黑盒测试技术
black-box-testing

Dynamic black-box testing

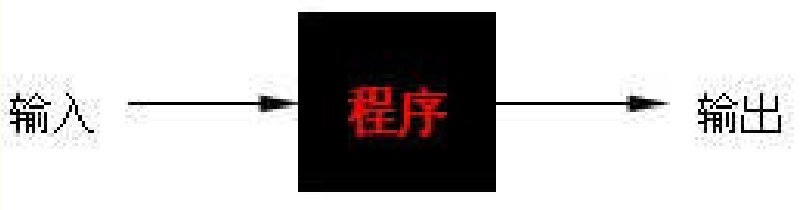
- Dynamic black-box testing is testing without having an insight into the details of the underlying code.
  - Dynamic, because the program is running
  - Black-box, because testing is done without knowledge of how the program is implemented.
- Sometimes referred to as *behavioral testing*.
- Requires an executable program and a specification (or at least a user manual).
- Test cases are formulated as a set of pairs
  - E.g., (input, expected output)

14.3黑盒测试用例的设计方法

- 黑盒测试的概念
  - 什么是黑盒测试?

    黑盒测试又称功能测试、数据驱动测试或基于规格说明书的测试，是一种从用户观点出发的测试。

    黑盒测试示意图



测试人员把被测程序当作一个黑盒子。

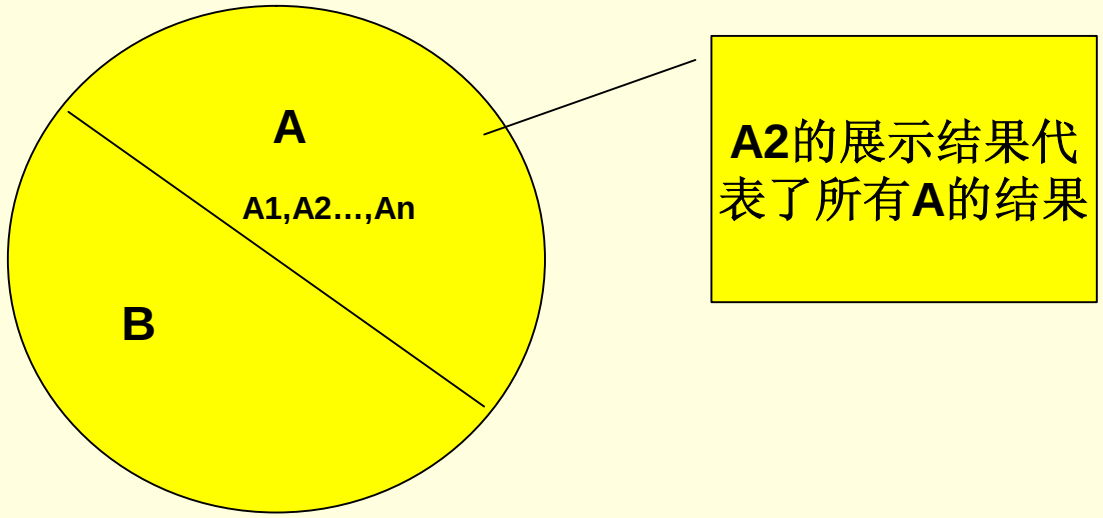
黑盒测试

- **测试用例设计技术**
  - 等价类划分方法
  - 边界值分析方法
  - 错误推测方法
  - 判定表驱动分析方法
  - 因果图方法
  - 场景法

等价类划分法

- 等价类划分法是把程序的输入域划分成若干部分，然后从每个部分中选取少数代表性数据当作测试用例。
- 每一类的代表性数据在测试中的作用等价于这一类中的其他值，也就是说，如果某一类中的一个例子发现了错误，这一等价类中的其他例子也能发现同样的错误；反之，如果某一类中的一个例子没有发现错误，则这一类中的其他例子也不会查出错误。

等价类划分法



- 这样，在每一个等价类中取一个数据作为测试的输入条件，就可以用少量代表性的测试数据取得较好的测试结果。可以减少测试用例的量

划分等价类

- 等价类划分有两种不同的情况:
  - 有效等价类：是指对于程序的规格说明来说是合理的、有意义的输入数据构成的集合。利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。
  - 无效等价类：与有效等价类的定义恰巧相反。
- 设计测试用例时，要同时考虑这两种等价类。因为软件不仅要能接收合理的数据，也要能经受意外的考验。这样的测试才能确保软件具有更高的可靠性。

根据等价类创建测试用例的步骤

- 建立等价类表，列出所有划分出的等价类：

输入条件	有效等价类	无效等价类
…	…	…
…	…	…

·为每个等价类规定一个唯一的编号；

·设计一个新的测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类。重复这一步，最后使得所有有效等价类均被测试用例所覆盖；

·设计一个新的测试用例，使其只覆盖一个无效等价类。重复这一步使所有无效等价类均被覆盖。

例2：NextDate 函数等价类测试用例

- NextDate 函数包含三个变量：month 、 day 和 year ， 函数的输出为输入日期后一天的日期。例如，输入为 2006年3月 7日，则函数的输出为 2006年3月8日 。

要求输入变量 month 、 day 和 year 均为整数值，并且满足下列条件：

（1）1≤month≤12

（2）1≤day≤31

（3） 1812≤year≤2012

例2: NextDate 函数等价类测试用例

- 有效等价类为：
  整数 and
  M_1＝{月份：1≤月份≤12} and
  D_1＝{日期：1≤日期≤31} and
  Y_1＝{年：1812≤年≤2012}

- 若条件（1）~（3）中任何一个条件失效，则 NextDate 函数都会产生一个输出，指明相应的变量超出取值范围，比如“month 的值不在 1-12 范围当中”。显然还存在着大量的 year、month、day 的无效组合，NextDate 函数将这些组合作统一的输出：“无效输入日期”。其无效等价类为：

  非整数 or
  M_2＝{月份：月份<1} or
  M_3＝{月份：月份>12} or
  D_2＝{日期：日期<1} or
  D_3＝{日期：日期>31} or
  Y_2＝{年：年<1812} or
  Y_3＝{年：年>2012}

例2: NextDate 函数等价类测试用例

等价类测试

用例ID	月份	日期	年	预期输出
WR_1	6	15	1912	1912年6月16日
WR_2	－1	15	1912	月份不在1～12中
WR_3	13	15	1912	月份不在1～12中
WR_4	6	－1	1912	日期不在1～31中
WR_5	6	32	1912	日期不在1～31中
WR_6	6	15	1811	年份不在1812～2012中
WR_7	6	15	2013	年份不在1812～2012中

边界值测试用例设计方法

- 边界值分析法：
  - 是等价划分法的一个补充
  - 程序的很多错误发生在输入或输出范围的边界上，因此针对各种边界情况设置测试用例，可以发现不少程序缺陷。
  - 设计方法：
    - 确定边界情况（输入或输出等价类的边界）
    - 选取正好等于、刚刚大于或刚刚小于边界值作为测试数据

边界值分析法

- 与等价划分的区别
  - 边界值分析不是从某等价类中随便挑一个作为代表，而是使这个等价类的每个边界都要作为测试条件。
  - 边界值分析不仅考虑输入条件，还要考虑输出空间产生的测试情况。

边界值设计原则

1.如果输入条件规定了值的范围，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边界的值作为测试输入数据。 。

例如，如果程序的规格说明中规定：“重量在10公斤至50公斤范围内的邮件，其邮费计算公式为……”。作为测试用例，我们应取10及50，还应取10.01,49.99,9.99及50.01等。

边界值设计原则

2.如果输入条件规定了值的个数，则用最大个数、最小个数、比最小个数少一、比最大个数多一的数作为测试数据。

比如，一个输入文件应包括1~255个记录，则测试用例可取1和255，还应取0及256等。

边界值设计原则

3.将规则1和2应用于输出条件，即设计测试用例使输出值达到边界值及其左右的值。

边界值设计原则

- 一般情况，一个边界可以给出3个边界测试用例，自身+左右值

例1：NextDate 函数测试用例

- NextDate 函数包含三个变量：month 、 day 和 year ， 函数的输出为输入日期后一天的日期。例如，输入为 2006年3月 7日，则函数的输出为 2006年3月8日 。

要求输入变量 month 、 day 和 year 均为整数值，并且满足下列条件：

（1）1≤month≤12

（2）1≤day≤31

（3） 1812≤year≤2012

例1: NextDate 函数用例

- 有效等价类为：
- 整数 and

  M_1＝{月份：1≤月份≤12} and

  D_1＝{日期：1≤日期≤31} and

  Y_1＝{年：1812≤年≤2012}
- 若条件（1）~（3）中任何一个条件失效，则 NextDate 函数都会产生一个输出，指明相应的变量超出取值范围，比如“month 的值不在 1-12 范围当中”。显然还存在着大量的 year、month、day 的无效组合，NextDate 函数将这些组合作统一的输出：“无效输入日期”。其无效等价类为：
- 非整数 or

  M_2＝{月份：月份<1} or

  M_3＝{月份：月份>12} or

  D_2＝{日期：日期<1} or

  D_3＝{日期：日期>31} or

  Y_2＝{年：年<1812} or

  Y_c＝{年：年>2012}
- 边界值：M4＝{1，12}, orD4＝{1，31},orY4＝{1821，2012}

例1: NextDate 函数用例

等价类测试

用例ID	月份	日期	年	预期输出
R_1	6	15	1912	1912年6月16日
R_2	－1	15	1912	月份不在1～12中
R_3	13	15	1912	月份不在1～12中
R_4	6	－1	1912	日期不在1～31中
R_5	6	32	1912	日期不在1～31中
R_6	6	15	1811	年份不在1812～2012中
R_7	6	15	2013	年份不在1812～2012中

边界值测试

R_8	1	15	1912	1912年1月16日
R9	0	15	1912	月份不在1～12中
R10	13	15	1912	月份不在1～12中
R11	12	15	1912	1912年12月16日
R12	6	0	1912	日期不在1～31中
R13	6	32	1912	日期不在1～31中
R14	6	15	1812	1812年6月16日
R15	6	15	2012	2012年6月16日

边界的左右值

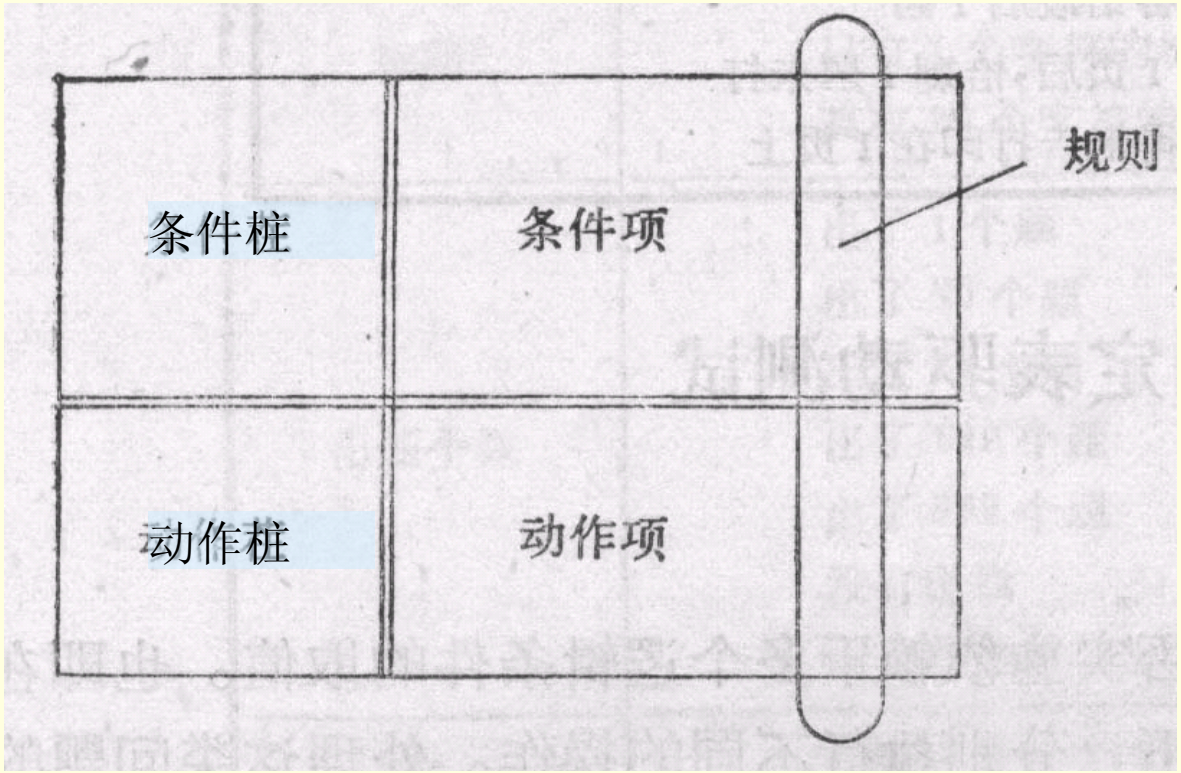
。。。。。。

14.3.3错误推测法

- 错误推测法: 基于经验和直觉推测程序中所有可能存在的各种错误, 从而有针对性的设计测试用例的方法。
- 错误推测方法的基本思想: 列举出程序中所有可能有的错误和容易发生错误的特殊情况,根据他们选择测试用例。
  - 例如, 输入数据和输出数据为0的情况；输入表格为空格或输入表格只有一行. 这些都是容易发生错误的情况。可选择这些情况下的例子作为测试用例。

判定表组成

- 判定表通常由五个部分组成：
  - 条件桩
  - 动作桩
  - 条件项
  - 动作项
  - 规则



判定表

下表是一张关于科技书阅读指南的判定驱动表：3个问题8种情况 2*2*2=8

		1	2	3	4	5	6	7	8
问题	你觉得疲倦吗？	Y	Y	Y	Y	N	N	N	N
	你对内容感兴趣吗？	Y	Y	N	N	Y	Y	N	N
	书中内容使你胡涂吗？	Y	N	Y	N	Y	N	Y	N
建议	请回到本章开头重读	x				x			
	继续读下去		x				x		
	跳到下一章去读							x	x
	停止阅读，请休息			x	x				

”读书指南”判定表

规则合并

- 合并的时候：
  - 要求动作项相同
  - n个条件桩，要求n-1个条件桩的条件项取值相同
  - 将相同的动作项和n-1个条件项对应进行and操作，剩余的1个条件项进行or操作，进行合并
  - 可以进行多次合并，一直至合并结束

		1	2	3	4	5	6	7	8
问题	你觉得疲倦吗?	Y	Y	Y	Y	N	N	N	N
	你对内容感兴趣吗?	Y	Y	N	N	Y	Y	N	N
	书中内容使你胡涂吗?	Y	N	Y	N	Y	N	Y	N
建议	请回到本章开头重读	x				x			
	继续读下去		x				x		
	跳到下一章去读							x	x
	停止阅读，请休息			x	x				

”读书指南”判定表

		1	2	3	4
问题	你觉得疲倦吗?	-	-	Y	N
	你对内容感兴趣吗?	Y	Y	N	N
	书中内容使你胡涂吗?	Y	N	-	-
建议	请回到本章开头重读	x			
	继续读下去		X		
	跳到下一章去读				x
	停止阅读，请休息			x	

化减后的”读书指南”判定表

11.8判定表的建立步骤

- 判定表的建立步骤:（根据软件规格说明）

  ①确定规则的个数.假如有n个条件桩。每个条件桩有两个取值（Y,N）,故有2^n种规则。

  ②列出所有的条件桩和动作桩。

  ③填入条件项。

  ④填入动作项。得到初始判定表。

  ⑤简化.合并相似规则（相同动作）。

  6、设计测试用例覆盖所有合并后的规则

因果图法

概念：

•借助图的方式，设计测试用例，被测程序有多种输入条件，输出结果依赖于输入条件的组合；

•着重分析输入条件的各种组合，每种组合就是一个“因”，它必然有一个输出的结果，这就是“果”；

•与其他的方法相比，更侧重于输入条件的组合

作用：

•能有效检测输入条件的各种组合可能引起的错误和结果；

•借助图进行测试用例的设计

因果图方法举例1

例1：某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，但如果第一列字符不正确，则给出信息L；如果第二列字符不是数字，则给出信息M。

解答：根据题意，原因和结果如下：

因果图方法举例1

编号	原因（条件）	编号	结果（动作）
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		

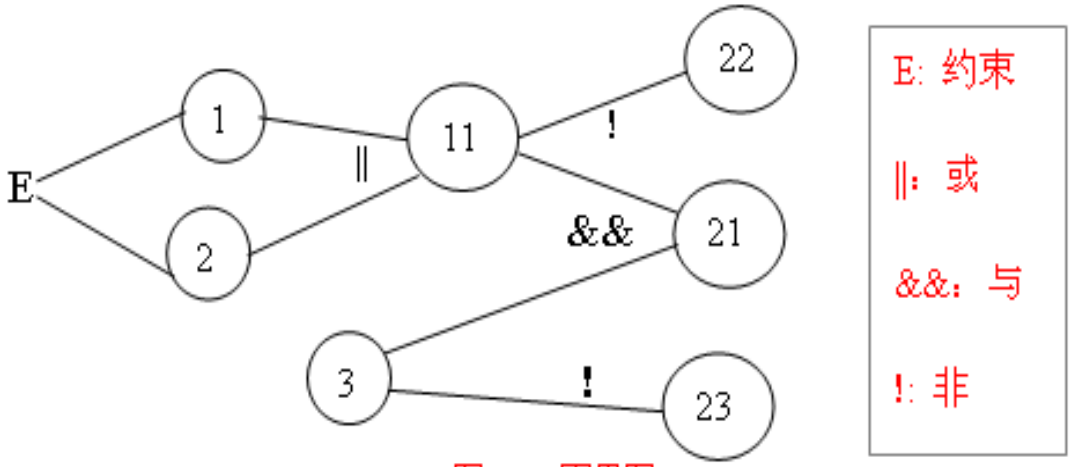


图 6-7 因果图

因果图方法的基本步骤

- 因果图方法最终生成的是判定表。它适合于检查程序输入条件的各种组合情况。利用因果图生成测试用例的基本步骤:

  (1) 分析软件规格说明描述中, 哪些是原因(即输入条件或输入条件的等价类),哪些是结果(即输出条件), 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, 原因与原因之间对应的关系. 根据这些关系,画出因果图。

  (3) 由于语法或环境限制, 有些原因与原因之间,原因与结果之间的组合情况不可能出现. 为表明这些特殊情况, 在因果图上用一些记号表明约束或限制条件。

  (4) 把因果图转换为判定表。

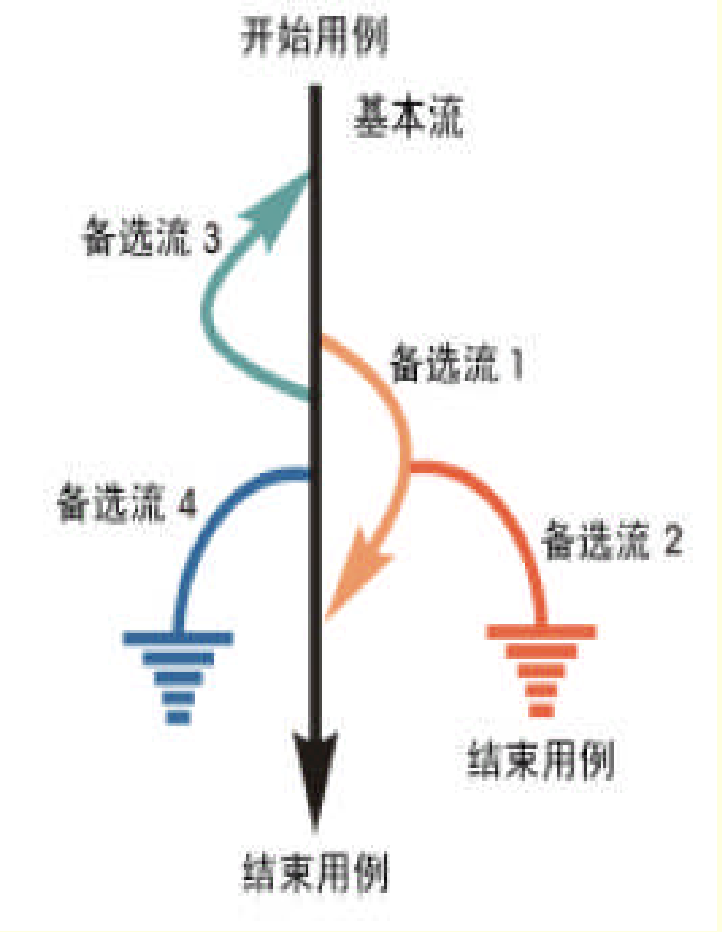
  (5) 把判定表的每一列拿出来作为依据,设计测试用例。

14.3.6场景法

- 现在的软件几乎都是用事件触发来控制流程的，事件触发时的情景便形成了场景，而同一事件不同的触发顺序和处理结果就形成事件流。这种在软件设计方面的思想也可引入到软件测试中，可以比较生动地描绘出事件触发时的情景，有利于测试设计者设计测试用例，同时使测试用例更容易理解和执行。
- 提出这种测试思想的是IBM Rational
- 用例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基本流和备选流。

基本流和备选流

- 右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的路径。备选流用不同的彩色表示，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）；也可能起源于另一个备选流（如备选流 2），或者终止用例而不再重新加入到某个流（如备选流 2 和 4）。



基本流和备选流

- 按照上图中每个经过用例的路径，可以确定以下不同的用例场景：
  - 场景 1 基本流
  - 场景 2 基本流 备选流 1
  - 场景 3 基本流 备选流 1 备选流 2
  - 场景 4 基本流 备选流 3

基本流和备选流

- 场景 5 基本流 备选流 3 备选流 1
- 场景 6 基本流 备选流 3 备选流 1 备选流 2
- 场景 7 基本流 备选流 4
- 场景 8 基本流 备选流 3 备选流 4

注：为方便起见，场景 5、6 和 8 只考虑了备选流 3循环执行一次的情况。

场景法的测试用例设计

- 设计测试用例覆盖所有场景

黑盒测试方法选择的策略

- 对于总策略的选择：可以选择判定表法进行贯穿整个测试案例过程(如功能测试)，在案例中综合使用各种测试方法。
- 如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法或判定表法，在案例中综合使用各种测试方法。
- 对于业务流清晰的系统，可以利用场景法贯穿整个测试案例过程，在案例中综合使用各种测试方法。

黑盒测试方法选择的综合策略

- 黑盒测试：首先进行等价类划分，包括输入条件的等价划分，将无限测试变成有限测试，这是减少工作量和提高测试效率的最有效方法。
- 在任何情况下都必须使用边界值分析方法。经验表明用这种方法设计出测试用例发现程序错误的能力最强。
- 用错误推测法再追加一些测试用例。

测试方法选择的综合策略

- 在单元测试中可以先黑盒测试，后白盒测试，原因是白盒测试比较细致复杂，耗费的测试成本高
- 后白盒测试:根据前期的黑盒测试用例，对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度。如果没有达到要求的覆盖标准，应当再补充足够的测试用例。

测试用例设计的步骤

- 为测试需求确定测试用例
- 为测试用例确定输入输出
- 编写测试用例
- 评审测试用例
- 跟踪测试用例