CHAPTER

第5章 常用黑盒测试方法

本章学习目标

- 理解黑盒测试的常用方法
- 掌握黑盒测试方法的原理
- 理解各种黑盒测试方法的优缺点
- 熟练运用黑盒测试法开展软件测试

本章重点介绍黑盒测试常用方法的基本概念、基本原理，每种测试方法均结合相关案例展开，以便读者更好地将测试方法与测试实践应用相结合，深刻理解相关概念，掌握应用方法。

5.1 Ad-hoc 测试和 ALAC 测试

5.1.1 Ad-hoc 测试

Ad-hoc 测试，又叫随机测试，“随机”意味着没有计划、没有目的地随意测试。大多数软件从业人员认为 Ad-hoc 测试是在浪费时间和资源，实际上，Ad-hoc 测试在软件测试的生命周期里扮演着非常重要的角色，利用它可以快速地发现一些严重缺陷，审查测试用例是否完备，所以，Ad-hoc 测试是其他有计划、有目的测试的一种必要补充，是保证测试覆盖完整性的有效方式和过程。

1. 什么是 Ad-hoc 测试

Ad-hoc 测试是一种在没有测试计划和文档的情况下，由测试人员根据经验采用尽可能合适的方法随机地对软件进行测试的一种方法。所以，Ad-hoc 测试是非结构化的探究性测试，是最小形式的测试方法。例如，使用 Ad-hoc 测试方法测试某一函数时，可以采用随机生成的函数参数作为输入数据；当使用 Ad-hoc 测试方法测试某个 API 接口时，可以采用随机生成的请求信息作为输入请求；当使用 Ad-hoc 测试方法测试用户界面时，可以采用随机生成的操作序列作为测试输入。

理论上，在整个测试过程中，每一个软件只需要进行一次随机测试，除非发现缺陷。

实际上，Ad-hoc 测试适用于许多情况。

1）项目早期需使用 Ad-hoc 测试

在项目早期，测试人员需要通过软件需求规格说明书等获取软件测试需求，但这不足以使其完全弄清楚一个软件系统的实际行为，还需要亲自感受、测试、使用软件。这时，Ad-hoc 测试就是一种最好的发现探索的方法，它可以帮助测试人员理解被测试软件，获取测试需求，编写更有效、质量更高的测试用例。

2）项目中期需使用 Ad-hoc 测试

在项目中期，Ad-hoc 测试所获得的数据可以帮助测试人员发现测试策略的漏洞，找到本来没有明显关系的子系统之间的关系，发现漏掉的测试用例，帮助设定测试用例的优先级以及应用软件的发布日期。这时，Ad-hoc 测试是一种检验测试完整性的工具。如果测试人员对软件进行 Ad-hoc 测试后，仍没有发现任何现有测试用例之外的问题，说明当前的测试用例比较完备。

3）项目晚期需使用 Ad-hoc 测试

在项目晚期，如果一个软件已经完成测试文档里规定的全部测试，进入由用户或产品管理团队验收测试阶段，Ad-hoc 测试可以帮助检查应用软件质量，此时 Ad-hoc 测试是能够执行的唯一一种测试方式。

4）Ad-hoc 测试是常规测试的必要补充

在常规测试中，当测试人员发现某一软件问题后，可以使用 Ad-hoc 测试方法探索是否存在相关问题，相关模块是否存在相似问题。

5）Ad-hoc 测试与回归测试互补使用

当测试人员在回归测试中发现问题时，需要通过 Ad-hoc 测试方法分析和定位缺陷，反复尝试，确定问题复现步骤；测试人员在 Ad-hoc 测试中发现问题时，需要在缺陷跟踪系统中记录问题，开发人员将问题修正后，测试人员需要通过回归测试验证该问题。

在 Ad-hoc 测试过程中，测试人员不需要执行任何测试用例，不与任何分配的测试任务绑定，所以，一般情况下，测试人员不会花费很多时间进行 Ad-hoc 测试，主要针对被测试软件的一些重要功能点、新增加的功能点、特殊情况点、特殊使用环境、并发性等情况，尤其是当前测试用例没有覆盖的部分，凭借他们的直觉和经验寻找 Bug。因此，Ad-hoc 测试最好由经验丰富并且熟悉被测试软件的测试人员开展，发现常规测试中没有发现的 Bug。

假设一个求和函数 int sum(int a, int b)，接收两个输入参数 a 和 b，将两个参数相加后，结果作为返回值返回。如果这一函数在 16 位系统中运行，那么 a、b 的取值范围是－32768～32767。两个输入参数的取值范围形成一个矩形，测试人员可以凭借经验在这个矩形中任意取一点作为 sum 函数的输入进行测试。又如，测试一个画图软件，打开画图软件后，测试人员使用鼠标拖曳的方式，在画图窗口中任意绘制不同的图形进行测试。

2. Ad-hoc 测试方法

Ad-hoc 测试的形式多种多样：可以由开发人员和测试人员共同在同一个模块中发

现 Bug，这有助于开发人员及早发现问题，更改设计，也有助于测试人员发现更好的测试用例；可以由两个测试人员分享测试想法共同测试同一模块，一人负责执行测试，一人负责记录上报 Bug；还可以像猴子、猩猩一样随机执行测试，没有模块分配，没有步骤，随意操作，以便破坏系统，寻找缺陷等。总之，能找到 Bug 的 Ad-hoc 测试就是好的 Ad-hoc 测试。Ad-hoc 测试不仅可以应用到手工测试过程中，也可以应用到自动化测试中，比较经典的有效的自动化 Ad-hoc 测试主要有 Fuzzing 测试、Monkey 测试、混沌测试和基于属性的测试。

1）Fuzzing 测试

Fuzzing 测试，即模糊测试，是针对一个被测试对象，不做任何假设，随机生成无数个合法和非法的输入数据，输入到程序中，然后观察被测试对象行为的测试，如崩溃、断言失败、内存泄漏等。通常，Fuzzing 测试发现的问题是严重的、容易被黑客攻击的错误，所以，它常被用来做可靠性和安全测试。但是 Fuzzing 测试无法全面了解整个安全威胁或 Bug，效率比较低，一个 Fuzzing 任务可能会执行几天几夜。为了提高 Fuzzing 测试的有效性和效率，AFL（american fuzzy lop）、libFuzzer、Honggfuzz 等操作简单友好的工具相继出现，极大地降低了门槛。

2）Monkey 测试

Monkey 测试是 Android 应用程序自动化测试的一种，它通过产生随机用户事件和系统事件，模拟用户按键输入、触摸屏输入、手势输入等，对设备上的程序进行测试，观察应用是否发生崩溃或抛出未处理的异常以及多长时间发生，从而检查程序长时间运行时的稳定性。从本质上说，Fuzzing 测试和 Monkey 测试是相同的，只是 Fuzzing 测试强调的是随机产生测试数据（data），而 Monkey 测试强调的是随机产生测试动作（action）。

3）混沌测试

混沌测试（chaos testing）是通过人为地故意向系统注入错误、故障，测试系统是否发生崩溃，从而检测系统对故障的处理能力和恢复能力。在混沌测试鼻祖 Netflix 的工具箱中，有许多基于随机算法生成各种异常和错误的 Monkey 工具。例如，Latency Monkey 可以通过随机化服务器端的响应时延来模拟服务器死机、服务降级等异常现象。

4）基于属性的测试

在功能测试中，被测试对象不随测试输入（外部条件）变化而发生改变的性质，称为对象的属性。如一个字符串拼接函数，对于任何输入字符串 a 和 b，拼接结果一定包含字符串 b，这一性质是字符串拼接函数的属性；在一个编解码模块中，给定任意输入 X，对其编码结果进行解码，结果一定是 X，这一性质是编解码模块的属性。基于属性的测试（property-based testing）就是使用自动化工具随机生成测试输入，观察程序接受输入后，对象属性是否保持不变。如果某个输入导致对象违反其中一条属性，则说明程序中存在错误。

3. 如何进行 Ad-hoc 测试

由于不考虑任何其他信息，所以 Ad-hoc 测试的错误检测效率较低，测试人员需要在进行测试之前做足准备，如熟悉软件产品的各项功能和正常输出，选择测试重点等，一般需要重点关注以下几个方面。

(1) 选择常规测试缺陷集中点开展测试。

在常规测试阶段,发现缺陷最多的功能模块被称为缺陷集中点。根据 80-20 原则,80%的缺陷集中在 20%的模块中,测试人员需要找到程序最复杂、最薄弱的功能模块,也就是缺陷集中点,构建粗略想法,开展 Ad-hoc 测试。

为了更快地找到缺陷集中点,可以采用自适应 Ad-hoc 测试方法。如前面所述的求和函数 sum,如果输入的两个数 a 与 b 的和超出系统可以表示的最大范围,系统会溢出,这样的数据全部集中在输入矩阵的左下角和右上角。科学家对这一现象进行了总结,认为能够导致程序出错的测试用例通常会呈现出矩形状分布、条带状分布以及散点状分布。也就是说,如果某一条测试用例运行通过,那么与它相关的其他测试用例运行通过的概率会比较大;如果某一条测试用例运行失败,那么与它相关的其他测试用例运行失败的概率也会比较大。所以,Ad-hoc 测试选择测试用例时,可以根据测试结果决定下一条测试用例的选择情况。比如在一个输入域中,选择一条测试用例,如果这条测试用例运行通过,那么就定义一个距离 D1 选择下一条测试用例;如果第二条测试用例运行仍然通过,那么就定义一个更大的距离 D2 选择第三条测试用例。以此类推,直到找到一条运行无法通过的测试用例。这种方法称为自适应 Ad-hoc 测试方法。

(2) 选择常规测试中缺陷概率低的功能点进行测试。

在测试过程中,可以选择在常规测试中发现的出现概率比较低的缺陷涉及的功能点进行测试,因为重现率比较低的缺陷是隐藏得比较深的缺陷,通常是 Ad-hoc 测试的重点。

(3) 针对测试用例未覆盖范围进行测试。

在测试过程中,测试人员可以针对设计测试用例未涵盖的范围进行重点测试,以求发现更多的问题。

(4) 在容易犯错的地方进行测试。

测试人员可以与开发人员进行沟通,从而了解开发人员容易犯错、容易忽略的地方,这也是 Ad-hoc 测试的重点。

(5) 根据经验选取重点测试。

测试人员可以根据自身测试经验选择重点测试的功能点进行测试。

Ad-hoc 测试就是结合这些重点测试功能,随机选取其他功能点,通过各种工具,逐步测试软件。在测试的过程中,尽量记录偏差,如果发现缺陷,则上报缺陷,创建相应的测试用例,协助测试人员重现测试场景。

4. Ad-hoc 测试特点

Ad-hoc 测试可以帮助测试人员检查现有测试用例是否完备,当发现新的测试用例时,测试人员需要考虑测试策略、测试用例设计是否存在漏洞,是否需要修改或重新制定;自动化 Ad-hoc 测试可以使用各种编程语言自带的伪随机数生成器生成各种符合规则的输入数据。理想情况下,随机产生的数据可以覆盖整个输入空间,但从实际情况看,由于时间和资源有限,在 Ad-hoc 测试中不会花费大量时间,所以 Ad-hoc 测试覆盖率并不高。总之,Ad-hoc 测试主要具有如下优点:

- 能够帮助测试人员更好地理解应用软件的行为或者应用软件的某一特性。

- 能够帮助测试人员设置测试的优先级。如果在某个功能点没有发现任何缺陷，那么可以将该功能点的优先级降低；如果在某个功能点发现了重大问题，或者发现了很多问题，则应该将该功能点的优先级提高。
- 有经验的测试人员可以发现重要的 Bug，通过这些 Bug 可以帮助测试团队更新测试用例，并将这些漏洞添加到初始的测试策略中。
- 不需要做详细的测试计划，不需要设计测试用例，随时可以开始执行测试，容易上手，节省大量时间。

因此，Ad-hoc 测试能否成功在很大程度上取决于测试人员的能力。测试人员必须依靠自己的直觉，在没有任何合适的计划和文档的情况下找到 Bug。如果测试人员非常有经验，可能测试几个小时就能找到很多严重的 Bug；如果测试人员是新手，可能测试一天都没有发现任何问题。

5.1.2 ALAC 测试

ALAC 是 act-like-a-customer 的简写，是一种基于客户使用产品的知识开发出来的测试方法。其测试原则如下。

- 一个软件产品或系统中全部功能的 20％是常用功能，用户 80％的时间都在使用这 20％的功能，而软件产品或系统中剩余 80％的功能不是经常使用的功能，用户只有 20％的时间在使用剩余的 80％的功能。
- 测试发现的所有错误的 80％很可能都集中在 20％的程序模块中，另外 20％的错误很可能集中在其他 80％的程序模块中。

ALAC 测试主要针对客户经常使用的模块进行测试，查找和修正客户最容易遇到的错误，所以它的成本低，测试时间短，最大的受益者是用户。ALAC 测试适合于演示版产品和开发预算很低、测试时间不充足、开发计划日程表很紧的项目。

5.2 等价类划分法

假设现有一个编辑框，只允许输入至多 6 个英文字母或数字，如果采用穷举法的思想设计测试用例，则英文字母共有 26 个小写字母和 26 个大小字母，数字有 10 个。所以，如果只输入 1 个英文字母或数字，共有 26×2＋10＝62 种取值；如果输入 2 个英文字母或数字，则有 62×62＝3844 种取值；如果输入 3 个，4 个呢？这里仅考虑了正常范围取值，如果考虑不合法取值呢？可以看出，穷举测试工作量很大，根本无法完成，穷举测试是不可行的。

5.2.1 等价类划分法概述

等价类划分法是一种常见的黑盒测试方法，它把所有可能的输入数据，即程序的输入域，划分成若干部分(子集)，然后从每一个子集中选取少数具有代表性的数据作为测试用例。在每一个子集中，各个输入数据对于发现程序中的缺陷是等价的，也就是测试同一个

子集中的代表数据相当于测试了这一子集的其他所有数据，如果这一子集中代表数据能发现软件中的缺陷，那么这一子集中其他数据也能发现同样的缺陷；反之，如果这一子集中的代表数据没有发现软件中的缺陷，那么这一子集中的其他数据也不会查出缺陷，这些输入域的子集合就被称为等价类。

等价类分为有效等价类和无效等价类。有效等价类是由对于程序的规格说明来说合理的、有效的输入数据构成的集合，用来验证软件实现的需求规格说明书中规定的功能和性能。无效等价类是由对于程序规格说明来说不合理的、无效的输入数据构成的集合，用来检查软件异常情况。在软件测试中，软件既要能接收有效数据的输入，也要能接收无效数据的输入考验，因此，使用等价类划分法设计测试用例时，既需要考虑有效等价类，又需要考虑无效等价类。

使用等价类划分法测试程序有两个主要的步骤：划分等价类和设计测试用例。

1. 划分等价类

在测试过程中，即使一个很小的程序，它的输入数据域也是非常大、非常广泛的，如输入参数、公共变量、用户级输入、读文件、读其他的 I/O 等，所以划分等价类是一个非常必要的步骤。划分等价类就是将输入域划分成若干个子集，每一个子集应该是互不相交的，这样可以保证通过等价类选取的测试用例能够使程序进行无冗余的测试；所有子集的并集应该是整个输入数据域，这样才可以保证通过等价类选取的测试用例能够对程序进行完备的测试。通常，划分等价类需要遵循以下原则。

(1) 如果输入条件规定了一个取值范围(例如，数量可以是 1～999)，那么可以确定出一个有效等价类(1≤数量≤999)，以及两个无效等价类(数量＜1，数量＞999)。例如，输入值是学生成绩，范围是 0～100，可以将整个输入域分成三部分，如图 5-1 所示，0～100 的成绩为有效等价类，小于 0 和大于 100 的成绩构成了两个无效等价类；

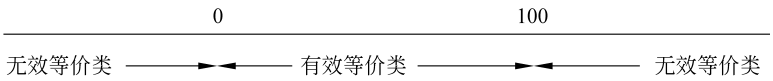

图 5-1　学生成绩等价类划分

(2) 如果输入条件是一个布尔量，则可确定一个有效等价类(满足条件)和一个无效等价类(不满足条件)；

(3) 如果输入条件规定了输入值的集合或规定了“必须如何”，例如，“标识符的第一个字符必须是字母”，那么就可以确定一个有效等价类(首字符是字母)和一个无效等价类(首字符不是字母)；

(4) 如果规定了输入数据的一组值(假定有 n 个值)，并且程序要对每个输入值分别处理(例如，交通工具的类型必须是公共汽车、卡车、出租车、火车或摩托车)，可以为每个输入值确定一个有效等价类(如公共汽车类、卡车类、出租车类、火车类或摩托车类)，不属于输入值的其他所有数据构成一个无效等价类(如拖车类)；

(5) 如果规定了输入数据必须遵守规则，则可以确定一个有效等价类(符合规则)和

若干个无效等价类(从不同角度违反规则);

(6) 如果在已划分的同一个等价类中存在两个以上的元素在程序处理中的方式不同,则应将该等价类进一步划分为更小的等价类。

常见的等价类划分法有两种:一种是基于接口的,另一种是基于功能的。基于接口的划分是对接口的每一个输入参数的输入特征进行划分,测试人员并不关心整个系统实际功能;基于功能的划分需要测试人员了解整个系统或者这一模块的具体功能,根据功能进行划分。

- 基于接口的划分法通常把所有的输入参数作为独立个体看待,不关心数据之间的相互关系,采用基于某种语法或者某种数学特征划分等价类。例如,某程序规定:“输入 3 个整数 a、b、c 作为边长构成三角形。通过程序判定所构成的三角形的类型,当此三角形为一般三角形、等腰三角形及等边三角形时,分别作计算……”。该程序有 3 个输入参数 a、b、c,要求都是大于零的整数,根据第 1 条和第 3 条等价类划分原则,可以将每个参数的输入域划分为 3 个等价类,分别为 b_1、b_2、b_3,即大于零、小于或等于零、非整型数。因此,存在 $3\times3\times3$ 个组合输入数据,即需要进行 27 次测试;
- 基于功能的划分法需要测试人员确定模块或者系统功能,理解程序的隐义或者业务逻辑,考虑不同输入参数之间的关系。以判断三角形类型程序为例,此时考虑的不是输入参数的数据特征,而是将 3 个参数关联起来,根据共同特征规则,判断形成三角形的类型。三角形的类型可以划分为 4 种,即一般三角形、等腰三角形、等边三角形和非三角形,也可以将第 2 种等腰三角形细分为等腰不等边三角形,保证三角形的第 2 种和第 3 种类型不重合,形成 4 种等价类划分,分别为 b_1、b_2、b_3、b_4,即一般三角形、等腰不等边三角形、等边三角形和非三角形。根据这 4 个划分,可以产生 4 个具有代表性的测试数据,如 $a=4$、$b=5$、$c=6$ 是一般三角形,$a=3$、$b=3$、$c=4$ 是等腰不等边三角形,$a=3$、$b=3$、$c=3$ 是等边三角形和 $a=3$、$b=4$、$c=8$ 是一个非三角形。

在实际应用中,可以将这两种方法结合起来,确定划分等价类。仍然以判断三角形类型程序为例,分析题目中所给出的条件以及隐含条件,它们分别如下。

(1) 整数　(2) 三个数　(3) 非零数　(4) 正数

(5) 两边之和大于第三边　(6) 等腰不等边　(7) 等边

如果输入参数 a、b、c 满足条件(1)~(4),则输出下列 4 种情况之一。

- 如果不满足条件(5),则程序输出“非三角形”;
- 如果三条边相等即满足条件(7),则程序输出“等边三角形”;
- 如果只有两条边相等即满足条件(6),则程序输出“等腰三角形”;
- 如果三条边都不相等,则程序输出“一般三角形”。

2. 设计测试用例

确立等价类后,可建立等价类表,列出每一个输入条件对应的有效等价类和无效等价

类，并为每个等价类设置唯一的编号，如表 5-1 所示。

表 5-1　判断三角形类型的等价类表

		有效等价类	编码	无效等价类	编码
输入条件	输入 3 个整数	整数	1	a 为非整数	12
				b 为非整数	13
				c 为非整数	14
				a、b 为非整数	15
				b、c 为非整数	16
				a、c 为非整数	17
				a、b、c 均为非整数	18
		3 个数	2	只有 a	19
				只有 b	20
				只有 c	21
				给出 a、b	22
				给出 b、c	23
				给出 a、c	24
				给出 3 个以上	25
		非零数	3	a 为 0	26
				b 为 0	27
				c 为 0	28
				a、b 为 0	29
				b、c 为 0	30
				a、c 为 0	31
				a、b、c 均为 0	32
		正数	4	$a<0$	33
				$b<0$	34
				$c<0$	35
				$a<0$ 且 $b<0$	36
				$a<0$ 且 $c<0$	37
				$b<0$ 且 $c<0$	38
				$a<0$、$b<0$、$c<0$	39

续表

输出条件	构成一般三角形	$a+b>c$	5	$a+b<c$	40
				$a+b=c$	41
		$b+c>a$	6	$b+c<a$	42
				$b+c=a$	43
		$a+c>b$	7	$a+c<b$	44
				$a+c=b$	45
	构成等腰三角形	$a=b$	8		
		$b=c$	9		
		$a=c$ 且两边之和大于第三边	10		
	构成等边三角形	$a=b$ 且 $b=c$ 且 $a=c$	11		

创建等价类表后，利用等价类表生成测试用例，其过程如下。

- 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。因为某些特定的输入错误检查可能会屏蔽或忽略掉其他输入的错误检查。

仍然以表 5-1 为例，从等价类表中生成覆盖有效等价类的测试用例如表 5-2 所示，覆盖无效等价类的测试用例如表 5-3 所示。

表 5-2　判断三角形类型的有效等价类测试用例

a	b	c	覆盖等价类编码
3	4	5	(1)～(7)
4	4	5	(1)～(7),(8)
4	5	5	(1)～(7),(9)
5	4	5	(1)～(7),(10)
4	4	4	(1)～(7),(11)

表 5-3　判断三角形类型的无效等价类测试用例

a	b	c	覆盖等价类编码	a	b	c	覆盖等价类编码
2.1	4	5	12	0	0	5	29
3	4.1	5	13	3	0	0	30
3	4	5.1	14	0	4	0	31

续表

a	b	c	覆盖等价类编码	a	b	c	覆盖等价类编码
3.1	4.1	5	15	0	0	0	32
3	4.1	5.1	16	−3	4	5	33
3.1	4	5.1	17	3	−4	5	34
4.1	4.1	5.1	18	3	4	−5	35
3			19	−3	−4	5	36
	4		20	−3	4	−5	37
		5	21	3	−3	−5	38
3	4		22	−3	−4	−5	39
	4	5	23	3	1	5	40
3		5	24	3	2	5	41
3	4	5	25	3	1	1	42
0	4	5	26	3	2	1	43
3	0	5	27	1	4	2	44
3	4	0	28	3	4	1	45

等价类划分的测试用例设计方法减少了穷举法带来的大量测试用例，保证了测试的效果和效率。

5.2.2 等价类划分法案例

1. 档案管理系统案例

现有一个档案管理系统，要求用户输入以年月表示的日期。假设日期限定在 1990 年 1 月～2049 年 12 月，并规定日期由 6 位数字字符组成，前 4 位表示年，后 2 位表示月。现用等价类划分法设计测试用例，测试程序的“日期检查功能”。

该程序要求输入的日期数据是 6 位数字字符，且限制在 1990 年 1 月～2049 年 12 月，如果条件满足，则认为日期输入有效，否则认为输入无效。按照下列步骤将输入情况划分为不同的等价类。

(1) 判断是否输入 6 位数字字符，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。

- 有效等价类：输入 6 位数字字符；
- 无效等价类：输入的数据中包含非数字字符；
- 无效等价类：输入少于 6 位数字字符；
- 无效等价类：输入多于 6 位数字字符。

(2) 在输入 6 位数字字符的基础上，判断年份是否在 1990～2049，可以将输入情况划