1 模块 1 软件测试基础知识

软件开发的基本要求是按时并高质量地发布软件产品，而软件测试是保证软件质量的重要手段之一。不论采用什么技术和方法来进行软件开发，软件产品中仍然或多或少地存在错误。采用先进的开发方式和较完善的开发流程可以减少错误的引入，但是不会根除软件中的错误，这些引入的错误需要通过测试来发现。软件测试伴随着软件开发，以检验每一个阶段性的成果是否符合质量要求和达到预先定义的目标，以尽早地发现错误并及时修正。

1.1 软件测试的发展简介

在信息化时代，人们的手机、计算机上都安装有各种各样的软件。软件是人们工作和学习的好助手，我们平时常用的软件包括 Windows 操作系统、Office 办公软件、QQ、微信等。软件是通过软件开发流程开发出来的产品，它包括计算机系统运行的指令、数据和相关文档的集合，即软件等于程序、数据加上文档。程序是事先按照预定功能、性能等要求设计和编写的指令序列；数据是使程序正常处理信息的数据结构及信息表示；文档是与程序开发、维护和使用有关的技术数据和图文资料。

软件测试是伴随着软件的产生而产生的。早期的软件规模都很小、复杂程度低，软件开发的过程混乱无序、相当随意，测试的含义比较狭窄，开发人员将测试等同于“调试”，目的是纠正软件中已知的故障并，常常由开发人员自己完成这部分工作。对测试的投入极少，测试介入也较晚，常常是等到形成代码，产品已经基本完成时才进行测试。

到了 20 世纪 80 年代初期，IT 行业进入了大发展时代，软件趋向大型化、高复杂度，软件的质量越来越重要。这时，一些软件测试的基础理论和实用技术开始形成，并且人们开始

为软件开发设计了各种流程和管理方法，软件开发的方式也逐渐由混乱无序的开发过程过渡到结构化的开发过程，以结构化分析与设计、结构化评审、结构化程序设计及结构化测试为特征。人们还将“质量”的概念融入其中，软件测试定义发生了改变，测试不是一个单单发现错误的过程，而是将测试作为软件质量保证（SQA）的主要职能，包含软件质量评价的内容，Bill Hetzel 在《软件测试完全指南》（*Complete Guide of Software Testing*）一书中指出：“测试是以评价一个程序或者系统属性为目标的任何一种活动。测试是对软件质量的度量。”这个定义至今仍被引用。

软件测试已有了行业标准（IEEE/ANSI），1983 年，IEEE 提出的软件工程术语中给软件测试下的定义是：“使用人工或自动手段来运行或测试某个系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别。它是帮助识别开发完成（中间或最终版本）的计算机软件（整体或部分）的正确度、安全度和质量的软件过程”。这个定义明确指出：软件测试的目的是为了检验软件系统是否满足需求。它再也不是一个一次性的、开发后期的活动，而是与整个开发流程融合成一体。软件测试已成为一个专业，需要运用专门的方法和手段，由专门人才来承担。

1.2 软件测试的目的和必要性

1.2.1 软件测试的目的

在 1.1 节的介绍中，我们已经知道在软件测试的定义中提到了软件测试的目的：检验软件被测对象是否满足规定的需求或弄清预期结果与实际结果之间的差别。如果从软件参与者的角度来看软件测试的目的性，则可以分为以下几点。

（1）从软件开发者角度来看，软件测试活动发现被测对象与用户需求之间的差异即缺陷，并解决存在的缺陷，增强对软件质量的信心。而且通过测试积累经验，预防后续缺陷的出现，提高开发效率。

（2）从软件项目管理者角度来看，通过测试活动了解被测对象的质量状况，为项目决策提供数据依据。

（3）从用户角度来看，通过软件测试中提供的数据，了解软件质量状况，为软件的评审和验收提供依据。

1.2.2 软件测试的必要性

基于上述对软件测试的认识，我们知道软件测试不仅仅是为了查找软件的错误，而是作为软件质量的保障手段。对于相对复杂的软件系统来说，开发中的零错误是很难做到的，所

以参考 Glenford J. Myers 的说法，软件测试的必要性包括以下几点。

（1）软件测试过程是为了发现程序中的错误。

（2）设计出好的测试方案和测试用例才能发现很难发现的错误。

（3）软件测试中通过分析错误产生的原因和错误的发生趋势，可以帮助项目管理者发现当前软件开发过程中的缺陷，以便及时改进。

（4）软件测试中对错误的分析也能帮助测试人员设计出有针对性的测试方法，提高测试的效率和有效性。

（5）没有发现错误的测试也是有价值的，完整的测试是评定软件质量的一种方法。

1.3 软件测试的分类

软件测试发展至今，已经形成了一个庞大的系统工程，有许多常用的测试方法，如黑盒测试、白盒测试、自动化测试等，种类、名称十分庞杂。因此，需要对软件测试的方法进行类型上的划分，以便于明确测试过程、完成哪些操作步骤以及要达到的测试目标，尽量保证测试的完整性。按照不同的分类标准，我们可以将软件测试分成很多不同的种类，如图 1-1 所示。

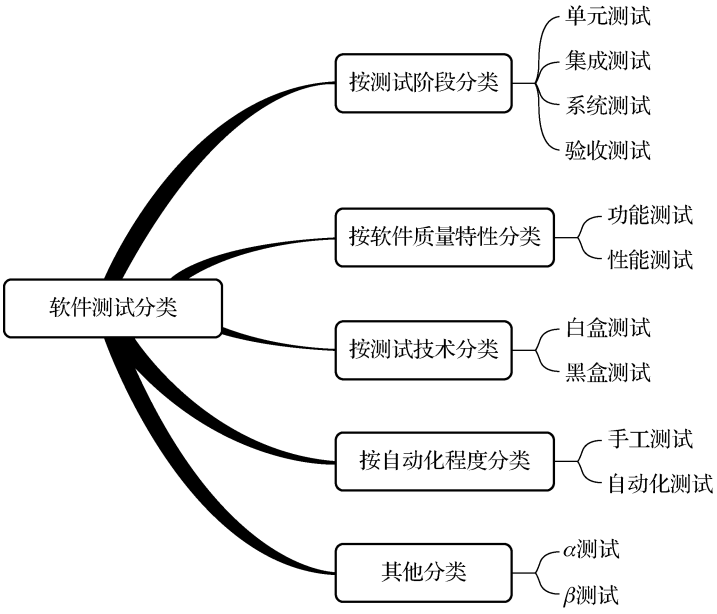

图 1-1　软件测试分类图

1. 按照测试阶段分类

按照测试阶段可以将软件测试分为单元测试、集成测试、系统测试和验收测试。这种分

类方式与软件开发过程相契合，它是为了检验软件开发各个阶段是否符合要求。

（1）单元测试。单元测试是软件开发的第一步测试，目的是验证软件单元是否符合软件需求与设计要求。单元测试大多是开发人员进行的自测。

（2）集成测试。集成测试是将已经被测试过的软件单元组合在一起以测试它们之间的接口，用于验证软件是否满足设计要求。在持续集成（Continuous Integration，CI）的场景中，常见的测试方式包括冒烟测试和回归测试。

①冒烟测试。冒烟测试最初是从电路板测试得来的，当电路板做好以后，首先，会加电测试，如果电路板没有冒烟再进行其他测试，否则就必须重新设计后再进行测试。后来这种测试理念被引入软件测试中。在软件测试中，冒烟测试往往是软件每日版本构建后（daily build），对系统的基本功能进行简单的测试，重点验证的是新版本是否存在致命性错误，而不会对具体功能进行深入测试。若测试未通过，则需要将程序返回给开发人员进行修正；若测试通过，则再进行其他测试。因此，冒烟测试是对新构建版本软件进行的最基本的测试。

②回归测试。回归测试是指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误。回归测试在整个软件测试过程中占有很大的工作量比重，在渐进和快速迭代开发中，新版本的连续发布使回归测试进行得更加频繁。因此，选择合适的回归测试策略来提高回归测试的效率和改进回归测试的有效性是很有意义的。回归测试可能采取的策略包括全部重复和选择性重复。

（3）系统测试。系统测试是在完成集成测试之后，把软件部分与系统的其他组成部分，包括硬件、外部软件、操作人员等结合起来，在实际运行或模拟环境下对计算机系统进行的一系列严格有效的测试，以发现潜在的问题。系统测试包括包括恢复测试、安全测试、压力测试等类型。

（4）验收测试。验收测试主要是对软件产品说明进行验证，逐行逐字地按照说明书的描述对软件产品进行测试，以确保其符合用户的各项要求。

2. 按照测试技术分类

按照使用的测试技术可以将软件测试分为黑盒测试和白盒测试，如图 1-2 所示。

（1）黑盒测试就是把软件（程序）当作一个有输入和输出的黑匣子，它把程序当作一个输入域到输出域的映射，只要输入的数据能输出预期的结果即可，不必关心程序内部是如何实现的。

（2）白盒测试又称透明盒测试，它是指测试人员了解软件程序的逻辑结构、路径与运行过程，在测试时，按照程序的执行路径得出结果。白盒测试就是把软件（程序）当作一个透明的盒子，测试人员清楚地知道从输入到输出的每一个过程。

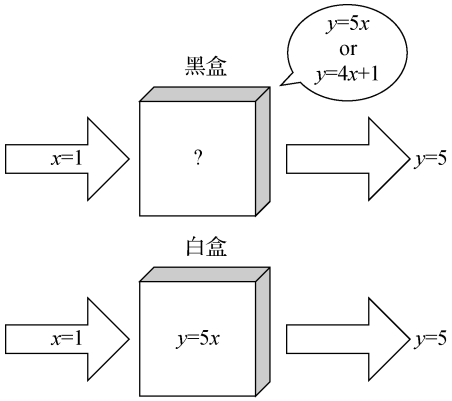

图 1-2　黑盒测试和白盒测试

相对于黑盒测试来说，白盒测试对测试人员的要求会更高一些，它要求测试人员具有一定的编程能力，而且要熟悉各种脚本语言。但是在软件公司里，黑盒测试与白盒测试并不是界限分明的，在测试一款软件时往往是黑盒测试与白盒测试相结合对软件进行完整、全面的测试。

3. 按照软件质量的特性分类

按照软件质量特性可以将软件测试分为功能测试和性能测试。

（1）功能测试就是测试软件的功能是否满足用户的需求，包括准确性、易用性、适合性、互操作性等。

（2）性能测试就是测试软件的性能是否满足用户的需求，包括负载测试、压力测试、兼容性测试、可移植性测试、健壮性测试等。

4. 按照自动化程度分类

按照自动化程度可以将软件测试分为手工测试和自动化测试。

（1）手工测试是测试人员一条一条地执行代码，完成测试工作。手工测试比较耗时费力，而且测试人员在疲惫状态下很难保证测试的效果。

（2）自动化测试是借助脚本、自动化测试工具等完成相应的测试工作，它也需要人工参与，但是可以将要执行的测试代码或流程写成脚本，从而执行脚本来完成整个测试工作。

5. 其他分类

在测试行业中也会经常进行 α 测试、β 测试等。具体介绍如下。

（1）α 测试是指对软件最初版本进行的测试。软件最初版本一般不对外发布，在上线之前，由开发人员和测试人员或用户协助进行测试。测试人员记录使用过程中出现的错误与问题，整个测试过程都是可控的。

（2）β 测试是指对上线之后的软件版本进行测试，此时软件已上线发布，但发布的版本

中可能会存在轻微的 bug，由用户在使用软件过程中发现错误和问题并记录下来，然后反馈给开发人员进行修复。

1.4 常见的软件测试模型

软件测试和软件开发一样，都遵循软件工程原理。测试专家通过实践总结出了很多很好的测试模型。这些模型对测试活动进行了抽象，明确了测试与开发之间的关系，是软件测试管理的重要参考依据。下面介绍几种比较有代表性的模型。

1. 瀑布模型

传统的瀑布模型包括项目计划—需求分析—软件设计—程序开发—软件测试—集成维护这几个阶段。由于这个模型难以适应频繁变化的需求，现在已经不常用，但是模型中这几个阶段都很有代表意义，对后面的模型有一定的参考价值。

（1）项目计划阶段：主要是制订项目总体研发计划，树立项目里程碑节点，输出项目计划书。

（2）需求分析阶段：明确用户的需求定义，并对这个定义进行清晰的描述，是充分理解用户需求，描述产品功能的重要阶段，这个阶段会输出产品的需求规格说明书。

（3）软件设计阶段：会根据需求的定义来确定产品实现的方案，包括软件和硬件的结构定义，组建模块的实现方法，接口、界面、数据的组织方式，这个阶段会输出包括概要设计、详细设计在内的多份说明书。

（4）程序开发阶段：这个阶段是开发团队根据需求定义和设计要求具体实现产品，根据编程规范，构建组件模块，最后输出产品版本。

（5）软件测试阶段：这个阶段是通过独立的测试小组或 QA 团队评估产品是否满足需求定义，最后输出测试结果报告。

（6）集成维护阶段：产品经过测试后，交付给用户使用，开发人员根据用户的使用情况，对产品进行维护，并进行必要的修改、升级等操作。

2. V 模型

V 模型使用范围最广，它是从瀑布模型演化而来的，包括需求分析—概要设计—详细设计—软件编码—单元测试—集成测试—系统测试—验收测试这几个阶段，如图 1-3 所示。

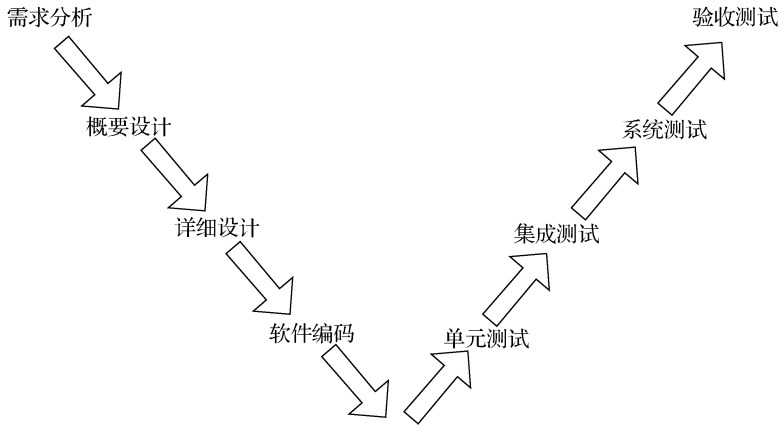

图 1-3　V 模型

V 模型按图 1-3 中箭头的方向，指出了不同阶段的顺序和依赖性。每个阶段的分工和解决的问题不同：

(1)单元测试和集成测试：检测程序是否满足设计要求。

(2)系统测试：在功能、性能这些质量特性上是否满足系统要求的指标。

(3)验收测试：确定软件是否满足用户的需求及合同的规定。

3. W 模型

W 模型其实是双 V 模型，它的设计思想是并发执行软件测试与软件开发。开发与测试并行，有利于尽早发现问题，有利于及时了解项目的测试风险，及早地执行相应的应对方案，加快项目的开发进度。W 模型存在的缺点是整个开发阶段仍然是串行的，上一阶段未完全完成，无法进入下一阶段，不支持敏捷模式的开发。W 模型如图 1-4 所示。

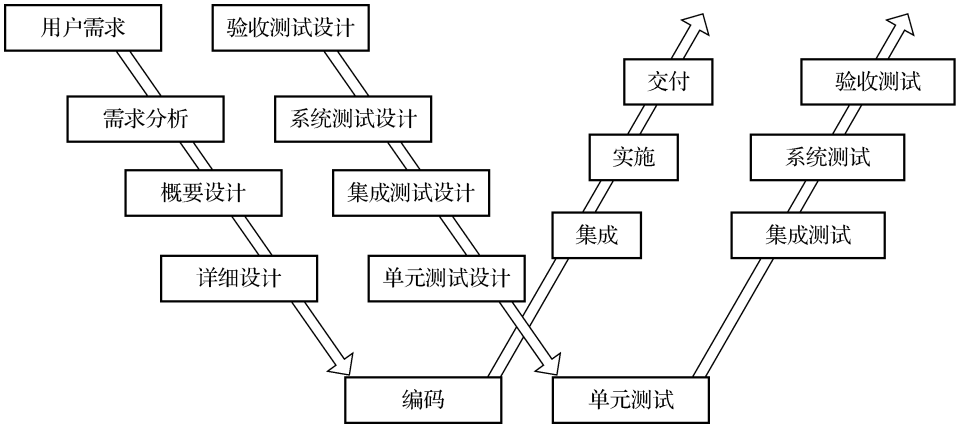

图 1-4　W 模型

4. X 模型

X 模型的基本思路是分离出多个程序片段，并对各片段独立地进行编码和测试，此后进行频繁的交接，再通过集成，最终合成可执行的程序。已经通过测试的程序可以进行封版提交给用户，也可以作为更大集成的一部分。X 模型的右下部还定位了探索式测试，探索式测试是不进行事先计划的特殊类型的测试，能够帮助测试人员在测试计划之外发现更多的错误。X 模型如图 1-5 所示。

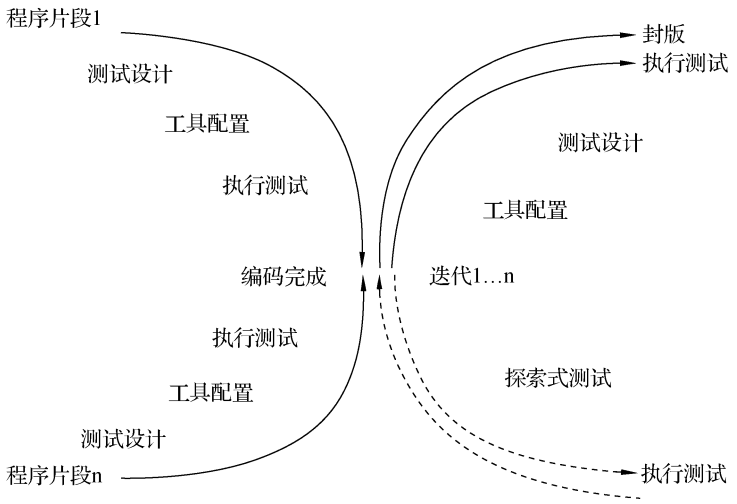

图 1-5　X 模型

5. H 模型

H 模型强调把测试分为测试准备和测试执行两个阶段，如果其他流程的进展使测试点准备就绪，测试执行活动就可以进行。在 H 模型中，测试是一个完全独立的模型，所以可以和其他流程交叉地进行，也便于尽早地执行测试。H 模型如图 1-6 所示。

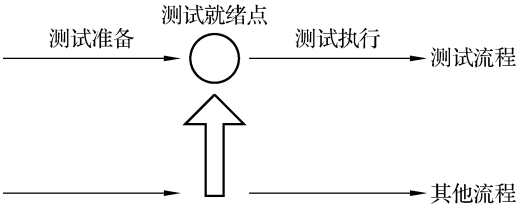

图 1-6　H 模型

1.1 软件测试的原则

软件测试经过了多年的实践积累，人们为了尽可能地发现软件中的错误，提高软件产品的质量，总结出了一些基本原则用于指导软件测试工作。测试人员如果能把握好测试原则，有助于提高测试工作的效率和质量，以最少的时间、精力发现软件中存在的问题。下面介绍一下业界公认的六个基本原则。

1. 测试应基于用户需求

所有的测试工作都应该建立在满足用户需求的基础上，从用户的角度来看，最严重的错误就是软件无法满足需求。应依照用户的需求配置环境，依照用户的使用习惯进行测试并评价结果。如果系统不能完成用户的需求和期望，那么这个系统的研发就是失败的。同时，在系统中发现和修改缺陷也是没有任何意义的。在开发过程中用户的早期介入和接触原型系统就是为了避免这类问题发生的预防性措施。

2. 测试要尽早进行并不断迭代

由于软件的复杂性和抽象性，在软件生命周期各阶段都可能产生错误，所以不应把软件测试仅仅看作软件开发的一个独立阶段，而应当把它贯穿到软件开发的各个阶段中去。在需求分析和设计阶段就应开始进行测试工作，编写相应的测试计划及测试设计文档，同时坚持在开发各阶段进行技术评审和验证，这样才能尽早发现和预防错误，杜绝某些缺陷和错误，提高软件质量。尽早开展测试准备工作以使测试人员能够在早期了解到测试的难度，预测测试的风险，有利于开发人员制定出完善的计划和方案，提高软件测试及开发的效率，规避测试中存在的风险。尽早开展测试工作，有利于测试人员尽早发现软件中的缺陷，大大降低错误修复的成本。测试工作进行得越早，越有利于提高软件的质量，这是预防性测试的基本原则。

3. 穷举测试是不可能的

由于时间和资源的限制，穷举测试（各种输入和输出的全部组合）是不可能的，测试人员可以根据测试的风险和优先级控制测试的工作量，在测试成本、风险和收益之间求得平衡。此外，应避免冗余测试。

4. 遵循 GoodEnough 原则

GoodEnough 原则是一种权衡投入/产出比的原则：不充分的测试是不负责任的，而过分的测试则是一种资源的浪费，同样也是一种不负责任的表现。测试不充分无法保证软件

产品的质量，但测试投入过多会造成资源的浪费。随着测试资源投入的增加，测试的产出也是增加的，但当投入达到一定的比例后，测试的效果就不会明显增强了。这个原则实施的困难之处在于如何界定什么样的测试是不充分的，什么样的测试是过分的。针对这种情况，测试人员最好制定最低测试通过标准和测试内容，然后具体问题具体分析。

5. 缺陷要符合“二八”定理

缺陷的“二八”定理就是一般情况下，软件 80％的缺陷会集中在 20％的模块中，缺陷并不是平均分布的。存在这种现象的原因是：对一个程序模块进行测试，已发现的错误数越多，其中存在的错误概率也就越大。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。因此，在测试时要抓住主要矛盾，如果发现某些模块比其他模块具有更多的缺陷，则要投入更多的人力重点测试这些模块以提高测试效率。

6. 杀虫剂悖论

杀虫剂用得多了，害虫就有免疫力，杀虫剂就发挥不了效力。在测试中，同样的测试用例被一遍遍反复使用时，发现缺陷的能力就会越来越差。这种现象的主要原因在于测试人员没有及时更新测试用例，同时对测试用例及测试对象过于熟悉，形成思维定势。

要克服这种情况，就要不断对测试用例进行修改和评审，不断增加新的测试用例，这样软件中未被测试过的部分或先前没有被使用过的输入组合就会被重新执行，从而发现更多的缺陷。同时，测试人员也要有探索性思维和逆向思维，不能只是做输出与期望结果的比较。

1.6 软件测试的一般流程

软件测试流程没有完全统一的标准，允许存在个性化差异。不同类型的软件产品测试的方式和重点不一样，测试流程也会不一样。即使是同类型的软件产品，不同的公司所制定的测试流程也会有些差异。尽管如此，软件测试所遵循的基本测试流程总体是一样的：评测测试需求——制订测试计划——设计测试用例——执行测试——编写测试报告。

1.6.1 评测测试需求

评测测试需求并不是需求分析，而是测试人员在制订测试计划之前先对软件需求规格说明书进行评测，按检查项逐项检查和判断，从而得出前期需求分析的过程和结果的评价，活动产出的结果是一张检查表。检查表可以对需求说明的正确性、完整性和清晰性，以及其他指标给予评测，测试人员从中发现不合理的地方要加以改进，有些需求不明确的地方，需

要尽早与用户进行沟通，消除分歧。明确测试对象及测试工作的范围和测试重点，为后续制订测试计划做好准备。

测试人员一般会根据软件开发需求文档制作一个软件需求规格说明书检查清单，按照各个检查项对软件需求进行分析校验，见表 1-1。

表 1-1　软件需求规格说明书检查列表

序号	检查项	检查结果	说明
1	系统定义的目标是否与用户的要求一致	是【】否【】NA【】	
2	系统需求分析阶段提供的文档资料是否齐全	是【】否【】NA【】	
3	文档中的所有描述是否完整、清晰、准确地反映了用户要求	是【】否【】NA【】	
4	与其他系统成分的重要接口是否都已经描述	是【】否【】NA【】	
5	被开发项目的数据流与数据结构是否足够、确定	是【】否【】NA【】	
6	所有图表是否清晰，在不补充说明时能否被理解	是【】否【】NA【】	
7	主要功能是否已被包括在规定的软件范围之内，是否都已充分说明	是【】否【】NA【】	
8	软件的行为和它必须处理的信息、必须完成的功能是否一致	是【】否【】NA【】	
9	设计的约束条件或限制条件是否符合实际	是【】否【】NA【】	
10	是否考虑了开发的技术风险	是【】否【】NA【】	
11	是否考虑过软件需求的其他方案	是【】否【】NA【】	
12	是否考虑过将来可能会提出的软件需求	是【】否【】NA【】	
13	是否详细制定了校验标准，它们能否对系统定义是否成功进行确认	是【】否【】NA【】	
14	有没有遗漏、重复或不一致的地方	是【】否【】NA【】	
15	用户是否审查了初步的用户手册或原型	是【】否【】NA【】	
16	项目开发计划中的估算是否受到了影响	是【】否【】NA【】	

表 1-1 列出了需要对软件需求进行什么样的检查，测试人员按照检查项逐条检查和判断，如果满足要求，则在“是”的右侧框中打勾，如果不满足要求，则在“否”右侧框中打勾，如果某个检查项不适用，则在“NA”右侧框中打勾。表 1-1 只是一个通用的软件需求规格说明书检查列表。在实际测试中，要根据具体的测试项目进行适当的增减或修改。

1.6.2　制订测试计划

测试工作是软件开发过程中不可缺少的组成部分，在软件开发工作的开始阶段就要制

订好软件测试计划，作为软件开发的保障性工作，它涉及的内容很多，而且不是一成不变的，随着项目推进或需求变更，测试计划也会不断地发生改变，因此，测试计划的制订是一个随着项目发展不断调整、逐步完善的过程。

测试计划一般要包括以下内容：

（1）确定测试内容：列出包括哪些子系统及下级各个模块，这些模块的测试点及优先级都要进行准确说明。

（2）制定测试规则：设计测试活动的前置进入准则、暂停/退出准则，描述采取的测试流程（如黑盒或白盒），采用的测试手段（如手工测试、自动测试或两者相结合），测试要点、测试工具等。

（3）设定测试环境：描述测试的软件、硬件、网络通信、安全性要求及其他可能的环境需求。

（4）安排测试任务：根据软件开发计划、产品的整体计划来安排测试工作的进度，同时还要考虑各部分工作的变化。在安排工作进度时，最好在各项测试工作之间预留一个缓冲时间以应对计划变更。

（5）计划实施：列出系统各测试阶段所使用的资源及资源如何安排。

（6）风险管理：详细描述测试所面临的各类风险（包括人力资源、测试技术、测试资源、质量保障）及相应的建议及解决办法。

本书“8.4 制订测试计划”给出了测试计划的示例，供读者参考使用。

1.6.3 设计测试用例

测试用例是套详细的测试方案，包括测试环境、测试步骤、测试数据和预期结果。不同的公司会有不同的测试用例模板，虽然它们在风格和样式上有所不同，但本质上是一样的，都包括了测试用例的基本要素。

测试用例编写的原则是尽量以最少的测试用例达到最大的测试覆盖率。测试用例常用的设计方法包括等价类划分法、边界值分析法、因果图与决策表、正交实验设计法等，后面会陆续讲解。

1.6.4 执行测试

测试执行就是按照测试用例执行测试的过程，这是测试人员最主要的活动阶段。在执行测试时要根据测试用例的优先级进行。测试执行过程看似简单，测试人员只要按照测试用例完成测试工作即可，但实际并非如此。测试用例的数目非常多，测试人员需要完成所有测试用例的执行，每一个测试用例都可能会发现很多缺陷，测试人员要做好测试记录与跟

踪，衡量缺陷的质量并编写缺陷报告。

当提交后的缺陷被开发人员修改之后，测试人员需要对其进行回归测试。如果系统对测试用例产生了缺陷免疫，测试人员则需要编写新的测试用例。在单元测试、集成测试、系统测试、验收测试各个阶段都要进行功能测试、性能测试等，这个工作量无疑是巨大的。除此之外，测试人员还需要对文档资料，如用户手册、安装手册、使用说明等进行测试。因此不要简单地认为测试执行就是按部就班地完成任务，可以说这个阶段是测试人员最重要的工作阶段。

1.6.5 编写测试报告

测试报告是对一个测试活动的总结，对项目测试过程进行归纳，对测试数据进行统计，对项目的测试质量进行客观评价。不同公司的测试报告模板不同，但测试报告的编写要点都是一样的，一般都是先对软件进行简单介绍，然后说明这份报告是对该产品的测试过程进行总结，对测试质量进行评价。

一份完整的测试报告必须要包含以下几个要点。

（1）引言：描述测试报告编写的目的、报告中出现的专业术语解释及参考资料等。

（2）测试概要：介绍项目背景、测试时间、测试地点及测试人员等信息。

（3）测试内容及执行情况：描述本次测试模块的版本、测试类型、使用的测试用例计方法及测试通过覆盖率，依据测试的通过情况提供对测试执行过程的评估结论，并给出测试执行活动的改进建议，以供后续测试执行活动借鉴参考。

（4）缺陷统计与分析：统计本次测试所发现的缺陷数目、类型等，分析缺陷产生的原因，给出规避措施等建议，同时还要记录残留缺陷与未解决问题。

（5）测试结论与建议：从需求符合度、功能正确性、性能指标等多个维度对版本质量进行总体评价，给出具体、明确的结论。

小 结

本模块主要介绍了软件测试的基本知识，首先介绍了软件测试的发展过程；其次介绍了软件测试的目的和必要性；接着介绍了软件测试的分类、常见的软件测试模型及软件测试原则；并比较详细地讲解了软件测试的一般流程。读者在学习完本模块的知识之后，能对软件测试有比较直观的认识，为其后续学习打下基础。

思考与练习

一、单选题

1. 按照测试阶段分类，哪一项不属于该分类？（　　）

A. 单元测试　　B. 冒烟测试

C. 集成测试　　D. 验收测试

2. 下面哪一种说法是错误的？（　　）

A. 在软件测试中，冒烟测试是指软件构建版本建立后，对系统的基本功能进行简单的测试，这种测试重点验证的是程序的主要功能。

B. 系统测试：将经过测试的软件在实际环境中运行，并与其他系统的成分组合在一起进行测试。

C. 回归测试是指把已经测试过的软件单元进行组合，重新进行测试。

D. 验收测试主要是对软件产品说明进行验证，确保其符合用户的各项要求。

3. 下面哪一种说法是错误的？（　　）

A. 黑盒测试要了解程序内部是如何实现的。

B. 白盒测试在测试时，按照程序的执行路径得出结果。

C. 相较于黑盒测试，白盒测试对测试人员的要求会更高一点。

D. 回归测试策略按覆盖程度包括全部重复和选择性重复。

4. 下列哪一种是常用的测试模型？（　　）

A. 瀑布模型　　B. V 模型

C. W 模型　　D. H 模型

5. 软件测试原则包括哪些？（　　）

①测试应基于用户需求；②测试要尽早进行；③穷举测试是不可能的；④遵循 GoodEnough 原则；⑤缺陷要符合“二八”定理；⑥避免缺陷免疫

A. ①②③④⑤⑥　　B. ⑤⑥

C. ①②④⑤⑥　　D. ①②③④

二、填空题

1. 在执行测试需求评测时，测试人员一般会根据软件开发需求文档制作_____。

2. 一份完整的测试报告必须要包含以下几个要点：_____、_____、_____、_____、_____。

3. Bill Hetzel 在《软件测试完全指南》(Complete Guide of Software Testing)一书中指

出测试的定义：_____。这个定义至今仍被引用。

4. 1983 年，IEEE 指出软件测试再也不是一个一次性的、开发后期的活动，而是_____。

5. 软件测试的必要性包括：_____、_____、_____、_____、_____。

三、简答题

1. 简述软件测试的一般流程。

2. 如何制订测试计划？测试计划的主要作用是什么？

2 模块 2 黑盒测试

黑盒测试常用于系统集成测试、系统测试阶段。测试人员将系统的内部看作一个黑盒子，不用查看内部构造，不用关心功能如何实现及代码如何编写，只看由输入的数据能否产生正确的输出数据即可，黑盒测试也因此而得名。黑盒测试主要针对软件界面和功能进行测试。

2.1 等价类划分法

等价类划分法是一种常用的黑盒测试方法，它的主要思路是从大量甚至是无穷多的数据中选取一部分作为测试输入，希望测试的数据没有冗余，即尽可能地使用最少的测试覆盖最多的数据，以发现更多隐藏的软件缺陷。本节将详细介绍等价类划分法的概念及使用方法。

2.1.1 等价类划分法概述

一个程序可以有多个输入，等价类划分法就是将这些输入数据按照输入需求进行分类，将它们划分为若干个子集，这些子集即为等价类，在每个等价类中选择有代表性的数据设计测试用例。例如，将实数的平方根运算函数设计为 y＝sqrt(x)，假设 x 的约束为(0≤x≤10)，这样 x 就可以分为 3 个等价类，包括 x＜0、0≤x≤10、x＞10。

使用等价类划分法测试程序需要经过划分等价类和设计测试用例两个步骤，具体介绍如下。

1. 划分等价类

软件不可能只接收有效、合理的数据，还可能面临意外情况，如用户可能输入无效和不

合理的数据。软件只有经受了这样的考验，才能说明它的可靠。因此，等价类要区分有效等价类与无效等价类两种情况。有效等价类是针对软件规格说明而言的，它们是符合程序要求、合理且有意义的输入数据。无效等价类是针对软件规格说明而言的，它们是不符合程序要求、不合理或无意义的输入数据。

如何确定等价类是等价类划分方法的重要问题，一般在划分等价类时需要遵守以下原则。

(1)按区间划分。如果规格说明要求输入值是一个输入取值范围或者有限数量的值，则可以将输入数据划分为一个有效等价类和两个无效等价类，有效等价类为指定的取值区间，两个无效等价类分别为有限区间两边的值。例如，某程序要求输入值 x 的范围为[1，100]，则有效等价类为 1≤x≤100，无效等价类为 x<1 和 x>100。

(2)按限制条件或规则划分。如果程序要求输入值是一个“必须成立”的情况，则可以将输入数据划分为一个有效等价类和一个无效等价类。例如，某程序要求密码包含字母、数字，长度大于 8 位，则正确的密码为有效等价类，错误的密码为无效等价类。

(3)按数值划分。如果程序要求输入数据是一组可能的值，或者要求输入值必须符合某个条件，则可以将输入数据划分为一个有效等价类和一个无效等价类。例如，某程序要求输入数据必须是以数字开头的字符串，则以数字开头的字符串是有效等价类，不是以数字开头的字符串是无效等价类。

(4)细分等价类。如果等价类中每个输入数据在程序中的处理方式各不相同，则可将该等价类划分成更小的等价类。

2. 设计测试用例

根据测试用例的完整性分为四种：弱一般等价类测试、强一般等价类测试、弱健壮等价类测试和强健壮等价类测试。健壮是指要考虑无效值。强是指要考虑组合情况，使用笛卡尔积算出测试用例个数。下面以某城市电话号码为例来说明：某城市电话号码由三部分组成。地区码：空白或 4 位数字；前缀：以 2 开头的四位数字；后缀：4 位数字。根据上述信息划分等价类，见表 2-1。

表 2-1　某城市电话号码划分等价类

输入条件	有效等价类	无效等价类
地区码	空白(a)	有非数字字符
	4 位数字(b)	少于 4 位数字
		多于 4 位数字

续表

输入条件	有效等价类	无效等价类
前缀	2000～2999 之间的数(c)	有非数字字符
		起始位为 0
		起始位为 1
		少于 4 位数字
		多于 4 位数字
后缀	4 位数字(d)	有非数字字符
		少于 4 位数字
		多于 4 位数字

1)弱一般等价类

“弱”表示测试用例只需覆盖 3 个输入条件的所有有效等价类即可,不用考虑它们之间的组合,“一般”表示只考虑有效等价类,因此,最少需要 2 个测试用例即可满足弱一般等价类测试的要求。上面例子弱一般等价类的用例为 acd、bcd,具体测试用例,见表 2-2。

表 2-2　弱一般等价类测试用例

方案	输入			预期输出	说明
	地区码	前缀	后缀		
1	空白	2500	1234	有效	
2	1234	7890	3377	有效	

2)强一般等价类

“强”表示测试用例须覆盖 3 个输入条件的所有有效等价类的可能组合,“一般”表示只考虑有效等价类。地区码有 2 个有效等价类,前缀有 1 个有效等价类,后缀有 1 个有效等价类,因此共需要 2×1×1=2 个测试用例才能满足强一般等价类的测试要求。因为只有地区码有两个维度,其他都是一个维度,所以测试用例数量恰好与弱一般等价类相同。上面例子弱一般等价类的用例为 acd、bcd,具体测试用例,见表 2-3。

表 2-3　强一般等价类测试用例

方案	输入			预期输出	说明
	地区码	前缀	后缀		
1	空白	2670	1357	有效	
2	2345	7896	9177	有效	

3)弱健壮等价类

“弱”表示测试用例只需覆盖 3 个输入条件的所有有效等价类,不用考虑它们之间的组合,“健壮”表示不仅要考虑有效类,还须考虑无效类。在弱一般等价类的基础上,增加取值为无效值的 11 种情况。注意在编写测试用例时,每个用例只覆盖一个无效值,其他值是有效的,具体测试用例如表 2-4 所示。

表 2-4 弱健壮等价类测试用例

方案	输入			预期输出	说明
	地区码	前缀	后缀		
1	空白	2670	1357	有效	
2	2345	7896	9177	有效	
3	12a	7896	9177	无效	地区码有非数字字符
4	234	7896	9177	无效	地区码少于 4 位数字
5	23456	7896	9177	无效	地区码多于 4 位数字
6	2345	7896	9177	无效	前缀有非数字字符
7	0123	7896	9177	无效	前缀起始位为 0
8	1234	7896	9177	无效	前缀起始位为 1
9	2345	789	9177	无效	前缀少于 4 位数字
10	2345	56789	9177	无效	前缀多于 4 位数字
11	2345	7896	A123	无效	后缀有非数字字符
12	2345	7896	135	无效	后缀少于 4 位数字
13	2345	7896	13579	无效	后缀多于 4 位数字

4)强健壮等价类

“强”表示测试用例须覆盖三个输入条件的所有有效等价类的可能组合,“健壮”表示不仅考虑要有效类,还须考虑无效类。地区码有 2 个有效等价类,前缀有 1 个有效等价类,后缀有 1 个有效等价类,因此,共需要(2+3)×(1+5)×(1+3)=5×6×4=120 个测试用例才能满足强一般等价类的测试要求。因为用例数量较多,所以在此不再对测试数据进行列表,读者可以参照前述表格自己进行练习。

上面从 4 个方面思考了如何设计测试用例,那么要如何选择测试策略才能使得测试用例集尽可能最小且能发现更多的问题呢? 参考上述举例,弱健壮等价类是符合用例数较少且覆盖面较大要求的。使用等价类划分法设计测试用例的重点在于划分有效等价类和无效等价类粗细的粒度。粒度越粗,设计测试用例越少,粒度越细,设计测试用例越多。相对来说,粒度越细越能发现更多的问题。

2.1.2 实例:三角形问题的等价类划分

三角形问题是测试中广泛使用的一个经典案例,下面按照以下几个步骤使用弱健壮等价类方法来设计测试用例。

(1)按照输入条件建立有效等价类和无效等价类,列出所有划分出的等价类。

(2)为每一个等价类设置一个唯一的编号。

(3)设计一个新的测试用例,使其尽可能多地覆盖尚未被覆盖地有效等价类,重复这一步骤,直到所有的有效等价类都被覆盖为止。

(4)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,重复这一步骤,直到所有的无效等价类都被覆盖为止。

1. 划分等价类

它要求输入 3 个正数 a、b、c 作为三角形的三条边,要判断三角形的性质。三条边长对应的 3 个数决定了三角形是一般三角形、等边三角形、等腰三角形,还是无法构成三角形。如果使用等价类划分法设计三角形程序的测试用例,首先需要将所有输入数据划分为不同的等价类。

对该案例进行分析,程序要求输入 3 个数且是正数,在输入 3 个正数的基础上判断这 3 个数能否构成三角形,如果能构成三角形,那么再判断它构成的是一般三角形、等腰三角形还是等边三角形,如此可以按照下列步骤将输入情况划分为不同的等价类。

(1)判断三个数是否是正数,可以将输入情况划分为 1 个有效等价类和 3 个无效等价类,下列表述中表达式中的||表示用例是并列关系,&& 表示是同一用例关系。

①有效等价类:3 个数都是正数,a>0 && b>0 &&c>0。

②无效等价类:任意一个数小于等于 0,a≤0 || b≤0 || c≤0。

(2)在输入 3 个正数的基础上,判断 3 个数能否构成三角形,可以将输入情况划分为 3 个有效等价类和 3 个无效等价类,具体如下。

①有效等价类:任意两个数之和大于第三个数,a+b>c || a+c>b ||b+c>a。

②无效等价类:任意两个数之和小于等于第三个数,a+b≤c || b+c≤a || c+a≤b。

(3)在 3 个数构成三角形的基础上,判断三个数能否构成等腰三角形,可以将输入情况划分成 3 个有效等价类和 1 个无效等价类。

①有效等价类:其中有两个数相等,a=b||a=c||b=c。

②无效等价类:3 个数均不相等,a! =b && b! =c && c! =a。

(4)判断这 3 个数能否构成等边三角形,有 1 个有效等价类和 3 个无效等价类,具体如下。

①有效等价类:3 个数相等,a=b && b=c && a=c。

②无效等价类:3 个数不相等,a! =b || b! =c || c! =a。

2. 设置等价类编号

上述分析一共将三角形输入划分成了 15 个等价类,给这些等价类确定编号并建立等价类表,见表 2-5。

表 2-5 三角形输入等价类表

要求	有效等价类	编号	无效等价类	编号
3 个数是否都是正数	a>0 && b>0 && c>0	1	a≤0	2
			b≤0	3
			c≤0	4
3 个数能否构成三角形	a+b>c	5	a+b≤c	8
	b+c>a	6	b+c≤a	9
	c+a>b	7	c+a≤b	10
3 个数能否构成等腰三角形	a=b	11	a! =b && b! =c && c! =a	14
	a=c	12		
	b=c	13		
3 个数能否构成等边三角形	a=b && b=c && a=c	15	a! =b	16
			b! =c	17
			c! =a	18

3. 设计新用例

不断设计新的测试用例,使其尽可能多地覆盖尚未被覆盖的有效等价类,直到所有的有效等价类都被测试用例覆盖为止

接下来设计测试用例,设计测试用例的原则是:用尽可能少的测试用例覆盖尽可能多的等价类。既要考虑输入情况的全面性,又要考虑对有效等价类的覆盖情况。在设计新的有效等价类的测试数据时,增加等价类的覆盖范围,包括更多的有效等价类的编号,在这个例子中因为有效等价类存在递进关系,所以使得测试数据变得更加特殊化。根据表 2-1 中的有效等价类设计测试用例,见表 2-6。

表 2-6 覆盖有效等价类的测试用例

测试用例 ID	输入 3 个数(a,b,c)	覆盖有效等价类的编号
1	1,2,3	1
2	3,4,5	1,5,6,7
3	6,6,8	1,5,6,7,11
4	6,8,6	1,5,6,7,12

续表

测试用例 ID	输入 3 个数(a,b,c)	覆盖有效等价类的编号
5	8,6,6	1,5,6,7,13
6	6,6,6	1,5,6,7,11,12,13,15

表 2-6 设计了 6 组测试用例覆盖了全部有效等价类。

4. 覆盖所有无效等价类

不断设计新的测试用例,使其仅覆盖一个尚未被覆盖的无效等价类,直到所有无效等价类都被覆盖为止。

无效等价类测试用例的设计原则与有效等价类测试用例的设计原则相同,无效等价类的测试用例,见表 2-7。

表 2-7　无效等价类的测试用例

测试用例 ID	输入 3 个数(a,b,c)	覆盖无效等价类的编号
7	0,1,2	2
8	1,0,2	3
9	1,2,0	4
10	1,2,3	8
11	3,2,1	9
12	1,3,2	10
13	3,4,5	14
14	3,4,4	16
15	3,4,3	17
16	3,3,4	18

由表 2-7 可知,设计了 10 个测试用例覆盖了全部无效等价类。用户在测试三角形程序时,使用上述测试用例可以最大限度地检测出程序中的缺陷与不足。

2.1.3　实例:银行转账的等价类划分

银行转账是人们经常进行的操作,现在可以通过 PC 或手机 APP 进行操作。但是每天转账金额有上限,即转账额度。假设某银行已经开发了一款软件实现用户自助转账功能,要求转账笔数不限,但是每天转账上限为 5 万元整。

如果是当天的第一笔转账,则只需考虑转账上限 5 万元,如果是第 n 笔转账,最多只能提取账号内的余额数。因此,需要分别设计两个情况的等价类。

(1)第一笔转账,可将转账功能划分为 1 个有效等价类和 2 个无效等价类,具体如下。

①有效等价类：0＜提现金额≤Min(50 000，余额)。

②无效等价类：提现金额≤0。

③无效等价类：提现金额＞Min((50 000－已转账金额)，余额)。

(2)第 n 笔转账，可以将提现功能划分为 1 个有效等价类和 2 个无效等价类，具体如下。

①有效等价类：0＜提现金额≤Min((50 000－已转账金额)，余额)。

②无效等价类：提现金额≤0。

③无效等价类：提现金额＞Min((50 000－已转账金额)，余额)。

根据上述分析，银行转账功能一共可以划分为 6 个等价类，见表 2-8。

表 2-8　银行转账的等价类表

要求	有效等价类	编号	无效等价类	编号
	0＜转账金额＜＝Min(50 000，余额)	1	转账金额≤0	2
			转账金额＞Min(50 000，余额)	3
转账（第 n 笔）	0＜转账金额＜＝Min((50 000－已转账金额)，余额)	4	转账金额≤0	5
			转账金额＞Min((50 000－已转账金额)，余额)	6

建立了等价类表，接下来设计测试用例进行测试，假如现在银行账户中有 60 000 余额，则覆盖有效等价类的测试用例与覆盖无效等价类的测试用例如表 2-9 和表 2-10 所示。

表 2-9　覆盖有效等价类的测试用例

测试用例 ID	功　能	金　额	覆盖有效等价类的编号
1	转账(第 1 次)	1 000	1
2	转账(第 n 次，当天已转账 3 000)	47 000	4

表 2-10　覆盖无效等价类的测试用例

测试用例 ID	功　能	金　额	覆盖有效等价类的编号
转账（第 1 笔）	快速到账(第 1 次)	－10 000	2
		55 000	3
转账（第 n 笔）	快速到账(第 n 次，已转账 10 000)	－2 000	5
		45 000	6

表 2-9 和 2-10 共设计了 6 个测试用例，这些测试用例覆盖了全部等价类，基本可以检测出转账功能所存在的缺陷。

2.2 边界值分析法

边界是指对于输入等价类和输出等价类而言，稍高于其边界值及稍低于其边界值的一些特定情况。边界值分析法也是一种常用的黑盒测试方法。分析边界的原因是大量的错误经常发生在输入或输出范围的边界上，而不是在输入范围的内部。边界值分析法就是对边界值进行测试的一种方法，本节将针对边界值分析法进行详细的讲解。

2.2.1 边界值分析法概述

边界值分析法是对软件的输入或输出边界进行测试的一种方法，它通常作为等价类划分法的一种补充，其理论基础是假定大多数的错误是发生在各种输入条件的边界上的，如果边界附近的取值不会导致程序出错，那么其他取值导致程序出错的可能性很小。边界值分析法是在等价类的边界上执行软件测试工作的，测试用例都是设计在等价类的边界上。

在等价类中选择边界值时，如果输入条件规定了取值范围或值的个数，则要对边界值及边界值两边的数分别进行测试。例如，输入姓名(1～20 个字符)，要求测试 0、1、2 个字符和 19、20、21 个字符；某商品信息查询系统，每页最多显示 10 条商品信息，就应该准备足够的商品信息，使能够查询出 9、10、11、1、0 条商品记录。

边界值和等价类的区别在于，边界值分析不是从某等价类中随便找一个作为代表，而是这个等价类的每个边界都要作为测试条件。

如果软件要求输入或输出是一组有序集合，如数组、链表等，则可选取第一个和最后一个元素作为测试数据。如果被测试程序中有循环，则可选取第 0 次、第 1 次与最后两次循环作为测试数据。

常见的边界值有：

(1)文本框接收字符个数，如用户名长度、密码长度等。

(2)报表的第 1 行和最后 1 行。

(3)数值元素的第 1 个和最后 1 个。

(4)循环的第 1 次、第 2 次和倒数第 1 次、第 2 次

除了上述举例之外，测试时须仔细分析软件规格的需求，找出其他可能的边界条件。

边界值分析法只在边界取值上考虑测试的有效性，相对于等价类划分法来说，它执行起来更加简单易行，但缺乏充分性，不能整体全面地测试软件，因此它只能作为等价类划分法的补充测试。

2.2.2 实例:三角形问题的边界值分析

在 2.1.2 节,我们学习了三角形问题的等价类划分,在等价类划分中,除了要求输入数据为 3 个正数之外,没有给出其他限制条件,如果要求三角形边长取值范围为 1～10,则可以使用边界值分析法对三角形边界边长进行测试,在设计测试用例时,分别选取 1、2、5、9、10 五个值作为测试数据,其中,选择 5 作为取值范围内的任意一点,三角形问题边界值分析测试用例,见表 2-11。

表 2-11 三角形问题边界值分析测试用例

测试用例 ID	输入 3 个数(a,b,c)	被测边界	预期输出
1	5,5,1	1	等腰三角形
2	5,5,2		等腰三角形
3	5,5,5	无	等边三角形
4	5,5,9	10	等腰三角形
5	5,5,10		不构成三角形

在表 2-11 中,测试用例 1、2 测试的是等于和略大于最小值 1 的数据,测试用例 3 的值 5 是 1～10 范围内的任意值,测试用例 4、5 测试的是等于和略小于最大值 10 的数据,应用这几组测试用例基本可以检测出三角形边界存在的缺陷。

2.2.3 实例:银行转账的边界值分析

在 2.1.3 节中,我们学习了银行转账案例的等价类划分,每天银行转账笔数不限,转账上限为 5 万元。分别设计两种情况的等价类:如果是当天的第一笔转账,金额数介于 0 和 Min(50 000,余额);如果是第 n 笔转账,金额数介于 0 和 Min((50 000－已转账金额),余额)。

假设银行账号中的余额为 4 万元,在进行边界值分析时,如果是第一次转账,边界值为 0 和 Min(50 000,4 000),即 0 和 40 000,则分别对 0 和 40 000 两个边界值进行测试,分别取－1、0、1、20 000、39 999、40 000、40 001 七个值作为测试数据;如果是第 n 次转账,账号余额为 30 000,边界值为 0 和 Min(50 000,30 000),即 0 和 30 000,则分别对 0 和 30 000 两个边界值进行测试,分别取－1、0、1、20 000、29 999、30 000、30 001 七个值作为测试数据;如果银行账号的余额大于 5 万,按上述方法设计测试用例。

根据上述分析,设计银行转账边界值分析测试用例,见表 2-12。

表 2-12　银行转账边界值分析测试用例

测试用例 ID	功　　能	金　　额	被测边界	预期输出
1	转账，卡上余额 4 万（第 1 笔）	−1	0	无法转账
2		0		无法转账
3		1		1
4		20 000	无	20 000
5		39 999	40 000	39 999
6		40 000		40 000
7		40 001		无法转账
8	转账，卡上余额大于 5 万（第 1 笔）	−1	0	无法转账
9		0		无法转账
10		1		1
11		25 000	无	25 000
12		49 999	50 000	49 999
13		50 000		50 000
14		50 001		无法转账
15	转账（第 n 笔，Min((50000−已转账金额)，余额)=30000）	−1	0	无法转账
16		0		无法转账
17		1		1
18		15 000	无	5 000
19		29 999	30000	29 999
20		30 000		30 000
21		30 001		无法转账

由表 2-12 可知，一共设计了 21 个测试用例来测试银行转账金额的边界值，这些测试用例基本覆盖了全部边界值，对边界可能存在的缺陷具有检测能力。

2.3 因果图与决策表

等价类划分法与边界值分析法主要侧重于输入条件，却没有考虑这些输入之间的关系，如组合、约束等。如果在测试时必须考虑输入条件的各种组合，可能的组合数将是天文数字。因此，使用一种适合于描述多种条件的组合，相应产生多个动作的形式来设计测试用例，这就需要利用因果图。因果图法最终生成的是决策表。它适合于检查程序输入条件的各种组合情况。

2.3.1 因果图法

因果图法是一种利用图解法分析输入的各种组合情况的测试方法，它考虑了输入条件的各种组合及输入条件之间的相互制约关系，并考虑到输出情况。例如，某一自动售货机软件要求先投币，后选择商品，如果投币金额不足，则无法选择价格较高的商品，只能选择价格较低的商品。可选择的商品受到了投币金额的约束，像这样多个输入之间有相互制约关系的情况，就无法使用等价类划分法和边界值分析法设计测试用例。因果图法就是为了解决多个输入之间的作用关系而设计的测试用例设计方法。

下面介绍如何使用因果图展示多个输入/输出之间的关系，并学习如何通过因果图法设计测试用例。

1. 因果图

因为输入、输出之间逻辑关系有时比较复杂，因此习惯用图示的方式及因果图来表达。通常，因果图中使用了一些简单的逻辑符号和直线将因（输入）和果（输出）连接起来，一般用 c_i 表示因，用 e_i 表示果。各节点可以取“0”或“1”，其中“0”表示状态不出现，“1”表示状态出现。

c_i 与 e_i 之间有恒等、非（～）、或（∨）、与（∧）4 种关系，如图 2-1 所示。

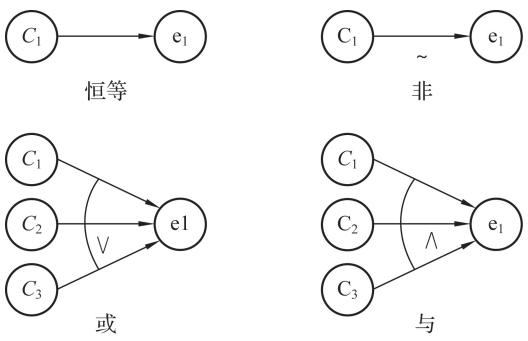

图 2-1 因果图

图 2-1 中展示了因果图的 4 种关系，每种关系的具体含义如下。

（1）恒等：若原因出现，则结果出现。若原因不出现，则结果也不出现。

（2）非：若原因出现，则结果不出现。若原因不出现，则结果出现。

（3）或（∨）：若几个原因中有一个出现，则结果出现，若几个原因都不出现，则结果不出现。

（4）与（∧）：只有几个原因都出现，结果才出现。若其中有一个原因不出现，则结果不出现。

为了表示原因与原因之间，结果与结果之间可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号。某一软件用于统计体检信息，在输入个人信息时，性别只能输入男或女，这两种输入不能同时存在，而且如果输入性别为女，那么体检项就会受到限制。这些依赖关系在软件测试中被称为“约束”，从输入（原因）考虑，有 4 种约束，包括 E（exclusive or）、I（in）、O（only）、R（request），从输出（结果）考虑，还有一种约束 M（mask）。约束的类别可分为 4 种，在因果图中，用特定的符号表明这些约束关系，如图 2-2 所示。

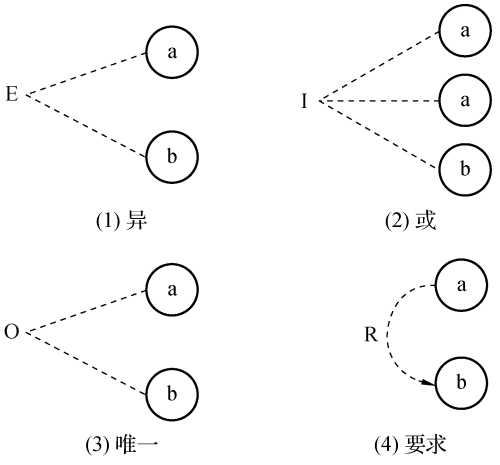

图 2-2　多个输入之间的约束符号

图 2-2 展示了多个输入之间的约束符号，这些约束关系的含义如下。

- E（互斥，异）：a 和 b 中最多只能有一个为 1，即 a 和 b 不能同时为 1。
- I（包含，或）：a、b 和 c 中至少有一个必须是 1，即 a、b、c 不能同时为 0。
- O（唯一）：a 和 b 中有且仅有一人为 1。
- R（要求）：a 和 b 必须保持一致，即 a 为 1 时，b 也必须为 1，a 为 0 时，b 也必须为 0。

上面这 4 种都是关于输入条件的约束，除了输入条件，输出条件也会相互约束，输出条件的约束只有一种 M（Mask，强制），在因果图中，使用特定的符号表示输出条件之间的强制约束关系，如图 2-3 所示。

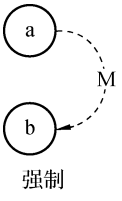

图 2-3　输出条件之间的强制约束关系

在输出条件之间的强制约束关系中，若 a 为 1，则 b 强制为 0，若 a 为 0，则 b 强制为 1。

2. 因果图法设计测试用例的步骤

使用因果图法设计测试用例需要经过以下几个步骤。

(1)分析程序规格说明书所描述的内容，哪些是原因(输入条件或输入条件的等价类)，哪些是结果(输出条件)，并给每个原因和结果赋予一个标识符。

(2)分析软件规格说明描述中的语义，找出原因与结果之间，原因与原因之间的对应关系，并根据这些关系，画出因果图。

(3)由于语法与环境的限制，有些输入与输入之间、输入与输出之间的组合情况是不可能出现的，为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。

(4)将因果图转换为决策表。决策表将在下一小节讲述。

(5)把决策表的每一列拿出来作为依据，根据决策表设计测试用例。

因果图法考虑了输入情况的各种组合及各种输入情况之间的相互制约关系，可以帮助测试人员按照一定的步骤高效率地开发测试用例，此外，因果图是由自然语言规格说明转化成形式语言规格说明的一种严格方法，它能够发现规格说明书中存在的不完整性和二义性，帮助开发人员完善产品的规格说明。

2.3.2 决策表

在实际测试中，如果输入条件较多，再加上各种输入与输出之间的相互作用关系，画出的因果图会比较复杂，容易导致混乱，为了避免这种情况的发生，往往使用决策表法代替因果图法。

决策表也称为判定表，其实质就是一种逻辑表。在程序设计发展初期，决策表就已经被当作程序开发的辅助工具来帮助开发人员整理开发模式和流程，因为它可以把复杂的逻辑关系和多种条件组合的情况表达得既具体又明确，利用决策表可以设计出用例集合。

为了让读者理解前面所述的因果图和决策表的联系，下面通过一个常见的自动售货机的例子来说明。其规格说明如下：自动售货机可以接收 1 元、5 元、10 元等类型的纸币，投入金额不限。售货机内现有可乐和橙汁两种商品，价格均为 3 元，投入钱币之后按下“可乐”或“橙汁”按钮，若顾客投入的金额不足，则一个显示“金额不足”的红灯亮；若金额足够，则相应的饮料就送出来。同时更新投入售货机内金额的余额，顾客可继续选购，直到按下购买结束按钮。

根据上述规格说明，分析出原因和结果及中间过程(节点)。

原因如下：

(1)未投币。

(2)余额大于等于 3 元。

(3)余额小于 3 元。

(4)按下“可乐”按钮。

(5)按下“橙汁”按钮。

(6)结束购买。

结果如下：

(1)“余额不足”灯亮。

(2)送橙汁。

(3)送可乐。

(4)更新余额。

(5)退钱。

分析出原因和结果后，接下来画出因果图，如图 2-4 所示，其中设计 2 个中间状态节点：

(1)按下“橙汁”或“可乐”按钮。

(2)余额大于等于 0。

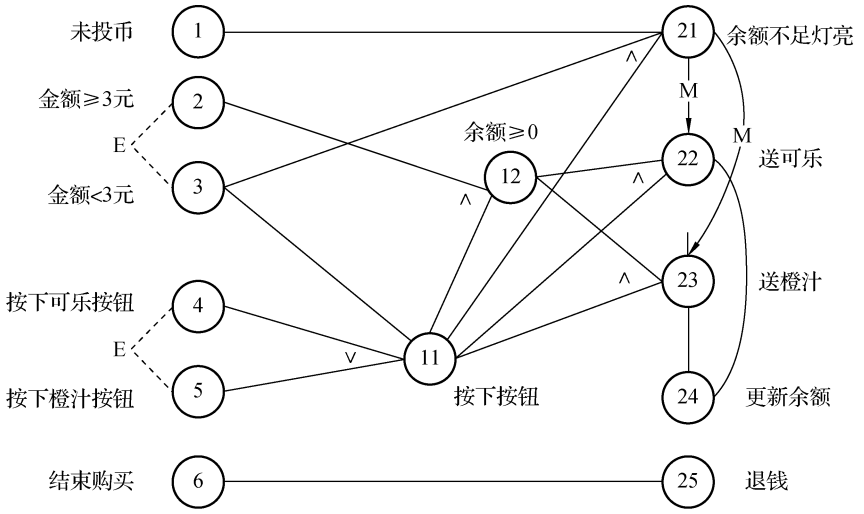

图 2-4　自动售货机因果图

由于 2 与 3 ，4 与 5 不能同时发生，分别加上约束条件 E。22 与 24、23 与 24 为恒等关系，21 与 22，21 与 23 为 M 约束关系，即若 21 为 1，则 22 和 23 必为 0，详细情况就不一一列举了，请读者查看图示。因果图法是一种非常有效的黑盒测试方法，它能够生成没有重复性的且发现错误能力强的测试用例，而且对输入、输出同时进行了分析。

接下来将因果图转换为决策表，见表 2-13。