第 1 章　软件测试概述

知识目标

◎ 软件开发与软件测试
◎ 软件测试与 CMMI
◎ 缺陷管理、测试用例和测试环境
◎ 软件测试职业

技能目标

◎ 了解软件测试的定义和重要性
◎ 理解软件测试技术的概念及 CMMI 模型的作用
◎ 掌握 BUG 的定义、分类、处理流程，以及缺陷报告的撰写形式
◎ 掌握测试用例的定义、评价标准及其设计的基本原则
◎ 掌握测试环境的定义、要素、规划及维护和处理
◎ 了解国内外软件测试的现状、软件测试人员的构成及素质要求

软件测试就是利用测试工具按照测试方案和流程对产品进行功能和性能测试，或是根据需要编写不同的测试工具，设计和维护测试系统，对测试方案可能出现的问题进行分析和评估。执行测试用例后，需要跟踪故障，以确保开发的产品适合需求。软件测试是信息系统开发中不可缺少的一个重要步骤，随着软件变得日益复杂，软件测试也变得越来越重要。

1.1　软件开发与软件测试

软件开发与软件测试是两个相互联系的概念，软件测试贯穿于软件开发的整个生命周期。

1.1.1　软件开发

1. 软件开发的定义

软件开发是一个把用户需要转化为软件需求，把软件需求转化为软件设计，用软件代码来实现软件设计，对软件代码进行测试，并确认它可以投入运行使用的过程。在这个过程中的每一阶段，都包含有相应的文档编制工作。

2. 软件开发过程

1)需求分析

软件需求分析就是回答系统“做什么”的问题。它是一个对用户的需求进行去粗取精、去伪存真,正确理解,然后把它用软件工程开发语言(形式功能规约,即需求规格说明书)表达出来的过程。本阶段的基本任务是和用户一起确定要解决的问题,建立软件的逻辑模型,编写需求规格说明书文档并最终得到用户的认可。需求分析的主要方法有结构化分析方法、数据流程图和数据字典等。本阶段的工作是根据需求规格说明书的要求,设计建立相应的软件系统的体系结构,并将整个系统分解成若干个子系统或模块,定义子系统或模块间的接口关系,对各子系统进行具体设计定义,编写软件概要设计说明书和详细设计说明书、数据库或数据结构设计说明书、组装测试计划等。

2)设计

软件设计可以分为概要设计和详细设计两个阶段。实际上软件设计的主要任务就是将软件分解成模块,即能实现某个功能的数据和程序说明、可执行程序的程序单元。这些程序单元可以是一个函数、过程、子程序,也可以是可组合、可分解和可更换的功能单元。

概要设计就是结构设计,其主要目的是给出软件的模块结构,用软件结构图表示。

详细设计的首要任务是设计模块的程序流程、算法和数据结构,次要任务是设计数据库,常用方法是结构化程序设计方法。

3)编码

软件编码是指把软件设计转换成计算机可以接受的程序,即写成以某一程序设计语言表示的“源程序清单”。充分了解软件开发语言、工具的特性和编程风格,有助于开发工具的选择以及保证软件产品的开发质量。

当前软件开发中除专用场合外,已经很少使用 20 世纪 80 年代流行的计算机高级语言了,取而代之的是面向对象的开发语言,而且面向对象的开发语言和开发环境大都合为一体,这就大大提高了开发的效率。

4)测试

软件测试的目的是以较小的代价发现尽可能多的错误。实现这个目标的关键在于设计一套出色的测试用例(测试用例由测试数据和预期的输出结果组成)。如何才能设计出一套出色的测试用例? 关键在于理解测试方法。不同的测试方法有不同的测试用例设计方法。常用的两种测试方法是白盒法和黑盒法。

白盒法的测试对象是源程序,依据程序内部的逻辑结构来发现软件的编程错误、结构错误和数据错误。结构错误包括逻辑、数据流、初始化等的错误。白盒法用例设计的关键是以较少的用例覆盖尽可能多的内部程序逻辑结果。

黑盒法依据软件的功能或软件行为描述来发现软件的接口、功能和结构错误。其中接口错误包括内部/外部接口、资源管理、集成化以及系统错误。黑盒法用例设计的关键是以较少的用例覆盖模块输出和输入接口。

5)维护

维护是指在已完成对软件的研制(需求分析、设计、编码和测试)工作并交付使用以后,对软件产品所展开的一些软件工程活动,即根据软件运行的情况,对软件进行适当修改,以适应新的要求,以及纠正运行中发现的错误,最后,还需编写软件问题报告、软件修改报告。

一个中等规模的软件,如果研制阶段需要 1～2 年的时间,在它投入使用以后,其运行或

工作时间可能持续 5～10 年，那么它的维护阶段即在运行的这 5～10 年期间。这段时间，软件维护人员需要着手解决研制阶段所遗留的各种问题，同时还要解决某些维护工作本身所特有的问题。做好软件维护工作，不仅能排除障碍，使软件能正常工作，而且还可以扩展功能，提高性能，为用户带来明显的经济效益。然而遗憾的是，现在人们对软件维护工作的重视往往远不如对软件研制工作的重视。而事实上，和软件研制工作相比，软件维护的工作量和成本都要大得多。

在实际开发过程中，软件开发并不是从第一步按顺序进行到最后一步，而是在任何阶段，在进入下一阶段前一般都有一步或几步的回溯。测试过程中发现的问题可能要求修改设计，用户可能会提出一些需求来修改需求说明书等。

1.1.2　软件测试

1. 软件测试的定义

软件测试是指使用人工和自动手段来运行或测试某个系统的过程，目的在于检验其是否满足规定的需要或弄清楚预期结果与实际结果之间的差别。

2. 软件测试的重要性

在软件业比较发达的国家，软件测试不仅早已成为软件开发的一个有机组成部分，而且在整个软件开发的系统工程中占据着相当大的比重，软件测试是十分重要的。具体而言，软件测试的重要性体现在以下几个方面。

1）寻找软件错误，以便进行修正

这是保证软件质量的重要一环，也是测试人员需要持续不断做的工作，特别是通过回归测试可以防止无意识的行为引入一些将来可能出现的错误。

2）验证软件是否符合要求

从用户的角度出发，用户希望通过测试来核实开发方交付的软件是否符合自己的真实需求，系统运行在真实的应用环境下是否存在关键功能或关键性能上的缺陷，这主要是指 β 测试。在测试过程中，用户的目标是全方位使用系统，尽可能多地暴露软件中的问题，以考虑是否接受该产品。实施良好的 β 测试非常有利于产品的成功发布。

3）证明软件符合要求，是可用的

从开发方的角度出发，开发方希望通过测试向用户证明其所开发的软件正确地实现了用户需求，树立用户对软件质量的信心，为用户选择与接受软件提供有力的依据，这主要是指 α 测试。这时，开发方的目标是确保程序不要出大的问题，避免延误产品的正常交付。

4）指导软件的开发过程

就宏观而言，不同的软件开发过程，有不同的测试模型与之对应，以此能够更好地指导开发与测试实践；就微观来看，测试可以保证对需求和设计的理解与表达的正确性、实现的正确性及运行的正确性，无论哪个环节存在问题，都将在测试中表现出来。通过分析缺陷的分布和产生原因可以帮助测试人员发现当前软件开发的缺陷，有助于过程改进，通过分析整理测试结果，可以进一步完善软件。

5）提供软件的相关特征

测试是对软件质量的度量和评估。测试可提供测试数据及结果来对质量进行细化和量化。

软件开发与软件测试是相互联系的，软件开发过程中不能缺少软件测试这一环节，软件测试应当伴随着软件开发的整个过程。

1.2 软件测试与 CMMI

在软件开发的瀑布模型中，测试是一个非常重要的工程阶段。从保证软件质量的角度来说，软件测试是软件质量保证工程的一个重要组成部分，也是最重要的保证手段。为了保证所提交的软件产品能够满足客户的需求，以及在使用中的可靠性，就必须对所开发的软件产品进行系统而全面的测试。基于这一需求，软件测试作为软件开发过程中的一个重要阶段，受到了软件开发组织的普遍重视，并形成了一整套比较成熟的测试理论和技术方法。

然而，随着软件开发技术的不断发展，以及软件系统的规模和复杂性的不断增加，传统的软件测试理论和技术已经不能很好地满足开发组织在产品质量、开发成本以及研制周期等方面的需求。本节主要从软件测试的组织和管理角度，阐述 CMMI 模型规范对软件测试技术的应用和扩充，对于软件开发组织如何发展和完善软件开发中的测试工作进行初步探索。

1.2.1 传统的软件测试技术和测试过程模型

传统的软件测试只是作为软件开发过程中的一个特定阶段，并且只针对软件成品进行测试。如图 1-1 所示，在瀑布式开发过程模型中，测试是在编码完成之后软件产品交付运行之前的一个工程阶段，所有的审查和评审活动都是针对成型软件产品而开展的。这样的软件测试主要关注的是对软件的验收测试，在一定程度上保证了所提交的软件产品的质量。但是，全面质量管理的理论认为，软件的高质量是开发和设计出来的，而不是测试出来的，因此，仅仅依靠对软件成品进行测试的质量保证活动显然是远远不够的。随着软件开发过程模型和开发技术的不断发展，软件测试理论和技术也应该得到相应的发展。

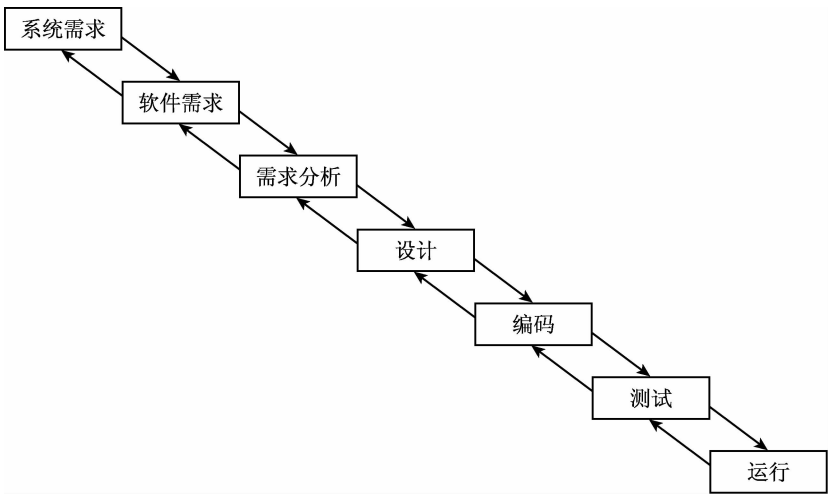

图 1-1 软件测试在软件开发过程的瀑布模型描述中所处的地位

随着全面质量管理思想在软件开发领域的应用，软件测试也由最初的只针对软件成品扩展到了针对软件半成品和过程产品的全过程测试。这是对软件测试的一种扩充。扩充后的软件测试贯穿了软件开发的全过程，包括软件需求分析、软件概要设计、软件详细设计、编码、集成、验收等各个工程阶段。相应地，各阶段所开展的测试分别为需求测试、架构测试、详细设计测试、单元测试、集成测试以及验收测试等。这样的软件测试涵盖了软件开发的整个工程过程，对于识别和控制软件缺陷、提高软件质量有很明显的效果。

从本质上来说，无论是传统的软件验收测试，还是面向整个开发过程的全过程软件测试，其所针对的测试对象都是软件成品、半成品或者过程工作产品，其所报告的测试结果也只是为了识别出现在阶段产品的缺陷，并加以纠正，以支持下一阶段的开发工作。从软件开发组织的长远发展来看，仅仅做到这些是不够的。软件测试作为软件质量保证的一种重要手段，不仅要能够识别软件产品的缺陷并加以改正，还应该在软件测试中结合统计技术方法，给出对软件开发过程的度量，从而支持组织对软件开发过程的评估和改进。由美国国防部和卡耐基—梅隆大学的软件工程研究所联合开发的 CMMI 模型，正是从软件过程改进和评估的角度出发，对软件开发中的测试技术给出了充分的支持和扩充。

1.2.2　CMMI 模型对软件测试的支持和扩充

CMMI 模型在开发过程中注重对过程和产品的度量，以量化的形式提供对管理过程的支持，以及对过程进行相应的评估和改进。这实际上就是对软件测试技术的一种应用和扩充。CMMI 模型将测量和分析作为一个单独的过程域，充分体现了对开发过程中的测量技术的重视，该过程域的目的就是开发和维持度量能力，以便对管理信息的需要提供支持。

测量和分析过程域共有 3 个目标，其中两个为特定目标，一个为共性目标。

第一个目标是协调测量和分析活动。为实现这一目标，模型中给出了 4 个方面的特定实践，它们分别是：确定测量对象，建立测量目标；详细说明度量值，以处理测量目标；规定数据收集和存储规程，说明如何获得并存储测量数据；规定分析规程，说明如何对度量数据进行分析和报告，并且安排优先顺序。该目标中所针对的测量对象既包括组织所开发出的软件成品、半成品以及过程产品，也包括对开发过程本身的度量。因此，在测量和分析过程中不仅要用到传统软件测试中的一些技术和方法，还需要在测量过程中引入统计过程控制等理论方法，提供对过程度量的改进和支持。

第二个目标是提供度量结果，以便处理信息需要和目标。为实现这一目标，模型中也给出了以下 4 个方面的特定实践：收集度量数据，即获得制定的度量数据；分析并解释度量数据；管理并存储度量数据、度量规范和分析结果；通报分析结果，向所有的相关人员报告测量和分析活动的结果。在这一目标中，主要关注的是对测量结果的分析和使用。而在传统的软件测试中，只要产品通过了需求方的验收，达到了合同要求，开发组织一般也就不再重视对软件测试结果数据的管理和使用，从过程改进的角度来说，这是很不科学的。CMMI 对集成化过程进行了改进和评估，提出了建立开发过程数据库的思想，以作为开发组织进行过程改进的基础。而建立过程数据库的过程，实际上也就是对测试和度量数据的积累和存储过程。从这一点来说，在开发过程中开展软件测试以及针对开发过程的度量，是建立过程数据库的必要步骤。

第三个目标是共性目标，即将测量和分析活动制度化为可管理的过程。这一目标主要

关注的是对软件测试和过程度量活动的管理以及制度化。针对这一共性目标，CMMI 模型从 4 个不同方面给出了 10 个共性实践。首先，作为执行测量和分析活动的承诺，要求组织建立方针，为策划和执行“测量和分析”过程提供组织级的支持；其次，在执行能力方面，组织应该制订测量和分析过程计划，提供必要的资源，分配相应的责任，并且对相关人员进行培训；第三，为了指导该过程的实施，组织应该将测量和分析过程指定的工作产品置于配置管理的适当层次，确定与过程相关的人员并使之介入，同时还要对测量和分析过程进行监督和控制；最后，作为对测量和分析活动的验证实施，组织应该客观评价测量和分析过程以及过程的工作产品和服务的遵循情况，同时，由高层管理者审查测量和分析过程的活动、状态和分析结果，并解决相应的问题。这一共性目标的实现，实际上就是把测量和分析活动制度化为一种组织级的行为，在整个组织的范围内加强了对软件测试和过程度量活动的组织和管理工作。

从以上分析可以看出，CMMI 模型主要从以下 3 个方面扩充了传统的软件测试技术。

(1)从单纯的对软件产品的测试活动，扩展为软件产品的测试和开发过程的度量。

这一方面主要体现在过程度量对软件测试的依赖和应用。对开发过程进行度量，需要利用对软件成品、半成品以及工作产品的测试结果，从而建立对开发过程的软件产品缺陷可跟踪性。从这一点来说，对开发过程的度量，实际上也就是对软件产品的测试活动的扩展，其与传统的软件测试的不同之处就在于它关注对软件测试结果数据的分析和利用，将测试数据有效地转换成能够标识过程缺陷的统计数据。

(2)软件测试由原来的事后测试行为发展为全过程测试和分析，成为一种缺陷预防的有效方式。

统计技术方法的应用，将传统的软件测试活动扩展为一种全过程测试行为。从质量工程的角度来说，这是一种质量保证思想的转变。传统的软件测试，只针对软件成品而开展，找到缺陷之后再加以改正和修补，这是一种“亡羊补牢”式的质量管理方式；而针对开发全过程所开展的软件测试和过程度量，则根据测试数据的统计分析结果来判断软件产品的未来质量趋势，并提前予以预防和控制，属于一种“防患于未然”式的质量管理方式。与传统的软件测试相比，全过程测试不仅可以有效降低产品的质量风险，而且还可以提前对软件产品缺陷进行规避，这不仅缩短了缺陷的反馈周期和整个项目的开发周期，而且也大大降低了对软件产品的修改和维护费用，对于软件产品的整个生命周期都有很重大的意义。

(3)软件测试与开发过程的其他阶段不再是串行工作方式，而是与整个开发过程并行进行。

与瀑布模型相比，CMMI 模型中所描述的软件测试和过程度量工作与整个开发过程是并行进行的，是一种基于并行工程的测试和度量行为。基于并行工程开展的软件测试活动，存在于软件生命周期的各个阶段，其基本特点是以质量保证和客户要求为核心开展对软件产品和开发过程的测试和度量，力争将缺陷控制在软件开发过程的每一个阶段，从而可以有效缩短开发周期，降低质量风险，并且可以及时吸取经验教训，提供对过程改进的支持。这也体现了 CMMI 模型对并行工程思想的一种支持和应用。

除了测量和分析过程域之外，CMMI 4 中的量化过程管理过程域也是对软件测试和过程度量技术的一种更高层次上的应用。在该过程域中，测试和度量已经不仅仅是一种被管理的过程，而且其本身也成为了一种有效的辅助管理手段，从定量化的角度对软件开发过程的管理和组织活动给出了支持。开发过程管理从定性到定量的转化，是 CMMI 集成化过程

改进所追求的目标之一，也是开发组织一直追求的一种更高水平的管理方式。因此，随着软件开发组织过程能力的不断提高，软件测试和度量技术也将会得到不断的发展和完善。

1.3　缺 陷 管 理

缺陷管理(defect management)是在软件生命周期中获取、管理、沟通任何变更请求的过程(从变更的建议到变更的解决)。缺陷管理可以确保问题(如需求或者缺陷)被跟踪管理而不丢失。在程序中存在的软件缺陷(如文档缺陷、代码缺陷、测试缺陷、过程缺陷)导致系统或部件不能实现其功能，引起系统的失效。对软件缺陷测试和管理是不可缺少的。

1.3.1　BUG 的定义与分类

缺陷(BUG)是指程序中存在的错误，如语法错误、拼写错误或者一个不正确的程序语句。缺陷可能出现在设计中，或是在需求分析、规格说明或其他文档中。软件缺陷导致系统或部件不能实现其功能，引起系统的失效。缺陷定义如下：

- 软件没有实现产品说明书标明的功能。
- 程序中存在语法错误。
- 程序中存在拼写错误。
- 程序中存在不正确的程序语句。
- 软件出现了与产品说明书不一致的表现。
- 软件功能超出产品说明书的范围。
- 软件没有达到用户期望的目标。
- 测试人员或用户认为软件的易用性差。

从不同的分类角度，可以将 BUG 分为多种类型。

1)按严重程度划分

严重程度是指 BUG 对软件质量的破坏程度，即此 BUG 的存在将对软件的功能和性能产生怎样的影响。按照严重程度由高到低的顺序可以将其分成 5 个等级：系统崩溃、严重、一般、次要、建议。需要说明的是，在具体的项目中，要根据实际情况来划分等级，不一定是 5 个等级，如果 BUG 数比较少，就可以划分为 3 个等级：严重、一般、次要。一般的缺陷管理工具会自动给出一个默认的 BUG 严重程度划分。

2)按优先级划分

优先级是指处理和修复软件缺陷的先后顺序的指标，即哪些缺陷需要优先修正，哪些缺陷可以稍后修正。按照优先级由高到低的顺序可以将其分成 3 个等级：高(high)、中(middle)、低(low)。其中高优先级的 BUG 是应该立即修复的 BUG，中优先级的 BUG 是应该在产品发布之前修复的 BUG，低优先级的 BUG 是指如果时间允许应该修复的 BUG 或是可以暂时存在的 BUG。和 BUG 的严重程度的划分一样，优先级的划分方法也不是绝对的，需要根据实际情况灵活划分。

3)按照测试种类划分

按照测试种类可以将 BUG 分为逻辑功能类(function)、性能类(performance)、界面类(UI)、易用性类(usability)、兼容性类(compatibility)等。可以这样理解：有一种测试方法，

就有一种对应的 BUG 种类。当然这里列举的都是黑盒测试的测试方法，还有白盒测试的测试方法也可以作为 BUG 的种类，如边界值类、内存溢出类、逻辑驱动类等。

4)按功能模块划分

一般的软件产品都是分为若干个功能模块的，如在 Word 2000 中有文件、编辑视图、帮助等功能模块。80%的缺陷集中在 20%的模块里，测试时，可以统计一下 BUG 主要集中在哪些模块里面，以便投入重点精力去测试。

5)按 BUG 的生命周期划分

可以把 BUG 看做是一个有生命的“小虫子”，每个 BUG 都有其生命周期，可以这样划分：新建(new)、确认(confirmed)、解决(fixed)、关闭(closed)、重新打开(reopen)。

1.3.2 缺陷报告

在实际项目中，需要根据固定的模板提交 BUG，这个模板可以是 Word、Excel 或是缺陷管理工具自带的模板，其基本形式如表 1-1 所示。

表 1-1 缺陷报告模板

<table>
<tr><td colspan="2">缺陷记录</td><td>编号：0001</td></tr>
<tr><td>软件名称：××字处理软件</td><td>模块名：帮助</td><td>版本号：2000</td></tr>
<tr><td>严重程度：次要</td><td>优先级：低</td><td>状态：新建</td></tr>
<tr><td colspan="3">测试人员：×××</td></tr>
<tr><td colspan="3">分配给：</td></tr>
<tr><td colspan="3">日期：2010-08-01</td></tr>
<tr><td colspan="2">硬件平台：
CPU-P4 2.0G
RAM-521M</td><td>操作系统：
Windows 2000 Professional SP4</td></tr>
<tr><td colspan="3">缺陷概述：“关于”对话框的界面中英文混合</td></tr>
<tr><td colspan="3">详细描述：
1. 打开××字处理软件，执行“帮助”→“关于”菜单命令
2. 在“关于”对话框中存在中英文混合的现象：3 个英文单词</td></tr>
<tr><td colspan="3">附件：可附图</td></tr>
<tr><td>解决信息</td><td colspan="2">验证信息</td></tr>
<tr><td>解决人：××程序员</td><td colspan="2">验证人：××测试员</td></tr>
<tr><td>解决版本：1.0</td><td colspan="2">验证版本：2.0</td></tr>
<tr><td>解决时间：2010-08-01</td><td colspan="2">验证时间：2010-08-02</td></tr>
<tr><td>备注：已解决</td><td colspan="2">备注：已通过验证</td></tr>
<tr><td>……</td><td colspan="2">……</td></tr>
</table>

下面介绍提交缺陷报告的一些注意事项。

1)确保重现 BUG

如果无法重现测试人员所提交的 BUG,将会影响开发人员的开发效率,也会影响测试人员自身的声誉,因此在提交 BUG 之前一定要确保 BUG 能够重现。对于严重程度较高的 BUG,一般要重复测试两次以上;对于即时产生的 BUG,要在其他机器上测试一下,看看是否是自己机器的原因。

2)要用最少且必要的步骤描述 BUG

使用最少且必要的步骤,是为了节省开发人员定位问题的时间,例如:

(1)运行客户端。

(2)为输入新的条目查询数据库。

(3)打开一个浏览器。

(4)在 www. yahoo. com 上浏览新闻。

(5)关闭浏览器。

(6)选择一个条目。

(7)把种类从“鲜花”更改为“水果”。

(8)使数据库服务器脱机。

(9)尝试保存记录。

(10)收到一个超时的错误。

(11)退出客户端。

这个缺陷报告的详细描述中共有 11 个步骤,其中步骤(3)、(4)、(5)、(7)是多余的,应该去掉。

3)简洁、准确、完整

测试人员在提交缺陷报告的时候,要站在开发人员的角度上思考问题,要确保开发人员拿到缺陷报告后马上就能够定位问题,而不会产生理解上的歧义。下面是几个基本的要求:

- 缺陷概述:简洁、准确、完整,揭示错误实质,最好用陈述句,一般不超过 15 个字。
- 详细描述:简洁、准确、完整,保证快速准确地描述错误。“简洁”即没有多余的步骤,“准确”即步骤正确,“完整”即没有缺漏。
- 尽量使用业界惯用的表达术语和表达方法。
- 检查拼写和语法错误。

4)一个 BUG 一个报告

有的测试人员喜欢在一个缺陷报告里面提交多个 BUG,这种习惯不值得提倡,原因有以下两点:

- 不便于分配 BUG。如一个缺陷报告里面有两个 BUG,分别属于甲和乙两个开发人员,那么应将该报告提交给甲还是乙?
- 不便于回归测试。测试人员在回归测试的时候需要关闭 BUG,如一个缺陷报告里有两个 BUG,BUG1 已经解决,BUG2 还没有解决,那么缺陷报告是否应该关闭?

总之,尽量使一个缺陷报告只包含一个 BUG。

1.3.3　BUG 的处理流程

很多公司招聘测试工程师经常考的一道题目就是画出 BUG 的处理流程,以考查应聘者

是否具有一定的缺陷管理经验。

实际上,不同公司的BUG处理流程一般是不同的,同一公司不同项目的BUG处理流程也不尽相同。下面给出某公司BUG处理的大致步骤,读者可以结合自己的实际情况灵活修改。

1)发布BUG

主流程:

(1)测试人员测试功能,并发现BUG。

(2)测试人员将BUG录入到VSTS对应的团队项目下,设置BUG状态为"已建议",并认真填写发现环境、发现途径、症状和重现步骤,如有必要还可添加BUG的截图。其中重现步骤必须详细到说明直接引发BUG的每个用户操作(被操作的控件、被选择的数据等)。

(3)测试人员确定BUG修改人,通常情况下,BUG所属功能的开发人员就是BUG修改人。

(4)测试人员在VSTS中指派BUG的修改人。

(5)如BUG修改人在现场实施,测试人员将BUG记录在"质量问题反馈表"中,并在每天下班前用邮件方式通知该修改人,如必要可用其他方式及时通知。

2)接收BUG

主流程:

(1)开发人员在每天上班的第一时间登录VSTS或接收邮件,查看需要修改的BUG。

(2)开发人员确定BUG的处理顺序,通常情况下,优先处理高严重级别和高优先级别的BUG。

(3)开发人员确定能够在规定时间内执行修改的BUG,将BUG状态改为"活动"。

3)修改BUG

主流程:

(1)开发人员确定在开发环境下可重现BUG。

(2)开发人员修改BUG。

(3)开发人员修改BUG成功,并测试修改后的功能,在确定没有因修改而产生其他BUG之后,将BUG状态改为"已解决",原因改为"已修复"。如在修改过程中遇他人大力协助,可在建议的修改方案中进行说明。

(4)位于公司的开发人员在每天的16:30之前将修改成功的代码嵌入VSTS。位于现场的开发人员在每天的16:00之前将修正过的"质量问题反馈表"和修改成功的源代码发给相关测试人员。

(5)位于公司的测试人员在16:05接收现场开发人员发回的"质量问题反馈表"和源代码,并在16:30之前更新VSTS中相关的BUG信息和源代码。

1.4 测 试 用 例

1.4.1 测试用例的定义

测试用例(test case,TC)简单来讲,是指执行条件和预期结果的集合;完整来讲,是针对

要测试的内容所确定的一组输入信息，是为达到最佳的测试效果或高效地揭露隐藏的错误而精心设计的少量测试数据。

统一软件开发过程(rational unified process，RUP)认为测试用例是用来验证系统实际做了什么的方式，因此，测试用例必须可以按照要求来跟踪和维护。

IEEE Std 610—1990 给出的定义为：测试用例是一组测试输入、执行条件和预期结果的集合，目的是要满足一个特定的目标，如执行一条特定的程序路径来检验是否符合一个特定的需求。

从以上定义来看，测试用例设计的核心有两方面：一是要测试的内容，即与测试用例相对应的测试需求；二是输入信息，即按照怎样的操作步骤，对系统输入哪些必要的数据。测试用例设计的难点在于如何通过少量测试数据来有效地揭示软件缺陷。

测试用例可以用一个简单的公式来表示，即：

测试用例＝输入＋输出＋测试环境

其中，输入是指测试数据和操作步骤，输出是指系统的预期结果，测试环境是指系统环境设置，包括软件环境、硬件环境和相关数据，有时还包括网络环境。

1.4.2　测试用例的评价标准

根据多年的实践经验，测试用例的标准不能局限于一个层次，因为测试用例设计类似于软件设计，软件设计有概要设计(结构设计/架构设计)和详细设计两种，所以对于测试用例的质量标准，也应分为两个层次来考虑高层次和低层次。高层次满足某一个测试目标或测试任务，从整体测试用例看，需衡量一组测试用例的结构、设计思路和覆盖率等指标。低层次从单个测试用例看，需衡量测试用例描述的规范性、可理解性和可维护性等指标。

1. 高层次标准

高层次(high-level)标准是从满足某一个特定的测试目标出发来进行定义，分析一组测试用例的设计思路、设计方法和策略，包括测试用例的层次、结构等。从高层次看，测试用例设计的关键点是：始终从客户需求的角度出发，始终围绕测试的覆盖率和执行效率不断思考，最终通过有效的技术方法完成测试用例的设计。

对于一整套的测试用例组(集合)，可定义如下的质量标准：

(1)测试用例的目标清楚，并能满足软件质量的各个方面的需求，包括功能测试、性能测试、安全性测试、故障转移测试、负载测试等。

(2)设计思路正确、清晰。例如，通过序列图、状态图、工作流程图、数据流程图等来描述待测试的功能特性或非功能特性。

(3)在组织和分类上，测试用例层次清楚、结构合理。测试用例的层次与产品特性的结构/层次相一致，或者与测试的目标/子目标的分类/层次相一致，并具有合理的优先级或执行顺序。

(4)测试用例覆盖所有测试点，覆盖所有已知的用户使用场景，也就是说，每个测试点都有相应数量的测试用例来覆盖，可将各种用户使用场景通过矩阵或因果图等方式列出来，找到相对应的测试用例。

(5)测试手段的区别对待。在设计测试用例时，要全面考量测试的手段，哪些方面可以通过工具测试，哪些方面不得不用手工测试，对不同手段的测试用例应区别对待。

(6)有充分的负面测试。作为测试用例,不仅要测试正确的输入和操作,还要测试各种各样的例外情况,如边界条件、不正确的操作、错误的数据输入等。

(7)没有重复、冗余的测试用例,满足相应的行业标准等。

2.低层次标准

低层次(low-level)标准考查单个测试用例是否满足测试的需求,是否能被更有效地执行。测试用例设计的结果就是交付测试用例,使测试用例被执行,所以除了覆盖率,执行的效率也是测试用例的一个重要属性。测试用例越清楚,也就越容易被理解和执行。执行效率越高就说明测试用例越好。

对于具体的某个测试用例,不妨可定义如下的质量标准:

(1)测试用例的出发点是发现缺陷,即单个测试用例在“暴露缺陷”上具有较高的可能性。

(2)测试用例的单一性。一个测试用例面向一个测试点,不要将许多测试点揉在一起。例如,通过一个测试用例只能发现 1～2 个缺陷,而不能发现 5～10 个缺陷甚至更多的缺陷。

(3)符合测试用例设计规范或测试用例模板。

(4)描述清楚,包括特定的场合、特定的对象和特定的术语,没有含糊的概念和一般性的描述。例如,测试用例名称为“登录功能使用正常”就是一个描述不清楚的例子,而 “登录功能中用户名大小写不敏感性验证”、“登录功能中用户名唯一性验证”和“用户账号被锁定后再进行登录操作”等样式的描述就比较好。

(5)操作步骤的准确性。按照步骤操作得到唯一的测试结果。

(6)操作步骤的简单性。操作步骤不应该太复杂,过于复杂的操作步骤意味着测试用例需要被分解为多个测试用例或者分解为多个环节进行验证。

(7)所期望的测试结果是可验证的,即能迅速、明确地判断测试的实际结果是否与所期望的结果相同或相匹配。例如,在测试用例中描述期望结果为“登录成功”,这实际上是不可验证的。要使这个期望结果具有可验证性,就应该这样描述所期望的结果——“‘退出(logout)’按钮出现”。

(8)测试环境的正确性,测试数据的充分性。

(9)前提条件、依赖性被完全识别出来。

这样,测试用例就具有很好的可理解性和可维护性,可以提高测试执行的效率,并能保证不同的人员执行相同的用例能获得统一的结果。步骤的准确性和期望结果的可验证性,非常有助于测试执行的自动化实现,也只有实现了测试执行的自动化,测试执行的效率才是最高的,测试人员才有更多的时间去思考、去设计更优秀的测试用例,进入良性循环,不断地提升测试的质量和效率。

1.4.3 测试用例设计的基本原则

对于不同类别的软件,测试用例的设计重点是不同的。例如,公司管理软件的测试通常需要将测试数据和测试脚本从测试用例中划分出来。测试用例更趋于一种针对软件产品的功能、业务规则和业务处理所设计的测试方案,对软件的每个特定功能或运行操作路径的测试构成了不同的测试用例。

一般情况下,测试用例设计的基本原则有 3 条。

(1)测试用例的代表性。能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的以及极限的输入数据、操作和环境设置等。

(2)测试结果的可判定性。这是指测试执行结果的正确性是可判定的,每一个测试用例都应有相应明确的预期结果,而不应存在二义性,否则将难以判断系统是否运行正常。

(3)测试结果的可再现性。这是指对同样的测试用例,系统的执行结果应当相同。测试结果确保缺陷的重现,为缺陷的快速修复奠定基础。

而以上 3 条原则中,最难保证的就是测试用例的代表性,这也是设计测试用例时最为关键的内容,即如何确定测试用例中输入的数据集合。一般地,在有多个输入条件的情况下,应首先分析出哪些是核心的输入条件,针对每个核心的输入条件,其数据大致可分为以下 3 类。

1)正常数据

符合需求规格说明,合理的、有效的输入数据。例如,某个输入的有效取值范围内的数据。

2)边界数据

介于正常数据与错误数据之间的临界数据,边界数据可能是有效的输入数据,也可能是无效的输入数据,这要根据需求规格说明的具体规定而定。

3)错误数据

不符合需求规格说明、无意义的、无效的输入数据。需要注意的是,对于无效输入有两种情况:一是满足所有输入的数据类型要求,但不在有效取值范围内;二是部分输入不满足输入的数据类型要求。更严格地来讲,还应考虑缺少输入条件的情况。

测试数据就是从以上 3 类数据中产生的。

1.4.4　测试用例模板

一个优秀的测试用例,应该包含以下信息:

(1)软件或项目的名称。

(2)软件或项目的版本(内部版本号)。

(3)功能模块名。

(4)测试用例的简单描述,即该用例执行的目的或方法。

(5)测试用例的参考信息(便于跟踪和参考)。

(6)本测试用例与其他测试用例间的依赖关系。

(7)本用例的前置条件,即执行本用例必须要满足的条件,如对数据库的访问权限。

(8)用例的编号,如可以是“软件名称简写_功能块简写_NO.”。

(9)步骤号、操作步骤描述、测试数据描述。

(10)预期结果(这是最重要的)和实际结果(如果有 BUG 管理工具,这条可以省略)。

(11)开发人员(必须有)和测试人员(可有可无)。

(12)测试执行日期。

如图 1-2 所示的模板就是一个测试用例模板。

项目/软件	技术出口合同网络申领系统		项目版本	1. 0. 25		
功能模块名	Login		编 制 人	×××		
用例编号	TC-TEP_Login_1		测试执行时间	2010. 10. 12		
相关用例	无					
功能特性	用户身份验证					
测试目的	验证是否输入合法的信息，允许合法登录，阻止非法登录					
预置条件	无		特殊规程说明	对数据库访问权限		
参考信息	需求说明中关于“登录”的说明					
测试数据	用户名=yiyh , 密码=1					
操作步骤	操作描述		数 据	期望结果	实际结果	测试状态
1	输入用户名称，单击“登录”按钮		用户名=yiyh，密码为空	显示警告信息“请输入用户名和密码!”		
2	输入密码，单击“登录”按钮		用户名为空，密码=1	显示警告信息“请输入用户名和密码!”		
----------->>>						
测试人员		开发人员			项目负责人	

图 1-2 测试用例模板

1.5 测 试 环 境

1.5.1 测试环境的定义

软件的测试环境就是软件运行的平台，即软件、硬件、网络和历史数据的集合，如下式所示：

测试环境＝软件＋硬件＋网络＋历史数据

- 硬件：主要包括 PC、笔记本电脑、服务器、各种 PDA 终端等。不同的机器类型、不同的配置，会导致程序的反应速度不一样，所以在测试一款软件时一定要考虑硬件配置的因素。

• 软件：这里主要指的是软件运行的操作系统。许多软件在使用时，有兼容性的问题。

• 网络：主要针对的是 C/S 结构和 B/S 结构的软件。

• 历史数据：指测试用例执行所需初始化的各项数据。

以上就是搭建软件测试环境所需要考虑的 4 个方面。作为一名合格的软件测试工程师，不仅要熟悉软件的知识，也要了解硬件和网络的相关知识。

1.5.2　测试环境的要素

良好的测试环境应具备以下 3 个要素。

1）良好的测试模型

良好的测试模型有助于高效地发现缺陷，它并不仅仅包含一系列测试方法，更重要的是，它是在长期实践中积累下来的一些历史数据，包括有关某类软件的缺陷分类规律、有关项目小组历次测试的过程数据等。这些历史数据可以相应地反映出针对某类软件的测试关注点、项目小组的开发与测试效率等内容，有助于提高后续类似产品的开发与测试的效率。

2）多样化的系统配置

测试环境在很大程度上应该是用户的真实使用环境，或至少应搭建模拟的使用环境，使之尽量逼近软件的真实运行环境。为此，应测试在各种系统配置条件下软件的运行情况，特别是通用型软件系统的测试。

3）熟练使用工具的测试人员

在系统测试尤其是性能测试环节，通常需要有自动化测试工具的支持，否则将无法模拟或再现系统运行环境；但若仅有测试工具，没有熟悉如何使用工具的测试人员，或缺乏了解被测软件系统特性的测试人员，是无法发挥自动化测试工具的巨大优势的。

1.5.3　测试环境的规划

一般情况下，单元测试和集成测试由开发人员在开发环境中进行，验收测试主要在用户最终应用环境中进行，在此暂不讨论。因此，需要搭建的环境主要用于被测软件系统（包括 Web 应用）的测试。

规划测试环境的第一步应明确如下问题：

（1）执行测试所需的计算机数量和对每台计算机的硬件配置要求，包括 CPU 速度、硬盘和内存容量、网卡支持的速度等。

（2）部署服务器所需的操作系统、数据库管理系统（DBMS）、中间件、Web 服务器等（以下统称支撑软件环境）的名称、版本，必要时还需明确相关补丁的版本。

（3）用于保存文档（这里主要是指测试过程中生成的文档，而非测试参考文档或存放测试结果的最终文档）和数据的服务器必需的支撑软件环境中各软件的名称、版本，必要时也应明确相关补丁的版本。

（4）测试机所需支撑软件环境中各软件的名称、版本，必要时应明确相关补丁的版本。

（5）用于对被测软件系统的服务器环境和测试管理服务器环境进行备份的专用计算机（该环节是可选的）。这对安全性测试、可恢复性测试等会导致重大软件缺陷的测试类型是非常重要的，测试后若出现重大缺陷，应能利用备份来恢复测试前的原始环境。

(6)测试所需的网络环境。

(7)执行测试工作所需的一些辅助软件。例如,文档编写工具、测试管理系统、性能测试工具、缺陷管理系统等,应明确这些软件的名称、版本、授权证书数量和可能需要的相关补丁的版本。对于性能测试工具,还需要重点留意是否支持被测软件系统所用的协议。

(8)为执行测试用例所需初始化的各项数据。对性能测试而言,还需重点留意执行测试用例之前应满足的历史数据量,以及在测试过程中受到影响的数据的恢复问题。

明确以上问题之后,第二步是确定哪些条件可以得到满足,哪些条件需要其他部门来协调、采购或支援。

最后将上述问题整理为检查表,为每个问题制定负责人,在搭建测试环境的过程中,对照检查表来逐步完成每个问题的设置和检查,填写相关内容,最终形成的文档就作为测试环境的配置说明文档。若时间或其他条件允许,还需要做好应急预案,尽量保证在环境失效时不会对正常工作产生太大的影响。

1.5.4 测试环境的维护和管理

搭建好测试环境之后,通常还需要不断地调整环境。例如,软件新版本的发布、需求的变化等,都有可能对测试环境产生影响。为此,应考虑如下问题。

1)设置测试环境管理员

每个测试小组都应配备一名专门的测试环境管理员,其职责如下:

(1)搭建测试环境,包括安装和配置支撑软件环境涉及的所有必需软件,编写相关安装、配置手册。

(2)详细记录构成测试环境的各台机器的硬件配置、IP 地址、端口配置、机器用途和当前网络环境情况。

(3)部署被测软件系统,编写发布文档。

(4)执行和记录测试环境的各项变更情况。

(5)备份和恢复测试环境。

(6)有效地管理支撑软件环境所涉及的所有用户名、密码和权限。

(7)当测试小组内的多名成员都需要占用服务器,且相互存在冲突的时候,负责分配和管理服务器时间。

2)明确测试环境管理所需的文档

为了对测试环境进行管理,需要以下文档:

(1)各台机器上支撑软件环境中各项软件的安装配置手册。

(2)各台机器的硬件环境文档。

(3)被测软件系统的发布手册,特别要注意记录数据库表的创建、数据导入等内容。

(4)测试环境的备份和恢复方法手册。

(5)用户权限管理文档。

对于每个文档,都应详细记录各项信息的变更历史。

3)管理测试环境的访问权限

应为每个访问测试环境的测试人员和开发人员设置单独的用户名,根据工作需要设置各自的访问权限,以避免误操作从而破坏测试环境,具体原则如下:

(1)测试环境管理员统一管理访问支撑软件环境和被测软件系统涉及的所有用户名、密码及权限。

(2)测试环境管理员拥有全部的权限。

(3)对于开发人员,仅给予对被测软件系统的访问权限,如有特殊要求,可给予测试环境其他部分的只读权限。

(4)对于普通测试员,不给予删除权限。

4)管理测试环境的变更

对于变更,最重要的是能够对每次变更进行追溯和控制,基本的原则如下:

(1)由开发人员或测试人员提出书面申请变更测试环境,由测试环境管理员负责执行,不接受任何非正式的变更申请(如口头申请)。

(2)对测试环境的任何变更都应该记入相应的文档。

(3)与每次变更相关的变更申请文档、软件、脚本等,一律保留原始备份,并作为配置项进行管理。

(4)开发人员将整个系统(包括数据库、应用层、客户端等)打包为可直接发布的格式,由测试环境管理员负责实施发布。测试环境管理员不接受任何不完整的版本发布申请。

5)备份和恢复测试环境

测试环境必须可恢复,因为缺陷必须重现,无法恢复测试环境就意味着无法执行原有的测试用例,缺陷重现更加无从谈起,更不要说修复了。一般应在以下情况下考虑对测试环境进行备份:

(1)当测试环境(尤其是软件环境)发生重大变动时需对其备份,主要是在安装或卸载测试中会碰到这样的情况。

(2)每次发布软件新版本时,应做好当前版本的数据备份工作。

(3)性能测试之前,应做好数据库备份或充分准备好数据的恢复方案。

1.6　软件测试职业

1.6.1　国内外软件测试的现状

在软件行业比较发达的国家,特别是美国,软件测试已经发展成为一个独立的产业,主要体现在以下几个方面:

(1)软件测试在软件公司中占有重要的地位。比尔·盖茨曾在麻省理工学院的一次演讲中说:“在微软,一个典型的开发项目组中的测试工程师要比编码工程师多得多,可以说我们花费在测试上的时间要比花费在编码上的时间多得多。”

(2)软件测试理论研究蓬勃发展。每年举办各种各样的测试技术年会,发表大量的软件测试研究论文,引领软件测试理论研究的国际潮流。

(3)软件测试市场繁荣。美国有一些公司专门开发软件测试标准与测试工具,MI、Compuware、Macabe、Rational 等都是著名的软件测试工具提供商,它们出品的测试工具已经占领了国际市场,目前的主流测试工具大部分都是它们的产品,可见国外的软件测试已经形成

了较大的产业规模。

中国的软件测试技术研究起步于“六五”期间，主要是随着软件工程的研究而逐步发展起来的，由于起步较晚，与国际先进水平相比差距较大。直到 1990 年，成立了国家级的中国软件评测中心，软件测试服务才逐步开展起来。因此，我国无论是在软件测试理论研究还是测试实践上，和国外发达国家相比都有不小的差距，主要体现在对软件产品化测试的技术研究还比较贫乏、从业人员较少、测试服务没有形成足够的规模等方面。但是，随着我国软件产业的蓬勃发展以及对软件质量的逐渐重视，软件测试也越来越被人们所看重，软件测试正逐步成为一个新兴的产业。

1.6.2 软件测试人员结构

软件测试工程师是软件行业中一种既年轻又古老的职业，进入 21 世纪以来，随着中国加入 WTO，从事这项职业的人也越来越多。一个公司在组建一支测试队伍的时候如何分配人员结构，从而使公司软件测试水平得到提高，是一个备受关注的问题。下面介绍软件测试人员的基本结构。

1）测试经理

测试经理主要负责测试队伍的内部管理以及与其他外部人员、客户的交流，具体来说，主要包括进度管理、风险管理、资金管理、人力资源管理、交流管理等。测试经理需要具有项目经理的知识和技能，同时测试工作开始前测试经理需要撰写测试计划书，测试结束需要书写测试总结报告。

2）测试文档审核师

测试文档审核师主要负责前置测试，包括对在需求期与设计期间产生的文档进行审核，比如业务建模书、需求规格说明书、概要设计书、详细设计书等。审核需要撰写审核报告。当文档确定后，需要整理文档报告，并反映给测试设计师。

3）测试设计师

测试设计师主要根据需求分析期与设计期产生的文档来设计各个测试阶段的测试用例。一般来说，测试文档审核师、测试设计师可以由一组相同的人员来完成。

4）测试工程师

测试工程师按照测试用例来完成测试工作。

1.6.3 软件测试人员的素质要求

1）工作态度好，主动性高

工作态度如何，是评价一个软件测试人员的很重要的方面。一个技术能力强的软件测试人员如果没有好的工作态度，在测试团队中不但不能对测试工作起到推动作用，反而可能起到阻碍作用；而一个工作态度好的测试人员，哪怕他的技术水平不高，人也不聪明，但对自己的工作认真负责，也会对测试工作起到很大的促进作用。

2）认真、细心、耐心

软件测试工作是一项烦琐的工作，需要测试人员具备认真、细心、耐心等素质。

有一句话说：细节决定成败。这句话格外适用于软件测试人员。软件测试，简单来说，就是找缺陷以保证产品质量。在一轮又一轮、成千上万的测试用例中发现尽可能多的缺陷，

认真、细心、耐心是一名优秀的测试人员必备的素质要求。

3)学习理解能力强,善于学习总结

不断地学习新技术,不断总结在实际工作中遇到的问题以及解决的方法,并把它们整理归纳,这是一名软件测试人员提高自己技术水平的最好方法。

4)软件测试理论的掌握

开发工具在变,软件测试工具在变,被测试的系统在变……一切都在变。作为一个测试人员,应该怎么去变呢? 测试的类型有很多种,有软硬件测试,有黑白灰盒测试,有功能/系统/压力/Beta(即 β)测试等,但不管系统用的是什么测试,基本步骤是不变的。首先都需要开发人员提供比较好的需求文档、概要/详细设计文档。需求文档是制定测试需求的标准,也是判断系统是否存在问题的标准;概要/详细设计文档是制作测试用例的依据,划分等价类、边界值等基本测试方法都需要这些文档的支持。当然,每一种不同类型的测试都有其特殊的地方,如蓝牙测试就需要对其协议/通信理论(系统环境)有一定的了解,也就是说,好的测试人员必须能够熟练掌握测试理论,做到举一反三。

5)熟悉开发工具和平台

不了解开发平台是无法做单元测试的,也无法做好性能测试,更无法扩展自己的软件测试知识面,了解测试深度。

6)掌握测试工具

测试人员必须至少熟练掌握 1～2 种测试工具。

习　题　1

1. 软件开发与软件测试有何关系?

2. “BUG 就是程序运行时产生的错误”这句话对吗? 为什么?

3. 怎样的测试用例才算是一个良好的测试用例?

4. 什么是软件测试环境?

第 2 章　软件测试原理

知识目标

◎ 软件测试原则
◎ 软件测试的分类
◎ 软件测试过程
◎ 软件测试的过程模型

技能目标

◎ 掌握软件测试的原则
◎ 掌握软件测试的不同的分类标准
◎ 了解不同的软件测试类型
◎ 掌握软件测试的几种不同的过程模型

软件测试涉及技术和管理两个层面的工作，看似头绪纷繁，实际只要了解了测试的主线，就能清楚了解每个阶段的不同角色的职责。作为一名优秀的测试人员，必须了解软件测试的原理。本章就软件测试原理进行系统的介绍。

2.1　软件测试原则

做每一个项目、每一个工程，都有其对应的原则，软件测试也有其对应的原则，这里将作详细介绍。

2.1.1　应尽早和不断地测试

由于软件的复杂性和抽象性，以及项目涉及人员之间沟通的不畅等原因，导致在软件生命周期各阶段都可能产生缺陷，因此，不应该将软件测试看成是程序写完之后才开始的一项工作。缺陷发现得越早，修复缺陷的代价就越小；反之，缺陷发现得越晚，缺陷修复的成本就越高。若缺陷遗留到用户手中，则将对公司、用户都有可能带来极其严重的后果。例如，加拿大的 Therac 25 放射治疗仪事件（由于软件系统总体安全设计出问题，导致多起医疗事故的发生）等，类似的案例不胜枚举。

不断地测试应表现为一个反复、递增的过程。在最初的阶段，可能会采用一些简单的测试方法和模糊的测试过程，测试的效果也许会很差。但经过每次迭代的修正之后，部分测试

将进入回归测试形式，下一次的测试将覆盖前几次测试的范围，从而逐步加大测试的覆盖面；另一方面，在多次迭代过程中，将逐渐加强对测试过程的认识，强化集成测试和系统测试，规范单元测试，最终可能会重新修正整个测试过程。

2.1.2　不可能穷尽测试

想要穷尽测试，就必须在有限的时间和资源（包括人力、物力和资金等）条件下，找到软件中所有潜在的缺陷。开发完美的软件，是不可能的。从总体来看，软件实现的途径太多。从软件的基本要素——输入、输出、数据和计算来分析，能进一步证明穷尽测试是不可能的，原因如下：

（1）从输入来看，每个输入条件的数据量太大，不同输入条件之间的组合情况太多。要得到健壮的软件，还需要考虑更多无效的输入，而无效的输入情况相比有效输入而言，其数据量和组合情况则更加纷繁复杂，无法穷尽。

（2）从输出来看，输出结果太多。输出的种类比输入要少很多，大部分情况下，可以分析出所有可能的输出。但针对每种具体的输入都必须有一个明确的系统输出，既然输入无法穷尽，那么对应的输出也是无法穷尽的。

（3）从数据来看，数据的处理方式也不过是常规的方法，如添加数据、删除数据等。然而，由于要处理的数据量太大，每种数据类型所包含的有效和无效数据往往是无穷多的，针对每组数据来测试各种操作无疑也是异想天开。

（4）从计算来看，由于算法的复杂度越来越高，结合业务的复杂性，导致路径组合近似天文数字。遍历每条路径的穷尽测试，即使是一个非常熟练的测试员，也是无法做到的。

因此，只能通过调节资源，采用多种测试方式，尽可能地发现缺陷，并敦促开发小组修复缺陷。测试的目标不是零缺陷，而是足够好，即通过制定最低质量标准通过测试和测试内容，并利用资源的调节来控制测试程度和测试范围，如程序逻辑覆盖准则。测试出口条件的设置对不同项目应区别对待，但可以参考如下标准：

- 遗留缺陷数量低于 10 个，其中严重的缺陷少于 5 个。
- 测试用例的执行率为 100%，通过率为 95%。
- 对于单元测试，关键模块的语句覆盖率为 100%，判定覆盖率为 85%。

2.1.3　良好的测试态度

1）避免测试自己的程序

测试应由独立的第三方机构来完成。但国内的测试环境并不成熟，因此黑盒测试多由测试人员来完成，而白盒测试则由开发人员通过交叉测试来完成。

让开发人员测试自己的代码是一件相当不合理的事情，理由如下：

（1）不愿否定自己的工作。程序员一向认为测试人员总是在挑刺，让他们自己挑自己程序的毛病，更是难上加难。

（2）受到思维定势的局限。程序员对自己开发的程序很熟悉，测试时难以跳出自己的编码思路，若设计时就存在理解错误，或受不良编码习惯影响而导致缺陷，一般都很难发现。

（3）受进度压力的影响。程序员总是在赶进度，通常仅能在规定时间内写完代码，没有时间去做测试。“让测试人员去测吧”，这是很多开发人员的态度。

（4）程序员对程序的功能和接口很熟悉，这与最终用户的情况往往并不吻合，开发人员

自己来测试程序难以具有典型性。

2)增量测试

应采用增量测试的方式,即测试范围应从小规模开始,逐步转向大规模。所谓小规模,是指测试的粒度,或某种程序的单元测试。进一步的测试将从单个单元的测试逐步过渡到多个单元的组合测试,即集成测试,最终过渡到系统测试。随着从单元测试到集成测试再到系统测试,测试时间、可用资源和测试范围不断扩大。

3)测试应该分级别

测试应该分级别。不同级别的测试,采用的测试活动、测试方法和测试重点等各有不同,如表 2-1 所示。表中,DT&E 测试是指开发测试和评价(development test and evaluation),OT&E 是指操作测试和评价(operational test and evaluation)。

表 2-1 测试级别

测试级别	测试活动	测试方法	测试的文档基础	测试责任主体	测试重点
级别 0	结构化检查	静态测试	各类文档	检查小组	各方面
级别 1	单元测试	白盒测试	软件详细设计文档	开发人员	软件单元设计
级别 2	配置项集成测试	白盒测试	软件概要设计文档	独立测试组	配置项设计/架构
级别 3	配置项资格测试	黑盒测试	软件需求规格说明书	独立测试组	配置项需求
级别 4	集成测试	白盒测试	系统的子系统设计文档	独立测试组	系统设计/架构
级别 5	系统测试	黑盒测试	系统规格说明书	独立测试组	系统需求
级别 6	DT&E 测试	黑盒测试	用户手册	独立测试组	用户手册一致性
级别 7	OT&E 测试	黑盒测试	可操作性需求文档	可操作性测试组	可操作性需求
级别 8	外场测试	黑盒测试	交付计划(场地配置)	外场安装组	场地需求

4)测试应有重点

尽管测试应按一定级别进行,但受到资源和时间的限制,不可能无休止地测试下去。因此,在有限的时间和资源下如何有重点地进行测试是测试管理者需充分考虑的事情。测试的重点选择需进行多方面的考虑,包括测试对象的关键程度、可能的风险、质量要求等。这些考虑与经验有关,随着实践经验的增长,判断也会更准确。

5)避免测试的随意性

软件测试是一项系统工程,是有组织、有计划、有步骤的活动。因此,测试应制订合理的测试计划。测试计划反映一个测试团队在正常情况下需完成的工作远景描述,一般包括测试需求、测试策略、资源、项目里程碑、可交付工作等关键内容;但通常情况下,将资源从测试计划中分离出来是一种更好的习惯。

2.1.4 缺陷的基本特点

一般情况下,缺陷具有如下几个基本特点。

1)缺陷的群集现象

一般情况下,80%的软件缺陷集中在 20%的模块中。造成缺陷群集现象的主要原因如下:

(1)程序员的疲劳。程序员长期疲劳工作,会造成大量代码缺陷,若让程序员进行 12 小时或更长时间的持续编码,后期程序段中出现的缺陷比例会呈直线上升。

(2)程序员的习惯。程序员可能因个人编程习惯而反复犯同样的错误。

2)缺陷有免疫力

Boris Beizer 曾描述了缺陷的“杀虫剂怪事”现象,即软件测试越多,缺陷的免疫力越强的现象。因此,在测试中应使用不同的测试方法、针对程序的不同部分进行测试。一般每修复 3～4 个缺陷,就会产生一个新缺陷,所以在回归测试中要充分注意缺陷的修复所产生的波及面,确定合适的回归测试的范围。

3)缺陷之间的关联和依赖

某个缺陷因其他缺陷而出现或消失,关闭某个缺陷必须先关闭其父类缺陷,这类“缺陷关联”现象表明缺陷之间存在着一定的依赖关系。只有经过回归测试才能确保缺陷被正确地修复。

2.1.5　测试结果的处理原则

1)对缺陷进行复查和确认

一般由测试员(也可以是开发人员和客户)测试出来的缺陷,一定要由项目经理(或由开发人员、开发经理)来确认,严重的缺陷还应召开评审会议进行讨论和分析,从而防止无效的缺陷对资源的浪费。有的缺陷可能是由测试人员的失误造成的,有的缺陷可能是一个重复提交的缺陷,有的缺陷可能会被开发人员认为是符合设计的,这些都应该通过复查予以确认。测试过程混乱和对设计的歧义理解是造成无效缺陷的主要来源,在回归测试中尤其应该注意这些,应利用工具对缺陷进行良好的管理。

2)测试结果的全面检查

测试结果中可能夹杂着大量正确和错误的输出信息,应仔细区分。要特别注意的是,应尽量用自动检查的方式来确认每个测试结果。自动测试工具可以方便地鉴别测试后输出的数据,并给出清晰的缺陷分析报告。在单元测试、系统测试阶段均可使用自动测试工具。

3)出错统计和分析

应对测试过程中找到的所有缺陷进行定期统计和分析,包括对所有缺陷进行定性分析,确定其严重程度和优先级别,同时应支持缺陷的统计分析,如累积打开/关闭缺陷总数、不同模块中的缺陷总数、不同类型的缺陷总数、不同阶段发现的缺陷总数等,以便根据这些统计结果决定是否可以进入下一个测试阶段。若找到项目进展过程中最薄弱的环节,如设计较差的模块、编程能力较弱的程序员等,在后续开发过程中应加以改进,包括人员的培训、缺陷检查表的扩充等。

4)妥善保存测试过程文档

整个测试过程中,应妥善保存一切测试过程文档,便于后续的开发及测试过程改进,提高软件产品质量。测试过程文档包括各阶段的测试计划、测试设计说明书、测试用例说明书、测试脚本、测试总结报告等。

2.2 软件测试的分类

2.2.1 按是否需要查看代码分类

按照是否需要查看代码可将测试分为黑盒测试和白盒测试两种。

1. 黑盒测试

黑盒测试是将被测试软件看做一个黑盒子，只考虑系统的输入和输出，完全不考虑程序内部逻辑结构和处理过程。黑盒测试的依据是各阶段的需求规格说明，如需求分析阶段是产品的需求规格说明书，单元测试阶段是模块的详细设计说明书。

黑盒测试的特征如下：

- 黑盒测试用例与程序如何实现无关。因此，若程序内部逻辑结构和处理过程发生变化，将不会影响该黑盒测试用例。
- 测试用例的设计与程序的开发可以并行进行。
- 没有编程经验的人也可以设计黑盒测试用例（这也是造成当前测试人员的素质偏低的主要原因之一）。当然，一般情况下，具备一定的编程经验最好。

黑盒测试的局限性如下：

- 不可能做到穷举测试。由于输入控件太大，包括的输入条件太多、输入数据量太大、输入条件的组合太复杂等因素，导致黑盒测试不可能做到穷举测试。
- 因不可能做到输入的穷尽，而只能从中选择部分输入构成测试用例，故黑盒测试是很有可能存在漏洞的。

黑盒测试又称功能性测试（functional testing）或数据驱动测试（data-driven testing）。但应注意，功能性测试不等于功能测试。黑盒测试是从功能的角度检查软件是否满足需求规格说明的要求，但并不仅限于功能性测试。

2. 白盒测试

白盒测试是将黑盒打开，研究源代码和程序内部的逻辑结构。白盒测试的依据是程序代码。

利用白盒测试的覆盖指标所设计的测试用例与采用黑盒方法所得到的测试用例常常存在重复。因此，白盒测试一般充当黑盒测试的补充。

程序代码往往具有多个分支。白盒测试可以利用不同的覆盖准则来测试这些分支，黑盒测试则无法做到这一点。

白盒测试的覆盖指标可以充当黑盒测试的检查手段。例如，若采用黑盒方法设计的测试用例（如边界值测试）没有满足某些白盒测试覆盖指标（如判定覆盖）的要求，则证明该测试用例集合必然存在漏洞。

与黑盒测试相比，白盒测试具有如下特殊的应用领域：

- 代码中常存在内存泄漏的问题，尤其是 C/C++ 程序，白盒测试可以方便地发现内存泄漏，且是直接定位缺陷，而黑盒测试只能通过长时间运行程序，并仔细地检查用例执行结果，才能发现这类问题。

- 有时只有在某种极端的条件下才会出现的情况，是难以直接进行功能性测试的，如卫星在太空中收到电磁辐射。这时，缺陷预防是测试的主要目的，白盒测试通过对源代码的静态分析可以发现该类问题。

白盒测试也存在局限性，特别是不可能做到穷举测试，原因如下：

- 程序中的逻辑路径太多，往往需要面对路径爆炸的问题。例如，有的代码可能不到 100 行，却具有 520 条可能的路径，若每条路径设计一个测试用例，且每个测试用例执行一次需要 1 ms，则连续执行所有的测试用例会需要很长时间。
- 由于设计的原因，程序的实际执行路径可能比逻辑路径略微少一点，但实际应用的程序也很多。

实际上，穷举路径测试也不等于完全的测试，因为它无法发现程序违反设计规范的地方；无法发现程序本身是否缺少某些路径；不能发现程序中已经实现但不是用户所需要的功能；可能无法暴露数据敏感错误，即与数据相关的错误；经常发现不了用户操作行为的缺陷。

白盒测试又称结构性测试(structural testing)或逻辑驱动测试(logic-driven testing)。

值得注意的是，白盒测试并非仅限于对程序源代码的测试。对于高层的测试，仍然可以采用某些白盒测试的方法。

3. 黑盒测试与白盒测试的比较

请大家思考一下，当受到进度压力的时候，通常会减少黑盒测试的工作量还是白盒测试的工作量呢？

黑盒测试和白盒测试从不同的方面来检查被测软件，无论单独采用黑盒测试，还是单独采用白盒测试都无法做到全面的测试。仅用黑盒测试方法，将无法发现程序内部结构的缺陷。这些缺陷可能暂时不会导致软件的失效，但可能是个隐患，当特定的条件被触发(如大数据量处理)时，就可能使软件出现问题，而白盒测试可以及时发现这些缺陷。仅用白盒测试的方法，则无法从宏观上来观察软件运行结果，如程序是否满足设计要求、软件是否符合用户需求等，这些都是更为严重的缺陷。

黑盒测试通常用于软件的系统测试、验收测试、功能和性能测试等方面，由测试人员来完成；白盒测试主要在单元测试、集成测试中采用，通常由开发人员来完成。

2.2.2　按是否需要执行被测试软件分类

按照是否需要执行被测软件进行分类，可将软件测试分为静态测试和动态测试。

1. 静态测试

静态测试(static testing)又称为静态分析(static analysis)，它不实际运行被测软件，而是通过分析软件的形式和结构查找缺陷。静态测试主要包括对源代码、程序界面和各类文档及中间产品(如产品规格说明书、技术设计文档等)所做的测试。

1)对于源代码

静态测试主要是看代码是否符合相应的标准和规范，如可读性、可维护性等，其工作过程类似一个编译器，随着语法分析的进行做特定工作，如分析模块调用图、程序的控制流图等图表，度量软件的代码质量等。一般各公司内部都有自己相应的编码规范，如 C/C++ 编码规范、Java 编码规范等，测试人员应按照规范中所列出的条目逐条测试。

源代码中含有大量原设计信息和程序异常的信息。利用静态测试，不仅可以发现程序

中明显的缺陷，还可以帮助程序员重点关注那些可能存在缺陷的高风险模块，如多出口的情况、程序复杂度过高的情况、接口过多的情况等。

当然，对源代码进行静态测试并非易事，可以借助一些自动化的静态分析工具来降低测试人员的劳动强度。目前市面上已有很多静态分析工具，如 Telelogic 公司的 Logiscope、Parasoft 公司的C++ Test 等，这些静态分析工具一般由 4 个部分组成：语言程序预处理器、数据库、错误分析器和报告生成器。

2)对于程序界面

静态测试主要是查看软件的实际操作和运行界面是否符合需求中的相关说明。

3)对于文档

静态测试主要是检查用户手册与需求说明是否真正符合用户的实际要求。

程序界面和文档的静态测试相对容易一些，但要求测试人员应充分熟悉用户需求，且比较细心。从实际情况来看，程序界面和文档的测试常常是不受重视的。

静态测试是采用走查、同行评审、会审等方法来查找错误或收集所需度量数据的。它不需要运行程序，所以相对动态测试，可以更早地进行。

静态测试的查错和分析功能是其他方法所不能替代的，静态测试能发现文档中的问题(也只能通过静态测试发现)，通过文档中的问题或其他软件评审来发现需求分析、软件设计中的问题，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自动化代码校验方法。

2. 动态测试

动态测试(dynamic testing)又称动态分析(dynamic analysis)，是指需要实际运行被测软件，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，发现程序所表现出的行为与设计规格或客户需求不一致的地方。

动态测试是一种经常使用的测试方法，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。但它也存在很多局限性，主要体现在以下几方面：

(1)往往需要借助测试用例来完成，即通过执行测试用例，分析测试结果来对被测软件重点考查，以便发现缺陷。相比静态测试，动态测试增加了测试用例的设计、执行和分析，以及由测试用例所带来的用例组织与管理等一系列活动。

(2)需要搭建软件特定的运行环境，增加了有关测试环境的配置、维护和管理的工作量。

(3)不能发现文档问题，必须等程序代码完成后进行，发现问题相对迟得多。

3. 静态测试与动态测试的比较

表 2-2 中对静态测试和动态测试进行了简单的比较。大家不妨思考一下，是不是静态测试的成本更低，是否可以用静态测试替代动态测试。

表 2-2　静态测试与动态测试的比较

比较的项目 / 测试方法	是否需要运行软件	是否需要测试用例	可否直接定位缺陷	测试实现难易程度
静态测试	否	否	可以	容易
动态测试	是	是	不可以	困难

静态测试与动态测试之间既具有一定的协同性，同时又具有相对的独立性。程序静态分析的目标不是证明程序完全正确，而是作为动态测试的补充，在程序运行前尽可能多地发现代码中隐含的缺陷。静态测试是不能完全代替动态测试的，可以通过以下两个方面进行比较。

1）协同性

静态测试和动态测试在各自的优缺点上具有互补性。静态测试是保守和健壮的，其测试结果离期望值可能还有距离，但它保证了将来的执行；动态测试是有效和精确的，它不需要花费大量的分析过程，尽管它确实需要测试用例的设计、执行和结果分析。动态测试给出了高度精确的结果。

2）独立性

静态测试需要建立程序的状态模型（如函数调用图、控制流图等），在此基础上确定程序对该状态的反映，如通过各种图表分析，找出多入口多出口的模块、高层控制模块等。因系统可能执行的状态有很多，测试必须跟踪多个不同的状态，通常经过大量细致的分析后也不一定能考虑到所有的系统状态，因此，静态分析通常采用程序状态的抽象模型，并需要较长时间的等待。

而动态测试过程中不存在近似和抽象的概念，它直接执行程序段，检查实时的行为，在控制流程路径中，几乎不存在不确定因素。

2.2.3　按测试阶段分类

按照测试的阶段进行分类，可将软件测试分为单元测试、集成测试、系统测试等。

1. 单元测试

单元测试（unit testing）又称模块测试（module testing），是指对软件中的最小可测试单元进行测试，目的是检查每个单元是否能够正确实现详细设计说明中的功能、性能、接口和设计约束等要求，发现各个模块内部可能存在的各种缺陷。

在程序员编码之后，代码通过编译一般就可以进行单元测试了。由于国内的单元测试往往不正规，因此单元测试多由开发人员自己来完成，一般是交叉测试。

单元测试的主要依据是程序代码和详细设计文档。根据设计文档、编码规范和注释，从程序内部结构出发，查看代码是否符合设计、规范以及注释的相关说明。单元测试过程中应结合黑盒与白盒测试方法，且大部分软件测试方法基本都适用于单元测试。

单元测试的优点在于：

- 它是一种管理和组合测试元素的手段。通过单元测试，可实现测试从“小规模”向“大规模”的逐步转变，从而降低测试难度，提高测试效率。
- 降低调试的难度。调试是准确定位并纠正某个已知缺陷的过程，在测试用例指导下的单元测试针对性更强，更加有利于缺陷的定位。
- 提供同时测试多个单元的可能。多个相互独立或关联性不大的单元可以并行地、独立地进行单元测试，从而加快整体的测试速度和项目进度。

2. 集成测试

集成测试（integration testing）又称组装测试，是在单元测试的基础上，按照设计要求，将通过单元测试的单元组装成系统或子系统而进行的有序的测试，目的是检验不同程序单元或部件之间的接口关系是否符合概要设计的要求，能否正常运行。

集成测试并非等到所有单元测试完成之后才开始，而是同步进行的，即在单元测试中先对几个单元进行独立的单元测试，然后将其组装起来进行集成测试，查看其接口是否存在缺陷。集成测试一般也由白盒测试人员或开发人员来完成。

集成测试的主要依据是单元测试的单元及概要设计文档。

由于软件的集成往往是持续的过程，将形成很多临时版本，在不断的集成过程中，功能集成的稳定是需要解决的主要问题。在每个版本提交的时候，应先进行冒烟测试（即版本验证测试）来验证主要功能是否能够正常执行。

与单元测试相比，集成测试主要考查单元的外部接口，而单元测试主要从单元内部来测试。

3. 系统测试

系统测试（system testing）是为了验证和确认系统是否达到其原始目标，而对集成的硬件和软件系统进行的测试，是在真实或模拟系统运行的环境下，检查完整的程序是否能和环境系统（包括硬件、外设、网络和系统软件、支持平台等）正确配置、连接，满足用户需求。

系统测试主要由黑盒测试人员在整个系统集成好之后进行。前期主要看系统功能是否满足需求，这被称为功能测试；后期主要测试系统运行是否满足要求，以及系统在不同硬件和软件环境中的兼容性等，这被分别称为功能测试、兼容性测试、用户界面测试等。有人也喜欢将功能测试从系统测试中单独提出来，作为集成测试和系统测试之间的一个测试环节。

系统测试的主要依据是软件的需求规格说明文档。

4. 验收测试

验收测试（acceptance testing）又称接受测试，是一种正式的测试，是在系统测试之后，以用户测试为主，或有测试人员等质量保证人员共同参与的，并由客户或其他权威机构来决定是否可以接受一个产品（系统或组件）的验证性测试。验收测试是软件正式交付用户使用的最后一个测试环节，用户将决定是否最终验收签字和结清所有应付款。

验收测试的主要依据是软件需求规格说明文档和验收标准。验收测试的测试用例可以直接采用内部测试组所设计的系统测试用例的子集，也可以由验收人员自行设计。

验收测试又分 α 测试和 β 测试。

1）α 测试

α 测试也称开发方测试，开发方通过检测和提供客观证据，证明软件运行是否满足用户规定的需求。它是在软件开发环境下，由用户、测试人员和开发人员共同参与的内部测试，属于产品早期性测试。该测试一般在可控制环境下进行，可以是用户在开发环境下进行的测试，也可以是公司内部用户在模拟实际操作环境下进行的受控测试。α 测试被戏称为“放任内部成员胡作非为的测试”。

2）β 测试

β 测试是内部测试之后的外部公开测试，是将软件完全交给用户，让用户在实际使用环境下进行的对产品预发布版本的测试。该测试是在开发者无法控制的环境下进行的软件现场应用，其实施过程是先将软件产品有计划地分发到目标市场，让最终用户大量使用、评价和检查，从而发现软件缺陷，然后从市场收集反馈信息，把关于反馈信息的评价制成容易处理的数据表，再将这些数据表分发给所涉及的各个部门进行修改。它通常被看成是一种“用户测试”，这也是 β 测试定义的核心思想。从有效的 β 测试中可以获取大量信息。β 测试被戏称为“让全世界的坏人都胡作非为的测试”。

β 测试的优势如下：

- 提升了产品价值。它使得“实际”的客户有机会将自己的意见渗透到公司产品的设计功能和使用过程中。这些意见不但可对测试产品起到非常重要的作用，还有利于将收集的数据有效利用并促进公司未来新产品的研发。
- 可发现一些在测试实验室无法发现，甚至重复出现的缺陷。β 测试常常会发现产品的局限所在，并测试产品的整个开发过程。
- 可以将公司推到“基准框架”之外，使得公司更了解用户的需求，并为产品设计提供指南，有助于对产品的未来作出重要决策。
- 有助于产品的成功发布。

5. 各阶段测试的比较

各阶段测试的比较如表 2-3 所示。

表 2-3　各阶段测试的比较

测试阶段	测试对象	测试依据	涉及人员	测试方法	公司的重视程度
单元测试	单元	代码和详细设计文档	白盒测试人员或开发人员	主要为白盒测试	不重视
集成测试	单元之间的接口	概要设计文档	白盒测试人员或开发人员	黑盒和白盒测试	不太重视
系统测试	整个软、硬件系统	需求规格说明文档	黑盒测试人员	黑盒测试	重视
验收测试	整个软、硬件系统	需求规格说明文档和验收标准	用户或测试人员	黑盒测试	重视

2.2.4　按测试执行时是否需要人工干预分类

按照测试执行时是否需要人工干预进行分类，可将软件测试分为手工测试和自动测试。

1. 手工测试

手工测试是完全由人工完成测试工作，包括测试计划的制订、测试用例的设计和执行，以及测试结果的检查和分析等。传统的测试工作都是由人工来完成的。

2. 自动测试

自动测试是各种测试活动的管理与实施，是使用自动化测试工具或自动化测试脚本来进行的测试，包括测试脚本的开发与执行等，它以某种自动化测试工具来验证测试需求。这类测试在执行过程中一般不需要人工干预，通常在功能测试、回归测试和性能测试中使用较为广泛。

自动测试的优点如下：

(1)能够产生可靠的系统。自动测试可改进所有的测试领域，若自动测试工具和方法被正确地使用，且遵循定义的测试过程，则可以改进需求定义，改进性能测试，改进压力测试、质量测量并使测试最佳化，改进与开发组的伙伴关系，改进系统开发生存周期。

(2)提高测试工作质量。使用自动测试工具可增加测试的深度和广度，包括改进冒烟测试，改进回归测试，改进多平台兼容性测试，改进普通的执行测试，集中解决高级测试问题，

提高重视缺陷的能力，增强业务专业知识，完成手工测试无法完成的测试，等等。

(3)减少测试工作量并缩短进度。

2.2.5 其他测试类型

其他重要的测试类型包括冒烟测试、随机测试等。

1. 冒烟测试

冒烟测试是自由测试的一种。通过测试发现问题，找到了一个 BUG，然后开发人员会来修复这个 BUG，这时想知道这次修复是否真的解决了程序的 BUG，或者是否会对其他模块造成影响，就需要针对此问题进行专门测试，这个过程就被称为冒烟测试(smoke test)。在很多情况下，冒烟测试是防止开发人员在试图解决一个问题的时候，造成了其他功能模块一系列的连锁反应，原因可能是开发人员只集中考虑了一开始的那个问题，而忽略了其他问题，引起新的 BUG。冒烟测试的优点是节省测试时间，防止修复失败；缺点是覆盖率比较低。

2. 随机测试

随机测试是根据测试说明书执行样例测试的重要补充手段，是保证测试覆盖完整性的有效方式和过程。随机测试主要是对被测软件的一些重要功能进行复测，也包括测试那些当前的测试样例(test case)没有覆盖到的部分。另外，对于软件更新和新增加的功能要重点测试，重点对一些特殊情况点、特殊的使用环境、并发性进行重点检查，尤其需要对在以前的测试中发现的重大 BUG 进行再次测试，可以结合回归测试(regressive testing)一起进行。

理论上，每一个被测软件版本都需要执行随机测试，尤其对于最后将要发布的版本更要重视随机测试。随机测试最好由具有丰富测试经验的熟悉被测软件的测试人员进行。对被测试的软件越熟悉，执行随机测试越容易。

2.3 软件测试过程

软件测试过程是一种抽象的模型，用于定义软件测试的流程和方法。众所周知，开发过程的质量决定了软件的质量，同样，测试过程的质量将直接影响测试结果的准确性和有效性。软件测试过程和软件开发过程一样，都遵循软件工程原理，遵循管理学原理。

随着测试过程管理的发展，软件测试专家通过实践总结出了很多很好的测试过程模型。这些模型将测试活动进行了抽象，并与开发活动进行了有机的结合，是测试过程管理的重要参考依据。

2.4 软件测试的过程模型

2.4.1 V 模型

V 模型是最具有代表意义的测试模型，如图 2-1 所示。V 模型最早是由 Paul Rook 在 20 世纪 80 年代后期提出的，V 模型在英国国家计算中心文献中发布，旨在改进软件开发的

效率和效果。

在传统的开发模型中(如瀑布模型),人们通常把测试过程作为在需求分析、概要设计、详细设计和编码设计全部完成之后的一个阶段,尽管有时测试工作会占用整个项目周期一半的时间,但是有人仍然认为测试只是一个收尾工作,而不是主要的过程。V 模型的推出就是对这种认识的改进。V 模型是软件开发瀑布模型的变种,它反映了测试活动与分析和设计的关系。该模型从左到右,描述了基本的开发过程和测试行为,非常明确地标明了测试过程中存在的不同级别,并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系,如模型图 2-1 的箭头代表了时间方向,左边依次下降的是开发过程各阶段,与此相对应的是右边依次上升的部分,即测试过程的各个阶段。

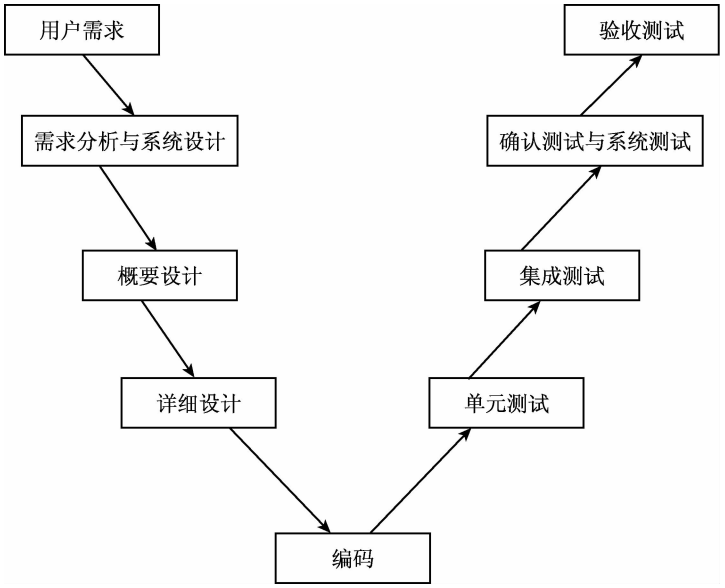

图 2-1　软件测试 V 模型

V 模型的软件测试策略既包括低层测试,又包括高层测试,低层测试是为了源代码的正确性,高层测试是为了使整个系统满足用户的需求。

V 模型指出,单元测试和集成测试是验证程序设计的,开发人员和测试组应检测程序的执行是否满足软件设计的要求;系统测试应当验证系统设计,检测系统功能、性能的质量特性是否达到系统设计的指标;由测试人员和用户进行软件的确认测试和验收测试,按照软件需求说明书进行测试,以确定软件的实现是否满足用户需求或合同的要求。

V 模型存在一定的局限性,它仅仅把测试过程作为在需求分析、概要设计、详细设计及编码之后的一个阶段,容易使人理解为测试是软件开发的最后一个阶段,主要是针对程序进行测试以寻找错误,而需求分析阶段隐藏的问题一直到后期的验收测试才被发现。

2.4.2　W 模型

1. W 模型的建立

V 模型的局限性在于没有明确地说明早期的测试,不能体现“尽早地和不断地进行软件测试”的原则。在 V 模型中增加软件各开发阶段应同步进行的测试,被演化为一种 W 模型,

因为实际上开发是 V,测试也是与此并行的 V。基于“尽早地和不断地进行软件测试”的原则,在软件的需求和设计阶段的测试活动应遵循 IEEE Std 1012—2004《软件验证和确认(V&V)》的原则。

一个基于 V&V 原则的 W 模型示意图如图 2-2 所示。

2. W 模型的应用

W 模型由 Evolutif 公司提出,相对于 V 模型,W 模型更科学。W 模型可以说是 V 模型自然而然的发展。它强调:测试伴随着整个软件开发周期,而且测试的对象不仅仅是程序,需求、功能和设计同样要测试。这样,只要相应的开发活动完成,就可以开始执行测试,可以说,测试与开发是同步进行的,从而有利于尽早地发现问题。以需求为例,需求分析一完成,就可以对需求进行测试,而不是等到最后才进行针对需求的验收测试。

如果测试文档能尽早提交,那么就有了更多的检查和测试的时间,这些文档还可用于评估开发文档。另外还有一个很大的益处是,测试者可以在项目中尽可能早地找出缺陷所在,从而帮助改进项目内部的质量。参与前期工作的测试者可以预先估计问题和难度,这将可以显著地减少总体测试时间,加快项目进度。

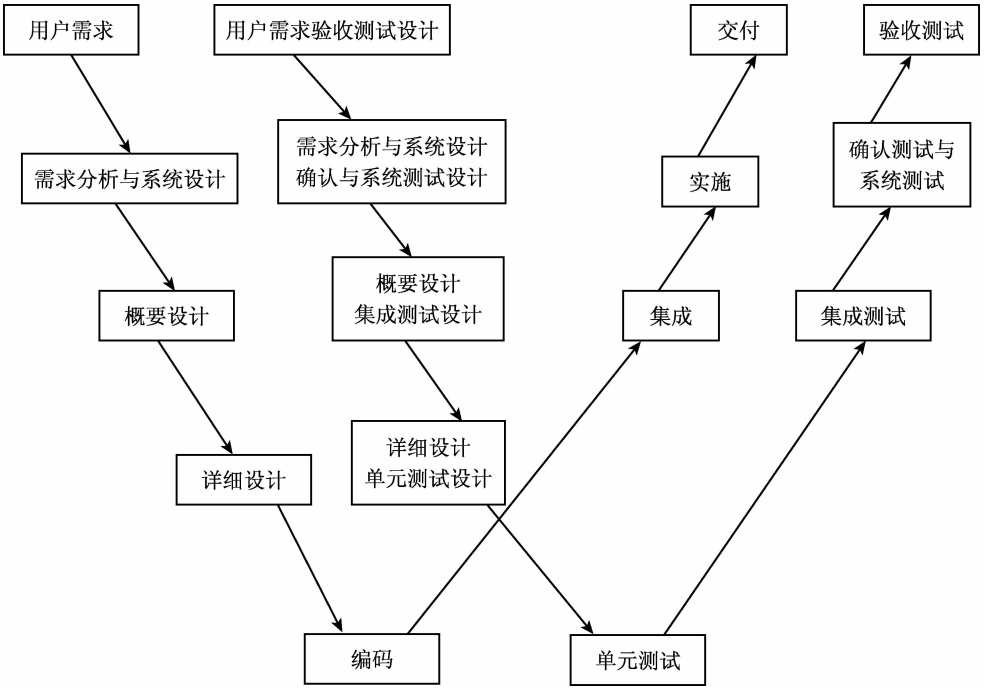

图 2-2 软件测试 W 模型

根据 W 模型的要求,一旦有文档提供,就要及时确定测试条件,以及编写测试用例,这些工作对测试的各级别都有意义。当需求被提交后,就需要确定高级别的测试用例来测试这些需求;当概要设计编写完成后,就需要确定测试条件来查找该阶段的设计缺陷。

W 模型也是有局限性的。W 模型和 V 模型都把软件的开发视为需求、设计、编码等一系列串行的活动。同样,软件开发和测试保持一种线性的前后关系,需要有严格的指令表示上一阶段完全结束才可正式开始下一阶段。这样就无法支持迭代、自发性以及变更调整。

2.4.3　H 模型

1. H 模型的建立

V 模型和 W 模型均存在一些不妥之处。首先，它们都把软件的开发视为需求、设计、编码等一系列串行的活动，而事实上，虽然这些活动之间存在相互牵制的关系，但在大部分时间内，它们是可以交叉进行的。虽然软件开发期望有清晰的需求、设计和编码阶段，但严格的阶段划分只是一种理想状况，所以相应的测试之间也不存在严格的次序关系。同时，各层次之间的测试也存在反复触发、迭代和增量关系。其次，V 模型和 W 模型都没有很好地体现测试流程的完整性。

为了解决以上问题，有专家提出了 H 模型。它将测试活动完全独立出来，形成一个完全独立的流程，将测试准备活动和测试执行活动清晰地体现出来。

2. H 模型的应用

H 模型的简单示意图如图 2-3 所示。

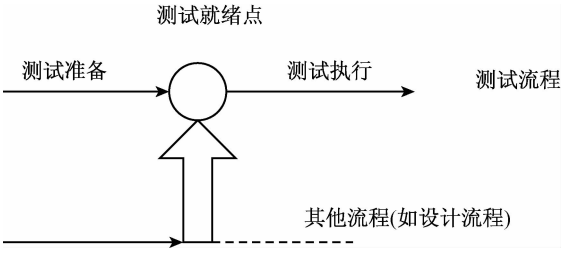

图 2-3　软件测试 H 模型

这个示意图仅仅演示了在整个生产周期中某个层次上的一次测试“微循环”。图中的其他流程可以是任意开发流程，如设计流程和编码流程；也可以是其他非开发流程，如 SQA 流程，甚至是测试流程自身。也就是说，只要测试条件成熟了，测试准备活动完成了，测试执行活动就可以(或者说需要)进行了。

概括地说，H 模型揭示了如下一些内容：

- 软件测试不仅仅指测试的执行，还包括很多其他的活动。
- 软件测试是一个独立的流程，贯穿产品整个生命周期，与其他流程并发地进行。
- 软件测试要尽早准备，尽早执行。
- 软件测试是根据被测软件的不同而分层次进行的。不同层次的测试活动可以是按照某个次序先后进行的，但也可能是反复进行的。

在 H 模型中，软件测试模型是一个独立的流程，贯穿于整个产品周期，与其他流程并发地进行。当某个测试时间点就绪时，软件测试即从测试准备阶段进入测试执行阶段。

2.4.4　X 模型

由于 V 模型受到了很多人的质疑，因此，也有人提出了一些不同的观点和意见。在此，介绍另外一种测试模型，即 X 模型，其目标是弥补 V 模型的一些缺陷。

X 模型的基本思想是由 Marick 提出的，但首先 Marick 不建议建立一个替代模型，同时，他也认为他的观点并不足以支撑一个模型的完整描述。不过，Robin F. Goldsmith 在自己的文章

里将 Marick 的思想定义为 X 模型，理由是，在 Marick 的观点中已经具备一个模型所需要的一些主要内容，其中也包括了像探索性测试这样的亮点。软件测试 X 模型如图 2-4 所示。

Marick 对 V 模型最主要的批评是，V 模型无法引导项目的全部过程。他认为一个模型必须能处理开发的所有方面的缺陷，包括交接、频繁重复的集成以及需求文档的缺乏等。Marick 认为一个模型不应该规定那些和当前所公认的实践不一致的行为。

X 模型左边描述的是针对单独程序片段所进行的相互分离的编码和测试，此后将进行频繁的交接，通过集成最终合成为可执行的程序。这一点在图的右上方得以体现，而且这些可执行程序还需要进行测试，已通过集成测试的成品可以进行封版并提交给用户，也可以作为更大规模和范围内集成的一部分。

同时，X 模型还定位了探索性测试，如图 2-4 中右下方所示。这是不进行事先计划的特殊类型的测试，诸如“我这么测一下，结果会怎么样”，这一方式往往能帮助有经验的测试人员在测试计划之外发现更多的软件错误。

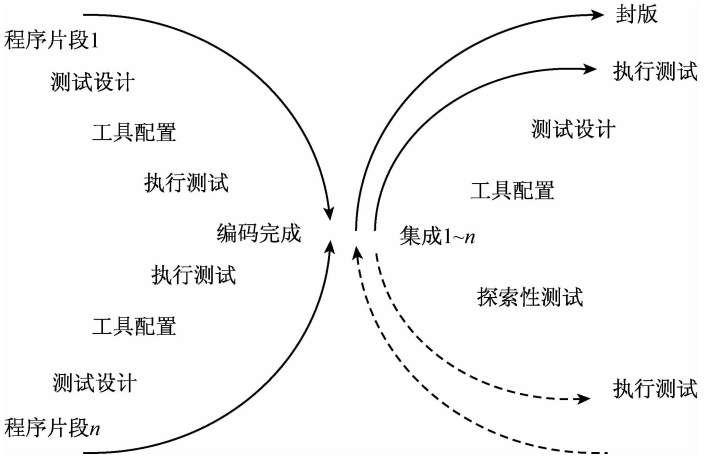

图 2-4　软件测试 X 模型

Marick 对 V 模型提出质疑，也是因为 V 模型是基于一套必须按照一定顺序严格排列的开发步骤，而这很可能并没有反映实际的实践过程。因为在实践过程中，很多项目是缺乏足够的需求的，而 V 模型是从需求处理开始的。

Marick 也质疑了单元测试和集成测试的区别，因为在某些场合人们可能会跳过单元测试而热衷于直接进行集成测试。Marick 担心人们盲目地跟随学院派的 V 模型，按照模型所指导的步骤进行工作，而实际上某些做法并不切合实际。

习　题　2

1. 缺陷具有哪些基本特点？
2. 良好的单元测试是否可以替代集成测试？
3. 自动化测试工具可以让繁重的手工劳动变得轻松，还有必要做手工测试吗？
4. 只要学会某种时下流行的自动化测试工具的使用，就可以轻松做好测试工作了吗？
5. 什么是冒烟测试？如何实施？
6. 如何应用各种测试过程模型？

第3章　黑盒测试

知识目标

◎ 黑盒测试的优点和缺点
◎ 黑盒测试的方法
◎ 黑盒测试的工具

技能目标

◎ 了解黑盒测试的优点和缺点
◎ 掌握几种常用的黑盒测试方法
◎ 认识和了解几种不同的黑盒测试工具

黑盒测试又称为数据驱动测试或基于规范的测试。它完全不考虑程序内部结构和内部特性,而只注重于功能测试。黑盒测试是软件测试的主要方法之一。

3.1　黑盒测试的优点和缺点

3.1.1　黑盒测试的优点

黑盒测试主要有如下几个优点:
(1)有针对性地寻找问题,并且定位问题准确。
(2)黑盒测试可以证明产品是否达到用户要求的功能,符合用户的工作要求。
(3)能重复执行相同的动作,测试工作中最枯燥的部分可交由机器完成。

3.1.2　黑盒测试的缺点

黑盒测试主要有如下几个缺点:
(1)需要充分了解产品用到的技术,测试人员需要具有较多经验。
(2)在测试过程中很多是手工测试操作。
(3)测试人员要负责大量文档、报表的编制和整理工作。

3.2 黑盒测试的方法

由于软件的开发速度比较快，用户的需求多变，需要不断地调整应用，因此要求对软件有更加严格、严密的测试。由于不断变化的需求将导致不同应用版本的产生，每一个版本都需要测试。测试工作头绪多，测试人员难以组织科学、全面的测试用例，从而影响测试的全面性和有效性。并且，测试过程要求大量的元素配合，包括许多的步骤、测试者、大量测试数据和不同应用的多种版本等。而黑盒测试着眼于程序的外部结构，不考虑内部逻辑结构，只针对软件界面和软件功能进行测试，因此，黑盒测试人员需要一个快速、可重用的测试过程，从而最有效地使用现有测试资料、测试方法和测试工具建立测试用例，自动执行测试并产生文档结果。

采用黑盒技术设计测试用例的方法主要有等价类划分法、边界值分析法、因果图法、决策表法、场景设计法、功能图分析法、正交试验法和错误推测法等。

3.2.1 等价类划分法

1. 等价类划分法简述

等价类划分法是一种黑盒测试技术，它不考虑程序的内部结构，只根据软件的需求说明来对输入的范围进行细分，然后再从分出的每一个区域内选取一个有代表性的测试数据。如果等价类划分得好，这个代表性的测试数据的作用就等价于其区域内的其他取值。

等价类又可分为有效等价类和无效等价类。

- 有效等价类：符合需求规格说明书，合理地输入数据集合。
- 无效等价类：不符合需求规格说明书，无意义地输入数据集合。

在利用等价类设计测试用例时，要同时考虑这两种等价类，因为软件不仅要能够接受合理的数据，也要接受不合理的数据进行检验，这样的测试才能确保软件具有更高的可靠性。

2. 划分等价类的原则

划分等价类的原则如下：

(1)在输入条件规定了取值范围或值的个数的情况下，可确立一个有效等价类和两个无效等价类。

(2)在输入条件规定了输入值的集合或规定了“必须如何”的条件的情况下，可确立一个有效等价类和一个无效等价类。

(3)在输入条件是一个布尔变量的情况下，可确定一个有效等价类和一个无效等价类。

(4)在规定了输入数据的一组值(假定 n 个)，并且程序要对每一个输入值分别处理的情况下，可确立 n 个有效等价类和一个无效等价类。

(5)在规定了输入数据必须遵守的规则的情况下，可确立一个有效等价类(符合规则)和若干个无效等价类(从不同角度违反规则)。

(6)在确定已划分的等价类中各元素在程序处理中的方式不同的情况下，应再将该等价类进一步划分为更小的等价类。

3. 划分等价类的要求

划分等价类的要求如下：

(1)测试完备合理，避免冗余。

(2)划分输入条件、有效等价类和无效等价类时最重要的是将集合划分为互不相交的一组子集。

(3)整个集合完备。

(4)子集互不相交，保证一种形式的无冗余性。

(5)同一类中标识(选择)一个测试用例，在同一等价类中，往往处理过程相同，将相同处理映射到“相同的执行路径”。

4. 等价类表的建立

在建立等价类后，可建立等价类表，从而列出所有划分出的等价类。等价类表的建立如表 3-1 所示。

表 3-1 等价类表的建立

输入等价类	有效等价类	无效等价类

等价类表建立后，从划分出的等价类中按以下步骤确定测试用例：

- 为每一个等价类规定一个唯一的编号。
- 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止。
- 设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止。

5. 测试用例等价类表的建立

【例 3-1】 建立小区物业住宅管理系统“日期检查功能”的测试用例等价类表。

有一个小区物业住宅管理系统，要求住户输入以年月表示的日期。假设日期限定在 1988 年 1 月到 2068 年 12 月，并规定日期由 6 位数字组成，前 4 位表示年，后两位表示月。现用等价类划分法设计测试用例，“日期检查功能”的测试用例等价类表如表 3-2 所示。

表 3-2 “日期检查功能”的测试用例等价类表

输入等价类	有效等价类	无效等价类
日期类型及长度	①6 位数字	④有非数字字符；⑤少于 6 位数字；⑥多于 6 位数字
年份范围	②1988～2068	⑦小于 1988；⑧大于 2068
月份范围	③01～12	⑨等于 00；⑩大于 12

6. 测试用例的设计

【例 3-2】 沿用【例 3-1】的小区物业住宅管理系统“日期检查功能”的测试用例设计。

覆盖所有的有效等价类，在表中列出了 3 个编号，分别为①、②、③；覆盖所有的无效等价类，在表中列出了 7 个编号，分别为④、⑤、⑥、⑦、⑧、⑨、⑩，设计一个测试用例，设计的测试用例结果如表 3-3 所示。

表 3-3 设计的测试用例结果

测试数据	期望结果	覆盖的有效/无效等价类
200611	有效输入	①、②、③
199901	有效输入	①、②、③
205901	有效输入	①、②、③
9954￥9	无效输入	④
20097	无效输入	⑤
20120607	无效输入	⑥
198401	无效输入	⑦
208401	无效输入	⑧
200400	无效输入	⑨
200422	无效输入	⑩

3.2.2 边界值分析法

1. 边界值分析法简述

边界值分析法用于列出单元功能、输入、状态及控制的合法边界值和非法边界值，对数据进行测试，检查用户输入的信息、返回结果以及中间计算结果是否正确，补充等价划分的测试用例设计技术。边界值分析法比较简单，仅用于考察正处于等价划分边界或在边界附近的状态，选择输入和输出等价类的边界，选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值作为测试数据。它是对等价类划分方法的补充，不仅重视输入条件边界，而且也从输出域中导出测试用例。典型的边界值分析包括 IF 语句中的判别值、定义域、值域边界、空输入或畸形输入等。边界值分析法是以边界情况的处理作为主要目标专门设计测试用例的方法。

2. 设计边界值测试用例的原则

对边界值设计测试用例，应遵循以下原则：

- 如果输入条件规定了值的范围(或是规定了值的个数)，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围的边界值作为测试输入数据。
- 如果输入条件规定了值的个数，则用最大个数、最小个数、比最小个数少 1、比最大个数多 1 的数作为测试数据。
- 如果程序的规定说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。
- 如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试用例。
- 分析规格说明，找出其他可能的边界条件。

3. 常见的边界值

常见的边界值如下：

- 屏幕上光标在最左上、最右下位置。
- 报表的第一行和最后一行。

• 数组第一个和最后一个元素。
• 循环的第0次,第1次,第2次,…,最后一次。

测试所包含的边界检验有几种类型:数字、字符、位置、大小、方位、尺寸、空间等。

3.2.3　因果图法

1. 因果图法简述

因果图法是一种适合于描述对于多种条件的组合、相应产生多个动作形式的方法。它利用图解法分析输入的各种组合情况,从而设计测试用例,适合于检查程序输入条件的各种组合情况。等价类划分法和边界值分析法都着重考虑输入条件,但没有考虑输入条件的各种组合和输入条件之间的相互制约关系。虽然各种输入条件可能出错的情况已经测试到了,但多个输入条件组合起来可能出错的情况却被忽视了,要检查输入条件的组合不是一件容易的事情,即使将所有输入条件划分成等价类,它们之间的组合情况也相当多,因此必须考虑采用一种适合于描述多种条件的组合,相应产生多个动作形式的方法来设计测试用例,这就需要采用因果图法。

采用因果图法能按照一定的步骤选择一组高效的测试用例,同时,还能指出程序规范描述中存在的问题。

因果图法最终生成的是判定表,适合于检查程序输入条件的各种组合情况。

2. 因果图的关系符号和约束

1)关系符号

(1)恒等。“恒等”关系符号如图3-1所示。

(2)非。“非”关系符号如图3-2所示。

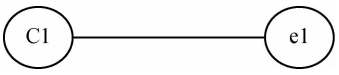

图3-1　“恒等”关系符号

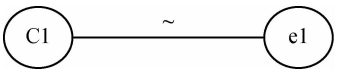

图3-2　“非”关系符号

(3)或。“或”关系符号如图3-3所示。

(4)与。“与”关系符号如图3-4所示。

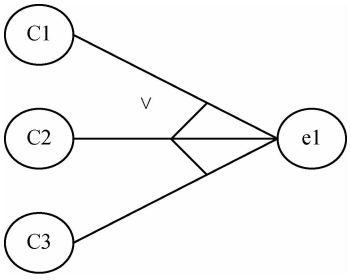

图3-3　“或”关系符号

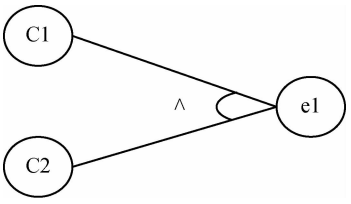

图3-4　“与”关系符号

对图的说明如下:

• 因果图中使用了简单的逻辑符号,以直线连接左右结点。左结点表示输入状态,右结点表示输出状态。

• C*i* 表示原因，通常置于图的左部；e*i* 表示结果，通常置于图的右部。C*i* 和 e*i* 均可取值 0 或 1，0 表示某状态不出现，1 表示某状态出现。

2)约束

输入状态之间还可能存在某些依赖关系，这种关系称为约束。例如，某些输入条件本身不可能同时出现。在因果图中使用特定的符号标明这些约束。

(1)E 约束符号。E 约束(异)是指：a 和 b 中至多有一个可能为 1，即 a 和 b 不能同时为 1。E 约束符号如图 3-5 所示。

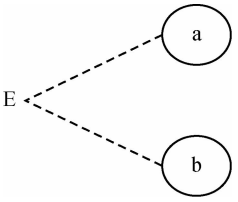

图 3-5 E 约束符号

(2)I 约束符号。I 约束(或)是指：a、b 和 c 中至少有一个必须是 1，即 a、b 和 c 不能同时为 0。I 约束符号如图 3-6 所示。

(3)O 约束符号。O 约束(唯一)是指：a 和 b 必须有且仅有一个 1，O 约束符号如图 3-7 所示。

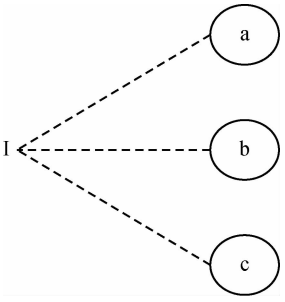

图 3-6 I 约束符号

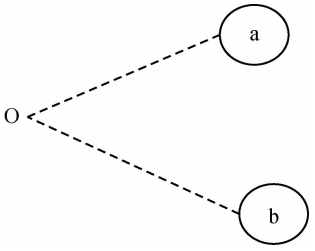

图 3-7 O 约束符号

(4)R 约束符号。R 约束(要求)是指：a 是 1 时，b 必须是 1，即不可能 a 是 1 时 b 是 0。R 约束符号如图 3-8 所示。

(5)M 约束符号。M 输出条件的约束(强制)是指：若 a 是 1，则 b 强制为 0。M 约束符号如图 3-9 所示。

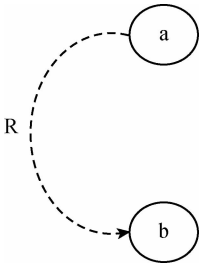

图 3-8 R 约束符号

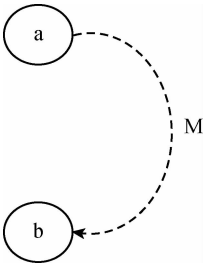

图 3-9 M 约束符号

3. 利用因果图导出测试用例的基本步骤

利用因果图导出测试用例的基本步骤如下：

(1)分析程序规范、规格说明描述中哪些是原因，哪些是结果，原因常常是输入条件或是输入条件的等价类；结果是输出条件，并给每个原因和结果赋予一个标识符。

(2)分析程序规范、规格说明描述中语义的内容，找出原因和结果之间、原因和原因之间的关系，根据这些关系画出因果图。

(3)在因果图上用一些记号标明约束或限制条件。

(4)把因果图转换为判定表。

(5)把判定表的每一列拿出来作为依据，设计测试用例。

4. 案例分析

某些软件规格说明书包含这样的要求：第一列字符必须是 A 或 B，第二列字符必须是一个数字，在此情况下进行文件的修改；如果第一列字符不是 A 或 B，则给出信息 L；如果第二列字符不是数字，则给出信息 M。

根据提议列出如下原因。

- 原因 1：第一列字符是 A。
- 原因 2：第一列字符是 B。
- 原因 3：第二列字符是一个数字。
- 原因 11：中间原因。

结果如下：

- 结果 21：修改文件。
- 结果 22：给出信息 L。
- 结果 23：给出信息 M。

对应的因果关系如表 3-4 所示。

表 3-4　因果关系

编　　号	原　　因	编　　号	结　　果
1	第一列字符是 A	21	修改文件
2	第一列字符是 B	22	给出信息 L
3	第二列字符是一个数字	23	给出信息 M
11	中间原因		

对应的因果关系如图 3-10 所示。

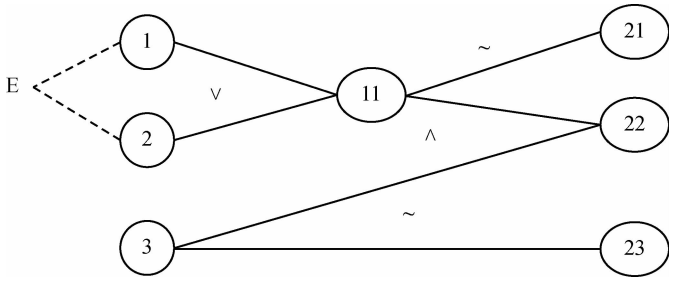

图 3-10　对应的因果关系

11 为中间原因。考虑到原因 1 和原因 2 不可能同时为 1，因此在因果图上施加 E 约束。

根据因果图建立的判定表如表 3-5 所示。

表 3-5 根据因果图建立的判定表

因/果 \ 组合情况		组合情况产生对应的动作							
		1	2	3	4	5	6	7	8
原　　因	1	1	1	1	1	0	0	0	0
	2	1	1	0	0	1	0	1	0
	3	1	0	1	0	1	0	1	0
	11			1	1	1	1	0	0
结　　果	22			0	0	0	0	1	1
	21			1	0	1	0	0	0
	23			0	1	0	1	0	1
测试用例				Y	Y	Y	Y	Y	Y

对表的说明如下。

- Y：测试用例。表的最下面一栏给出 6 种情况的测试用例。
- 3 列：结果 21＝1。
- 4 列：结果 23＝1。
- 5 列：结果 21＝1。
- 6 列：结果 23＝1。
- 7 列：结果 22＝1。
- 8 列：结果 22＝1，结果 23＝1。

按条件的各种组合情况产生对应的动作，在组合产生对应动作的 8 种情况中，原因 1 和原因 2 同时为 1 是不可能出现的，故应排除这两种情况。

3.2.4 决策表法

1. 决策表法简述

决策表又称判定表，是分析和表达多逻辑条件下执行不同操作情况的工具，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。因此，利用决策表设计的测试用例称为决策表驱动分析方法。在一些数据处理问题中，某些操作的实施依赖于多个逻辑条件的组合，即针对不同逻辑条件的组合值，分别执行不同的操作。决策表很适合于处理这类问题。

2. 决策表的组成

决策表通常由4个部分组成，如表3-6所示。

表3-6 决策表

条件桩	条件项
动作桩	动作项

对决策表的4个部分说明如下：

- 条件桩(condition stub)：列出了问题的所有条件。通常认为列出的条件的次序无关紧要。
- 动作桩(attion stub)：列出了问题规定可能采取的操作，这些操作的排列顺序没有约束。
- 条件项(cordithon enay)：列出针对条件桩排列的条件的取值。
- 动作项(althon 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称为规则。在决策表中贯穿条件项和动作项的一列就是一条规则，若有 n 个条件，每个条件有两个取值(0、1)，则有 2^n 条规则。显然，决策表中列出多少组条件取值，也就有多少条规则。

3. 规则合并

规则合并是指由两条或多条规则合并为一条规则。

规则合并如图3-11所示。

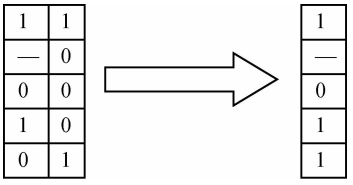

图3-11 规则合并

- 左端1、1，合并为右端“1”。
- 左端—、0，合并为右端“—”。
- 左端0、0，合并为右端“0”。
- 左端1、0，合并为右端“1”。
- 左端0、1，合并为右端“1”。

无关条件项“—”与取值无关，可包含其他条件项取值，具有相同动作的规则可合并。

与其他测试技术一样，基于决策表的测试对于某些应用程序(例如，NextDate函数)很有效，但是对另外一些应用程序就不值得耗费如此多的精力。毫不奇怪，基于决策表所适用的情况都是要发生大量决策(例如，三角形问题)以及在输入变量之间存在重要的逻辑关系的情况。

3.2.5 场景设计法

场景用来描述流经用例的路径，从用例开始到结束遍历这条路径上所有的基本流和备选流，如图3-12所示，图中经过用例的每条路径都用基本流和备选流来表示，直黑线表示基本流，是经过用例的最简单路径；备选流可以用不同的色彩表示，一个备选流可能从基本流

开始，在某个特定条件下执行，然后重新加入基本流中（如备选流 1 和 3）；也可能起源于某个备选流（如备选流 2），或者终止用例而不再重新加入到基本流（如备选流 2 和 4）。

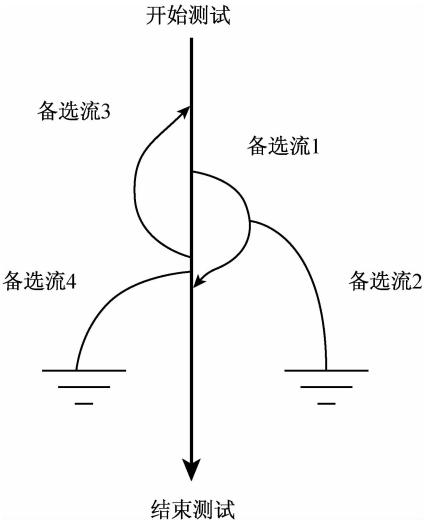

图 3-12　场景设计法的基本流和备选流

3.2.6　功能图分析法

功能图分析法是用功能图 FD 形式化地表示程序的功能说明，由状态迁移图和布尔函数组成。状态迁移图用状态和迁移来描述，一个状态指出数据输入的位置（或时间），而迁移则说明状态的改变，同时要依靠判定表或因果图表示的逻辑功能，机械地生成功能图的测试用例。

功能图模型由状态迁移图和逻辑功能模型构成。

- 状态迁移图用于表示输入数据序列以及相应的输出数据，在状态迁移图中，由输入数据和当前状态决定输出数据和后续状态。
- 逻辑功能模型用于表示在状态中输入条件和输出条件之间的对应关系，逻辑功能模型只适合于描述静态说明，输出数据仅由输入数据决定。

功能图分析法中需要用到逻辑覆盖和路径测试的概念及方法，这部分内容属于白盒测试的范畴，这里不再赘述。

3.2.7　正交试验法

正交试验法是依据 Galois 理论，从大量的（实验）数据中挑选适量的、有代表性的点（例），从而合理地安排实验（测试）的一种科学实验设计方法。

正交试验法是使用已经创建的正交表格来安排试验并运行数据分析的一种方法，目的是用最少的测试用例达到最高的测试覆盖率。

利用正交试验设计测试用例的步骤如下。

（1）提取功能说明，构造因子状态表：把影响试验指标的条件称为因子，而影响试验因子的条件称为因子的状态。利用正交试验法来设计测试用例时，首先要根据被测试软件的规格说明书找出影响其功能实现的操作对象和外部因素，把它们当做因子，而把各个因子的取值当做状态，对软件需求规格说明中的功能要求进行划分，把整体的概要性的功能要求进行层层

分解与展开，分解成具体的具有相对独立性的基本的功能要求，这样就可以把被测试软件中所有的因子都确定下来，并为确定各因子的权值提供参考依据。确定因子与状态是设计测试用例的关键，因此要求尽可能全面、正确地确定取值，以确保测试用例的设计完整、有效。

(2)加权筛选，生成因子分析表：对因子与状态的选择可按其重要程度分别加权，可根据各个因子及状态的作用大小、出现频率的大小、测试的需要确定权值的大小。

(3)利用正交表构造测试数据集。

利用正交试验法设计测试用例，与使用等价类划分、边界值分析、因果图等方法相比，有以下优点：节省测试工作工时；可控制生成的用例的数量；测试用例具有一定的覆盖性。

3.2.8 错误推测法

错误推测方法的基本思想是：利用直觉和经验猜测出错的可能类型，列举程序中所有可能的错误和容易发生错误的情况。其基本思想是列举出可能犯的错误或错误易发情况的清单，然后依据清单来编写测试用例，并且在阅读规格说明时联系程序员做的假设来确定测试用例。这种方法在很大程度上是凭经验进行的，即凭借测试人员对过去所做测试工作结果的分析，对所揭示的缺陷的规律性作直觉的推测。

3.3 黑盒测试的工具

目前，用于黑盒测试的工具软件很多，不同架构的软件工具不断推陈出新，这里介绍QACenter、QuickTest Professional、LoadRunner、TestDirector 等几个软件。

3.3.1 QACenter 介绍

QACenter 是 Compuware 公司研发的一款自动化的黑盒测试工具，它能帮助测试人员创建一个快速、可重用的测试过程。这一工具自动帮助管理测试过程，快速分析和调试程序，包括针对回归、强度、单元、并发、集成、移植、容量和负载建立测试用例，自动执行测试和产生文档结果。

QACenter 主要包括以下几个模块：

- QARun：功能测试模块。
- QALoad：性能测试模块。
- EcoTools：可用性管理模块。
- EcoScope：性能优化模块。
- TestBytes：测试数据自动生成模块。

1. 功能测试模块

在 QACenter 测试工具中，功能测试模块 QARun 主要用于 C/S、电子商务与企业资源规划(ERP)应用，提供企业级的功能测试，主要包括对应用的图形用户界面(GUI)的测试及对客户端事物逻辑的测试。QARun 组件的测试实现方式是：通过鼠标移动、键盘操作得到相应的测试脚本，再对该脚本进行编辑和调试。在记录的过程中可针对被测应用中所包含的功能点建立期望值，也就是说，插入检查点在 QARun 提示目标系统执行一系列时间之后被执行。检查点用于确定实际结果与期望结果是否相同。

2. 性能测试模块

性能测试模块 QALoad 是企业范围的负载测试工具。性能测试模块支持的范围广，测试的内容多，可以帮助软件测试人员、开发人员和系统管理人员对分布式的应用执行有效的负载测试。负载测试能够模拟大批量用户的活动，从而发现大量用户在负载下对 C/S 系统的影响。

性能测试模块的主要特点如下：

- 操作简便。测试人员只需操作被测应用，执行关键性能的事物处理，然后在 QALoad 脚本中通过服务器上应用调用的需求类型，开发这些事务处理即可。
- 广泛的实用性。QALoad 支持 DB2、DCOM、ODBC、Oracle、ONETLoad、Corba、QARun、SAP、SQL Server、Sybase、Tuxedo、Uniface、WinSock 等。

3. 可用性管理模块

可用性管理模块 EcoTools 提供一个广泛范围打包的 Agent 和 Scenarios，可以立即在测试或生产环境中激活、计划以商务为中心的可用性管理。EcoTools 支持一些主流成型的应用，如 SAP、Baan、Oracle、Uniface、LotusNotes 以及定制的应用。可用 EcoTools 监控服务器性能和服务器资源，尤其是监控 Windows NT、UNIX、Oracle、Sybase、SQL Server 和其他应用包。通过使用 QALoad 与 EcoTools 可以在系统中生成一个负载，同时监控资源的利用情况。

4. 性能优化模块

性能优化模块 EcoScope 是一套定位于应用(即服务提供者本身)及其所依赖的所有网络计算资源的解决方案。EcoScope 可以提供应用视图，并标出应用是如何与基础架构相关联的。EcoScope 使用综合软件探测技术无干扰地监控网络，可自动发现应用、跟踪在 LAN/WAN 上的应用流量，采集详细的性能指标。EcoScope 将这些信息关联到一个交互式用户界面中，自动识别低性能的应用、受影响的服务器与用户、性能下降的程度，能自动调整应用和定位基础架构上的缺陷。

5. 测试数据自动生成模块

测试数据自动生成模块 TestBytes 是一个用于自动生成测试数据的工具，为测试应用程序对数据库的访问自动生成测试数据。通过简单的点击式操作，就可以确定需要生成的测试的数据类型(包括特殊字符的定制)，并通过与数据库的连接来自动生成数百万行的正确的测试数据，从而提高数据库开发人员、测试人员、应用开发人员的工作效率。

3.3.2 QuickTest Professional

1. QTP 简述

QTP 是 QuickTest Professional 的简称，是 MI 公司继 WinRunner 之后开发的一款功能测试工具。它能够测试 Windows 标准应用程序、各种 Web 对象、ActiveX 控件、Visual Basic应用程序等，同时也会将应用程序的所有操作都记录下来，并且可以在录制完毕之后，对脚本进行编辑整理以使其功能更加强大。

2. QTP 的工作流程

QTP 的工作流程如下：

(1)录制测试脚本前的准备。预先检查应用程序及 QuickTest 是否符合测试需求，确认

如何对应用程序进行测试，如测试哪些功能、操作步骤、预期结果等，同时也要检查 QuickTest 的设定，以确保 QuickTest 能正确地录制并储存信息。确认 QuickTest 的存储模式。

(2)录制测试脚本。操作应用程序或浏览网站时，QuickTest 会在 Keyword View 中以表格的方式显示录制的操作步骤。每一个操作步骤都是使用者在录制时的操作，如在网站上单击链接或在文本框中输入信息。

(3)加强测试脚本。在测试脚本中加入检查点，可以检查网页的链接、对象属性或字符串，以验证应用程序的功能是否正确。使用逻辑或者条件判断式，可以进行更复杂的测试。

(4)对测试脚本进行调试。修改过测试脚本后，需要对测试脚本作调试，以确保测试脚本能正常并且流畅地执行。

(5)执行测试脚本。通过执行测试脚本，QuickTest 会在网站或者应用程序上执行测试，检查应用程序的功能是否正确。

(6)分析测试结果。分析测试结果，找出问题所在。

3. QTP 进行功能测试的主要步骤

QTP 进行功能测试的测试流程如图 3-13 所示。

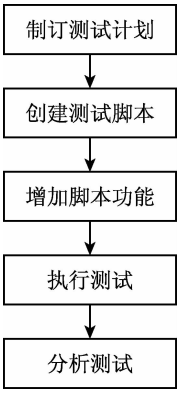

图 3-13　QTP 测试流程图

对图 3-13 的说明如下：

(1)制订测试计划。自动测试的测试计划是根据被测项目的具体需求以及所使用的测试工具而制订的，它指导测试全过程。

(2)创建测试脚本。当测试人员浏览站点或在应用程序上操作时，QTP 的自动录制机制能够将测试人员的每一个操作步骤及被操作的对象记录下来，自动生成测试脚本语句。

(3)增加脚本功能。录制脚本只是实现创建或者设计脚本的第一步，基本的脚本录制完毕后，测试人员可以根据需要增加一些扩展功能。QTP 允许测试人员通过在脚本中增加或更改测试步骤来修改或自定义测试流程，如增加多种类型的检查点功能，既可以让 QTP 检查在程序的某个特定位置或对话框中是否出现了需要的文字；也可以检查一个链接是否返回了正确的 URL 地址；还可以通过参数化功能，使用多组不同的数据驱动整个测试过程；等等。

(4)执行测试。QTP 从脚本的第一行开始执行语句，运行过程中会对设置的检查点进行验证，用实际数据代替参数值，并给出相应的输出结构信息。测试过程中测试人员还可以调试自己的脚本，直到脚本完全符合要求。

(5)分析测试。运行结束后系统会自动生成一份详细的测试结果报告。

3.3.3 LoadRunner

LoadRunner 是一种预测系统行为和性能的工业标准级负载测试工具，以模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题，并且能够对整个企业架构进行测试。通过使用 LoadRunner，企业能最大限度地缩短测试时间，优化性能，缩短应用系统的发布周期。

目前，企业的网络应用环境都必须支持大量用户，网络体系架构中含各类应用环境，且由不同供应商提供软件和硬件产品。难以预知的用户负载和越来越复杂的应用环境使公司时时担心会发生用户响应速度过慢、系统崩溃等问题，这些问题都不可避免地导致公司收益的损失；而 LoadRunner 能让企业保护自己的收入来源，无需购置额外硬件并最大限度地利用现有的软件资源，确保终端用户在应用系统的各个环节中对负载测试应用的质量、可靠性和可扩展性都有良好的评价。LoadRunner 是一种适用于各种体系架构的自动负载测试工具，能预测系统行为，并优化系统性能。

1. 轻松创建虚拟用户

使用 LoadRunner 的 Virtual User Generator 能很简便地建立起系统负载。该引擎能够生成虚拟用户，以虚拟用户的方式模拟真实用户的业务操作行为。它先记录下业务流程（如下订单或预订机票），然后将其转化为测试脚本。利用虚拟用户，可以在 Windows、UNIX 或 Linux 上同时产生成千上万个用户访问。所以 LoadRunner 能极大地减少负载测试所需的硬件和人力资源。

2. 创建真实的负载

虚拟用户建立后，需要设定负载方案、业务流程组合和虚拟用户数量。用 LoadRunner 的 Controller 功能，能很快组织起多用户的测试方案。Controller 的 Rendezvous 功能提供一个互动的环境，在其中既能建立起持续且循环的负载，又能管理和驱动负载测试方案。同时，还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，就能将测试过程自动化。

LoadRunner 通过它的 AutoLoad 技术，提供更多的测试灵活性。使用 AutoLoad，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，设定的目标可以确定应用系统承受的每秒点击数或每秒的交易量。

3. 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，可以监测应用系统的运行性能。这些性能监测器能实时显示交易性能和其他系统组件实时性能的数据。这样，相关人员就可以在测试过程中从客户和服务器的双方面评估系统组件的运行性能，从而更快地发现问题。

利用 LoadRunner 的 ContentCheck TM，可以判断负载下的应用程序功能是否正常。ContentCheck TM 在虚拟用户运行时，检测应用程序的网络数据包内容，从中确定是否有错误内容传送出去。

4. 分析结果

一旦测试完毕后，LoadRunner 就收集汇总所有的测试数据，并提供高级的分析和报告

工具,以便用户迅速查找到性能问题并追溯缘由。使用 LoadRunner 的 Web 交易细节监测器,用户可以了解到将所有的图像、框架和文本下载到每一网页上所需的时间。例如,这个交易细节分析机制能够分析是否因为一个大尺寸的图形文件或第三方的数据组件而造成应用系统运行速度减慢。另外,Web 交易细节监测器能够分解用于客户端、网络和服务器上端到端的反应时间,这就便于确认问题,定位查找真正出错的组件。

5. 重复测试

负载测试是一个重复过程。每次处理完一个出错情况,都需要对应用程序在相同的方案下再进行一次负载测试,以此检验所做的修改是否改善了运行性能。

6. Enterprise Java Beans

LoadRunner 完全支持 Enterprise Java Beans(EJB)的负载测试。这些基于 Java 的组件运行在应用服务器上,提供广泛的应用服务。通过测试这些组件,可以在应用程序开发的早期就确认并解决可能产生的问题。

利用 LoadRunner 可以很方便地了解系统的性能。它的 Controller 功能允许重复执行与出错修改前相同的测试方案。它的基于 HTML 的报告为我们提供了一个比较性能结果所需的基准,以此衡量在一段时间内,有多大程度的改进并确保应用成功。由于这些报告是基于 HTML 的文本,因此可以将其公布于公司的内部网站上,便于随时查阅。

7. 支持无线应用协议

随着无线设备数量和种类的增多,测试计划需要同时满足传统的基于浏览器的用户和无线互联网设备,如手机和 PDA。LoadRunner 支持两项最广泛使用的协议:WAP 和 I-mode。此外,通过负载测试系统整体架构,LoadRunner 能让用户只需通过记录一次脚本,就可完全检测上述无线互联网系统。

3.3.4 TestDirector

1. 概述

TestDirector 是 Mercury Interactive 公司推出的基于 Web 的测试管理工具,无论是通过 Internet 还是通过 Intranet 都可以以基于 Web 的方式来访问 TestDirector。

TestDirector 是 B/S 结构的软件,只需要在服务器端安装软件,所有的客户端就可以通过浏览器来访问 TestDirector,方便测试人员的团队合作和沟通交流。表 3-7 中列出了其所具有的主要功能模块。

表 3-7 TestDirector 的主要功能模块及其说明

功能模块	说 明
需求管理	定义测试要求,包括定义测试的内容、主题和条目并分析这些要求
测试计划	开发测试计划,包括定义测试目标和策略、将测试计划分为不同的类别、对测试进行定义和开发、定义哪些需要自动化测试、将测试与需求进行连接、分析测试计划
测试执行	运行测试并分析结果
缺陷管理	增加新缺陷、确定缺陷修复属性、修复打开的缺陷、分析缺陷数据

程序测试是一个非常复杂的过程，它需要开发和执行数以千计的测试用例。通常情况下，测试需要多样式的硬件平台、多重的配置（计算机、操作系统、浏览器）和多种应用程序版本。管理整个程序测试过程中的各个部分是非常耗时和困难的。

TestDirector 能够使用户系统地控制整个测试过程，并创建整个测试工作流的框架和基础，使整个测试管理过程变得更为简单和有组织。

TestDirector 能够帮助用户维护一个测试工程数据库，并且能够覆盖应用程序功能性的各个方面，在工程中的每一个测试点都对应着一个指定的测试需求。它还提供了直观而有效的方式来计划和执行测试集，收集测试结果并分析数据。

TestDirector 还专门提供了一个完善的缺陷跟踪系统，它能够跟踪缺陷从产生到最终解决的全过程。TestDirector 通过与邮件系统相关联，将缺陷跟踪的相关信息共享给整个应用开发组、QA、客户支持、负责信息系统的人员等。

2. TestDirector 测试流程

1）总体管理流程

TestDirector 的测试管理共有 4 个流程，如图 3-14 所示。各阶段含义如下所述：

- 需求定义（specify requirements）：分析应用程序并确定测试需求。
- 测试计划（test plan）：基于测试需求，建立测试计划。
- 测试执行（execute test）：创建测试实例并执行测试。
- 缺陷跟踪（track defects）：缺陷跟踪和管理，并生成测试报告和各种测试统计图表。

2）确认需求阶段的流程

确认需求阶段可以进一步分解为 4 个环节，如图 3-15 所示。

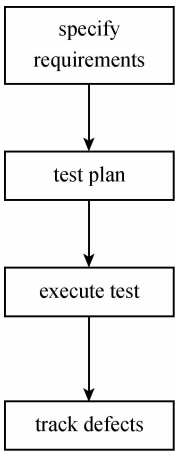

图 3-14　TestDirector 的测试管理流程

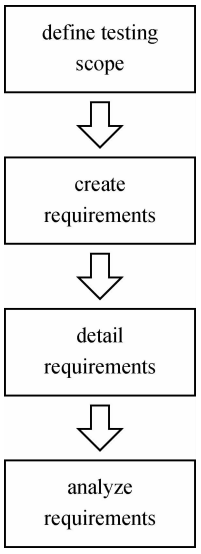

图 3-15　确认需求的流程图

各环节的具体含义解释如下：

- 定义测试范围（define testing scope）：检查应用程序文档，并确定测试范围（测试目的、目标和策略）。
- 创建需求（create requirements）：创建“需求树”（requirements tree），并确定它涵盖所

有的测试需求。

- 描述需求(detail requirements):为“需求树”中的每一个需求主题建立了一个详细的目录,并描述每一个需求,给它分配一个优先级,如有必要,还可以加上附件。
- 分析需求(analyze requirements):产生报告和图表来帮助分析测试需求,并检查需求以确保它们在测试范围内。

3. 制订测试计划的流程

制订测试计划阶段又可进一步分为 7 个环节,如图 3-16 所示。各环节的具体含义如下:

- 定义测试策略(define testing strategy):检查应用程序、系统环境和测试资源,并确定测试目标。
- 定义测试主题(define testing subject):将应用程序基于模块和功能进行划分,并将其应用到各个测试单元或主题中,构建“测试计划树”(test plan tree)。
- 定义测试(define tests):定义每个模块的测试类型,并为每一个测试添加基本的说明。
- 创建需求覆盖(create requirement coverages):将每个测试与测试需求进行连接。
- 设计测试步骤(design testing steps):对于每个测试,先决定其要进行的测试类型(手动测试或自动测试),若准备进行手动测试,需要为其在“测试计划树”上添加相应的测试步骤。测试步骤描述测试的详细操作、检查点和每个测试的预期结果。
- 自动测试(automate test):对于要进行自动测试的部分,应该利用 MI、自己或第三方的测试工具来创建测试脚本。
- 分析测试计划(analyze test plan):产生报告和图表来帮助分析测试计划数据,并检查所有测试以确保它们达到测试目标。

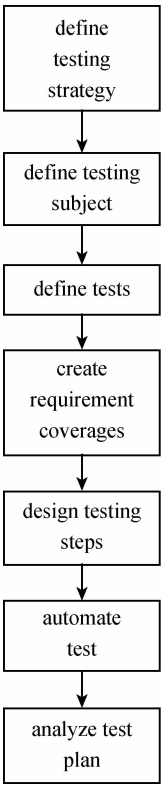

图 3-16 制订测试计划的流程

4. 执行测试的流程

执行测试阶段又可以进一步分解为 4 个环节,如图 3-17 所示。各环节的具体含义如下:

- 创建测试集(create test sets):创建测试集,一个测试集可以包含多个测试项。
- 确定进度表(schedule runs):为执行测试制作时间表,并为测试员分配任务。
- 运行测试(run tests):自动或手动执行每一个测试集。
- 分析测试结果(analyze test result):查看测试结果并确保应用程序缺陷已经被发现。生成的报告和图表可以帮助分析这些结果。

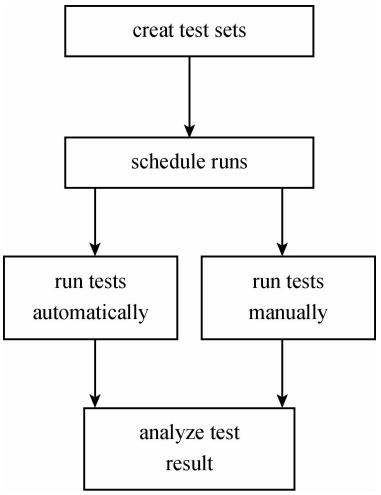

图 3-17 执行测试的流程

5. 缺陷跟踪的流程

缺陷跟踪阶段可分为 5 个环节，如图 3-18 所示。

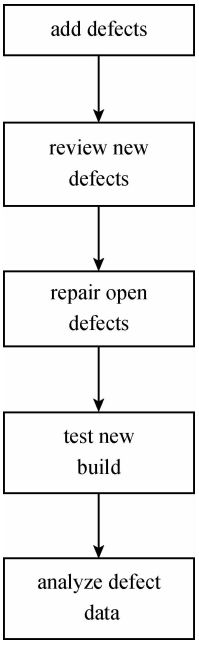

图 3-18 缺陷跟踪的流程

- 添加缺陷(add defects)：报告程序测试中发现的新的缺陷。在测试过程中的任何阶段，质量保证人员、开发者、项目经理和最终用户都能添加缺陷。
- 检查新缺陷(review new defects)：检查新的缺陷，并确定哪些缺陷应该被修复。
- 修复打开的缺陷(repair open defects)：修复那些决定要修复的缺陷。
- 测试新构建(test new build)：测试应用程序的新构建，重复上面的过程，直到缺陷被修复。

- 分析缺陷数据(analyze defect data):产生报告和图表来帮助分析缺陷修复过程,并帮助决定什么时候发布该产品。

习 题 3

1. 功能测试可以在(　　)执行。

A. 系统级　　B. 单元级

C. 系统级和单元级　　D. 以上都不是

2. 黑盒测试一般从(　　)执行。

A. 程序员的观点　　B. 设计人员的观点

C. 最终用户的观点　　D. 以上都不是

3. 边界值分析法是(　　)。

A. 等价类划分法的细化　　B. 等价类划分法的扩展

C. 等价类划分法的细化和扩展　　D. 以上都不是

4. (　　)的目标之一是在非法条件下引起软件失败,从而检测潜在缺陷。

A. 回归测试　　B. 系统测试

C. 集成测试　　D. 以上都不是

5. 等价类划分法只要求选择(　　)。

A. 一个测试用例　　B. 一个或者多个测试用例

C. 无穷数量的测试用例　　D. 以上都不是