计的执行平台，运行于 IE 的最高安全级别之上。
- Selenium Remote Control（远程控制）：它是一个代理与控制端，可代替 Selenium core/ Selenium IDE 的客户端，并支持多语言。

详细比较见表 5-4。

表 5-4 Selenium 几个独立测试组件的比较

比较的内容	Selenium IDE	Selenium Remote Control	Selenium Core	Selenium Core HTA
功能	录制、编辑、回放			
多语言支持	仅 Selenese（Selenium 简单的脚本语言，没有条件、循环结构）	很多，包括 Java、.NET、Pertl、Python 和 Ruby	仅 Selenese	仅 Selenese
支持HTTPS/SSL	是	是，0.9 及其以上版本	是	是
浏览器支持	仅 Firefox	直接支持 Firefox 和 IE，借助手动配置以支持更多的浏览器	所有	仅 IE

续表

比较的内容	Selenium IDE	Selenium Remote Control	Selenium Core	Selenium Core HTA
需要 Java	否	是，运行代理服务器	否	否
需要远程安装	否	否，通过代理服务器欺骗浏览器，使 JavaScript 看似来自相同的远程服务器，从而符合同源策略	是，因为存在阻止进行跨网站脚本访问的安全策略，它需要安装在 AUT 的网站内	否
支持跨域的应用程序/服务器	是	是	否	是
将测试结果保存到磁盘	是，因为是 FireFox 扩展	是，因为支持多语言	否，只支持 JavaScript。但可以将测试结果发送到另外一台服务器上保存	是

2. Selenium 命令

Selenium 命令分成两类，即操作（action）和断言（assertion）。

- 操作命令模拟用户与 Web 应用程序的交互。例如，单击一个按钮和填写一个表单，这些都是常见的用户操作。
- 断言命令是验证一个操作的预期结果，如验证页面内容或当前位置是否正确。

在 Selenium 网站上可以找到可用命令的完整列表。通过 Selenium 命令，可以描述在浏览器中执行的操作。

3. Test runner 脚本开发模式

Test runner 脚本，就是用 HTML 语言中简单的表格格式来编写的**测试用例**，所以 Test runner 脚本不仅易于阅读，也易于编写，如表 5-5 所示。

表 5-5 Selenium 自动化脚本（浏览器中显示的表格方式）

命令（Command）	对象（Target）	值（Value）	注释
open	/config/login_verify2?.src=yc&.intl=cn&.partner=&.done=http%3a//cn.calendar.yahoo.com/?		访问雅虎日历站点：*http://cn.calendar.yahoo.com/*
Type	username	test1	输入用户名 "test1"
Type	passwd	1234567	输入密码 "1234567"
clickAndWait	//input[@value='登录']		点击 "登录" 按钮
verifyTextPresent	"登出，我的账户"		验证用户登录成功

- 第一列包含**命令**或**断言**。
- 第二列包含命令或断言的**目标**（target），通常使用组件的 ID 或名称，也支持 XPath 和 DOM 的定位符。
- 第三列为**目标的属性值**或命令或断言指定参数的**值**。例如，当使用 type 命令时，这一列可能就是一个文本域所期望的值。
- 无论 Test Suite 还是 Test Case，表格的第一行都是描述性文字，Selenium 的引擎是不会处理这一行的内容的，实际的脚本都是从第二行开始的。

表 5-5 用直观的表格方式显示脚本，其实际脚本的源代码是 HTML 格式，如下面"示例"所描述的。可以看到，登录雅虎日历所做的操作已经在 Selenium 里面被记录成对应的命令，而手工测试的验证点（确认用户登录成功）已经在 Selenium 里面转换成判断页面上是否存在"登出，我的账户"这样的文字。

示例

Selenium 自动化脚本（HTML 格式）

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>login</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">login</td></tr>
```

```
</thead><tbody>
<tr>
    <td>open</td>
      <td>/config/login_verify2?.src=yc&.intl=cn&.partner=
         &.done=http%3a//cn.calendar.yahoo.com/?</td>
      <td></td>
</tr>
<tr>
      <td>type</td>
      <td>username</td>
      <td> test1</td>
</tr>
<tr>
      <td>type</td>
      <td>passwd</td>
      <td>xxxxxx</td>
</tr>
<tr>
      <td>clickAndWait</td>
      <td>//input[@value='登录']</td>
      <td></td>
</tr>
<tr>
      <td>verifyTextPresent</td>
      <td>登出，我的账户</td>
      <td></td>
</tr>
</tbody></table>
</body>
</html>
```

4. Selenium 脚本的开发

Selenium 的脚本开发也是比较灵活的，不仅提供了几百个命令，而且可以在脚本中引用其他文件（类似于 C 语言的头文件），如表 5-6 第 3 行 Include 的使用；还可以使用变量，如表 5-6 中的${Site_URL}、${userName}和${password}，这可以解决测试输入数据问题。例如用户名和口令就不需要放在脚本中，而是单独存入一个文件中，如表 5-7 所示。

表 5-6 Selenium 测试脚本（引用文件和变量）

Command/Assertion	Target	Value
include	../../ common/SetVariable.html	
open	${Site_URL}	
pause	2000	
selectWindow	Header	
waitForTextPresent	Logon	

续表

Command/Assertion	Target	Value
click	//a[contains(@href, 'javascript:Logon();')]	
selectWindow	mainFrame	
waitForTextPresent	userName	
type	userName	${userName}
type	Password	${password}
clickAndWait	//input[@name='Submit']	
verifyTextPresent	Logon Success	

表 5-7 common/SetVariable.html 的内容

设置变量的值（Set variable）		
storeGlobal	cn.calendar.yahoo.com	siteURL
echo	${siteURL}	
storeGlobal	testuser	userName
storeGlobal	1234567	password

5. Driven 脚本开发模式

Driven Selenium 脚本支持多种编程语言，它可在浏览器之外的一个单独的进程中运行。Driven 脚本比 test runner 脚本更强大、更灵活，可以将它们与 xUnit 框架集成，但是 Driven 脚本编写和部署相对复杂，需要经过下列过程：

（1）启动服务器，并部署被测试的应用程序（AUT）。

（2）部署测试脚本。

（3）启动浏览器，发送命令到 browser bot。

（4）验证 browser bot 执行的命令的结果。

Driven 脚本更依赖于应用程序运行时的环境。例如，Java 驱动程序使用一个嵌入式 Jetty 或 Tomcat 实例来部署所测试的应用程序。Browser bot 就是 Selenium Core，负责执行从测试脚本接收到的命令，而驱动程序与 Browser bot 之间的通信使用一种简单的特定于 Selenium 的连接语言 Selenese。

6. 测试用例和测试套件

测试用例开发涉及 4 类文件，除了引擎库以外，其他 3 类文件都是可以根据具体情况去修改的。

（1）主文件：TestRunner.html/TestRunner.hta（.hta 文件是 html application、Windows 平台特有的）。

（2）Test suite 和 Test case 文件，通过以表格为基础的 html 文件来实现。

（3）引擎库 js 文件：它是位于 Selenium 根目录下的核心文件，其中 html-xpath 目录下的那个文件是必须的库文件。

（4）user-extensions.js：它是用来扩展 Selenium 的文件，用户自己编写的函数和扩展的命令都应该放在这个文件中。

测试套件和测试用例一样，都是用简单的 HTML 表编写的。测试套件用于将具有类似功能的一些测试用例编成一组，以便能按顺序执行一系列的测试用例。Selenium 执行的缺省测试套件的名称是 TestSuite.html。测试套件使用一个只包含一列的表，表中的每一行指向一个包含某个测试用例的文件。对于一个有着多个功能模块和组件的 Web 应用，编写的测试脚本 html 必然比较多。因此，应该建立一个合理的目录结构来组织这些脚本。

Selenium 部署完毕后，可以打开浏览器通过 url 来访问 TestRunner.html 文件。由 TestRunner.html 调用相应目录下的测试套件的 HTML 格式文件——TestSuite.html，也就是 TestSuite.html 所在的完整路径及其自身成为 TestRunner.html 的一个参数。再由 TestSuite.html 调用相应的测试用例（测试脚本），即 TestSuite.html 列出的一系列测试脚本的 html 文件。可以直接修改 TestSuite.html 文件，让其指向自己开发的 Test case html 文件。浏览器 URL 中运行的格式就是：

http://localhost/selenium/TestRunner.html?test=/testDir/myTestSuite.html。

而 MyTestSuite.html 被设计成定义全局变量的 setVariable1.html 和两个测试用例的文件 login.html 和 logout.html，如表 5-8 所示。

表 5-8 测试套件的 HTML 文件示例

选 择	名 称	对应的测试用例脚本文件
☑	SetVariable	../../ common/SetVariable.html
☑	login	../module/login/login.html
☑	logout	../module/login/logout.html

5.4.3 强大的性能测试工具——JMeter

JMeter 最早是为完成 Tomcat 的前身 Jserv 的性能测试而诞生的，随着 J2EE 应用地不断发展，Jmeter 自身也得到了很好地发展，其功能不再局限于 Web 服务器的性能测试，还涵盖了数据库、FTP、LDAP 服务器等各种性能测试，以及与 JUnit、Ant 等工具的集成应用。

在 Jmeter 安装之前，需要先安装 JDK 1.4 或更高版本、Ant 1.6.5 或更高版本。安装 Jmeter 非常简单，直接解压即可，然后运行其 bin 目录下的批处理文件 jmeter.bat。

1. JMeter 主要构成组件

- 测试计划（Test Plan）作为 JMeter 测试元件的容器，是使用 JMeter 进行测试的起点。
- 线程组（Thread Group）代表一定数量的并发用户，用来模拟并发用户发送请求。实际的请求内容是在采样器（Sampler）中定义的。
- 逻辑控制器（Logic Controller）可以自定义 JMeter 发送请求的行为逻辑，它与 Sampler 结合使用可以模拟复杂的请求序列。
- 采样器（Sampler）定义包括 FTP、HTTP、SOAP、LDAP、TCP、JUnit、Java 等各类请求。如 http 请求默认值负责记录请求的默认值，如服务器、协议、端口等。
- 配置单元（Config Element）维护采样器需要的配置信息，并根据实际的需要修改请求的内容。配置单元包括登录配置单元、简单配置单元、FTP/HTTP 配置单元等。
- 定时器（Timer）负责定义请求之间的延迟间隔。
- 断言（Assertions）可以用来判断请求响应的结果是否如用户所期望的。它可以用来隔离问题域，即在确保功能正确的前提下执行压力测试。这个限制对有效的测试是非常有用的。
- 监听器（Listener）负责收集测试结果，并可以设置所需的、特定的结果显示方式。
- 前置处理器（Pre Processors）和后置处理器（Post Processors）负责在生成请求之前和之后完成工作。前置处理器常常用来修改请求的设置，后置处理器则常常用来处理响应的数据。

2. 如何使用 Jmeter 进行性能测试

使用 Jmeter 进行性能测试，其操作很简单。例如，以 Web 服务器的性能测试为例，

按下面 5 个步骤进行操作就可以了，如图 5-2 所示。

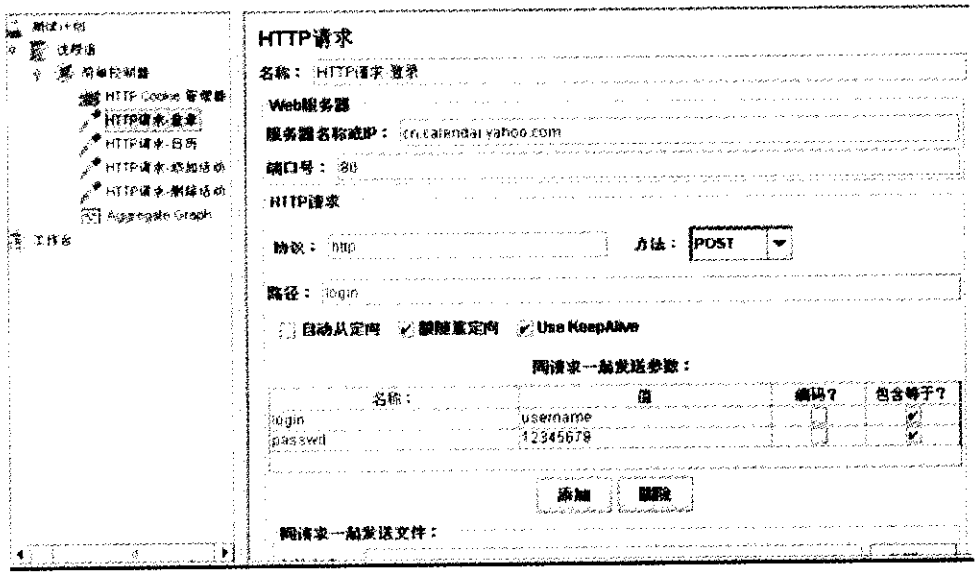


图 5-2 Jmeter 界面和 http 请求设置

（1）在 Jmeter 里增加一个线程组、一个简单控制器、一个 Cookie 管理器、一个综合图形器（Aggregate Graph） 和若干个 HTTP 请求。

（2）在线程组中定义线程数、线程产生的时间和测试循环次数。

（3）在 http 请求中定义服务器、端口、协议和方法，以及请求路径等。

（4）配置用户登录信息，进行安全设置，例如完成“Http URL 重写修饰符”或“Http Cookie 管理器”的有关配置。

（5）表格监听器负责收集和显示结果。

技巧

（1）当 Web 应用程序的参数通过 POST 方式发送给服务器的时候，在浏览器的地址栏里是看不到这些参数的，所以也就无法在 JMeter 里填写这些参数。这时，可以通过一些工具如 HttpWatch 来抓取送给服务器的所有 URL，然后根据这些 URL 来填写 Jmeter 里参数。

（2）为了避免偶然因素影响，可以将线程组里循环次数设置为 50，这样做可以使同样的测试将会被执行 50 次，然后取其平均值作为测试结果。在配置完所有的参数后，保存脚本。然后使用工具 DUSuperControl 将网络速度设置为 80～120 Kbp。

（3）Session 状态的跟踪方式

- 使用 Cookie 保留和传递 sessionid。要求浏览器允许保存 Cookie，通过 Jmeter 的 HTTP Cookie 管理器可以实现。
- 使用 URL 重写，每次显式浏览器和服务器之间传递的 sessionid。它要求程序对 URL 进行编码，但对浏览器没有要求。可使用 JMeter 前置管理器中的 http url 重写修饰符来完成，但要将浏览器的 Cookie 功能关闭。

示例

例子及其所用的文件

- AuthorizenFilter.java：过滤器负责检验 session 中是否存在用户信息。如果没有，那么就转向到 login.jsp。
- User.java：用户类负责记录用户的信息。
- login.jsp 负责生成 User 对象，并调用 User.java。当 login 返回为 true 时转向到 welcome.jsp。
- web.xml 用于配置 filter 拦截所有访问 JSP 页面的请求。
- 建立下列测试单元，如两个 http 请求和 Cookie 管理器。Cookie 管理器负责管理整个测试过程中使用的 Cookie，不需要设置任何属性。
  - 第一个 HTTP 请求，请求 login.jsp，并附加验证所需要的参数（user=，pwd=，Submit=）。其响应的断言验证 url 中包含“welcome.jsp”。
  - 第二个 HTTP 请求，请求的是 welcome.jsp，其响应的断言验证应包含用户名（参数 user 的值，成为 welcome.jsp 页面逻辑的一部分）。
  - 循环控制器设置发送第二个请求的循环次数，表格监听器负责收集和显示第二个请求的测试结果。

执行的顺序是：

- 发送第一个请求“登录页”进行登录。
- 成功登录之后，使用循环控制器执行第二个请求。请求 welcome.jsp 时，断言以验证是否确实是 welocme.jsp 来处理请求，而不是其他页面。

这里采用了 HTTP Cookie 管理器，它使得第二个请求顺利地发送到 welcome.jsp 进行处理，而不会因为缺少用户安全信息转发到 login.jsp。

3. 使用 Jmeter 完成数据库和 JMS 服务器等的性能测试

（1）数据库服务器。JMeter 使用 JDBC 连接配置（如数据库 url、驱动类名、用户名和密码等）并发送请求，实施对数据库的测试。在实际的项目中，至少有两种类型的 JDBC 请求需要关注：select 语句和存储过程。

（2）JMS 服务器。在测试之前需要将 MOM 软件包（如 ActiveMQ）复制到 %JMETER_HOME%/lib 下。JMeter 在测试时使用了 JNDI，即要将 jndi.properties 放到 JMeter 的 classpath 中，建议将 jndi.properties 与 bin 下的 ApacheJMeter.jar 打包在一起。测试时，分两种情况处理。

- 通过设置发布者（JMS Publisher）和订阅者（JMS Subscriber）的属性完成。
- 对于点对点，通过采样器 JMS Point-to-Point 来完成。

示例

jndi.properties 的示例

```
java.naming.factory.initial =
org.activemq.jndi.ActiveMQInitialContextFactory
java.naming.provider.url = tcp://localhost:61616
connectionFactoryNames = connectionFactory
#注册 queue，格式：#queue.[jndiName] = [physicalName]
queue.MyQueue = example.MyQueue
#注册 topic，格式：# topic.[jndiName] = [physicalName]
topic.MyTopic = example.MyTopic
```

5.5 测试脚本录制和回放

测试脚本的开发非常依赖于测试工具或测试脚本的开发语言，如 VBscript、SQABasic、JavaScript、Perl、Python 和 Ruby 等。由于篇幅有限，这里就不做讨论。我们可以换个角度——从测试脚本录制和回放来讨论脚本的开发，在“脚本录制和回放”上多数工具的差异较小。实际在脚本录制和回放前，就像软件编程之前一样要创建一个项目——自动化测试项目，以便对脚本及其存取权限等进行有效地管理和维护。

5.5.1 创建自动化脚本项目

创建自动化项目相对比较简单，与日常软件项目管理比较相近，通常需要经历下列

过程：

（1）创建 TA 测试项目。

（2）创建测试计划（Test Plan）和测试用例文件夹（Test Case Folder）。

（3）创建测试用例。

（4）灵活构建测试套件（Test Suite）。

（5）环境设置。

这里以 IBM-Rational 的 adminstrator、TestManager 和 Robot 为例，展示这样一个创建的过程。

1. 创建 TA 测试项目

在 adminstrator 创建一个 TA 项目，其操作简单，要设置名称、文件存储路径、连接的数据库等，如图 5-3 所示。而且要为这个项目增加所有能存取的用户，不同的用户可能有不同的权限（如读、修改和管理的权限等）。

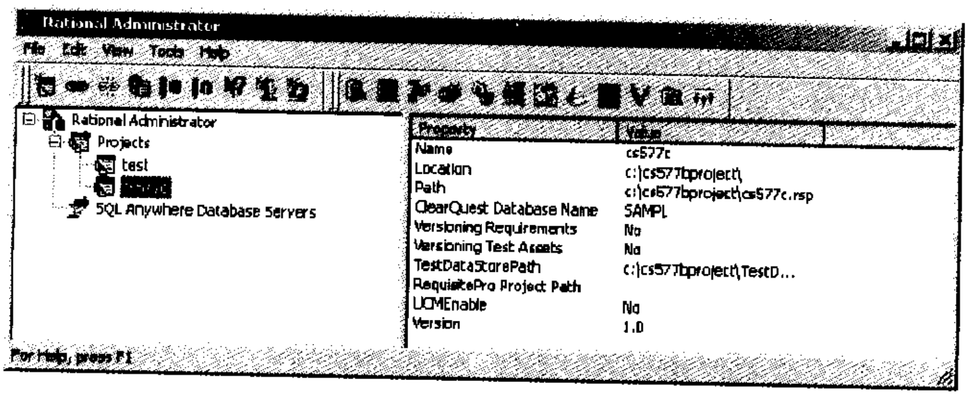


图 5-3 TA 项目创建的界面

2. 创建测试计划和测试套件

先创建测试计划，包括设置计划名称、管理者/拥有者（Owner）、迭代阶段/方式、可用的测试机器配置等。其迭代阶段来自统一过程（RUP），有 4 种：

- 初始阶段，也称先启阶段（inception phase）。
- 细化阶段，也称精化阶段（elaboration phase）。
- 构造阶段（construction phase）。
- 交付阶段，也称转化阶段、产品化阶段（transition phase）。

在测试计划基础之上，可以创建测试用例文件夹。并把它视作测试用例的模块管理，不同的模块放在不同的文件夹中，如图 5-4 所示。文件夹可以像在 Windows 操作系统中一样，形成多层次的文件夹。

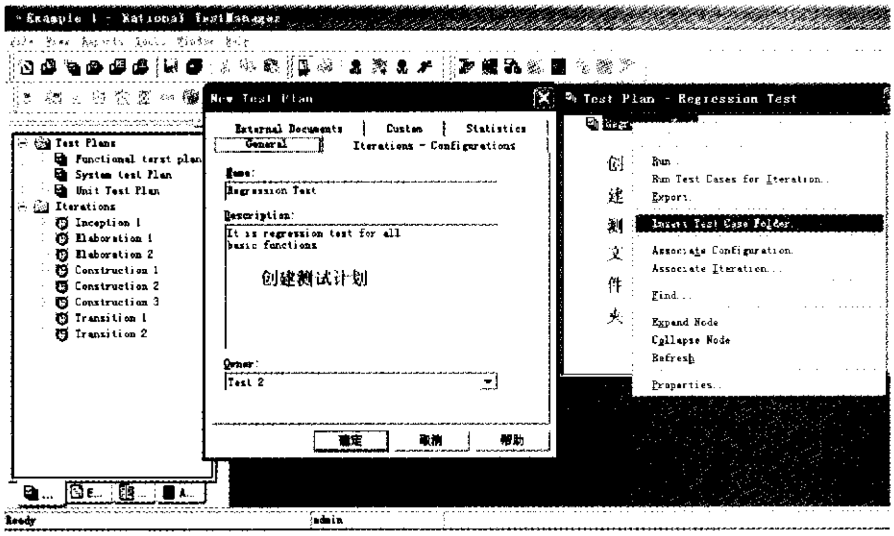


图 5-4　测试计划和测试用例文件夹的创建

3. 创建测试用例

在测试文件夹中，可以插入新的测试用例。新的测试用例可以直接编写，也可以从文本文件或其他格式的文件中导入，如图 5-5 所示。

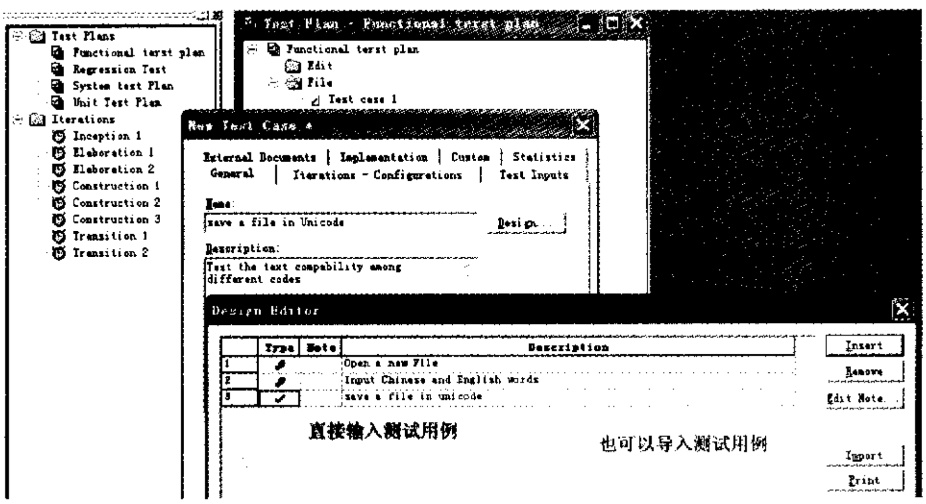


图 5-5　测试用例创建和导入

4. 灵活构建测试套件

录制并修改完所有的脚本后，可以将完成某个测试任务的脚本组织在一个 Test Suit 里。例如，把要完成首轮“测试登录”的所有测试脚本都组织在 Login Suit 里，把和聊天有关的脚本放在 Chat Suit 里等，如图 5-6 所示。这样，在测试时可以选择性地运行部分脚本，在进行局部测试时不必运行全部的脚本，这增加了脚本在测试中的灵活性，同时减少每个 Test Suit 里所包含脚本的数量，可以提高脚本运行时的稳定性。

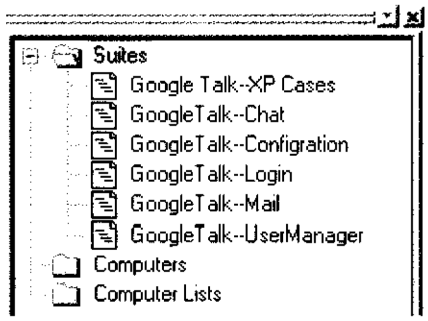


图 5-6 测试套件的创建和组织

5. 设置测试用的客户端

许多时候要完成不同环境的客户端测试任务，如 Google Talk 支持多个平台（Windows、Mac OS、Linux 等），所以需要设置相应的客户端。而且，像即时消息、网上会议等通讯或协同工作软件需要多个模拟用户共同参与测试，也就是需要多台机器并行运行一套测试用例。测试机器的设置如图 5-7 所示。

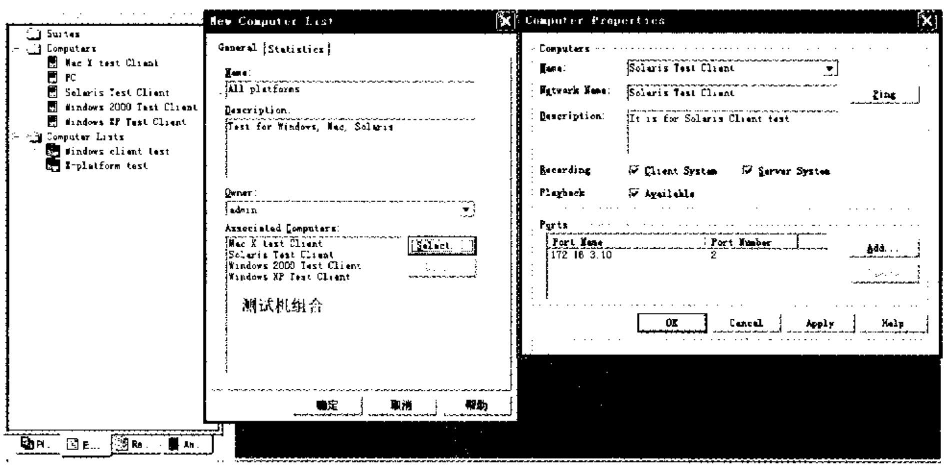


图 5-7 测试机器的设置

5.5.2 录制 Selenium 脚本

首先，下载最新的 Selenium IDE（以 xpi 为扩展名的插件，如 selenium-ide-0.9.xpi）到本地目录（下载路径：*http://www.openqa.org/selenium-ide/download.action*），然后，可以直接拖动这个文件到浏览器 Firefox 窗口中，或者在 Firefox 中打开链接后安装。在 Firefox 中安装好 Selenium 的 IDE 后，就可以开始录制测试脚本了。

打开 Firefox 里面的 Selenium 工具，打开 Firefox 的时候默认 Selenium 的记录功能是自动启动的，然后在“Base URL”输入雅虎门历网站并进行操作，此时在页面上的命令就会被记录下来，如图 5-8 所示。

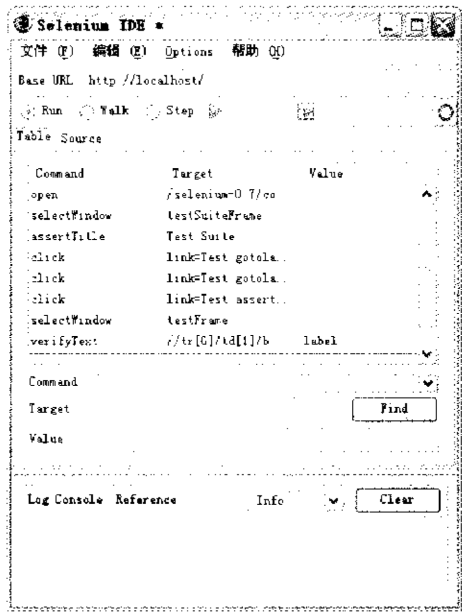


图 5-8 Selenium IDE 的录制脚本的界面

录制好的测试脚本以 HTML 文件格式保存下来，但录制完脚本后要验证测试回放是否成功，以及脚本的验证点（Verification Point，VP）是否符合测试用例要求，这需要通过执行脚本、人工审查脚本来得到确认。

录制一个 Selenium 测试脚本的基本步骤，可以简单描述如下：

（1）启动 Selenium IDE，设置待测试 Web 站点的 Base URL。

（2）确信录制按钮是激活状态。

（3）使用 Firefox 访问被测试的 Web 站点，如 *cn.calendar.yahoo.com*。

（4）按照实际操作完成要做的动作。

（5）停止录制。

（6）回放脚本、调试脚本，如加入 pause 命令及其参数（2000 或 3000）。

（7）保存录制脚本。

示例

示例 1：活动增加的录制脚本

测试脚本：增加活动

clickAndWait	//button[@type='button']	
type	charCountTitleTA	周日羽毛球活动
verifyValue	charCountTitleTA	charCountTitleTA
clickAndWait	SaveBottom	
type	search_words	周日羽毛球活动
clickAndWait	SubmitSearch	
verifyTextPresent	周日羽毛球活动	

示例 2：修改活动名称的脚本

这个测试脚本是修改一个已经添加的活动的名称，然后验证是否成功修改内容。其实现方法是先去查询一个活动，找到后打开这个活动进行修改。修改后，再去查找这个活动，验证其名称是否被修改了。

测试脚本：修改活动

open	/YYY,96b93b/srt,0/?v=26&pv=61	
clickAndWait	link=效率手册	
type	search_words	周日羽毛球活动
clickAndWait	SubmitSearch	
clickAndWait	link=周日羽毛球活动	
type	charCountTitleTA	周日羽毛球活动－修改
clickAndWait	UpdateBottom	
type	search_words	修改
clickAndWait	SubmitSearch	
verifyTextPresent	周日羽毛球活动－修改	

要点补充

脚本录制的要点说明

（1）为了确保自动化脚本的稳定性和准确性，首先要对手工测试用例进行分解和转换，因为手工测试用例不一定完全适合自动化测试。例如，一些测试用例步骤过多、验证点复杂，所以需要进一步细化，使其适合测试脚本的要求。

（2）录制脚本时，不要快速地切换页面，必须等到页面的各种元素被充分的加载，这样 Selenium IDE 才可以完整地完成录制的动作。一旦录制完所有的动作，按下“Recorder”按钮就可以停止录制。

（3）回放脚本，主要验证脚本是否可以顺利通过。如果打开脚本后，立即尝试回放，Selenium 将会报错。这个错误是因为 Selenium 里面的 base URL 没有设置，Selenium 无法访问录制的 Web 的 URL，下面的图 5-9 给出了怎样设置 Selenium 的 base URL。

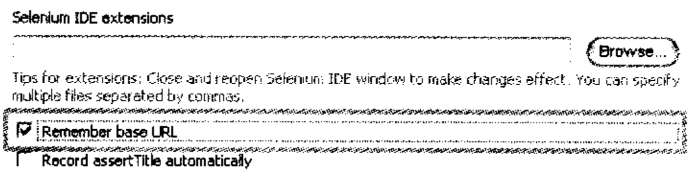


图 5-9

（4）设置断点（Breakpoints）。复杂的脚本含有很多命令，有许多需要注意的细节。一旦脚本出错，可以通过添加断点来快速定位脚本出错的位置。当脚本运行到断点的时候将自动停止，需要手动去点击“回放 play”按钮来继续执行脚本。怎样设置一个断点？有下面两种方法：

➢ 右击菜单选择“Toggle Breakpoint”。

➢ 按下键盘的“B”去设置这个断点。

一旦脚本运行到断点处，可以使用 Selenium IED 菜单工具栏中的“pause/resume”按钮来单步调试脚本。如果想去掉断点，同样操作——右击菜单选择“Toggle Breakpoint”即可，如图 5-10 所示。

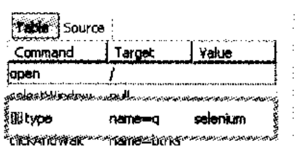


图 5-10

5.5.3 用 Robot 录制客户端测试的脚本

使用 Robot 录制脚本，可分为两种方式，一是为功能测试（GUI），二是为性能测试（VU）。这里，主要介绍功能测试的脚本，点击 GUI 录制图标，然后输入名称和设定选型，如图 5-11 所示。

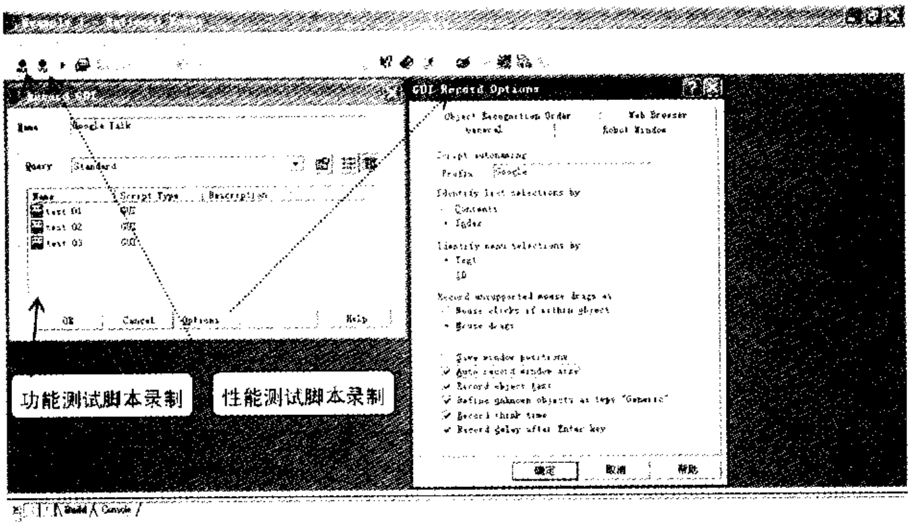


图 5-11 Robot 录制启动的界面

所有手工操作的步骤都可以录制下来，这时，在录制的脚本中没有验证点，需要在脚本中插入验证点，如表 5-9 中的 3 个用例。

表 5-9 录制的脚本经修改后的对比

直接录制下来的脚本（缺乏验证点）	经修改后并增加验证点后的脚本
'用例一：设 google talk 窗口为当前窗口，然后插入验证点，检查登录按钮的状态 Window SetContext, "Caption=Google Talk;Class=Google Talk-Google Xmpp Clie", ""	'用例一 '修改脚本：首先启动 google talk StartApplication "C:\Program Files\Google\Google Talk\googletalk.exe" ' 设 google talk 窗口为当前窗口 Window SetContext, "Caption=Google Talk;Class=Google Talk - Google Xmpp Clie", "" '插入验证点——检查登录按钮状态（禁用）

续表

直接录制下来的脚本（缺乏验证点）	经修改后并增加验证点后的脚本
	Window SetTestContext, "Caption=Google Talk;Class=Google Talk-Google Xmpp Clie", "" Result = PushButtonVP (CompareProperties, "Text=Sign In", "VP=Button Before Enter;Wait = 2,30") Window ResetTestContext, "", ""
'用例二: 输入用户名和密码, 检查登录按钮的状态 ComboEditBox Click, "ObjectIndex=1", "Coords=13,5" InputKeys "UserName" EditBox Click, "ObjectIndex=1", "Coords=54,4" InputKeys "Password"	'用例二: 输入用户名和密码 ComboEditBox Click, "ObjectIndex=1", "Coords=13,5" InputKeys "UserName" EditBox Click, "ObjectIndex=1", "Coords=54,4" InputKeys "Password" '验证点——检查登录按钮状态（可用） Window SetTestContext, "Caption=Google Talk;Class=Google Talk - Google Xmpp Clie", "" Result = PushButtonVP (CompareProperties, "Text=Sign In", "VP=Button After Enter;Wait=2,30") Window ResetTestContext, "", ""
'用例三: 清空用户名和密码, 检查登录按钮的状态 ComboEditBox Left_Drag, "ObjectIndex=1", "Coords=61,7,-68,7" InputKeys "{BKSP}" EditBox Left_Drag, "ObjectIndex=1", "Coords=40,6,-32,9" InputKeys "{BKSP}"	'用例三: 清空用户名和密码 ComboEditBox Left_Drag, "ObjectIndex=1", "Coords=61,7,-68,7" InputKeys "{BKSP}" EditBox Left_Drag, "ObjectIndex=1", "Coords=40,6,-32,9" InputKeys "{BKSP}" '插入验证点——检查登录按钮状态（禁用）

续表

直接录制下来的脚本（缺乏验证点）	经修改后并增加验证点后的脚本
	Window SetTestContext, "Caption=Google Talk;Class=Google Talk - Google Xmpp Clie", "" Result = PushButtonVP (CompareProperties, "Text=Sign In", "VP=Button After Enter;Wait=2,30") Window ResetTestContext, "", "" '修改脚本：插入关闭 google talk 的脚本 Window CloseWin,"Caption=Google Talk; Class=Google Talk - Google Xmpp Clie", ""

5.6 测试脚本的开发

测试脚本的开发，经常需要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和管理，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。

测试脚本也是程序，所以应该遵守已有的、规范的编程标准和规则。用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字与开发中其他环节一样，应进行统一规划，按规定进行配对，以便把自动化测试的结果进行对比分析。

只要是程序，就可能存在缺陷，所以测试脚本或测试程序也需要进行测试，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。脚本的测试简单、容易些，一般是将脚本测试和脚本执行结合起来，也就是通过大量的单元测试——脚本执行来完成。一旦使用脚本进行测试时发现了问题，要么是被测试的对象有问题，要么是测试脚本有问题，这样，脚本问题容易被发现出来。

5.6.1 适应测试脚本开发的测试用例

在编写自动化测试脚本前，首先需要写好自动化测试用例。因为测试工具不会想象，

没有发散思维，所以自动化测试用例不同于手工测试用例，自动化测试用例的描述需要精确，即每条自动化测试用例的逻辑不要太复杂，验证点不宜繁多，应该清晰地描述操作步骤和验证点的信息。从而造就清晰、简短的与测试用例对应的脚本，避免在脚本编写中出现较多的错误，更有助于今后的脚本维护。例如，表 5-10 通过与 Google Talk 登录按钮状态的特定测试用例的对比，向我们展示了自动化测试用例的特点。

表 5-10 自动化测试用例和手工测试用例的对比

手工测试用例	自动化测试用例
标题：登录按钮状态 步骤： 启动 Google Talk; 保持用户名和密码框为空，登录按钮状态应为不可用；当用户名和密码框都不为空时，登录按钮状态为可用。	用例 1: 标题：用户名密码编辑框为空时，登录按钮状态为禁用（1）。 步骤： 启动 Google Talk; 验证点：登录按钮状态为禁用。
	用例 2: 标题：清空用户名密码后，登录按钮状态为禁用（2）。 步骤： 在用户名和密码框中输入字符; 然后清空用户名和密码的内容; 验证点：登录按钮状态为禁用。
	用例 3: 标题：用户名密码编辑框不为空时，登录按钮状态为可用。 步骤： 在用户名和密码框中输入字符; 验证点：登录按钮状态为可用。

经验

自动化测试用例需要写得比较详细，每条测试用例应完整地描述所有操作步骤以及所有验证点。不要使用较为含糊的描述法，如“显示正常”，应清楚地说明需要验证哪几个关键点的内容。

需要根据项目的特点，选择合适的脚本开发方式。对于小型或一次性的项目，只需使用录制/回放的方式录完所有的脚本，然后修改并优化，增加上足够的验证点即可。

而对于会不断升级的大型项目，则需要依据产品的功能特点，订制一套和产品相配套的函数库，将常用的一些操作过程都封装在库函数中。比如登录功能是其他模块测试都调

用的一个功能，所以可将登录过程的脚本封装在一个函数中，如 LoginSite (UserName, PassWord)。在写其他模块的脚本时，只需调用该函数并传入用户名和密码参数即可，不用将登录过程重写一遍。这样，可以有效地提高脚本的开发效率，并能大大降低今后脚本维护的工作量。

5.6.2 测试脚本的重构和优化

如何降低成本，可以相对增加产出或者说更快地收回成本？关键是提高脚本开发速度，提高脚本运行的稳定性和降低维护脚本的工作量，主要方法有：

- 选择较好的、更适合的测试工具，其脚本语言简单、具有丰富的插件等。
- 尽量将脚本写成结构化的数据驱动或关键字驱动的脚本。
- 多录制脚本，然后优化脚本。不是所有的模块都可以变为数据驱动方式，这时就要抽象出脚本的单元，进行层次化组合，形成高效的结构化脚本。
- 测试和脚本开发合二为一，效率更明显。
- 选择适宜进行自动化的软件模块或功能特性。

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、移动和输入数据等，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。线性脚本适合于那些简单的测试（如 Web 页面测试）及一次性测试，多数用于脚本的初始化或演示等。

（1）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和灵活性，所以结构化脚本易于维护。

（2）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。可以将线性脚本转化为共享脚本。

（3）数据驱动脚本，将测试输入存储在独立的（数据）文件或数据库中，而不是存储在脚本中，使脚本的指令和数据分离。这样，脚本适应性好，维护成本低。

（4）关键字驱动脚本，是数据驱动脚本的逻辑扩张。在数据和指令之间再增加一层——关键字，用关键字来描述软件业务逻辑中的最小单元，业务逻辑的变化不会影响其脚本的底层——最小单元，而是反映在上层，从而使业务逻辑的变化对脚本影响最小。

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。表 5-11 显示了手工、线性脚本、共享脚本、数据驱动脚本和关键字驱动脚本等不同形式的测试带来的成本和效益，通过比较，我们可以清楚地知道数据驱动脚本和关键字驱动效率最高。

表 5-11 不同结构的测试脚本的效率

结 构	成 本	收 益	净收益
手工测试 （基准）	1	1	0
线性脚本（录制和回放）	8.3	11	2.7
数据驱动脚本	8.4	18	9.6
框架结构	9.8	15	5.2
基于应用逻辑的框架结构和数据驱动脚本的混合（关键字驱动）	11.6	19	7.4

5.6.3 测试脚本开发的框架

以一个开源软件框架——STAF(Software Testing Automation Framework)为例来说明，这个框架作为测试的中间件，它能建立测试自动化框架的底层通讯，并使不同的测试工具进程之间交换数据。STAF 可以和软件自动化框架支持（SAFS，Software Automation Framework Support）及自动化框架执行引擎（STAX, STAf eXecution engine）结合起来使用。

作为评估报告，需要从各个方面来论述各自的优缺点，并给出一些例子说明。这里由于篇幅所限，只给出一个比较分析对照表 5-12。根据比较分析对照表，就比较容易做出决策。

表 5-12 工具评估分析示意表

考察的项目	说 明	只有 STAF	STAF + STAX	STAF + SAFS
支持脚本	支持脚本，扩展、维护容易		√	√
控制中心	有对多台测试机的管理和控制能力，启动分布式的测试客户机	√	√	√

续表

考察的项目	说　明	只有 STAF	STAF + STAX	STAF + SAFS
监控水平	监控测试运行过程，及时了解各个测试任务的运行状况，并能根据需要进行调整、容错（跳过错误）和详细的日志（Log）等	√	√	√
自动化程度	测试任务的调度、执行、监控和测试结果报告		√	√
功能	是否能实现功能测试、性能测试、压力测试、黑盒测试等各种测试形式，是否能进行白盒测试自动化		√	√
方法	是黑盒测试方法，还是白盒测试方法		√	√ √
易用性	框架是否容易使用	有一定难度	有一定难度	较易使用
扩展能力	是否容易扩展	√	√	√
兼容性	现有的工具是否需要要经过很大改动才能纳入到框架中去	改动不大	改动不大	改动不大
集成性	框架是否是与应用无关的，能否融合各种已有工具和一些著名的商业工具			√
适应范围	框架是否能进行各种级别的自动化测试：轻量级 、企业级		适合轻量级	适合企业级
测试用例的管理	框架是否具有Case脚本的管理功能		基本 Case 管理功能	良好的 Case 管理功能
脚本的简单性	脚本是否简单，测试人员是否需要很强的编程能力才能掌握		较复杂	简单

续表

考察的项目	说 明	只有 STAF	STAF + STAX	STAF + SAFS
可重用性	测试脚本是否可重用，是否能跨平台运行		可以	可以
效率	性能问题	较好	较好	
邮件	E-mail 通知	√	√	
软件包构建	和 DailyBuild 系统集成			√

5.7 小结

由于手工测试具有一定的局限性，如手工测试无法模拟系统运行几十年，无法模拟大量用户等，所以需要引入软件测试自动化。软件测试自动化主要是通过开发软件测试工具、脚本（Script）等来实现的，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，应结合自己的实际应用特点来做出决定。最流行的工具不一定适合自己，真正适合自己的工具才是最好的。

在开源工具越来越多，其功能越来越强的时候，开源工具可能是更明智、更经济的选择。本章先给出了一个完整的开源工具解决方案，涵盖了测试的各个领域，并用了较大篇幅介绍了开源 Web 功能测试工具 Selenium 和性能测试工具 Jmeter。

另外，本章对脚本的录制、修改、开发和重构也做了全面介绍，包括录制脚本的步骤和技巧，以及如何在已录制的脚本中增加验证点，如何在线性脚本基础上构造共享脚本、结构化脚本、数据驱动脚本和关键字驱动脚本，以达到最高的自动化测试的产出投入比。

第6章

测试和编程的交互

单元测试的概念，实质是强调一个基本的质量思想和软件开发的理念——如果要保证一个系统的质量，首先要保证构成这个系统的所有组成单元的质量。

编程和单元测试伴随在软件测试用例设计和脚本开发过程中。虽然单元测试主要由开发人员自己来执行，但单元测试完全可以列入测试范畴之中，并占据着重要的位置。随着测试驱动开发思想不断深入人心、极限编程的运用越来越多，单元测试的地位也今非昔比，在软件开发的质量保证过程中越来越重要。

从另一方面讲，源程序对测试用例的设计和脚本的开发有较大的影响，包括代码的规范性、一致性和可测性等。为了提高脚本运行的稳定性，以及测试用例的覆盖面，测试人员需要关注编程的过程，同时应和开发人

员充分沟通、并行工作，创造出一种测试和编程的和谐环境。

在编程过程中每时每刻都要进行单元测试，所以说，这个过程是编程和测试交互的过程。在这交互的过程中，开发人员是主角，但测试人员也不是旁观者，他有许多工作要做，除了继续进行测试用例的设计和测试脚本的开发外，直接与编程相关的主要工作还有：

- 参与代码的检查、走查等。
- 参与单元测试、集成测试。
- 进行 Ad-hoc 测试，熟悉产品，完善测试用例。

6.1 程序代码的审查

程序代码的审查（Code Review）是一种静态测试，其效率是非常高的。据有关数据统计，代码中的缺陷60%以上可以通过代码审查（包括互查、走查、会议评审等形式）发现出来。代码审查，不仅能有效地发现缺陷，更重要是为预防缺陷获取各种经验，为保证代码质量打下坚实的基础。虽然代码审查要占用较多的时间，但从其作用看，它是不容忽视的。

软件的覆盖率测试和结构测试属静态分析的范畴，可以依靠一些软件工具进行。但是，代码审查必须依靠具有软件开发和编程经验的技术人员集体进行。在实际工作中，这个重要的环节往往被忽视，常常是没有经过静态分析和代码审查，就直接进入了单元测试。为什么在这里要花较大的篇幅讨论“代码审查”？就是希望能够引起大家对“代码审查”的足够重视，减少单元测试和后继的集成测试、功能测试，从根本上提高代码的质量。

6.1.1 代码审查的方法和范围

通常合格的代码应具备正确性、清晰性、规范性、一致性和高效性等，即：

- 正确性是指代码逻辑必须正确，能够实现预期的功能。
- 清晰性是指代码必须简明、易懂，注释准确没有歧义。
- 规范性是指代码必须符合企业或部门所定义的共同规范，包括命名规则、代码风格等。
- 一致性是指代码必须在命名上（如相同功能的变量尽量采用相同的标示符）、风格上都保持统一。
- 高效性是指代码不但要满足以上性质，而且需要尽可能地降低代码的执行时间。

代码审查的目的就是为了产生合格的代码，检查源程序编码是否符合详细设计的编码

规定，确保编码与设计的一致性和可追踪性。检查的方面主要包括：书写格式、子程序或函数的入口和出口、参数传递、存储器的使用、逻辑表达式的正确性和代码结构合理性等。其次，软件编程中所存在的一些共同点是可以规范和控制的，如语句的完整性、注释的明确性、数据定义的准确性、嵌套的次数限制、特定语句的限制等。概括起来，代码审查的工作涵盖下列方面（一个相对完整的代码审查列表，请参考附录 E）。

- 业务逻辑的审查。主要审查是否是按照实现设计的规格说明书展开的编程，逻辑是否正确且简单，思路是否清晰？
- 算法的效率。无论是内存处理还是统计排序算法，或者 SQL 查询语句，都需要精心设计其算法，确定最优化的方法。
- 代码风格。代码的命名规则、注释行、嵌套的次数、书写格式等直接影响了软件的可读性、可维护性和可靠性等方面的质量。
- 编程规则。包括语句的完整性、数据定义的准确性、常量和变量的定义、函数的调用、参数的使用、内存管理、逻辑表达式等，这些规则保证程序的运行正确性、准确性、性能、稳定性、可扩充性等。

代码审查的方法很多，如互查（Peer-to-Peer）、走查（Walk-Through）、会议评审（Inspection）等。互查，就是处在相同模块或相近模块的编程人员互相检查对方的代码，方法简单有效，容易开展。对于新人编写的程序，一般需要采用走查的方式，从头到尾将写好的程序检查一遍。而会议评审就更为正式，主要是审查关键性的代码，会议评审和走查的对比如表 6-1 所示。

表 6-1　走查与会议评审的对比

事　项	走　查	会议评审
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

对于所有关键性的代码，编程人员需要结合系统架构设计图和程序框图等来解释代码的实现方法和思路，回答大家提出的各种问题，有助于验证设计和实现之间保持一致。会议评审代码，要注意时间的控制和避免现场修改代码。时间控制是为了避免跑题，不要针对某个技术问题进行无休止的讨论。发现问题时，也不要现场修改，而是记录下来，会后再进行修改。否则，会浪费大家的时间，而且参加这类活动的人会越来越少。

在IBM、微软等很多公司都有一个很好的实践方法，那就是公开展示性的代码审查。这种代码审查的过程，不是将代码发给某一个人或某几个人去看，而是强调程序员自己定期走上台，向其他人讲解自己的源程序。因为要向大家讲解自己的程序，将自己的代码展现给大家看，程序员会极其重视自己的工作进度、代码质量，不希望由于代码写得很差而出现难堪的局面。所以，在写代码时，他就时刻想着——可能随时会被选中去做代码展示，所以会非常认真地对待每一行代码。这种代码审查方式还有其他一些好处：

- 互相学习程序设计思想、方法和技巧，共同提高。
- 及时发现代码的问题，包括不同代码模块、函数之间相互依赖、冲突的关联问题。
- 有更多的人明白他人写的代码，今后代码的维护也变得容易。

6.1.2 代码风格的审查

优良的代码风格可以加快软件开发进度，提高软件的可维护性和可移植性，能帮助测试人员更早地发现缺陷。代码风格是影响软件质量的重要因素之一，代码风格的审查有助于代码质量的提高。但就代码风格来说，审查是次要的，更重要的是推行“保持优良的代码风格”的理念和与之对应的规则。简单地说，就是要培养程序员的一种良好的编程习惯，代码风格的审查就是及时发现一些问题，帮助程序员遵守规则，养成好的习惯。

代码风格的审查，一般以新人写的程序为审查的主要对象，所有新人的代码都要进行严格的代码风格的审查，直至通过为止。而对资深程序员的代码，则以抽查为主，会随机选择一些模块。尽量使每个程序员都有机会被抽查到。

审查代码风格，首先会审查编程习惯问题，包括格式、变量定义和引用、注释行等。下面这个简单的例子就很能说明这个问题。

示例

示例1：

```
If(1==j)    // 好的代码风格
If(j==1)    // 一般的代码风格
```

在上面的例子中，如果 1==j 由于粗心写成了 1=j，在编译的时候，编译器就能给出错误信息。而对于后者，如果 j==1 被错写成了 j=1，编译器会认为是合法代码，这样一个逻辑错误就隐藏在程序中了。我们清楚地知道，修正一个逻辑错误的代价，远远大于一个编译错误。

示例 2：

```
If(j>MAX_NUM)  // 好的风格
If(j>2000)     // 差的代码风格
```

前者的可读性和可维护性都要好于后者。对于前者，如果最大的数量发生变化，只需要修改宏定义，而后者要在代码中把所有引用到 2000 的地方都要进行修改。

1. 命名规则

在代码风格中，很重要的一点就是命名规则，如著名的匈牙利命名规则和 GNU 命名规则。匈牙利命名表示法能够帮助我们及早发现并避免程序中的错误。由于变量名既描述了变量的作用，又描述了变量类型，所以比较容易避免产生类型不匹配的错误。但是在实际使用中，可以结合软件企业自身的情况，包括软件产品的单一性或复杂性，对“匈牙利”命名规则进行适当的改进。在保证可读性等的前提下，适当简化是必要的，简单易用，程序员乐意接受。例如：

- 类名和函数名由以大写字母开头的单词组合而成，变量和参数由以小写字母开头的单词组合而成，而常量全用大写的字母，并用下划线分割单词，如 `int MAX_LENGTH`。
- 静态变量加前缀 `s_`（表示 static），全局变量加前缀 `g_`（表示 global），类的数据成员加前缀 `m_`（表示 member）。
- 变量的名字应当使用“名词”或者“形容词＋名词”。全局函数的名字应当使用“动词”或者“动词＋名词”（动宾词组），如 `DrawBox()`。类的成员函数只使用“动词”，被省略掉的名词就是对象本身，`box->Draw()`。
- 为了防止某一软件库中的一些标识符和其他软件库中的产生冲突，可以为各种标识符加上能反映软件性质的前缀。例如三维图形标准 OpenGL 的所有库函数均以 `gl` 开头，所有常量（或宏定义）均以 `GL` 开头。

概念

匈牙利命名规则是纪念传奇性的 Microsoft 程序员 Charles Simonyi 给变量取名字的一种方式。非常简单，变数名以一个或者多个小写字母开始，这些字母表示变量的类型。这

样，看看变量的名字就知道变量的类型了。例如，iNumber 代表“是储存数字的整型变量”。在 cName 和 cSex 中的 c 表示“数据类型”，后面跟着的是变量的含义，一般用大写表示，如果是两个或两个以上的单词（或拼音）组成，例如 cStudentName，每个单词首字母大写，用以区分，方便阅读。

GNU 命名规则：文档名+主版本号+辅版本号+补丁编号。其命名规则：以所有字母都小写的主文档名称作为前缀，后跟一个破折号，再跟一个版本号、扩展说明以及其他后缀，如 Foobar-1.2.3.src.tar.gz 和 foobar.1.2.3.bin.ELF.tar.gz 等。

2. 函数处理

除了用标识符命名，函数处理中的有些规则也是编程风格的一部分。我们知道，函数接口的两个要素是参数和返回值，而在 C 语言中，函数的参数和返回值的传递方式有两种：“值传递”和“指针传递”。C++语言中多了“引用传递”。在审查代码风格时，检查函数参数、返回值和接口部分是否遵守一些常用规则，例如：

- 不要省略返回值的类型。如果函数没有返回值，则在函数前加 void。
- 注意参数的顺序，如 void StringCopy(char *strDestination, char *strSource)。
- 避免函数参数太多，尽量不要使用类型和数目不确定的参数。
- 如果参数是指针且仅作输入用，则应在类型前加 const，以防止该指针在函数体内被意外修改。例如：void StringCopy(char *strDestination, const char *strSource);
- 如果输入参数以值传递的方式传递对象，则宜改用“const &”方式来传递，这样可以省去临时对象的构造和析构过程，从而提高程序处理的效率。
- 不要将正常值和错误标志混在一起返回。正常值用输出参数获得，而错误标志用 return 语句返回。

3. 注释行

一般来讲，一个完整的软件程序应有明确的注释，注释主要包括：序言性注释、功能性注释、数据注释、软件模块注释等。注释的比例一般应大于源程序总行数的 1/5。

- 序言性注释，一般应置于每个源程序模块的顶部，它会给出整体说明，包括该模块的目标、主要功能、作者、审查者、修改记录和相应的日期等。
- 功能性注释包括功能实现的思路、方法和一些关键算法。
- 模块注释则集中在模块之间的关系、接口描述、输入和输出等。

- 数据注释用于说明软件单元中用到的变量定义，包括类、对象的名称、属性和成员函数等。

6.1.3 编程规则的审查

有了好的编程风格之后，即使是新手，程序看上去也像“老手”的作品，但是好的代码风格本身并不能覆盖团队中的整体编程技术方面的缺陷。为了使程序的质量和创造代码的效率有根本上的提高，应更多依赖于编程规则。

一方面，编程规则依赖于特定的语言，C++和 Java 都有自己特定的编程规则；另一方面，编程规则依赖于编程的思想，而向对象编程强调对类的封装和继承、函数重载等的处理，结构化编程强调程序模块的高内聚性和低耦合性等。

C++/Java 编程规则很多，应参考相关书籍。这里通过几个简单的 Java 最基本的语句和 C++重载函数、内存分配、指针引用等例子来说明如何针对编程规则进行代码的审查，以提高代码质量的作用。

1. Java 最基本语句的使用

在 Java 编程中，`try…catch`、`Throws` 和 `Finally` 所代表的基本语句是经常使用的，出现的问题也比较多，是 Java 编程审查的重点之一。

（1）`Try` 可以用来指定一段预防所有异常的程序。紧跟在 `Try` 程序后面，应包含一个 `Catch` 子句来指定想要捕捉的异常类型。在编程过程中，当同一段程序可能产生不止一种异常情况时，可以放置多个 `Catch` 子句。其中每一种异常类型都将被检查，第一个与之匹配的就会被执行。如果一个类和其子类都有的话，应把子类放在前面，否则程序将永远不会执行到子类。当一个 `Try` 语句不能包含所有出现的异常情况时，可以使用嵌套 `Try` 来实现。代码的审查目的就是确保所有的异常情况都被处理（抓住，catch）。

（2）`Throws` 用来标明一个成员函数可能抛出的各种异常。对大多数 `Exception` 子类来说，`Java` 编译器会强迫声明一个成员函数中抛出的异常类型。如果异常的类型是 `Error` 或 `RuntimeException`，这个规则则不起作用，因为这种异常类型不应该存在于正常的程序中。如果程序中想明确地抛出一个 `RuntimeException`，必须在 `Throws` 语句中声明其类型。`Throw` 语句要抛出一个异常，应事先得到一个可抛出实例（Throwable Instance）的控制句柄，并通过参数传到 `Catch` 子句，或者用 `new` 操作符来创建一个。

（3）线程一旦启动，就相当于一个主程序。它的 `run` 方法中如果出现异常或错误，将不会有其他主程序来捕获它。一般在写程序时，会捕获线程中的异常，不捕获将通不过

编译，但要记住同时需捕获 Error 或 RuntimeException，否则如果一个线程在运行中出现这样的问题，很有可能程序会无声无息地死去。所以一个线程的 run 方法中最好要有一个包含整个方法的 try…catch 块。

（4）当一个异常被抛出时，程序的执行就不再是线性的，它会跳过某一行，甚至会由于没有与之匹配的 Catch 子句而过早地返回。不管发生什么异常，有时需要保证每一段代码都要被执行到，关键词 Finally 就是用来标识这样一段代码的。即使没有 Catch 子句，Finally 程序块也会在执行 Try 程序块后的程序之前执行。每个 Try 语句都需要至少一个与之相配的 Catch 子句或 Finally 子句。

2. 重载函数的审查

在 C++中，重载函数经常出错的原因是 C++编译器对重载函数不能正确识别。编译器容易区分重载函数的参数类型、个数、顺序，但不能正确区分重载函数不同的返回值类型，所以对于不同返回值的重载函数，最好使用不同的函数名。

另外，因为重载函数是由编译器自动在多个函数中确定参数匹配是否正确的函数，所以这种多选一的方式，容易因为参数而出现歧义。例如，使用数字作为参数，系统可能会进行数据类型的隐式转换，有时会产生编译错误。

概念

函数重载。在 C++中函数重载就是允许两个或两个以上的函数名称相同，而参数类型或参数个数不同的函数存在于程序的某一相同的声明域中。当函数被调用时，编译器根据函数实参的类型和个数在多个名称相同的函数中寻找最佳的匹配函数，然后自动调用该函数。函数重载是面向对象程序设计中函数多态性的一种体现方式，而且使程序员便于记忆的同时提高了函数的易用性。

C++函数重载的实现是通过编译器对被调用函数的参数类型和个数的检查来实现的。在 C++中，对数据类型的检查和控制非常严格，如果编译器能找到参数的数据类型和个数都能匹配的函数，则调用成功，否则调用失败，编译器生成错误信息。

---

3. 内存分配和管理

当进行内存分配和管理时，很重要一点就是确保内存的及时释放和避免缓冲区溢出。内存没有及时被释放，系统运行时间长了，其性能就会大大降低。缓冲区溢出可能会产生严重的安全性问题，如一般用户得到超级用户的权限和破坏堆栈。

由于 C / C++语言不进行数组的边界检查，因此，在许多用 C / C++语言实现的程序中，总是假定缓冲区的长度是足够的。然而，事实并非如此，当程序出错或者一恶意用户故意写入一个过长的字符串时便会产生溢出，覆盖相邻的其他变量的空间，使变量的值不可预料。如果溢出部分碰巧覆盖了程序的返回地址单元，则子程序执行完毕返回时，就转到一个无法预知的地址，使程序流程执行发生错误。甚至会使应用程序访问了不在进程地址空间的地址，导致进程发生段违例故障。

避免缓冲区溢出的一种有效方法就是对指针的完整性进行检查和保护，即通过在所有的代码指针后面放置附加字节来检验指针在被调用之前的合法性。如果检验失败，则报警并退出程序执行，从而保证了系统的安全性。

4. 程序性能的审查

在代码级上提高程序性能，也是非常必要的。如何提高程序性能呢？首先，大家可能想到算法的优化，一些关键的排序、分类和计算的确需要这类工作以获得最优的算法。即使在今天，计算机的内存已很大且 CPU 运算能力也很强，算法还是不能忽视。算法的针对性很强，不同的问题有不同的算法，所以更重要的是，不要忽视一些通用的性能提高方法，例如：

- 减少创建对象。系统不仅要花时间生成对象，还需花时间对这些对象进行垃圾回收和处理，所以生成过多的对象将会给程序的性能带来很大的影响。
- 减少循环体的执行代码，能放在循环体外的代码要尽量放在循环体外。
- 提高处理异常出错的效率。由于用 try / catch 进行异常处理使用起来十分方便，导致过度使用异常成为一个趋势。尽管系统设置一个异常处理器无须开销，但对于一个异常的真正处理却总会花费很长时间。
- 减少 I/O 操作时间。如果在 Java 编程中大量地使用单字符 IO 操作（如 `InputStream.read`，`Reader.read` 方法），就会使 IO 性能大大降低。

示例

示例 1：

```
String name: new String("Hello," );
System.Out.println( name + "World!"):
```

这两行简单的代码，会生成 5 个对象变量，经过 15 个步骤完成初始化、复制、加载、调用、保存等一系列操作，非常耗资源。但如果将上述两条语句改为下列两条语句，则减少一个对象、4 项操作，性能提高一倍。

```
StringBuffer name=new StringBuffer("Hello, ");
System.out.println(name.append ("World!").toString())
```

示例2：

代码段1:

```
String name=new StringBufer("Hello," );
name + = "World!";
name + = " This is a test!";
```

代码段2:

```
StringBuffer name=new StringBuffer("Hello,");
name.append("World!");
name.append (" This is a test!"). toString();
```

代码段1生成8个对象，而代码段2只生成了4个对象，性能有很大提高。

6.2 单元测试

从广义上看，单元测试包括静态测试和动态测试，静态测试就是上一节所讨论的“程序代码的审查”活动，而动态测试就是本节要讨论的，通过单元测试用例、测试工具来执行程序以便更好地实现测试的目标。在静态测试中，也可以使用测试工具来完成对代码的检查，可以从编程语言的语法、规则和代码风格等各个方面来检查是否满足实现定义的要求。

单元测试和编程很难分离出来，两者完全处在并行、交互的过程中。一个良好的开发环境，要求在编程过程中随时根据需要进行单元测试。在任何一次代码的检入（Check-in）之前，都要先进行单元测试。只有测试通过了，代码才能被检入到由CVS、Subversion等构造的代码库中。也只有这样，才能保证代码的质量。

6.2.1 什么是单元测试

工厂在组装一台电视机之前，会对每个电子元件进行检验，或者要求电视机组装厂的每个供应商提供符合质量标准的电子元件，这种对电子元件的检验，就相当于软件中的单元测试。

单元测试可以说是软件测试中最基础的测试，集成测试、功能测试和系统测试都建立在单元测试之上，因为单元测试的对象是构成软件产品或系统的最小的独立单元，如封装

的类或对象、独立的函数、进程、子过程、组件或模块等。在单元测试活动中，软件的独立单元将与程序的其他部分一起被隔离开来。单元测试目的是检验每个软件单元能否正确地实现其功能，满足性能和接口要求，还要验证程序和详细设计说明的一致性。

单元测试主要强调被测试对象的独立性，也就是为了避免其他单元对该单元的影响，以获得被测单元的实际状态。所以，单元测试中的单元是软件系统或产品中可以被分离又能被测试的最小单元。如果某个组件比较大，可以进一步分离某些部分，直至分离到一个可接受的程度，这取决于可测试性。也就是说，可以分解到某个类、某个函数或某个程序子过程，但也不需要无止境地分解下去，如不需要分解到每一行代码或每一个变量。

更确切地说，单元测试的概念强调的是基本的质量思想，如果要保证一个系统的质量，首先要保证构成这个系统的所有组成成分的质量。例如，要保证一栋房屋的质量，不仅要保证其框架设计合理、基础厚实、造型美观等整体质量，还需要考虑柱和地基的连接、梁和柱的连接、门框和墙的连接，而且需要从一砖一瓦、一钉一铆开始，使柱和梁的浇注、砌墙、门和窗户的油漆、水管和电线的布置等各个部分自身的质量得到充分保证。如果不这样，柱、梁、墙等任何一部分出问题，这个房屋结构都会受影响而成为危房，甚至倒塌。即使没有成为危房，也会在使用门、窗、水电等的过程中出现许多其他的质量问题。成熟的制造业（如冰箱、彩电、汽车等）非常注重配套厂商所提供的电子元件、机械零件的质量，他们会要求元件供应商严格遵守国家质量标准，并在产品组装之前进行全面的质量检查。

所以说，单元测试是一种思想和理念的体现，相对集成测试、系统测试而存在，不在于究竟“什么是单元”。单元是构造系统的基础，只有使每个单元得到足够的测试，系统的质量才能有可靠的保证，所以单元测试是非常重要的。单元测试就是确保构成软件系统的各个成分得到足够的测试，只有通过了单元测试，后续的集成测试和系统测试才能顺利实施。甚至可以说，只有每行代码、每个变量、每个输入数据、每个函数调用的参数和返回值等都被测试过，我们才能对软件质量有足够的信心。

单元测试强调的另一个思想就是，开发人员不能有依赖性，直接写好的每一行代码都应是正确的、有效的，交到测试人员手中的软件包应具有很高的质量。如果没有进行单元测试，交给测试人员的软件包可能一运行就漏洞百出，功能测试和系统测试很难进行下去。单元测试的这种思想，进一步演化，就成为测试驱动开发或极限编程的思想，从测试、用例出发来进行软件开发，单元测试应伴随着编程过程中的每一个时刻。

概念

单元测试的概念

RUP 的定义：单元测试的目的是把已实现的构件作为个体单元进行测试，主要有以下两类单元测试：

- 规格说明测试或“黑盒测试”，用于验证单元外观上可观察的行为。
- 结构测试或“白盒测试”，用于验证单元的内部实现。

软件工程知识体系指南的定义：单元测试是验证那些可单独被测试的构件相互隔离的功能特性。依据上下文（根据不同的情况），被测试的部分（单元）可能是单个的子程序，也可能是由若干个密切相关的单元构成的更大的组件。

调试与单元测试的对象及采用的方法有一定的相似性，它们都会用到断点跟踪和控制等排错方法，但两者的目的却完全不同。测试是为了找出软件中存在的缺陷，而调试是为了修正测试已发现的缺陷并试图找出缺陷产生的根本原因。

6.2.2 单元测试的现状和作用

实际测试工作的经验告诉我们，如果仅对软件进行功能测试、验收测试，似乎缺陷总是找不完，不是这边出现错误，就是那个角落发现问题，每天报告的缺陷虽不多，但总能发现新的且比较严重的缺陷，测试没有尽头。为什么会出现这种情况？

产生这种现象的主要原因就是在功能测试、验收测试之前没有进行充分的单元测试。虽然，我们清楚测试是不能穷尽所有程序路径的，但单元是整个软件的构成基础，如果没进行单元测试，基础就不稳，光靠功能测试、验收测试在上面捅来捅去，不能彻底解决问题。单元的质量是整个软件质量的基础，所以充分的单元测试是必要的。

每一个程序员都知道应该为自己的代码编写测试程序，但却很少这样做。当人们问为什么的时候，最常听到的回答就是：“任务安排太紧，没时间”。却不知这会导致恶性循环，越是没空进行单元测试，代码的效率与质量就越差，以致于不得不占用更多的时间修正缺陷，实际的开发效率反而大大降低了。由于效率降低了，因此时间更紧张，压力更大，代码质量更差，进入恶性循环。

一旦编码完成，有的开发人员就会迫不及待地进行软件集成工作。在这种情形下，系统能进行正常工作的可能性不大，在多数情况下是充满了各式各样的缺陷。有些缺陷在单

元测试中很容易被发现，但如果没有进行单元测试而留到后期，缺陷就隐藏得越来越深，在系统测试中则难以发现，最终的结果将导致测试成本急剧增加。从这个角度看，单元测试的效率高，能降低软件开发的成本。缺陷在软件里遗留的时间越长，所带来的不良成本就越大，而且时间和成本之间的关系不是简单的线性关系，而是成指数的非线性关系。

即使在功能测试、系统测试阶段发现了缺陷，为了修正缺陷而修改了代码。如果在修正缺陷的时候不进行单元测试，修正一个缺陷，可能会产生两个、3 个或更多的回归缺陷，结果缺陷没有减少，反而增多了。这样，项目就会进入恶性循环，没完没了。这肯定不是我们希望的，所以单元测试在后期阶段也是重要的，无论是写新的代码，还是修改代码，只要代码有变化，就需要单元测试。

比单元测试更重要的是要以“缺陷预防”为主，建立严格的编程规范，加强培训，不产生缺陷就可以减少单元测试的工作量。为了更好地理解这一点，可以看看工匠是如何砌墙的。

- 工匠甲，先拉上一根水平线，砌每一块砖时，都与这根水平线进行比较，使得每一块砖都保持水平。
- 工匠乙，等一排砖都砌完后，再拉上一根水平线，看看哪些砖有问题，对有问题的已砌好的砖进行调整。

显然，工匠甲的做法是聪明的，也是被广泛采用的，而工匠乙的办法是很笨的，其结果往往会推倒已砌好的墙，重头再来，是不是太浪费时间？然而仔细想想，许多程序员在编写程序的时候，不正是像工匠乙那样编程吗？甚至有时候，有些程序员比工匠乙还笨，是等到整面墙都砌完了，再拉上几根水平线来检查（等到集成测试和系统测试），然后不得不推倒重来。

6.2.3 单元测试的方法

出于实施的方便性和高效性，单元测试主要以编程人员为主体来进行。单元测试和编程同步、交互式进行，每完成一个函数、一个类、一个模块就及时进行相应的单元测试，确保持续测试和持续集成。

其实，程序员随时都在做单元测试。当程序员写完一个函数，总是要调试一下的，给这个函数几个不同的参数来检查一下运行是否正常，返回值是否正确。这种随意的单元测试，对提高软件质量有帮助，但不够可靠，缺乏系统性的测试代码，代码覆盖率往往比较低，在 50%以下。要保证可靠的软件质量，必须有意识地、系统地进行单元测试。

有效的单元测试需要采用正确的方法，设计完整的测试用例。单元测试用例的设计方法可归属于白盒测试方法与黑盒测试方法，但以白盒测试方法为主，并适当地结合黑盒测试方法。

1. 白盒方法的单元测试

白盒测试将被测程序看作一个打开的盒子，对代码进行了全面的逻辑分析（如条件和路径分析）之后准确定位，进行有效的测试。对于一个单元（一般是几十行或几百行代码），这种逻辑分析是现实的，但对于一个复杂的软件系统（可能是几十万行或几百万行代码）来说，根本不可能全面地完成逻辑分析，所以白盒测试非常适合进行单元测试。

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。白盒测试基于对模块内部结构清晰的了解，要求验证内部动作是否按照规格说明书的规定正常进行，并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，使程序中每一条可执行语句至少被执行一次。
- 判定覆盖（也叫分支覆盖）。选择足够的测试用例，使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。
- 条件覆盖。选择足够的测试用例，使程序中每一个分支判断中的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，使之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，使程序中每一个分支判断中的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，使程序中所有可能的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。在对单元进行白盒测试的过程中，有时需要基于被测试单元的接口，开发出相应的驱动程序/模块（Driver）和桩程序/模块（Stub）。当然，Driver 和 Stub 更多的是应用在集成测试中。

2. 黑盒方法的单元测试

通过白盒测试方法，虽然测试覆盖了每一行语句、所有条件和分支等，可以保证在程序上没有问题，但还是不能保证在产品功能特性上没有问题。因为白盒测试只关注代码，如果编程之前对产品特性的理解就有问题，这种误解遗留在代码中是很难仅仅通过代码测试发现的。所以，单元测试也需要黑盒测试方法。

黑盒测试方法主要运用于测试单元的功能和性能，以检验程序真正的行为是否与产品规格说明、客户的需求保持一致。单元测试，不仅要验证软件的功能表现，而且要验证代

码结构上的可靠性、健全性和性能，如对单元进行数据组合的测试、负载测试和性能测试等。在功能性测试方面，通常会利用 3 种数据来进行测试，即正常数据、边缘数据和错误数据，也就是第 4 章介绍的等价类划分法、边界值分析法等。

概念

- **驱动模块**（Driver）：在对底层或子层模块进行单元测试或集成测试时，所编制的调用被测模块的程序用于模拟被测模块的上级模块。驱动模块接受测试数据，把相关的数据传送给被测的下一层模块，启动/完成对该模块的测试。
- **桩模块**（Stub）：对顶层或上层模块进行单元测试或集成测试时，所编制的替代下层模块的程序用于模拟被测模块工作过程中所调用的模块。桩模块相当于电路中的短路器，使上层模块不需要调用真实模块就能获得所需要的参数、返回值等。

驱动模块和桩模块都尽量简单，基本没有数据运算处理等，可直接调用函数或返回某个参数值（如图 6-1 所示）。

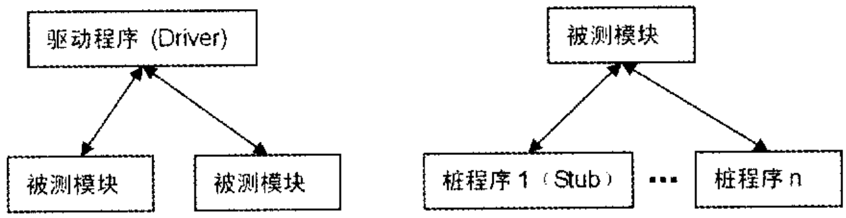


图 6-1

6.3 单元测试用例的设计

在第 4 章介绍了如何用黑盒方法设计测试用例，对于单元测试用例的设计和程序的实现过程，主要集中在白盒测试方法之上，并力求达到下列测试要求。

- 对程序模块所有独立的执行路径至少要测试一次。
- 对所有的逻辑判定，其结果为真、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。

6.3.1 语句覆盖法

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被执行一次。

示例

示例程序 3：

```
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
```

对于给出的源程序示例 3，可以得到程序流程图和对应的程序路径图，如图 6-2 所示，其中

- 条件 M={a>0 and b>0}。
- 条件 N={a>1 or c>1}。

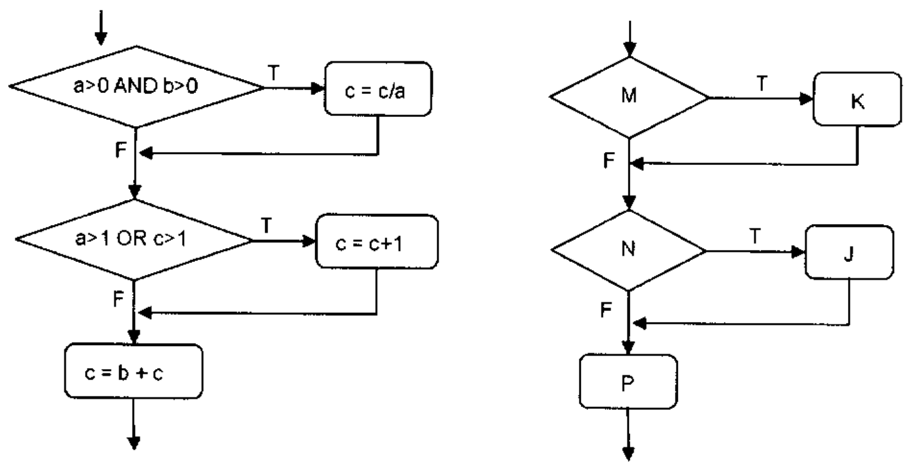


图 6-2 程序流程图和简化路径图

根据程序流程/路径图可以看出，该程序模块有 **4** 条不同的路径。它们实际对应条件 M 和条件 N：

- P1：（M -K -N -J -P） 对应 M = .T. and N= .T.
- P2：（M -K -N -P） 对应 M = .T. and N= .F.
- P3：（M -N -J -P） 对应 M = .F. and N= .T.
- P4：（M -N -P） 对应 M = .F. and N= .F.

因为 P1 包含了所有可执行语句，所以按照语句覆盖的测试用例设计原则，只要设计一组测试用例覆盖 P1 路径，就可达到要求。如果设定输入数据 a=2，b=1，c=6，此时满足条件M＝{a>0 and b>0} 和条件N＝{a>1 or c>1}，即只要一条测试用例就能覆盖所有程序语句。

在使用语句覆盖法进行测试用例设计时，能够使得所有的执行语句都被测试，但是不能准确判断运算中的逻辑关系错误。在这个例子里面，如果程序将 M 的条件“（a > 0 AND b > 0）”写成“（a > 0 OR b > 0）”，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，而且可能会导致第 4 行语句“c = c/ a”被零除而溢出。从这一点看，仅仅语句覆盖测试是远远不够的。

6.3.2 判定和条件覆盖法

判定覆盖法和条件覆盖法都是基于逻辑判断来设计测试用例的，即通过设计若干用例运行被测程序，使程序中的各个分支都至少被覆盖一次。判定覆盖法，是使每个判断的取真值和取假值都至少经历一次，但真假分支也可以看作是条件分支的一种特殊情况。

1．判定覆盖

针对上面提到的测试用例设计，按照判定覆盖的基本思路有两种组合，如表 6-2 所示，这两种组合都可以达到判定覆盖的要求——即 M 和 N 的每个判定都至少有一次取真、一次取假。

表 6-2 两种组合的判定覆盖

组合 1: P1 + P4	组合 2: P2 + P3
P1: M = .T. and N= .T. P4: M = .F. and N= .F.	P2: M = .T. and N= .F. P3: M = .F. and N= .T.
输入数据 {a=2，b=1，c=6} {a=2，b= -2，c=6}	输入数据 {a=1，b=1，c= -2} {a= 2，b= -2 ，c=6}
M = {a>0 and b>0}，N = {a>1 or c>1}	

用判定覆盖法设计测试用例时，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”，组合 1 的测试结果没有差别。

2. 条件覆盖

对于第一个判定条件 M，可以进一步分为：

- 条件 a>0： 取真（TRUE）时为 T1，取假（FALSE）时为 F1。
- 条件 b>0： 取真时为 T2，取假时为 F2。

对于第二个判定条件 N：

- 条件 a>1： 取真时为 T3，取假时为 F3。
- 条件 c>1： 取真时为 T4，取假时为 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 6-3 所示。从表中的结果分析，可以得到：

- 保证每个条件至少有一次取真、取假（T1/F1，T2/F2，T3/F3，T4/F4），满足了条件覆盖的要求，但是覆盖了相同的路径 P3。这说明，即使做到条件覆盖，也不能保证覆盖所有路径（P1、P2、P3 和 P4）。
- 这个例子还告诉我们，测试用例满足看似更复杂、要求更高的条件覆盖，事实上也不一定能满足判定覆盖的要求。这里，每个条件至少有一次取真、取假（T1/F1，T2/F2，T3/F3，T4/F4）。而 M 只获得假值、N 只获得真值，不是每个判定条件（M 或 N）“真”、“假”值至少被执行一次。

只是满足一种覆盖条件，还会遗漏程序逻辑路径或判定错误，依旧存在较大风险。所以，白盒测试要求同时满足两种以上的测试覆盖要求，测试的覆盖率才能满足质量要求，将风险降到很低。

表 6-3 条件覆盖的测试用例

测试用例（输入数据）	取值条件	具体取值条件	通过路径
a=2，b=－1，c=－2	T1，F2，T3，F4	M = T1 AND F2 = .F. N = T3 OR F4 = .T.	P3: （M –N –J–P）
a=－1，b=2，c=3	F1，T2，F3，T4	M = F1 AND T2 = .F. N = F3 OR T4 = .T.	P3: （M –N –J–P）

4. 判定—条件覆盖

判定—条件覆盖，是将前两种方法结合起来的一种测试用例设计方法，它弥补了单个覆盖方法的不足。判定—条件覆盖方法，确保设计足够或精巧的测试用例，可使得判断条件中所有条件的可能取值至少被执行一次，同时，所有判断的可能结果也至少被执行一次。

按照这种思想，在前面的例子中，保证判定条件 M 和 N 至少取真/假一次，同时要保证 8 个条件取值（T1，F1，T2，F2，…，F4）至少被执行一次。表 6-3 的测试数据被修改为如表 6-4 所示。这种方法的覆盖率高，但也不能保证覆盖所有路径，如本例中，只覆盖了路径 P1 和 P4，没有覆盖 P2 和 P3。

表 6-4 判定—条件覆盖的测试用例

测试用例（输入数据）	取值条件	具体取值条件	通过路径
a=2，b=1，c=6	T1，T2，T3，T4	M = T1 AND T2 = .T. N = T3 OR T4 = .T.	P1:（M –K –N –J –P）
a= – 1，b= – 2，c= – 3	F1，F2，F3，F4	M = F1 AND F2 = .F. N = F3 OR F4 = .F.	P4:（M –N–P）

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。与判定—条件覆盖的差别是，条件组合覆盖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，要设计测试用例满足 8 种组合，如表 6-5 所示。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 6-6 所示。从表 6-6 可以看出，条件组合覆盖就是将条件覆盖中的条件组合成满足判定覆盖的要求，即将条件覆盖和判定覆盖两种方法有机的结合起来。即使这样，从上面的测试用例中仍可以看到，还是不能保证所有的路径被执行（P2 没有被执行），所以条件组合覆盖设计方法也有不足之处。

表 6-5 需满足的 8 种组合条件

编 号	组合条件要求	判定条件取值	编 号	组合条件要求	判定条件取值
1	T1，T2	M = .T.	5	T3，T4	N = .T.
2	T1，F2	M = .F.	6	T3，F4	N = .T.

续表

编　号	组合条件要求	判定条件取值	编　号	组合条件要求	判定条件取值
3	F1，T2	M = .F.	7	F3，T4	N = .T.
4	F1，F2	M = .F.	8	F3，F4	N = .F.

表 6-6　条件组合覆盖的测试用例

测试用例（输入数据）	覆盖条件	覆盖路径	覆盖组合
输入：a=2，b=1，c=6	T1，T2，T3，T4	P1（M→K→N→J→P）	1，5
输入：a=2，b=－1，c=－2	T1，F2，T3，F4	P3（M→N→J→P）	2，6
输入：a=－1，b=2，c=3	F1，T2，F3，T4	P3（M→N→J→P）	3，7
输入：a=－1，b=－2，c=－3	F1，F2，F3，F4	P4（M→N→P）	4，8

6.3.3　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，导出基本可执行的路径集合，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被执行一次。基本路径测试法，包括以下5个方面：

（1）**程序的流程图**：程序流程控制图是描述程序控制流的一种图示方法，我们可以使用流程图软件，如 Microsoft 的 Visio 来实现程序流程地描述。

（2）**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的复杂性。从程序的环路复杂性可导出程序基本路径集合中的独立路径条数。

（3）**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。

（4）**准备测试用例**，确保基本路径集的每一条路径都被执行。

（5）**图形矩阵**：它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。

顾名思义，路径覆盖就是设计所有的测试用例，以覆盖程序中的所有可能的执行路径。可以采用下面的测试用例，来完全覆盖路径 P1，P2，P3，P4，如表 6-7 所示。但也要看到，路径覆盖法没有涵盖所有的组合条件，如本例子中，没有覆盖组合2、3和6。

表 6-7 路径覆盖的测试用例

测试用例	覆盖路径	覆盖条件	覆盖组合
输入：a=2，b=1，c=6	P1（M–K–N–J–P）	T1，T2，T3，T4	1，5
输入：a=1，b=1，c=－3	P2（M–K–N–P）	T1，T2，F3，F4	1，8
输入：a=－1，b=2，c=3	P3（M–N–J–P）	F1，F2，F3，T4	4，7
输入：a=－1，b=－2，c=－3	P4（M–N–P）	F1，F2，F3，F4	4，8

6.3.4 多种白盒测试方法的比较和总结

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，而且，每种方法有其特定的优点和缺点，如表 6-8 所示。因此，在实际的测试用例设计过程中，可以根据测试用例设计的需要和不同的特征，将不同的设计方法有效地结合起来使用，以设计出覆盖面最大、最有效的测试用例。

表 6-8 白盒测试方法对比

方 法	分支覆盖	条件覆盖	条件组合覆盖	基本路径覆盖
优点	简单，无须细分每个判定	增加了对符合判定情况的测试	对程序进行较彻底的测试，覆盖面广	清晰、测试用例有效
缺点	往往大部分的判定语句是由多个逻辑条件组合而成（如包含 AND、OR、CASE 等的组合），若仅仅判断其组合条件的结果，而忽略每个条件的取值情况，必然会遗漏部分测试场景	要达到条件覆盖，需要足够多的测试用例，但条件覆盖还是不能保证判定覆盖。这是由于 AND 和 OR 不同的组合效果造成的	对所有可能的条件进行测试，需要设计大量、复杂的测试用例，工作量比较大	基本路径法，类似于分支的方法，不能覆盖一些特定的条件，这些条件往往是容易出错的地方

最完善的单元测试是，结合条件组合和路径覆盖两种方法来重新设计测试用例，如表 6-9 所示，这个测试用例就能覆盖该例子程序的所有测试。

表 6-9　完整的测试用例

测试用例	覆盖路径	覆盖条件	覆盖组合
输入：a=2，b=1，c=6	P1	T1，T2，T3，T4	1，5
输入：a=1，b=1，c=－3	P2	T1，T2，F3，F4	1，8
输入：a=－1，b=2，c=3	P3	F1，F2，F3，T4	4，7
输入：a=－1，b=－2，c=－3	P4	F1，F2，F3，F4	4，8
输入：a=2，b=－1，c=－2	P3	T1，F2，T3，F4	2，6
输入：a=－1，b=2，c=3	P4	F1，T2，F3，T4	3，7

6.3.5　循环结构的测试用例

循环结构在软件程序中的应用较多，但循环结构的测试用例相对比较特殊，上述白盒测试方法都不完全适应，需要将条件覆盖方法、路径覆盖方法和黑盒测试方法中的等价类划分、边界值分析结合起来，才能解决问题。

循环结构有单循环、嵌套循环、并列循环等多种形式，如图 6-3 所示。为了更清楚地讨论问题，我们先从单循环结构开始，然后逐步深入。

1. 单循环结构

单循环结构，如果按照条件覆盖或路径覆盖方法完成测试，只要覆盖两种情况，假定 n 表示循环允许的最大次数、i 表示循环变量，其测试用例如下所示：

- 满足循环条件（i <= n），在循环体内执行一遍。
- 不满足循环条件，执行循环体外语句。

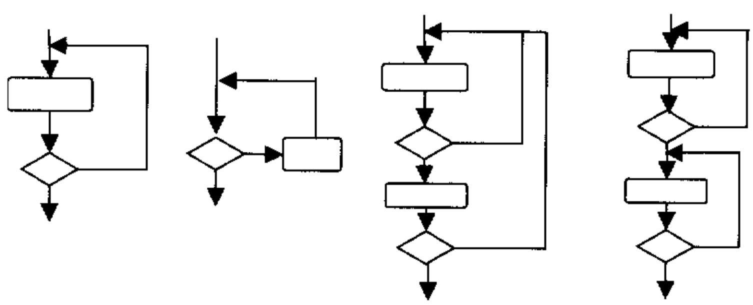


图 6-3　循环结构的各种形式

但这肯定远远不够，需要考虑循环次数的边界值和接近边界值的情况，即一般要执行7个测试用例，对应i=0，1，2，m，n-1，n，n+1，因为边界值是1和n。

（1）零次循环：从循环入口直接跳到循环出口。

（2）一次循环：查找循环初始值方面的错误。

（3）二次循环：查找循环初始值方面的错误。

（4）m 次循环，此时的 $m<n$ ，也是检查在多次循环时才能暴露的错误。

（5）比最大循环次数少一次。

（6）最大循环次数。

（7）比最大循环次数多一次。

如果想提高测试效率，至少要执行5个测试用例，即i=0，1，m，n，n+1。

2. 嵌套循环

对于嵌套循环，不能将单循环的测试方法简单地扩展到嵌套循环，因为这样测试数目可能将随嵌套层次的增加呈几何倍数增长。例如，两层嵌套循环，可能要运行 $7^2=49$ 个测试用例；如果3层嵌套循环，可能要运行 $7^3=343$ 个测试用例。

对于嵌套循环，下面的方法更有效，有助于减少测试用例数目。

- 从最内层循环开始（不含最内层循环），将所有其他层的循环置为最小值。
- 对最内层循环做简单循环的全部测试。测试时保持所有外层循环的循环变量取最小值，另外，对越界值和非法值做类似的测试。
- 逐步外推，对其外面一层循环进行测试。测试时保持所有外层循环的循环变量取最小值，所有其他嵌套内层循环的循环变量取“典型”值。
- 反复进行，直到所有各层循环测试完毕。

对全部各层循环，同时取最小循环次数或同时取最大循环次数。对于后一种测试，由于测试量太大，需人为指定最大循环次数。最大循环次数也应该是测试的重点。

3. 并列循环

并列循环要区别两种情况。如果各个循环是互相独立的，则自然可以简化为两个单循环来分别处理。但如果几个循环不是互相独立的，例如前一个循环的循环变量值作为后一个循环的初值，则需要使用嵌套循环的测试用例设计方法来处理。

6.3.6　单元测试的典型实例

通过一个简单的函数测试，以小见大，了解单元测试的方法和如何设计测试用例。以雅虎日历的“添加待办事项”功能为例，如图 6-4 所示。查看源文件（如示例程序 4），可以看到如下的一个函数 charCountFx，可用它来判断字符串长度是否符合预设的最大值要求。

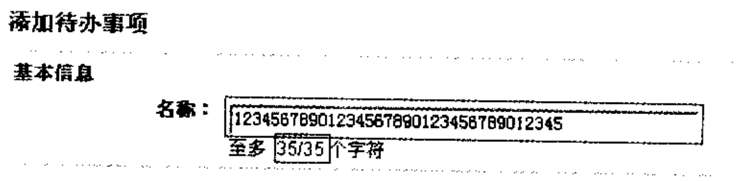


图 6-4　添加待办事项的名称

在 B/S 结构的程序，众所周知，对输入数据长度的判断是个非常通用的功能，charCountFx 就是完成这样一个功能的函数。在雅虎日历中，待办事项名称最大可允许输入的字段长度为 35 个字符。当用户输入字符时，“至多输入 N/35 个字符”提示中的“N”就会替换为当前文本区域的字符个数。当输入长度超过最大上限时，输入框的底色则会从白色转换成黄色，以示警告。由此可以得出函数 charCountFx 主要完成 3 项功能：

- 输入长度显示。
- 字符在最大限度内时，设定的“正常效果”显示。
- 字符超过最大限制时，设定的“错误的效果”显示。

示例

示例程序 4：

```
/**function charCountFx 文本框输入限定字数
curField,待处理的文本输入框
fieldName,用于显示字数统计的区域
maxLimit,文本框可输入的最大限制
errClass,当超过文本字数时，文本区域显示的效果 css
normalClass,当字数没有超过时，文本区域显示的正常效果 css
**/
function charCountFx(curField, fieldName, maxLimit, errClass,
   normalClass) {
      var maxCharCount = maxLimit;
      var charCount = curField.value.length;//取得输入框里输入数据的
         长度
      if (charCount >= maxLimit) {
         if (errClass != "") {    //如果指定了错误的显示效果类
```

```
            curField.className = errClass;
        }
    } else if (normalClass != "") {
        curField.className = normalClass;//设定正常效果的显示
    }
    var remField = document.getElementById(fieldName);
            //取得显示字数统计的域
    var babel = remField.innerHTML;//取得该域的文本
    babel = babel.replace(/(\d+)\/(\d+)/, "$2");
            //转换文本中数字格式为m/n
    babel = babel.replace(/(\d+)/,(((charCount > maxCharCount ) ?
        maxCharCount : charCount) + "/$1"));
        //将m替换为当前文本的字符个数
    remField.innerHTML = babel;//获得新的显示文本
}
```

根据上面 charCountFx 函数的流程图可以看到 4 条基本路径，如图 6-5 所示：

（1）A->B->F->G

（2）A->B->D->G

（3）A->C->D->G

（4）A->C->E->G

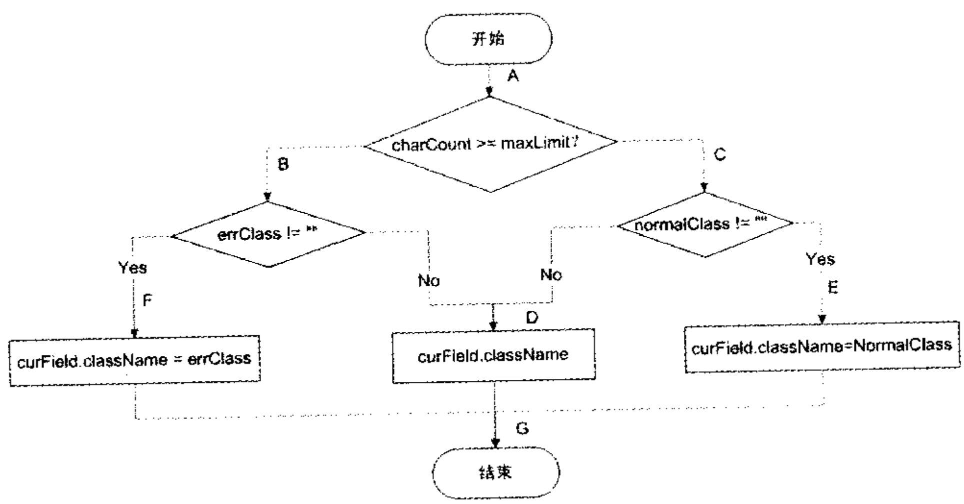


图 6-5 charCountFx 函数的流程图

在进行单元测试时，首先会想到用表 6-10 中的测试用例来覆盖 4 条基本路径。

表 6-10 条件（分支）覆盖的测试用例

用例序号	curField 长度	maxLimit 限定长度	errClass	normalClass	路 径
1	5	4	空		ABDG
2	5	4	非空字符串		ABFG
3	2	5		空	ACDG
4	2	5		非空字符串	ACEG

由于这里都是单一的判断条件，即判定条件就等于分支条件，所以基本路径与条件覆盖、分支覆盖的组合是一致的，无需再设计其他的测试用例，一种方法就达到了单元测试所需要的覆盖率。另外，为了提高测试覆盖率，可增加边界条件的测试用例，如“curField 长度=maxLimit”这种情况，如表 6-11 所示。

表 6-11 边界条件的测试用例

用例序号	curField 长度	maxLimit 限定长度	errClass	normalClass	路 径
1	5	5	空		ABDG
2	5	5	非空字符串		ABFG

6.4 单元测试工具

由于单元测试一般采用白盒测试方法，它会针对程序内部的实现结构来展开，所以单元测试工具一般都是针对不同的编程语言和不同的开发环境而设计开发的，如：

- 针对 C/C++语言的单元级测试工具有 CppUnit、Parasoft C++Test、Panorama C++ 和 Mercury Numega 等；
- 针对 Java 语言的单元测试工具有 Parasoft Jtest、开源工具有 JUnit 等。

当然，也可以分为静态的单元测试工具和动态的单元测试工具，而且以静态测试工具居多。

如果根据工具本身的功能特点进行分类，单元测试工具可分为以下几类：

- 内存资源泄漏检查工具，如 Numega 中的 BounceChecker, Rational 的 Purify 等。
- 代码覆盖率检查工具，如 Numega 的 TrueCoverage, Rational 的 PureCoverage,

TeleLogic 公司的 Logiscope。

- 代码性能检查工具，如 Logiscope 和 Macabe 等。

6.4.1 静态代码分析

静态源代码分析工具，如表 6-12 所示，类似于编译器，能够检查源代码，发现违反编程语言语法规则和大量定义编程规范的代码。下面以 C++Test 和 JTest 作为例子，来介绍静态源代码分析工具一般具有的功能特性。

表 6-12 静态单元测试工具

类 别	工 具
C 语言	C++ Test、CppUnit、QA C/C++、CodeWzard、Insure++6.0
Java 语言	Jtest、JUnit、Jmock、EasyMock、MockRunner
JUnit 扩展框架	TestNG 、JWebUnit 和 HttpUnit
GUI （功能）	JFCUnit、Marathor
通用的	Rexelint、Splint、McCabe QA、CodeCheck、GateKeeper
.NET	.TEST、NUnit
Data Object，DAO	DDTUnit，DBUnit
EJB	MockEJB 或者 MockRunner
Servlet，Struts	Cactu，StrutsUnitTest
XML	XMLUnit
内嵌式系统等	Logiscope、JTestCase

1. JTest

JTest 通过自动生成和执行全面测试类代码及其分支的测试用例，来较彻底地检查被测类的结构。JTest 使用一个符号化的虚拟机执行类，并搜寻未捕获的运行异常。对于检测到的每个异常情况，JTest 报告都一个错误，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）而产生一个异常时。

- 行为正确的方法，即方法的正确输出是产生一个异常。在这种情形下，建议修正代码，将这类异常的产生置于方法的 throw 子句中。
- 仅为开发人员使用的方法。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唯一使用者，并且不传递这些输入参数。

2. C++Test

C++Test 能够自动测试 C/C++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），其单元级的测试覆盖率可以达到 100%。C++Test 具有的特性有：

- C++Test 直接安装在 DevStudio 环境中，即时测试类/函数。
- 支持极限编程（XP）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，并允许定制这些桩函数的返回值或加入自己的桩函数。
- 单键执行白盒测试的所有步骤。
- 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）。
- 高度可定制的，例如，可以改变测试用例的生成参数，过滤一定的文件、类或方法，在任何层次上进行测试。

6.4.2 测试覆盖率工具 EMMA

EMMA 是用于检测和报告 Java 代码覆盖率的开源工具，支持多种级别的覆盖率指标：包（package）、类（class）、方法（method）、语句块（basic block）和行（line），而且能测出某一语句行是否只是被部分覆盖，如条件语句短路的情况。

EMMA 能生成 text、xml 或 html 等形式的报告，以满足不同的需求。其 html 报告提供下钻功能，能够从 package 开始一步步链接到代码中所关注的某个方法。EMMA 能和 Makefile 及 Ant 集成，效率很高，便于应用于大型项目。

EMMA 通过 byte instrumentation 生成.class 文件的增强版本，其中加入了统计测试覆盖率的代码 Hook，相当于用 EMMA 实现的 application class loader 替代了原有 Java 的 application class loader。其中使用了两种模式——“offline”和“on-the-fly”来实现覆盖率的统计。

- 对于“offline”模式，它从硬盘读入.class 文件，然后输出经过 Instrumented 的增强版本。即在类被加载前，加入字节码。
- 对于“on-the-fly”模式，操作发生在 JVM 内部，不保存在硬盘中。

Offline 是通用的模式，虽然 on-the-fly 模式比较方便，但只能用于简单的 Java 应用程序，因为它不能为被 boot class loader 加载的类生成覆盖率报告，也不能为被那种独特的 class loader 加载的类生成覆盖率报告。

示例

```
<!- 在 Ant 脚本允许 emma、并指示 emma.jar 和 emma_ant.jar 的路径-->
... ...
<path id="emma.lib">
  <pathelement location="${libs}/emma.jar" />
<pathelement location="${libs}/emma_ant.jar" />
</path>
<property name="emma.enabled" value="true" />

<!-为 ANT 添加 EMMA 任务-->
<taskdef resource="emma_ant.properties" classpathref="emma.lib" />

<!- 对编译在路径 bin.main.dir 中的 Java 类注入字节码，并且把注入字节码的新 Java 类存放
到路径 bin.instrument.dir -->
<!-覆盖率的元数据存放在路径 coverage.dir 中-->
<target name="instrument">
    <mkdir dir="${bin.instrument.dir}" />
    <mkdir dir="${coverage.dir}" />
    <emma enabled="${emma.enabled}">
        <instr instrpathref="classpath.main"
            destdir="${bin.instrument.dir}"
            metadatafile="${coverage.dir}/metadata.emma" merge="true">
        </instr>
    </emma>
    <copy todir="${bin.instrument.dir}">
        <fileset dir="${bin.main.dir}">
            <exclude name="**/*.java" />
        </fileset>
    </copy>
</target>
```

通过上面的示例，可以基本了解 EMMA 的具体使用方法。如果想深入了解和使用 EMMA，请访问相关站点查阅资料，如 *http://emma.sourceforge.net/*。图 6-6 是一个 EMMA 产生的真实代码覆盖率的报告。图中把包类（class）、方法（method）、语句块（basic block）和行（line）等 4 个指标的覆盖率一一列出，并针对每个包进行了详细的分析。

EMMA Coverage Report (generated Tue Mar 18 22:13:27 CDT 2007)

[all classes]

OVERALL COVERAGE SUMMARY

name	class, %	method, %	block, %	line, %
all classes	86% (132/153)	62% (884/1429)	56% (24654/44008)	58% (5260.5/9141)

OVERALL STATS SUMMARY

```
total packages:          26
total executable files:  136
total classes:           153
total methods:           1429
total executable lines:  9141
```

COVERAGE BREAKDOWN BY PACKAGE

name	class, %	method, %	block, %	line, %
org.apache.velocity.app.tools	0% (0/3)	0% (0/23)	0% (0/338)	0% (0/66)
org.apache.velocity.convert	0% (0/1)	0% (0/10)	0% (0/447)	0% (0/65)
org.apache.velocity.io	0% (0/1)	0% (0/20)	0% (0/348)	0% (0/97)
org.apache.velocity.runtime.compiler	0% (0/1)	0% (0/2)	0% (0/330)	0% (0/50)
org.apache.velocity.servlet	100% (1/1)	33% (6/18)	16% (65/399)	18% (18.5/10[illegible]
org.apache.velocity.runtime.visitor	67% (2/3)	26% (20/77)	28% (152/551)	32% (44/137)

图 6-6 EMMA 生成的覆盖率报表示意图

概念

测试覆盖率（Code Coverage）是评价测试活动覆盖产品代码的指标，也就是评价代码测试工作的质量。测试越充分，软件产品的质量越高，所以测试覆盖率可以看作是产品代码质量的间接指标。基于白盒测试方法，测试覆盖率比较容易得到确定。如果在代码中事先嵌入一些跟踪代码，并借助特定的工具，也可以确定黑盒测试方法的测试覆盖率。

测试覆盖率的几种指标：

- 语句覆盖率（Statement coverage），或者称代码行覆盖率（code line coverage），用于评价测试的代码语句覆盖率。
- 基本语句块覆盖率（Basic block coverage）是把没有分支的代码块（若干行代码构成的区域）作为一个计量单位，而不是简单的代码行。
- 决策覆盖率（Decision coverage），也称作分支覆盖率（Branch coverage），用于评价测试的程序分支覆盖率。
- 路径覆盖率（Path coverage）用于评价代码从开始到结束所经历的路径中有多少路径被测试。
- 功能覆盖率（Function coverage），用于评价代码方法被测试的覆盖率。

6.5 小结

在编程和单元测试交互的过程中，首先要强调代码的评审，因为 60%以上的代码缺陷可以通过这种手段发现出来。代码的审查，不仅能有效地发现缺陷，更重要是为缺陷预防获取各种经验，从根本上提高代码的质量。代码审查，可以采取互查、走查、会议评审等多种形式：

- 互查，是非正式的形式，但方便、有效，可以经常使用。
- 走查，较为正式，对代码从头到尾的检查，能比较全面地发现代码中的缺陷。
- 会议评审，最为正式的一种形式，主要用于关键性代码的审查。

单元测试是一种思想和理念的体现，只有使每个单元得到足够的测试，系统的质量才有可靠的保证。单元测试的对象是程序系统中可分解的、可测试的最小单元，其目标不仅是测试代码的功能性，还须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，其中编程人员起主要作用。

单元测试动态技术主要是白盒测试方法，辅之以黑盒测试法方法。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。由于软件单元规模小、功能单一和逻辑简单，通过模块说明书和源程序能清楚地了解单元的 I/O 条件和逻辑结构，所以采用白盒方法的测试用例，可以对单元进行彻底的测试，达到很高的测试覆盖率。白盒测试方法有逻辑驱动法和基本路径测试法，包括语句覆盖法、判定覆盖法、条件覆盖法和条件覆盖法。

单元测试更有效的方法是借助各种单元测试工具实现测试的自动化，包括：

- 商业化测试工具和开源的测试工具。
- 代码静态分析工具和动态测试工具。
- 针对 C/C++的测试工具和更多的 Java 测试工具。
- 其他各种工具。

7

第7章

功能测试的执行

一个有效且高质量的测试执行过程是一个立体作战的过程，从有计划的测试到 Ad-hoc 测试，从每日构建验证到回归测试，无不孕育着智慧和策略。

有效的测试计划是指导测试用例设计，以及测试执行的纲要性文件，是测试项目成功的前提和必要条件，而测试用例则是测试的基础，测试用例的设计是测试的核心工作之一，可以说，测试用例的成功设计已经完成了一半的测试任务。但是，测试的执行也是测试计划贯彻实施的保证，是测试用例实现的必然过程，严格地测试执行使测试工作不会半途而废。

在整个测试执行阶段中，我们需要面对一系列问题，如：

- 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和要达到的目标？

- 如何保证测试用例得到百分之百的执行？
- 如何保证所报告的软件缺陷正确、描述清楚、没有漏掉信息？
- 如何在验证 Bug 或新功能与回归测试之间寻找平衡？
- 如何跟踪 Bug 处理的进度使严重的 Bug 及时得到解决？

下面，我们就逐一回答这些问题，帮助大家顺利完成功能测试的执行。

7.1 测试执行概述

当测试计划、测试用例和测试脚本都已经完成时，我们就要开始执行测试了。这时候，可以说，万事具备，只欠东风，即需要准备好测试环境和安排好执行测试的人员。所以，在测试执行前，应做好如下两项准备工作。

（1）测试环境的准备。构建测试运行的平台和安装需要的软硬件系统。例如，Yahoo 日历测试环境中就要安装操作系统 Linux、Web 服务器 Apache 和 WebLogic、数据库 Oracle 和自身应用软件包等。

（2）人员的安排。不仅包括指定哪些人参加功能测试，哪些人参加系统测试和谁负责测试环境的维护等，还要包括人员的培训，知识的传递。

测试的执行过程，也不是想象的那样简单——按照测试计划、测试用例执行就可以了。一个有效且高质量的测试执行过程是一个立体作战的过程，会有许多并发的工作同时进行，包括功能测试、系统测试、每日构建（Daily Build）验证、Ad-hoc 测试、每日缺陷验证和回归测试等，它们构成了多个层次并行的测试空间，如图 7-1 所示。

在这里，特别要说一下功能测试过程、每日构建验证、Ad-hoc 测试和每日缺陷验证，使大家对测试执行的立体作战过程有更深刻的理解。

（1）功能测试的安排要从缺陷产生的概率、修正的难度来考虑。缺陷产生的概率越高，或者开发人员修正某类缺陷的难度越大，其测试的执行就越要靠前。所以，测试执行的原则有：

- 先安排新功能的测试，后安排完整的功能（包括可能会受影响的原有的正常功能）测试。
- 先进行某一两个平台的测试（即图中的快速测试），后进行更广泛的环境、平台组合上的测试。

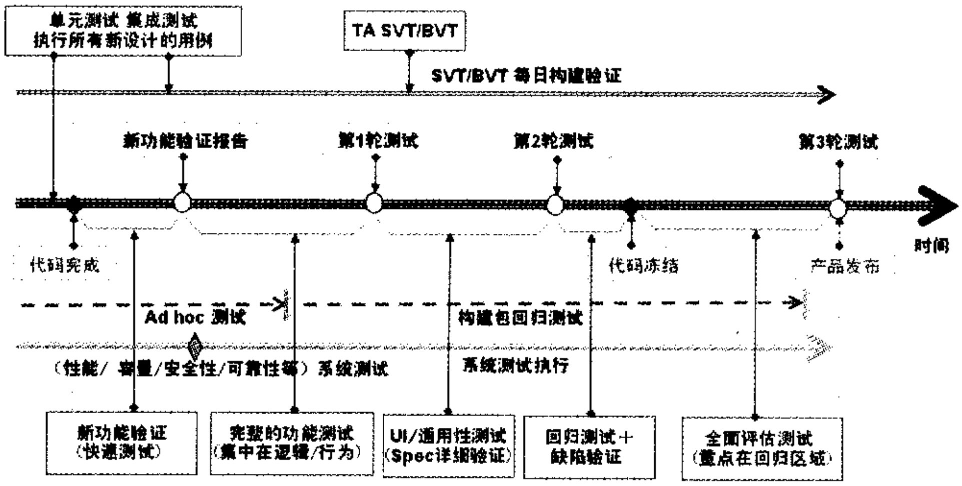


图 7-1 测试执行的立体作战过程

- 先安排功能的逻辑和行为方面的测试，后安排界面测试。
- 在代码冻结前，要为功能测试留下一个特定的时间段，用于最后的缺陷验证及其回归测试。

（2）每日构建验证是在不影响功能测试整体计划的前提下进行的，它能完成对每日构建的软件包进行基本功能的验证，避免由于新产生或修改的代码引起一些非常严重的缺陷，特别是在模块或组件接口方面产生的问题，以保证持续集成的有效性。针对这种验证，事先开发了测试脚本，其执行由测试工具自动完成，这就是著名的 BVT/SVT（Build Verification Test/Sanity Verification Test）。

（3）Ad-hoc 测试，指进行一些非计划内的、随机的、猜测性的测试，目的是给测试人员一个自由的想象空间，可以让测试人员不受约束地展开思维，随意的猜测，以发现隐藏更深的缺陷或偏僻的缺陷，即补充一些边界和特殊情况（Corner case），进一步完善测试用例，而且也借此机会更好地熟悉产品。

（4）每日缺陷验证。发现了缺陷，就需要及时修正这些缺陷，然后让测试人员拿到新的软件包之后再进行验证。由于是每日构建软件包，所以每天都会有一些已修正的缺陷验证任务。建议每天先验证已修正的软件缺陷，然后处理计划好的测试任务。

概念

代码完成。开发人员完成了规定的设计和编程任务，即实现了新功能和功能增强等所需要的代码，并完成了单元测试，则宣布代码完成，这时可以全面执行功能测试和系统测试等。

代码冻结。所有计划的测试任务完成，已发现的严重缺陷得到了修正和验证，代码质量达到了事先定义的标准，这时代码实施冻结，所构建的新软件包将作为软件发布的候选。虽然代码已被冻结，但并不是一点代码都不能改。如果发现了严重的缺陷，还是可以改，但会实施非常严格的变更控制流程，将从批准、代码检查、代码检入等各个方面控制。代码冻结之后，还要全面执行一轮测试，以全面评估软件产品的质量。

每日构建。只要代码有变动和修改，每天都要重新构建完整软件包（Build）。每日构建是持续集成、持续测试的主要质量保证措施之一。

7.2 测试执行的准备

如前面所述，在测试执行前需要做些准备，包括环境建立、人员培训等，正所谓磨刀不误砍柴工，做好了测试执行的准备工作，实际的执行过程就会顺利，执行的效率和质量都会有保证。

7.2.1 培训和知识传递

因为功能测试的执行对人员要求相对低一些，所以某些情况下，测试用例设计人员和测试脚本开发人员并不是后阶段测试执行的人员。由资深的或富有经验的测试工程师负责测试用例设计，由具有较好编程经验的工程师负责测试脚本的开发，而由经验相对少些或能力弱些的工程师来完成具体的测试执行工作。

对于功能测试，一般难以到达较高的测试自动化程度。如果测试自动化程度不高，测试执行也必定要求更多的人员参加。

因此，有一部分将要参与测试的执行人员，不是从一开始就参加了项目的需求分析和设计的审查，以及测试用例设计，而是在测试用例设计快结束、代码完成之前的某个时间进入该项目的。这部分测试人员对项目背景不够了解，对产品特性不够熟悉，对测试策略和测试用例设计的要点还不清楚，这就要求对测试人员进行相关的培训和产品知识的传

递，甚至包括召开测试执行前的动员会议，以便给测试执行以积极的影响，所以这会显得非常重要。

在实际操作中，可以将培训和知识传递两者的工作结合起来，通过事先分发和阅读文档，然后请前期参与项目的测试工程师、设计和编程人员进行讲解等来达到目标。需求文档、产品规格说明书、设计的技术文档、测试计划和用例等的评审和复审起着一箭双雕的作用，既是质量保证的一种措施，也是一种知识传递的方式，可加强测试执行人员对客户需求的理解，以及对业务领域的认识。

保证知识传递的有效性、及时性、正确性和完整性是必要的。知识在传递过程中失真越早，在后继的过程中这个知识的失真就会放大得越厉害，经过的环节越多失真也越厉害。所以培训和知识传递要尽早开始，要尽可能获得第一手资料。从这一点上看，应该让所有的测试人员尽早参与项目的各种活动。即使有一部分测试人员，由于在项目前期还在其他项目上工作，不能参与测试用例设计、脚本开发等工作，但也要让他们参与一些项目介绍会、技术讨论会和测试计划/用例评审会。参与这些会议占用的时间不多，又能获得第一手信息。所以在实际工作中，我们常常有计划、有意识地安排这样的活动，尽量创造条件为大家及时提供必要的项目知识。

知识传递和转换的主体是人，知识传递的重点就是做好人的工作，即在组织过程管理上加强这一个环节，包括团队文化的建设，员工的教育和培训等。如：

- 创造愉快、活跃的团队关系，可以促进充分的、有效的知识传递。
- 对团队适时、定期的培训是保证知识传递的及时性和正确性的常用手段。
- 对新进的员工进行足够的培训，并为每个新人配一个资深的工程师辅导或帮助，建立和实施师傅带徒弟、伙伴关系（Buddy System）等。

另外，建立良好的反馈机制、文档管理系统、知识库（knowledge base）和论坛等，都会有助于知识的共享和传递。

7.2.2 测试任务安排

小项目测试任务安排比较简单，而对于大项目，测试任务的安排相对要复杂些，不得不考虑下列的问题或困难。

（1）针对工程师个人的特点和特长来安排适合工程师的特定任务。例如，某些部分（像日历的选项等）简单但需要逐项测试，测试工作量大，就安排一个平时工作非常细心的工程师来负责测试。而对于接口测试，需要安排技术强、思维开阔的工程师负责测试。

（2）不同的阶段可以适当交叉互换测试人员，既能发挥每个人的不同思维空间，提高测试覆盖率，又能起着相互检查、督促的作用，更好地保证测试的质量。如在第一轮测试中，测试人员甲负责日历显示测试，测试人员乙负责输入部分；在第二轮，则可以让测试人员甲与乙的工作实行交换。

（3）任务安排均匀、公平，不要造成一部分人的任务过重，一部分人的任务过轻。这要求分配任务的测试组长非常了解被测试的各部分，对各个模块的测试难度和对应的测试用例数了解得一清二楚。另外一种方法是，为每个测试用例设定估计时间，有了这样的数据，就很容易获得测试的工作量。

（4）将关联性很强的若干个（子）任务安排给一个人。例如，日历选项和日历显示关联性很强，不宜将它们分开交给不同的人测试。如果那样，容易忽视相互之间的影响（集成关系），存在测试的漏洞。

（5）任务不能安排太紧，适当留有余地。任务安排很紧，测试执行会被打折扣，或者测试中的一些细节、疑点会被忽略，测试人员也会缺少思考，不会举一反三。

任务的安排，在测试计划中已经得到较多的考虑，但肯定还是不够或不准确的。随着测试用例设计工作的不断深入，测试用例的数量及其测试执行的工作量越来越清楚，测试资源的安排会作相应的调整。在工作中，建议按照下列步骤去安排测试任务：

（1）在做测试计划时，对测试执行所需要的资源进行初步规划，一般会增加比较多的余量（15%～20%），使测试资源有足够的准备。

（2）在设计测试用例时，就预估每个测试用例执行所需的时间，并记录在测试用例数据库中，为后期估计备用。

（3）根据每个测试用例的预估时间，可以算出每个测试模块的工作量。

（4）分析软件模块之间的关系，然后根据模块的关联性和相应的工作量进行模块组合。

（5）根据每个人的特点，将组合的模块分配给各个测试人员。

（6）一轮测试结束后，交叉互换测试的模块组合。

7.2.3 测试环境的建立

在测试执行过程中，测试环境的建立和维护是主要工作之一。当然，更理想的是在测试计划过程中，就有充分考虑和进行良好的设计，使测试环境能在更早阶段准备好，如在

设计验证阶段或测试用例设计阶段。

测试环境包括被测试的软件系统环境，还包括执行测试用例的自动化环境或测试服务器的客户端环境。为了确定测试环境的组成，需要明确以下问题：

- 所需要的计算机数量，以及对每台计算机的硬件配置要求，包括 CPU 的速度、内存和硬盘的容量、网卡所支持的速度、打印机的型号等。
- 测试中需要使用的网络环境。例如，Web 应用的测试结果同 Internet 接入速率相关，那么应考虑为测试环境租用某些特定速率（ISDN、ADSL 或 T1）的线路，或者通过软件工具（如 DUSuperControl）的方法，限定网络传输的速率。
- 部署被测应用的服务器所必需的操作系统、数据库管理系统、中间件、Web 服务器以及其他所需的组件等名称、版本，包括使用许可证（License）的数量。
- 作为测试的客户端所需的操作系统、应用软件、测试工具等的名称及版本，包括使用许可证（License）的数量。
- 用来保存各种测试工作中生成的文档和数据的文件服务器及网络驱动器、路径、系统等。
- 是否需要专门的计算机用于被测的应用服务器环境和测试管理服务器的环境及数据备份；
- 为了执行测试用例需要初始化的各项数据，例如登录被测应用所需的用户名和访问权限，或者其他基础资料、业务数据。对于性能测试，还应当特别考虑执行测试场景前应当满足的历史数据量。当然，还有另外一个非常关键的问题：在测试过程中受到影响的数据如何恢复？

根据上述内容，可以为雅虎日历的测试环境建立起全面的检查列表（Checklist），如表 7-1 所示。

表 7-1 测试环境检查列表

测试环境需求	说 明	是否已准备
硬件：		√
计算机 10 台（CPU 1^{+}GB，内存 512^{+}MB，硬盘 40^{+}GB，网卡）	测试用的客户机	√
服务器 4 台（Dual CPU 2.8GB，内存 2^{+}GB）	Web 服务器、管理服务器各两台，构造备份、故障转移机制	√

续表

测试环境需求	说 明	是否已准备
服务器 4 台（Dual CPU 2.8GB，内存 8⁺GB）	中间件服务器、数据库服务器各两台，构造备份、故障转移机制。	√
互连网接入（10MB/100MB）	连接入 Internet	√
软件：		√
Linux 操作系统	安装于被测软件系统的服务器端	√
Appach Server		√
TomCat Server		√
MySQL DB		√
Windows XP OS	安装于测试用的客户机端（支持不同的操作系统）	√
Windows 2000 Professional OS		√
Vista		√
Windows 2003 OS		√
IE6.0, IE 7.0	安装于用的客户机端（支持不同的浏览器）	√
FireFox 1.5~FireFox2.0		√
Nestscape 6.0~ Nestscape 8.0		√
Mozilla 1.5~ Mozilla 1.7		√
JMeter	性能测试工具	√
Selenium	自动化测试工具	√

7.2.4 测试环境的设置

没有测试环境就无法执行测试用例，测试环境是测试执行的保证。如果测试环境设置不对，所发现的缺陷可能不是缺陷，同时，又可能会漏掉一些存在的缺陷。测试环境不对，测试结果就可能不对，测试工作就失去了价值。

1. 测试环境的正确性

测试环境不仅包括硬件，还包括软件；不仅包括客户端、服务器，还包括网络环境、

测试数据等。测试环境在硬件上可能也有一些特定要求，例如选用特定的显示卡、声卡等，但在测试环境上出错的地方往往是在网络设置、软件环境配置等方面。在 Web 测试上，包括 DNS、SSL 协议、负载均衡器、防火墙、Apache/WebLogic 等的设置，还包括虚拟 IP、虚拟目录、网络文件系统等。例如，一般要求被测试的服务器必须配置 DNS，而不能直接使用 IP 地址来访问服务器以进行测试。

2. 测试环境的可靠性

测试环境的正确性是最基本的要求，要提高测试效率，实施测试自动化，要给测试环境提供更高的要求。例如，对于一个测试项目，要将性能测试环境和功能测试环境分开，因为在进行性能测试时，对功能环境影响较大。即使进行功能测试，最好也建立两套同样环境，这样可以提高测试效率：

- 每天要更新构建的软件包，如果只有一套环境，在系统安装、配置时，该项目的所有测试人员不能进行测试会耽误大量的时间。更糟糕的时候，新构建的软件包安装完之后，由于配置改变和存在严重的回归缺陷，导致不能正常执行测试，也会耽误很长时间。如果有一个备份的测试环境，这样可以交替更新软件包，总有一个环境是处在正常的情况，供大家测试用，即今天更新环境 A，明天更新环境 B，后天更新环境 A，……无论出现什么情况，都能保持测试正常的执行。
- 当在新构建的软件包上发现某个严重缺陷后，可以在上一个软件包上进行验证，以确认是否为回归缺陷。如果有两套软件测试环境，并按上面的交替方式更新，就非常容易进行两个软件包的比较，以验证是否为回归缺陷。从代码或缺陷修正的质量看，回归缺陷应该越少越好。所以，回归缺陷率往往作为考察开发人员绩效的指标之一。

3. 测试环境的多样性和复杂性

在企业内部的计算机应用系统，可以要求所有客户端使用某种特定的操作系统、浏览器等，即客户端测试平台只有一种。但在互联网上的软件应用，情况就完全不一样了，世界各地的用户所使用的环境是各种各样的。软件服务商也无法规定用户的环境，如果其软件系统只支持某种操作系统、某种浏览器，则会失去大批用户，失去市场。

对于互联网上的软件应用，应尽量满足客户的环境需求，所以客户端应用的测试环境具有多样性，涉及操作系统、浏览器、代理服务器/防火墙等。

示例

环境的多样性示例

- 操作系统：Windows Vista、Windows XP Home、Windows XP professional、Windows 2000 pro、Linux、Solaris 9、Solaris 10、Solaris x86、Mac OS 9、Mac OS X、IBM AIX、HP-UX。
- 浏览器：IE 6.0、IE 7.0、FireFox 1.5、FireFox 2.0、Mozilla1.7、Safari 1.3、Safari 2.0、NS7、NS 8.1。
- 代理服务器/防火墙：ISA 2000、ISA 2004、Blue Coast、Cisco PIX、Linux squid、Checkpoint、……
- 防火墙验证方式：无口令、口令，Script、……
- 传输协议：TCP、HTTP、SSL。

除此之外，如果测试的对象是应用系统的服务器端软件平台，可能还涉及产品的客户端不同的版本，第3方产品的集成，以及本地化语言版本等兼容性测试，测试的环境进一步扩展，需综合考虑集成的测试环境。

示例

集成的测试环境的示例

- 客户端不同的版本：1.0、2.0、3.0 <用户可能没升级，如 MSN、Yahoo IM 等客户端不同版本是同时并存的> 。
- 第3方产品集成，例如，与 OutLook/Lotus Notes 的邮件/日历的集成，以 Lotus Notes 为例，就可能包括 Notes 6.5.2、Notes 6.5.3、Notes 6.5.4、Notes 6.5.5、Notes 7.0、Notes 7.0.1。
- 本地化语言：英文（EL）、中文简体（GB）/中文繁体（B5）、日文（JP）、德文（GE）、……

概念

回归缺陷。在前一个版本（Release）或前一个软件包（Build）上不存在的缺陷，在当前（新）的版本或当前软件包上存在的缺陷，称回归缺陷。根据参照对象（版本、软件包），回归缺陷可以分为版本回归缺陷和软件包回归缺陷。

7.2.5 测试自动化运行平台

测试自动化对环境要求更高——高稳定性、配备齐全，满足无人职守的情况下执行测试任务等。理想的自动化测试情景是这样的：

- 能够把大量测试个案分配到不同的测试机上去同时运行。
- 可以让某台服务器管理测试的机器，调度测试任务，即可以根据机器空闲状态，能及时将任务安排在空闲的测试机上。
- 可以在某个测试环境上，运行不同的测试工具，而且是并行、协同地完成同一个测试任务。
- 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。如在下班前将所有要运行的测试脚本准备好，预先安排好，并由系统定时自动启动测试工具和运行测试脚本。如在晚上 10 点以后开始执行测试任务，第二天一上班（早上）就能拿到测试结果。

要做到这些，就需要构造一个稳定的测试环境和一个灵活的、自动化程度很高的测试工具运行平台（框架），如图 7-2 所示。这里，对图中所描述的自动化测试运行平台作一个简单的说明：

（1）控制器是这个环境的核心，负责测试的执行、调度，从服务器读取测试用例，向测试环境中的代理（Agent）发布命令，获取代理的反馈，直至借助代理跟踪测试机的状态。代理是驻留在每台测试机上独立的、非常小的监控程序。

（2）机器池（Machine Pool）是由一组测试用的服务器或 PC 机器构成的，负责测试任务的执行。每台机器运行代理程序，可以接受控制器发来的指令，启动测试工具，执行任务，而且可以将机器状态（忙、闲）反馈给控制器，有益于硬件资源调度。

（3）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，有利于开发团体的任何人员方便地查询测试结果，也方便测试人员在自己的办公室预先安排任务、提交任务和运行测试。

（4）文件服务器存放待测试的软件包和测试工具、测试脚本，并能实现向机器池自动部署新的软件包，并使测试工具可以存取脚本。

（5）管理服务器负责测试用例和测试套件的创建、编辑和其他维护工作。

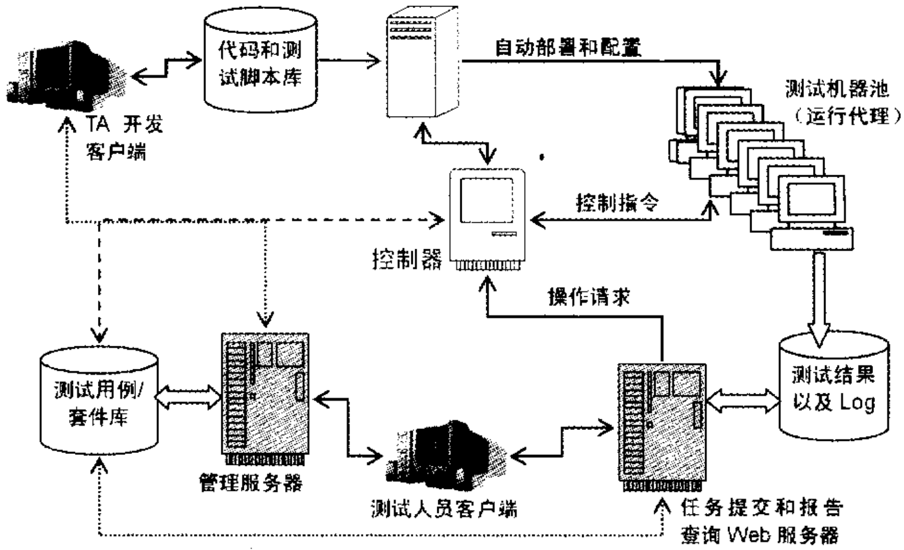


图 7-2 测试自动化运行平台

7.3 如何有效地创建测试套件

功能测试是最基本的测试，功能测试是系统测试的基础。虽然功能测试和系统测试可以并行，但是如果功能测试中发现过多的缺陷，系统测试就无法执行，系统测试结果也就不会准确。在修正了大量缺陷之后，还需要重新做系统测试，来获得性能指标。

在实施测试时，测试用例作为测试的标准，测试人员一定要按照测试用例的前提、测试数据、测试步骤等完成具体的测试，并在测试用例管理软件中记录测试结果。在实践中，测试数据是与测试用例分离的，按照测试用例准备一组或若干组供测试用的原始数据，以及标准测试结果。尤其测试像报表之类数据集的正确性，按照测试用例规划，准备测试数据是十分必须的。除正常数据之外，有时还要根据测试用例设计大量边缘数据和错误数据。

功能测试执行过程中，如图 7-1 所示，测试要经过几轮不同的测试执行的子阶段：

- 新功能的快速测试。
- 完整的功能性测试，并集中在逻辑性、行为方面的测试。
- 界面、适用性测试。
- Ad-hoc 测试和回归测试。

为了有效地执行这些子过程，要建立与之对应的测试套件（Test Suite）。

7.3.1 功能测试套件的创建

测试执行的任务，由测试套件来体现出来。借助测试套件更好地管理被执行的测试用例，更有效地执行测试任务；借助测试套件，也比较容易完成测试审查、跟踪的工作。创建测试套件，则主要考虑以下几个方面：

- 所选择的测试用例，需要涵盖所要测试的功能点，这些功能点包括新增功能、增强功能和受代码改动影响的原有功能。
- 要包含所有必要的、不同的测试环境。
- 应将所选择的测试用例和测试环境统一起来考虑，因为不同的测试用例或模块对环境的敏感度是不一样的，也就是说，环境对不同的测试用例影响是不一样的。
- 测试套件的组织合理，结构清楚，规模适中。
- 测试套件的复用性。建一个有效的测试套件，可以用较多的时间，以后可以被多次使用。

1. 依据功能点的测试用例组合

功能点的考虑比较自然，从模块开始，直至单个的测试用例。对于模块，确定哪些功能模块在这个版本中被改动；对于改动的模块，进一步分析，哪些测试用例是新加的，哪些测试用例做了修改。然后，再分析哪些模块可能受到影响，虽然它们没有直接被修改。基于这些分析，可以按下列一些原则来组合测试用例。

- 所有新增加的、最近修改的测试用例会被优先选择。
- 对于改动大的模块，其所有的测试用例都会被优先选择。
- 改动小的模块、受影响大的模块，大部分的测试用例会被选择。
- 受影响较小的模块，少量的测试用例会被选择。
- 不受影响的模块，不需要选择这部分测试用例。
- 在最后一轮测试，根据资源和时间来决定，一般选用较多的测试用例完成全面质量评估的测试。

历史的数据也能有效地帮助测试用例的选择和组合。例如，发现缺陷的测试用例比从来没有发现缺陷的测试用例更有价值，应被优先选择到测试套件中。当然，像这种选择方法，要借助缺陷和测试用例的管理系统来实现，记录相关数据，并建立起缺陷和测试用例的关联关系。然后设定过滤条件，自动生成相关的测试套件。

2. 测试用例的环境组合

测试用例必然要运行在一定的环境上，对于客户端软件（如 Google Talk），着重考虑操作系统、网络协议、通讯端口和防火墙的影响；而对于 Web 应用（如雅虎日历），则主要考虑浏览器、SSL 协议、API 等的影响，而操作系统对 Web 应用影响较小或几乎没有影响。在建立测试套件时，充分考虑它们各自的特点，尽量选择有效的环境组合。

不同的功能模块，受环境的影响也是不一样的。就是在同一个模块中，不同的测试用例受环境的影响程度可能差异很大。所以，如果分析得越透彻，对测试环境的影响越清楚，最终高效率的测试套件得以被创建。概括起来，理想的测试用例的环境组合，可以从 3 个层次去考虑：

- 系统层次，在测试计划上整体考虑。
- 功能模块或组件层次，测试组长针对测试用例模块去考虑。
- 特定的功能特性（测试用例）层次，每个测试人员针对每个测试用例去考虑。

示例

测试用例在不同层次受环境的影响

Google Talk的下载与浏览器有较多的关联，而其安装需要在不同的操作系统（如Windows/ Mac/ Linux）或同一个操作系统不同版本（如Windows 2000/XP/Vista等）上运行，而且还要针对不同的用户权限（如administrator和normal user）进行安装测试。但其聊天的功能，只要在不同的操作系统上运行，无须在同一个操作系统的不同版本上运行。Google Talk，Gmail邮件、Google日历的集成应用，与浏览器的关联比较大，所以在进行集成测试时，又必须考虑在不同浏览器上进行测试。

7.3.2 测试环境的爆炸性组合

测试用例的环境组合，是一个值得研究的课题。在实际工作中，绝不能简单地叠加起来。试图进行完全组合的测试，这是不可能的。以上节 7.2.4 中介绍的环境因素为例：

- 操作系统有 12 种。
- 浏览器有 9 种。
- 代理服务器/防火墙有 6 种。

- 防火墙验证方式有 3 种。
- 传输协议有 3 类。
- 客户端版本有 3 个。
- 第 3 方产品集成，Lotus Notes 有 6 种。
- 本地化语言 —— 常用的有几十种，这里选 4 种：英文（EN）、中文繁体（B5）、日文（JP）、德文（DE）。

如果用一个完全的组合，测试工作量将非常大。比如产品的功能测试用例为 1000 个，其完全组合数是：

$12 \times 9 \times 6 \times 3 \times 3 \times 3 \times 6 \times 4 = 419\,904$

即近42万个组合，测试用例数从原来的1000个猛增到4亿个。假定每个人日（man-day）完成的测试用例数是 50，原来 1000 个测试用例，需要 20 个人日，即一个人一个月内可以完成。而 4.2 亿个测试用例，则需要 840 万个人日，也就是 3 万多人工作一年。

可以进一步简化，如过滤一些很少用的操作系统，忽略相近的浏览器而尽量选用差异较大的、不同的浏览器等。这样，环境因素的列表可以列为：

- 操作系统。Windows 只选 Windows Vista、Windows XP 和 Windows 2000，其他系统（Mac、Solaris、Linux 等）各选一种，共 6 种。
- 浏览器，各选一种，即 IE 7.0、FireFox 2.0、Safari 2.0、NS 8.1，共 4 种。
- 代理服务器/防火墙，选 ISA 2004、Blue Coast 和 Cisco PIX，共 3 种。
- 防火墙验证方式选 2 种，口令方式和脚本命令方式。
- 传输协议，只选 HTTP+SSL 这一种。
- 客户端 3 个版本不变。
- 第 3 方产品集成，选 Lotus Notes 6.5.3/6..5.5/7.0.1，共 3 种。
- 本地化语言，只选 2 种，英文（单字节的代表）、日文（多字节的代表）。

则环境组合减少到

$6 \times 4 \times 3 \times 2 \times 1 \times 3 \times 3 \times 2 = 2592$

组合有很大的降低，降到原来（42 万）的近二百分之一，但工作量依然很大，需要 210 人工作一年。但同时会带来比较大的风险，因为有许多环境，在上面没有进行任何测试。所以，到这里，问题并没有结束。

7.3.3 环境组合优化

软件测试的目的就是发现缺陷，从理论上看，如果试图发现100%的缺陷，就要对各种组合进行测试。而对应用软件的测试目标没有必要设在100%，也不可能达到100%。可以设定测试目标——通过一个优化组合去发现99.5%～99.9%以上的缺陷，而且产品的所有主要特性得到验证就足够了。

这时，可以引进正交试验方法的思想来帮助环境组合的优化。首先，考察所有这些环境因素（操作系统、浏览器、代理服务器/防火墙、传输协议、客户端版本、第3方产品集成和本地化语言）之间的关系，然后根据因素之间的关联强度和对测试结果的影响程度，选定不同水平的正交表进行试验，通过分析找出对结果有影响的组合。

为了使问题简单些，我们可以将因素分类，先在各个类别进行组合分析，然后再进行大类别的正交组合设计，使整个测试套件的设计策略得到最优实现。

1. 测试用例的分类

为了更准确地分析组合并降低测试风险，可以从两个角度将测试用例进行分类：用例的优先级和用例的目标域。

用例按优先级（Priority）可以分为3类：

- Cp_1——最高级，这些测试用例所发现的缺陷是致命的，必须在当前版本修正。
- Cp_2——中级，这些测试用例所发现的缺陷是较严重的，必须在下一个版本前修正。
- Cp_3——最低级，这些测试用例所发现的缺陷是一般的，有时间最好修正。

用例按目标域（Objectives）可以分为3类：

- C_{ui}——测试客户端功能界面的用例。
- C_{lg}——测试客户端功能逻辑的用例。
- C_{if}——测试客户端集成（接口）的用例。

当然，针对产品的不同特点/规模等，可以进行不同的、更多层次的分类。

2. 环境因素的关联性

这里着重考察来自测试结果的影响，就是在测试中发现缺陷的可能性——每一种组合可以或可能发现缺陷的概率，也就是产品的特性对任一种组合的敏感度。我们再将“试验”的概念扩展一下，可以把以前测试所积累下来的历史数据看作是这种试验的可用结果之一，根据历史积累的数据，获得如表7-2所示的环境因素关联矩阵。顺便提一下，这里考

虑的是功能特性，对于非功能特性（如性能、安全性），有特定的、不同的测试用例，而且影响因素也是不一样的。

表 7-2 操作系统和浏览器的正交性关联矩阵

OS\Browser	IE7.0	IE6.0 SP1	Firefox 2.0	Firefox 1.5	Mozilla 1.4	Mozilla 1.7.13	NS 8.1	Safari 1.3	Safari 2.0
Windows									
Windows Vista	Y	Y	Y			Y	Y		
Windows XP SP2	Y	Y	Y	Y		Y	Y		
Windows 2000 SP4		Y		Y		Y	Y		
Windows 2003 R2		Y		Y		Y	·		
Mac									
Mac OS X（10.3）			Y	Y	Y		Y	Y	
Mac OS X 10.4（PowerPC）			Y	Y			Y	Y	Y
Mac OS X 10.5（Intel）			Y	Y			Y		Y
Solaris									
Solaris 9			Y	Y	Y	Y			
Solaris 10			Y	Y		Y			
Linux									
Red Hat Enterprise			Y	Y	Y	Y	Y		
SuSE Linux 10			Y	Y	Y	Y	Y		

3. 考虑市场因素，优化环境因素关联矩阵

其次可以根据操作系统和浏览器在市场的份额和所发现的缺陷等数据来优化上面的

组合关系表。根据市场数据，假定操作系统和浏览器在市场的份额（不是真实的数据，仅供参考），如表 7-3 所示。

表 7-3 操作系统和浏览器的市场份额

Windows Vista	5%	IE 7.0	5%
Windows XP SP2	60%	IE 6.0 SP1	55%
Windows 2000 SP4 Update Rollup 1	20%	Firefox 2.0	5%
Windows 2003 Server R2	1%	Firefox 1.5	15%
Mac OS X Panther （10.3）	2%	Mozilla 1.4	3%
Mac OS X 10.4 （Power PC）	2%	Mozilla 1.7.13	6%
Mac OS X 10.4.6 （Intel native）	1%	NS 8.1	5%
Solaris 9	1%	Safari 1.3	2%
Solaris 10	2%	Safari 2.0	4%
Red Hat Enterprise Linux 4	3%		
SuSE Linux 10	8%		

可以得到新的、量化的正交表，如表 7-4 所示。可以消除所有低于 0.2%的组合，如 Windows 2003 和 Firefox 1.5 的组合，结果只有 0.15%，则不予考虑。

表 7-4 操作系统和浏览器的已优化的关联矩阵

OS\Browser		IE7.0	IE6.0 SP1	Firefox 2.0	Firefox 1.5	Mozilla 1.4	Mozilla 1.7.13	NS 8.1	Safari 1.3	Safari 2.0
Windows		5%	55%	5%	15%	3%	6%	5%	2%	4%
Windows Vista	5%	0.25%	2.75%	0.25%			0.30%	0.25%		
Windows XP SP2	60%	3.00%	33.00%	3.00%	9.00%		3.60%	3.00%		
Windows 2000 SP4	20%		11.00%		3.00%		1.20%	1.00%		
Windows 2003 R2	1%		0.55%		~~0.15%~~		~~0.06%~~			
Mac										
Mac OS X （10.3）	2%			~~0.09%~~	0.26%	~~0.05%~~		~~0.09%~~	~~0.03%~~	

续表

OS\Browser		IE7.0	IE6.0 SP1	Firefox 2.0	Firefox 1.5	Mozilla 1.4	Mozilla 1.7.13	NS 8.1	Safari 1.3	Safari 2.0
Mac OS X 10.4 (PowerPC)	2%			~~0.10%~~	0.30%			~~0.10%~~	~~0.04%~~	~~0.08%~~
Mac OS X 10.5 (Intel)	1%			~~0.06%~~	~~0.18%~~			~~0.06%~~		~~0.05%~~
Solaris										
Solaris 9	1%			~~0.05%~~	~~0.15%~~	~~0.03%~~	~~0.06%~~			
Solaris 10	2%			~~0.10%~~	0.30%		~~0.12%~~			
Linux										
Red Hat Enterprise Linux 4	3%			~~0.15%~~	0.45%	~~0.09%~~	~~0.18%~~	~~0.15%~~		
SuSE Linux 10	8%			0.40%	1.20%	0.24%	0.48%	0.40%		

4. 考虑各种产品特性及其分类，进一步优化

根据产品相应的历史缺陷数据（所占比率——百分比值），可以知道测试用例目标域各个分类（C_{ui}、C_{lg}、C_{if}等）在各个平台的正交数据，并可以进一步优化组合。

对于客户端版本的构成，可以以 1.0 版本为基线，把 2.0 版本看作“1.0 版＋新增功能＋修改/影响区域”，所以对于 2.0 版本的测试集中在“新增功能”、“修改/影响区域”。对于 3.0，同样类推。

对于第 3 方产品、本地化语言进行分析，可以和客户端软件的“1.0 版本、2.0 新增功能、2.0 修改/影响区域、3.0 新增功能、3.0 修改/影响区域”等 5 项因素构成正交表。

5. 总结

根据上述 4 个方面的分析，可以构成优化的组合。为了更有效地提高测试效率，还可以将测试组合分解到多个阶段（如新功能验证阶段、第 1 回归测试阶段、第 2 回归测试阶段），例如：

- 新功能验证阶段：

英文 + 3.0 新增功能 + 协议/防火墙强组合 + 操作系统和浏览器强组合

英文 + 3.0 修改/影响区域 + 协议/防火墙中组合 + 操作系统和浏览器中组合

- 第 1 轮回归测试阶段：

中文 + 2.0 新增功能 + 协议/防火墙强组合 + 操作系统和浏览器中组合

中文 + 2.0 修改/影响区域 + 协议/防火墙中组合 + 操作系统和浏览器强组合

- 第 2 轮回归测试阶段：

德文 + 1.0（基本功能） + 协议/防火墙弱组合 + 操作系统和浏览器弱组合

组合强弱，可以按照缺陷数概率和把本身（协议/防火墙或操作系统和浏览器）组合概率相乘获得的值来决定，值越高，说明其组合强度越高，应该优先得到测试。

7.4 功能测试自动化的执行

执行自动化测试前，需要根据配置清单先准备好测试环境，然后在测试机器上运行相应的测试脚本。最后根据测试报告来分析和复验产品的缺陷。

对于可以自动执行的测试任务，通常会在上班时间将任务定制好，下班后由系统自动启动任务执行，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，第二天却发现才执行了 5 分钟，就因为程序的一些异常错误而终止了，而且这种情况经常发生。因此，测试工应具有相应的容错处理特性，可以自动处理一些异常情况并对系统进行复位，或者允许用户设置跳过某些错误，然后继续执行下面的测试脚本。

自动化执行的另一个方面，就是对结果的分析。如果系统报告的测试用例执行失败，不能直接确定就是软件的缺陷，可能是脚本的问题或脚本运行的问题，需要对每个失败的结果进行具体的分析，以确定问题出自哪里。如果是脚本问题，需要进一步调试，并修改脚本，直至没有问题。如果确定是软件缺陷，就进入缺陷报告的流程。

许多测试工具都可以监控测试的执行过程，并自动生成测试过程的 Log 和测试结果报告，有些工具提供接口，由用户自己来处理测试结果。图 7-3 就是用 Rational Robot 进行 Google Talk 测试，TestManager 监控测试执行过程的界面。

当一轮测试运行完毕后，Rational Test Manager 会形成一个详细的测试报告。该报告详细地记录了每条脚本测试的结果，包括操作指令执行的情况和所有验证点的验证结果。同时使用不同的颜色来区分测试结果：

- 绿色表示测试通过。
- 黄色表示警告，测试通过，但需要注意的地方。

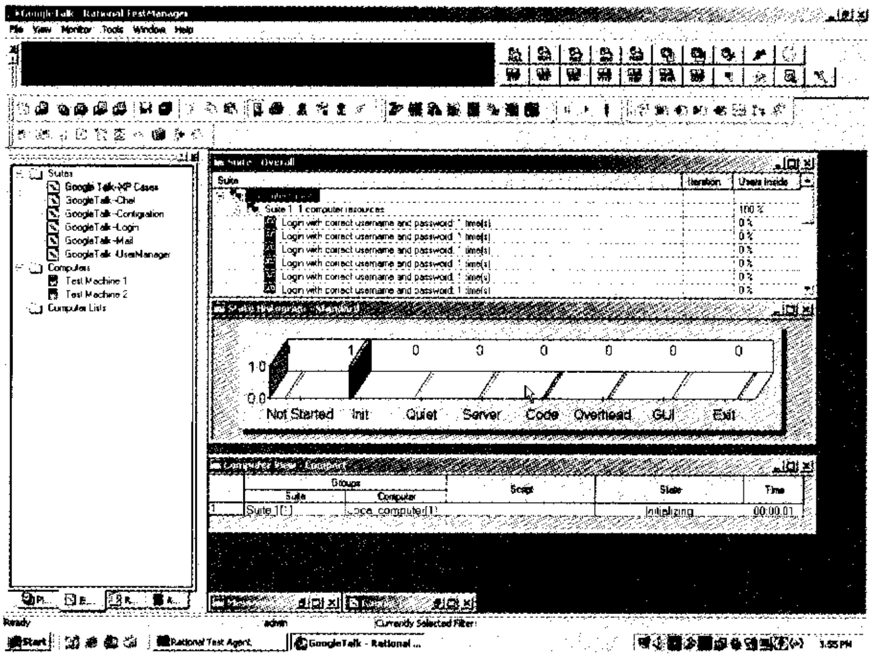


图 7-3 TestManager 监控测试执行过程的界面

- 红色表示测试失败。
- 蓝色表示有指令没有成功执行或发生其他意外。

同时还需收集测试过程中的失败结果，将成功和失败的内容同时列出来，方便分析原因。当测试结果中没有出现红色和蓝色，只有绿色或黄色时，表示所有测试都顺利地通过了。

对于失败的测试用例，可以展开看到更多的细节，包括出错的脚本位置和原因。例如，名为“temp1088UndoRedo annotation-pre”的脚本执行失败了，展开该脚本的报告，发现是一个验证点（Verification Point）失效，如图 7-4 所示。

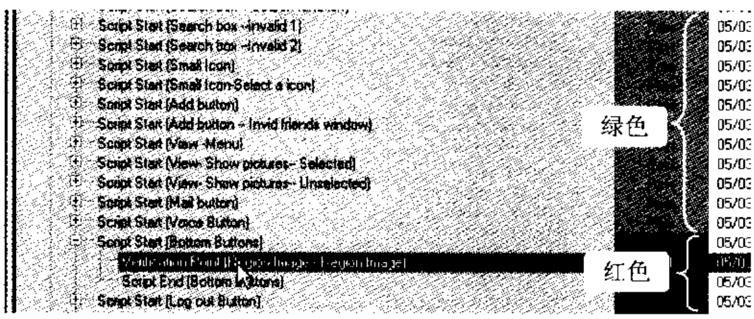


图 7-4 失败的测试结果被展开进行分析

再进一步分析下去，双击该验证点将可以看到如图 7-5 的验证结果。左边是正常的结果，右边是实际采样的结果。机器认为两幅图中红方块处不一样，但我们用肉眼检测认为两边是完全一样的。这是因为 Robot 对于硬件平台的微小差异非常敏感。

图 7-5　预期结果和实际结果的比较（图像验证点）

对于这种情况，可以点击工具栏上的 Replace 按钮，用测试机器上的实际结果作为基准结果。在同样的平台上再次运行时，就可以避免这种失败。

图 7-6 显示的是另外一个例子——编辑框属性值比对失败。检查出失败后，可以手工复验，分析检查一下，确认是否为产品缺陷。

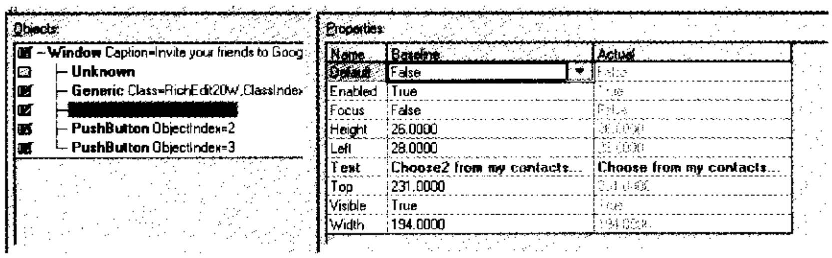


图 7-6　编辑框属性值对比的示例

如果报告中出现蓝色的告警，表示 Robot 出现指令执行失败。展开该脚本的报告，可以找到 Script Command Failure，在右键菜单中选择 Properties，弹出如图 7-7 所示的窗口。

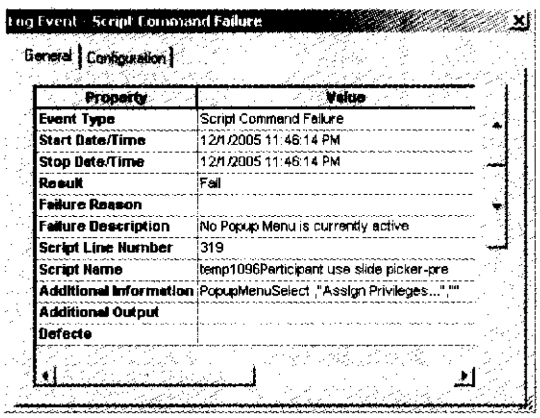


图 7-7　脚本命令出错的详细信息

在 Additional information 一栏中清楚地描述了当 Robot 试图在弹出菜单中选择 Assign Privileges... 项时执行失败了。根据这个重要的信息，可以手工复验一下，确认问题所在。当然还有其他许多种情况，这里仅仅列出了一些常见的结果。分析这些结果时可以结合报告、测试用例及其脚本的逻辑来综合分析。

经验

有时产品的缺陷会导致后续大量的脚本都执行失败，出现大面积连续的红色或蓝色。比如出现一个意外的模式窗口，它会阻断后续所有的操作。如果出现这种结果，重点分析第一条或前几条脚本失败的原因。

7.5 用户界面和适用性测试

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，人们在使用时更加灵活、舒适。软件产品也是一样，始终关注软件使用者——客户的体验，让用户获得赏心悦目的体验，这就依赖于精心设计的软件用户界面（user interface, UI）。这也告诉我们为什么每个产品的开发都需要资深的产品经理和 UI 设计师，而且 UI 设计师发挥越来越大的作用。

简单地说，适用性就是容易发现、容易学习和容易使用。更深地看这个问题，就需要全面评估用户界面，让它包含 7 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，这也是适用性测试所依据的标准。

正确性和一致性比较容易理解。正确性的问题一般都很明显，比较容易发现，例如某个窗口没有被完整显示，文字不对齐，文字拼写错误，密码输入时没有用“*”自动屏蔽等。软件的一致性包括自身的一致性以及软件与其他软件的一致性，如使用的术语、字体是否一致，界面的各元素风格是否前后一致等。下面着重讨论另外 5 个要素。

1. 符合标准和规范

对于现有的软件运行平台（如 Windows、Mac OS 或 Linux），通常其 UI 标准已不知不觉地被确立了，已成为大家的共识。如软件安装界面应有什么样的外观，何时使用复选框，何时选用单选按钮，在什么场合使用恰当的对话框——提示信息、警告信息或者严重警告信息等。如图 7-8 所示，3 种不同提示信息的图标就不一样，提示信息是白色，而警告信息是黄色，严重警告是红色，警醒效果不断升级。

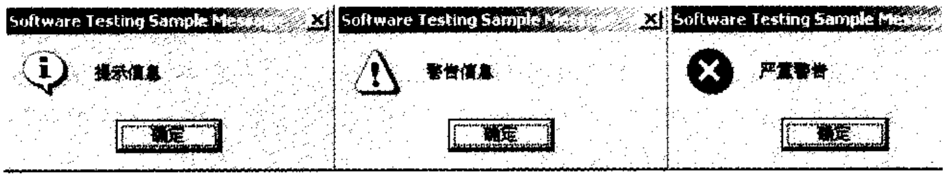


图 7-8 Windows 的 3 种不同的提醒方式

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。如果用“提示信息”代替“严重警告”，很难引起用户的重视，用户随手关闭这个窗口，可能会造成严重的后果。所以，用户界面上各种信息应该符合规范和习惯，否则得不到用户的认可，使得产品不受欢迎。测试人员就把与规范、习惯不一致的问题报告为缺陷。软件所运行的平台标准和规范，可视为和产品规格说明书同样重要，是测试执行的依据。

2. 直观性

用户界面的直观性，要求软件功能特性易懂、清晰，用户界面布局合理，对操作的响应在用户的预期中，如某个对话框在预期出现的地方出现。例如，输入日期采用日历形式，数据统计结果用报表的形式等都很直观。Google 搜索引擎受大家欢迎，不仅因为其搜索速度快，结果准确，而且因为其界面非常洁净，没有多余的功能，非常明显地突出了搜索功能，这是这一个典型的例子。

图 7-9 日期输入界面的直观性

3. 灵活性

软件可以有不同的选项满足不同用户的需求、喜好，用不同的方式来完成相同的功能，会深受用户的欢迎，如 MP3 播放器软件可以设置面板颜色、形状，来迎合年轻人的特征。但灵活性也可能发展为复杂性，太多的状态和方式的选择，不仅增加用户理解和掌握的困

难程度，而且多种状态之间的转换，操作路径的复杂，增加了编程的难度，可能会降低软件的可靠性。计算器程序提供了两种方式（标准型和科学型）以满足不同用户的需求，充分体现了灵活性，如图 7-10 所示。

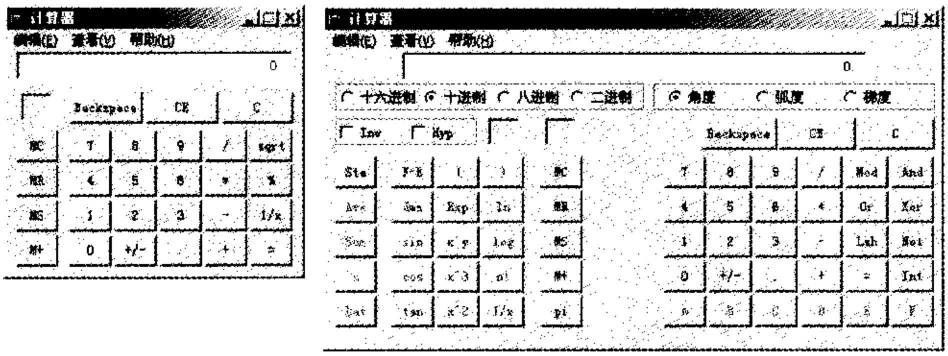


图 7-10 计算器程序的灵活性

4. 舒适性

舒适性主要强调界面友好、美观，如操作过程顺畅，色彩运用恰当，按钮的立体感以及增加动感等。 如操作系统 Windows Vista 在窗口打开、关闭过程中动感很好，许多对象的立体感和色彩表现丰富。图 7-11 中所示的状态信息让用户清楚目前的工作状态，即使多等些时间，心情也不会烦躁。另外，左手鼠标的设置给惯用左手的人带来了便利，也为右手十分劳累时提供了另一种途径。

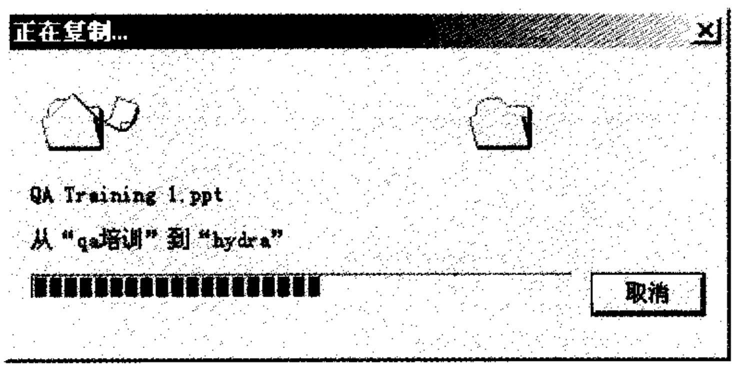


图 7-11 拷贝文件的状态

5. 实用性

实用性不是指软件本身是否实用，而仅仅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是否对软件具有实际价值，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。

大型软件的开发周期很长，经过多次反复地修改、调整，容易产生一些没有实用价值的功能。由于增加某项新的功能，可能导致原先设计的某些功能已没有多大价值。

总而言之，软件适用性测试，除了一些 UI 标准和规范之外，更多需要的是经验和直觉。从上述 7 个要素出发，全面衡量软件的适用性，最终会给用户一个好的体验。另外一种办法就是，临时聘请真正的用户——社会上不同职业的人员来参与这方面的测试，并观察测试过程中的表情、行为，或者倾听用户的意见，获得第一手的反馈信息。例如，微软公司就建立了 20 多个专门为适用性测试的实验室，如图 7-12 所示。

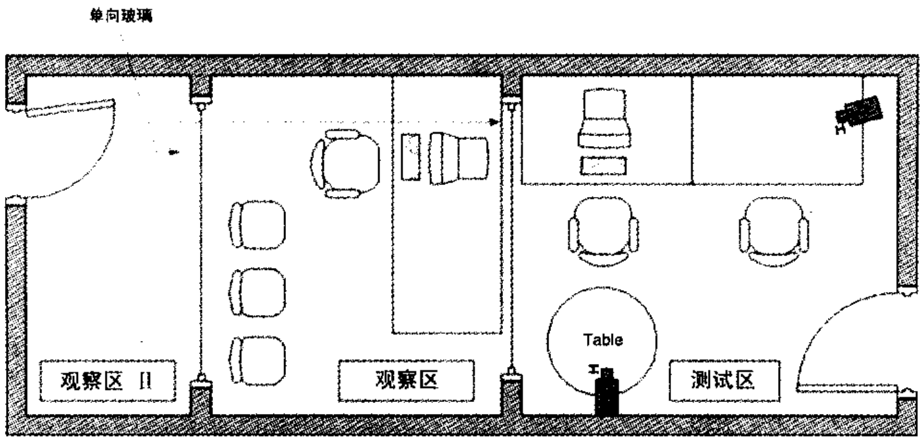


图 7-12 适用性测试实验室的示意图

示例

雅虎日历是一个Web应用程序，所以对页面布局、响应时间等要求较高。在安排用户界面和适用性测试时，注意检查事件浏览各种方法（日/周/月/年）的方便性、直观性和灵活性，检查各种窗口对象及其特征（菜单、大小、位置、状态和中心）是否都符合标准等。可以建立一个如表7-5所示的检查表。

表 7-5

检查项	测试结果或评估意见
窗口切换、移动、改变大小时正常吗？	
各种界面元素的文字正确吗？（如标题、提示等）	

续表

检查项	测试结果或评估意见
各种界面元素的状态正确吗？（如有效、无效、选中等状态）	
各种界面元素支持键盘操作吗?	
各种界面元素支持鼠标操作吗?	
对话框中的缺省焦点正确吗?	
数据项能正确回显吗?	
对于常用的功能，用户能否不必阅读手册就能使用?	
执行有风险的操作时，有“确认”、“放弃”等提示吗?	
操作顺序合理吗?	
按钮排列合理吗?	
导航帮助明确吗?	
提示信息规范吗?	
窗口切换、移动、改变大小时正常吗?	

7.6 回归测试

在软件生命周期中，需要修正已发现的缺陷，或者是增强原有的功能、增加新的功能，这些活动都可能会触及到其他地方的代码，影响正常运行的原有功能，从而导致软件未被修改的部分产生新的问题。因此，每当软件发生变化时，就必须重新测试原来已经通过测试的区域，验证修改的正确性及其影响，即实施回归测试。但是，对于一个长期的软件产品来说，它会拥有相当多的测试用例，这样，由于项目有限的时间和成本的约束，每次回归测试都重新运行所有的测试用例是不切实际的，而且从执行效率看，也是不必要的。所以，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、优化的测试用例套件来完成回归测试。

回归测试的价值在于能够检测到回归缺陷，它是一个受控的测试。当测试组不得不构造缩减的回归测试用例组时，有可能忽略了少数将发现回归缺陷的测试用例，错失了发现回归缺陷的机会。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被删除后不会影响回归测试的结果。

1. 回归测试的方法

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，它们在效率和有效性的侧重点上是不同的。

（1）再测试全部用例。测试用例库中选择全部测试用例构成回归测试包，这是最安全的方法，遗漏回归缺陷的风险也最低，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。但它缺乏策略，而且受预算和进度的限制比较大，不建议使用。

（2）基于风险选择测试。基于一定的风险标准从测试用例库中构造缩减的回归测试用例套件。首先执行关键的、风险系数大的和可疑的测试，跳过那些次要的、例外的测试用例或那些功能相对稳定的模块。执行那些次要用例时，即便发现缺陷，这些缺陷的严重性也较低。

（3）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将反映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。

（4）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。在条件允许时，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的代码分析。

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。

2. 回归测试的组织和实施

（1）通过代码相依分析，识别出软件中被修改的部分和被影响的范围。

（2）从原有测试用例基线库中，排除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，从而建立起新的测试用例基线库 T0。

（3）基于操作剖面和风险选择相结合的策略，从新的测试用例基线库中选择测试用例构造有效的套件，测试被修改的软件。

（4）如果回归测试套件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，生成的新的测试用例集 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。

随着软件缺陷的不断修正，软件修改的范围会越来越小，所以从回归测试的效率来看，所选择的测试用例也越来越少。再说，逐步递减测试用例的过程，也是细化的、深入的过程，在这个过程中不断增加一些新的测试用例，这些测试用例可能是先前遗失的、未曾设计的。回归测试的过程，一方面，测试用例随测试范围的缩小而减少，另一方面，随着测试范围不断细化并逐渐清晰，需要增加前期忽视的测试用例，以完善测试用例。但后者增加的测试用例有限，从整体上看，测试用例绝对数量在不断减少，如图 7-13 所示。

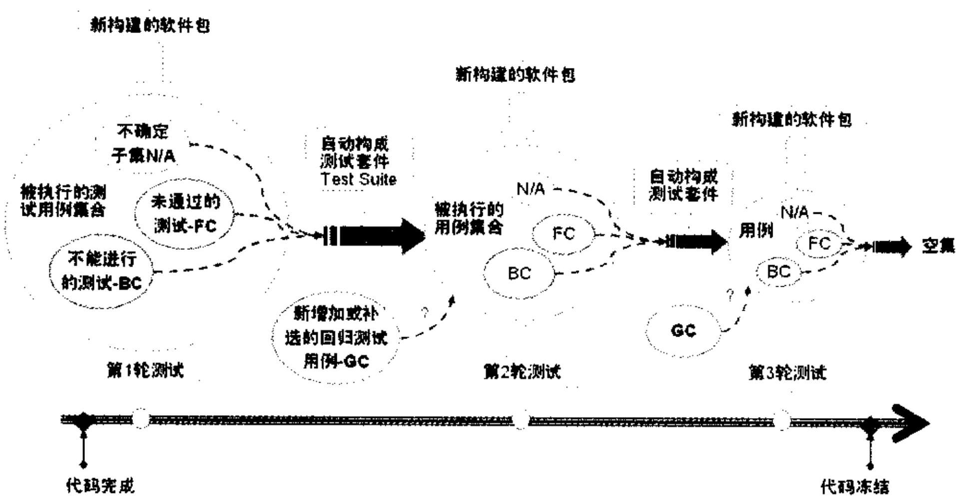


图 7-13 回归测试用例套件变化的过程

7.7 软件缺陷的报告

软件测试就是为了更早、更快地发现缺陷，换句话说，缺陷的发现可以看作是测试工作的主要成果之一。所以，从这样的角度看，缺陷报告是测试人员的一项重要工作，是测试成果的文档化，是联系测试人员和开发人员的纽带。将缺陷准确地报告出来，至少有 3 个基本的目的：

- 加快缺陷的修正。任何一个开发人员可以按照缺陷报告的描述再现软件缺陷。多数情况下，只有再现软件缺陷，开发人员才能确定缺陷产生的根本原因，从而修正缺陷。
- 产品的质量评估。记录下缺陷，可以了解各个模块、各个功能特性存在哪些缺陷，根据缺陷数据，能客观地对产品质量（包括设计、编程等）有一个客观的评价。
- 预防缺陷。通过所记录的缺陷分析获得经验和教训，有助于在今后的软件开发中预防缺陷。

7.7.1 缺陷的属性

为了准确报告、清楚地描述软件缺陷，首先需要全面地定义软件缺陷的属性。从实践经验知道，为了提高工作效率，预防缺陷和改进开发流程，软件缺陷被定义了 10 多种属性，并被分类组织，如表 7-6 所示。

表 7-6 软件缺陷的属性

类 别	属性名称	含义简要说明
可跟踪信息	缺陷标识	缺陷的唯一标识，用于识别、跟踪、查询、排序、存储管理等，可以使用数字序号表示
基本描述信息	标题	对缺陷的概括性描述，方便列表、浏览、管理等
	详细描述	包括前提、操作步骤、期望结果、实际结果等
	环境	缺陷发现时所处的测试环境，包括操作系统、浏览器等
	所属项目/模块	缺陷所属哪个具体的项目或模块，要求精确定位至模块、组件级
	产品信息	属于哪个产品、哪个版本等
	状态	缺陷一旦被发现之后，其被跟踪过程中所处的状态，如“激活的、修正的、关闭的”等状态构成了缺陷自身的生命周期
修正所需信息	严重程度	指因缺陷引起的故障对软件产品使用或某个质量特性的影响程度。其判断完全是从客户的角度出发，由测试人员决定。一般分为“致命”、“严重”、“一般”、“较小”等 4 种程度

续表

类 别	属性名称	含义简要说明
修正所需信息	优先级	缺陷被修复的紧急程度或先后次序，主要取决于缺陷的严重程度、产品对业务的实际影响，需要考虑开发过程的需求（对测试进展的影响）、技术限制等因素，由项目管理组（产品经理、测试/开发组长）决定。一般分为"最高的、正常的、最低的"3种级别
	类型	属于哪方面的缺陷，如功能、用户界面、性能、接口、文档、硬件等
	可能性	缺陷产生的频率。大多数缺陷产生的频率是100%，但也有一些缺陷产生的频率在1%~90%之间
	缺陷提交人	缺陷提交人的名字（会和邮件地址联系起来），即发现缺陷的测试人员或其他人员
	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；自动和该开发人员的邮件地址联系起来。当这缺陷被报出来时，系统会自动发出邮件
	来源	缺陷产生的地方，如产品需求定义书、设计规格说明书、代码的具体组件或模块、数据库、在线帮助、用户手册等
供事后分析所需的信息	产生原因	产生缺陷的根本原因，包括过程、方法、工具、算法错误、沟通问题等，以寻求流程改进、完善编程规范和加强培训等，有助于缺陷预防
	构建包跟踪	用于每日构建软件包跟踪，是新发现的缺陷，还是回归缺陷，基准（baseline）是上一个软件包
	版本跟踪	用于产品版本质量特性的跟踪，是新发现的缺陷，还是回归缺陷，基准是上一个版本
	提交时间	缺陷报告提交的时间
	修正时间	开发人员修正缺陷的时间
	验证时间	测试人员验证缺陷并关闭这个缺陷的时间

下面借助3个列表（表7-7、表7-8和表7-9），对几个重要属性——状态、严重性、优先级等进一步说明。一般来讲，缺陷的严重等级和优先级相关性很强，但是，低优先级和高严重性的错误也是存在的，反之亦然。例如，产品徽标很重要，一旦它丢失了，这种

缺陷是非常严重的，但这时，如果有更多的缺陷影响测试的进程，那么修正这个缺陷的优先级相对来说就不高了。因为修正这个缺陷，不影响其他任何地方，假如重新设计产品徽标，可以留到最后修正。

表 7-7 软件缺陷状态列表

缺陷状态	描 述
激活的或打开（Active or Open）	缺陷的起始状态。问题还没有解决，存在于源代码中，等待修正，如新报的缺陷
已修正的或已修复的（Fixed or Resolved）	已被开发人员检查、修复过的缺陷，通过单元测试，认为已解决但还有待测试人员验证
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 （reopen）	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（deferred）	这个缺陷不严重并被推迟修正，可以在下一个版本中解决
保留（Onhold）	由于技术原因或第三者软件的缺陷，开发人员目前不能修复的缺陷
功能增强	该问题符合当前的设计规格说明书，是一个有待改进的问题
不是缺陷	开发人员认为这不是问题，是测试人员误报的缺陷
不能重现（cannot duplicated）	开发不能复现这个软件缺陷，需要测试人员检查缺陷复现的步骤
需要更多信息（NeedMoreInfo）	开发能复现这个软件缺陷，但开发人员需要一些信息，例如：缺陷的日志文件、图片等

表 7-8 软件缺陷状态列表

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，都是致命的
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、操作时间稍长等问题
较小 （Minor）	操作者不方便或遇到麻烦，但不影响功能的操作和执行，如字体不美观，按钮大小不是很合适，文字排列不对齐等一些小问题

表 7-9 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

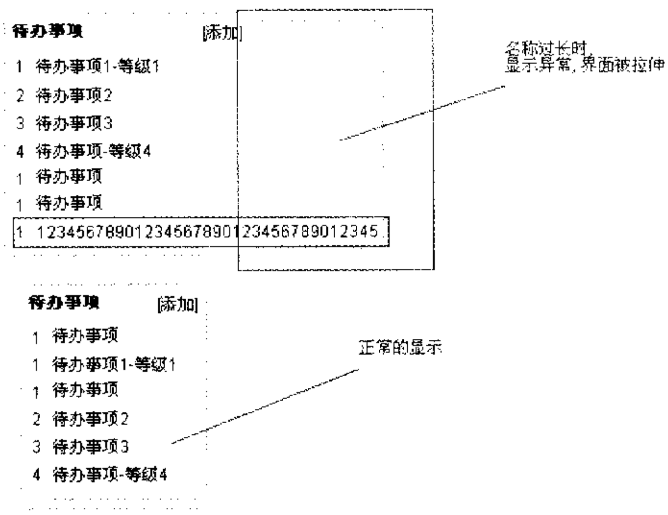
7.7.2 缺陷的详细描述

软件缺陷的详细描述，除了基本的 4 部分——“前提/测试数据、操作步骤、期望结果、实际结果”之外，还要考虑抓住一些软件运行出错时的 log，例如出现软件崩溃现象，需要把使用 Soft_ICE 或 WinDBG 等工具捕捉的日志文件作为附件提供给开发人员。有时，文字描述不够清楚，不过一张图片就能解决问题，如表 7-10 所示。

表 7-10 缺陷报告的示例

项 目	描 述
缺陷 ID	00002
缺陷状态	激活的
缺陷标题	待办事项标题大于 20 位时，显示异常
缺陷的严重程度	一般
缺陷的优先级	是正常的
缺陷的产生频率	100%
缺陷提交人	XXXXXX
缺陷提交时间	2007-3-26
缺陷所属项目/模块	雅虎日历输入模块－显示待办事项
缺陷指定解决人	YYYYY
缺陷指定解决时间	2007-3-30
缺陷验证人	XXXXXX
缺陷验证结果描述	不通过
缺陷验证时间	2007-4-1

续表

项 目	描 述
步骤	添加待办事项 输入名称长度大于20位 保存后，查看待办事项显示区域
期望的结果	正常的格式显示（超过长度部分用...代替，鼠标浮动在标题上时，显示全称）
实际发生的结果	显示模块出现了显示异常，界面被拉伸
测试环境	Windows系统，IE浏览器
图片、Log文件	

（1）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，被视为修复缺陷的向导。开发人员有时抱怨糟糕的缺陷报告，往往就是因为测试人员在这方面没有做好。

（2）“期望结果”与测试用例标准或设计规格说明书、用户需求等一致，达到了软件预期的功能。测试人员要站在用户的角度对它进行描述，这个描述提供了验证缺陷的依据。

（3）“实际结果”是测试人员收集的结果和信息，用以确认缺陷，并标识那些影响到缺陷表现的要素。

（4）相关信息和附件。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的什么位置，有什么问题等。有时还要附上日志（Log）文件，这有利于开发人员 Debug。

7.7.3 如何报告缺陷

软件缺陷的描述是软件缺陷报告的基础部分，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。准确报告软件缺陷是非常重要的，因为：

- 清晰准确的软件缺陷描述可以减少被开发人员退回来的缺陷数量。
- 提高软件缺陷修复的速度，使每一个小组能够有效地工作。
- 提高测试人员的可信任度，可以得到开发人员对有效缺陷的快速或及时响应。
- 加强开发人员、测试人员和管理人员的协同工作，让他们可以更好地工作。

在多年实践的基础上，我们积累了较多的软件缺陷的有效描述规则，主要是：

- 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缺陷常常只是部分被修复，不能得到彻底的修正。
- 可以再现。提供缺陷产生的精确操作步骤，使开发人员容易看懂，可以自己再现这个缺陷，通常情况下，开发人员只有再现了缺陷，才能正确地修复缺陷。
- 完整统一。提供完整、前后统一的产生软件缺陷的步骤和信息，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既短小简练，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”中的“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才会存在缺陷，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），这些条件是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”。
- 不做评价。在软件缺陷描述中不要带有个人观点，不要对开发人员进行评价。软件缺陷报告是针对产品、针对问题本身，将事实或现象客观地描述出来就可以了，不需要任何评价或议论。

7.8 小结

测试的执行是测试计划贯彻实施的保证，是测试用例实现的必然过程，严格地执行测试不会使测试工作半途而废。

在测试执行前，首先实施有效的培训和知识传递，准备好测试环境和安排好执行测试的人员，并确立正确的测试执行策略，因为一个有效、高质量的测试执行过程是一个立体作战的过程，有许多并发的工作同时进行，包括功能测试、系统测试、每日构建(Daily Build)验证、Ad-hoc 测试、每日缺陷验证和回归测试等。

其次，需建立有效的测试套件，优化组合测试的环境。然后，在测试执行的过程中，清楚地知道如何执行测试自动化的任务、如何采取适当的回归测试策略。最后，介绍如何报告缺陷，包括哪些内容构成缺陷的属性和描述信息。

在测试执行过程中，需要注意的事项有：

- 严格审查测试环境，包括硬件和网络配置、软件设置以及相关的或具有依赖性的第三方产品的版本号等，确保测试环境的准确性。
- 测试执行的过程也充满了挑战，需要不断思考，优化环境组合、精心挑选测试用例、确定有效的回归测试范围等，不断提高测试执行的效率，同时降低测试执行的风险。
- 所有测试套件、测试任务和测试执行的结果，都通过测试管理系统进行管理，使测试执行的整个操作过程记录在案，保证其良好的可跟踪性、控制性和追溯性，便易控制好测试的进度和质量。
- 对每个阶段的测试结果进行分析，保证阶段性的测试任务得到完整地执行并达到预定的目标。

准确、清楚和完整地报告缺陷，使开发人员能及时再现缺陷，从而促进缺陷得到及时的修正。

8

第8章

国际化和本地化测试的执行

软件国际化是软件工程技术的解决方案，是软件本地化的基础和前提。软件国际化理想水平是使软件本地化过程不需要改动任何源代码就能完成。

开发的软件产品，如果不局限于国内市场，则需要考虑软件产品的国际化和本地化，以适应国际市场的需求。对于互联网上的软件服务，像雅虎日历、Google talk 等软件产品的国际化和本地化需求表现就更为迫切一些。

软件产品的国际化是赋予软件产品的一种能力，这种能力可以使软件产品的本地化非常容易。理想情况下，将国际化的软件本地化时，不需要修改源代码，只需要翻译资源库，进行特定的设置和定制就可以了。

软件本地化是一个将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，包括翻译、界面设置、功能调整以及参数设置

等，还要验证是否符合各个地方的习俗、文化背景、语言（包括方言）等。

在完成基本的语言版本（base code）测试之后，我们还要进行国际化、本地化的测试。通常情况下，国际化测试可以和基本语言版本测试同时进行，而本地化测试是在国际化版本测试完成之后进行的。

概念

（1）**本地化**（L10N，英文的 Localization 一词的简写），由于首字母“L”和末尾字母“n”间有 10 个字母，所以简称 L10N。软件本地化是在源语言版本的基础上，通过翻译、定制和参数配置等工作，使软件产品或系统在语言、时区、度量衡、文化、风俗习惯等各个方面与国家和地区的相应内容一致，从而满足特定地区用户的使用需求。

（2）**国际化**（I18N，internationalization 的简写），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。

（3）**全球化**（Globalization），它是一个概念化产品的过程，是基于全球市场考虑的，以便一个产品只须做较小的改动就可以在世界各地出售。全球化可以看作是国际化和本地化两者合成的结果。

8.1 国际化测试

国际化是将来本地化的基础，即使现在还没有具体的本地化计划，软件产品也应该按照国际化要求去做，否则，将来需要本地化时工作量会很大，几乎所有的代码都要修改。所以国际化是核心工作，只有满足了国际化的要求之后才比较容易实现本地化，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念。图 8-1 简要说明了翻译、本地化与国际化等之间的关系。

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在不同语言、不同风俗的国家和地区使用的问题，从而对计算机设计和编程作出的某些规定。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改动代码就能完成。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。

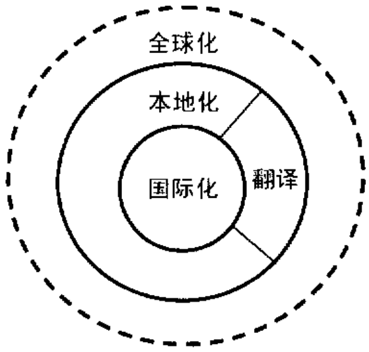


图 8-1 翻译、本地化与国际化之间的关系

8.1.1 软件国际化的基本要求

在介绍软件国际化之前，先举和日常生活密切相关的几个简单的例子。

示例

（1）**姓名**。英文中一个人的名字为 3 部分——First Name（名）、Middle Name（字）和 Last Name(姓)，在软件中显示时可以省去中间部分，即 First Name + 空格 + Last Name。英文名和中文名在软件中的显示至少有两点不同：

- 姓和名次序正好相反。
- 英文名在名和姓之间必须有一个空格。

在雅虎日历中文版上，用户登录后显示的只有名而没有姓，这不合乎中文的习惯，如下图所示，“欢迎光临 琪”是一个缺陷（如图 8-2 所示）。

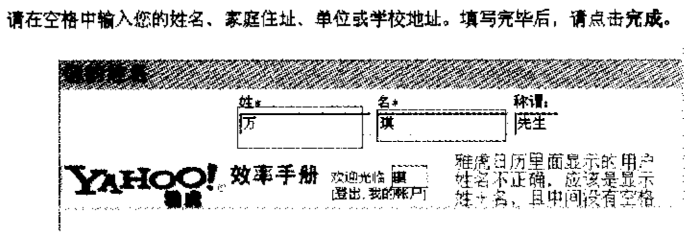


图 8-2

（2）**日期**。美国表示日期的习惯是“月/日/年”，而英国人喜欢用“日/月/年”，如 2007 年 1 月 31 日，美语中表示为“1/31/2007”，而英语中表示为“31/1/2007”。

（3）时区，我国处在东 8 区，即北京时间比格林威治时间早 8 个小时，而美国西海岸的时间比格林威治时间迟 8 个小时，也就是比北京时间迟 16 个小时。在北京用户使用的软件上显示 2007 年 1 月 31 日上午 9 点，则在美国用户的计算机上应显示当地时间为 “5:00pm 1/30/2007” 。

通过这几个例子可以理解软件国际化可能要碰到的各种问题，而且这些问题需要解决掉。例如，在数据库中，不能用一个字段去存储人名，必须将“姓”和“名”分开存储；在取得“姓”和“名”后，也不能直接将“姓”和“名”两字符串直接相加，而必须用单独的函数处理，即先判断这个用户处在什么地区，然后采用对应的字符串处理方法。对于时间，绝不能直接在数据库中存储各种不同时区的时间，而应该转化为格林威治标准时间（Greenwich Mean Time，GMT）后存储到数据库中，即在数据库中的所有时间都是零时区 GMT 时间，当客户端要显示时间时先把它转化为当地的时间，这样就不会把时间搞乱了。

从上面几个例子可以看出来，国际化软件需要从设计、编程等多个方面来实现。从程序角度看，国际化软件的编程不能像一次性软件项目那样随意，许多东西都不能简单处理，也不能写死（hard code）。例如，对于姓名处理、日期处理，不能仅仅通过一个简单的程序语句来处理，而需要通过一个函数处理，根据用户所处的时区、所用的语言和所在的国家，分别进行相应的处理。其次，软件处理和输出的文字、图片等数据，都应该从程序中分离出来，存储在单独的资源文件中，为以后软件本地化创造良好的条件。

国际化的软件，其用户的分布很广，用户的喜好和宗教等相差甚远。从设计角度看，系统首先要支持多字节字符的处理，支持 UniCode 字符集，然后设计出灵活的组件结构，使之能根据要求进行剪裁、定制等；客户端的时区和语言设置被抽象出来，融入到系统的整体设计中。如果是提供软件服务，则设计的系统架构和数据应具有很强的分布性，不仅可以在全球范围内构造分布式的网络传输系统，而且应很好地满足数据的异地备份，系统异地故障转移和用户就近访问系统的网络节点等一系列设计要求。

在软件国际化（I18N）测试中，设计部分的审查纳入到第 3 章中，而本章主要讨论软件国际化的实现部分，即着重讨论如何测试或验证下列特殊需求。

- 支持 Unicode 字符集。如建立用于本地字符编码（ANSI 或 OEM）和 Unicode 之间变换的字符映射表，既可以处理类似于英文的单字节语言，又能处理类似于中文、日文等双字节或多字节语言。
- 支持不同时区的设定、显示和切换。
- 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）。如建立资

源文件（*.rc）来存储这些内容。

- 消除硬代码（Hard code。指程序代码中所包含的一些特定的数据），而尽量使用变量处理，将数据存储在数据库或初始化文件中。
- 使用 Header files 定义经常被调用的代码段。
- 弹出的窗口、按钮、菜单等的尺寸具有自动伸缩性或可灵活地调整，以适应不同语言显示文本的长度变化。
- 支持各个国家的键盘设置，但要统一热键。
- 支持文字排序和大小写转换。
- 支持各国家的度量衡、时间、货币单位等不同格式的显示方式。
- 支持颜色字体等自定义。
- 拥有国际化用户界面设计。

概念

字符集是操作系统中所使用的字符映射表。最早的字符集，可以认为是 UNIX 系统使用的，包含 128 个字符的 7- bit ASCII 字符集（包括 tabs、空格、标点、符号、大小写字母、数字和回车键等）。随后，就是标准 8- bit ASCII，包含 256 个字符，早期的 Windows 操作系统使用 8- bit ASCII 字符集。由于扩展后的 ASCII 字符集还是无法满足所有语言的需求，如汉语、日语和韩语这些语言的字符都高达几万个字符。所以产生了 16-bit 字符集（双字节、多字节或变数字节）——统一的字符编码标准为 Unicode。

Unicode 是一个国际标准，采用双字节字符进行编码，提供了在世界主要语言中通用的字符，所以也称为基本多文种平面。Unicode 以明确的方式表述文本数据，简化了混合平台环境中的数据共享。目前，很多操作系统都支持 Unicode，包括 Windows 系统、Linux 系统和 Mac OS、Solaris、IBM-AIX、HP-UX 等。Unicode 简称为 UCS，现在用的是 UCS-2，即 2 个字节编码，与国际标准字符集 ISO 10646-1 相对应。UCS 的最新版本是 2005 年的 Unicode 4.1.0，而 ISO 的最新标准是 ISO 10646-3:2003。

Codepage 是各国的文字编码和 Unicode 之间的映射表。例如，简体中文和 Unicode 的映射表就是 CP936，其他的映射关系有：

- codepage=950 繁体中文 BIG5
- codepage=437 美国/加拿大英语
- codepage=932 日文

- codepage=949 韩文
- codepage=866 俄文

UTF-8/ UTF-16/ UTF-32。 UCS 只是规定如何编码，并没有规定如何传输、保存编码。所以有了 UniCode 实用的编码体系，如 UTF-8、UTF-7、UTF-16。UTF-8(UCS Transformation Format) 和 ISO-8859-1 完全兼容，解决了 UniCode 编码在不同的计算机之间的传输、保存的问题，使得双字节的 Unicode 能够在现存的单字节的系统上正确传输。UTF-8 使用可变长度的字节来储存 Unicode 字符，这能解决敏感字符引起的问题。前面有几个 1，就表示整个 UTF-8 串是由几个字节构成的。以下是 Unicode 和 UTF-8 之间的转换关系表：

```
U-00000000 - U-0000007F: 0xxxxxxx
U-00000080 - U-000007FF: 110xxxxx 10xxxxxx
U-00000800 - U-0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx
U-00010000 - U-001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U-00200000 - U-03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U-04000000 - U-7FFFFFFF: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
10xxxxxx
```

示例

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<%@page pageEncoding="UTF-8"%>
```

比较完整的字符集列表

utf-8 Unicode, worldwide	shift_jis Japanese, Win/Mac
utf-16 Unicode, worldwide	iso-2022-jp Japanese, email
iso-8859-1 Western Europe	euc-kr Korean
iso-8859-2 Central Europe	gb2312 Chinese, simplified
iso-8859-3 Southern Europe	gb18030 Chinese, simplified
iso-8859-4 North European	big5 Chinese, traditional
iso-8859-5 Cyrillic	tis-620 Thai
iso-8859-6-i Arabic	koi8-r Russian
iso-8859-7 Greek	koi8-u Ukrainian
iso-8859-8 Hebrew, visual	iso-ir-111 Cyrillic KOI-8
iso-8859-8-i Hebrew, logical	macintosh MacRoman
iso-8859-9 Turkish	windows-1250 Central Europe
iso-8859-10 Latin 6	windows-1251 Cyrillic
iso-8859-13 Latin 7, Baltic Rim	windows-1252 Western Europe
iso-8859-14 Latin 8, Celtic	windows-1253 Greek
iso-8859-15 Latin 9	windows-1254 Turkish
iso-8859-16 Latin 10	windows-1255 Hebrew
us-ascii basic English	windows-1256 Arabic
euc-jp Japanese, Unix	windows-1257 Baltic Rim

8.1.2 国际化测试

软件国际化的测试就是验证软件产品是否支持上述特性，包括多字节字符集的支持、区域设置、时区设置、界面定制性、内嵌字符串编码和字符串扩展等。软件国际化的测试通常在本地化开始前进行，以识别潜在的不支持软件国际化特性的问题。理想的情况是，国际化测试在英文版本完成时就已结束。

实际上，设计评审和代码审查是国际化测试中最有效的方法，首先在设计上，要验证其是否遵守软件国际化的软件开发标准，是否具有国际化特性的一些基本功能——用户的时区、语言和地区等设置，然后审查程序代码和资源文件，确认源代码和显示内容是否被分离、是否使用各类正确的数据格式处理函数等。代码审查，可以采用走查的方法。先列出一个简单的检查列表（Checklist），依据这个列表，从头到尾快速地浏览所有代码，确保在代码上对 I18N 的充分支持。通过代码审查，可以发现大部分有关 I18N 的问题。

除了设计评审和代码审查之外，I18N 测试有两种基本方法：

- 一种是针对源语言的功能测试。在源语言版本中，直接检查某些功能特性是否符合要求，如不同的区域设置、不同的时区显示等。
- 另一种是针对伪翻译（pseudocode，pseudo-translation）版本的测试。即文字、图片信息中的源语言被混合式的多种语言（如英文、中文、日文和德文等）替代，然后进行全面的 I18N 测试，包括相关的功能测试、界面测试，但不包括翻译验证等。

概念

伪翻译（Pseudo Translation）是软件国际化测试的重要手段之一，它可以选择一种以上的本地化语言模拟本地化处理的结果。可以在进行实际本地化处理之前预览和查看本地化的问题。通过伪本地化翻译，可以发现源语言软件的国际化设计中的错误，方便后续本地化时处理，提高软件的可本地化能力。在某些本地化工具中，可以设置在进行了本地化翻译后字符长度的扩展比例，替换字符，前缀和后缀字符。

1. 针对源语言的功能测试

I18N 的测试不同于本地化的测试，其中部分测试工作可以在源语言中进行。假定源语言是英文，我们可以在英文版本中进行下列测试：

- 时区的设置及其相应的时间显示。
- 地区的设置及其相应的日期、货币显示。

- 可以选择不同的语言。
- 输入多字节字符串。

图 8-3、图 8-4 就是两个例子，用户可以设置时区、地区和语言的界面。

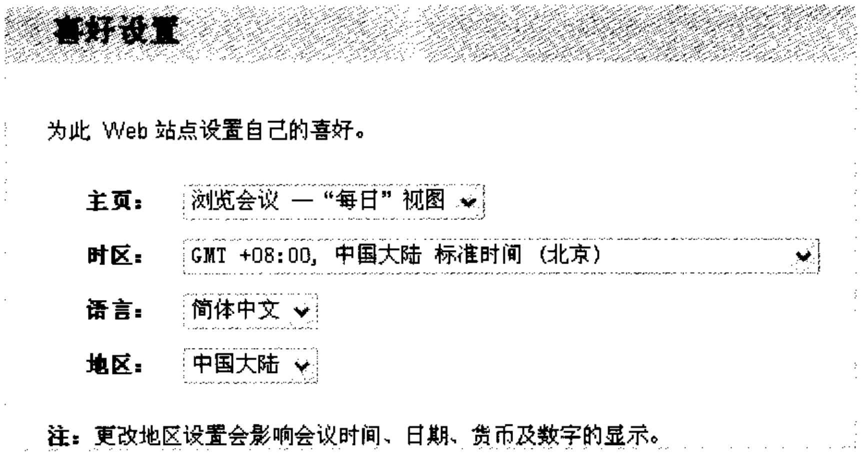


图 8-3 时区、地区和语言的设置界面

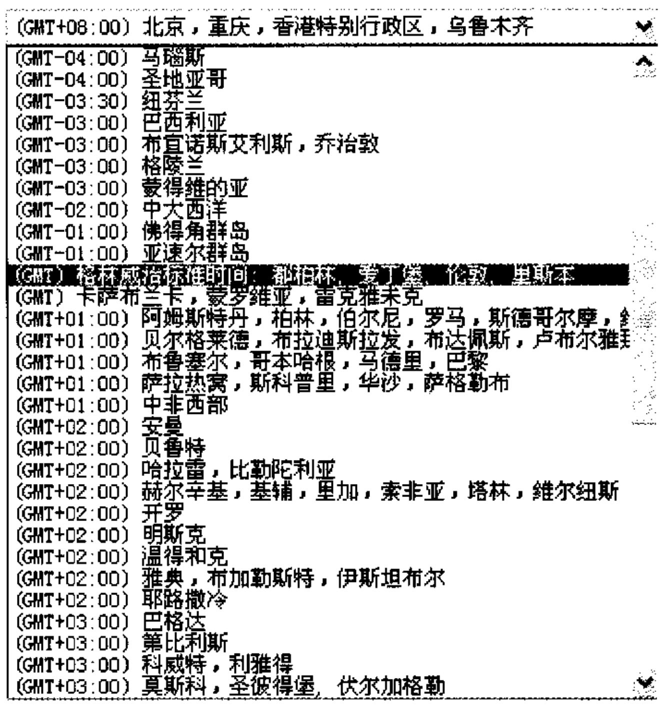


图 8-4 以 GMT 为中心的时区划分

2. 针对伪翻译版本的测试

源语言的测试具有局限性，不能完全验证软件产品是否能很好地支持多字节字符集，例如弹出的窗口是否可以根据显示内容的长度进行自我调整，窗口中显示的内容是否会出现会乱码现象等？这时，可采用伪翻译（pseudo-translation 或 pseudocode）的版本，进一步完成 I18N 的测试。采用伪翻译版本，可以完成下列测试任务：

- 测试多字节字符集和脚本。
- 测试多字节字符串的输入显示是否正确。
- 测试多字节字符文件、文件名、文件夹及其处理。
- 测试索引和排序。
- 测试本地化的操作系统、键盘支持等。

如果采用正式翻译的版本，也可以进行相关的 I18N 测试，例如当源语言是英文，可以采用繁体中文、日文和阿拉伯文等作为几种典型的语言版本来对 I18N 进行更充分的测试。I18N 测试的重点在功能上，但将 2～3 种语言的本地化测试和 I18N 测试结合起来也没有坏处。采用伪翻译版本的好处是：

- 可更早地进行 I18N 测试，能够比较快也更容易在源语言版本之上构造伪翻译版本，因为不需要准确翻译，甚至不需要翻译，仅仅是替换成不同语言的文字。
- 可以一举两得，即同时验证多种语言的不同特性，如中文的多字节文字又精炼简短，德文的长度，以及中英文混合等。
- 给测试者一个清晰的信号，让他知道这是 I18N 测试，不是 L10N 测试。

小技巧

Windows 是支持 Unicode 的多语言操作系统，所以在进行 I18N 测试时，可以充分利用它支持多语言的特性，安装多种语言的字库，通过改变 Windows 控制面板中“区域和语言”选项的“Unicode 支持和系统显示语言”，使测试机可以测试多种语言的安装包。

8.1.3 I18N 测试实例

如果软件支持 Unicode，即使同时显示不同的语言，也不会出现乱码。据此可分别对雅虎日历和 Google Talk 进行 Unicode 支持方面的测试。

在雅虎日历中，输入简体中文的活动名称，然后把这个用户的语言切换到繁体中文上，

看其是否存在乱码。结果，活动名称的显示出现了乱码，这表明雅虎日历不支持 Unicode 和 UTF-8/UTF-16，如图 8-5 所示。

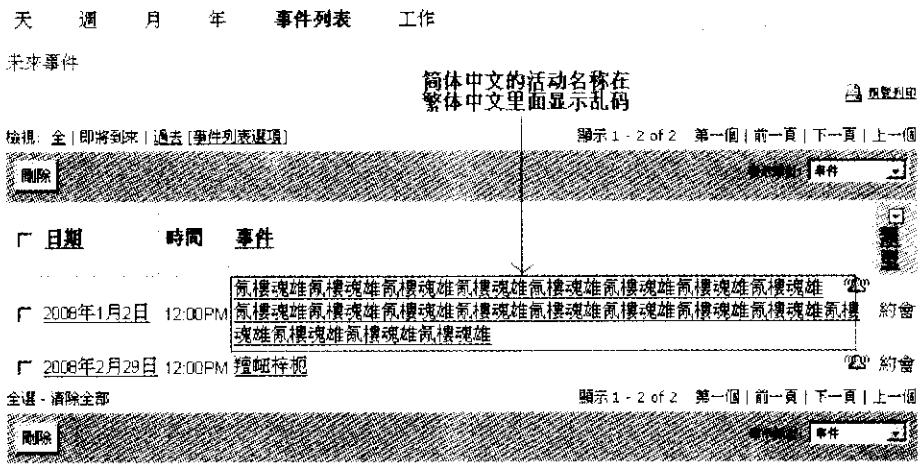


图 8-5 不支持 Unicode 而出现乱码

Google Talk 的“Unicode 支持”的测试结果又是怎样呢？为了验证这个问题，可先安装简体中文和日文的 Google Talk，然后在他们之间相互发送本语言文字的聊天内容。测试结果表明 Google Talk 支持 Unicode，中文和日文都可以正确地显示在对方的聊天对话框里面，没有任何乱码，如图 8-6 所示。

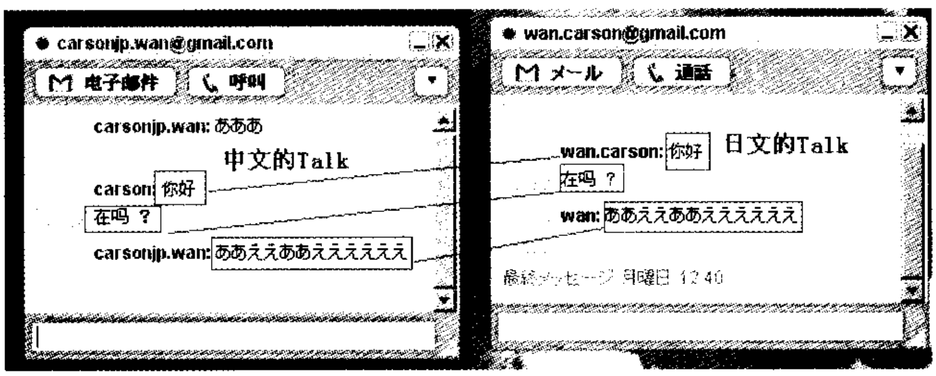


图 8-6 在客户端同时显示中文和日文

图 8-7 更能说明 Google Talk 完全支持 Unicode，在发送的一条信息中，同时显示了土耳其文、繁体中文、法文、日文、简体中文、葡萄牙文、俄文、意大利文、德文、荷兰文、韩文、英文等。

图 8-7 同时显示多种文字的界面

8.2 本地化测试

软件本地化是建立在软件国际化的基础上的，也就是说，软件的国际化特性在源语言版本中获得了充分的支持后，源语言版本则容易转化为多种目标语言的版本。在向目标语言转化的过程中，需要完成翻译，其他本地化开发工作，测试等任务，最后发布多语言版本的产品，如图 8-8 所示。

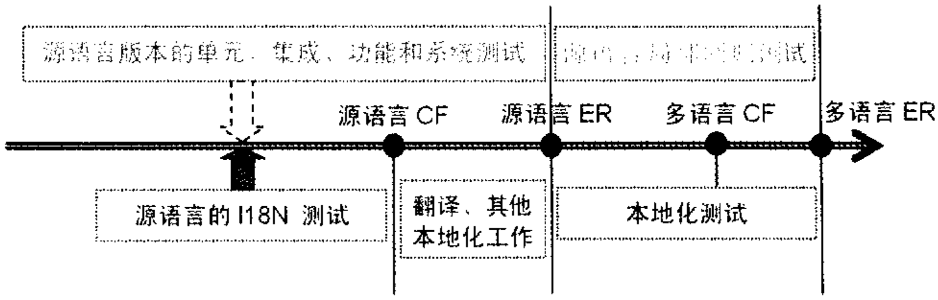


图 8-8 软件国际化、本地化过程示意图

8.2.1 软件本地化的质量需求

提到本地化，大家首先想到的就是翻译问题。毋庸质疑，翻译在本地化工作中占据着很重要的地位，但绝不能把翻译等同于本地化，翻译和本地化有很大的不同。翻译只是本地化工作的一部分，翻译只是将所提取的各种源语言文字信息转换为另一种目标语言的文字信息。当文字被翻译后，不仅要完成各种目标语言文字的替换或写入，构成新的资源库、资源文件等，而且需要进行有关本地化的技术和文化层面的修改，使软件产品在内容、外观、设置等多方面适应目标地区的需求。

1. 技术层面的更改

- 调整软件元素尺寸大小，因为翻译常会引起输入框、文本框、表格不再适应的问题。
- 重新创建图标、图形和图片，图中显示的文字也需要改为目标语言。
- 根据特定语言的习惯调整默认设置，可以竖向显示日文，或者从右向左显示阿拉伯文字。
- 调整热键，其他不适合的功能也要重新设置。

2. 文化层面的更改

- 包装。由于民族风俗的不同，有些国家对某些颜色和数字比较敏感，需要区别对待。如我国和日本都忌讳数字“4”。
- 图标。由于文化背景有区别，所以应该慎用动物图案。
- 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。如地方规章和宗教信仰等需区别对待。

3．本地化工作步骤

虽然在具体操作时本地化的工作步骤可能会有不同，但软件本地化的基本步骤没有本质区别。

（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，并存储在相关的数据库中。

（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。源语言、目标语言可以存储在相关的数据库中，以便实现本地化的自动化工作更容易。

（5）把翻译好的文本、图片重新插入目标语言的源代码或资源文件中。

（6）如果需要，编译目标语言的源代码，构造目标语言软件包。

（7）定制软件的界面，包括改变颜色、默认值、列表排序等。

（8）测试翻译后的软件，调整 UI 以适应翻译后的文本。

（9）测试本地化后的软件，确保格式和内容都正确。

知识点

国家与颜色。我国喜欢红色、黄色，而美国人不喜欢鲜艳的红色，如美国的安全最高警戒级的颜色是红色，股票上涨是绿色，下跌是红色，正好和我国相反。日本人认为黄色是不吉利的，而视紫色为富贵。我国以红色为喜庆的颜色，传统的结婚礼服是红色的，而英美等国家则是白色的礼服，如表 8-1 所示。

表 8-1

地　区	国　家	爱好的颜色	禁忌的颜色
亚洲	中国	红、黄、绿	黑、白
中东	沙特阿拉伯	绿、深蓝与红白相间	粉红、紫、黄
非洲	埃及	红、橙、绿、青绿、浅蓝	暗淡色、紫色
欧洲	挪威	红、蓝、绿、鲜明色	
南美洲	阿根廷	黄、绿、红	黑紫、紫褐相间

国家与数字。日本人、中国人比较忌讳数字“4”，若是四个（单件）一组包装的产品在日本就不容易卖出去。西方人比较忌讳“13”，大酒店没有第 13 层，从第 12 层就直

接跳到第 14 层。

地区宗教。几乎每个国家都有自己信仰的宗教，如佛教、道教、伊斯兰教、基督教、天主教、犹太教 、东正教、印度教、锡克教、 拜火教、波斯明教等。

8.2.2 本地化测试的基本内容

软件在全部翻译完毕，对技术部分也做了必要的调整之后，还有一个软件测试的问题，即不论原来的软件产品有多成熟，本地化时经过了很多人的重新创造，所以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。

软件本地化测试则检查为适应某一特定文化或地区而进行本地化的产品的质量。这个测试是基于国际化测试的结果进行的，本地化测试包括 6 个方面：

- 功能性测试。
- 数据格式测试。
- 可用性测试。
- 翻译验证测试。
- 兼容性测试。
- 文档测试。

1．功能性测试

- 在本地环境上进行安装、升级等测试。
- 为索引和排序功能进行的测试。
- 集成测试。多字节文字构成的接口参数传递出错，可能会导致集成失效。包括函数之间、模块之间、客户端和服务器之间、服务器之间等不同层次和类型的集成。
- 数据库的默认值、初始化等。
- 有些地区支持夏令时（Daylight save time, DST），需与时区选择和时间显示的功能进行联合测试。
- 受多字节文字影响的其他功能。

2．数据格式测试

数据格式是本地化测试的重点，可能遍布软件系统的各个地方，要仔细检查各类数据

格式的输入、存储和输出是否正确，包括数字、货币、时间、日期、度量衡等。如图 8-9 所示，中文版的日历中日期显示有问题，因为中文的年份应在月份前。

图 8-9 本地化版本的日期显示问题

3. 可用性测试

- 界面是否美观，是否符合本地用户的审美观和对颜色的喜好。
- 界面信息的一致性。
- 操作是否符合本地习惯，包括快捷键的设置，命令键的顺序等。
- 受本地化文字影响的其他用户界面，如文字超出按钮和菜单的边界、乱码等。

4. 翻译验证测试

翻译验证测试（Translation verification testing，TVT）虽然不能等同于本地化测试，但在本地化测试的工作量中占有较大的比重。包括：

- 检查所有专业术语的建立、翻译和存储。
- 确保所有菜单、按钮、提示、说明、图片等各类文字信息被翻译了。
- 确保翻译内容的正确性、准确性和语言完整性。如“OK”在日文里，不需要翻译。
- 特殊的语言和文化环境检查。
- 政治敏感内容的检查。
- 宗教内容的检查。

5. 兼容性测试

兼容性测试包括硬件和软件兼容，可适应本地的一些特殊应用环境要求，所以兼容性

测试也被称为本土测试（In-country testing），它能使软件产品或系统真正适应本土的环境，如：

- 兼容第3方本地化软件。
- 兼容本地流行的硬件，如键盘、打印机、扫描仪等。

6. 文档测试

检查各类文档翻译的质量、完整性，以及在所有的文档和应用程序界面中术语使用的一致性。要检查的文档包括产品发布信息（Release Note, Readme）、用户手册、技术参考手册、在线帮助、错误信息列表等。

8.2.3 L10N的功能测试

任何一件软件产品，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的。如果软件得到了充分的I18N测试，在软件本地化的实施过程中一般不会产生功能方面的缺陷，功能方面的风险很小。但是，在实际工作中，I18N测试的覆盖率只在80%左右，还有20%的I18N特性需要在本地化测试阶段被验证，因此，软件本地化（L10N）的功能测试还是必要的。

本地化测试，不仅要查看用户界面，而且要对文件保存、打印等类似的功能进行测试，特别要注意语言环境特定的组件，比如对时间、日期格式以及文字处理等相关方面的功能进行测试。对软件进行L10N测试，关键一点是要在相对真实的本地用户环境上运行测试，如在本地化的操作系统、浏览器和特定的字符集（如中文GB2312、日文SHIFT_JIS、韩文EUC-KR等）、网络设置和硬件等上进行测试。比如，在日本有一种NEC9800系列计算机的特定版本，这些系统与在美国使用的系统截然不同。那么就需要在客户实际使用的设备上进行测试。在环境的验证测试中，安装测试是必要的一个环节，能比较全面地验证环境的适应性和兼容性。在安装完之后，进一步验证功能有没有受到本地化工作的影响。

下面通过一些例子来说明如何进行L10N各个方面的功能测试。

1. 集成测试

集成测试的一个重点是测试在客户端和服务器端之间的相互作用，客户端的本地化往往考虑得比较充分，而服务器端有时会被忽视。如果源语言是单字节字符集的欧美语系（如英文、法文等），在转换为多字节字符集的目标语言（如中文、日文等）后，容易引起问题，如客户端发出的请求，服务器端不能识别，甚至会出现崩溃。

2. 默认值

当从源语言转化到目标语言时，软件功能特性中许多设置的缺省值（时区、城市、数字等）是不同的，需要进行全面验证。例如，在中文里，时间缺省值从 0 点开始；而在英语里，没有 0 点，是从 12 点开始的，即 12:00am, 1:00am, …, 11:00am 后仍为 12:00pm, 1:00pm, …, 11:00pm。12:00am 相对于我国的 0:00am，而 12:00pm 不是半夜，是中午。

3. 索引和排序

英文的排序和索引习惯上按照字母的顺序来编排，但是对于一些非字母文字的国家（亚洲、阿拉伯国家等）来说，这种方法就不适用了。如中文就有拼音、部首和笔画等不同的索引方法。即使是使用字母文字的国家，其排序方法和英文也有较大不同。比如瑞典语，它的字母比英文字母多 3 个，在索引排序时也应加以考虑。所以，在本地化软件时，应该根据不同国家和地区的语言习惯分别加以考虑，在进行本地化测试的时候更应该仔细核对这些问题，如把英文软件本地化的瑞典语软件版本中，用来排序的有 29 个字母，在字母 A, B, C, ..., X, Y, Z 后会增加几个特殊的字母——瑞典语中的 3 个字母，即 Ä、Å、Ö。

4. 联机文档的功能测试

不论是 HTML 格式的，还是 PDF、Flash 等格式的联机文档，都需要得到测试。因为在这些文档中包含有菜单、链接、按钮等操作，测试可确保这些操作达到预期的结果，如链接能正常跳转，能打开正确的页面，包括页面中的图片可以正确地显示出来。测试人员需要关注页面上的超级链接，以发现其“提示未被本地化或链接到非本地化的页面上”等一系列问题。即使暂时不能完成所有页面的本地化，不得不引用部分源语言（英文）的页面，也需要用目标语言在这些链接旁给出提示，指出这些站点是英文的。

5. DST 的处理

一般情况下夏令时（DST）在程序中的处理比较简单。相当于多了一个时区，如美国西海岸采用太平洋时区（GMT-8:00），比格林威治标准时间迟 8 个小时，如果在夏令时（从每年 3 月最后一周的星期日到 11 月第一周的星期日）时区就相当于变成了 GMT-7:00，比格林威治标准时间迟 7 个小时。

但是，如果某些软件功能特性是跨时间段的，如 MS OutLook、Lotus Notes、雅虎日历等功能，这时夏令时的处理会变为一个很复杂的问题。例如，当安排一个年内的每周会议时，规定在每周星期四 9:00am 开，不管是不是在 DST，会议总应该在 9:00am 开，不能出现下列情况：

- 在非 DST 期间安排的会议，在 DST 期间会议则变为 8:00am 开。

- 在 DST 期间安排的会议，在非 DST 期间会议则变为 10:00am 开。

示例

如果某个会议被邀请的人员有的来自于美国旧金山，有的来自于中国北京，有的来自于澳大利亚悉尼。这其中中国不实行 DST，美国、澳大利亚实行 DST，但 DST 起始和结束的日期又是不一样的，因为澳大利亚悉尼的 DST 起始时间是 10 月的第 3 周星期日，结束时间是第二年 3 月的第 3 周星期日。要保证来自于不同国家的用户都能准时、同步加入会议，客户端显示的时间绝对不能出错。

假如会议开始时间是北京时间的星期四 9:00am，这时至少要完成下列一些测试用例，如表 8-2 所示。

表 8-2

用例时间	北京用户客户端显示时间	旧金山用户客户端显示时间	悉尼用户客户端显示时间	说 明
3月第3周的星期四	星期四 9:00am	星期三 5:00pm	星期四 11:00am	旧金山和悉尼都不在 DST 期间
3月第1周的星期四	星期四 9:00am	星期三 5:00pm	星期四 11:00am（DST）	旧金山不在 DST 期间 悉尼在 DST 期间
10月第2周的星期四	星期四 9:00am	星期三 5:00pm	星期四 11:00am（DST）	旧金山在 DST 期间 悉尼不在 DST 期间
10月第3周的星期四	星期四 9:00am	星期三 5:00pm	星期四 11:00am（DST）	旧金山和悉尼都在 DST 期间

如果会议在中国的星期一早上开，则美国为周日，这时要考虑 DST 切换的那个周日，即增加对边界条件的测试，问题更复杂一些。极端情况下，在北京时间周日下午 4 点的活动，会碰到美国周日凌晨 12:00am，即在 DST 切换的那一个时刻，需要进行更多的测试。

8.2.4 L10N 的数据格式验证

在本章开始时曾介绍过姓名、日期等格式的例子，除此之外，数字、货币和度量衡等的表达方法或格式在不同的国家也是不相同的，所以在把软件本地化的测试中，数据格式的验证也是重要的工作之一。如果是 Windows 编程，可以使用其标准应用接口（API）来处理这类转换的问题，问题会简单得多。但在一些 Web 开发和一些特殊的客户端开发，

还要依赖于程序本身的处理，通过定义相关变量和函数，来对这些本地化的数据格式进行处理。但从测试的角度来看，不管是哪种方式，都需要对这些数据格式的输入和输出进行充分的测试。

在 Java 2 SDK 中，已能很好地支持国际化的编程，已建立了支持国际化和本地化的类（如 Calendar、TimeZone、DateFormat、SimpleDateFormat 等）和方法。如果所使用的编程语言没有特定的类或函数支持，就不得不自己构造一些特殊的函数来处理不同语言的数据格式了。例如使用自定义函数 LocLongdate（）、LocShortdate（）等替换原来的 date（）函数，来处理日期的完整格式或简单格式的显示。

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。否则，有可能一个顾客存入了 5000 欧元，却只能取出 5 美元。例如，同一个数字（5700），在美国、意大利和瑞士有 3 种不同的表达方式：

- 美国：5,700
- 意大利：5.700
- 瑞士：5 700

2. 货币

除了数字转换外，几乎每一个国家都有标志本国货币的特殊符号，这些符号应该出现在金额的前面或后面也各不相同。例如，我国人民币符号“￥”出现在数量前，而美元符号出现在数量后面。

- 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。如晚上 10 点 45 分，其表达方式有：

- 美国：10:45 PM，
- 德国：22.45
- 加拿大法裔：22 h 45

4. 日期格式

不同国家的日期显示格式也不是一致的。美国的标准是用 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，如在美国，一个星期的第一天是星期天，而法国的日历每周第一天是星期一。

5. 度量衡的单位

虽然许多国家开始使用国际公制度量系统，如米、公里、克、千克（公斤）、升等，但美国、英国等一些国家仍旧使用英式度量单位，如英尺、英里、盎司、英磅等。因此，软件本地化必须解决公制和英式度量单位的问题，对于不同的国家可以显示其适用的度量衡体系。在比较理想情况下，软件应提供度量衡的用户设置功能和不同度量单位之间的转换功能。

6. 复数问题

生成复数的规则会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的，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。如搜索结果显示可以表示为：

“%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，则该信息显示格式如下：

“1 program searched” and “1 file searched”

或者

“3 programs searched” and “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 而其复数 programs = programma’s

file = bestand 而其复数 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。

表 8-3 一些国家和地区的日期、时间和货币格式[1]

Argentina	17/06/03	Martes, 17 de Junio de 2003	2:30 pm	1.234,56	Pesos
Australia	5/2/01	Tuesday, 17 June 2003	2:30 pm	1,234.56	Australian Dollars
Canada	05/02/01	Monday, February 5, 2001	2:30 pm	1,234.56	Canadian Dollars
French Canada	05/02/01	17 juin 2003	14:30	1 234,56	Canadian Dollars
Chile	17-06-03	Martes, 17 de Junio de 2003	2:30 pm	1.234,56	Pesos
China	01-2-5	2001 年 2 月 5 日	14:30	1,234.56	Yuan Renminbi
Columbia	17/06/03	Martes, 17 de Junio de 2003	2:30 pm	1.234,56	Pesos
France	05/02/01	mardi 17 juin 2003	14:30	1 234,56	French Francs
Germany	05.02.01	Mittwoch, 5. Februar 2001	14:30	1.234.56	Deutschmarks
Italy	05/02/01	lundi 5 febbraio 2001	14.30	1.234,56	Lira
Japan	2001/05/02	2001 年 2 月 5 日	14:30	1.234.56	Yen
Korea	2001/05/02	2002년 1월 15일 화요일	14:30	1,234.56	Won
Mexico	17/06/03	martes, 17 de junio de 2003	2:30 pm	1,234.56	Pesos
New Zealand	5/02/01	Monday, 5 February 2001	14:30	1,234.56	New Zealand Dollars
Spain	05/02/01	lunes, 5 febrero 2001	14:30	1.234,56	Peseta
Sweden	2001-02-05	mÂndag 5 februari 2001	14:30	1 234.56	Krona
Switzerland	5.2.2001	Montag, 7. Februar 2001	14:30	1'234.56	Swiss Francs
U.S.	2/5/01	Monday, March 5, 2001	2:30 pm	1,234.56	US Dollars
U.K.	05/02/01	Monday, 5 February 2001	14:30	1,234.56	British Pounds
Venezuela	17/06/03	Martes, 17 de Junio de 2003	2:30 pm	1.234,56	Bolivares

[1] 更多的可以参考：*http://l10n.openoffice.org/nonav/i18n_framework/cldr/LocaleDataAudit_OOo202.html; http://java.sun.com/docs/books/tutorial/i18n/index.html*。

8.2.5 L10N 的 UI 验证

软件首先展示给用户的就是界面（UI）和包装，所以 UI 的重要性是不言而喻的。如果在 UI 上出现错误甚至有歧义，会让用户对该软件质量产生不信任的心理。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。

由于源代码没有充分考虑到 I18N 的特性要求，很多软件本地化之后在 UI 外观上会出现一些不尽如人意的地方。常见的问题有：

- 有一些没有翻译的字段。
- 左右对齐问题，中文比较容易对齐，英文就比较难了。
- 大小写问题。英文有大小写问题，简单地采用翻译过来的英文字母是不够的。
- 连字符号的连接不规范。
- 文字方向的问题，如希伯莱文和阿拉伯文。
- 文字遮挡图像，文字超过边界等问题。
- 显示乱码。

有些问题，表面看是本地化问题，实质是软件国际化特性支持不够在本地化版本中的表现。从根本上解决这类问题还需依赖于源语言版本的兼容性设计，这种设计需支持 Unicode 或 UTF-8/UTF-16 字符集，并且各种对象（如菜单、表格和按钮等）的尺寸应具有动态适应性，以适应翻译后不同语言文字的扩展。

在进行本地化 UI 测试时，要认真细致地排除各种可能出现的问题。例如，HTML 页面上有些文字不是一眼就能看到的，所以需要特别注意这些不起眼的地方：

- 在浏览器界面顶部的显示页面标题。
- 图片的标题，当图片正在下载或者鼠标指向该图形时所显示的内容（ALT 属性）。
- 超级链接的标题。

8.2.6 L10N 的配置和兼容性验证

如上所述，软件本地化测试的配置和兼容性测试，会考虑到与本地的硬件兼容、与第 3 方本地化软件兼容。比较有利的一面是多数硬件支持国际标准，如大多数外部设备都支

持 USB 接口，驱动程序也支持多字节字符集。这里以数据库、热键的兼容问题和一个有关日文的示例进一步阐述软件本地化的配置和兼容性测试。

1. 数据库问题

软件本地化会涉及到数据库的改动。例如，由于文本的"最大长度"属性只限制输入字符的长度，而非字节长度，如果把多字节字符解析成 NCR 形式（&#dddd;），会导致输入的字符长度超出数据库字段宽度，从而引起数据库一类的问题。在本地化过程中有不同的解决方案来避免上述问题的产生。例如，可以在输入页面提交之前检测输入字符的宽度是否超长等。

2. 热键

在做本地化测试的时候，还有一个不能忽略的问题——热键问题。许多程序都为不同的命令设置了热键（键盘快捷方式），例如，在微软的 Word 中，可以通过同时按下 Ctrl 和 F 键来打开"查找"对话框。通常，文字被翻译之后，原来的热键很可能不再适用，需要为翻译过的文本设定新的热键。例如，当"Close"被翻译成德语"Schließen"之后，原有的热键 Alt-C 也应该相应的变为 Alt-S。新的热键应该和本地操作系统环境相匹配，确保所有的热键都是唯一的。不过中国、日本和韩国的版本，都沿用英文原有的热键，所以本地化之后不存在这个问题。

示例

日文版的 Google Talk 输入法测试结果

在实际测试的过程中发现一个问题——输入日文后，如果没有把鼠标的焦点移到 Google Talk 的信息输入框里面，必须连续按两次 Enter 键，文字才被发送出去。做一个对比，打开微软的记事本输入日文，结果一样——必须连续按两次 Enter 键，文字才可以输入到记事本中。但是，在 Google Talk 里面输入文字后，如果用鼠标点击输入框，文字下面的波浪线就会被自动去掉，只按一次 Enter 键信息就可以发送。由此可以看出 Google Talk 针对日文输入法的鼠标消息作了特殊处理，他们的考虑更周到、更人性化，如图 8-10 所示。

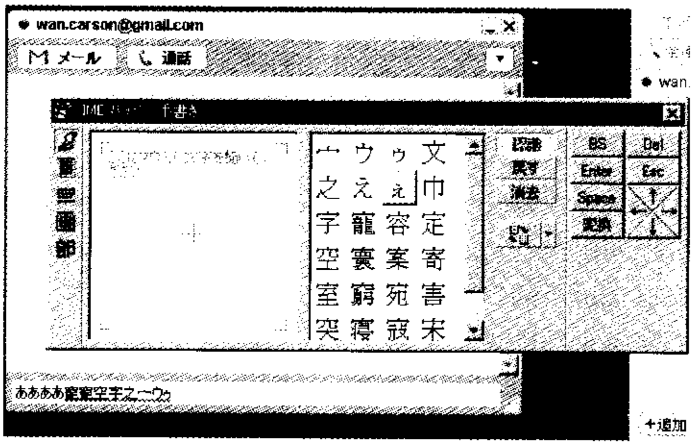


图 8-10

8.2.7 L10N 的翻译验证

翻译是本地化的一个重要的步骤，但是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，所以直接采用词对词的翻译远远不够，因此，应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。首先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），都可以使用相应的工具或者自己开发的工具，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，可以说大部分本地化测试工作都是在此基础上展开的。

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，整个本地化过程涉及翻译、编程、本地化数据库建立、测试和其他一些本地化工作，而且资源文件本地化翻译后可能会带来一些本地化错误，例如，控件的大小和位置错误，丢失热键（Hotkey）等。测试人员有义务帮助翻译人员发现翻译中的错误和不妥之处，指出一些晦涩或表达不准确的地方。从语言本身讲，翻译的质量可通过目标语言的语言专家审稿来把握，但是：

- 语言专家不一定了解专业术语、软件的上下文关系等，这些还需要通过软件测试人员来进行验证。
- 由于翻译工作的工作量很大，翻译人员可能会遗漏或疏忽某些地方。
- 检查不符合本地文化和软件产品的措辞。

所以，测试人员要配合翻译人员，及时对翻译的内容进行验证，确保翻译的质量。

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，汉语则用“《》”表示书名。

几乎各国都有表示自己货币的货币符号，如：美元$、人民币￥等。在翻译的过程中，这些符号是绝对不能出差错的。如果一个金融软件把本该用“￥”的地方，用了“$”，后果将是不堪设想的。这些也都是本地化测试应该特别注意的细节问题。

关于翻译验证，概括起来有以下几点建议：

- 检测翻译的句子是否复杂、难懂，能否拆分为简单句型。
- 检测翻译的内容是否脱离了上下文关系，意义表达不准确。
- 检测翻译后的文字中是否使用了意义含糊的词。
- 如果使用了缩写词，检查缩写词在第一次出现的时候是否正确地标出了全称，以便用户能够明白其含义。
- 检查不同国家的标点符号、货币单位等是否显示正确。

如图 8-11 所示，在 Google Talk 的安装界面中，就有 3 个翻译错误：

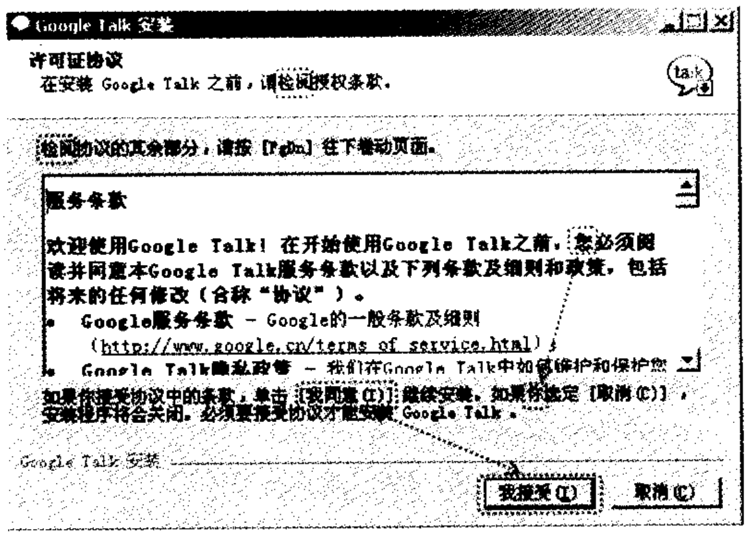


图 8-11 Google Talk 的安装界面中的缺陷

- “检阅”用词不准确。

- 用了“您”和“你”，宜统一用“您”。
- 按钮是“我接受（I）”，而叙述中错用“我同意（I）”。

8.3 I18N 和 L10N 测试工具

在软件国际化和本地化中，依然离不开软件工具，其中多数工具用于本地化工作（包括翻译内容的管理），测试工具并不多，这也许是由国际化和本地化的特点所决定的。在国际化和本地化测试中，设计评审和代码审查是最有效的手段，其次，手工测试也是不可缺少的，毕竟测试工具发挥的空间有限。

1. 国际化测试工具

OneRealm 的工具套件（*http://www.onerealm.com*）可以评估代码并识别国际化问题。

2. 本地化安装测试工具

- 本地化安装/卸载测试工具：InCtrl
- MSI 文件测试辅助工具：Orca
- ISO 文件测试辅助工具：Daemon Tools

3. 用户界面测试工具

Alchemy 公司的 Catalyst（*www.alchemysoftware.ie*）是一款功能强大、可视化的软件本地化工具，它遵循 TMX 等本地化规范，具有自定义解析器的功能，支持多种资源文件格式，适用于软件资源（Resource）文件本地化翻译和工程处理。Catalyst 所提供的验证专家（Validate Expert）可用于本地化测试，以检查各种类型的本地化处理错误，例如，热键重复、热键不一致、控件重叠、控件截断、拼写错误等。Catalyst 还提供了其他功能：

- Leverage Expert——重用本地化翻译资源。
- Update Expert——更新资源文件。
- Translator Toolbar——本地化翻译。
- Pseudo Translate Expert——伪翻译专家。
- Generate Report——生成字数统计报告。
- QuickShip Expert——打包项目文件。
- Extract Terminology Expert——抽取本地化术语。

Tool Proof 是由 Translation Craft（*http://www.tcraft.com*）开发的本地化测试工具，与

Catalyst 类似，可以显示用户界面元素和对话框，可以查看控件尺寸是否正确以及检查其他本地化 UI 问题。

4. 在线文档测试工具

SDL International 公司（Translation Craft 被其收购）的 HelpQA 和 HTML-QA 是用于分析大型帮助系统的测试工具，可以检查链接和跳转，识别其他本地化问题。如果出现由于本地化引起的编译错误还会发出警告。

HtmlQA（*http://www.sdl.com/products/htmlqa.htm*）用于测试源语言和本地化语言项目文件的本地化质量。HtmlQA 可以执行一系列本地化 Html 文件检查，以确定本地化的 Html 文件与源语言对应的 Html 文件具有一致的功能。

5. 翻译管理工具

翻译管理工具很多，主要有 TRADOS、SDLX、Wordfast、Catalyst、Passolo Software Localizer、LocStudio、Language Studio、Helium，RC-WinTrans、IBM Translation Manager、MultiTrans、Multilizer、Trans Suite 2000、Transit、Translator's Intuition、Visual Localize 等，国内的产品有雅信 CAT、华建 IAT 等。这些软件功能都比较强大，能满足软件本地化翻译工作的需要。例如，以 Passolo Software Localizer 为例，它的功能特性有：

- 可以直接将程序的 GUI（图形用户界面）本地化为用户需要的语言。
- 在可视化环境下可以对程序实行本地化处理。
- 内置有翻译记忆技术，可以重复使用现有的译文。
- 有模糊匹配技术，可搜索相似文本的译文，为翻译人员提供参考。
- 还可以与其他软件本地化工具交换数据，方便用户协同工作。
- 可以自动识别用户在本地化过程中出现的错误，并提出修复建议。
- 提供脚本引擎和自动化对象的接口，通过编写宏来扩展现有功能。

8.4 小结

国际化（I18N）是本地化（L10N）的基础和前提，本地化是国际化向特定语言环境的转换，其理想的状态是，源语言版本完全按照国际化版本的要求去做，本地化过程不需要修改源代码就能完成。然而，如果该软件不能充分地支持软件国际化特征，在本地化过程中就很有可能会产生新的功能方面的问题，包括功能调用出错、输入和输出等问题。

在国际化测试中，设计评审和代码审查是最有效的方法，验证软件是否在设计上具有

国际化软件的一些基本功能——用户时区、语言和地区等设置，确认程序代码和显示内容是否被分离。在国际化功能测试中，有两种方法——针对源语言的功能测试和针对伪翻译版本的测试。在源语言中，可以进行下列内容的测试：

- 时区的设置及其相应的时间显示。
- 地区的设置及其相应的日期、货币显示。
- 是否可以选择不同的语言。
- 是否可进行多字节字符串的输入。

而针对伪翻译版本的测试，可以完成下列测试任务：

- 多字节字符集和脚本是否正确。
- 多字节字符串的输入显示是否正确。
- 多字节字符文件、文件名、文件夹及其处理方式是否正确。
- 索引和排序是否正确。
- 本地化的操作系统、键盘支持等是否正确。

翻译只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、配置和兼容性等问题。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。

第9章

系统测试的执行

如果系统只满足了功能要求，没有满足一些非功能特性的要求，那么最终系统还是不能满足客户的需求。

为了区分功能测试与UI测试，一般将系统测试定义为非功能特性的测试，即针对系统进行负载测试、性能系统、容量测试、安全性测试、兼容性测试、可靠性测试等。系统测试的这种定义，和传统的定义有一定的差别，它越来越得到大家的认可。

虽然功能测试可以在单元测试、集成测试阶段进行，但功能测试的大部分工作是在系统集成测试完成之后进行的。在功能趋于稳定之后，才适合开展系统测试的工作。如果功能不稳定，随会就不得不做比较大的修改，则此前已做的性能测试可能就会失去意义。虽然负载测试、故障转移（fail-over）测

试等也受功能影响，但程度要低一些，可以较早地实施。所以，系统测试还是可以和功能测试并行推进，以缩短测试周期。

在进行系统测试时，强调测试环境的真实性。如果测试环境无法与系统的实际运行环境一模一样，也要尽可能逼近真实的运行环境。如果测试环境不能等价于系统的实际运行环境，系统测试的结果就值得怀疑。作为系统测试，不仅要考虑所依赖的运行环境，还要考虑与系统有关联的硬件、网络、第3方软件的影响。

9.1 如何进行系统测试

在测试计划与设计验证的过程中，我们已经对系统的性能、安全性、兼容性和可靠性等进行了充分讨论，而且对如何设计系统故障转移、安全性等测试用例，也进行了讨论。随着时间的推移，软件系统测试的目标、指标和用例越来越清晰。这时，可以改进系统测试的有效策略，采用适宜的测试方法和工具，以更有效地完成系统测试任务。

9.1.1 进一步了解系统测试

如果系统只是满足了功能要求，没有满足一些非功能特性（性能、安全性等）要求，还是不能满足客户的需求，不能获得用户的信任。如某个网站功能（注册、信息查询等）齐全，也可以被访问，但是，每打开一个页面都需要两分钟。结果，用户不能忍受，再也不会访问这个网站。这种非功能性的需求满足和功能性的需求满足同样重要。

为了验证系统是否符合非功能特性的质量需求而进行的测试是系统测试，如图9-1所示。例如，性能测试的目的之一就是为了验证当前系统实际所具有的性能。如果实际性能达不到系统使用的需求，就需要改进设计，优化算法或程序代码，直至达到要求。除了以上的目的之外，性能测试还可以进一步分为基准测试和规划测试。

- 对于新建立的系统，测试人员并不了解某些具体的性能指标，所以性能测试的首要任务就是获取这些指标的标准值，然后基于由这些标准值所设定的基准，进一步制定产品性能改进计划，也就是性能指标的变更需求计划。
- 产品最终要被部署到运行环境中，在部署之前要进行规划，例如，根据用户的数量或数据负载来决定服务器的选型和数量，如果10万个用户需要4台双CPU、内存4G的服务器，如果是100万个用户是否需要16台双CPU、内存8G的服务器等。这些规划的数据依赖于性能的规划测试。
- 容量测试可以看作是性能测试的一种，或者认为系统的容量是系统的性能指标

之一。如某个 Web 站点可以支持多少个并发用户、网络在线会议系统中与会者的人数。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。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上明确容量上的限制，避免引起软件产品使用的纠纷。

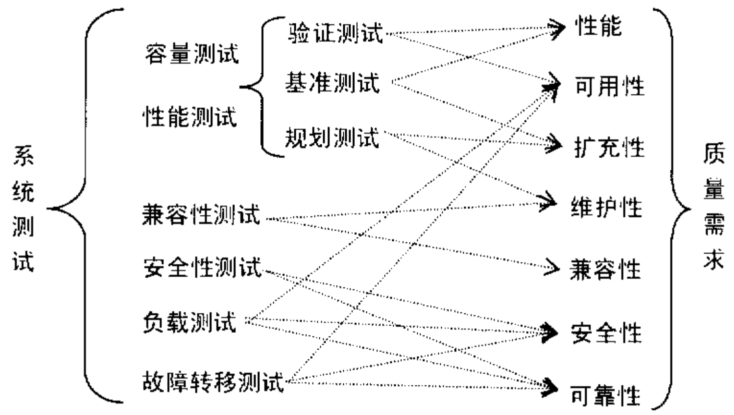


图 9-1 系统测试和质量需求的关系示意图

系统的兼容性测试要考虑软、硬件的兼容，就软件兼容来说，不仅要测试系统自身的版本兼容、用户已有数据的兼容，还要测试和第 3 方系统、第 3 方数据的兼容性。系统安全性测试，其方法虽与兼容性测试完全不一样，但同样需要考虑两者——软件系统自身和用户数据的安全性，忽视任何一方都是不可以的，如图 9-2 所示。系统安全总是相对的，就好比是矛与盾的关系，不能保证系统 100%安全或永远安全，所以安全性测试和软件系统的安全策略密切相关。

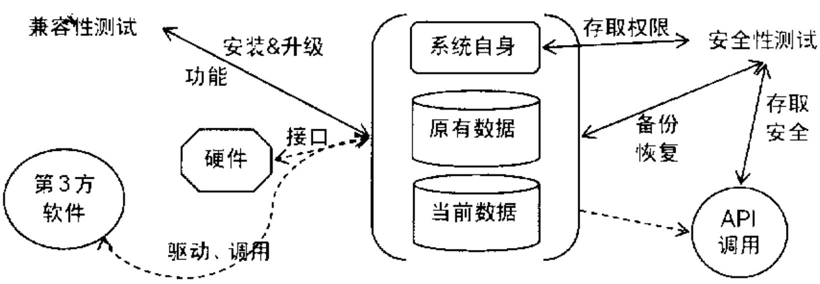


图 9-2 系统测试不可忽视的对象——数据、外部接口

概念

（1）**负载测试**（Load Test），也称压力测试（Stress test）、强度测试。负载测试通过模拟实际应用的软硬件环境及用户使用过程的系统负荷，逐渐加载或一次性加载，长时间或超大负荷地运行软件，以测试系统的稳定性，并试图找出系统性能的瓶颈和异常的地方等。通过负载测试，也可以确定系统的正常工作条件、极限条件等，并了解系统可靠性等，从而提高软件系统的可靠性、稳定性，减少系统的宕机时间。

（2）**性能测试**（Performance test），通过测试确定系统运行特性的性能指标数据，如数据吞吐量、响应时间、CPU使用率等。性能测试可以分为3类：

- 验证测试，针对系统验证事先（如产品规格说明书）已定义的性能指标。
- 基准测试，就是在系统标准配置下获得有关的系统指标数据，其测试结果应具有高度的一致性、标准性，可作为将来性能改进的基准线。
- 规划测试，是为软件部署而进行的测试，即在多种特定的环境下，获得系统不同性能的指标，从而决定在系统部署时采用什么样的软、硬件配置。

（3）**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，了解该软件系统的承载能力或提供服务的能力。系统在极限值状态下，主要功能还能正常运行。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（数据量、事件规模等）。容量测试可以看作负载测试和性能测试的组合。

（4）**安全性测试**（Security test），检验系统权限设置的有效性，防范非法入侵的能力，数据备份和恢复的能力等。例如，测试人员可以假扮非法入侵者，试图采用各种办法突破系统防线，修改权限或存取权限之外的数据。

（5）**容错测试**（Recovery test），检查软件在异常条件下是否具有防护性的措施或者恢复某种灾难性破坏的手段或能力。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，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或在指定时间间隔内恢复。

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。容错测试和故障转移（fail-over）、可用性测试等有直接的关系。

9.1.2 系统测试的实施策略

对于每一个应用软件系统，非功能特性的质量需求都是存在的，只是对于不同的应用系统，这些需求的程度、重要性不同。由于项目预算、时间的限制，所以不一定要完成所有的系统测试内容，而应该针对系统的应用特点及其需求，采取正确的策略实施系统测试，例如确定测试重点和测试的先后次序，有效地完成主要的系统测试任务。

1. 应用领域的不同需求

应用领域，对系统测试的需求差异很大。对于提供类似网上银行、信用卡服务的系统，安全性、可用性和可靠性等多方面的测试都是至关重要的，因为，任何一个严重的缺陷都会给用户造成比较大的损失或产生很坏的影响。这些系统需要得到充分的安全性测试、容错性测试和负载测试。多数情况下，还需要独立第三方的安全认证。

而对于局域网内的企业级应用来说，有关权限控制、口令设置等安全性测试依然重要，但兼容性测试就相对简单，因为可以指定某些特定的硬件和软件，如打印机只用 HP LaserJet，操作系统和浏览器只用 Windows XP 和 IE，无须对各种各样的硬件和软件进行兼容性测试。对于客户端软件，一般情况下，性能不是问题，而容错性、稳定性的测试则显得重要些。

2. 必要的测试工具

系统测试的方法，具有很强的针对性和技术性。不同的测试目标，其具体测试方法是不一样的，性能测试主要是模拟实际的操作行为，而安全测试是扫描系统所有物理的、逻辑的入口，包括通讯端口、URL、输入域等。但有一点是相同的，这就是，在系统测试中，靠手工测试来完成测试任务是很困难的，需要借助测试工具。系统测试工具比较多，有负载测试工具，也有安全检测工具等。

3. 合理的时间安排

前面已讨论过，系统测试一般滞后于功能测试，但也有很长一段时间，两者是并行执行的。仅对系统测试而言，各部分的测试在时间先后次序上，也有不同的要求，如图 9-3 所示：

- 负载测试在前，性能测试在后。
- 容错性在前，兼容性测试在后。
- 系统安全性测试在前，数据安全性测试在后。

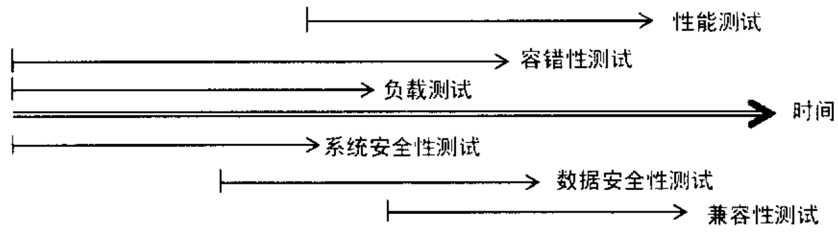


图9-3 系统测试的时间安排策略

4. 苛刻的环境要求

系统测试对环境要求苛刻，测试时硬件、网络、软件及其配置要尽量和实际运行环境保持一致。如硬件包括主板型号、CPU、内存、硬盘接口等，而网络则要考虑带宽、协议、负载平衡器、代理服务器、防火墙配置等。

示例

一个J2EE平台构造的企业级应用环境（如图9-4所示）

1. 硬件

- Web服务器：Dual-CPU xeon 2.8GB, 2GB DDR2 & ECC, 80GB HDD （SATA,7200）
- 数据库服务器： Dual-CPU xeon 2.8GB, 8GB DDR2 & ECC, 400GB HDD
- WebLogic应用服务器：Dual-CPU xeon 2.8GB, 8GB DDR2 & ECC, 200GB HDD
- WebLogic管理服务器：Dual-CPU xeon 2.8GB, 2GB DDR2 & ECC, 80GB HDD
- 负载均衡器：F5 BigIP 6400

2. 软件

- 操作系统：CentOS 4.4
- Apache
- Oracle 10G
- WebLogic 9.0

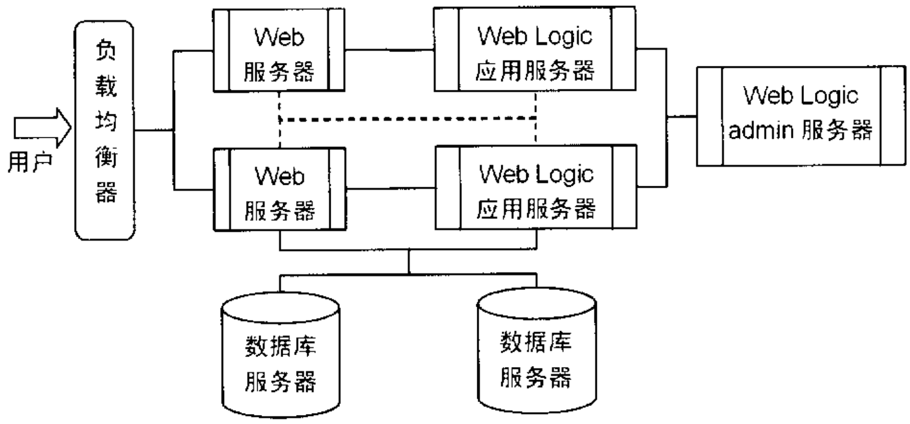


图 9-4 企业级应用环境的构造逻辑图（非网络拓扑图）

9.2 Web 应用服务器的负载测试

负载测试是在一种需要反常数量、频率或资源的方式下，对系统整体执行重复性的测试，以检查软件系统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，难怪在进行负载测试时，他们常常问自己："我们能够将系统折腾到什么程度而又不会出错？"。这种折腾系统的方式，就是对异常情况的设计。异常情况主要指的是峰值（瞬间使用高峰）、处理大量数据的能力、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：

- 如果中断的正常频率为每秒 1～2 次，则运行每秒产生 10 个中断的测试用例。
- 定量地增加数据输入量，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。
- 运行可能导致虚拟机崩溃或对磁盘进行大数据量存取操作的测试用例等。

9.2.1 负载测试的加载方式

要高质量地完成负载测试，必须通过测试工具准确模拟被测系统实际运行时所受到的真实负载。虽然这种模拟不可能和现实完全吻合，但借助一些方法基本上能做到这一点。例如，以并发用户为例，如图 9-5 所示，最常见的加载方法有 4 种。

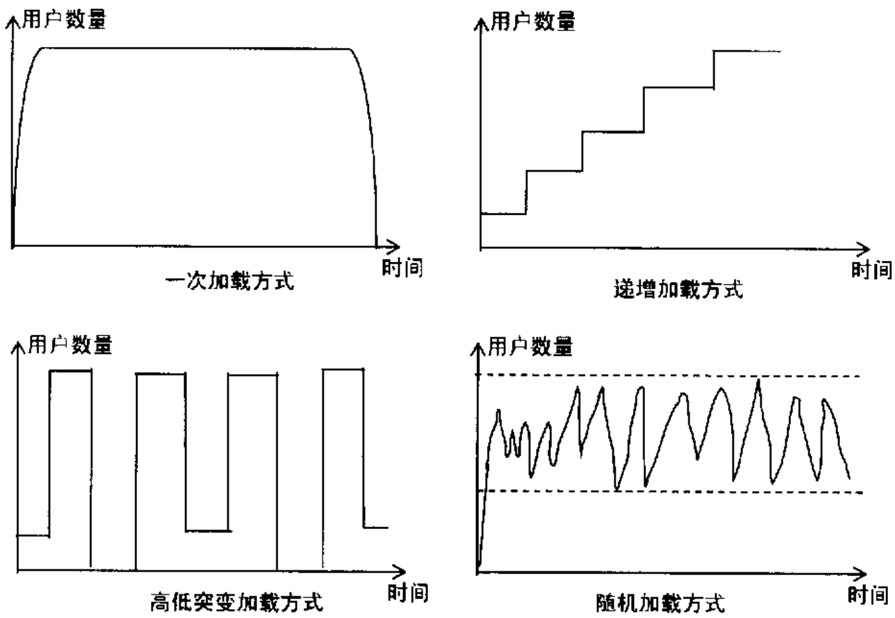


图 9-5 负载测试的几种加载方式

（1）一次加载。一次性加载某个数量的用户，在预定的时间段内持续运行。例如，早晨上班，用户访问网站或登录网站的时间非常集中，基本属于 Flat 负载模式。这种方式也称为扁平（Flat）模式。

（2）递增加载。有规律地逐渐增加用户，每几秒增加一些新用户，交错上升，这种方式又称为 ramp-up 模式。借助这种负载方式的测试，容易发现性能的拐点，即性能瓶颈的地方。

（3）高低突变加载。某个时间用户数量很大，突然降到很低，然后，过一段时间，又突然加到很高，反复几次。借助这种负载方式的测试，容易发现资源释放、内存泄漏等问题。

（4）随机加载方式，由随机算法自动生成某个数量范围内变化的、动态的负载，这种方式可能是和实际情况最为接近的一种负载方式。虽然不容易模拟系统运行出现的瞬时高峰期，但可以模拟长时间的高位运行过程。

9.2.2 负载测试的准备工作

测试环境准备包括硬件环境（服务器、客户机等）、网络环境（网络通讯协议、带宽等）、测试程序（能正确模拟客户端的操作）、数据准备等。虽然测试环境的准确配置，

在任何测试中都是重要的，但在负载测试中，在配置环境之前就需要关注负载测试中的输入参数（测试条件），例如系统关键业务流程、所承受的最大负载、负载模拟的持续时间和间隔以及负载测试的输出参数。

1. 负载用例——关键业务流程的确定

负载测试不可能执行所有的系统操作或不可能模拟所有的业务流程，所以要确定关键的业务流程和与其相关联的操作，这些业务流程的操作被设计为负载测试的用例，并作为系统负载测试输入的等价类。如果这些测试用例通过了，也就说明这个系统的负载测试通过了。

通过和用户、产品经理的沟通，了解系统的操作特点，如经常进行的业务类型、业务操作的频率。对于 Web 应用，以雅虎日历为例，可以选定一些主要的页面操作，如：

- 登录页面，容易发生在高峰期。
- 日历浏览页面（当日/本周）并打开已存在的事件，这是用户最常用的功能。

对于复杂的 Web 应用，可能存在多种角色，如网上会议系统，有主持人（host）和与会者（attendee），关键业务对于不同的角色要分别处理，以便最恰当地模拟最终用户的负载情况。如主持人需要导入通讯录、预订会议、浏览/修改/启动会议等，而与会者比较简单，主要是浏览会议、加入会议等。

2. 系统的最大负载

系统负载，可以根据产品说明书的设计要求和以往版本的实际运行经验来估算，也可以和产品经理讨论，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。那么事先预测满足要求的负载为 500 个用户的 1.5 至 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负载。

3. 负载模拟的持续时间和间隔

负载测试的持续时间对测试结果有影响，对于 Web 服务器来说，建议采用 8 个小时作为一次循环的时间。其次，用户进行操作时，不可能像计算机似的不停顿地操作，往往需要停顿或思考，这就是间隔时间，也被称为思考时间（think-time）。持续时间和间隔时间作为负载测试的参数，需要不断地变化、调整，用来考察系统所作出的反应。

4. 负载测试的输出参数

加载模式、关键业务流程、持续时间和间隔时间等，都是负载测试的输入参数，这还

不够，我们还要确定哪些参数是系统的输出参数，需要得到监控。因为，最终是通过输出结果来发现系统在可靠性、健壮性和可伸缩性等方面的问题的。作为 Web 服务器负载测试，一般要监控下列响应参数：

- 数据传输的吞吐量（Transactions）。
- 数据处理效率（Transactions per second）。
- 数据请求的响应时间（Response time）。
- 内存泄漏。
- CPU 使用率。

每个值都可以获得 4 个值——最小值、最大值、平均值和当前值，而且也可以将数据成功（Successful）和失败（Failed）的两部分分开考虑。许多负载测试工具还可以提供更多的监控指标，如图 9-6 所示。例如，增加下列一些项：

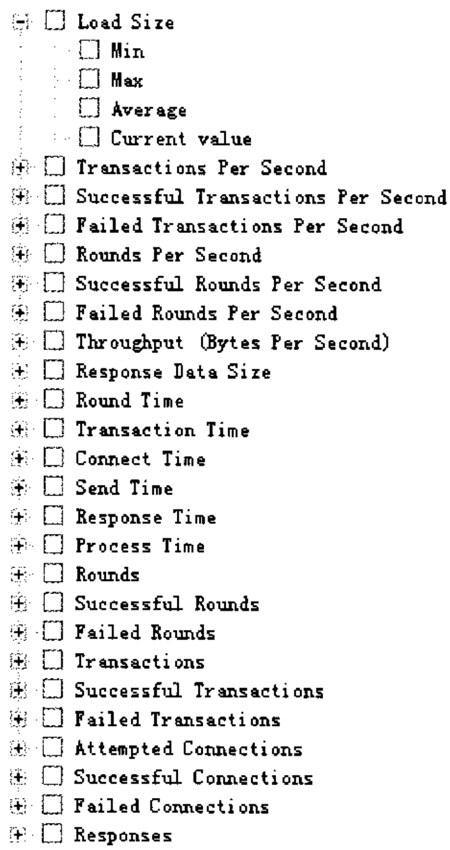


图 9-6 负载监控的各项指标

- 负载数据量（Load size）。
- 连接时间（Connect Time）。
- 发送时间（Sent Time）。
- 处理时间（Process Time）。
- 一轮来回时间（Round time）。
- 平均事务响应时间。
- 每秒事务总数。
- 每秒点击次数图。
- 每秒 HTTP 响应数。
- 每秒下载页面数。
- 每秒重试次数。
- 连接数。
- 每秒连接数。
- 每秒 SSL 连接数。
- 页面下载时间。
- 第一次缓冲时间。
- 已下载组件大小。

概括起来，负载测试的准备工作有：

（1）选定一种或多种加载方式。主要的加载方式有：一次加载、递增加载、高低突变加载和随机加载等。

（2）了解系统的用户角色，确定所要模拟的角色及其对应的关键业务操作路径。对用户组可以设置具体业务负载百分比，用以模拟不同用户组对被测系统造成的负载比例。

（3）确定模拟关键业务操作的负载持续时间和间隔时间，这应包含多种组合，以及循环次数等。

（4）需要的话，可选择随机选择器，模拟用户组内部各种随机业务操作（用例及其事件流）所占的不同负载比例。比例的调节，可以通过加权因子来实现。

（5）确定要监控的测试指标，包括数据吞吐量、响应时间、资源占有率等。

9.2.3 负载测试的执行

在作好充分的准备工作之后，执行就比较容易了。录制或开发测试脚本，通过相应的负载测试工具（如 Jmeter、WebLoad 等）来运行脚本，获得测试结果，并通过图表工具来分析结果。由于刚开始设定的负载量、持续时间、间隔时间甚至加载方式等都不一定合理，有时需要反复执行几次，不断调整这些输入参数，以达到预期的测试效果。负载测试执行的步骤，如图 9-7 所示。

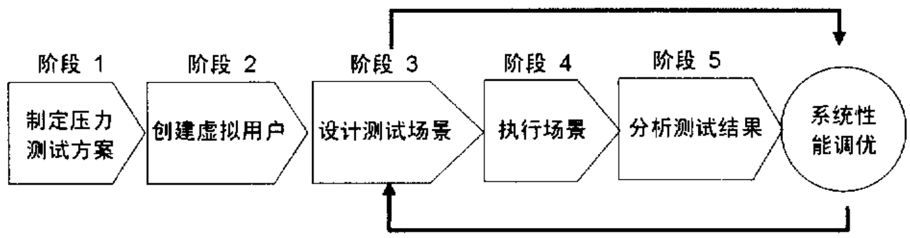


图 9-7 负载测试执行的步骤示意图

执行负载测试时，必须借助测试工具来模拟用户，而不是让成百上千的测试工程师亲自来执行测试。所模拟的用户，称之为“虚拟用户（Virtual Users，VU）”。大量的虚拟用户要运行在多个客户端，并由控制器管理、代理（agent）驱动。一般来说，一个客户端可以运行 10~100 个虚拟用户，如图 9-8 所示。

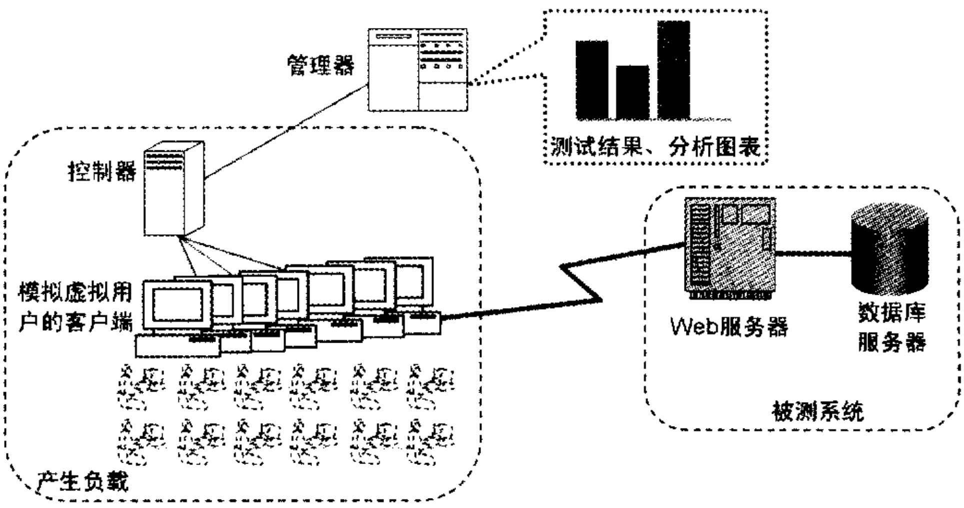


图 9-8 负载测试现场环境示意图

服务器上的负载受两个因素影响：同时与服务器通信连接（或虚拟用户）的数目，以

及每个虚拟用户请求之间的考虑时间的长短。很明显，与服务器通信的用户越多，负载就越大。同样，请求之间的间隔时间（也称思考时间）越短，负载也越大。这两个因素的不同组合会产生不同的服务器负载等级。

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了累积效应，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。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，在平常的测试中无法发现，最终却可能导致系统的崩溃。

负载测试通常采用黑盒测试方法，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，所以在负载测试执行中，适当的分析和详细的记录是十分重要的。

（1）查看服务器上的进程及相应的日志文件也许可以立刻找到问题的关键（如某个进程的崩溃）。如果程序不产生日志文件，就不能满足负载测试的要求，所以需要开发人员配合，临时增加后台写日志文件的功能，以供参考。

（2）查看监视系统性能的日志文件，记录问题出现的关键时间、在线用户数量、系统状态等数据。

（3）检查测试运行参数，进行适当地调整并重新测试，看看是否能够再现问题。如果可以重现，可以进一步分解问题，屏蔽某些因素或功能，再看看能否重现问题。如果又能重现，问题产生的真正原因可能就被找到了。

概念

（1）**稳定性负载测试**。在选定的负载值下，持续运行24小时以上，以进行稳定性测试。客户端通常由测试工具模拟真实用户不停地进行各种操作。监视服务器和真实客户端的必要性能指标。这类负载测试的质量标准指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。

（2）**破坏性负载测试**。在稳定性负载测试中，可能会出现一些问题，如系统性能明显降低，但仅仅依靠稳定性负载测试，又很难暴露出其真实的原因。通过破坏性测试，采用不断加载的手段，往往能快速造成系统的崩溃，让问题尽快地暴露出来。

（3）**渗入测试**。为了检验内存泄漏、垃圾收集（GC）或系统的其他问题，渗入测试（soak test）使用固定数目的并发用户来测试系统的总体健壮性。虽然渗入测试是一种比

较简单的负载测试，但测试所需时间较长（几天甚至几周的连续运行），因为只有通过长时间运行，才能发现这样一类问题。测试运行的时间越久，对系统就越了解。因为连续测试时间很长，必须通过工具进行准确地监控，并动态地记录下时间序列的性能数据。

（4）**峰谷测试**（peak-rest test），顾名思义就是测试在系统使用高峰期的性能情况。这种测试一般采用高低突变加载方式进行，先加载到高水平的负载，然后急剧降低负载，稍微平息一段时间，再加载到高水平的负载。重复这样过程，并监控这个过程，了解第二次高峰的性能指标是否和第一次高峰的性能指标十分接近？下一次的性能指标是否与上一次的性能指标吻合？如果系统存在问题，通过比较，就容易发现一些蛛丝马迹。

9.2.4 负载测试的结果分析

在测试过程中，要善于捕捉被监控的数据曲线发生突变的地方——拐点，这一点就是饱和点或性能瓶颈。例如，以数据吞吐量为例，刚开始，系统有足够的空闲线程去处理增加的负载，所以吞吐量以稳定的速度增长，然后在某一个点上稳定下来，即系统达到饱和点。在达到饱和点后，所有的线程都已投入使用，传入的请求不再被立即处理，而是放入队列中，新的请求不能及时被处理。因为系统处理的能力是一定的，如果继续增加负载，执行队列开始增长，系统的响应时间也随之延长。当服务器的吞吐量保持稳定时，就表示达到了给定条件下的系统上限。这个结果，可以通过图 9-9 给出清晰的描述。

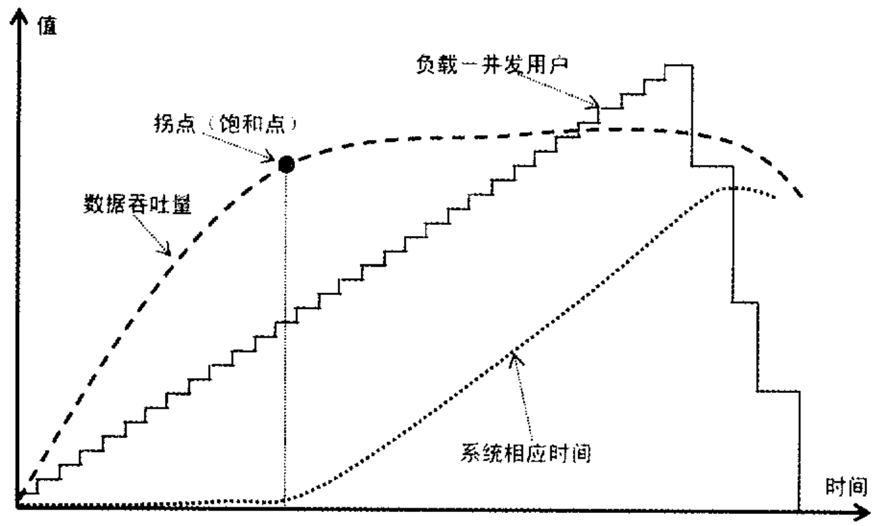


图 9-9 系统吞吐量、响应时间随负载增加的变化过程示意图

如果继续加大负载，系统响应时间可能会发生突变，即执行队列排得过长，无法处理，

服务器接近死机或崩溃，响应时间就变得很长或无限长。但这种极限点有参考价值，可帮助改进设计和系统部署，但不应该作为正常的控制点。正常的控制点，应该是饱和点。

分析负载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。对于 Web 服务器的测试，可以重点分析 3 项参数：

- 页面性能报告显示每个页面的平均响应时间。
- 响应时间总结报告（Response vs. Time Summary）显示所有页面和页面元素的平均响应时间在测试运行过程中的变化情况。
- 响应时间详细报告（Response vs. Time Detail）即详细显示每个页面的响应时间在测试运行过程中的变化情况。

9.3 Web 应用服务器的性能测试

在对负载测试作了全面介绍之后，理解性能测试也就不难了。虽然负载测试和性能测试不是一回事，但负载测试和性能测试有许多相似的地方：

- 测试方法比较接近，而且多数情况下可以使用相同的测试工具。
- 借助测试脚本来模拟用户的操作过程和负载变化的过程。
- 测试环境一致，都是由管理器、控制器、虚拟用户客户端等构成的。
- 在测试过程中关注系统的性能。

所不同的是，性能测试对加载有非常严格的要求——会有几个特定的负载值，而且事先所定义的性能指标也很明确。负载测试的重点在于发现功能性测试不易发现的系统方面的缺陷，而性能测试更着力于提供系统性能指标方面的数据。

Web 应用服务器的性能测试，对系统的负载有明确的要求，如并发用户的数量为 5、50、100、200、500、1000 等，非常精确。对性能指标（响应速度、内存/CPU 占用率等）事先也有明确规定，如果系统性能测试的实际结果不符合指标要求，则需进一步分析、优化，最终使系统的性能指标达到要求。

1. 基准测试

在实施基准测试时，一次只改变一个输入参数，这样获得的结果会很可靠。例如，要测试 Web 服务器的内存对系统性能的影响，可以逐渐递增内存，从 1GB、2GB、4GB 到

8GB 等。每次测试，都要认真收集，记录环境数据和测试结果。测试完内存对性能的影响之后，再测试多 CPU 对性能的影响。运行一系列这样的基准测试，确立一个已知的可控环境，然后可以进行环境更为复杂的性能测试。

2. 容量测试

容量测试不能只考虑用户数，需要和其他因素共同考虑，如用户发出请求的间隔时间。因为间隔时间越短，系统所能支持的并发用户就越少。假定每个用户平均发出请求的间隔时间是 1 秒，系统可以支持 500 个并发用户，那么当平均间隔时间是 20 秒时，系统则可能支持近万个并发用户。所以间隔时间设定，对容量测试显得非常重要。要确定间隔时间，可以先做些试验，获取较多的样本值，然后取其平均值。

第一次容量测试可以采用递增加载的方式，先获得一个基本的容量值或容量范围，然后再考虑用一次性加载的方式，来决定实际可支持的容量值。两种加载方式，也可以交叉使用，虽然没有很好的可比性，但相互参考（取其两者的均值），对获取更准确的容量会有帮助。

另外一个例子，就是数量不同的用户组的影响，如网上聊天室软件。同样是 1000 个用户，有不同的组合：

- 在一个聊天室内有 1000 个用户。
- 100 个聊天室每个聊天室内有 10 个用户。

在服务器端数据处理量和传输量是不同的，所以会表现出性能上的差异。

3. 实例

使用 Jmeter 对雅虎日历进行测试。根据测试用例的要求建好所有的 HTTP 请求，并配置所要求的参数。然后将线程组里的线程数量设置成希望模拟的用户数量，比如：

- 线程数为 500。
- 递增加载周期（ramp-up period）：2 秒。
- 循环次数：20 次。

然后启动并运行，获得测试结果。从 Jmeter 测试报告中，如图 9-10 所示，可以看到在用户数量为 500 的情况下页面的平均响应时间、最小响应时间、最大响应时间等信息，没有页面出错，测试结果基本在质量允许的范围内。

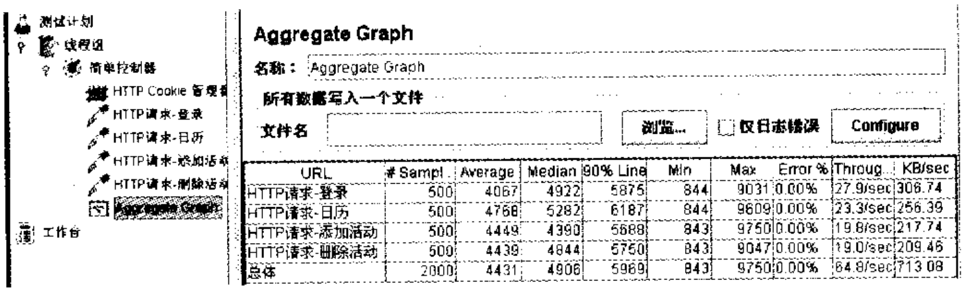


图 9-10 雅虎日历页面性能测试结果报告

通过设置线程组的数量，可以获得不同用户数量下的测试数据，然后绘制成如图 9-11 所示的曲线图进行更直观的分析。通过该曲线分析可以看出，各页面的响应时间随着并发用户数量地不断增加而增加。当并发用户数量达到 1000 时，各个页面依旧工作正常，只有添加和删除活动页面的响应时间达到了 10 秒左右。同时，可以看出登录页面的性能始终是最高的，这可能是因为登录页面简单、对数据库的访问量最少。

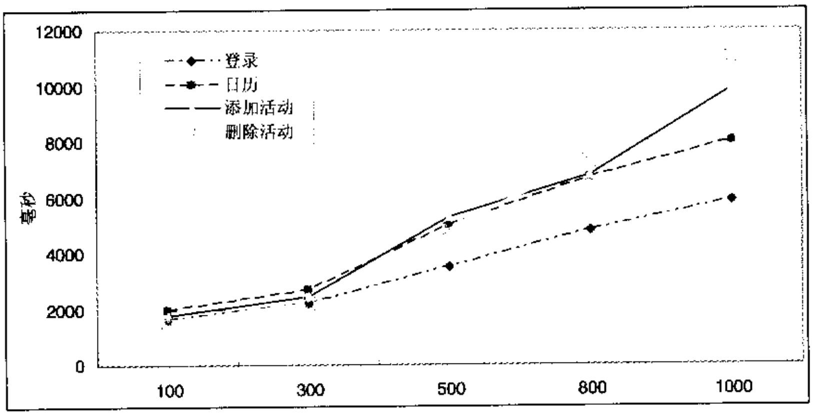


图 9-11 不同用户数量下页面的平均响应时间

9.4 Web 安全性测试

在 4.3.2 节中讨论了安全性测试用例的设计，这里以雅虎日历为例，简单介绍如何进行 Web 安全性的测试。

1. 验证 Yahoo 网站是否存在 SQL 注入式攻击的漏洞

如果用户登录通过下列程序实现，可以看出，程序在得到用户的数据后，并没有进行任何的过滤和处理，而是直接用来构成 SQL 语句，其危险性是最高的，攻击者只要根据

SQL 语句的编写规则就可以跳过身份认证。

```
Username=Request.from("username")
Password=Request.from("password")
Sql="select * from admin where username='"&usename &' and
password='"&password &"'"
Rs.open sql.com.0.3
If not rs.eof then
        Session("login")=true
        Response.redirect("next.asp")
End if
```

攻击者只需要在用户名的输入框中输入“ ‘or ‘1’=’1”就可以不经过认证而直接进入页面。但在雅虎日历登录页面进行同样输入的尝试，系统提醒该用户不存在，说明雅虎日历首先对用户名进行了认证，不存在类似的安全漏洞。再仔细分析雅虎日历登录页面的部分代码，说明采用了 md5 加密和认证口令，这使系统的安全性更高。

2. URL 和 API 的身份验证

首先以正常的用户登录雅虎日历，登录的时候不要选中“记住我的账号”的选择框。当用户登录后，点击雅虎日历的“效率手册”到日历的主页面，拷贝浏览器里面的链接，然后在一个新打开的浏览器窗口尝试直接打开这个 URL，但没有成功，系统提示用户登录。测试结果表明雅虎日历对 API 采取了 cookie 验证机制，不允许绕过登录直接来处理页面的操作。

如果在一个新打开的浏览器中，尝试直接打开“活动添加”页面的 URL（*http://cn.calendar.yahoo.com/YYY,a2ef70/srt,0/?v=33&pv=61&t=1177842436*），结果是成功了，没有提示用户登录。但是，当用户点击“保存并继续添加”按钮的时候，雅虎日历却开始提示用户登录。这说明了雅虎日历里面不是每一个页面都采取了验证机制，这会导致出现更严重的问题。比如，此时退出雅虎日历，关闭浏览器，然后再打开浏览器，输入雅虎日历的 URL: *http://cn.calendar.yahoo.com/*。现在看到了什么？没有提示用户输入用户名和密码就直接登录了雅虎日历。

3. 跨站脚本攻击

在雅虎日历里面添加一个活动或者事件，然后在“便笺”里面输入一些 JavaScript 脚本，接着设置该活动为公开的，允许其他用户查看这个活动。这时在当前页面上，有一个链接供用户拷贝，可以把它发给其他用户以了解这个活动。但这个链接中，就包含了 JavaScript 脚本。虽然脚本没有执行，但允许在“便笺”里面输入 JavaScript，这就隐含了被攻击的危险。

4. 其他 Web 安全测试

浏览器缓存、认证和会话数据不应该作为 GET 的一部分来发送，应该使用 POST。借助 HttpWatch，可以看出雅虎日历采用的安全方式——Post 方式。

在抛出异常的时候，系统给出比较详细的内部错误信息，这容易暴露不应该被显示的执行细节，从而存在安全隐患。下面针对这点对雅虎日历进行测试。如将“添加待办事项”页面的链接：*http://cn.calendar.yahoo.com/YYY,9a85ab/srt,0/?v=33&pv=61&t=1177850907* 中的参数 V=33 改成－1，然后重新打开已修改的 URL，结果页面就出错了。但是雅虎日历封装了出错的信息，没有给出 Server 出错的具体、详细信息。

9.5 容错性测试

容错测试是一种对抗性的测试过程。在这种测试中，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、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。

我们已经知道，容错性包括两个方面，一方面对输入异常数据或操作有保护作用，另一方面，当系统确实出错了时有自我恢复的能力。对于后者，故障转移也可以有效解决单个系统不能正常工作的问题，前面已作了比较多的讨论，这里着重讨论对输入异常数据或操作的测试。

以 Google Talk 和雅虎日历为例，有哪些异常数据或异常操作呢？

1．异常数据输入

- 在 Google Talk 中输入很长的超文本内容发送出去。
- 在雅虎日历中，输入事件或活动名称，使其长度超过其可输入字符长度的上限。
- 在雅虎日历或 Google Talk 中，上载一个很大的文件。
- 在 Google Talk 的“设置”中，设置字体为最大字体，结果就出现问题，如图 9-12 所示，“忘记了密码？”没有被完全显示出来。而且字体不统一，有些内容是大字体，有些内容的字体没变。

2．异常操作

- 在 Google Talk 正在运行并不断发送聊天信息的过程中，通过网络工具来模拟网络中断。

- 运行 Google Talk，在用户甲发送聊天信息给用户乙时，使用户乙客户端的网络中断，观察用户甲客户端的反应现象。
- 在雅虎日历中输入一个新活动或事件的信息，在保存该活动或事件信息时，通过网络工具来模拟网络中断。
- 在没有声卡的计算机上设置音频选项。
- 运行 Google Talk，让用户甲的计算机没有声卡，用户乙音频正常，这时请求 VoIP 通讯。

图 9-12 字体设置后的界面

技巧

在异常操作时，也可以通过停止服务端的进程和服务来测试客户端的容错性和服务器端的自我恢复能力。在Windows系统平台上，可以直接使用“管理工具”的“服务”管理功能来停止某个服务。在Linux服务器上，可以使用两个命令：

- 使用命令“pgrep –fl sub-string”列出目前在运行的进程。
- 再使用“kill -9 进程号”杀掉（停止）某个进程。

9.6 数据库的性能测试

数据库容量是表征数据库服务器性能的一个重要标准，数据库中存在大量数据时，系统的性能肯定会受到影响。例如，当数据库中存在几十万、几百万条记录时，进行数据库操作（记录更新、查询等），数据库表的索引定义、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，包括 CPU 的使用率。

在数据库性能测试过程中，还需要考虑另外一种大容量数据的测试。例如，可以做这样的测试，在数据库中插入 1000 条图片文件，每个图片文件的大小为 4MB，这样在插入这些数据后，数据库的容量为 4GB。然后，再通过在空表中插入 1000 条数据、选择 900 条数据、更新 900 条数据、删除 900 条数据等操作过程来进行测试，跟踪和分析所需要的时间，进程占用的 CPU，内存信息等，以获取数据库的性能情况。

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。

这里通过 3 张表格，描述数据库性能测试中最重要的内容：

（1）主要数据库性能测试的监控指标，如表 9-1 所示。

（2）对数据库测试工具进行了全方位的比较，如表 9-2 所示。

（3）常见的数据库性能测试用例，包括被考察的、相应的性能指标，如表 9-3 所示。

表 9-1 数据库性能监控的主要指标

指标名	说　明
SGA- system global area	数据库系统全局区的大小（Oracle 中的 SGA，System Global Area）。SGA 存放的是数据库中所有用户的共同信息，SGA 的大小会严重影响数据库的性能
Logical I/O	每秒逻辑读/写的次数。当我们向数据库作任何的操作时，Oracle 都会先到 SGA 中查找一下有没有我们所需要的数据，如有就即时返回，没有再通过 Physical I/O 来向 data file 读数据，从而减少 Physical I/O 的读写
Buffer Cache Hit Ratio	共享池的命中率。这是衡量 Logical I/O 的一个指标，如果命中率低，表明 Physical I/O 的读写次数必然增多，数据库性能必然下降。一般命中率都在 95% 以上，低于这个比率说明缓存区的设置太小

续表

指标名	说 明
DBWR （N）	数据库磁盘写进程的数目。当并发写多数据的时候，如果 DBWR 的数据不足以应付，DBWR 写进程耗时必然多，容易产生等待时间或锁的时间
Buffer Cache	数据共享区的大小。如果 Buffer Cache 太小，会造成 Physical I/O 过多
LGWR （N）	重做日志文件写进程的数目。磁盘 I/O 过多是会严重影响数据库性能
Redo Buffer Cache	重做日志缓存区的大小。这个缓存区的大小会影响到 LGWR（N）的读/写次数

表 9-2 数据库测试工具比较

工具名称	OraRap	DBMonster	Quest Toad	Loadrunne +Spotlight
简介	PL/SQL 性能监控工具，SHELL 程序，支持 Oracle	用随机生成的数据来测试 SQL 数据库的压力测试工具，它来源于 Java 的开源项目	基于 Windows 的 PL/SQL 开发工具	用 Loadrunner 录制和执行访问数据库的脚本，用 Spotlight 对数据库进行性能分析，支持 Oracle、DB2
特点	格式良好的 HTML 报告；提供数据库 instance 的总体概况；数据库性能优化的提示；方便地在一个 Cron 任务中设置循环的执行；有选择地创建一些独立的以图表形式显示测试结果的页面报告	通过 JDBC 方式连接数据库；协助产生大量规则或不规则的数据；通过两个 XML 文件（配置文件和 schema 文件）控制数据产生的行为	图形式的操作接口；编辑器能同时编辑多个不同类型的文件；透过整合性 Result Set、Explain Plan、tracing/DBMS_OUTPUT 的运用，有效地简化修改与测试的流程	Spotlight 是一种图形化的实时性能诊断工具，提供了全面的数据库配置、运行状态、性能状况等信息

续表

工具名称	OraRap	DBMonster	Quest Toad	Loadrunne +Spotlight
优点	很好地控数据库中SQL语句的执行情况，并有针对性地给出TOP SQL的报告，便于分析SQL是否存在性能问题	Java脚本，跨平台支持各种Jdbc驱动的数据库；自动产生负载数据	Windows平台运行，可视化界面。集开发与数据库监控于一体。具备LiveReorg功能的表空间管理、SQL Tuning，数据库管理、分析功能	Loadrunner模拟应用程序对数据库的频繁访问，从而增强数据库负载测试。Spotlight实时监控，能够真实地反映数据库负载状况
缺点	只能在UNIX/Linux下运行，不支持Windows；只支持ORACLE；不能自行生成测试数据	不能进行性能分析，只能模拟数据库测试环境	占用空间及内存都比较大。单项的数据库监控数据不够详细	

表 9-3 常见的数据库性能测试用例

用例编号、名称	测试目标	测试的数据库性能指标
1.模拟多人同时并发新增多记录到**单表**	在左边场景（插入记录、修改记录或删除记录到单表、多表）下，写表所需要的时间，所建立的表空间是否合理，回滚段的使用情况	表空间，DBWR（N）和LGWR（N）进程，Buffer Cache，Redo Buffer Cache，Redo Generation（重做日志的生成时间），磁盘写的速度，Rollback Segment Logical I/O，磁盘数据空间、索引空间的大小变化
2.模拟多人同时并发新增多记录到**多表**		
3.模拟多人同时并发修改多记录到**单表**		
4.模拟多人同时并发修改多记录到**多表**		
5.模拟多人同时并发删除**单表**中多个记录		
6.模拟多人同时并发删除**多表**中多个记录		
7.模拟多人调用修改性的存储过程，进行多记录运行	在左边场景下，了解CPU占用情况，内存使用量，I/O设备访问速度，回滚段使用情况	

续表

用例编号、名称	测试目标	测试的数据库性能指标
8.模拟多人同时并发从单表中查询多个不同记录	在左边场景下，读表所需要的时间，所建立的表的索引是否合理，对缓存区的命中率	磁盘读的速度，命中率，Parallel query, Logical Reads, Miss Rate，Block wait
9.进行模拟多人同时并发从多表中查询多记录		

最后，通过两个图来展示数据库性能测试的结果：

- TOAD 直观的图形显示，如图 9-13 所示。
- Spotlight 丰富的界面，如图 9-14 所示。

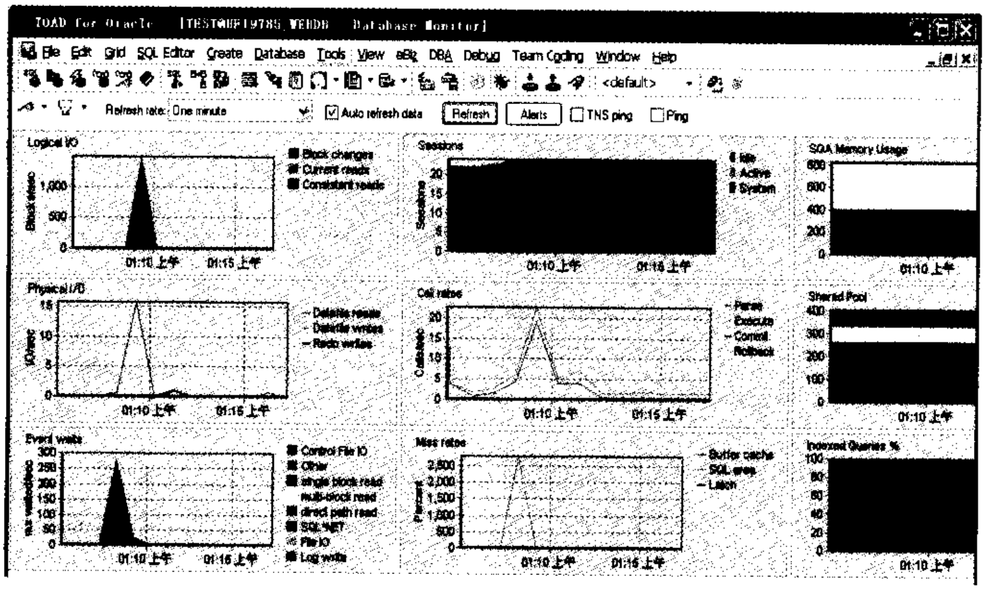


图 9-13 TOAD 结果显示界面

对于 Spotlight 所显示的结果，屏幕中央是数据库不同组件之间数据的流量及比率，其中包括 SESSION、SERVICE、SGA、PROCESS、DATAFILE 等，单击任意一部分则显示针对该部分的详细监测图表。

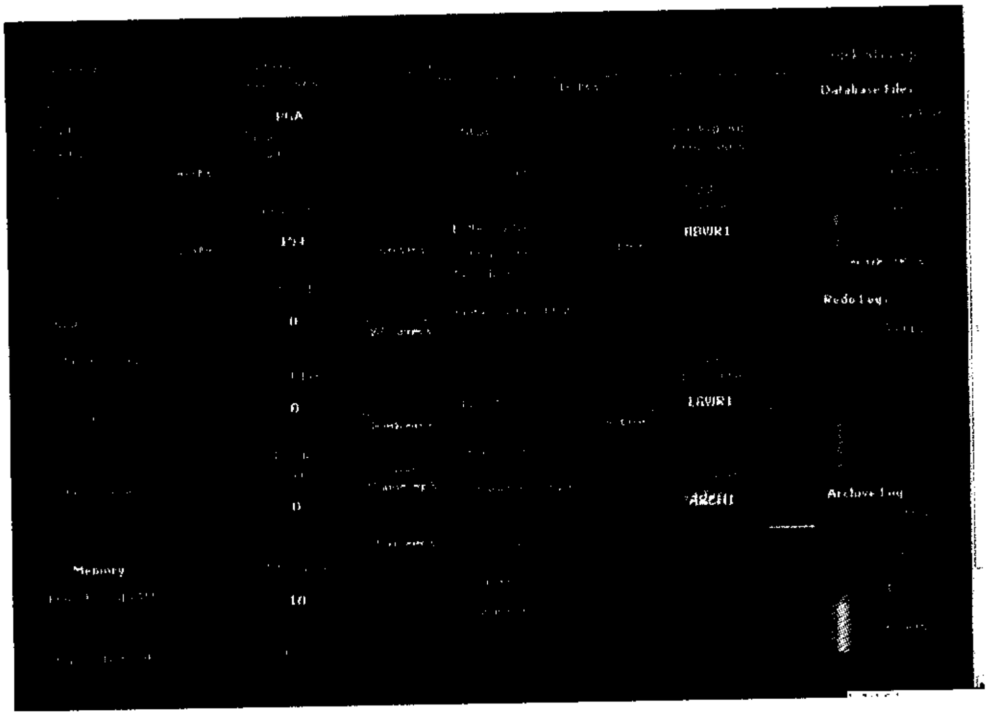


图 9-14 Spotlight 结果显示界面

9.7 兼容性测试

兼容性测试包括了软件兼容性、数据共享兼容性、硬件兼容性三个方面。例如，针对一个新开发的图像处理软件，自定义一种特殊的图形存储格式以适应特殊的应用。那么，兼容性测试就是试图回答下列问题：

- 该软件是否能在不同的操作系统上正常工作？
- 是否可以将图像文件转换为.bmp、.gif 或.jpg 格式的文件？
- 是否可反过来读取.bmp、.gif 或.jpg 格式的图像文件，并转换为自己新定义的格式？
- 是否支持市场上流行的显卡？
- 是否符合图像处理的业界标准？

1. 向前和向后兼容性测试

虽然在软件系统自身上也需要考虑向前兼容和向后兼容，包括软件升级和要求服务器端软件新版本对软件客户端旧版本的支持，但向前兼容和向后兼容测试主要是在数据兼容

性上进行的测试。

以微软的 Office Word 产品为例来说明数据的兼容性。Office Word 2003 的版本如何与此前 Office Word 98/2000/XP 等多个版本的文件兼容，以及如何与 Office Word 2007 产生的文件兼容：

使用 Office Word 2003 打开此前版本产生的 Word 文件，经测试是没有问题的。如何让 Office Word 2003 打开 Word 2007 产生的文件？实际有多种方法，例如：

- 最佳方法。Word 2007 可以将自己的文件另存为 Word 2003 格式的文件，这就是提供了“数据导出”功能。
- 较佳的方式。把文件转换成 XML 格式或 HTML 格式的文件，大部分格式都会被保留下来。
- 不够理想的方法。用 Word 2007 将文件另存为 RTF（Rich Text Format）格式的文件，所有的 Word 软件，都可以打开这种格式的文件，而且还保留了像字体、大小、下划线、居中等已有格式。这也就是为什么当初设计 RTF 格式文件的原因。

但在 Office Powerpoint 2003 中还是出现一个兼容性缺陷，如果由 Powerpoint 2000/XP 创建一个.ppt 文件，然后由 Powerpoint 2003 打开并做了一点修改。保存之后，Powerpoint 2003 却打不开这个文件。后来微软不得不发布了一个紧急的补丁包来修复这个缺陷。

概念

数据的向后兼容（backward compatibility）和向前兼容（forward compatibility），可以描述为：

- *向后兼容是指新发布的软件版本可以使用该软件以前的版本所产生的数据。*
- *向前兼容是指在设计和开发软件一个新版本时，考虑如何与未来版本的数据兼容。*

当然，并非所有软件都要进行向前兼容或向后兼容测试，在进行软件设计时就决定了产品的特性，测试人员根据产品特性决定是否进行相应的测试。向后兼容是必须满足的质量特性，向前兼容没有必要性，但最好要考虑，以减少带给未来版本的麻烦。

2. 和第三方的数据兼容

软件自身不仅要做到数据兼容，而且要做到和第三方数据的兼容。和第三方数据兼容的方法主要有 3 种：

（1）遵守统一的标准：遵守业界认可的事实标准或国际标准，并存储在介质上，能

被第三方同类软件正确地读取。如 PDF 文档格式、XML 扩展的媒介语言 XML、图像的 GIF 或 JPG 格式。

（2）提供数据导入和导出，可以导入第三方同类软件产生的数据，也可以导出自己的数据供不同的软件使用。如某个数据库管理系统软件，能导入竞争对手的数据库管理系统中的数据；许多网上交流的软件，可以导入 OutLook 或 Lotus Notes 地址簿中的邮件地址；微软的 Excel 软件可以导出.csv 数据格式的文件，供数据库软件 Access 或 SQL server 导入。

（3）剪贴板或 ODBC 等类似方法：通过 OLE（对象链接和嵌入）、ODI（开放式数据连接接口）、ODBC（开放式数据库互接）等各种技术来实现在不同的软件之间的数据共享。

以 Excel 软件数据和其他软件（Word、浏览器、邮件客户端、数据库等）的数据兼容渠道为例，如图 9-15 所示。

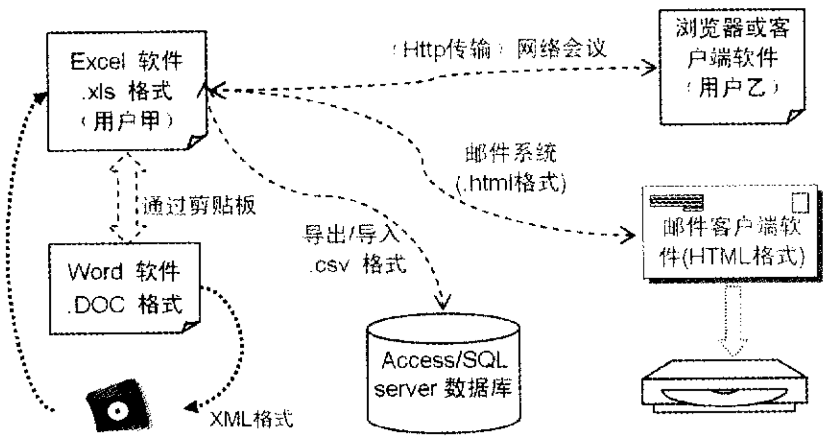


图 9-15 Excel 数据和其他软件兼容的各种形式

3. 多版本的测试

在软件服务和部署中，无论在服务器端还是在客户端，多版本共存的机会越来越多，所以对多版本的兼容性测试不可忽视。如最典型的例子，远程控制和存取软件（Access Anywhere）是由服务器、代理和客户构成的一个系统。由于已部署的服务器多达几千台，不可能在很短的时间内完成所有新版本（假定是 3.0 版本）的部署，这时 2.0 版本还存在。客户端也有同样情况，有些用户没有及时更新客户端软件，而有些用户及时更新了客户端软件，但远程代理不能及时更新，需要保留 24 小时，这样可以保证客户能及时控制远程

机器，如图 9-16 所示。

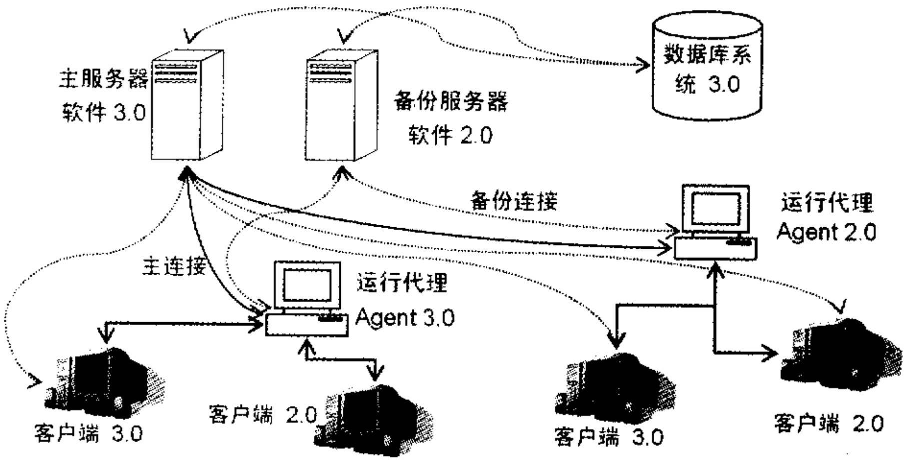


图 9-16 由多个部分构成典型的系统兼容性例子

这样，兼容性测试的组合比较多，可以通过表 9-4 来描述。对于第 1、5 两种情况（代理版本是 3.0 而客户端版本是 2.0）相对比较少，也可以强迫客户端升级。

表 9-4 系统兼容性测试的组合情况

<table>
<tr><th>序 号</th><th>数据库版本</th><th>服务器版本</th><th>代理版本</th><th>客户端版本</th></tr>
<tr><td>1</td><td rowspan="8">3.0</td><td rowspan="4">3.0</td><td rowspan="2">3.0</td><td>2.0</td></tr>
<tr><td>2</td><td>3.0</td></tr>
<tr><td>3</td><td rowspan="2">2.0</td><td>2.0</td></tr>
<tr><td>4</td><td>3.0</td></tr>
<tr><td>5</td><td rowspan="4">2.0</td><td rowspan="2">3.0</td><td>2.0</td></tr>
<tr><td>6</td><td>3.0</td></tr>
<tr><td>7</td><td rowspan="2">2.0</td><td>2.0</td></tr>
<tr><td>8</td><td>3.0</td></tr>
</table>

9.8 安装测试

许多测试人员认为软件安装测试比较简单，所以安装测试容易被忽略。大家可能还记得有关多媒体光盘游戏“狮子王童话”的故事。

故事

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，但对迪斯尼公司来说，这还是第一次进军这个市场。由于迪斯尼公司的品牌著名，并且事先进行了大力的宣传和良好的促销活动。所以，市场销售非常不错，该游戏成为父母为孩子过圣诞节必买的礼物。

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，客户支持部就淹没在家长愤怒的责问声和玩不成游戏的孩子们的哭诉声之中，报纸和电视也开始不断报道此事。

后来证实，迪斯尼公司没有对当时市场上的各种PC机机型进行完整的测试，而只是在2～3种PC机机型上进行了相关测试。结果，导致这个游戏软件在其他一些常见系统中不能安装。

9.8.1 客户端软件安装测试

1. 安装前

考虑不同的机器环境，如安装过先前版本的或从来没有安装过的机器；安装过许多应用软件的崭新机器。如果计算机已存在某个旧版本，这时有两种方式：先卸载再安装和自动安装。**不管是哪种方式，一定要检查用户的数据，确保原有的用户数据不被删除，完好无损**。数据比系统重要，安装测试时，这个测试点不能放过。

在崭新的机器上进行安装测试也是必要的，因为用户的计算机环境和技术人员的计算机环境差异很大。

实例

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发生了意外，在按安装说明书进行安装时，总是不能成功。测试经理对测试结果产生了很大的疑问，经过再三分析，发现真正的原因就是测试人员的测试用机已安装了Java实时运行环境程序（JRE），而安装该软件需要Java运行环境，但在安装过程中既没有提示安装JRE，也没有判断机器是否已安装JRE，从而导致在客户机器上安装失败。

2. 安装测试过程

在安装测试过程中，要严格按照安装说明书或相关文档进行。如果是电子文档，所使用的命令应该用“copy/paste”来完成，以保障安装文档得到100%的测试。除此之外，还有其他一些要点：

- 安装的容错性，即安装过程中是否会出现不可预见的或不可修复的错误；若出现错误，是否有提示；若出现错误，是否能退出；系统安装中止、卸载后是否可恢复到原来状态。
- 安装的灵活性，即是否提供了多种安装模式（快捷的、用户定制的、完整的）；安装步骤是否可以倒退（退到前一步）；用户是否可以中途退出安装；是否能选择、更改安装目录。
- 易安装性，即安装过程是否简单、清楚；是否有简明、正确的提示信息。
- 安装程序是否占用过多的系统资源；是否与其他系统有冲突；是否影响整个系统的安全性；是否会破坏其他文件或配置。
- 安装过程中是否对硬件有自动识别能力。
- 应按照文档去做，不要根据自己的理解来改变安装的步骤和设置。
- 在安装文档中，是否有过于专业的术语；所有描述是否都容易理解。
- 安装过程不应该复杂。如果过于复杂，某些复杂的安装步骤是否可以由计算机自动完成。

3. 安装结束

检查安装是否达到目标，包括系统是否能正常运行，系统是否可以使用原有的一切数据等。对安装过程中发现的问题，需要报告缺陷，和产品本身的缺陷没有两样。

9.8.2 软件服务模式下的安装测试

在软件服务非常发达的今天，软件产品发布方式发生了变化，软件系统的安装和纯客户端软件的安装差别很大。首先在客户端，有两种安装方式：

（1）下载类似于.msi 格式的客户端软件包到本地，然后进行安装。这与纯客户端软件的安装没有区别。

（2）只在浏览器中安装控件或插件，没有客户端软件包下载。这时软件系统的安装和客户端操作系统、浏览器及其设置都有关系。需要在不同操作系统和浏览器中进行测试，检查是否能自动判断客户端所使用的操作系统进而自动下载native客户端或Java客户端？

检查是否能自动判断浏览器（IE、FireFox 等）进而下载相应的 ActiveX 或 Plug-in？

服务器端安装测试和客户端安装测试有什么区别呢？在细节上没有大的差别，也需要按照文档一步一步进行，不能忽视任何安装文档上的错误。但在宏观的测试任务上，至少有几点不同：

（1）安装测试要和系统升级策略、数据兼容性、数据迁移等一同考虑，才能圆满地完成安装测试。

（2）在安装环境上的要求则高得多，要满足产品运行环境的实际要求，包括多台服务器集群、分布式网络环境、第三方系统集成等。

（3）服务器的配置也复杂得多，包括各种配置文件。要避免“hard code”的配置参数，否则到了真实的运行环境，系统仍不能正常工作。

（4）必须能恢复到安装之前的状态，在提供安装文档的同时，要提供回退（roll back）的程序。

多数情况下，服务器端多个软件版本并存，系统安装时，要考虑不同的软件升级方案或途径，并考虑系统数据的迁移和备份，如图 9-17 所示。

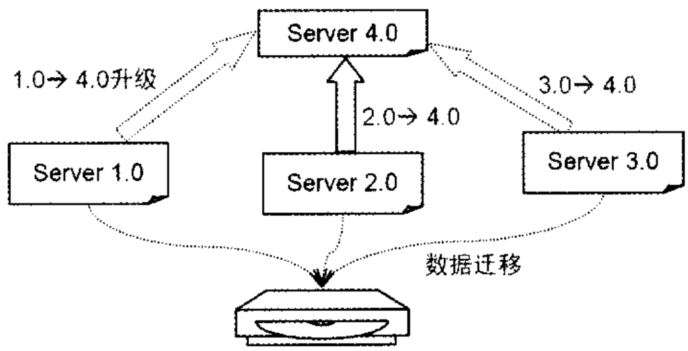


图 9-17 多个版本向一个版本升级的示意图

9.9 小结

系统测试就是验证系统是否符合非功能特性的质量需求，包括性能、安全性、兼容性、可靠性等，而且这些特性相互之间有一定的关系，所以在进行系统测试之前，要规划好系统测试执行的先后次序以及测试结果的共享。

负载测试是本章重点讨论的内容，在本章中介绍了负载测试的4种加载方式，以及如何做好负载测试之前的准备工作，负载测试的环境和步骤，负载测试的结果分析，其中包括如何确定系统关键业务流程、系统所承受的最大负载、负载模拟的持续时间以及负载测试的输出参数等。

在性能测试中，我们了解了基准测试、规划测试和容量测试，以及如何进行 Web 的性能测试，并给出了具体的实例。随后又详细讨论了：

- 安全性测试的分析和执行细节。
- 容错性测试。给出了一些异常输入、异常操作的示例。
- 数据库性能测试的任务、监控指标、常见的测试工具比较和测试用例等。
- 兼容性测试。包括数据的向前兼容和向后兼容、与第三方软件的数据兼容、系统自身多版本的兼容。
- 安装测试。包括客户端软件安装测试、软件服务模式下的安装测试等。

10

第10章

后续测试

只要有新的版本发布，就有新的测试。测试过程，也是一个螺旋上升的过程，周而复始，直至软件生命周期的结束。

在完成了软件功能测试、系统测试之后，测试过程并没有结束。为什么说软件测试还没有结束？那是因为还有许多事等着去做，例如：

- 用户验收测试。
- 文档测试。
- α测试和β测试。

测试过程是伴随着整个软件的生命周期的，软件在整个生命期中不断得到维护，而在维护过程中，要么修正软件缺陷，要么增强产品特性，软件一直都处在变更之中，设计及代码修改和重构，都离不开测试，包括新功能特性的测试、回归测试等。

10.1 验收测试

验收测试，从其原意上看，是和用户共同进行测试的活动，强调的是用户的参与和在用户的环境上进行。对于一个为最终用户开发的软件项目，验收测试是非常合适和必要的。但是，如果是开发通用性产品或提供软件服务，则很少说“验收测试”。在产品发布前，与用户共同完成验收测试的可能性就比较小，往往会转变为另两种测试——α测试和β测试。从这种意义上看，验收测试就不存在，或者说，我们不得不将“验收测试”扩展为广义的概念——在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。所以，如果是公司自行开发的产品，可以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、培训等部门人员。

验收测试，则是技术测试的最后一个阶段，也称为交付测试。即当软件完成功能测试、界面测试和各种系统测试之后，我们需要依据软件规格说明书，对软件进行一次全面的测试，完成对软件整体质量的评估。

概念

α测试（Alpha test）。先在公司内部的环境上运行，由公司员工自己先试用。在许多公司，α测试成为必须经过的一个环节，并写入流程中。微软公司在其产品上市之前，都会在内部全面进行试用，尽力去发现软件中所存在的错误，从而可以及时地向开发团队提供意见反馈。微软公司将这种α测试戏称为“吃自己做的狗食（eating its own dog food）”。如果自己都不喜欢，怎么能将产品推出去呢？

β测试（Beta test）。一般情况下，让少数用户或公司的合作伙伴使用，提出反馈意见和发现缺陷，像微软公司、Oracle公司等都是这样做的，如果没有真正的用户参与β测试，是不会将像操作系统、数据库管理系统这样非常关键（critical）的产品推向市场的。在软件服务（主要是免费的软件服务）中，产品可能先上线，对用户宣称，这不是正式版本，而是一个试用版本（Beat版），让用户试用，提供反馈、报告缺陷。甚至，有人开始喊“永远的Beta版”。

1. 产品规格说明书的验证

验收测试一般会根据产品规格说明书，严格检查产品，逐字逐句地对照说明书上对软件产品所作出的各方面要求，确保所开发的软件产品符合用户预期的各项要求。即验收测试是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，同时应该考虑用户的实际使用情况。

在验收测试阶段，产品规格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规格说明书，尽管它已经通过了审核。审核不充分、需求的变更等因素，都会导致产品规格说明书不得不作相应的更改。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，还要针对上述的因素进行分析，设计测试用例，确保产品规格说明书验证的完整性。

2. 用户环境的验证

验收测试必须在真实的运行环境中或非常接近实际的环境中进行。测试环境与实际运行环境的不同往往会导致遗漏严重的缺陷，或者用户到时根本不能完成安装或操作，这样的例子已有很多。例如，一个数据库查询系统，在拥有少量数据时工作正常，而在实际环境中，可能拥有海量的数据，由于查询操作语句没有优化，耗时太长，所以不能被用户接受。

验收测试，强调须在用户环境下进行，这和 β 测试是一致的。β 测试一般运行在产品实际的环境上。规范的软件服务公司，在部署产品发布给用户之前，先会在内部同样标准的环境（β 测试环境）上运行一段时间（如两周、一个月或几个月等）。验收测试，要么在用户现场进行，即在真正的用户环境上运行，要么在公司内部或真实模拟用户环境的测试环境下进行。

概念

模拟用户的环境也不容易，因为用户各种各样的环境都可能存在的。

- 在服务器端，用户之间的数据差别较大，而且还会出现意想不到的垃圾数据。
- 在网络方面，防火墙的产品类型、安全策略和配置等，千变万化。
- 在客户端，用户装了各种各样的、不同的应用程序，它们中某一种或几种程序可能和被测试的软件有冲突。测试时，至少要考虑多种情况，如在全新机器（fresh machine）安装，在比较脏（应用程序很多）的机器上安装，在原来老版本上直接升级，先卸载旧版本再安装新版本等测试场景。

3. 通过准则和验收结果

通过了解软件功能和性能要求、软硬件环境要求等，特别是要了解软件的质量要求和验收要求，来制定验收通过准则。对于验收测试，一般要求编写单独的测试计划，在测试计划中定义明确的验收通过准则。验收通过准则需要得到用户的评审和同意。根据验收通过准则分析测试结果，作出验收是否通过的评估及测试评价。通常会有 4 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中改进；
- 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在验收通过准则中没有说清楚，则应该修改准则。

4. 验收测试结束的标志

（1）完全执行了验收测试计划中的每个测试用例。

（2）在验收测试中发现的错误已经得到修改，并且通过了测试或者经过评估留待下一版本中修改。

（3）完成软件验收测试报告。验证报告的内容，应包括所有特性的清单：

- 已经实现的特性，标识为通过。
- 特性没有实现的缺陷报告。
- 特性基本实现，但与产品说明书中的内容有不相一致的缺陷报告。
- 特性基本实现，但软件程序本身存在一些问题或错误的缺陷报告。

10.2 文档测试

软件文档是软件的一部分，文档的错误也是软件缺陷。错误的解释可能会导致用户无法使用某些软件已具有的功能。如安装文档不正确，用户无法自行完成安装，自然是严重的软件缺陷。

- 好的文档能提高易用性和可靠性，降低技术支持的费用，从而提高产品的整体质量。
- 用户通过文档可以掌握具体的使用方法，提高了易用性。
- 避免了用户在摸索使用中的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发生，从而提高了产品的可靠性。
- 当用户遇到问题时，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。如果约 30%的用户通过文档解决了问题，也就大大降低了技术支持的次数。

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。软件驱动的文档还得

像程序一样运行起来实施测试。

- 正确性是指不要把软件的功能和操作写错，也不允许文档的内容前后矛盾。
- 完备性是指文档不可以“虎头蛇尾”，更不许漏掉关键内容。文档中很多内容对于开发者来说可能是“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。文档中的术语、缩写词，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。测试人员应与文档作者密切合作，仔细阅读文档，严格按照每个步骤进行操作，检查每个操作界面，尝试每个示例，这些都是进行文档测试的基本方法。

知识点

1. **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行而印制的小册子，通常是简单的软件使用入门指导书。而详细的帮助内容通常以联机帮助文档的形式出现。在帮助文档中有索引和搜索功能，用户可以方便快捷地查找所需信息。Microsoft Word 的联机帮助文档，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。

2. **指南和向导**。它们是程序和文档融合在一起形成的，是引导用户一步一步完成任务的一种工具。如 Microsoft Office 助手。

3. **安装、设置指南**。简单的指南可以是一页纸，复杂的可以是一本手册。

4. **示例及模板**。例如开发平台软件附带了许多例子、微软 Office 软件提供了大量的 Word/PPT 模板。

5. **错误提示信息**。这种信息常常被忽略，但也是软件/文档的一部分。一个较特殊的例子是服务器系统运行时检测到系统资源已达到临界值或受到攻击，给管理员发送警告邮件。

6. **用于演示的图象和声音**。现在越来越多的软件拥有这部分资料。

7. **授权/注册登记表及用户许可协议**。

8. **软件的包装、广告宣传材料**。有些用户会认真对待这上面的信息，并很好地利用它们。但错误或缺少必要的信息可能会带来麻烦，所以甚至标签上的信息等均是文档测试的内容。

10.3 α 测试和 β 测试

软件通过最后阶段的测试——验收测试或质量全面评估测试。从研发阶段来看，似乎这一阶段已告一段落，即达到了一个重要里程碑“工程发布（Engineering Release，ER）”阶段，随后将准备推向市场。但是，实际上并不能马上全面推向市场，如前面所说，要经过 α 测试和 β 测试，才能最终把软件推向市场。

为了更好地理解产品工程发布之后的过程，先介绍几个概念：

- LA，有限的可用（ Limited Available）。
- GA，全面可用（General Available）。
- SPR， 服务包发布（Service Pack Release）。
- EPR，紧急补丁包发布（Emergency patch）。

然后，我们再看看产品工程发布之后的后续过程，如图 10-1 所示。从图中可以清楚地看到 α 测试和 β 测试的位置。而且，对于每个 SP 的发布，一般也要经过 α 测试和 β 测试，但不再单独有 LA 和 GA 的里程碑，有时为了缩短 SP 的发布过程，可能只经过 β 测试，而不进行 α 测试。对于 EP 的发布，时间更短，由于紧急，则没有 α 测试和 β 测试，而是直接上产品线。

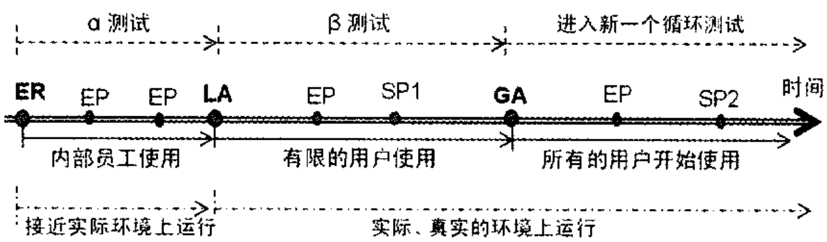


图 10-1 产品工程发布之后的后续过程

α 测试和 β 测试，没有什么特别的地方，这种测试，主要是真实的操作使用，通过实际的操作行为对产品进行最直接的测试。任何员工发现问题，都可以报告缺陷。为了控制缺陷报告的质量，可以报告给测试人员，通过测试人员来验证它是否有效，如果有效，就报出一个缺陷。由于一般的员工不了解已报但没修正的缺陷（不必要修正或无法修正的缺陷），所以这种控制是必要的。若客户发现问题，可以通过客户服务人员或技术支持人员来报告问题，在国外公司称为记录一个 Remedy Ticket（补救、修正通知单），然后由相关人员（技术支持人员或测试人员）验证，如果有效就处理为一个缺陷。

对于每一个有效的 Remedy Ticket，测试人员都应该认真分析，并找出原因。

- 没有清楚地了解需求？对需求产生误解？
- 有些使用情景没有想到？没有测试用例？
- 如果有测试用例，那么是环境不对？
- 如果有测试用例，那是不是没有执行？
- 回归测试是否不充分？
- 是无法避免的环境问题或非常特殊的使用场景（corner case）？

在α测试中，如果说有特殊的地方，那就是如何增加数据量、如何进行大量或长时间的操作，以便于尽量多地发现问题，减少将来客户发现的问题。例如，在公司内部，由于比较容易组织和协调，可以约定某个时间段（半个小时、一个小时等），大家同时进行各种各样的使用和操作，以发现在大负荷、多任务相互影响等情况下的问题，这常被人称为“消防演习或消防测试（Firedrill Test）”。Firedrill Test 对多任务、多服务相互影响的测试非常有效，因为在几十、上百个人共同操作下，很多过去没有出现的操作及其组合，都可能会出现。

概念

（1）LA，重要的里程碑之一——有限的可用（ Limited Available）。由于测试覆盖率不能做到 100%，以及测试环境不可能和真实环境一样，所以我们对产品的功能和质量还没有十分的把握或足够的信心，为了尽量降低商业风险，可借助于β测试，所以先让少数用户使用。相对来说，这些用户和公司保持着良好的（合作）关系，他们愿意试用，并提供反馈。这相当于化工产业的试生成，而在中试车间的实验，则是α测试。

（2）GA，重要的里程碑之一——全面可用（General Available），在修正完 LA 之后所发现的问题，公司对产品功能及其质量都有充分的信心，这时就可以全面推向市场，让所有用户使用。

（3）SPR，服务包发布（Service Pack Release）。它会在产品基础版本发布的基础之上，对原有功能增强，还可能增加少量新功能或修正较多的、已发现的缺陷等，并发布次要版本。这是软件开发分阶段增量模型的体现，以满足软件市场策略的需求，解决软件需求分析复杂性的问题。SP 的开发周期相对较短，一般不包含全部文件，只包含所修改或必需的文件，由于所有测试在主版本都做过，所以有些测试不需要做，但回归测试工作量仍较大，而且主要针对新功能或修改过的功能进行测试，性能或安全性等方面的对比或补充测试也少不了。

（4）EPR，紧急补丁包发布（Emergency patch）。主要是为被客户发现的、抱怨比较大或危害较严重的缺陷修正而紧急发布的软件补丁。EP 不增加任何新功能，只修正极少的软件缺陷，修改范围很小，开发周期也很短，一般是几天，最多不超过 2~3 周。如 IE/Windows XP 发布过多个有关安全性、病毒攻击等问题的补丁包。

10.4 产品后继版本的测试

一个软件的生命周期是很长的，产品工程发布之后，还要发布 SP 和 EP，如 1.1、1.1.1、1.1.2、1.2 等版本。GA 之后，产品可能要发行新的版本，如 2.0、3.0 等。只要有新的版本发布，就有新的测试。测试过程，也是一个螺旋上升的过程，或者说大循环中套小循环，周而复始，直至软件生命周期结束。对于这些 SP 和 EP 的测试，或者对 2.0、3.0 等的测试，有什么不同吗？下面，我们就来分析比较。

1. EP 的测试

让我们先来讨论相对简单的 EP 测试，因为 EP 只修正极少数的缺陷。如何在短时间内完成 EP 的测试？EP 测试没有产品需求文档、产品规格说明书的审查，EP 测试的目标也很清楚，就是保证要修正的缺陷确实被修正了，而且不会引起回归缺陷。这些都为测试赢得了时间。EP 测试包括 3 部分：

（1）缺陷的验证。

（2）围绕修正缺陷所进行的代码修改而进行的回归测试。

（3）验证新构建的软件包，确认没有丢掉文件，没有将文件的版本弄错等，保证所有主要的功能可以正常工作。

EP 测试的质量，主要取决于第（2）、（3）部分的工作质量。在有限的时间内，要准确地完成回归测试，需依赖于对代码的审查和产品测试上所积累的经验。而后者，可以借助代码比较或文件比较工具找出不同的文件，借助自动化测试工具完成 BVT。

2. SP 的测试

SP 测试相对要复杂些，一方面，修正的缺陷比较多，回归测试的范围更难确定；另一方面，有些功能会增强，还有少量的新功能加入进来，需要进行需求定义文档、设计规格说明书的审查，并设计相应的测试用例。虽然工作量不是很大，可由于时间紧，还是富有挑战性的。对于修正缺陷的测试方法和策略，可以参考 EP 测试。而对功能增强和新功

能的测试，实际也可分为 3 部分：

（1）功能增强和新功能的验证。

（2）围绕修改的功能进行有效的回归测试，包括新旧功能之间是否矛盾，是否相互影响等。

（3）验证新构建的软件包，保证所有的主要功能可以正常工作。

所以第（3）部分的工作，都可以和修正缺陷的同部分工作合并。而两者的第（3）部分，可能会存在部分内容重合的情况。

代码审查和自动化测试，在 SP 中依然是重要的。代码审查可以帮助确定回归测试的范围，而自动化测试可以快速地完成回归测试和软件包的验证工作。在 SP 测试中，与 EP 测试一样，主要实施功能测试，系统测试工作比较少，因为没有架构和部署设计上的变化，功能上也没有质的变化，所以性能、故障转移、可靠性等都没有变化（特殊情况例外），一般只需要进行少量兼容性、安全性方面的测试，还包括安装的快速测试。

在 SP 测试中，还有一个人力资源的定位问题，有两种解决方法：

（1）由测试基础版本的原有团队来完成 SP 的测试，优点是原有团队熟悉产品所有特性，不需要知识交换，缺点是资源冲突，因为当进行 SP 项目时，原有团队可能在该产品的下一个主要版本上工作。另外，SP 项目所需的资源，远远小于原有团队。

（2）由独立的团队完成 SP 的测试，其优缺点和上一种方法正好相反。SP 团队需要知识传递，开展更多的培训工作，但资源比较独立，项目之间没有影响。

3. 更高版本的测试

当开发产品的 2.0 或 3.0 版本的时候，它的测试工作将和一个新项目的测试工作非常接近，需要进行各种评审工作，包括完成测试计划、测试用例设计和测试脚本的开发等，不仅要完成功能测试工作，还要完成各种系统测试，包括性能测试及其比较分析。

不同的是，更高版本的测试有新旧功能交替的影响，并存在大量的回归测试工作。由于功能改动很大，回归测试范围也会比较大。

10.5 小结

验收测试是技术测试的最后一个阶段，是对软件进行一次全面的测试，将完成对软件整体质量的评估。验收测试要关注以下 4 个方面：

- 产品规格说明书的验证。
- 用户环境的验证。
- 通过准则和验收结果。
- 验收测试结束的标志。

除了验收测试之外，还要进行文档测试、α 测试和 β 测试，对软件发布的整个过程要有一个全面的理解，包括 LA 和 GA 这两个重要里程碑的作用。最后，要掌握 EP 测试、SP 测试的策略和方法。

11

第11章

测试的跟踪和管理

测试跟踪和管理从测试计划开始，围绕着测试用例和缺陷不断展开，深入，最终获得高质量、高效率的测试结果。

前面 10 章从具体实施的角度展示了测试的全过程，着重介绍了测试的方法、技术和工具的应用等。但要完美地实现测试目标，使测试方法和技术得到很好的发挥，构成一个符合要求的、有效的执行过程，则需要很好地实施测试过程的跟踪和管理，包括过程的度量和评审、测试管理系统等。

测试的跟踪和管理是测试组长、测试经理和项目经理所关注的主要方面，但一般测试人员也需要了解。因为这方面的知识不仅能帮助测试人员更快地成长，为将来成为测试组长和经理打下基础，而且还有助于测试人员配合项目组的工作，使团队工作更协调。

测试跟踪和管理从测试计划开始，围绕着测试用例和缺陷不断展开及深入，最终获得高质量、高效率的测试结果。

11.1 测试管理

当一个软件公司处在初级水平时，会先有测试活动，在发现问题后，为了解决问题才有测试管理。而对一个成熟的软件公司来说，测试管理在先，测试活动在后，即先有一套流程和过程跟踪方法等，然后开展测试活动，主动收集数据，进行分析并不断改进测试流程。

11.1.1 测试管理的全局性

从测试跟踪和管理的角度来看，虽然关注被跟踪的对象有轻有重（如测试用例是重点关注的对象，对测试用例设计和评审的管理是非常重要的），但测试过程的每一个环节都不能错过，包括同项目没有直接关系的内容，如测试工程师的培训，知识的横向传递，问题的跟踪，承诺，沟通等。测试作为质量保证的重要手段之一，自身应该形成一个较完善的体系，以强调测试管理的全局性，不要忽视任何一个环节，不要轻视任何一个细节，让大家把每件事都做好，达到很高的质量水准。

1. 管理活动

从产品需求文档（PRD，包括 UI Mock-up）审查开始到产品发布，我们要实施对测试全过程的跟踪和管理，涉及的内容非常广泛。在这过程中，其主要的管理活动有：

- 产品需求文档的审查活动和签发。
- 产品规格说明书（Spec）的审查和签发。
- 测试计划书的审查和签发。
- 测试策略有效实施的跟踪。
- 测试用例的审查和签发。
- 测试环境的审查。
- 单元测试报告的审查。
- 功能测试执行的跟踪。
- 系统测试执行的跟踪。
- 回归测试的风险管理。

- 验收测试的管理。
- 测试结束的评估。
- 测试人员的培训和教育。
- 测试和其他部门之间的沟通。
- 测试项目的分析和总结，以及改进计划和措施。

2. 管理的层次性

测试管理的内容很多，可以分为多个层次——团队、过程、方法、项目和执行，如图 11-1 所示，简单说明如下：

（1）人是决定的因素，团队是基础，应在招聘、培训、组织架构和绩效评估等各个方面造就一流的团队。

（2）基于什么样的团队，就建立什么样的过程，过程质量决定产品质量。为了保证过程质量，需要建立一套质量保证体系、定义测试计划、设计和执行流程、缺陷生命周期。

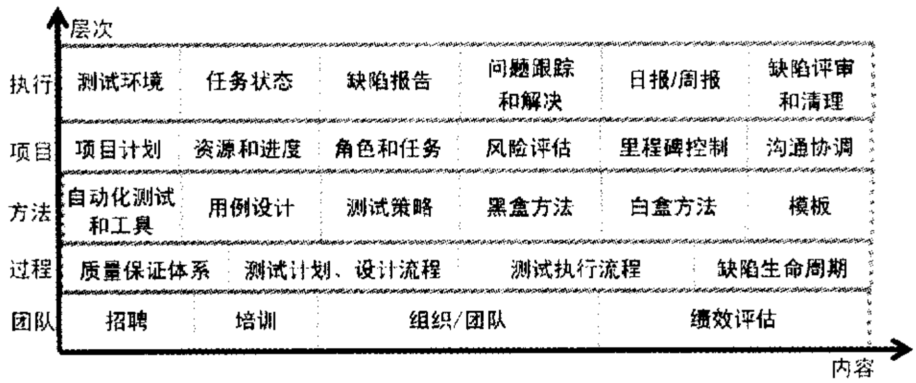


图 11-1 测试管理的层次性

（3）方法建立于过程之上，包括测试自动化方法及其工具、用例设计方法、测试策略、白盒/黑盒测试方法和各类测试模板等。

（4）有了团队、流程和方法，项目管理就水到渠成了。现在可以分配项目的任务和角色，确定项目的资源和进度，不断对测试风险进行评估以降低风险。

（5）执行是测试过程的具体化和测试方法的应用，执行还是项目计划的具体实施，也需要细致的管理，如检查测试环境的正确性，了解任务执行的状态，对缺陷进行评审并要求得到及时的清理，通过日报、周报全面掌握测试的进展情况和当前状态，发现问题、跟踪和解决问题等。

11.1.2 测试策略的执行

对于大型项目，除了需要很好的测试范围分析、测试计划制定和测试资源的分配与组织之外，软件测试的执行也是非常重要的。

对于大多数应用项目（非国防、载人飞船上天、净室工程等），我们都知道，测试不是为了证明所有的功能能正常工作，恰恰相反，测试就是为了找出那些不能正常工作及不一致性的问题，简言之，测试就是发现缺陷。当然，这些缺陷不仅仅是程序中的错误，还包括需求分析、设计等方面的缺陷。测试的启动和项目启动是同时发生的，测试的重要工作是测试用例的设计，这是随后测试执行的基础。同时，我们应该承认，测试的大部分工作还是测试的执行，当自动化测试工具在功能测试中发挥的作用依然有限时，测试执行的工作量还是很大的。

如何更早地发现缺陷又不增加风险？测试的本质是什么？发现缺陷还是风险评估？如何引导大家向着一个目标"及时地高质量发布产品"努力？除了第 1 章所作的讨论之外，还可以从下面几个方面进行考虑，确保高效的测试策略被贯彻执行。

（1）首先就要向测试人员不断灌输一个概念——"测试工作就是发现缺陷 （detect bug）"，让大家取得共识。这样，测试人员就知道什么是自己真正的工作，在清楚了测试的目标后，思维的方式都会发生改变。正确的测试理念，不仅有助于测试执行的效率，还有助于测试用例设计的质量。

（2）测试执行阶段可以划分为两个子阶段，如图 11-2 所示。前一个阶段的目的非常清楚，就是发现缺陷，越有可能发现缺陷的测试用例越要优先执行，一个测试用例的成功执行就是通过这个测试用例发现了缺陷。测试用例的执行，应该是帮助我们更快地发现缺陷，而不是成为"发现缺陷"的障碍——使发现缺陷的能力降低。从理论上说，如果缺陷都找出来了，质量也就有保证了。所以在这一阶段，要不顾风险地去发现缺陷，这样不仅对开发团队非常有利，可以让他们能尽早地修正大部分缺陷，也对测试有利，测试效率高，后面的回归测试也会稳定，测试人员的信心更充分。

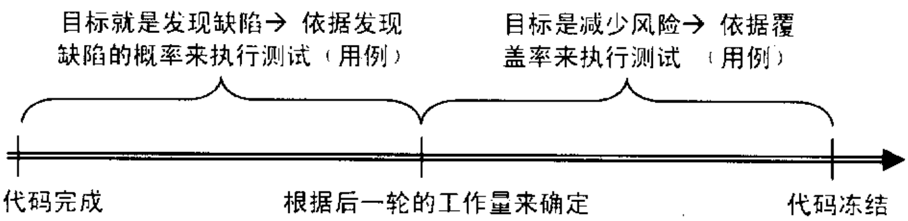


图 11-2 基于两阶段的测试执行策略

（3）在代码冻结或产品发布前的后期阶段，测试目标是增加测试的覆盖率及降低风险，这时测试的效率会低一些，以损失部分测试效率来获取质量的安全性，从而降低风险，获得更高质量的收益。

（4）在前一阶段，测试用例的执行速度要慢一些，测试人员应多思考，多做些 Ad-hoc 测试，这样可帮助提高测试用例的质量，从而对随后的回归测试提供更有力的保障。

（5）测试执行要进行有效的监控，包括测试执行效率（缺陷数/KTC，KTC = 1000 test cases）、缺陷历史情况和发展趋势等。必要时，根据获得的数据对测试范围、测试重点等进行调整，包括采取对测试人员的调整、互换模块等手段，以提高测试覆盖率，降低风险。

（6）测试总是有风险的，正是始终存在的风险，使测试更具有艺术性。

11.1.3 测试管理系统的应用

如果没有测试管理系统的帮助，测试几乎是不可想象的。要跟踪管理好测试过程，测试管理系统是必不可少的。测试管理系统可以管理下列内容：

- 测试用例。
- 测试套件。
- 测试执行结果。
- 缺陷记录。
- 缺陷跟踪、分析报告。
- 测试资源分配。
- 测试数据。
- 测试用的软件。
- 测试环境配置。

后面 5 项内容，可以归为测试管理系统，也可以由公司内部相应的系统整体管理，如测试脚本可以用产品源代码的配置管理工具（CVS、SubVersion、ClearCase 等）来统一管理，而另外两项内容可以由文档管理系统、文件服务器等实现。

- 测试脚本。
- 测试文档。
- 被测试产品的各种版本、软件包。

1. 以测试用例和缺陷为中心

在测试管理系统中，管理的核心是测试用例和缺陷。

- 测试套件是测试用例的组合，而测试数据、测试环境配置等可以看作是测试用例的组成内容，测试执行结果就是测试用例在不同环境中运行的记录。
- 缺陷是测试进度跟踪、质量评估等工作中所需要的重要依据。

所以测试用例的管理非常讲究，优秀的测试管理系统可以解决下列问题：

- 如何设计、构造灵活的测试套件？
- 如何有效执行计划中所要求的测试用例？
- 如何跟踪测试执行的结果？

而对于缺陷管理，指的是如何更好地跟踪缺陷状态，如何针对缺陷记录进行各类统计分析和趋势预测等。

2. 测试工件的映射关系

在测试管理系统中，很重要的一点就是在测试用例同缺陷之间建立必要的映射关系，即将两者完全地关联起来：

- 当知道一个缺陷时，就知道是由哪个测试用例发现的。
- 可以列出任何一个测试用例所发现的缺陷情况，根据这个，就知道哪些测试用例发现较多的缺陷，哪些测试用例从来没有发现缺陷。发现缺陷的测试用例更有价值，优先得到执行。

更理想的话，还要建立起需求、产品特性/功能点同测试用例之间的映射关系，如图 11-3 所示。在构成了这样的映射关系后，更容易解决需求变更，回归测试范围确定，质量评估等一系列重要的问题。需求变更是肯定会发生的，正如人们常说，变化是永恒的，不变是不存在的。借助这种映射关系，可以解决需求变化所带来的问题，如：

- 需求变化会影响哪些功能点？
- 功能点发生变化，需要修改哪些测试用例？
- 产品的某个特性或某个功能点所存在的缺陷有哪些？其质量水平如何？
- 如果一个缺陷是由于设计缺陷、需求定义引起的，如何追溯到原来的需求上去解决？
- 通过对缺陷的分析，如何进一步改进设计和提高需求定义的准确性？

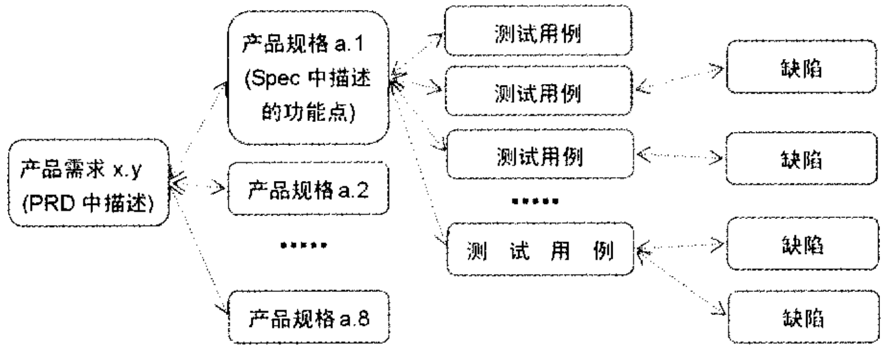


图 11-3 需求、功能点、用例同缺陷之间的映射关系

3. 测试管理系统的构成

对测试输入、执行过程和测试结果等进行管理，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需要，应该使用数据库技术、XML 技术或其他有效的数据格式等进行管理。

因为测试脚本由源代码配置管理系统控制，所以不包括测试脚本的管理。其次，资源、需求、变更控制等项目方面的管理，属于整个软件过程管理，不属于测试管理系统，虽然它们之间有关系。所以，测试管理系统以测试用例库、缺陷库为核心，覆盖整个测试过程的组成部分，如图 11-4 所示。

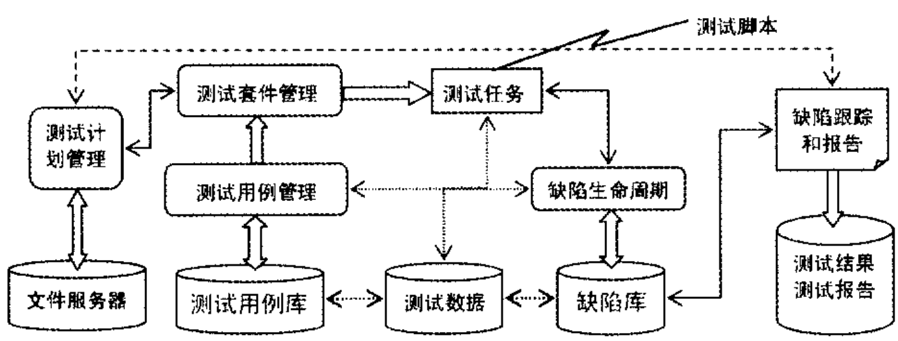


图 11-4 测试管理系统的构成示意图

11.1.4 测试管理工具

测试过程涵盖单元测试、集成测试、系统测试、回归测试、交付测试等各个阶段，如

何有效地组织管理起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、测试用例、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。

关于测试管理工具，先介绍商业软件，然后介绍开源工具。主要的商业工具有：

- 软件测试管理工具主要有 HP-Mercury 公司的 Test Director，IBM-Rational 公司的 Test Manager，Compuware 公司的 QADirector、Silicon Valley Networks 公司的 TestExpert 等。
- 缺陷跟踪工具主要有 Rational ClearQuest、Compuware 公司的 TrackRecord、WEBsina 公司的 Bugzero 以及 SilkRadar 等。

后面会以 Test Director 和 SilkRadar 分别作为示例，来说明一个优秀的综合测试管理工具和单项的缺陷跟踪工具所具有的特性，帮助大家更好地了解这些工具。

如果撇开商业工具，开源的测试管理工具和缺陷管理工具也已经能满足这方面的基本需求，对于多数中小型软件企业来说，使用开源工具足够了。如果想根据企业自身特点增加一些特定的功能，开源工具也是最好的选择。

（1）开源的缺陷跟踪工具

- Mantis，是一款基于 Web 的软件缺陷管理工具，配置和使用都很简单，适合中小型软件开发团队，详见 *http://mantisbt.sourceforge.net/*。
- Bugzilla，是比较流行的、免费的软件缺陷管理工具。详见：*http://www.mozilla.org/projects/bugzilla/*。

（2）开源的测试管理工具

- TestLink，是基于 Apache、MySQL、PHP 开发的测试管理和执行系统。测试小组在系统中可以创建、管理、执行、跟踪测试用例，并且提供在测试计划中安排测试用例的方法。详见 *http://testlink.sourceforge.net/docs/testLink.php*。
- Bugzilla Test Runner，是基于 Bugzilla 缺陷管理系统的测试用例管理系统。详见 *http://sourceforge.net/projects/testrunner/*。
- TestMaker，是普及的开放源代码测试框架与实用程序，详见 *http://www.pushtotest.com/*。
- Eclipse 是测试和性能工具平台（Test & Performance Tools Platform，TPTP）。

示例

Test Director是基于Web浏览器环境下的管理工具，它所具有的特点如下：

- 维护一个涉及程序中所有功能的测试数据库。
- 调度并且执行测试集，提供直观且有效的方法来收集测试结果和分析数据。
- 跟踪应用程序缺陷，从缺陷被发现之时直到这个缺陷被关闭。
- 通过与邮件系统相关联，缺陷跟踪的相关信息就可以被整个项目开发组、市场、客户支持和IT等的很多人员所共享。
- 能够与测试工具、配置管理工具和建模工具等集成，提供全套的自动化测试解决方案。

其测试过程管理包括4部分：明确测试条件和需求、测试计划、执行测试和跟踪缺陷。

（1）明确条件。检查应用程序的文档，以确定测试范围和测试目标，以及策略；构建“需求树”的目的是确保完全覆盖测试需求；为“需求树”中的每一个需求话题建立一个详细的目录，描述每一个需求，给它分配一个优先级，包括链接附件；产生的报告和图表可帮助分析测试需求。

（2）编制测试计划。定义测试策略，定义测试对象，定义测试，创建需求覆盖（连接每一个测试和测试需求），设计测试步骤，建立自动化测试脚本和分析测试计划。例如为每一个模块确定其测试类型，在测试计划树中为每一个测试点添加基本说明，它还描述了测试注意事项、检查点、每个测试的预期结果。

（3）测试运行：创建测试集合（套件）、分配测试任务和时间表，运行测试任务和分析测试结果。

（4）缺陷跟踪：添加缺陷，检查新的缺陷，修复开放的缺陷，测试新的软件包构建和分析缺陷数据。

SilkRadar对软件缺陷进行记录，并对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。
- 通过采用电子邮件通知及自动分配规则，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把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，让他进行处理。
- 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、用户定制的报表和图表，以及用于有效提高表达项目状态的查询（query），使用统一定义的信息，可提高沟通效率，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。

11.2 测试用例的管理

测试用例是测试的基础，测试用例的质量决定了测试的覆盖率，对产品最终的质量有很大的影响，所以加强测试用例的管理显得非常重要。测试用例的管理涵盖了测试用例的创建、执行和维护。

11.2.1 测试用例创建的管理

好的开始是成功的一半，在开始设计测试用例时，就要抓管理，确保在测试用例设计和使用过程中能开一个好头。

1. 意识和态度的教育

设计新的测试用例，是建立在对软件产品的需求、设计和代码实现等理解的基础之上的。如何通过管理来提高测试人员在这些方面的理解程度？如果做好下面几项管理工作，就可以提高大家的理解水平。

- 促进和客户、产品设计人员、开发人员等的直接沟通和充分交流。
- 加强培训和知识共享，让产品设计和开发人员作专项的介绍。
- 加强产品需求和设计文档的评审，强调要通读所有内容，澄清各种问题，使大家达成共识。
- 让测试人员讲解对产品特性和功能的理解。

但上面几项内容很难检验其效果，所以问题还没彻底解决。要想达到我们预期的目标，更多依赖于不断的教育，树立“一切从客户需求出发”的观念，建立积极主动的态度。例

如，每个测试人员在阅读 PRD/Spec 的时候，绝不能只想“这是文档定义”，将“功能”只看作“功能”，而应该多问几个“为什么”的问题，若自己能思考出这些问题的答案，就能深刻理解用户的需求。

- 为什么要加这个功能？为什么要做这样的改动？
- 这个功能对客户有什么价值？
- 用户会如何使用这个功能？在哪些情况下使用？
- 同其他地方没有冲突吗？

2. 责任到人

在测试用例设计的管理中，应将模块划分清楚，责任到人。任何一个测试模块都有专人负责，从需求分析开始到测试用例的维护，确保测试用例的质量。负责某个模块的测试人员要和开发人员一起工作，以便于对产品特性设计进行充分讨论，了解其实现的机理和薄弱地方，从而开发出有效的测试。

任何没有被及时发现的缺陷，都由专人负责分析，找出根本原因，举一反三，增加新的测试用例或者修改已有的测试用例，不断完善测试用例。

3. 方法和流程

最后，要在设计方法和流程上加强管理，包括：

- 采用测试用例的模板，参考已有的范例。
- 要求先设计工作流程图、数据流图。
- 要求测试人员相互审查、提问。
- 集体审查测试用例，邀请客户、产品设计人员、开发人员等参加。

11.2.2 测试用例执行的管理

测试用例能被有效、灵活地执行，就需要一套管理机制和相应的工具/系统的协助。前面已经介绍测试用例的结构，可以按照产品线、测试目标和功能模块等进行分类、组织和存储，以有利于测试用例的执行和维护。图 11-5 就是一个示例，说明如何进行有效的测试用例管理来更好地执行测试用例。

（1）可以从不同的测试模块中选择一部分测试用例，然后和所需的测试环境组合，生产测试套件。

（2）可以设定一些灵活的过滤条件，如未通过的测试用例、优先级高的测试用例、

某一类型的测试用例和曾发现缺陷的测试用例等，自动生成测试套件。

（3）可以选择若干个测试套件，设定进度，构造执行测试的不同过程，完成特定的测试计划或测试任务。

（4）针对每个测试计划，也可以分解成多个测试任务，然后分配任务给相应的测试人员。

（5）测试人员执行测试任务，完成测试过程，并报告测试结果。

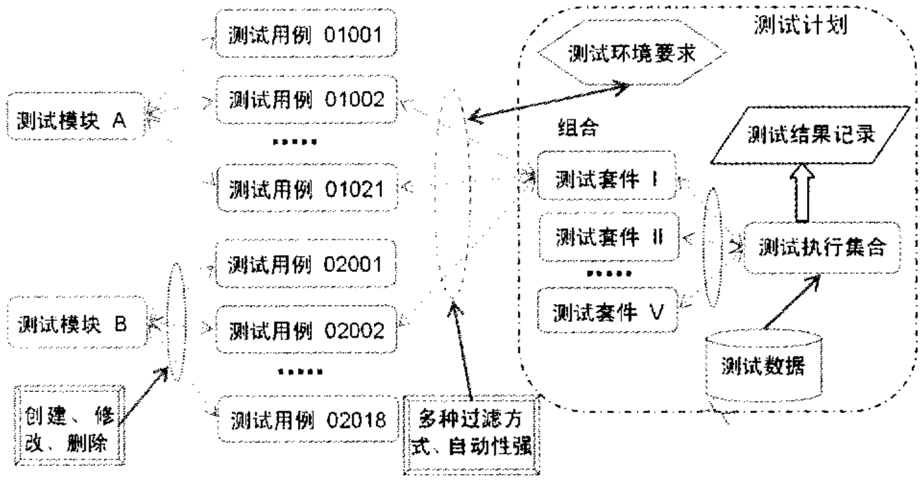


图 11-5 有效的测试用例管理示例

11.2.3 测试用例的维护

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。在一个阶段的测试过程结束后，或多或少会发现一些测试用例编写得不够合理或缺少测试用例覆盖一些应用场景。而且，当下一个版本在测试中使用前一个版本的测试用例时，其中部分功能可能发生了改变，这时候也需要去修改那些受功能变化影响的测试用例，使之具有良好的延续性。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，对某些功能、特性、逻辑等的理解越来越清楚、深刻。
- 所发现的严重的软件缺陷没有被目前的测试用例所覆盖。
- 新的版本中有新功能的需求或者原有功能的增强而需要发生改动。

- 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。

开发一个软件产品，会发布多个版本，伴随着测试用例的不断维护，测试用例也需要不断完善并与产品功能、特性的变化保持一致，从而使测试用例和产品版本相关联。在线软件服务中，由于不同的客户有不同的需求及定制，而且有些客户激进，有些客户保守，所以软件产品的多个版本常常共存，为不同的客户提供服务，这时测试用例多个版本并存，如表 11-1 所示。所以在新建、修改、删除测试用例时要十分小心，确定对正确的版本进行修改，不要错改其他版本的测试用例。无论是对软件产品还是软件服务，多个版本并存的可能性很大，而且可能为不同的主要版本发布不同的补丁包（patch）或小版本，这样早期的一些版本所拥有的测试用例还是有效的。

表 11-1 测试用例的多版本并存示意

Case ID	Case Name	1.0	1.1	2.0	2.1	2.2	2.3	3.0
101001	列表方式显示	✓	✓					
101002	折叠式列表方式显示			✓	✓	✓	✓	✓
101003	按字母顺序排列	✓	✓	✓	✓	✓	✓	✓
101004	模糊查询					✓	✓	✓

根据产品特性和一致性准则，测试用例的维护可按下面几种情况分别处理：

（1）产品特性没变，只是根据漏掉的缺陷来完善测试用例。这时候，增加和修改测试用例均可，因为当前被修改的测试用例对相应的版本都有效，不会影响某个特定版本所拥有的测试用例。

（2）原有产品特性发生了变化，不是新功能特性，而是功能增强，这时候原有的测试用例只对先前版本（如 1.0、2.0）有效，而对当前新的版本（如 3.0）无效。这时，绝不能修改测试用例，只能增加新的测试用例，不能影响原有的测试用例。

（3）原有功能取消了，这时只要将与该功能对应的测试用例在新版本上置为空标志或“无效”状态，但不能删除这些测试用例，因为它们对先前某个版本还是有效的。

（4）完全新增加的特性，很清楚，增加新的测试用例。

从表 11-1 可以看出，每个测试用例记录，针对一个有效版本都有对应的标志位，通过这个标志位，很容易实现上述维护需求。这样，新旧版本的相同测试用例得到一致的维

护，测试用例数也不会成几倍、十几倍的增加，可以真正保证测试用例的完整性、有效性。

维护测试用例的过程是一项长期的任务，而且保证其及时性。和编写一个新项目的测试用例不同，维护测试用例一般不会进行大动干戈，不会改变测试用例的结构。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，则可能需要重新定义一个独立的测试模块单元来重新组织测试用例。和测试用例编写过程相同的是，测试用例的维护流程，强调测试用例的所有性（Ownership）。

（1）测试工程师，是测试用例编写者，或者是其他使用/查看测试用例的人，当他发现测试用例有错误或者不合理时，可向编写者（用例的所有者）提出测试用例修改建议，并提供足够的理由（功能规格说明书中的特定描述、有效的用户使用场景等）。

（2）测试用例编写者根据测试用例的关联性和修改意见，对特定的测试用例进行修改。

（3）向开发、项目组长（经理）递交修改后的测试用例。

（4）在项目组长、开发人员以及测试用例编写者进行复核后提出意见，通过后，由测试用例编写者进行最后的修改，并提供修改后的文档和修改日志。

11.3 测试自动化的管理

测试自动化（Test Automation，TA）的开展，不仅需要具有很好编程经验的工程师，而且也需要测试工程师的积极配合，两者需要合作。不仅如此，TA 工程师如何与开发人员合作和交流，开发人员如何在设计和编程中确保可测试性，对象属性的稳定性等，也需要积极引导。当然，更重要的是建立相应的 TA 框架、流程和规范。

测试自动化需要有一个良好的框架，使之从开发到运行层次清楚，操作方便。比如通过 Web 方式提交任务、查看结果等，并和测试执行、Bug 跟踪系统集成。自然，测试自动化的脚本管理，像软件源代码一样，也有 Check-in/check-out 控制，如用 CVS、SubVersion 等工具管理起来。

11.3.1 测试自动化的框架

测试自动化的框架，可以理解为两个层次，一个层次只是在脚本开发上，相当于 TA 的开发平台，如何灵活地集成相应的测试工具，构造成关键字驱动或数据驱动的脚本。另一个层次是在更高层面上管理整个 TA 的开发、执行和维护。前一个层次，着重在 TA 技

术上实现，在第 5 章也作了些介绍，而这里主要是讨论 TA 的管理框架。从管理上看，这个框架要解决 3 个方面的问题：

（1）测试脚本的管理，如何定义公共脚本库、项目特定的脚本库？是将脚本分为库函数层和应用层，还是数据层、逻辑层和表现层？

（2）测试自动化的运行，如何更有效地提交任务？如何及时查到测试结果或报告？如何充分利用硬件资源？

（3）与现有测试管理系统、源代码管理系统、软件包构建系统或文件管理系统等集成起来。例如当一个新软件包被构建之后，能自动安装和配置到测试环境上。

1. 脚本的管理

对于脚本的管理，肯定要分层次，这不仅有利于管理，而且可以提高脚本的开发效率。正如第 5 章所讨论的，脚本的开发和软件产品的开发非常接近，所以从这个意义上看，脚本也可以分为数据层、逻辑层和表现层。另外一种方法就是比较传统的作法，按功能模块化分：

（1）基础函数库（LIB），它里面有所有模块都需要调用的公共函数。

（2）各个模块特定拥有的函数，包括了单个逻辑操作。这相当于逻辑层，同界面没关系，而且每个操作是非常独立的，也就是说，可以在这层上实现关键字驱动的脚本函数。这层上的函数会调用 LIB 中的公共函数。

（3）在最上层针对具体项目临时构建应用脚本，这相当于表现层。这层依赖于项目的具体应用，并调用和项目有关系的模块函数，即调用单个逻辑操作。如果中间层脚本函数实现了关键字驱动，这层脚本就变为了单个逻辑操作的组合，每个组合构成一个测试用例的脚本。

2. TA 执行的管理

TA 执行的要求很清楚——测试人员在客户端就可以以 Web 方式提交测试任务、查看测试结果。进一步的要求就是，任务可以预先安排，如下班前提交一个晚上 9 点运行的测试任务，到了晚上 9 点，系统会自动启动任务执行。这就依赖于管理服务器、控制器和运行在测试机器上的代理等一系列设置来完成。

3. 集成

集成情况更复杂，因为每个公司用的第 3 方产品不一样。但有一点是清楚的，就是将测试用例、缺陷和测试任务等集中起来进行管理，即把 TA 运行的测试用例/任务和手工运行的测试用例/任务集中起来一起管理，TA 所发现的问题，能方便地将缺陷加入到缺陷库

中，已知的问题能链接某个缺陷。

图 11-6 就是一个示例，描述了整个 TA 管理框架。

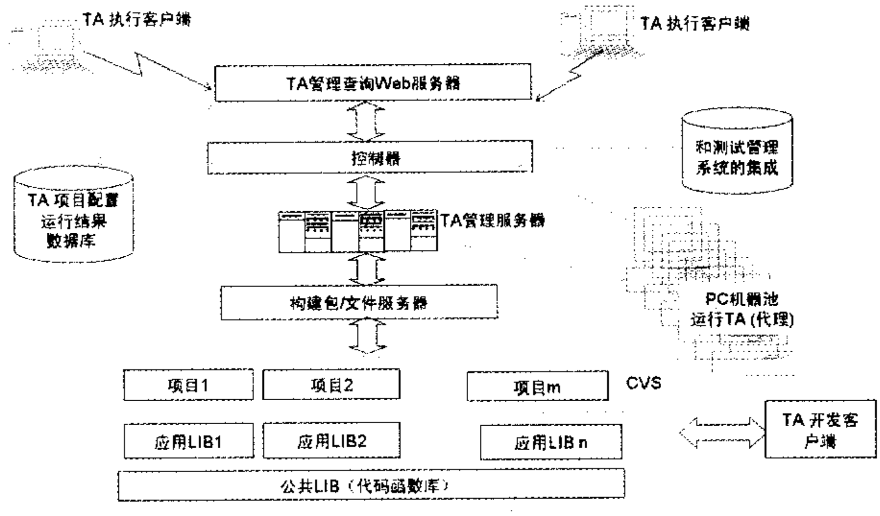


图 11-6 测试自动化的完整框架示意图

11.3.2 测试自动化的流程

测试自动化的流程，其目标是提高脚本开发效率和质量及 TA 执行的效率和质量。选择优秀的测试工具、构建合适的 TA 框架等，都有助于上述目标的实现，但流程的重要性是不可忽视的，如何通过流程来帮助这些目标的实现呢？

1. 脚本的开发

流程的定义，取决于脚本的开发模式和方法。脚本开发，一般可以分为两种模式：

- 根据产品需求和规格说明书，直接开发测试脚本。写测试脚本的同时，不断思考测试范围和测试点，借助注释来详细说明测试脚本设计的思路。这些注释相当于测试用例，可以被抽取出来转换为测试用例，从而进行评审。
- 根据产品需求和规格说明书的要求，先按惯例开发测试用例，然后基于测试用例开发测试脚本。

对于前面一种模式，测试用例评审比较迟。如果等所有脚本开发完了，再去评审，这时若是发现只丢掉了一些测试用例，结果还不算坏，补充这些测试用例的脚本就可以了。但若是发现设计思路、需求理解等不正确时，问题就大了，已写的许多脚本就没有用了。

对于这种模式，很明显，需要随时评审脚本及其相应的测试用例。

对于后一种模式，按照惯例进行测试用例的评审。也可以边评审边开发，即评审完一部分测试用例，随即为这部分测试用例开发脚本。这时，也会碰到另外一个问题，测试用例的结构、粗细颗粒度、格式是否适应TA的开发需求，且脚本能否全面、真实地反映测试用例，都需要仔细验证，尤其在验证点上。所以对于这种模式，应事先定义好测试用例的TA质量标准。其次，可以为测试用例增加一个TA属性来描述TA开发状态，监控其脚本的开发进程：

- M - Manual，手工的测试例子，不适合TA，或者还没有计划进行TA。
- S - Selected，适合TA的测试例子，开始进行脚本开发。
- A - Automated，脚本已开发完毕，等待测试工程师的验证。
- V - Verified，脚本通过验证，其TA测试的结果和手工测试的结果一致。

2. TA的执行

在流程上，TA的测试和手工测试有什么区别呢？在手工测试中，任务被分配到每一个具体的人，且手工执行测试用例，具有较大的灵活性，但执行速度很慢。而采用工具来执行测试，效率高，可靠，但缺乏灵活性和判断力，据此，其执行的流程可定义为：

- 所创建的测试套件应尽可能覆盖更多的测试用例。
- 为TA创建独立的任务，可以定期执行，如每日或每周定期进行。
- 对TA执行结果中没有通过的测试用例，要逐一复审或验证，确认是产品的问题还是脚本的问题。
- 发现任何产品问题，要及时报告缺陷。发现任何脚本问题，需要及时修正。
- 如果脚本没发现任何问题，需要重新评审脚本的测试范围、验证点的准确性等。
- 及时进行覆盖率分析，调整测试用例或测试范围。

11.4 缺陷跟踪和分析

根据质量特性的需求和测试目标，收集有关缺陷的各种数据，以进行宏观和微观的缺陷分析。

- 宏观分析，可以了解整体的测试效率、测试过程是否能达到已计划的预期目标。
- 微观分析，发现测试的漏洞，以及具体缺陷描述中所存在的问题。

测试漏洞反映了测试用例的不完善或者执行的不严格。如果是测试用例的不完善，应立即补充相应的测试用例。如果是执行的问题，则应完善执行的流程，加强监控，处理好需求或设计变更的问题。

11.4.1 缺陷生命周期

生命周期的概念是一个物种从诞生到消亡所经历的过程，同样，我们可以把这种概念赋予软件缺陷之生命。软件缺陷生命周期，经历了从被发现、报告到其被修复、验证、直至最后关闭的完整过程。为了描述软件缺陷生命周期，可设定不同的软件缺陷状态来体现缺陷不同的生命阶段。因此，对于一个软件测试人员来讲，需要关注软件缺陷状态的变化，并和开发人员保持良好的沟通，使缺陷能及时得到处理或修正。

在软件开发的实际过程中，软件缺陷生命周期并不是一个简单的线性过程——“发现 → 报告→ 修复→ 验证 → 关闭”，而是要考虑各种情况，包括：

- 缺陷描述不清楚，需要更多的补充信息。
- 缺陷不能再现，需要和测试人员的进一步合作。
- 缺陷需要审查，在即将要发布的版本中，有些缺陷不一定需要修正。
- 开发人员认为缺陷修正了，但是经过测试人员验证，缺陷还存在，需要重新置于激活（active）或打开（open）状态。

所以，软件缺陷的生命周期，如图 11-7 所示。

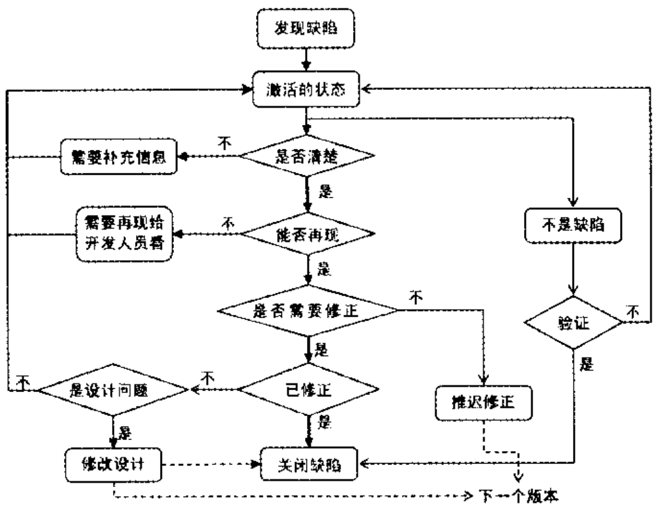


图 11-7 软件缺陷生命周期示意图

在软件生命周期中，通过邮件系统来及时通知缺陷状态的更新，例如，缺陷一旦被报出来，系统就会自动发送邮件给相应的开发人员；开发人员修正了缺陷并改变缺陷状态，系统也会自动发出邮件给相应的测试人员（缺陷报告者）。同时，可以建立各种查询机制，生成缺陷跟踪报告，随时可查到某个缺陷状态等。任何一个缺陷，在其生命周期中轨迹是清晰的、可跟踪的。

在生命周期中，缺陷经历了数次的审阅和状态变化，最终测试人员通过关闭软件缺陷来结束其生命周期。软件缺陷一旦被发现，便会受到严密跟踪或监控，直至被关闭，这样可保证在较短的时间内高效率地修正、验证和关闭所有的缺陷，缩短软件测试的进程，减少开发和维护成本。

技巧

在缺陷处理过程中，需要注意的一些细节。对图11-7的补充说明。

（1）软件缺陷生命周期中的不同阶段是测试人员、开发人员和项目经理协同工作的过程，要集体审查缺陷，保持良好沟通，尽量和相关的各方人员达成一致。

（2）测试人员在评估软件缺陷严重性和优先级上，应具有独立性、权威性。如果不能达成一致，最后由产品经理裁决。

（3）如果审阅者决定对某个缺陷描述进行修改，例如，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，由测试人员来修改缺陷报告，并添加相应的注释。

（4）当发现一个缺陷时，测试人员会将它分配给适当的开发人员。如果不知道具体的开发人员，可先分配给开发组长，由开发组长再分配给对应的开发人员。

（5）如果被修正的缺陷没有通过验证，那么测试人员会重新打开这个缺陷。在重新打开一个缺陷之前，最好和开发人员沟通并确认一下，至少加以详细的注释，否则会引起“打开 - 修复”操作被进行多个来回，造成测试人员和开发人员不必要的矛盾。

（6）一旦缺陷被置于“已修正”状态，不仅需要得到测试人员的验证，而且要围绕这个缺陷进行相应的回归测试，检查当前缺陷修复过程中是否引入了新的问题。

（7）只有测试人员有关闭缺陷的权限，开发人员没有这个权限。

（8）如果每个人都同意将确实存在的缺陷推迟到以后处理，应该指定下一个版本号或修改的日期。一旦开始开发新的版本时，这些推迟修正的缺陷应该被重新打开。

11.4.2 缺陷状态的跟踪

在单个缺陷经历上述的生命周期时，主要通过邮件系统集成获得跟踪，另外被跟踪的渠道是缺陷实时状态报告或其他查询报告，这种跟踪对单个缺陷来说比较被动。单个缺陷被跟踪的主要方式还是靠人，即这个缺陷的拥有者（owner）——修正缺陷的责任人（开发人员）和缺陷的报告者（测试人员），他们的责任心、敬业精神和沟通能力，都会保证单个缺陷得到最好的跟踪。

一个项目的缺陷整体跟踪，就是上面提到的缺陷实时状态报告。通过缺陷实时状态报告，在任何一个时刻，可以了解当前项目的缺陷状态，如表 11-2 所示，包括目前总共有多少个活动的（active）缺陷状态，各个级别又是多少，甚至包括每个开发人员有多少个缺陷需要修正（按名字列出），每个缺陷活了多少时间（处在 active 状态的时间）等。了解这些信息，对测试过程、项目进展的控制和管理都是有很大帮助的，可以督促开发人员尽快修正缺陷，可以调整测试或开发计划，如安排更多的测试人员参与测试或更多的开发人员修正缺陷等。在表 11-2 中，每个数据都是可以点击的，即可以打开详细的缺陷列表。

表 11-2 某个项目的缺陷实时状态报告

优先级	活动的	已修正	正在处理	Spec 需要修改	信息不够	不能再现	推迟修正	重复的	已知问题	有待增强	关闭的
P1	**0**	0	0	0	0	0	0	0	0	0	37
P2	**61**	75	33	1	0	4	4	5	1	17	1482
P3	**256**	13	5	0	0	1	0	0	0	1	71

缺陷年龄报告也是一种实时状态报告，可以实时显示缺陷处于活动状态的时间——已活了多少天数，从而了解哪些缺陷的修正被一拖再拖。例如，如果在报告中有超过 3 天的、优先级别为 P1 和 P2 的活动的缺陷，则说明这些缺陷没有得到及时修正，应和开发人员逐个查找原因，确定其预计修复的时间。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。

实际上，任何一种缺陷的属性都可以被跟踪，所以说，缺陷状态实时报告是测试过程管理中非常有力的工具。

11.4.3 缺陷的分析

通过缺陷的分析，可以了解更多的信息。可以根据缺陷状态来判断测试进展情况，开

发人员的编程质量，修正缺陷的进度。通过缺陷的分析，还可以完成产品质量的评估，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。通过缺陷的分析，还可以发现开发过程中存在的问题。无论对测试人员，还是对开发人员或管理人员，缺陷分析都是一项必不可少的工作。最常用的缺陷分析方法，可以分为两类——缺陷趋势分析和缺陷分布分析。

1. 缺陷趋势报告

缺陷趋势分析是缺陷的纵向分析，也就是在时间上对缺陷进行分析，有助于进度控制和测试过程的管理，应用最为广泛。缺陷趋势分析是考察缺陷随时间变化的趋势，如将各种状态（活动的、修正的、关闭的）的缺陷计数作为时间的函数来显示。缺陷趋势报告可以是实时的（如每日缺陷数、每周缺陷数随时间的变化曲线），也可以是累计的（如新报告的缺陷累计数和关闭的缺陷累计数比较曲线）。

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，缺陷增长率高。在达到顶峰后，缺陷会随时间以较低的速率下降，如图 11-8 所示。可以根据这一趋势复审项目时间表来检查进展情况。例如，测试周期是 4 个星期，缺陷率在第 3 周仍然处于增长状态，则很明显项目有问题，需要审查代码质量，调整时间表。

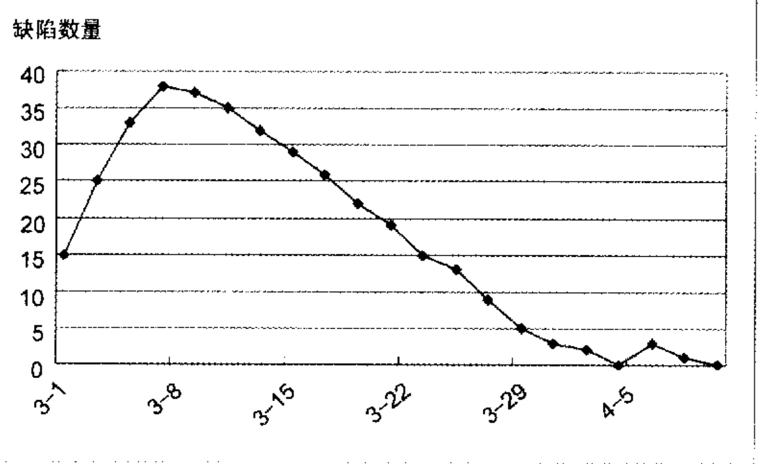


图 11-8 测试过程理想的实时缺陷趋势图

在实际测试过程中可能出现一些波动现象，这些波动现象甚至有一定的规律可循，如周一和周五发现的缺陷比较少，而周二和周三发现的缺陷会多些，所以实时缺陷曲线的数据按周来收集可能会更科学些，即每个数据点是一周所发现的缺陷数。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，其波动趋势会表现出不同的周期性。图 11-9 和图 11-10 是两个实际的实时缺陷趋势图，图 11-9 和理想趋势比较接近，意味着测试过程运行良好，而图 11-10 显示测试过程运行较差，和理想的趋势图所显示的结果差异较大，说明测试的进展不够顺利或测试的方法不够有效。

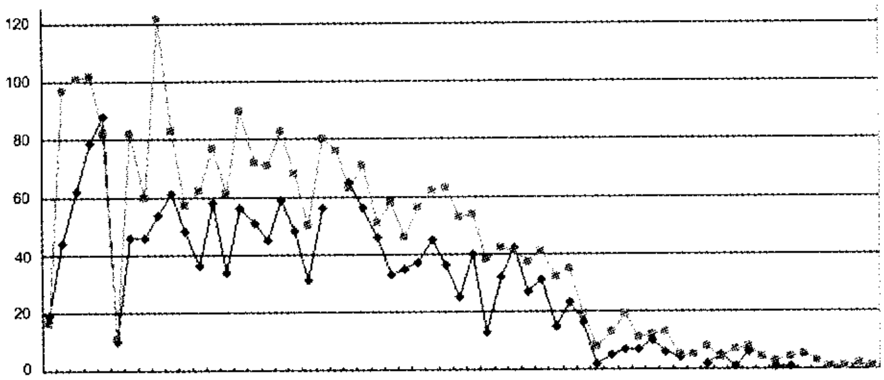


图 11-9　一个实际项目的实时缺陷趋势图（良好的测试过程）

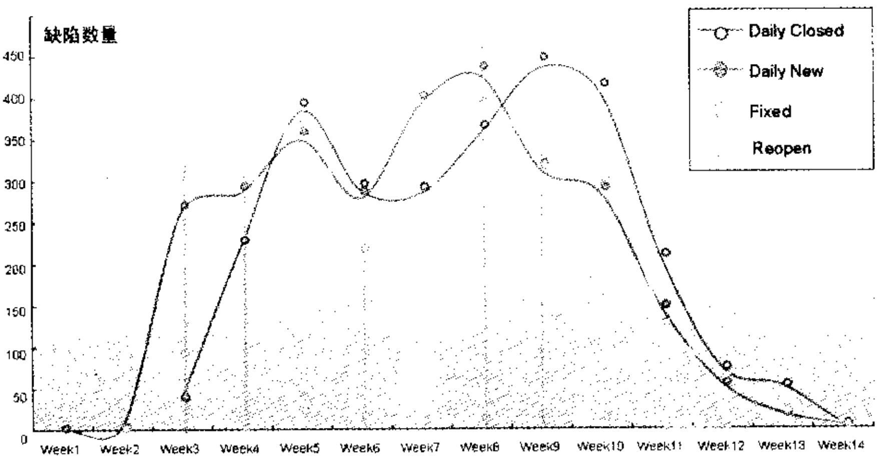


图 11-10　另一个实际项目的实时缺陷趋势图（较差的测试过程）

2. 缺陷分布分析

缺陷分布分析是缺陷的横向分析，也就是空间上的分析，一般在项目结束后进行。它可以针对一个或多个缺陷参数（严重性或优先级、来源、功能点、项目组等）进行缺陷分布分析，如功能上的分布分析，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；又如缺陷来源的分布分析，可以帮助找出缺陷产生的根本原因，包括需求定义、设计和编程上存在的问题等。

分析软件缺陷产生的根本原因，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频繁产生缺陷的领域。图 11-11 显示了软件缺陷产生的 3 个主要来源——用户界面显示、逻辑和规格说明，占据发现软件缺陷总数的 74%。如果从测试风险角度来看，这些区域可能是隐藏缺陷比较多的地方，需要进行更充分地测试。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了 10 000 个 Bug，代码在这 3 个区域减少一半，则共能减少 37%缺陷，即 3700 个缺陷，非常可观，代码质量可得到显著改善。

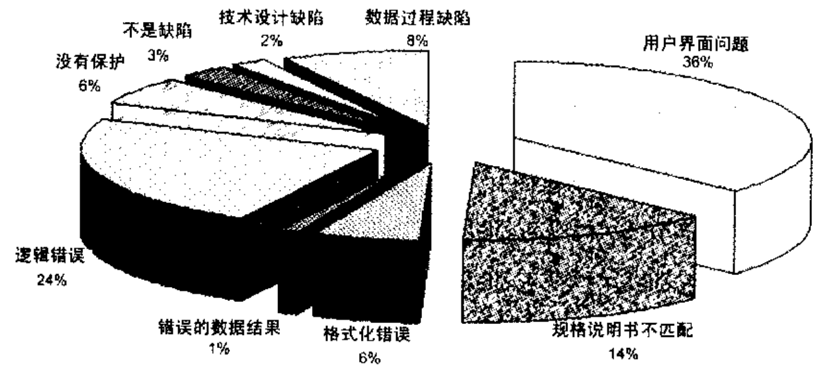


图 11-11 软件缺陷来源分析的分布图

缺陷在严重程度和优先级上的分布分析也是比较常见的，例如分析前 3 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 11-12 所示的优先级分布不符合上述正常规律，这说明代码质量不够好，需要进一步分析，以了解为什么会出现这种情况。进一步分析，可以通过计算出 P1 优先级缺陷从报告到关闭所需要的平均时间知道开发人员是否按照要求去做，一般来说必须在 8 小时之内解决 P1 优先级缺陷规定。

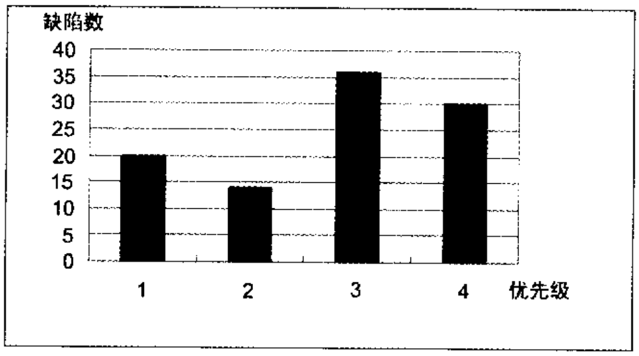


图 11-12 缺陷优先级分布示意图

11.4.4 累计缺陷趋势分析

在理想情况下，发现新的缺陷的速度快，缺陷关闭的速度也快，但随着时间推移，该速度不断降低，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，如图 11-13 所示。但已修复的、已关闭的缺陷数会有滞后现象，因为修正一个缺陷需要做较多的工作，如问题分析、Debug 和单元测试等；在缺陷修正后，还要构建和安装新的软件包、验证修正的结果等。对优先级为 P1、P2 的缺陷，应及时修复、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，只有这样，才能更好地保证项目顺利进行。

在测试和修复的后期，这 3 条曲线不断地收敛，当收敛成一个点时，开发人员就完成了缺陷修正的任务。如果“关闭的”缺陷曲线紧跟在“激活的”缺陷曲线的后面，这表明项目小组正在快速地推进问题的解决。图 11-13 给出的缺陷曲线是理想的趋势图，A 点所在的斜线就是一条切线，这说明了发现缺陷的速度，斜线越陡（和水平轴夹角越大）发现缺陷越快，斜线越平发现缺陷越慢。所以到了 B 点，就开始找不到什么缺陷了。从 B 点到 C 点这一段时间，几乎没发现多少缺陷，属于稳定时期。如果测试方法和效率等水平很高，没什么问题的话，C 点就是软件发布之日。

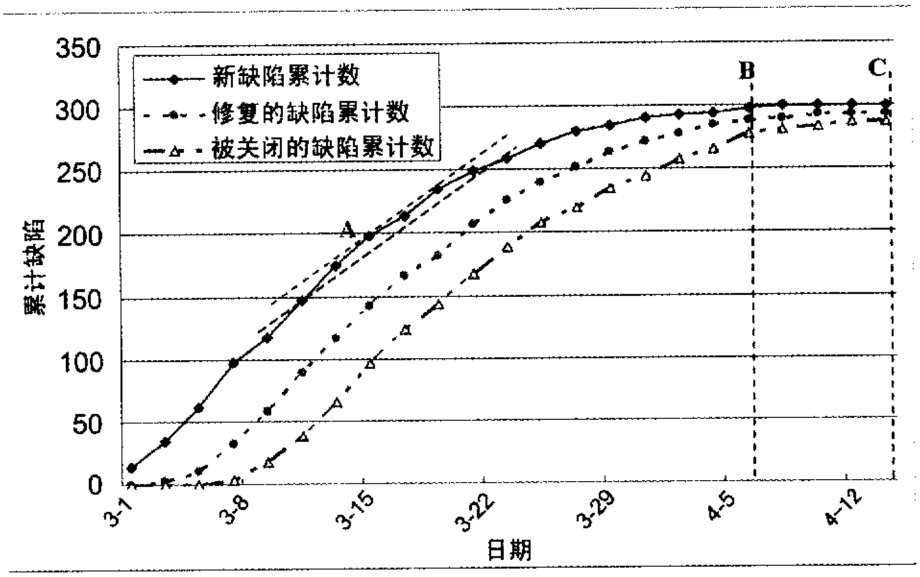


图 11-13　新发现的、修复的、关闭的累计缺陷数的理想趋势图

但是实际情况很难达到如此完美的地步，图 11-14 所示的就是一个实际项目的累计缺陷趋势曲线。如果它与图 11-13 的趋势差别显著，则表明软件开发过程中可能存在下列问

题：

- 缺陷处理流程有问题。
- 修正缺陷引起了更多的回归缺陷。
- 有较多的缺陷被遗漏到下一个阶段。
- 修复缺陷所需的资源不足。
- 回归测试策略不对等。

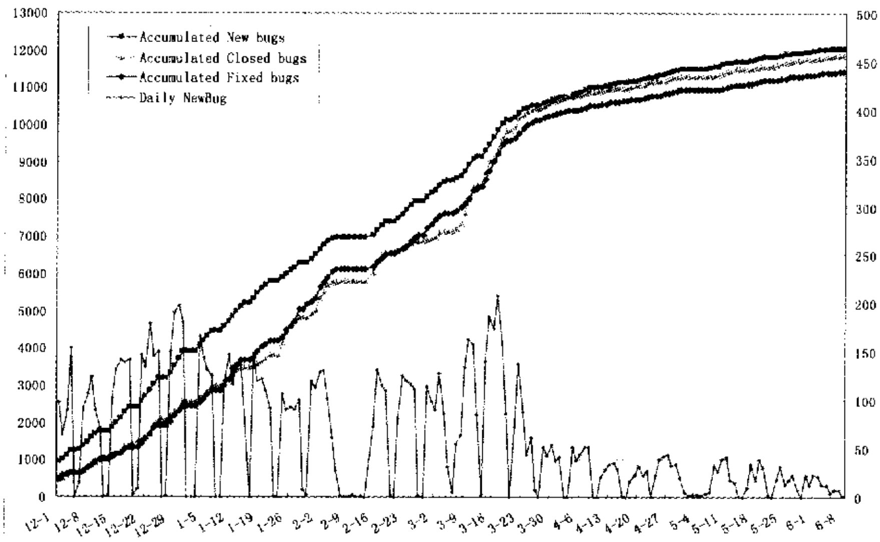


图 11-14 一个实际项目的实时缺陷趋势图

根据研究结果，可以建立更准确的数学模型，如 Rayleigh 模型。经验告诉我们，在缺陷出现累计模式的情况下，形状参数常为 m=1 和 m=2，如图 11-15 所示。

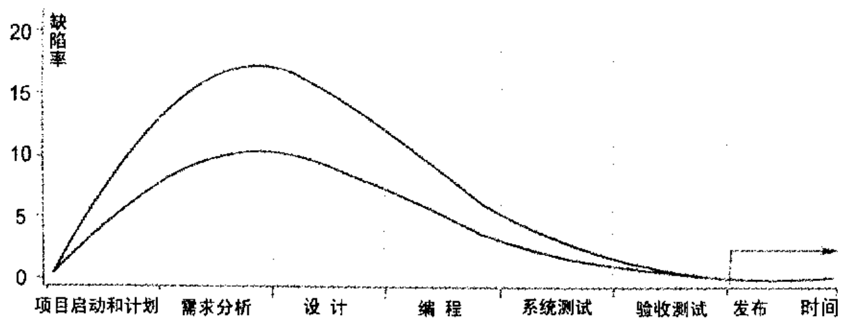


图 11-15 软件开发过程的 Rayleigh 缺陷模型形状

11.5 测试进度和风险的控制

测试的执行过程处在测试工作的后期，所以前期工作的延迟，如产品设计规格说明书评审、编写代码等，会对测试工作影响很大。从测试管理上看，测试的进度和风险控制是一个比较大的挑战，需得到高度的重视，必须做好这方面的管理工作。

11.5.1 测试进度管理

测试进度的管理是一门艺术，是一个动态平衡的管理过程，需要不断调度、协调和调整，以保证项目均衡而流畅的发展。在起草项目测试计划书时，对测试需求、测试范围以及测试工作量等有一个大体的认识，但深度不够。最初的测试进度表可能只是一个时间上的框架，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，在通过不断地和各方讨论、交流后，对测试的需求、范围以及工作量等会有深刻的认识，对很多问题估算都不再停留在比较粗糙的层面上，可以越来越详细、越来越准确地制定项目进度表。这也就是为什么我们一直强调“测试计划”是计划的过程，而不仅仅是一个文档的原因。

任何一项工作，刚开始总是很容易看到进度变化，就比如盖房子，从无到有，变化是很明显的。可是越到后来，它的进度变化越来越不明显。软件测试也是如此，测试开始之初，缺陷比较容易发现，但测试的进展并不是按缺陷的数量来计算的，越到后面，缺陷越来越难发现。要提高测试进度的质量，应在第一时间发现严重的问题，这样才不至于让开发人员在最后阶段对代码做大规模的修改，而且这样一来也无法保证对大规模改动进行测试的时间，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，这就要求我们在关注进度的同时把握好发现缺陷的效率和深度，在注重项目推进速度的同时还要关注进度前期所做工作的质量。

测试的进度管理以质量为首要目标，要把握好进度与质量、成本的关系，需通过对里程碑、关键路径的控制并借助测试管理系统来实现。

1. 里程碑进出标准的确认

对测试过程的每个里程碑都有明确的定义，其关键内容就是里程碑的结束标准，如测试计划和测试用例的评审结束标准可定义为：

- 由项目组有关人员（产品、开发、测试等人员）集体从头到尾审查至少一遍。
- 所有问题都被澄清或修正。
- 大家对所有内容的理解完全一致。

最值得关注的里程碑是完成测试或测试结束，如何定义“测试结束”的标准呢？这实际上是一个很难的问题。即使在过去的一周内没有发现新的缺陷，也不能说明产品不存在缺陷，因为测试方法不正确、测试水平不够等都会限制测试的能力，从而导致始终不能发现某些缺陷或者不能覆盖某些测试范围。从一般意义看，测试结束的标准可定义为：

- 所有计划的测试都已完成。
- 测试的覆盖率达到要求（可以借助工具度量代码行/块、分支/条件的覆盖率）。
- 发现的缺陷逐渐减少，直至有一段时间（一周左右）没有发现任何严重缺陷。
- 所有严重缺陷已被修正，并得到验证。
- 没有任何不清楚、不确定的问题。

2. 测试进度的管理方法

（1）随着测试时间的推移，缺陷发现的速率会降低。如果用新发现缺陷的累计曲线、实时周发现缺陷曲线等来进行跟踪，可以比较容易地控制测试进度。当累计曲线趋近于稳定时，也就预示着测试进入了尾声。

（2）尽量利用历史数据，可以与以前完成过的项目进行类比分析，以确定质量和进度所存在的某种数量关系，进而控制进度和管理质量。

（3）可以采用进度管理计划关联质量参数的方法，也就是通过参数调整进度和质量的关系。

（4）可以采用测试项目进度的度量方法：测试进度 S 曲线法和缺陷跟踪曲线法。在进度压力之下，被压缩的时间通常是测试时间，这导致实际的进度会随着时间的推移而推移，与最初制定的计划相差越来越远。而如果有了正式的度量方法，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。

概念

进度 S 曲线法是通过对计划中的进度、尝试的进度与实际的进度三者进行对比来实现的，其采用的基本数据主要是测试用例或测试点的数量。同时，这些数据按周统计的结果反映在图表中。“S”的意思是，随着时间的发展，积累的数据的形状越来越像 S。可以看到一般的测试过程中包含 3 个阶段：初始阶段、紧张阶段和成熟阶段。第一个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，所以导致曲线的形状像一个扁扁的 S。

（1）用趋势曲线（上方实线）代表计划中的测试用例数量，它是在实际测试执行之

前画上的（如图11-16所示）。

（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，浅色柱状数据代表截至本周累计尝试执行的测试用例数，深色柱状数据代表截至本周累计实际执行的测试用例数。

（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，而成功执行的测试用例则略低于原计划，这种情况是经常出现的。

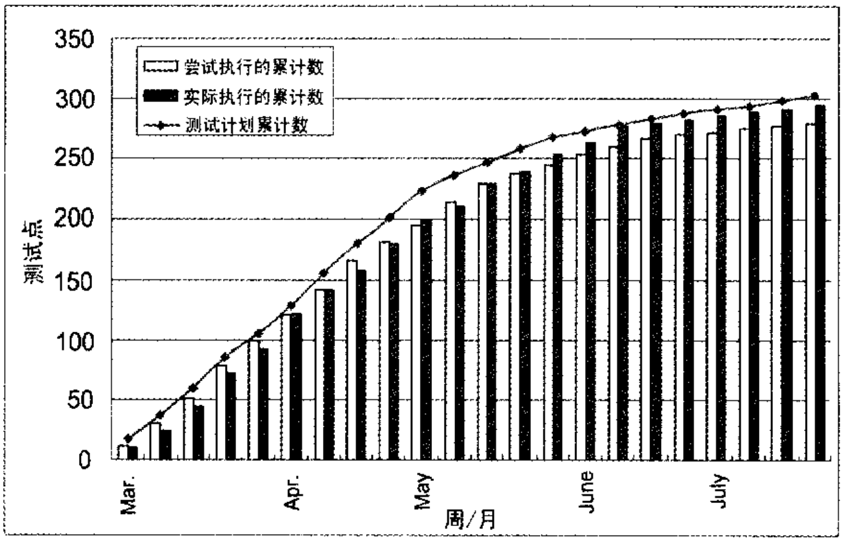


图 11-16

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）可以让 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）。

一旦把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。严格的评估是 S 曲线成功的基本保证，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，当计划或者尝试数与实际执行数之间存在 15%到 20%的偏差时，就需要启动应急行动来进行弥补了。

3. 测试进度的管理重在前期

正如前面所述，测试受前期工作影响较大，要保证及时地完成测试任务，一定要把管理工作的重点放在前面，而不是在测试执行的后期。在测试执行阶段，测试团队掌握着比较大的主动权，缺陷数据是量化的，比较容易推动开发修正缺陷的速度，而在前期，设计

人员、编程人员占据着主动的地位，测试人员要影响或控制他们的工作进度是很难的，这时就需要有更好的办法来控制他们的进度和工作的质量。

- 在公司层次上，建立“100%兑现承诺”的企业文化氛围，使每个人对里程碑都非常重视，让他们尽自己的努力及时地完成任务。
- 在公司层次上，建立良好的质量文化，使每个人不仅能及时完成任务，而且能高质量地完成任务。从开发人员的角度来看，就是要做好代码检查，进行足够的测试。
- 测试人员提供已设计好的测试用例和脚本给开发人员，帮助开发人员自我验证，进行更多的测试以提高交付内容的质量。
- 测试人员应与开发人员保持良好的沟通，并及时地从开发人员那里拿到部分完成的代码包，阅读代码或进行 Ad-hoc 测试。了解代码质量和实际的开发进度，有助于及时发现进度滞后迹象。

11.5.2 测试风险的控制

测试风险是不可避免的，它总是存在。测试是一项高风险的工作，所以对测试风险的控制是非常必要的。在测试管理中，必须尽力降低测试中所存在的风险，要最大程度地保证质量和满足客户的需求。在测试工作中，主要的风险有：

（1）对质量需求或产品的特性理解不准确，会造成测试范围分析的误差，结果某些地方始终测试不到或验证的标准不对。

（2）没有 100%地执行测试用例，如有些测试用例被有意或无意地遗漏。

（3）需求的临时/突然变化，导致设计的修改和代码的重写，从而使测试时间不够。

（4）质量标准并非都很清晰，如适用性的测试，仁者见仁，智者见智。

（5）测试用例设计不到位，忽视了一些边界条件、深层次的逻辑关系、用户场景等。

（6）测试环境，一般不可能和实际的运行环境完全一致，这使测试结果产生误差。

（7）有些缺陷出现的频率不是 100%，不容易被发现。

（8）如果代码质量差，软件缺陷很多，存在漏检缺陷的可能性就大。

（9）回归测试一般不会运行全部测试用例，它是有选择性地执行的，所以也必然带来风险。

前面 3 种风险是可以避免的，而后 6 种风险是不能避免的。虽然从理论上说，回归测试风险是可以避免，但由于时间或成本的限制，一般也是难以避免的。对于不能避免的风险，我们的目标就是将风险降到最低水平。

针对上述软件测试的风险，可以采取一些有效的测试风险控制方法，如：

- 测试环境不对，可以事先列出要检查的所有条目（如检查表），在测试环境设置好后，由其他人员按已列出的条目逐条检查。
- 有些测试风险带来的后果可能非常严重，可将它转化为其他一些不会引起严重后果的低风险。如产品发布前夕，在某个不是很重要的新功能上发现一个严重的缺陷，如果修正这个缺陷，很有可能引起某个原有功能上的缺陷。这时处理这个缺陷所带来的风险就很大，可以通过不激活（Diasble）那个新功能来转移这种风险。
- 如果有些风险不可避免，那就设法降低风险，如“程序中未发现的缺陷”这种风险总是存在，我们可以通过提高测试用例的覆盖率（如达到 99.9%）来降低这种风险。

为了避免、转移或降低风险，事先要做好测试风险管理计划和制定控制风险的策略，还须在风险的处理上制定一些应急的、有效的方案，如：

- 在做资源、时间、成本等估算时，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可能会有变化、难以控制的因素列入风险管理计划中。
- 为每个关键性的技术人员培养后备人员，作好人员流动的准备，采取一些措施确保一旦人员离开公司，项目仍能继续下去，不会受到严重影响。
- 制定文档标准，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，包括将不同的测试人员在不同的测试模块上进行调换。
- 对所有过程进行日常跟踪，及时发现风险出现的征兆，避免风险的产生。

要想真正回避风险，需要在企业文化、管理模式等高度上来预防。针对测试的各种风险，建立一种“防患于未然” 或“以预防为主”的管理意识。与传统的软件测试相比，广义的测试概念——静态测试和动态测试组合的全过程测试管理方式，不仅可以有效降低产品的质量风险，还可以提前对软件产品缺陷进行规避，缩短缺陷的反馈周期和整个项目的测试周期。

知识点

风险管理，一般可以分成5个步骤，即风险识别、风险分析、风险计划、风险控制以及风险跟踪。

（1）风险识别是试图用系统化的方法来确定威胁项目计划的因素。识别方法包括风险检查表、头脑风暴会议、流程图分析以及与项目人员面谈等。

（2）风险分析可以分为定性风险分析和定量风险分析。定性风险分析是评估已识别风险的影响和可能性的过程，以根据风险对项目目标可能的影响（包括风险的性质、范围和时间）对风险进行排序。定量风险分析用于量化分析每一种风险的概率及其对项目目标造成的后果（加权系数或损失代价），同时也要分析项目总体风险的程度。

（3）制定风险行动计划，包括参考计划、基准计划和应急计划等不同类型的风险防范计划，着重考虑风险责任、可用资源、时间、活动、应对措施等。建立示警的阈值是风险计划过程中的主要活动之一，阈值与项目中的量化目标紧密结合，并定义该目标的警告级别。

（4）风险控制方法，主要有风险避免、风险弱化、风险承担和风险转移等。

- 风险避免。通过变更软件项目计划消除风险或风险的触发条件，使目标免受影响。这是一种事前的风险应对策略。例如，采用更熟悉、更成熟的技术，澄清不明确的需求，增加测试的资源和时间，减少新功能等。
- 风险弱化。将风险事件的概率或结果降低到一个可以接受的程度，当然降低概率更为有效。例如，选择更简单的开发流程，进行更多的系统测试等。
- 风险承担：表示接受风险，不改变项目计划（或没有合适的策略应付风险），而是考虑当风险发生时如何应对。例如制定应急计划，编写风险应变程序，甚至仅仅进行应急储备和监控，在发生紧急情况时随机应变。可以事先制定应急计划，保障有后备人员可用（如软件项目正在进行中，以避免有人要离开测试组），同时可确定项目组成员离开时交接的程序。
- 风险转移：不去消除风险，而是将软件项目风险的结果连同应对的权力转移给第三方。这也是一种事前的应对策略，属于下策。

（5）在风险受到控制以后，要及时进行跟踪，监视风险的状况，例如，风险是已经发生还是仍然存在或已经消失。如果风险存在，则检查风险的对策或措施是否有效。可以通过以下几种方法进行有效地风险跟踪：

- 风险审计，检查监控机制是否得到执行，尤其对项目关键处的事件进行跟踪，

对主要风险因素进行跟踪。

- ➢ 风险的再评估指对没有预计到的风险制定新的应对计划。
- ➢ 偏差分析是指将应定期与基准计划进行比较，分析质量、成本和时间上的偏差。
- ➢ 技术指标分析是指比较原定技术指标和实际技术指标差异，例如测试未能达到性能要求等。

11.6 测试覆盖度和结果分析

测试评估是软件测试的一个阶段性的结论，它用生成的测试评估报告来确定测试是否达到标准。测试评估可以说贯穿了整个软件测试过程，可以在测试的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。软件测试评估，主要有两个目的：

- 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告一段落。
- 为最后的测试或质量分析报告生成所需的数据，如缺陷清除率、测试覆盖率等。

如果没有测试评估，就没有测试覆盖率的结果，就缺乏测试报告的依据。

11.6.1 测试覆盖评估

系统的测试活动至少建立在某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，以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合格的测试评估的量化指标。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完成评估，如已经核实了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的标准。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。测试的覆盖率的测试指标，可以根据表 11-3 来评估。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 11-3 测试覆盖项的例子

测试覆盖项	测试覆盖率指标测试描述
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

续表

测试覆盖项	测试覆盖率指标测试描述
动态功能覆盖	所有功能的转换正确
正常测试覆盖	所有硬件软件的正常处理
异常测试覆盖	硬件或软件的异常处理（不允许的操作）

1. 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化为评估测试用例的覆盖率——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，考虑质量要求、风险和严重性，制定不同的测试覆盖标准。

在执行测试活动中，评估测试用例覆盖率又可以进一步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的覆盖率，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_{ft} 是测试需求的总数：

$$已执行的测试覆盖 = Tx / R_{ft}$$

（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 Ts 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。

$$成功的测试覆盖 = Ts / R_{ft}$$

在实际过程中，很难确定 R_{ft} 的值，可以根据需求点和以往的经验来估算。

2. 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。如果是应用基于代码的覆盖，则测试策略是根据测试已经执行的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，以及在测试中真正执行的代码行数及其百分比，并将此结果记录在测试评估报告中。

与测试过程中已经执行代码的多少相对的是要执行的剩余代码的多少。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：

$$已执行的测试覆盖 = Tc / Tnc$$

其中 Tc 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来表示的已执行项目数，Tnc（Total number of items in the code）是代码中的项目总数。

知识点

软件需求的估算

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是唯一功能需求的数目，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的功能百分比，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表示已经确认是正确需求的个数，Fnv是尚未被确认的需求的个数。

11.6.2 基于软件缺陷的质量评估

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手段之一。

软件缺陷评估的方法相对比较多，从简单的缺陷计数到严格的统计建模。基于缺陷分析的产品质量评估方法有以下几种：

- 缺陷密度指缺陷在软件规模上的分布。它表示每 KLOC 或每个功能点（或对象点、特征点等）的缺陷数。缺陷密度越低意味着产品质量越高。
- 缺陷率指缺陷在时间上的分布。它是一定时间范围内的缺陷数与错误几率（OFE，opportunities for error）的比值。
- 整体缺陷清除率、阶段性缺陷清除率。
- 缺陷趋势、预期缺陷发现率。
- 软件产品性能评估技术。
- 种子方法、现成工具或其他方法。

1. 缺陷密度

Myers 有一个著名的关于软件测试的反直觉原则：在测试中发现缺陷多的地方，还有更多的缺陷将会被发现。这个原则的背景就在于：发现缺陷多的地方，漏掉的缺陷可能性也会越大。在测试效率没有被显著改善之前，在纠正缺陷时将会引入较多的错误。

- 如果缺陷密度跟上一个版本相同或更低，就应该分析当前版本的测试效率是不是降低了？如果不是，意味着质量的前景是乐观的；如果是，那么就需要额外的测试，还需要对开发和测试的过程进行改善。
- 如果缺陷密度比上一个版本高，那么就应该考虑在此之前为显著提高测试效率进行的有效策划是否在本次测试中得到实施？如果是的，虽然需要开发人员更多的努力去修正缺陷，但质量还是可以得到更好的保证；如果没有，意味着质量恶化，质量很难得到保证。这时，要保证质量，就必须延长开发周期或投入更多的资源。

要点

测试效率和缺陷数量的测试过程相关矩阵模型，就是将过程度量值和测试结果的缺陷度量拿来一起研究（如图11-17所示）。

- 情形 1 是最好情况，显示了软件良好的内在质量——开发过程中的错误量低，并通过了有效的测试验证。
- 情形 2 是一个较好的场景，潜伏的、较多的缺陷通过有效的测试发现。
- 情形 3 是不确定的场景，我们可能无法确定低缺陷率是由于代码质量好还是测试效率不高而造成的结果。通常，如果测试效率没有显著恶化，低缺陷率是一个好的征兆。
- 情形 4 是最坏的场景，代码有很多问题，但测试效率低，很难及时发现它们。

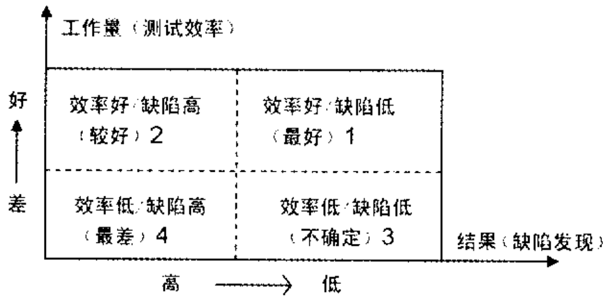


图 11-17

2. 软件产品性能评估

对软件产品性能进行评估，其技术性相对比较强。这种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计算结果。主要的性能评测包括：

（1）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，从而监测或评估性能测试执行情况。

（2）响应时间 / 吞吐量。针对特定条件下与某个测试对象相关的测量特性的行为，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。

（3）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指标的影响，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。

11.6.3　软件缺陷清除率

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。

对于一个应用的软件项目，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，若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= 23/100= 0.23（23%）
- 整体缺陷清除率= D1/D= 20/23= 0.8696（86.96%）

有资料统计显示，过去美国软件业的平均整体缺陷清除率只约达 85%，而对一些具有良好的管理和流程等的著名软件公司，其主流软件产品的缺陷清除率可以超过 98%。

阶段性缺陷清除率用于测试缺陷密度度量的扩展，可跟踪开发周期所有阶段中的缺陷，包括需求评审、设计评审、代码审查和测试等。因为编程缺陷中的很大百分比是同设计问题有关的，进行正式评审或功能验证以增强前期过程的缺陷清除率，有助于减少出错的注入。基于阶段的缺陷清除模型反映开发工程总的缺陷清除能力。

进一步分析缺陷清除有效性（DRE，Defect Remove Efficiency），DRE 可以定义为：

$$\frac{\text{开发阶段清除的缺陷数}}{\text{产品中潜伏的缺陷数}}\times 100\%$$

因为潜伏缺陷的总数是不知道的，所以必须通过一些方法获得其近似值，如经典的种子公式方法。当用于前期的和特定阶段的时候，DRE 相应地被称为早期缺陷清除有效性和阶段有效性，对给定阶段的潜伏缺陷数，可以估计为：

当前阶段的潜伏缺陷数 = 当前阶段排除的缺陷数＋以后发现的缺陷数

给定阶段的 DRE 度量值越高，遗漏到下一个阶段的缺陷就越少。

这样，阶段性的 DRE 又可以定义为：

$$\frac{\text{排除的缺陷数（这一步）}}{\text{这一步入口处存在的缺陷数}+\text{（这一步）开发过程中注入的缺陷数}}\times 100\%$$

清除的缺陷数等于检测到的缺陷数减去不正确修正的缺陷数。如果不正确修正的缺陷数所占的比例很低（经验数据表明，测试阶段大概为 2%），清除的缺陷数就近似于检测到的缺陷数。

示例

众所周知，清除软件缺陷的难易程度在各个阶段不同。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11-4所示。

表 11-4 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11-5反映的是CMM 5个等级是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 11-5 SEI CMM 级别潜在缺陷与清除率

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

11.6.4 测试报告的模板、实例

在国家标准 GB/T 17544－1998 中（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）有了一个清楚的描述，可对测试纪录、测试结果如实汇总分析，并报告出来。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。

（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，产品未作符合性测试的说明。

（7）测试结束日期。

主要内容集中在第四项内容中，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和性能、易用性、可维护性、 可移植性等的说明。

特别是在功能说明中，包括安装、功能表现以及功能使用的正确性、一致性；不仅需要概述产品的用户可调用功能、需要的数据等，还要将系统相应的边界值、安全性要求描述清楚。易用性说明中，包括易理解性、易浏览性、可操作性等方面，具体描述对用户界面、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。对于可靠性，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。

示例

一个测试分析报告的标准模板

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的介绍、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）结果与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果和发现。

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

列出测试结果的分析记录，并按所定义的模板产生BUG分布表和BUG分布图。用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。如:

5.1.1 对比分析

若非首次测试，将本次测试的结果与首次测试、前一次测试的结果进行对比分析。

5.1.2 测试评估

通过对测试结果的分析提出一个对软件能力的全面分析方案，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本软件的能力。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，以及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，并说明测得的全部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。

11.7 小结

对一个成熟的软件公司来说，测试管理在先，测试活动在后，即先有一套流程、过程跟踪方法等，然后开展测试活动，主动收集数据，进行分析并不断改进测试流程。测试管理的全局性，指的是不要忽视任何一个环节，不要轻视任何一个细节。从产品需求文档（PRD，包括 UI Mock-up）审查开始到产品发布，实施对测试全过程的跟踪和管理，它所涉及的内容非常广泛，其中主要讨论了下列内容。

（1）基于两阶段的测试执行策略，前阶段就是为了发现缺陷，强调测试效率；后一阶段，强调覆盖面，为的是降低风险。

（2）测试管理系统以测试用例库、缺陷库为核心，覆盖整个测试过程，并在测试用例、缺陷之间建立必要的映射关系，在选用商业或开源的测试管理工具之时，建议选用开源的测试管理工具，包括缺陷管理工具。

（3）测试用例的管理涵盖测试用例的创建、执行和维护，强调意识和态度的教育，它将模块划分清楚，责任到人，并在设计方法和流程上加强管理，例如可以按照产品线、测试目标和功能模块等进行分类、组织和存储，以利于测试用例的执行和维护。

（4）测试自动化管理需要一个良好的框架，从脚本开发到运行，力求层次清楚，操作方便，并能与测试执行、Bug 跟踪系统集成，不断改进测试自动化的流程。

（5）缺陷的跟踪和分析，从定义软件缺陷的生命周期开始，然后建立各种缺陷实时状态报告，了解当前项目任何一个时刻的缺陷状态；并通过缺陷趋势分析和分布分析，在时间上和空间上全面了解缺陷的规律和软件开发中的问题。找出缺陷产生的根本原因，有助于进度控制和测试过程的管理。

（6）测试进度管理，以质量为首要目标，要把握好进度与质量、成本的关系，通过对里程碑、关键路径的控制，并借助测试管理系统来实现。同时，它也是一门艺术，一个动态平衡的管理过程，需要不断调度、协调和调整，以保证项目均衡而流畅的发展。

（7）测试风险的控制是非常必要的，必须尽力降低测试中所存在的风险，最大程度地保证质量和满足客户的需求。要想真正回避风险，更重要的是建立一种“防患于未然”

或“以预防为主”的管理意识。

（8）测试评估工作主要是对测试覆盖率的评估，包括基于需求的测试覆盖评估和基于代码的测试覆盖评估。基于需求的测试覆盖评测可转化为评估测试用例的覆盖率。

（9）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，包括对缺陷密度、缺陷率、整体缺陷清除率、阶段性缺陷清除率、缺陷趋势、预期缺陷发现率和软件产品性能等一系列度量结果的分析。

12

第12章 总结和思考

没有总结，就不认识自己，就不知成功在哪里，失败在哪里。没有思考，就没有提高，就没有进步。

最后，我们要对软件测试进行思考和总结，帮助大家进一步获得提升。这种思考更多体现在理念上，或者在思想高度上。例如，我们需要思考下面问题：

- 软件测试能做什么？
- 软件测试不能做什么？
- 测试面对的挑战是什么？
- 软件测试的底线是什么？
- 如何高效地执行测试？

在实际测试过程中，有时存在许多窍门，这些窍门没有很多道理，而是根据实际工作中所得到的教训和经验总结出来的。最常见的一种方法就是每个项目结束时进行总结分析

（Postmortem），了解哪些地方做得比较好，哪些地方还有待提高。对有待提高的地方应采取什么措施，而在成功的方面，有哪些最佳实践可以保留下来，以用于今后的工作。

12.1 软件测试的现实和原则

在总结和思考的开始，我们就试图回答本章一开头所提出的问题。测试所面对的现实清楚地显示了软件测试会面临什么样的挑战，软件测试能做什么、不能做什么，从而迫使我们为软件测试建立一些基本的原则。这些原则能指导测试工作，在总体上把握测试的方针和方法。

12.1.1 测试的现实

测试始终是一个具有风险的工作。现在，越来越多地用“风险”概念来定义测试，测试被认为是“理解并评估与发布的软件系统有关的利益和风险状况的过程”，测试的作用则是管理或转移系统失败的风险，以及如何尽最大可能地消除给用户带来的不良影响。

测试工作为什么总存在风险呢？因为当我们测试某个应用系统或软件产品时，不可能把所有可能的情况都测试一遍。

示例

例如，即使测试一个计算器程序，要测试的数字可以从 0 开始到一个很大的数，就算 8 位数字（99999999），仅仅测试其加法运算的可能情况就是 10^{16}，要完成这些测试，即使借助计算机，每秒完成 10 万个测试用例，一个测试人员穷其一生也完成不了，因为需要 3100 多年。如果再加上负数、减法、乘法、除法、括号以及它们的各种组合，所有全部可能的情况将是一个巨大的天文数字，所以完成全部可能的情况是不可想象的。

所以，我们必须借助一些测试用例的设计方法，如边界值分析方法、等价类划分方法等来解决这个问题，通过选择样本数据，用极有限的、具有代表性的测试数据来代替实际的、巨大的测试数据。这些方法的应用是基于一个假设：如果程序在这些样本数据下运行正确，那么该程序也满足所有类似的数据。这种假设的存在，也就意味着一定的风险存在。对于简单的程序，这种风险很小，但是对越来越复杂的应用程序或软件系统，这种风险就越来越大。

即使完成了全部功能的测试，也很难完成所有用户环境下的测试。测试的环境是有限

的，而软件系统的实际运行环境是复杂的、千变万化的，不仅有不同的硬件（主板、CPU、内存、网卡、显示卡等）型号的差异，而且还有操作系统及其版本、驱动程序及其版本、已安装的应用程序等的差异。完成各种用户环境下的软件测试，也几乎是不可能，即使可以实现，其成本也是巨大的，一般软件企业难以承受。同样，系统的性能测试、有效性测试和可靠性测试等蕴含着较大的风险。例如，可靠性测试是通过模拟方法实现的，不能在完全真实的情况下进行，更不可能对真实系统连续进行 10 年或 20 年的不间断测试。

测试工作，除了始终存在的风险之外，还会受到其他多方面的挑战，主要有：

- 测试不能提高质量。软件产品发布后缺陷较多，往往被认为是测试人员的错。在许多人的心目中，测试人员是防止缺陷产生的盾牌或最后一堵墙。实际上，所有的软件缺陷都是在需求分析、设计和编程时被注入进去的，注入的缺陷越多，漏掉缺陷的可能性就越大。
- 测试人员的素质和待遇。国内还存在对测试理解的误区，如测试不需要技术或不需要过高的技术。所以在选用测试人才时，往往降低要求，所给的待遇也偏低，从而造成测试队伍的整体能力比较弱，测试人员的工作热情比较低，这对测试质量有较大的负面影响。
- 测试时间往往被压缩。虽然一旦软件项目启动，测试工作就会开始，包括产品需求文档审查、产品规格说明书审查、测试计划/用例的设计等，但是软件测试的主要执行时间是在代码完成后。由于对开发软件的日程估计不够准确，往往会导致代码完成的时间延迟，但管理层又不想推迟整个产品的发布日期，结果测试时间首当其冲被缩减，测试周期被缩短，造成测试不够充分或已计划好的测试项目不能保质保量的完成。

总之，测试工作所面临的挑战比较大，有时可以说是严峻的，但是，只要我们敢于面对现实，坚持测试原则，就能克服困难，我们需要运用正确的、有效的测试方法和工具，来设计全面的、完整的测试用例和测试数据，并制定有效的测试策略，进行充分的测试，使测试工作的质量得到保证。

12.1.2 测试的原则

原则是最重要的，方法应该在这个原则的指导下实施。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，并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（bug-free, zero-bug）是一种理念，足够好（Good-Enough）是测试的基本原则。在软件测试过程中，应注意和遵循一系列的具体原

则。

1. 想用户所想

在所有的测试活动过程中，都应该从客户的需求出发，想用户所想。正如我们所知，软件测试的目标就是验证产品开发的一致性和确认产品满足客户的需求，与之对应的任何产品质量特性都应追溯到用户需求。测试人员始终要站在用户的角度思考、分析产品的特性，例如，试问下列问题：

- 这个新功能对客户来说它的价值是什么？
- 客户会如何使用这个新功能？
- 客户在使用这个功能时，会进行什么样的操作？
- 按目前的设计，用户觉得方便、舒服吗？

如果发现缺陷，则判断软件缺陷对用户的影响程度，系统中最严重的缺陷是那些导致程序无法满足用户需求的缺陷。软件测试的目的就是揭示软件中所存在的错误、低性能、不一致等各种影响客户满意度的问题，一旦修正这些问题就能更好地满足用户需求和期望。

2. 尽早和不断地测试

软件项目一启动，软件测试也就开始了，也就是说在项目启动的第一天，测试人员就参与了项目的各种活动，并开展了对应的测试活动。测试工作进行得越早，越有利于提高软件的质量，这也是缺陷预防在测试活动中的一种体现。在代码完成之前，可以进行各种静态测试，从而主导或积极参与需求文档、产品规格说明书等的评审，并且充分做好测试的前期工作，包括计划测试、设计测试用例、开发测试脚本和准备测试环境。

从静态测试到动态测试，软件测试贯穿着整个软件开发生命周期。也只有这样，软件测试才能尽早发现和预防错误，降低软件开发成本，更好地保证软件质量。

所以，测试人员应将“尽早测试和不断地测试”作为座右铭。

3. 质量第一

测试的基本原则就是保证软件产品质量足够好，这要求进行充分的测试。充分的测试不仅需要有效的测试方法和良好的测试技术，而且需要时间来完成已计划的测试，做足够的测试，包括 Ad-hoc 测试，直至不能发现新的缺陷。这样的测试操作，正是基于“质量第一”的思想。

当时间和质量冲突时，时间要服从质量，测试人员应考虑如何获得更多的时间和资源

来保证质量。虽然有市场压力，但仍需要在效率和质量之间取得平衡，即使产品的开发要服从市场的策略和市场的需求，质量依旧是首要的。短时间内，一两个版本没有达到质量要求，可能不会给公司造成巨大损失，反而可能因为时间的优势为公司获得市场上更多的份额。但从长远看，质量还是决定企业生存的最关键因素。

质量第一的理念还表现在企业文化中，如“第一次就把事情做对”，在软件测试工作中，如何做到第一次就做好测试计划，第一次就设计出完美的测试用例等，是测试人员每时每刻都要思考的问题。

4. 有据可依

事先定义好产品的质量特性指标，测试时才能有据可依。有了具体的指标要求，才能依据测试的结果对产品的质量进行正确的分析和评估，例如，在进行性能测试前，产品规格说明书就已经清楚定义了各项性能指标。同样，测试用例应确定预期输出结果，如果无法确定所期望的测试结果，则无法进行正确的校验。

所有特定的质量要求及其指标，都应该在产品的需求文档、设计规格说明书中明确定义下来，这样才能使软件产品具有可测试性，重要的是实现软件时应充分考虑或依据这些要求，在软件开发过程中设法满足这些要求，以构筑高质量的产品。

5. 测试计划是过程，不仅是文档

虽然要通过文档来描述软件测试计划，最后还应归档，但计划是一个过程，由于对测试范围的掌控越来越准确，这个过程会对测试策略、资源、进度和执行等进行不断地调整。软件测试计划是做好软件测试工作的前提，但在一开始很难将所有的测试点、测试风险等都了解清楚，在与开发人员进行了足够的沟通，并对产品进行 Ad-hoc 测试后，情况会越来越清楚，从而可以进一步细化、丰富测试计划。其次，软件产品需求也会发生变化，开发进程也难以和事先预估的完全吻合，这样测试计划的修改也是必然的和必要的。在测试过程早期，应制定良好的、切实可行的测试计划，并在后期测试活动中严格执行，同时，要根据需要对测试计划进行调整以适应实际测试的状态变化。

6. 测试用例是设计出来的，不是写出来的

测试用例的目标就是帮助我们更快地、更多地发现缺陷，所以可以这样说：

- 测试用例发现缺陷的可能性越高，则说明测试用例越好。
- 一个成功的测试用例是能通过它发现所存在的缺陷。
- 测试用例帮助测试人员发现那些不容易发现的缺陷。

要确保测试用例的质量，就需要思考，需要画出工作流图、数据流图，全面理解产品

的特性及其实现的机制，从而可以针对产品设计与实现的薄弱点、边界点和逻辑路径等，设计出测试用例。而且，要选择适当的方法，以正确的逻辑思维去设计测试用例，从而提高测试效率。

7. 穷举测试是不可能的

一个大小适度的程序，其路径排列的数量也非常大，测试很难覆盖所有的路径。现在程序规模越来越大，在测试中不可能运行覆盖路径的每一种组合。如果要覆盖路径，其代价非常大，在有限的时间和资源下进行完全测试，找出软件所有的错误和缺陷是不可能的。对多数应用程序是没必要的，也是根本不可能的。因此，测试总是存在风险的，测试覆盖率是不可能达到100%。在测试用例的设计中，应设法确定所有的分支和边界条件，努力找出各种操作路径或逻辑路径，尽量使测试覆盖率达到很高的水平，如99%、99.5%、99.9%等。

8. 发现缺陷更多的地方，其风险更大

一般来说，一段程序中已发现的错误数越多，意味着这段程序的质量越不好。错误集中发生的现象，可能和程序员的编程水平和习惯有很大的关系，也许是写代码的程序员不够有经验，当时的情绪不够好或不在状态等。在同样的测试效率和测试能力的条件下，发现的缺陷越多，漏掉的缺陷就越多。当测试效率是90%时，如果某个模块发现10个缺陷，则只漏掉一个缺陷；而如果某个模块发现100个缺陷，则会漏掉10个缺陷。所以，对发现错误较多的程序段，应进行更深入的测试。

9. 更客观、更有效的第三方测试

程序员应避免测试自己的程序，为达到最佳的效果，应由独立的测试小组和第三方来完成测试。测试在一定程度上带有“挑剔性”，心理状态会影响对自己程序的测试。同时，对于需求规格说明的理解错误也很难由程序员本人进行测试时发现。在实际测试工作中，单元测试虽然由开发人员自己完成，也建议开发人员相互配对，为对方进行单元测试。

10. 重视回归测试

只要有缺陷修正、代码修改，就有回归测试，回归测试伴随着测试执行的过程。回归测试做好了，测试的成功才有保证。为修改之后的程序进行回归测试时，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大量新错误。回归测试的关联性也应引起充分的注意，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。

12.2 软件测试的多维空间

软件测试的空间可以按照图 12-1 来描述，可以看出这是一个包括了测试目标、测试方法和测试过程的三维空间。但图 12-1 还不能涵盖所有的测试内容，例如测试流程、自动化测试、测试团队建设等尚未包含在内，即没有构成一个完整的测试空间。如第 11 章的图 11-1 所显示的测试管理层次，包括测试团队、测试过程、测试方法、项目管理和测试的执行。所以，从这个意义上说，测试空间是一个多维的空间，除了质量特性的验证、测试方法和测试层次之外，还包括测试团队、测试流程、自动化测试等共享空间。

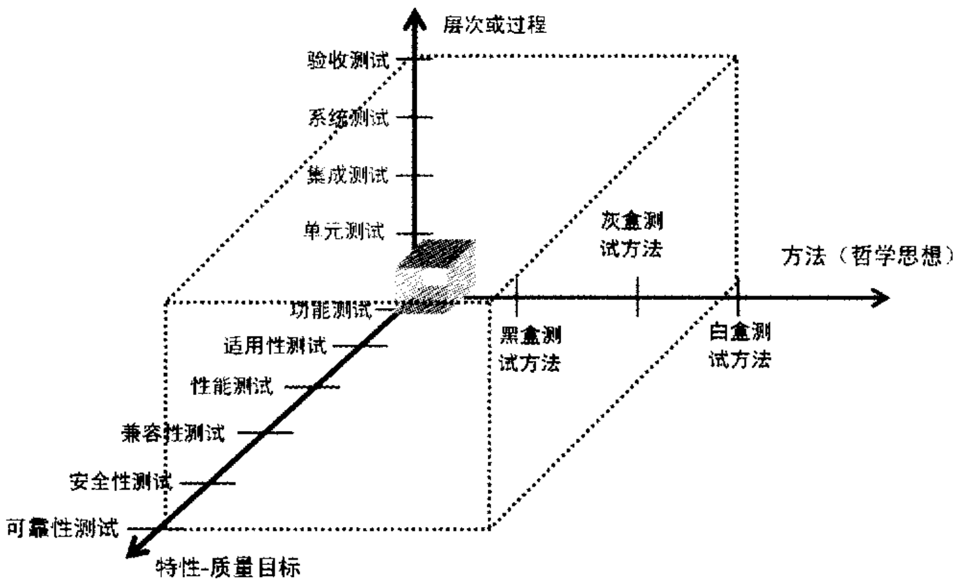


图 12-1 软件测试的三维空间

1. 测试目标——质量特性的验证

（1）正确性测试（Correctness testing）或功能性测试是基于产品功能规格说明书，并从用户角度针对产品特定的功能和特性所进行的验证活动，用以确认每个功能是否得到完整的实现，用户能否正常使用这些功能。功能测试一般要在集成测试完成后进行，而且是针对应用系统，在实际运行环境下进行的测试。

（2）性能测试（Performance testing）：它测试在一定条件下，系统的行为表现是否在设计的性能指标范围内。如测试网站在并发用户数为 10、100、1000、10000 等的情况下，页面的响应时间是否在 3 秒或 5 秒内，响应时间最长是否不超过 15 秒或 30 秒。性能测试不同于负载测试（Stress/load testing），性能测试是在定义的各种条件下去衡量系统

有关的性能指标，而负载测试只测试在一些极端条件下，系统还能否正常工作，或者加载到系统崩溃，找出系统性能的瓶颈，所以负载测试也可以和性能测试结合起来做。

（3）可靠性测试（Reliability testing）：它会评估软件在运行时的可靠性，即通过测试确认平均无故障时间（MTTF, Mean-Time-To-Failure）或最初平均寿命，即故障发生前的平均工作时间（MTTFF, Mean-Time -TO-First-Failure）。可靠性测试强调随机输入，并通过模拟系统实现，很难通过实际系统的运行来实现。可靠性测试，一般伴随着强壮性测试（Robustness/strong testing）。

（4）安全性测试（Safety or Security testing）：它将测试系统在应付非授权的内部/外部访问、非法侵入或故意损坏等情况时的系统防护能力，以检验系统是否有能力使可能存在的来自于内/外部的伤害或损害的风险被限制在可接受的范围内。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，因为安全性还涉及数据加密、保密、存取权限等方面的要求。

（5）容错性测试（Tolerance testing）：它用于检查软件在异常条件下自身是否具有防护性的措施或者某种灾难性恢复的手段。如当系统出错时，能否在指定的时间间隔内修正错误并重新启动。容错性测试可看作是由系统异常处理测试和恢复测试组成的。

（6）恢复测试（Recovery testing）是在系统崩溃、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，包括确定软件系统的平均修复时间（MTTR，Mean Time to Repair）。

（7）兼容性测试（Compatibility testing），它测试在各种硬件/软件/操作系统/网络环境下的软件表现，包括硬件接口、软件新旧版本的兼容性、已存在数据的兼容能力。

2. 测试方法——哲学的思考

测试的方法和技术，经过多年的发展，已经相当成熟。方法比较多，如白盒测试方法（White-box test）、灰盒测试方法（Gray-box test）和黑盒测试方法（Black-box test），它们是一种哲学思想在软件测试中的体现和延伸。从哲学观点来看，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。如果我们对被测的对象/世界（软件）的认知很少，可以不用了解其内部结构，完全只关注外部的变化，如外部的输入、外部作用或被测对象所处的条件，以及被测对象的输出结果，就可以完成测试，这就是黑盒测试方法。若对被测对象的认知越来越多，就可以采用灰盒测试方法；当我们完全认知被测的对象时，则可以用白盒测试方法。

3. 测试阶段——生命周期的显现

随着软件开发生命周期中各项活动的展开（即软件开发进程的不断推进），测试与之对应，也划分了不同的测试阶段，包括单元测试、集成测试、功能测试、系统测试和验收测试等。

12.3 软件测试方法的应用之道

软件测试的众多方法是辩证统一的，它们相互依赖、相互对立又相互补充，任何一种测试方法都有其优势，并可以在特定的测试领域发挥其优势。同时，任何一种测试方法都不能满足所有测试的需求，它们在某些场合存在一定的局限性和不足。这种测试的辩证统一，从下面这些相对应的测试方法中就得到了很好的印证。

- 白盒测试方法和黑盒测试方法。
- 静态测试和动态测试。
- 手工测试和自动化测试。
- 有计划测试和随机（ad-hoc）测试。
- 新功能测试和回归测试。

12.3.1 白盒测试方法和黑盒测试方法

黑盒测试方法，不考虑程序的内部结构和内部特性，而是从用户观点出发，针对程序接口和用户界面进行测试。它根据产品应该实现的实际功能和已经定义好的产品规格，来验证产品应该具有的功能是否被实现，是否能满足用户的要求。所以，黑盒测试方法主要测试整体系统的行为，它实施的是从头到尾（端到端，end-to-end）的数据流测试和功能逻辑测试。黑盒测试方法更适合系统的功能测试、易用性测试，也适合和用户共同进行验收测试、确认测试。虽然在单元测试和集成测试中也会应用黑盒测试方法，但相对来说，其局限性较大，更宜采用白盒测试方法。

白盒测试方法清楚产品的内部工作过程，可以对程序的每一行语句、每一个条件或分支进行测试，针对性很强，而且可以清楚已测试的覆盖程度。如果时间足够多，可以保证所有的语句和条件得到测试，测试的覆盖率高。如果要使测试覆盖率达到很高的水平，单元测试的时间往往超过编程的时间。

白盒测试方法可以在更早的时候进行，只要有一段程序被完成就可以进行，而黑盒测

试方法必须等到一个单元全部完成或者等到整个系统全部完成后才能进行。

从测试效率看，也不能说白盒测试方法一定高于黑盒测试方法，这取决于测试的对象和目标。对于单元测试、部分的功能测试来说，白盒测试方法的效率高，但对于端到端的数据流测试来说，输入数据驱动直至获得输出数据，需要经过一系列复杂的、很长的逻辑路径，黑盒测试方法此时就显得轻松得多，效率也高得多，因为在这路径上任何地方有问题，都会被发现出来，这也是白盒测试方法所不能比的。

其次，黑盒测试方法在编程能力和技术上相对要求低，而白盒测试方法要求测试人员具备良好的编程经验。黑盒测试方法，要求更高的综合分析能力、理解能力和反向思维能力。

最后，白盒测试方法也有其局限性，它能在代码上保证程序没有错误，但并不能保证软件没有缺陷。仅从代码的语句、条件和路径分析，不容易判断软件的行为是否完全符合客户的需求，也不容易发现功能上的适用性缺陷，这时候，黑盒测试方法反而有更大的用武之地。

在实际测试中，不需要强调测试方法是白盒的还是黑盒的，而是强调方法的有效性。在功能测试、系统测试中，更多时候是将两种方法有机地结合起来。

12.3.2 静态测试和动态测试

静态测试通过对软件的程序源代码和各类阶段性文档的成果采用走查、同行评审、会审等方法来查找其中的错误，不需要运行程序，所以相对于动态测试，可以更早地进行。

静态分析的查错和分析功能是其他方法所不能替代的。静态分析能发现文档中的问题（也只能通过静态测试实现），如通过软件评审方法可以发现需求分析、软件设计等问题，而且能有效地检查代码是否具有可读性、可维护性，是否遵守编程规范，包括代码风格、变量/对象/类的命名、注释行等。静态测试已被当做一种主要的代码校验自动化方法。

动态测试通过观察程序运行时所表现出来的状态、行为等发现软件缺陷，包括在程序运行时，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或进行跟踪对比，以发现程序所表现的行为与设计规格或客户需求不一致的问题。

动态测试是一种经常运用的测试方法，无论在单元测试、集成测试中，还是在系统测试、验收测试中，都是一种有效的测试方法。但动态测试不能发现文档问题，必须等待程序代码完成后进行，所以它发现问题的时间相对来说迟得多，而且一旦发现问题，必须重新设计、重新编码，必然增大不良质量的成本（劣质成本，COPQ）。

测试由静态测试和动态测试组成，两者相互补充，静态测试先行，动态测试紧随其后。静态测试蕴含着更多的理性思维，它从客户角度出发，以严格的标准来审查需求的定义和实施，而动态测试则体现更多的感性，对产品特性进行一步步的验证。

12.3.3 手工测试和自动化测试

手工测试是指通过测试人员手工操作来完成软件测试工作的方法，而自动化测试是通过计算机运行测试工具和测试脚本自动完成软件测试工作的方法。相对于手工测试，自动化测试具有很多优点，如：

- 能高速度运行，可 24 小时不间断地运行以缩短测试周期。
- 可以多次重复运行相同的测试，减少测试的单调性。
- 可以模拟成千上万的并发用户同时发出请求和操作。
- 测试具有准确性和客观性，能真实反映测试结果。
- 可提高测试人员对工作的兴趣和激情。

所以在测试工作中，我们尽力实现测试自动化，扩大自动化测试的覆盖范围。但是自动化测试前期投入大，对被测对象要求高，也存在其他的局限性。

手工测试在功能测试中的优势还是比较大的，测试人员具有直觉判断能力、灵活性和应变性，而测试工具没有想象力和灵活性。手工优势表现在：

- 可以灵活地随时引进一些操作，在可操作的范围内想到什么就做什么。
- 对于复杂的逻辑合理性、界面友好性，测试人员能迅速地做出判断，但测试工具却无能为力。
- 自动化测试只能按事先设定的测试用例执行测试，而手工测试具有创造性，在执行某个测试用例时，可以举一反三，设想出其他的一些测试用例，对原来的测试用例进行扩展，以覆盖一些特殊的场景或边界情况。

手工测试和自动化测试，两者相辅相成，又各自有相应的优势领域。在多数情况下，测试人员可采用手工测试和自动化测试相结合的方式，以最有效的方法来完成测试任务。

- 工具本身并没有想象力和灵活性，根据业界统计的结果，自动测试只能发现 15%~30%的缺陷，而手工测试可以发现 70%~85%的缺陷。所以自动化测试有其局限性，不适合软件的新功能测试，但特别适合回归测试，能及时发现原来运行正常的特性所存在的新问题。

- 在进行系统验收测试、适用性测试、涉及物理交互性测试和复杂逻辑测试时，也很难通过自动化测试来实现，多数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。负载测试、性能测试和可靠性测试等，不得不借助测试工具来完成。
- 当界面、需求的变化比较频繁时，或者软件的开发周期很短又或者做的是一次性软件开发项目（而不是做软件产品）时，自动化测试吃力不讨好，投入大而产出小。见下面的示例说明。
- 有些测试工具只能运行在 Windows 平台上，不能运行在 Mac/Unix 等平台上。

示例

功能测试自动化主要应用在回归测试中。当产品的界面（UI）和功能的变化较大，或者自动化的脚本（Script）维护成本较大时，投入和产出往往成为我们最关心的问题，我们会考虑在功能测试中实现测试自动化究竟是否合算？

举个例子：假如一个功能测试用例，手工运行需要 10 分钟，而为此开发测试用例脚本需要 4 个小时，即 240 分钟，那么意味着这个测试脚本要被运行 24 次后才能收回脚本开发的投入。如果在加上维护测试脚本版本的工作量（10%），需要运行更多次（假定 40 次），才能收回成本。如果在产品的一个版本中要进行 2~3 轮测试，这样需要发布近 15 个版本才能收回成本。纯粹从投入/产出上来算，自动化测试在功能测试上并没有多大优势。对于一个足够持续的产品开发链，TA 所带来的经济效益和可靠性等方面，还是显而易见的。

12.3.4 有计划测试和随机测试

在测试执行前，一般都需要进行测试的策划和计划，分析测试的重点和范围，精心设计测试用例，以做好测试执行前的准备。通过测试计划和测试用例进行的测试是有计划的测试，而随机的测试（ad-hoc test）不借助测试用例，完全凭感觉和猜测进行自由、灵活的测试。

有计划的测试，目标性强、易管理、覆盖面比较全，是测试过程的主旋律。但由于用户使用软件的情景很多，千变万化，事先设计好的测试用例很难覆盖各种情况，特别是一些边界和特殊的操作。根据经验和历史数据统计，对于大型系统，其软件测试用例的覆盖率一般在 90%到 95%之间。所以，必须借助一些自由的、随机的测试，充分发挥测试人员最大的灵动性、创造性，通过各种猜测和试探，去发现一些隐藏相对比较深或偏僻的软件

缺陷。随机（Ad-hoc）测试，也可作为一种重要的测试辅助手段，以帮助测试人员尽早地熟悉产品，发现测试用例的不足，添加或改进测试用例。

在实际测试过程中，不仅要严格执行已计划好的测试，同时还应留出一些时间和资源来进行 Ad-hoc 测试，而且 Ad-hoc 测试要尽早安排，这样做能一举两得，既能更好地理解产品特性，又能发现缺陷、完善测试用例。Ad-hoc 测试也可以穿插在有计划测试的过程中。像微软采用的 bug bash（缺陷大扫除），就可以看作是 Ad-hoc 测试的特殊例子。

知识点

Bug bash 就是为了发现那些隐藏很深、逃过了计划测试的缺陷。在 bug bash 活动中，可能会专门划出一个时间段（通常 1～3 天），让所有参与项目的人员集中全部精力，运用各方面的知识，尽全部智慧来搜寻软件产品的缺陷。这是一个非常有意思的活动，组织这样的测试活动的一些经验是：

- 尽管这是一个测试活动，但参与者并不仅仅局限于测试人员，还包括项目经理、开发人员和高层管理人员，如同全民动员起来进行测试，集思广益。
- 要鼓励各部门进行领域交叉搜索，因为新的思路和视角通常有助于发现更多的 Bug。
- 为了调动积极性，增强趣味性，可以适当引入竞争机制。例如，当活动结束时评出“发现最多缺陷”、“发现最严重缺陷”的个人，给以物质和精神奖励。
- 可以分专题展开缺陷大扫除活动，比如安全性、用户界面可用性、国际化和本地化等。

12.3.5 新功能测试和回归测试

因为修正缺陷时会发生代码变动，所以回归测试就少不了。在一般测试中，首先会进行新功能的验证，新功能出现错误的可能性要大得多。一旦发现缺陷后，就需要修正缺陷并构造一个新的软件包，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，还要保证以前所有运行正常的功能依旧保持正常，因为修正缺陷可能会影响其他的安全区域。回归测试，是为检验修正缺陷是否会引起原有正常功能出现新的缺陷（回归缺陷，regression bug）而进行的测试。不论是开发全新的软件（第 1 版本），还是不断升级已有的产品，回归测试都是不可少的。而且，对于后继版本的软件开发来说，回归测试所占的比重越来越大。所以，一个完整的测试可以看作是新改动的功能特性的验证测试和回归测试的组合。

在软件产品实现过程中，新功能的实现固然重要，可以增强产品的亮点和竞争力，增加市场份额，但是不能正常工作的已有功能所引起的客户抱怨会大得多，因为客户已经习惯使用已有功能了。而对于新功能，客户还没开始使用，没尝到甜头，所以不会那么敏感，甚至还不知道有这个新功能。曾经有一个调查显示，95%的用户喜欢功能稳定，不要经常变化的软件，只有5%的用户喜新厌旧。如果在软件发布时，发现某个新功能的问题较多，可以在客户知道前去掉这个功能，这对现有软件产品或软件服务的影响较小。所以，从这个意义上说，回归测试显得更为重要。

12.3.6 测试方法综合应用的总结

测试工作的质量，首先取决于先进的质量理念和文化，应坚持质量第一的原则，其次，取决于对各种测试方法辩证统一地理解和正确、有效地运用。在前面5节中，辩证地探讨了软件测试方法的应用之道，我们若一分为二地看问题，就能认识到各种方法都是相对的，总是优点和缺点并存。而测试方法的应用之道和之本，就是发挥每种方法的长处，避其短处，统一地运用多种方法，以取得最好的测试效果。表12-1对各种测试方法进行了总结和比较，形成了有效的方法组合，能更好地满足测试的需求。

表12-1 测试方法的有效组合

组 合	静态测试	动态测试	自动化测试
白盒测试	①静态白盒测试方法 走查、复审、评审程序源代码、数据字典、系统设计文档、环境设置、软件配置项等	②动态白盒测试方法 通过驱动程序、桩程序来调用、驱动程序，如进行单元测试、集成测试和部分性能、可靠性、恢复性测试等	白盒测试工具： Logiscope、C++ Test、Jtest、DevPartner、Purify、TrueCoverage 等
黑盒测试	③静态黑盒测试方法 文档测试，特别是针对产品需求文档、用户手册、帮助文件等的审查	④动态黑盒测试方法 通过数据输入，以及运行程序来检验输出结果，如功能测试、验收测试和一些性能、兼容性、安全性测试等	黑盒测试工具： Rational 公司的 Robot GUI，Compuware 的 QACenter 和 MI 的 WinRunner 等
自动化测试	静态测试工具： Logiscope、CheckMate、QA C++、QStudio Java、TrueJ 和语言编译器等	动态测试工具： DevPartner、Purify、Robot GUI、QACenter、WinRunner、Load Runner、WebKing	

续表

组 合	静态测试	动态测试	自动化测试
手工测试	走查、评审、会审	单元测试、集成测试、功能测试、安装测试、性能测试、可靠性测试等	测试用例和测试脚本依然是自动化测试中的关键内容之一，但这来自于手工，并依赖手工测试验证自动化测试结果
回归测试	复审、变更审查	所有测试领域	最好的结合区域：自动化回归测试
注：①②③④构成了测试的4种基本方法，基本覆盖了测试领域			

12.3.7 测试方法的有效性和风险性

软件测试方法蕴含着有效性和风险性的矛盾，而测试管理就是通过处理这种矛盾以获得最优平衡。软件测试的有效性和风险性的矛盾总是存在的，而且贯穿整个测试周期。有效性和风险性首先体现在一种测试理念的较量上。虽然，当我们问一个测试工程师，测试是什么时，她/他可能会毫不犹豫地说，测试是发现缺陷。但当她/他管理一个测试项目或进行测试时，总是想证明所有功能是正常的，这会大大降低测试的效率。有时，在她/他的心里会进行不断的斗争，结果就可能去冒一些风险，创新和采用一些手段去提高测试效率，但更多的时候还是处在保守状态，测试时只是试图验证所有的功能，牺牲了测试效率。

在制定测试策略、测试计划时，有效性和风险性的矛盾可能更突出，表现在测试的质量标准、测试范围和测试重点上。"如何减小测试范围，抓住测试重点"是需要技巧和经验的，技巧和经验可提高测试的有效性，但有时往往也会忽视一些风险。质量标准（产品特性的具体要求）也具有策略性，是市场需求和工程质量目标斗争的结果，极具平衡的艺术。

设计测试用例时，我们也常常为测试用例的"颗粒度"伤神。如果把测试用例设计得很细，照顾到每一个数据输入、每一个条件、每一个环境、每一个路径，那么测试用例的数量将是巨大的，虽然风险将很低，但是测试效率也会相应降低，并且测试执行时没有思考的空间，可能会使测试执行人员变得呆板（除非全部测试自动化），并且使他们不再有创造力和思考。测试用例的设计如果很粗，测试效率则可能比较高，测试人员有一个发挥的空间，他会觉得测试更有趣，但这依赖于个人的责任感和能力，风险便会增大很多。

所以说，测试用例并不是越多越好，而需根据测试目标、质量需求和软件产品本身的

特点来决定。运用有效的测试用例设计方法，尽量使用较少的测试用例去覆盖合理的测试需求。

12.4 软件测试的最佳实践

软件测试的最佳实践来自于测试工作，来自于项目过程中不断地思考和创新，来自于项目结束之后的经验和教训的总结。例如，在一个项目结束后，首先就会想什么做对了，什么没有做好，如表 12-2 所示。

表 12-2 项目教训和成功的总结

什么做对了？——成功的经验		什么没有做好？——有待提高的地方	
新定义的流程	每个人责任心、全局控制能力等增强了	评审签发的过程	时间耽误了，次序乱，不是所有人都能及时得到通知
I18N 测试	在设计、编程阶段进行了很好的 I18N 评审	功能特性分析不够	对产品架构理解不足，和开发人员技术交流不够流畅，个别特性定义比较迟
模块责任人和测试组长	相互之间知识传递做得好，而且互为评审也比较频繁	沟通	跨部门的沟通、跨产品线的沟通不够。事实上，在项目上这些部门或产品线之间存在着较强的关系
PRD/MRD/Spec 的理解	组织了很多在线评审会议，使评审效果达到预期效果	测试环境的依赖性	该项目的测试环境依赖第 3 方产品较多，但事先没有弄清楚
测试环境	建立了一套备份环境和相应的流程，环境更稳定了	安装测试	功能特性的改变影响安装，但被少数责任人忽视了
里程碑	里程碑预警机制实施很好，及时到达所有的里程碑	任务变动	有些任务变更没有经过“变更控制”流程

续表

什么做对了？——成功的经验		什么没有做好？——有待提高的地方	
回归测试	和开发人员共同审阅修改的代码，做代码比较、交叉测试等，提高了回归测试的准确性（效率和质量）		
管理团队	更具体地了解项目，对问题能刨根问底，给了项目更多的支持		

12.4.1 测试计划的最佳实践

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深刻的理解。对用户需求、产品特性和技术实现的理解，是制定测试计划的基础。除此之外，还会在测试的方法运用和策略的制定等各个方面，积累很好的经验。概括起来，在制定测试计划过程中，尝试采用下列最佳的实践。

（1）在确定测试项目的任务之前，应清楚测试的范围和测试的目标，如检查每一个测试点须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定，特别是在制定测试计划的早期。合适的相关人员包括相关的产品经理、开发人员、技术支持人员、系统部署人员和市场销售人员等。

（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，测试范围能分解的应尽量分解，针对每个测试任务仔细分析，做到位，尽量做到客观、准确、留有余地。把握好留有余地的度，一般定在10%~15%。

（4）制定测试项目的输入、输出和质量标准，并和有关方面达成一致。例如，系统的性能指标必须清晰地在PRD/MRD中定义出来。

（5）建立变化处理的流程规则，识别在整个测试阶段中哪些是内在的、不可避免的变化因素，以及如何进行控制。

测试计划强调测试计划和执行的原则，在测试计划中描述测试过程所需的测试用例、

执行顺序和操作规则等，这些将成为此后测试活动的指导框架和约束。在测试计划中的错误与欠缺，会使测试在设计和执行过程中遇到更大的问题，包括不正确的设计思路、测试资源不足、测试环境不可靠、较低的测试执行效率等，某些测试问题还难以补救。所以，测试计划的评审需要得到足够的评审，计划中的每一条款都需要得到实实在在的确认。

12.4.2 测试用例设计中的最佳实践

软件测试用例是为有效发现软件缺陷而编写的包含测试目的、测试步骤、期望测试结果的特定集合。正确认识和设计软件测试用例可以提高软件测试的有效性，便于测试质量的度量，增强测试过程的可管理性。在设计测试用例时，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. 责任到人

人是要考虑的第一因素，也是决定的因素。测试用例的设计人员，需要透彻地理解产品的特性、系统架构、功能规格说明书、用户场景以及具体的实现技术等。所以，不是每个测试人员都能胜任测试用例的设计工作，一般要求具有较强能力的测试人员来完成。资深的测试人员往往更适合测试用例的设计工作。

仅仅从人员的能力和水平来讨论测试用例设计的要求还是不够的，还应考虑人员是否熟悉被测试的产品模块和相应的知识领域，即术要专攻。如果某个测试人员较长时间一直工作于某个领域或模块，无论是经验或悟性都会有助于对产品特性的理解，有助于和开发人员进行深入的沟通，所以测试用例的设计、更新和维护要做到责任到人。

2. 持续改进测试用例

测试用例一般要求在代码完成之前就绪，包括评审和签发，以便此后可以开始全面执行测试，快速推进测试执行的过程。测试用例的设计也不是一次性的工作，应该随着测试的不断深入，持续改进测试用例，不断提高测试用例的覆盖率。我们知道，“变化是永恒的”，客户的需求发生变化，产品规格说明书则相应地被更新，系统设计也可能会被调整，程序实现的方法也会被优化、细化。软件测试用例设计，必然要做出相应的变化，增加新的测试用例或修改已有的测试用例，删除一些不再适用的测试用例。

随着产品版本的不断升级，软件测试用例也需要得到及时维护，有时还需要重构——对测试用例的结构进行调整，包括用例模块的合并和分解，提高测试用例的“新鲜度”，确保每一个测试用例都是有效的，所有无效的测试用例被置于“无效（invalid）”或“不可见（invisible）”状态。软件测试用例的维护，要做到“与时俱进”。

3. 测试用例设计不能局限于输入数据

对于软件功能测试和性能测试来说，确定测试的输入数据很重要，它决定了测试的有效性和测试的效率。但是，在测试用例中输入数据的确定方法只是测试用例设计方法中的一个子集，除了确定测试输入数据之外，测试用例的设计还需要综合考虑被测软件的功能特性、测试目标、风险等，以及如何根据测试需求、设计规格说明等确定测试用例的结构和设计策略、设计思路等。

4. 尽量避免含糊的、冗长的或复杂的测试用例

含糊的测试用例，如“输入正确的用户和密码，所有程序工作正常”，会给测试过程带来困难，甚至会导致测试过程中遗漏问题，影响测试的结果。避免使用冗长和复杂的测试用例，可以保证验证结果的唯一性，从而便于软件质量的跟踪和管理。当测试用例包含很多不同类型的输入或输出，或者测试过程的逻辑复杂而不连续时，需要对测试用例进行合理的分解，从而保证测试用例的准确性。

在测试过程中，测试用例的状态是唯一的，其执行的结果应是下列 3 种状态之一：

- 通过（pass）。
- 未通过（failed）。
- 未进行测试（not done or blocked）。

如果测试未通过，测试用例就会关联一个或几个软件缺陷。如未进行测试，则需要说明原因，指出是测试用例的条件不具备，缺乏测试环境，还是测试用例目前已不适用等。因此，清晰的测试用例不会让测试人员在测试过程中出现模棱两可的情况，一个具体的测试用例不会出现 “部分通过，部分未通过”这样的结果。如果通过某个测试用例的描述不能找到软件中的缺陷，但软件实际存在与这个测试用例相关的错误，那么这样的测试用例是不合格的，它会给测试人员和开发人员的判断带来困难，同时也不利于测试过程的跟踪。这样的测试用例需要修改或细化成若干个测试用例，使测试用例中所期望的结果是特定和清楚的。

5. 尽量将具有相类似功能的测试用例抽象并归类

我们一直强调不能穷举所有的测试路径和输入数据，因此，相类似的测试用例的抽象过程显得尤为重要，一个好的测试用例应该能代表一组同类的数据或相似的数据处理逻辑过程。一个测试用例可以代表一组输入数据的处理，但其操作步骤是相同的，所期望的结果也是一致的。这样可以降低测试用例的数量，有利于测试用例的管理，并让测试注意力集中在边界条件上，以提高发现缺陷的效率。

12.4.3 测试自动化中的最佳实践

测试自动化从启动开始到其实施、发展，都需要建立一套行之有效的办法和流程，以便于真正将测试自动化完全转化为生产力。在测试自动化的管理或实施过程中不断摸索，有失败的教训，也有享受自动化测试成果的快乐。但从过去的经验看，在不同的领域中失败和成功是不一样的，在负载测试或性能测试中，测试自动化容易成功；而在功能测试中，失败可能比成功多。

1. 建立正确的观念、设定现实的目标和找准切入点

观念决定目标的设定，目标决定结果。开展测试自动化工作，一定要清楚自动化测试和手工测试各自的优势和局限性，在不同的测试阶段和测试范围，两者的效果不一样。虽然软件测试自动化具有很多优点，但只是对手工测试的一种补充，绝不能代替手工测试。

工具本身并没有想象力和灵活性，自动测试只能发现15%～30%的缺陷，而手工测试可以发现70%～85%的缺陷。在进行功能测试时，TA工具准确的含义是回归测试工具，因为工具不能发现更多的新问题，但可以保证已经测试过的内容的准确性和客观性。

在正确的观念下，还需要建立现实的软件测试自动化目标。如果对测试自动化操之过急，或者认为测试工具可以发现大量新的缺陷，就会建立不现实的目标，最终可能会以失败而告终。人们往往容易对测试自动化存在过于乐观的态度和过高的期望，期望可以通过测试自动化解决目前测试中所有遇到的问题，同时，测试工具的软件厂商自然会强调有利的或成功的一面，但取得成功所需要的巨大投入和持之不懈的努力却只字不提。结果，最初寄予厚望的目标难以实现，然后会对测试自动化失望，一蹶不振。

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，可以从最基本的测试工作切入，如验证基本功能能否正常工作，验证新构建的软件包（Build）是否有严重的或致命的问题。也可以从一个适合自动化测试的模块（如API调用、邮件模板处理等）开始，先保证这个模块自动化测试的成功，再向其他模块推进。更好的切入点是周期长的全新产品、全新项目，测试和开发同步进行，有足够的时间进行测试脚本的开发，也可以更好地规范开发人员的行为。

2. 人才需求和有效的、充分的培训

有些软件公司舍得花几十万元去买测试工具软件，但不愿意在招聘有编程经验、素质高的测试人才上多投入。测试自动化，不是简简单单地使用测试工具，还需要完成测试脚本的开发，建立测试自动化框架和流程，这不仅要求测试人员熟悉产品的特性、领域知识和测试技术，而且要求测试人员具有很好的编程技术和创新的能力。

人员和培训是相辅相成的，有了良好的人员，如果没有有效的、充分的培训，测试人员可能会对测试工具的了解缺乏深度和广度，从而导致其使用效率很低，结果不理想。测试自动化的培训是一个长期的实践过程，不是通过一两次讲课就能达到效果的，而应提供具有实际项目背景的系列培训，从头开始，逐步深入，直至掌握主要的方法和技巧等。

在实际使用测试工具的过程中，测试工具的使用者可能还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，会严重影响测试工具的使用积极性。如果实施自动测试的脚本质量得不到保证，将直接影响到测试结果的正确性。但一般不会针对测试脚本进行大量的测试，所以脚本本身的质量更多地依赖于开发脚本的测试人员的经验和工作态度。

3. 把测试脚本开发纳入整个软件开发体系

把测试开发纳入整个软件开发体系是必要的，而且这个开发体系应涵盖流程控制、资源关联、测试脚本的开发和管理等多个方面。

（1）在流程上，由于要开发脚本，测试自动化前期投入大，而在执行测试脚本时，工具的效率很高，这样要求软件开发前期的时间比以前分配的更长，而代码完成后的测试执行时间可以短些。自动化测试工作在项目启动的第一天就开始介入，和开发人员同步工作。

（2）白盒测试工具的一般在单元测试阶段使用，而单元测试在多数公司是由开发人员自己完成，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，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定义出来，如在项目各个里程碑所提交的文档中必须包含某些测试工具生成的报告——如 DevPartner 工具生成的测试覆盖率报告、Logiscope 生成的代码质量报告。

（3）通过自动测试工具测试的测试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也体现自动测试的结果，是比较好的方向。

（4）测试脚本的开发和产品程序的开发非常相近，也需要建立相应的流程和规范，包括脚本的编写风格、脚本注释、版本管理、检入/检出、变更控制、单元测试等。在编写脚本时，不能有 Hard code，需要积极创造条件构造数据驱动的脚本和结构化脚本，也可以建立脚本库函数供上层脚本调用，并通过关键词驱动将测试的逻辑层分离出来。为了使测试自动化的脚本能多次重复使用，测试用例和测试脚本要写入数据库，并进行动态管理。

（5）人力分配也不同，如果没有测试自动化，测试后期需要较多的人力资源。如果全面开展测试自动化，前期（脚本开发）需要更多的人力资源，而后期（运行测试工具）

需要较少的人力资源。

4. 软件程序开发和测试自动化不可分离

若系统不具有可测试性，再好的工具也无能为力，所以要求在设计、编程中实现可测试性。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，另一方面还要保证软件的可测试性，使之在功能的灵活性和可测性之间达到平衡，有时可测试性显得更为重要。

（1）软件测试自动化的测试脚本的维护量很大，而且软件产品自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可重复。例如，程序中对象名称的改变对脚本的正常运行影响较大，所以在编程时，要尽量保证对象名称不变。

（2）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具体问题的解决能力。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，加之技术环境的不断变化，所有这些对测试自动化应用的推广和深入，都会带来一定的影响。

（3）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，可能是测试工具本身不够全面或测试的预期结果设置不对。

5. 测试自动化依赖测试流程和测试用例

不管是手工测试还是自动化测试，关键是测试流程的建立和测试用例的设计，只有在良好测试用例的基础上开发测试脚本，测试脚本的质量才有保证，才能最终保证自动化测试的执行效果。为了更好地满足测试脚本开发的需求，可以将测试用例转化为决策表、矩阵图，帮助实现脚本的结构化、脚本的数据驱动方式。

另外一种做法，先不写测试用例，而是直接开发脚本，然后将脚本中的注释转化为测试用例。但就其本质来说，无论是注释还是思维，都是测试用例自然形成的过程，并建立在对产品特性理解的基础之上。

如果将测试自动化的开发和执行分开，成立独立的测试自动化团队（TA Team），即设立专业的TA工程师。由于TA工程师对产品特性的理解不够，而测试用例是由测试工程师来开发的，这时测试自动化便更依赖于测试用例。

为了更好地保证测试脚本开发的结构化、有效性和效率，不建议设置独立的测试自动化团队，而应将测试自动化溶于一般的测试过程中，每个测试工程师将需求文档的审阅、产品规格说明书的审阅、测试用例的设计、脚本的开发和运行等集于一身，这将是最理想的。

有时考虑团队建设的问题，要一分为二地对待。如果所有测试工程师都具有良好的编程经验，那么可能导致团队的稳定性欠佳，人力成本也相对较高。由于软件界面变化较大，特别是 Web 应用，所以功能测试更多需要的是人的直觉和经验，从而手工测试所占比重较大，这时技术特别好的工程师对黑盒测试不感兴趣，精力投入不够，工作反而做得不够好。

6. 降低测试自动化的投入、提高其产出

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，除了购买测试工具软件的费用（一般来说，测试工具商业软件还比较昂贵）和大量的人员培训费用之外，还需要很长时间去开发和维护测试脚本等，所以有必要关注和思考测试自动化的投入和产出，不断调整测试自动化的目标和策略，这样有助于提高测试自动化的效率，达到事半功倍的效果。

如何降低测试自动化成本，增加产出（测试自动化的效果/效益）？关键是如何提高脚本开发速度，增强脚本运行的稳定性和降低脚本维护的工作量。其主要方法有：

- 针对被测试软件的特点和团队能力，选择合适的、有效的测试工具。
- 有些模块易于手工测试，有些模块易于工具测试，选择那些适宜的自动化测试模块。
- 尽量将脚本写成数据驱动、关键词驱动的脚本，这一点很重要。
- 多录制脚本，然后结构化脚本。我们知道，不是所有的模块都可以变为数据驱动方式，这时就要抽象出脚本的结构，并进行有效的组合，包括分层，形成有效的层次性。
- 把测试和脚本开发合二为一，效率更明显。

7. 进行资源的合理调度

开发中的产品在达到一定程度的时候，就应该开始进行每日构造新版本和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，紧接着进行自动化的构建包验证测试（BVT）和回归测试。第二天上班时，测试结果就已经在测试工程师的邮箱内了，然后测试工程师开始分析测试结果、报告问题，接着再修改脚本、开发和调试新的脚本，最后提交自动化测试的任务。

这种理想的自动化测试运行模式依赖于一个良好的自动化框架（TA Framework）。在这个框架中，集成了脚本开发和执行所需要的各个组件，它构成了一个完整的自动化系统。

这个框架层次要清楚，要和缺陷跟踪系统、测试用例管理系统等集成，要让操作很方便，它包括通过 Web 方式提交任务、查看测试结果等。

8. 测试工具的选择

测试工具的选择也是直观重要的，不仅要考虑商业化工具，而且要更多地考虑选择开源测试工具。我们应选择好的工具，最流行的工具不一定适合自己，真正适合自己的工具才是最好的。如 Rational Robot 不一定是最好的，但它所具有的多机交互协作能力是其他工具没有的，适合协作软件的自动化测试。

根据客户端、Web 和服务器的不同特点可选择不同的测试工具，如果要用于 Web 的链接或具有复杂的逻辑而且变化快的 UI，则工具的录制功能要强、要稳定，以适应不同的平台（Windows、Linux、Mac OS）和浏览器（IE、ForeFox、NS...）。而服务器一般不存在 UI 界面，所以选择的工具要能支持多种不同的协议，对性能指标有较强的监控和记录功能。

12.4.4 测试执行中的最佳实践

测试执行似乎没有测试用例设计和测试自动化那样有技术、有深度，但正如前面所说，测试执行依然是测试的主要活动之一，它还要占用很多的测试资源，执行的周期也很长，所以测试执行的环节也不能忽视。在测试执行的过程中，若有更多的人参与，每个人都想有所创新，这样就容易获得更多的反馈和建议，积累更多的最佳实践，哪怕是一个很小的措施和改进。

在测试执行过程中，常常会碰到一些困难，也会有这样那样的疑惑，例如：

- 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和要达到的目标？
- 如何保证每个测试用例得到百分之百的执行？
- 如何保证所报告的软件缺陷正确、描述清楚，没有漏掉信息？
- 如何在验证 Bug 和回归测试、新功能测试与回归测试之间寻找平衡？
- 如何跟踪 Bug 处理的进度使严重的 Bug 及时得到解决？

解决这些问题的方法不是绝对的，需要根据当时的实际情况去处理，但还是有一定的规律可循，能找到相应的解决办法。首先，要实施全程跟踪测试过程，建立较为完善的过程度量和评审机制，并借助有效的测试管理系统。其次，可以采用下列的方法和措施。

1. 测试执行前

- 执行前，开一个动员会是必要的，如同打战，要鼓舞士气，更重要的是阐述测试策略，回答大家的问题，把测试任务、测试范围和所有的条条框框都交待清楚。
- 严格审查测试环境，包括硬件型号、网络拓扑结构、网络协议、防火墙或代理服务器的设置、服务器的设置、应用系统的版本，包括被测系统以前发布的各种版本和补丁包，以及相关或依赖的产品。
- 将要执行的所有测试用例进行分类，基于测试策略和历史数据的统计进行分析，包括测试策略和缺陷的关联关系，构造有效的测试套件（Test Suite），然后在此基础上建立要执行的测试任务，这样任务的分解就有助于进度和质量的有效控制，降低风险。

2. 测试执行过程中

- 由几个经验丰富的测试工程师进行抽查性质的测试或 Ad-hoc 测试，以验证某些风险区域的测试质量。
- 由于不同的人员在经验、思维方式上存在差异，交叉互换测试人员所测试的模块，可以发挥互补作用，提高测试覆盖率，更多地降低风险。如在“雅虎日历”第一轮测试中，张三测试“显示”模块、李四测试“添加、修改事件”模块；在第二轮测试中，可以让张三测试“添加、修改事件”模块，而让李四测试“显示”模块。
- 在测试时间被压缩后，首先要思考测试策略的优化，调整计划，提高测试效率，找出解决问题的办法。如果没有优化的空间，就要考虑测试需求的优先级，调整测试范围和测试需求，优先测试重要的功能点，而一些不常用的和级别低的测试需求可以转移到用户现场或者后期进行。
- 进行常规的缺陷审查，如 Daily Bg review、bug scrub meeting，包括 Bug 的严重性、Bug 的描述、Bug 修正的反应速度等，及时发现问题、纠正问题，使整个测试进程在控制轨道上发展。
- 对每个阶段的测试结果进行分析，保证阶段性的测试任务得到完整地执行并达到预定的目标。
- 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，使手工测试和自动化测试分离，以消除它们之间的影响。对于像性能测试、负载测试等占有比较多资源的测试，可以调节各人休息的时间表（每个测试人员依然每周工作 5 天），保证测试每周 7 天都能运行，充分利用测试机器和其他资源。

3. 与测试执行相关的其他内容

- 良好的沟通。不仅要和测试人员保持经常的沟通，还应要求和项目组的其他人员保持有效的沟通，如每周例会，可以及时发现测试中的问题或不正常的现象。
- 要确保每一个测试人员理解测试策略、测试目标。应对测试进程进行审查(Audit)，确保测试策略得到执行，可以通过一些奖励手段进行引导。测试经理、组长要勇于承担风险，使测试人员有发挥、想象的空间，但同时也要给予适当的压力，提高工作效率和责任心。
- 所有测试用例、测试套件、测试任务和测试执行结果，都通过测试管理系统进行管理，使测试执行的操作、过程记录在案，从而具有良好的可跟踪性、控制性和追溯性，容易控制好测试进度和质量。
- 缺陷的跟踪和管理一般由数据库系统来执行，这样做容易对缺陷进行跟踪、统计分析和进行趋势预测，还可以设定一些有效的规则和流程来配合测试执行，如通过系统自动发出邮件给相应的开发人员和测试人员，使任何缺陷都不被错过，并能得到及时处理。而且事先建立基于缺陷跟踪系统的缺陷报表、缺陷趋势曲线，对各模块、各测试人员、整体项目等进行实时跟踪。

12.4.5 测试团队建设中的最佳实践

正如在第 11 章谈到管理层次之时所说，测试团队是第一层，是测试的基础，在测试团队的建设和管理中，测试团队也能获得许多最佳实践。一些通用的、适合各种团队建设的最佳实践，自然也适合测试团队的建设，如经常和团队成员沟通，使之在团队目标、价值取向、团队文化上获得共识等。

测试团队和其他的团队也没有太大的区别，例如拓展户外活动、经常聚会都有助于增长团队的士气和凝聚力。在这里，可以强调一些对测试团队来说相对更为重要的最佳实践。

1. 团队理念

测试团队在自己平时的工作就要起表率作用，须做事认真，不管是报告还是邮件，每样送出去的文档或材料，都应清清楚楚、完整无缺，从而建立起其他人员对自己信任。测试团队应崇尚下列理念：

- 质量第一。
- 第一次就将事情做对。
- 不断思考，追求完美。

2. 构造扁平化组织结构

团队层次越多，工作效率越低，越容易形成官僚，而且大多数团队成员心理感受也不会很好，觉得上面有那么多的领导在管着她/他。如果将团队管理扁平化，那么测试工程师的积极性、责任感会得到加强。对于测试工作，积极性、责任感是非常重要的。有时我们开玩笑说，开发人员犯错误没事，有测试人员给挡着；测试人员不能犯错误，因为我们不能让用户或客户给挡着。虽然这种说法不对，因为不能让开发人员有依赖性，但或多或少的说明了测试工程师的责任感是非常重要的。

测试团队的构造通常会按照公司的不同产品进行划分和组织，以使测试人员对产品的质量或对服务的质量那种负责任的意识更高，只有这样测试才会有深度和广度。

3. 教育比培训更重要

不断通过事例来教育、感化团队成员，不管这事例是小事情还是大事情。更多的是通过一个迟发现的缺陷、一个漏掉的缺陷来帮助大家认识到缺陷的危害，以及理解为什么这个地方没被测试到，为什么会产生这个测试漏洞。

软件质量的目标需要以每个团队成员的高度责任心做保障，要保证质量就应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敢于承担工作失误所带来的严重后果。

4. 培训

培训虽然没教育重要，但其重要性还是不言而喻的。培训应考虑项目的短期需求，也要考虑团队成长的长期需求；应考虑不同层次的人员需求，也要考虑不同角色的培训需求。培训要抓住下列几点。

- 新人的培训。所有的新人须得到良好的培训，做到“合格者才能上岗”，不合格的测试人员是不能着重参与到实践项目中去的，否则会让大家提心吊胆。
- 常规性培训。日常的培训主要为项目服务，次要的目的是考虑个人和团队的成长。培训起着引导作用，包括减少大家在学习上走弯路的机会，但测试人员更多要靠自己的学习，学以致用。
- 多层次培训。构造一个培训体系，使之可以覆盖各个层次的人员。新人或初级人员的培训很重要，通过培训可以确保测试工作达到一个基本要求。相对资深的测试员和老员工也需要得到关心，使他们的成长更快些，如图 12-2 所示。
- 培训要务实。要成为真正对工作有帮助的培训。培训要落到实处，培训目标和计划都必须可行，培训的过程也要跟踪，最后还要通过考试检查培训的效果。

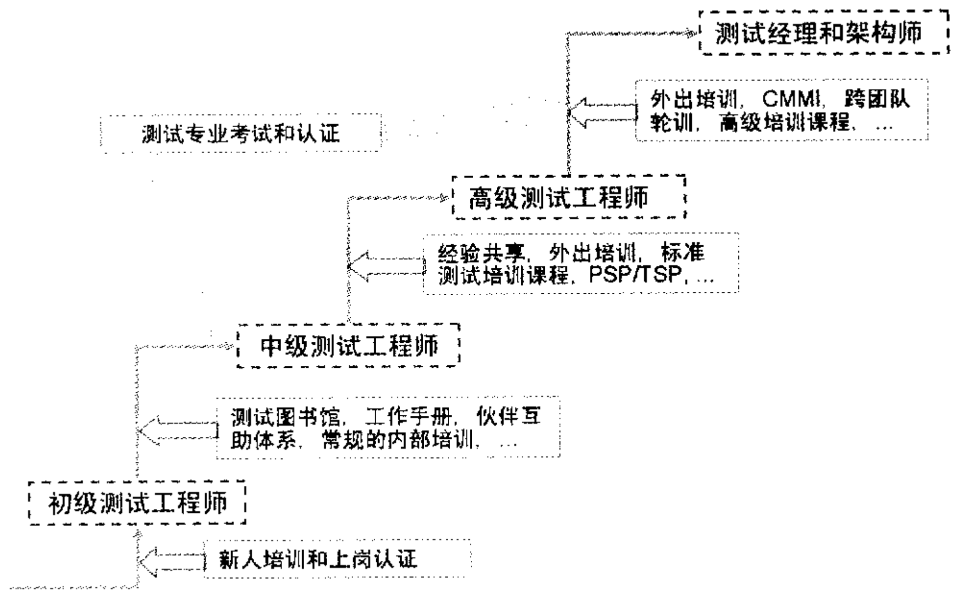


图 12-2 帮助测试人员发展的培训体系

12.5 软件测试成熟度模型

软件测试过程是软件开发过程的一部分，如果在软件过程中已经存在一个成熟度模型（CMM/CMMI），为软件测试过程建立一个成熟度是否有意义呢？可以从正反方面来考虑。

（1）从反面考虑，测试过程不是孤立的，是与软件开发的其他过程（工程过程、支持过程、管理过程、组织过程和客户－供应商过程等）相辅相成的、交织在一起的。软件过程中已有的东西（培训、组间协调、评审、技术革新等）都适合测试过程，而且测试过程要达到预期的目标，仅仅依靠测试过程自身的改进是不可能的，还需要其他过程的帮助和协同作用，如质量管理、配置管理、项目管理等。

（2）从正面看，对于测试团队，总想设定一个目标——如何将自己的团队建成一流的团队。要达到这个目标，不可能一蹴而就，所以需要分阶段设立目标，最终使自己的团队达到一个最高的水平。

从正面看，建立一个成熟的模型还是有意义的，至少对于许多独立的、具有一定规模的测试团队来说是有意义的，例如，软件外包趋势越来越明显，其中有比较多的软件外包公司就专门从事软件测试的业务，对这些公司，软件测试过程就是一个相对独立的过程。

从反面看，应时时刻刻提醒自己，测试过程的改进需要全公司的关注，需要和整个软件过程的改进一起来规划、设计和实施。

12.5.1 从 CMM/CMMI 得到的启发

当初建立 CMM，是出于支持美国国防部对软件承包商的能力进行客观评价的需要，这种能力最终体现在软件公司能否及时交付高质量的软件产品上，即控制项目进度和质量的能力。所以，CMM 的思想就是以过程为基础进行质量控制，把质量控制从事后检测转变为事前预防，尽量减少设计上比较大的修改。所以，在 CMM 中，有不少关键过程域都与软件测试相关，例如直接相关的有：

- 软件质量保证、软件质量管理。
- 同行评审。
- 软件定量管理。
- 缺陷预防。

在 CMMI 中，又增加了一些与测试直接相关的过程域，如验证、确认、风险管理、因果分析和解决方案，这些过程域越来越多地强调对缺陷因素的分析，以及找到产生缺陷的根本原因，创造“缺陷预防”的环境。

在 CMM 有一张图（如图 12-3 所示），它改变了传统的软件测试概念，并将软件测试扩展到整个软件开发过程。在图中包含了静态测试和动态测试，过程的成熟度越高，更多的缺陷被发现得越早，它和我们提倡的“软件测试就是尽早地发现尽可能多的缺陷”是一致的。

过程成熟度等级	需求	设计	编码	确认测试	系统测试	发布	备注
	10%	40%	50%	0%	0%	0%	缺陷分布
5	5%	20%	40%	20%	15%	<4%	各阶段缺陷发现分布
4	3%	13%	30%	30%	25%	5%	
3	0 %	3%	20%	40%	35%	8%	
2	0 %	0%	3%	30%	45%	15%	
1	0%	0%	2%	10%	55%	35%	
	1	1	1	5	15	20	修复成本

图 12-3 依据 CMM 等级设定的软件阶段性缺陷清除效率

从图中可以知道，需求和设计共占总缺陷数的 50%，编码占总缺陷数的 50%。在级别 2 和 3 的成熟度上，只有不到 5%的缺陷是在需求和设计阶段被发现的，说明这时需求评审和设计评审还没有启动或做得比较差。但是，在级别 3 有一个很大的进步，单元测试已有明显起色，从级别 2 的 3%提高到了 20%。级别 2 的改进出现在确认测试阶段，从 10%提高到了 30%。

到了级别 4，需求评审和设计评审的情况有了较大的改进，16%的缺陷可以在需求和设计阶段被发现。同时，单元测试得到进一步的加强，从 20%提高到了 30%。

到了级别 5，需求评审、设计评审和单元测试不断得到改进，25%的缺陷可以在需求和设计阶段被发现，也就是在本阶段产生的缺陷有一半可以被发现出来。而单元测试可以发现编码自身产生缺陷的 80%。

如果从缺陷所带来的成本（劣质成本）来进行分析，也可以知道，随着过程成熟度的提高，其劣质成本也越来越低，如表 12-3 所示。在图 12-3 中，随着时间的推移，成本的增大还是比较保守的，一般假定软件发布后，发现缺陷的成本是需求阶段的 100 倍。

表 12-3

级 别	公 式	成本值	相对级别 5 的倍数
5	65%×1 +20%×5 + 15%×15 + 4%×20	4.7	
4	46%×1 +30%×5 + 25%×15 + 5%×20	6.71	**1.43**
3	23%×1 +40%×5 + 35%×15 + 8%×20	9.08	**1.93**
2	3%×1 + 30%×5 + 45%×15 + 15%×20	11.28	**2.4**
1	2%×1 + 10%×5 + 55%×15 + 35%×20	15.77	**3.36**

知识点

（1）软件过程能力（software process capability），是软件过程本身具有的按预定计划生产产品的固有能力，或者说是遵循软件过程能够实现预期结果的程度。软件过程能力提供了预测软件项目开发的数据基础，即帮助预测实施新的软件项目时最有可能出现的结果。

（2）软件过程性能（software process performance）表示遵循软件过程所得到或软件过程执行的实际结果。

（3）软件过程性能关注已得到的结果，而软件过程能力则关注预期的结果。一个项目的软件过程性能决定于内部子过程的执行状态，但同时，由于一个特定项目的属性和执

行会受到实际环境的限制和影响，项目的实际性能并不能充分反映组织的软件过程能力。所以评价过程的成熟度，不能仅仅依据软件过程性能来判断，必须结合软件过程能力来进行综合评估。

（4）软件过程成熟度（maturity），是指一个具体的软件过程被明确地定义、管理、评价、控制和产生实效的程度。所谓成熟度包含着一种增长潜力，同时也表明了组织实施软件过程的实际水平。

（5）1987 年，卡内基—梅隆大学软件工程研究所（SEI）对支持美国国防部对软件承包商的能力进行了客观评价，提出了关于软件的《能力成熟度模型框架》，并于 1991 ~ 1993 年发表了《软件能力成熟度模型》即 SW-CMM 1.0 版和 SW-CMM 1.1 版。SEI 于 1999 ~ 2000 年发表了《系统工程和软件工程综合能力成熟度模型》CMMI-SE/SW 1.0 版。

（6）设计 CMM 是为了指导软件组织通过确定当前过程的成熟度和识别对软件质量和过程的改进等至关重要的少数问题，来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，来稳步地改善全组织的软件过程，使其软件过程能力持续不断地增长。

（7）CMMI 其实就是 SW-CMM 的修订本，是集成三个特殊过程的改进模型（软件 CMM、系统工程能力评估标准以及集成化产品和过程开发模型）而发展起来的。

详细内容见 *http://www.sei.cmu.edu*。

12.5.2 目前 TMM 存在的问题

表 12-4 对目前已经存在的测试成熟度模型（TMM 或 TCMM）进行了简单的分析，从表中可以看出，这样的模型存在以下一些问题：

（1）因为是从 CMM 直接延伸到测试，所以容易脱离测试过程的实际改进轨迹。例如，许多组织的测试是从缺陷开始的，而且也知道将缺陷划分为不同的等级，先有测试用例，最后才有测试计划，而不是先有计划。

（2）关注测试自身的活动不够，例如虽然谈到测试工具和测试数据库，但几乎没有涉及到缺陷清除率、自动化测试框架和测试流程等。

（3）测试领域内的深度不够，例如测试最大的特征就是风险因素，以及测试的核心——测试用例，如何降低测试风险，提高测试用例的质量，并没有清晰的目标。

（4）虽然在过程监控、度量方面的内容较多，但是如何对实际测试能力（测试覆盖

率、效率等）进行量化评估，并没有具体的对策，可实施性差。

表 12-4 目前测试成熟度模型的基本描述

级 别	简单描述	特 征	目 标
1	Initial（初始级）。 测试处于一个混乱的状态，缺乏成熟的测试目标，测试处于可有可无的地位	还不能把测试同调试分开； 编码完成后才进行测试工作； 测试的目的是表明程序没有错； 缺乏相应的测试资源	
2	Phase Definition（阶段定义级）。 测试目标是验证软件符合需求，会采用基本的测试技术和方法	测试被看做是有计划的活动； 测试同调试分开； 在编码完成后才进行测试工作	启动测试计划过程； 将基本的测试技术和方法制度化
3	Integration（集成级）。 测试不再是编码后的一个阶段，而是贯穿在整个软件生命周期中。测试建立在满足用户或客户的需求上	具有独立的测试部门； 根据用户需求设计测试用例； 有测试工具辅助进行测试工作； 没有建立起有效的评审制度； 没有建立起质量控制和质量度量标准	建立软件测试组织； 制订技术培训计划； 测试在整个生命周期内进行； 控制和监视测试过程
4	Management and Measurement（管理和度量级）。 测试是一个度量和质量的控制过程。在软件生命周期中评审被作为测试和软件质量控制的一部分	进行可靠性、可用性和可维护性等方面的测试； 采用数据库来管理测试用例； 具有缺陷管理系统并划分缺陷的级别； 还没有建立起缺陷预防机制，且缺乏自动地对测试中产生的数据进行收集和分析的手段	实施软件生命周期中各阶段评审； 建立测试数据库并记录、收集有关测试数据； 建立组织范围内的评审程序； 建立测试过程的度量方法和程序； 进行软件质量评价

续表

级 别	简单描述	特 征	目 标
5	Optimization（优化级） 具有缺陷预防和质量控制的能力; 已经建立起测试规范和流程，并不断地进行测试改进	运用缺陷预防和质量控制措施; 选择和评估测试工具存在一个既定的流程; 测试自动化程度高; 自动收集缺陷信息; 有常规的缺陷分析机制	应用过程数据预防缺陷，统计质量控制; 建立软件产品的质量目标; 持续改进测试过程; 优化测试过程

12.5.3 实用测试成熟度模型的建立

那么如何建立测试成熟度模型？上两节的讨论，是否对建立测试成熟度模型有启发和帮助？肯定是有的，从图 12-3 出发，就能找到量化的模型来衡量测试的成熟度。

1. 测试的成熟度

首先，应该定义测试的成熟度是什么？测试成熟度表现在测试的能力和测试的性能上。测试的能力应该包括下列内容：

- 对需求和设计评审的能力。
- 测试用例的覆盖率。
- 测试自动化的水平。
- 测试方法的运用能力。
- 测试过程的管理能力。
- 缺陷描述能力。
- 质量评估能力。

而测试的性能是测试过程的实际表现，最终表现为缺陷整体清除率和测试所用的成本。而影响测试性能的因素，除了测试能力之外，还有下列主要因素：

- 软件开发环境。
- 测试基础设施支持水平。
- 测试资源（包括人力资源、软硬件资源）。

- 测试培训状况。

我们清楚地知道，测试工作的主要对象是软件阶段性产品和最终发布的产品：

- 阶段性产品，包括需求定义文档、产品设计规格说明书、代码等，主要靠评审来保证质量。
- 最终发布的产品是软件系统，靠功能测试、UI 测试和系统测试等来保证质量。

实际成熟度和测试的工作对象有关系，换句话说，测试的成熟度是在某个阶段内或某几个阶段内的成熟度，也是在整个生命周期内的成熟度。很显然，成熟度是因整个生命周期而存在的，只是这个过程，是由阶段性发展到整个生命周期的。

2. 影响测试成熟度的关键因素

如何做好测试工作？有哪些因素会影响测试工作的质量？根据第 11 章的讨论，测试管理有 5 个层次，包括团队、流程、方法、项目和执行，那么这些层次又分别包含哪些因素呢？

（1）测试执行。执行相对容易，缺陷报告和跟踪是测试执行的中心，但依赖于基础设施，如测试环境、测试管理系统、测试自动化水平等。

（2）测试项目的管理，可以从一般软件项目管理中获得借鉴、帮助和支撑。在这一层次中，测试计划是重点，而测试策略、测试风险控制又是测试计划中的焦点。

（3）测试方法，例如黑盒测试方法和白盒测试方法已逐渐趋于成熟，但其应用、综合运用还是依赖于实践和经验积累。测试方法最终体现在测试用例和测试脚本之上，包括测试工具的选择和开发之上。

（4）测试流程是测试的核心，包括需求和设计评审的流程、测试计划和测试用例评审的流程、测试度量的流程、缺陷生命周期的定义等。如果流程定义好了，测试过程的改进和优化才能成为现实。

（5）测试团队，建立一支独立的测试团队是基本的要求，当然将这支团队建成一流的专业团队并不容易，需要进行培训、绩效考核和职业规划等，其中为测试人员制定科学、公正的绩效考核方法，会有一定的难度。

从上面分析就可以列出一系列影响测试成熟度的关键因素，例如基础设施建立、测试环境所需的资源、测试工具的使用、测试流程是否纳入整个开发流程管理体系等。除此之外，还有一些基本问题，如测试在组织中的地位、测试和质量保障的关系等。

3. 测试成熟度的模型

建立测试成熟度模型并不是一件容易的事，但目的应该是清楚的，就是要让每一个软件组织清楚地知道软件测试有哪些关键的领域，要向哪个方向努力，各个发展阶段的目标是什么。其次，作为一个模型，应该要比较容易理解，易于采用，因为模型的建立就是让别人用的。

测试成熟度模型的建立，应以测试自身工作为核心，并且所建立的模型应为测试团队和测试人员服务。虽然我们时刻要记住测试是软件开发的一部分，但是软件开发的成熟度模型是由 CMM/CMMI 来承担的。例如软件缺陷预防，更重要的是需求定义、设计和编程的关注点，质量是在软件构建中产生的，而对于测试人员，软件缺陷的分析可以帮助测试用例的设计、测试工具的改善和测试流程的改进。测试的成熟度应从缺陷报告、测试用例设计和测试自动化体现出来，并最终反映在测试所交付的结果中，例如缺陷清楚率就是一个重要的指标。

建立一套测试成熟度模型，需要一套完整的数据分析、详细说明的文档，由于篇幅所限，这里仅以表 12-5 呈现出来。这里的测试成熟度模型、自然的级别名称、每一项要求都很清楚，对实际测试工作有很好的参考实用价值。那么就以这个实用的测试成熟度模型作为全书的总结，画上圆满的句号。

表 12-5 实用的测试成熟度模型的基本描述

级别 项目及说明		I 混乱期	II 启蒙期	III 成长期	IV 稳定期	V 优化期
测试的地位	随机、可有可无	✓				
	编程之后（只有程序测试）		✓	⊙		
	整个开发生命周期（含评审）			⊙	✓	✓
测试结果	测试报告	⊙	✓	✓	✓	✓
	规范的测试报告模板			✓	✓	✓
	质量评估分析				✓	✓
	作为产品发布的最重要的依据					✓
	整体清除率	<85%	>=85%	>92%	>96%	>98%
	需求阶段产生的缺陷清除率（阶段评审）	0%	0%	>10%	>40%	>60%

续表

级别 项目及说明		I 混乱期	II 启蒙期	III 成长期	IV 稳定期	V 优化期
测试结果	设计阶段产生的缺陷清除率（阶段评审）	0%	0%	>20%	>50%	>70%
	编程阶段产生的缺陷清除率（单元测试）	5%	>20%	>40%	>60%	>80%
发现缺陷	目的不明确	✓				
	验证		⊙	✓		
	验证和确认				✓	✓
缺陷	严重性、优先级划分	⊙	✓	✓	✓	✓
	完整描述的模板			✓	✓	✓
	缺陷生命周期				✓	✓
	事后分析、跟踪				✓	✓
测试用例设计	针对设计规格说明书	⊙	✓	✓	✓	✓
	针对用户需求			✓	✓	✓
	系统的方法、持续改进				✓	✓
测试计划	软件项目计划的一部分		✓	✓	✓	✓
	独立的测试计划			✓	✓	✓
	包含测试策略				✓	✓
	测试风险控制机制				✓	✓
	质量目标融入测试计划					✓
测试方法运用	方法被隔离开来使用		✓	✓	✓	✓
	综合运用			⊙	✓	✓
测试范围	需求、设计评审				✓	✓
	单元测试		⊙	✓	✓	✓
	集成测试			✓	✓	✓

续表

级别 项目及说明		I 混乱期	II 启蒙期	III 成长期	IV 稳定期	V 优化期
测试范围	功能测试（包括UI）	⊙	✓	✓	✓	✓
	负载测试、性能测试			✓	✓	✓
	可靠性、可用性、安全性等测试				✓	✓
测试工具	简单使用		✓	✓		
	大量的脚本开发			✓	✓	✓
	根据流程选择工具				✓	✓
	工具测试部分占整个测试的比重	0%	>10%	>30%	>50%	>80%
	构成一个完整的框架体系					✓
测试管理系统	缺陷管理	⊙	✓	✓	✓	✓
	测试用例、套件等的管理			✓	✓	✓
	与需求、配置管理等集成				⊙	✓
测试环境	独立的资源，但不够充分		✓			
	充分的、独立的资源			⊙	✓	✓
	专业的测试实验室（性能、负载等）				⊙	✓
测试团队	专业测试人员	⊙	✓			
	独立的专业测试团队			✓	✓	✓
	明确的多样化的测试角色和责任			⊙	✓	✓
	测试专业性的绩效考核					✓
	配备合理、权威性				✓	✓

续表

级别 项目及说明		I 混乱期	II 启蒙期	III 成长期	IV 稳定期	V 优化期
测试流程	作为软件开发过程的一部分		⊙	✓	✓	✓
	一部分独立的测试流程			✓		
	完善的测试流程				✓	✓
	持续改进					✓
培训	针对项目的测试培训	⊙	✓	✓	✓	✓
	专业的、有层次的系统测试培训			✓	✓	✓
	从客户反馈中获得教育、经验				✓	✓
管理	里程碑控制		✓	✓	✓	✓
	全过程监控			✓	✓	✓
	基础设施的支持	0%	>10%	>30%	>60%	100%
	事先定义的应急方案				✓	✓
	技术创新				✓	✓
	不断评估测试方法					✓
	按流程引进测试新技术、新方法				✓	✓
测试评审	缺陷		⊙	✓	✓	✓
	测试用例、测试计划				✓	✓
	测试报告				✓	✓
度量	自动收集			✓	✓	✓
	缺陷状态报告	⊙	✓	✓	✓	✓
	缺陷分布、趋势分析			✓	✓	✓

续表

级别 项目及说明		I 混乱期	II 启蒙期	III 成长期	IV 稳定期	V 优化期
度量	测试用例深度、缺陷关联度分析				⊙	✓
	测试效率度量				⊙	✓
	测试覆盖率分析			⊙	✓	✓
	统计质量控制（如质量信任度）					✓

注：⊙ —— 表示不确定的、不定期活动，✓ —— 有计划的或制度化的活动。

A

附录A

软件测试全景图

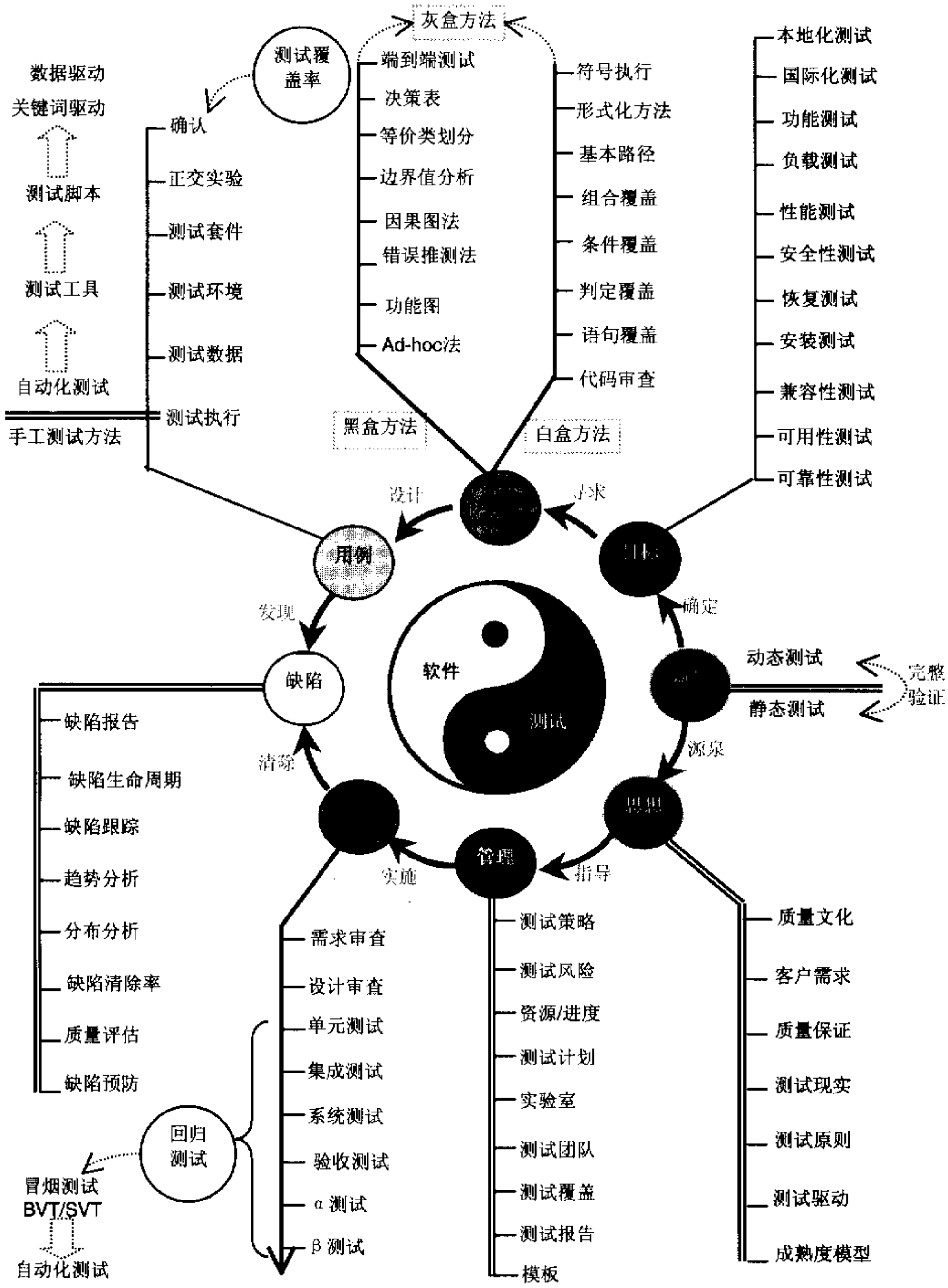


图 A-1 从内容和关系来描述测试的全貌

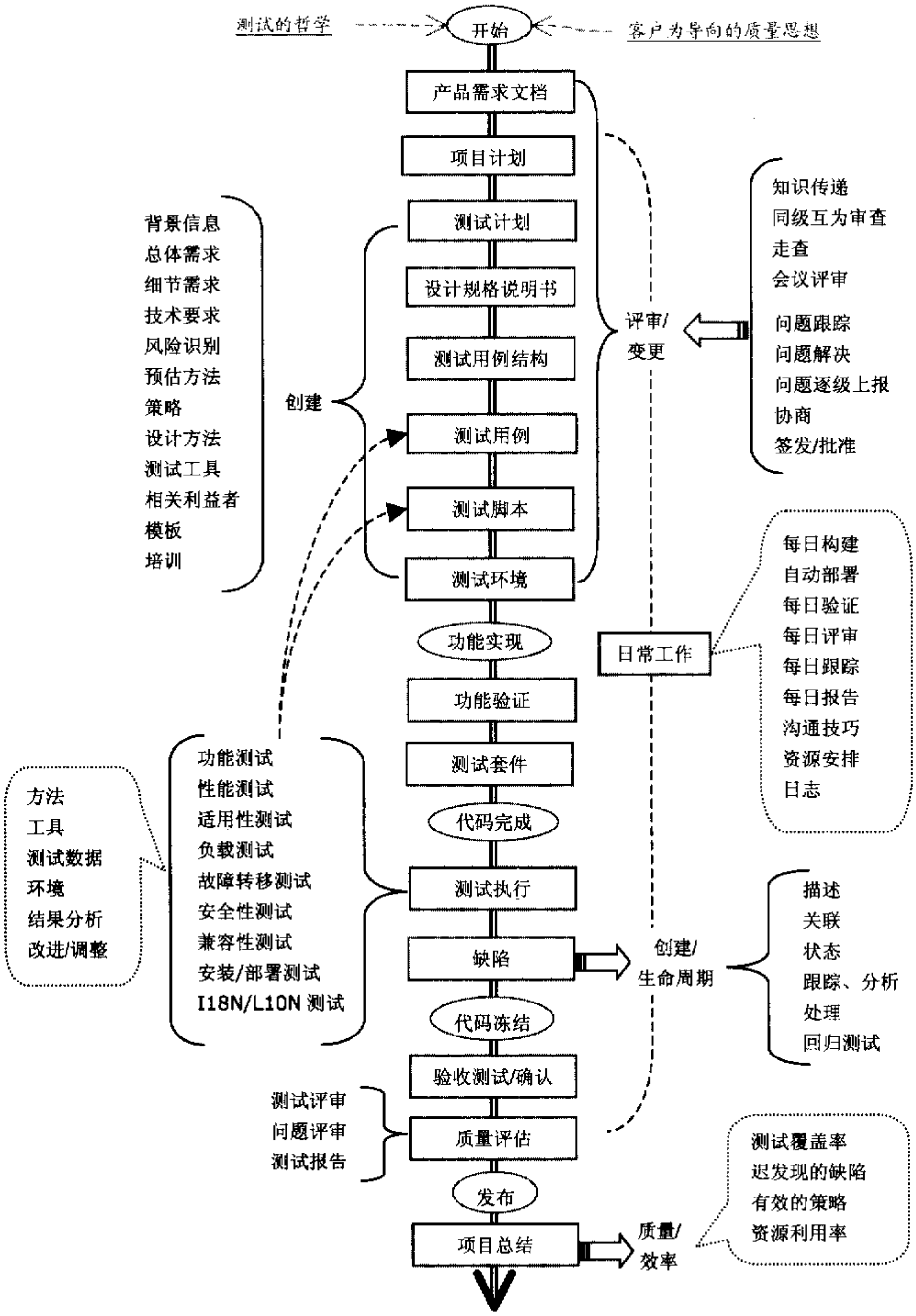


图 A-2　从测试过程的时间顺序来描述测试的全貌

B

附录B

完整的项目检查表

1. 项目管理

序号	类别	描述
1	开发	在开发新功能前，发一封概括性的 SPEC 邮件（标题“SPEC:”）给项目组邮件列表中的所有人（回应期为 2 天，如果被批准则继续）
2	开发	在提交新功能或可能妨碍现有测试的代码之时，需增加相应的自动测试用例并更新手工测试用例
3	CVS	只有指定的开发人员才可以提交代码改动，而且必须通过冒烟测试（手工和自动测试）
4	CVS	在对用户界面、公共 API、文件格式或任何其他公共界面进行改动后，需在某个特定的共享文件中增加一项记录来注明
5	发布	每 4~5 星期会进行一次小型公共发布；见里程碑清单
6	发布	每个星期会进行一次微型公共发布；见里程碑清单

2. 编码规范

序号	类别	描述
1	注释	每个源文件必须带有页眉注释
2	注释	每个源文件在类/接口名前应该带有一个介绍性的注释，以说明该类的用途和正确使用该类的建议
3	注释	如果计划有未来的代码改动，添加一项带有“@todo”关键字的注释备忘录
4	注释	所有的公共成员必须有至少一行 javadoc 注释
5	注释	所有带有 2^+ 同类参数的方法需要为每个参数写一个 javadoc 注释
6	清晰度	所有公共字符串和数值常量需在类文件顶部被声明
7	可移植性	所有的代码应具有可移植性
8	国际化	所有代码需要使用合适的 Java 国际化类（见 *http://java.sun.com/docs/books/tutorial/i18n/intro/checklist.html* 和 *http://java.sun.com/j2se/1.4.2/docs/guide/intl/intlTOC.doc.html*）
9	格式	使用 4 个空格进行缩进，而不用 tab；使用 UNIX 行结束约定
10	格式	参照 Java 代码编写规范（*http://java.sun.com/docs/codeconv/*），尤其在命名、注释和构架等方面

3. 用户界面规范

序号	类别	描述
1	可用性	遵循可用性规则
2	字体	尽可能使用所定义的默认字体
3	字体	其他字体，遵照 Spec 的定义
4	字体	只使用普通或粗体字格式，尽量不使用斜体字、下划线
5	间距	使用公司通用 UI 说明书和 Spec 中的对话框版面设计常数
6	控件	只使用 LEADING 和 TRAILING Component Orientation
7	控件	版面设计管理使用默认定位
8	版面设计	对话框应该把相关的组件排列入逻辑组中（行或列）
9	标签	对话框标签的尾部应该有一个 ':' 字符
10	标签	标签的大小写需要正确（如“Name:”，“Last Name:”）
11	质量	用户界面应完整、一致，全部有效功能的用户界面具有可视性

4. 可用性

序号	类别	描述
2	导航	组件的跳格顺序（当按下跳格键时）应该符合版面设计的逻辑
3	导航	每个按钮和菜单都应该有唯一的快捷键
4	导航	每个标签有一个快捷键，通过 setMnemonic（）定向焦点至相应的组件；使用 setLabelFor（）来定位相应的组件
5	导航	每个重要的菜单项拥有一个使用 setAccelerator（）设定的加速器
6	可辨认性	图形工具栏按钮和静态图像（包括树状图和列表中的图标）应当使用一个带有描述的图标
7	可辨认性	每个工具栏按钮应当有一个 tooltip （setToolTipText（））
8	可辨认性	每个不带标签的组件应当通过 setAccessibleName（）设置一个用户可读的名字
9	参考	*http://www-3.ibm.com/able/guidelines/java/accessjava.html* *http://java.sun.com/docs/books/tutorial/uiswing/misc/access.html*

5. 里程碑清单

序号	类别	描述
1	准备	更新“小”版本号 Y（x.Y.z）
2	子里程碑	里程碑的指导方针同样适用于子里程碑（macrostone）
3	质量	质量保证队伍须对所有被改动的功能进行功能测试
4	质量	质量保证组长发送邮件至项目组和管理人员
5	软件包	更新、测试并验证构建包小版本号（在“about”对话框中）
6	通告	项目组长发送一份通告邮件至指定的邮件组

6. 子里程碑清单

序号	类别	描述
1	准备	更新“微”版本号 Z （x.y.Z)
2	软件包	二进制文件是在一个新的干净 CVS 环境中打包的 （通过使用'cvs co'）

续表

序号	类别	描述
3	软件包	版本号更新在发布备忘录上，任何新的缺陷和更新的需求也要被确定
4	质量	100%单元测试通过
5	质量	100%冒烟测试通过
6	质量	如果某项单元或冒烟测试失败，发送标题为“BROKEN: ”的邮件给开发
7	软件包	下载更新的二进制文件
8	通告	项目组长发送一份通告邮件至指定的邮件组

7. 冒烟测试

序号	类别	描述
1	运行	删除$HOME/.project 下的文件后，软件仍能够正常运行
2	功能测试	在工具栏上按下日、工作周（5天）、周和月按钮
3	功能测试	双击以编辑一个日历事件和编辑 profile，该事件应当浏览中发生相应的变化
4	功能测试	在另一个客户端中打开日历，检查该事件改动是否已被保存
5	功能测试	在任务版面创建一个新任务；标记为已完成
6	功能测试	退出、重新启动，检查所有的改动是否已被记录

8. 提交清单

序号	类别	描述
1	编码	检查编码规范
2	用户界面	检查用户界面规范
3	提交	为每个 CVS 文件增加最少 1 行的注释；将相关联的文件合并后单次提交
4	搭建	必须在一个 CVS 创建的新环境中搭建
5	搭建	在已存在环境中搭建时，必须事先运行 'ant clean'
6	测试	冒烟测试必须通过

C

附录C

测试计划（GB8567-88）

1 引言

1.1 编写目的

写出本测试计划的具体编写目的，指出预期的读者范围。

1.2 背景

说明：

a. 列出测试计划所从属的软件系统的名称；

b. 指出该开发项目的历史，列出用户和执行此项目测试的计算中心，说明在开始执行本测试计划之前必须完成的各项工作。

1.3 定义

列出本文件中用到的专门术语的定义和缩写单词外文首字母的原词组。

1.4 参考资料

列出要用到的参考资料，如：

a．本项目经核准的计划任务书或合同、上级机关的批文；

b．属于本项目的其他已发表的文件；

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标准。列出这些文件的标题、文件编号、发表日期和出版单位，说明得到这些文件资料的来源。

2 计划

2.1 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。

2.2 测试内容

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

2.3 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。

2.3.1 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）。

2.3.2 条件

陈述本项测试工作对资源的要求，包括：

a．设备所用到的设备类型、数量和预定使用时间；

b．软件列出将被用来支持本项测试过程，但本身又并不是被测软件的组成部分的软

件，如测试驱动程序、测试监控程序、仿真程序、桩模块等；

c．列出预期在测试工作期间可由用户和开发任务组提供的工作人员人数，包括一些特殊要求的工作人员，如倒班操作和数据键入人员。并告知他们的技术水平及有关的预备知识。

2.3.3 测试资料

列出本项测试所需的资料，如：

a．有关本项任务的文件；

b．被测试程序及其所在的媒体；

c．测试的输入和输出举例；

d．有关控制此项测试的方法及过程的图表。

2.3.4 测试培训

通过说明或引用资料列出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。

2.4 测试2（标识符）

用与本测试计划2.3节相类似的方式说明用于另一项测试工作计划及其后各项测试内容的测试工作计划。

3 测试设计说明

3.1 测试1（标识符）

说明对第一项测试内容的测试设计考虑。

3.1.1 控制

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

3.1.2 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。

3.1.3 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。

3.1.4 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、中间步聚和运行结束方式。

3.2 测试2（标识符）

用与本测试计划3.1节相类似的方式说明第2项测试设计及其后各项测试工作的设计考虑。

4 评价准则

4.1 范围

说明所选择的测试用例能够接受的范围及其局限性。

4.2 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，将测试结果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。

4.3 尺度

尺度用来判断测试工作是否能通过，如合理的输出结果的类型、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。

附录D

完整的测试工具列表

通用功能测试工具

厂　商	工具名称	说　明
HP-Mercury	Winrunner	企业级自动化测试工具
	Quicktest pro （QTP）	B/S 系统的自动化功能测试
	XRunner	Unix 下的自动测试工具
Compuware	QARun	应用的功能测试工具
	WebCheck	Web 站点质量分析工具
	TestPartner	功能测试工具
Parasoft	SOA test	Web 服务功能测试、负载测试、客户端测试工具
	WebKing	强大的 Web 测试工具
IBM	Rational Robot	既可以做功能测试，也可以进行性能测试
	Rational Visual Test	用于 GUI 功能测试
	Rational Functional Tester	自动功能测试和回归测试工具

续表

厂　商	工具名称	说　明
Segue	SilkTest	功能测试工具
	SilkTest International	用于企业应用的功能测试工具
Empirix	e-Tester	易于使用的 Web 应用测试工具
Radview	WebFT	测试 Web 应用的有效性、完整性和正确性，只能模拟单个用户行为
AutomatedQA	TestComplete	自动测试管理工具，支持项目层级的单元测试、功能测试、回归测试、分布式测试以及 HTTP 性能测试
Seapine	QA Wizard	测试基于 Web 和.NET 的 Windows 应用程序
RedStone	EggPlant	为苹果操作系统开发的 GUI 测试工具，但也可用于测试其他操作系统，如 Windows、solaris、HP-UX、AIX 和 Linux
Microsoft	Visual Studio Test Edition	功能测试工具
Minq	PureTest	HTTP 请求验证和 Web 内容及其结构检查的测试工具
Autotester	Autotester	测试设计工具，详见 *http://www.autotester.com/*
Original Software	Testbench400	Web 测试工具，详见 *www.origsoft.com*
AutoTester	AutoTester Web	Web 测试工具
Open Source	MaxQ	Web 功能测试工具
	Selenium	Web 功能测试工具
	WebInject	针对 Web 应用程序和服务的测试工具
	Marathon	针对使用 Java/Swing 开发 GUI 应用程序的测试框架

性能测试工具

厂 商	工具名称	说 明
HP-Mercury	LoadRunner	性能测试、压力测试，模拟多用户、定位性能瓶颈
	SiteScope	性能监视的软件，可用于压力测试
	Topaz	可用于压力测试
Compuware	QaLoad	强负载下应用的性能测试工具
Quest	PerformaSure/benchmark	分布式 J2EE 应用系统的分析工具，适合集成测试
Segue	Silkperformer	企业级的关键任务的负载测试工具
	Silkperformer Lite	经济有效的 Web 应用负载测试工具
	SilkCentral™ Performance Manager	应用性能管理解决方案，用于关键任务应用的质量优化。在可用性、性能和精确性上对最终用户的体验进行监控
Empirix	e-Load	压力测试工具
IBM	Rational Robot	既可以做功能测试，也可以进行性能测试
	Rational Performance Tester	IBM 基于 Eclipse 平台及开源的测试及监控框架 Hyades，开发出来的最新性能测试解决方案
RadView	WebLoad	专业的 Web 性能测试和分析工具
Microsoft	Web applicaton stress tool	免费的 Web 性能测试工具
	Application center test	Web 应用程序性能测试
Empirix	Bean-test	对 EJB 应用软件进行负载和压力测试工具
Minq	PureLoad	负载测试软件
Metron	Athene APR	基于 ITIL 的容量、性能分析和报告工具
Facilita	ForeCast	来自于英国的性能和负载测试工具
Lawrence/Berkeley	Sniffer	针对 TCP / IP 协议的安全性的监听工具

续表

厂 商	工具名称	说 明
Open Source	Jmeter	功能和性能测试的工具，侧重 Web 性能的测试
	openSTA	分布式的性能、负载测试工具
	Siege	压力测试和评测工具
	StressMark	模拟多线程或多客户端的自动化压力测试工具
	DBMonster	数据库负载和压力测试

白盒测试工具

厂 商	工具名称	说 明
Parasoft	Jtest	Java 的单元测试和代码标准校验的工具
	C++test	针对 C/C++源代码进行自动化单元测试的工具
	.test	.NET 开发平台的单元级测试与静态分析工具
	Codewizard	高级 C/C++源代码分析工具，包括代码走读与自动防错
	Insure++	自动化的内存错误、内存泄漏的精确检测工具
	DataRecon	数据库设计认证、数据有效性校验和监视工具
Compuware	Numega devpartner studio	一组白盒测试工具，主要用于检查应用的可靠性和稳定性
	DevPartner Java Edition	Java 语言的单元测试静态分析工具
	BoundsChecker	代码检错工具，适合单元测试、集成测试
	SmartCheck	支持 VB 函数调用次数、所占比率统计以及稳定性跟踪

续表

厂　商	工具名称	说　明
	DBPartner	交互式的数据库存储过程开发、调试和优化工具
AutomatedQA	AQtime	隔离并排除代码中含有的性能问题和内存/资源泄露问题
	QESatJava	Java 软件分析与测试工具
Kailesoft	Visual Unit	可视化、标准化、高效率的新一代单元测试工具
Gimpel	Software PC-lint	C/C++语言的静态代码检测工具
Macabe	Macabe	代码静态度量分析度量检查工具
Borland	Optimizeit Suite	Java 代码优化工具包
Quest Software	JProbe Suite	Java 内存泄漏检测和代码级性能优化工具
	Sql optimizer	SQL 优化工具
ej-technologies	JProfiler	Java 性能跟踪
TeleLogic	Logiscope	静态测试工具，可用于单元测试和集成测试的覆盖率收集
	rulecheck	编程规则检查工具
IBM	Rational Purifyplus	运行时分析解决方案：内存错误检测、内存泄漏检测、应用程序性能概要分析和代码覆盖分析
	Rational Rational Test RailTime	查看源代码，检查是否遵守 C 源代码标准，并得出报告
开源	cactus	基于 JUnit 的简单测试框架，测试服务端 Java 代码
	Hansel	测试覆盖率的工具，可结合 JUnit 使用
	TestNG	单元测试工具
	StrutsTestCase	JUnit TestCase 类的扩展，提供基于 Struts 框架的代码测试

续表

厂　商	工具名称	说　明
开源	JFCUnit	Java 单元测试工具
	Java Test Runner（JTR ）	基于反转控制（IOC）的 J2EE 测试框架
	Xunit	最流行的单元测试工具，常用的根据语言不同分为 JUnit（Java），CppUnit（C++），DUnit（Delphi），NUnit（.NET），PhpUnit（Php）等
	JUnit	回归测试框架
	Httpunit	基于 JUnit 的一个测试框架，关注于测试 Web 应用
	DUnit	Xunit 家族中的一员，用于 Dephi 的单元测试
	cppunit	C++单元测试工具
	Nunit	为.NET 框架生成的开放源代码单元测试框架
Linux 平台工具	MallocDebug	调试跟踪工具
	Valgrind	Linux 系统的调试内存问题的工具
	Kcachegrind	--
	dmalloc	--
	ElectricFence	内存问题检测工具
	LeakTracer	是一个用于检查 C++程序内存泄漏的工具
	memprof	--
	ccmalloc	检查资源泄露
	mprof	--
	yamd	内存调试工具
	njamd	检测内存分配违例的调试程序
	mpatrol	用于代替 C 的 malloc 或 C++的 new 的动态库，可调试内存运行错误

嵌入式系统测试工具

厂　商	工具名称	说　明
Metrowerks	codetest	性能、代码覆盖率、动态内存分配的分析工具
IPL	Cantata/cantana++	面向源代码的测试分析工具
Reflex Technology	IceMaster	--
	System test	
DDC-I	scorecast	--
testquest	Testquest	通过提供一系列模拟信号输入来导引对被测系统的测试过程，包括键盘操作、鼠标或其他输入设备所产生的电信号
ATTOL	UniText	--
Vector software	vectorcast	扫描 Ada，C/C++和嵌入式 C++（EC++）源代码，自动生成测试代码来为主机和嵌入式环境构造可执行的测试架构
qronus	testrunner	
telelogic	Logiscope	测试覆盖分析，显示没有测试的代码路径，基于源码结构分析
TNI-Software	RT-Builder	用于建模、仿真、验证动态实时结构（包括多总线，多处理器）的一套完整解决方案

测试管理工具

厂　商	工具名称	说　明
HP-Mercury	TestDirector（QualityCenter）	能够在一个独立的应用系统中提供需求管理功能，并且可以把测试需求管理与测试计划、测试日程控制、测试执行和错误跟踪等功能融合为一体
Compuware	QADirector	测试的组织设计和创建以及管理工具
worksoft	certify	--
Silicon Valley Networks	VEReCOMM TestExpert	管理整个测试过程，从测试计划、用例、执行到报告

续表

厂 商	工具名称	说 明
aimware	Product manager	--
segue	SilkCentral Test Manager	全面的测试管理系统
telelogic	Doors	需求管理工具
empirix	e-manager	网站测试管理工具：可按需定制测试计划，文档，管理整个测试流程
IBM	Rational testmanager	计划、管理、组织、执行、评估以及报告个别测试用例或整个测试计划
RadView	TestView Manager	定制、管理各种测试活动
T-Plan	T-Plan Professional	支持任何规模的项目从单元测试直到用户验收测试的各个测试过程
Open source	testlink	开源的测试管理工具，提供了与 bugzilla、mantis 等工具整合的功能

缺陷管理工具

厂 商	工具名称
HP-Mercury	TestDirector （QualityCenter）
IBM	Rational ClearQuest
Compuware	TrackRecord
Seapine	TestTrack pro
McCabe	TrueTrack
Techexcel	Devtrack
IBM	Lotus Notes
Segue	SilkCentral Issue Manager
Merant	PVCS Tracker
Remedy	AR System
LealSoft	URTrack
华创	缺陷管理系统 BMS

续表

厂　商	工具名称
深软软件测试技术公司	Clarion
Hansky	Butterfly
开源	Bugzilla
	Mantis
	JIRA
	BugFree

配置管理工具

厂　商	工具名称
IBM	Rational ClearCase
Merant	PVCS Version Manager
Diamond	VCS
MKS	MKS Source Integrity Enterprise Edition
Borland	StarTeam
Computer Associates	AllFusion Harvest Change Manager
	CCC/Harvest （Change Configuration Control/Harvest）
Serena	ChangeMan Professional
Perforce	Perforce
McCabe	TRUEchange
Telelogic	SYNERGY CM
Microsoft	VSS
北大青鸟	JBCM
Hansky	Firefly
开源	CVS
	subversion
	SCCS
	RCS

E

附录E

代码审查的示范性列表

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模块的模式一致？
- 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整数值吗？下标总是在数组和字符串大小范围之内吗？
- 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？

- 为引用的指针分配内存了吗？
- 一个数据结构是否在多个函数或者子程序中引用，是否在每一个引用中明确定义了结构？

4. 数据声明错误

- 所有变量都赋予了正确的长度、类型和存储类吗？例如，在本应声明为字符串的变量却声明为字符数组了。
- 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？
- 变量有相似的名称吗？自定义变量是否使用了系统变量名？
- 是否存在声明过但从未引用或只引用过一次的变量？
- 在特定模块中所有变量都显式声明了吗？如果没有，是否可以理解为该变量将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了编译器对类型和长度不一致的变量的转换规则
- 赋值的目的变量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/模是否可能为零？
- 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

- 比较正确吗？如比较中应该是小于还是小于和等于？
- 存在分数或者浮点值之间的比较吗？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吗？逻辑计算如期进行了吗？求值次序有疑问吗？
- 逻辑表达式的操作数是逻辑值吗？例如，是否将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: 所需的全部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已正确设置; 是否处理对被调用的关键模块的意外情况(如丢失, 混乱)?
- 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 是否与之对应?
- 程序、模块、子程序和循环能否终止? 如果不能, 可以接受吗?
- 可能存在永远不停的循环吗?
- 存在的循环从不执行吗? 如果是这样, 可以接受吗?
- 如果程序包含像 switch—case 语句一样多的分支, 索引变量能超出可能的分支数目吗? 如果超出, 该情况能正确处理吗?
- 是否存在"丢掉一个"错误, 导致意外进入循环?
- 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匹配吗? 次序正确吗?
- 如果子程序有多个入口点, 引用的参数是否与当前的入口点没有关联?
- 常量是否当作形式参数传递, 意外在子程序中改动?
- 子程序更改了仅作为输入值的参数吗?
- 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, 在所有引用的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?

- 有处理文件或者外设不存在或者未准备好的错误情况吗？
- 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被占满等情况？
- 软件以预期方式处理预计的错误吗？
- 检查错误提示信息的准确性、正确性、语法和拼写了吗？

12. 逻辑和性能

- 全部设计已实现了吗？
- 逻辑被最佳地编码了吗？
- 提供正式的错误或例外子程序了吗？
- 每一个循环执行正确的次数是多少？

13. 维护性和可靠性

- 清单格式适于搞高可读性吗？
- 标号和子程序符合代码的逻辑意思吗？
- 对从外部接口采集的数据有确认吗？
- 遵循可靠性编程的要求吗？
- 是否存在内存泄露的问题？

F

附录F

软件测试术语中英文对照

Acceptance testing　验收测试

Accessibility 可接近性

Active or open 激活状态

Adaptability 适应性

Ad-hoc Test 随机测试

Architecture 体系结构

Audit　审核

Auditor　审核员

Auditor qualifications　审核员资格

Availability 可用性

Behavioral test　行为测试

Baseline 基线

Black-box test 黑盒测试

Bottom-up Integration 自底向上集成

Boundary condition　边界条件

Bug 缺陷

Bug crawl 缺陷评审会议

Build 软件构建包

Capability　能力

Capacity test 容量测试

Certification 认证

Change control 变更控制管理

CCB, Change Control Bard 变更控制委员会

Characteristic　特性

Close or inactive 关闭或非激活状态

Closure period 修复周期

Code audit 代码审计

Code Completed 代码完成

Code Freeze 代码冻结

Code inspection 代码审查

Code walk－through 代码走查

Cohesion 内聚度

Compatibility 兼容性

Compile 编译

Complexity 复杂性

Component testing 组件测试

Confirmation tests 确认测试

Configuration management 配置管理

Conformity 合格（符合）

Congruent 一致性

Continual improvement 持续改进

Corrective action 纠正措施

Coupling 耦合度

Coverage 覆盖率

Criteria 准则、指标

Critical bug 严重的缺陷

Customer satisfaction 顾客满意

C/S, Client/Server 客户端/服务器(结构)

Data dictionary 数据字典

Data structure 数据结构

Data flow testing 数据流测试

Debugging 调试

Defect 缺陷

Delivery 交付

Dependability 可信性

Design and development 设计与开发

Design specification 设计规格说明

Development life cycle 开发生存周期

Deviation permit 偏离许可

Distributed testing 分布式测试

Document 文件

Driver 驱动模块

Effectiveness 有效性

Efficiency 效率

Encapsulation 封装

Entry criteria 进入标准

Error seeding 错误播种

Error, faults and failures 错误、缺陷与失效

Escalate 向上呈交

Evaluation 评价 评估

Exit criteria 退出标准

Experience of quality 质量体验

Exploratory testing 探索性测试

Failure 失效

Fatal bug 致命的缺陷

Fault injection 错误注入

Fault of omission 遗漏缺陷

Feasible coverage 可行覆盖率

Feature 产品特性

Fidelity 逼真度

Field-reported bug 现场报告缺陷

First customer ship 首位客户送货

Fixed or Resolved 已修正状态

Flexibility 灵活性

FMEA 失效模型和效果分析

Functional Specification 产品功能规格说明书

Functional tests 功能测试

Functionality 功能性

GA 通用有效性

Grade 等级

Granularity 粒度

Ideal fault condition 理想缺陷条件

Identifier 标识符

Information 信息

Infrastructure 基础设施

Implementation requirement 实现需求

Input 输入

Inspection 检验

Integration testing 集成测试

Interoperability 互操作性

Isolation 隔离

Log file 记录文件

Maintainable 可维护性

Major bug 一般的缺陷

Management 管理

Management system 管理体系

Measurement control system 测量控制体系

Measurement process 测量过程

Metrological characteristic 计量特性

Metrological confirmation 计量确认

Metrological function 计量职能

Milestone 里程碑

Minor bug 微小的缺陷

Modified Top-down Integration 改进的自顶向下集成

Modularity 模块性

Module 模块

MTBF 失效平均时间

MTTR 平均维修时间

Necessity condition 必要性条件

Nest 嵌套

Objective evidence 客观证据

Organizational structure 组织结构

Orthogonal 正交

Output 输出

Peer review 同级评审

Performance Test 性能测试

Pilot testing 引导测试

Preventive action 预防措施

Priority 优先权

Procedure 程序 过程

Program Specification 概要说明

Process 过程

Product 产品

Project 项目

Qualification process 鉴定过程

Quality assurance 质量保证

Quality characteristic 质量特性

Quality control 质量控制

Quality improvement 质量改进

Quality management 质量管理

Quality manual 质量手册

Quality metric 质量度量

Quality objective 质量目标

Quality plan 质量计划

Quality planning 质量策划

Quality policy 质量方针

Record 记录

Recovery testing 恢复测试

Regression 回归

Regression tests 回归测试

regression test gap 回归测试间距

Release 产品发布

Reliability 可靠性

Reporting logs 报告日志

Requirement 要求

Reusability 可重用性 复用率

Review 评审

Risk 风险

Root cause 根本原因

Scalability 可扩展性

script 脚本

Security 安全性 保密性

Security testing 安全测试

Service manageability 可维护性

Severity 严重性

SDLC, Software Development Life Cycle 软件开发生命周期

Software development process 软件开发过程

Software engineering 软件工程

Specification (Spec) 规范

Stability 稳定性

Standard combining rules 标准组合规则

Stress testing 负载(压力)测试

structured Programming　结构化程序设计

Structural test　结构测试

Stub 桩模块

System testing　系统测试

Test automation 测试自动化

Test case　测试用例

Test case library　测试用例库

Test casually 随机测试

Test coverage　测试覆盖

Test environment　测试环境

Test phase　测试阶段

Test platform　测试平台

Test specification　测试规格说明

Test suite　测试包　测试套件

Test system　测试系统

Test to fail　基于失效的测试

Test to pass　基于通过的测试

Test tool　测试工具

Testability　可测试性

Tolerance Test 容错测试

Traceability　可追溯性

Unit testing　单元测试

Usability 可用性

Validation　确认

Verification　验证

Version 版本

White-box tests　白盒测试

Walk-through　走查

G

附录G

参考书目和资源

[1] 朱少民，软件测试方法和技术，清华大学出版社，2005.7

[2] 陈伟柱等译，单元测试之道，电子工业出版社，2006.3

[3] 韩柯等译，软件子系统测试，机械工业出版社，2003.9

[4] 朱少民，软件质量保证和管理，清华大学出版社，2007.1

[5] G.J.Myers, *The Art of Software Testing* (2nd Edition)

[6] Ron Patton, *Software Testing*, SAMS Publishing, 2001

[7] Mark Fewster & Dorothy Graham, *Software Test Automation*，Addison-Wesley, 1999

[8] Rex Black, *Managing the Testing Process* (2nd Edition), Jihn Wiley & Sons, Inc., 2002

[9] Robert Culbertson, Chris Brown, Gary Cobb, *Rapid Testing*, Prentuce Hall PTR, 2002

[10] Partha Kuchana, *Software Architecture Design Patterns in Java*, Auerbach

[11] Stephen T. Albin, *The Art of Software Architecture-Design Methods and Techniques*, John Wiley & sons, Inc

[12] John, Rings, Raymond, *C++ Code Standard & Practice*

[13] http://blog.csdn.net/KerryZhu

[14] http://www.openqa.org/

[15] http://www.opentest.net/softtest/mbug.htm

[16] http://www.ibm.com/developerworks/cn/

[17] http://www.eclipse.org/tptp/

[18] http://testingfaqs.org/

[19] http://sourceforge.net/

[20] http://www.opensourcetesting.org

[21] http://www.pushtotest.com/

[22] http://www.aptest.com/webresources.html

[23] http://www.sun.com/java/