哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

软件工程
第四章 软件测试
4-3 黑盒测试

杨大易
yangdayi@hit.edu.cn

2024年6月2日

主要内容

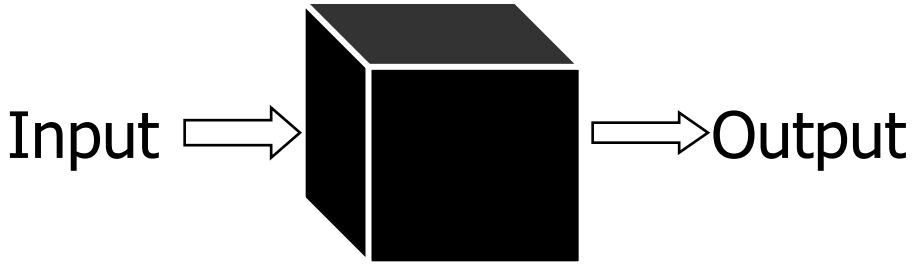
1. 黑盒测试概述
2. 等价类划分方法
3. 边界值方法
4. 黑盒测试 vs 白盒测试

1. 黑盒测试概述

黑盒测试概念

- **黑盒测试(black-box testing):**
  - 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。
  - 将测试对象看做一个黑盒子，测试人员完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。
  - 通常在软件接口处进行。

原理：任何程序都可以看作是将输入定义域取值映射到输出值域的函数



黑盒测试能发现的错误

- 是否有不正确或遗漏的功能？
- 在接口上，输入能否被正确的接受？
- 能否输出正确的结果？
- 是否有数据结构错误或外部信息错误？
- 数据文件访问错误？
- 性能上是否能够满足要求？
- 是否有初始化或终止性错误?

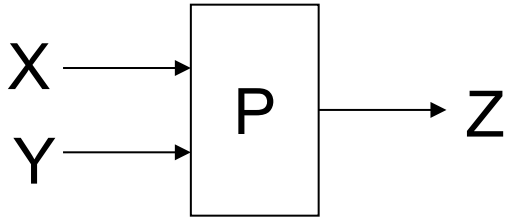
黑盒测试中的“穷举”

- 用黑盒测试发现程序中的错误，必须在所有可能的输入条件和输出条件中确定测试数据，来检查程序是否都能产生正确的输出，但这是不可能的。

——因为穷举测试数量太大，无法完成。

黑盒测试中的“穷举”

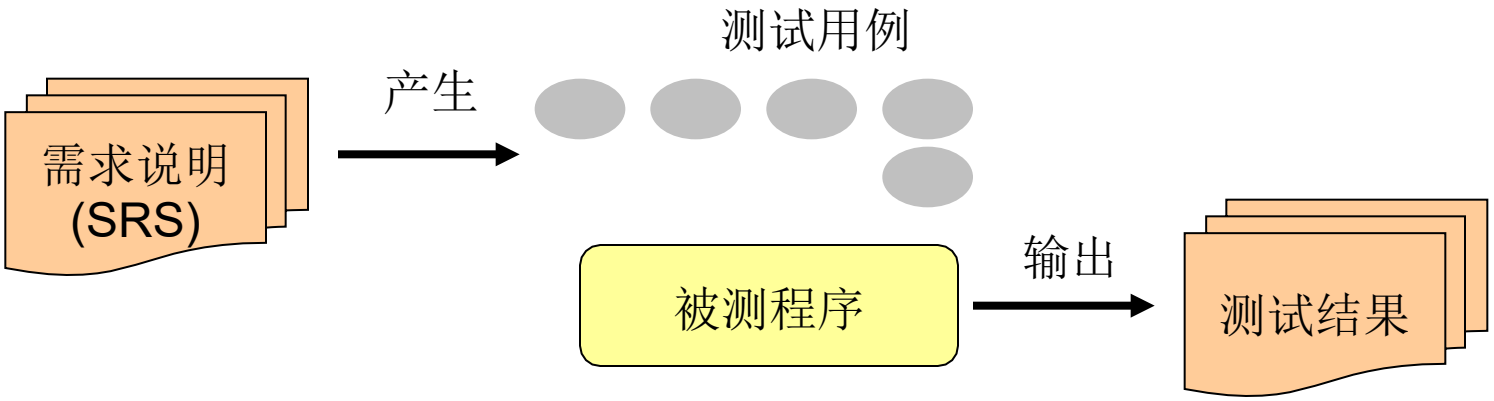
- **举例：程序P有输入整数X和Y及输出量Z，在字长为32位的计算机上运行。**
  - 可能采用的测试数据组：$2^{32}\times2^{32}=2^{64}$
  - 如果测试一组数据需要1毫秒，一年工作365×24小时，完成所有测试需5亿年。



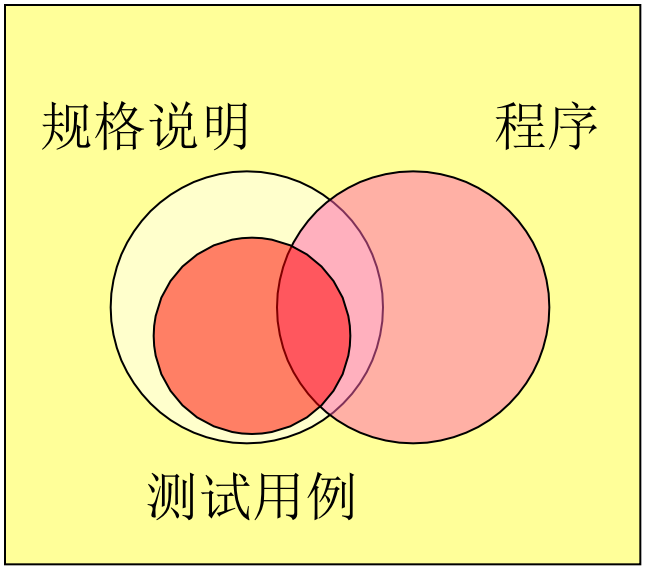
- **因此，测试人员只能在大量可能的数据中，选取其中一部分作为测试用例。**

黑盒测试的实施过程

- **测试计划阶段**
- **测试设计阶段**
  - 依据程序需求规格说明书或用户手册，按照一定规范化的方法进行软件功能划分和设计测试用例。输入数据和期望输出均从需求规格中导出。
- **测试执行阶段**
  - 按照设计的测试用例执行测试；
  - 自由测试(作为测试用例测试的补充)。
- **测试总结阶段**

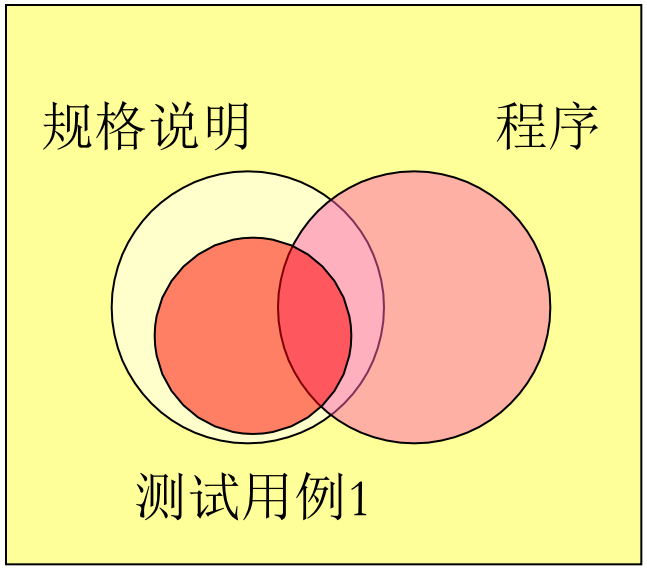


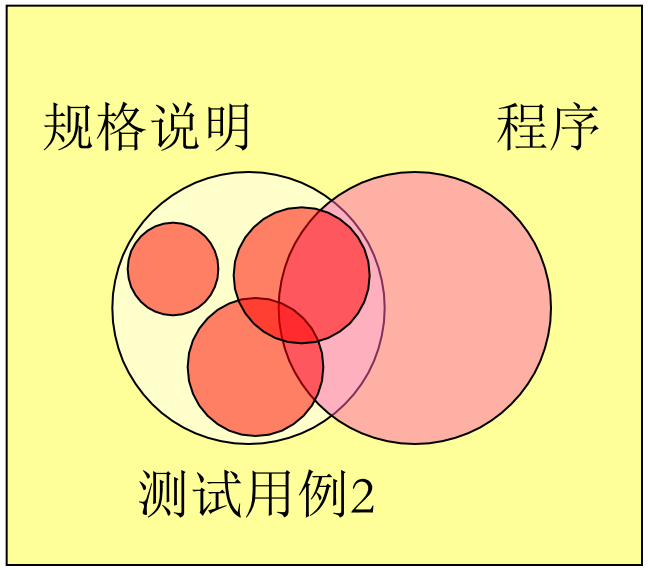
黑盒测试的Venn Diagram



注意：覆盖区域只能在规格说明部分

黑盒测试的Venn Diagram





测试用例所覆盖的规格说明范围越大，就越优良

设计良好的测试用例，使之尽可能完全覆盖软件的规格说明

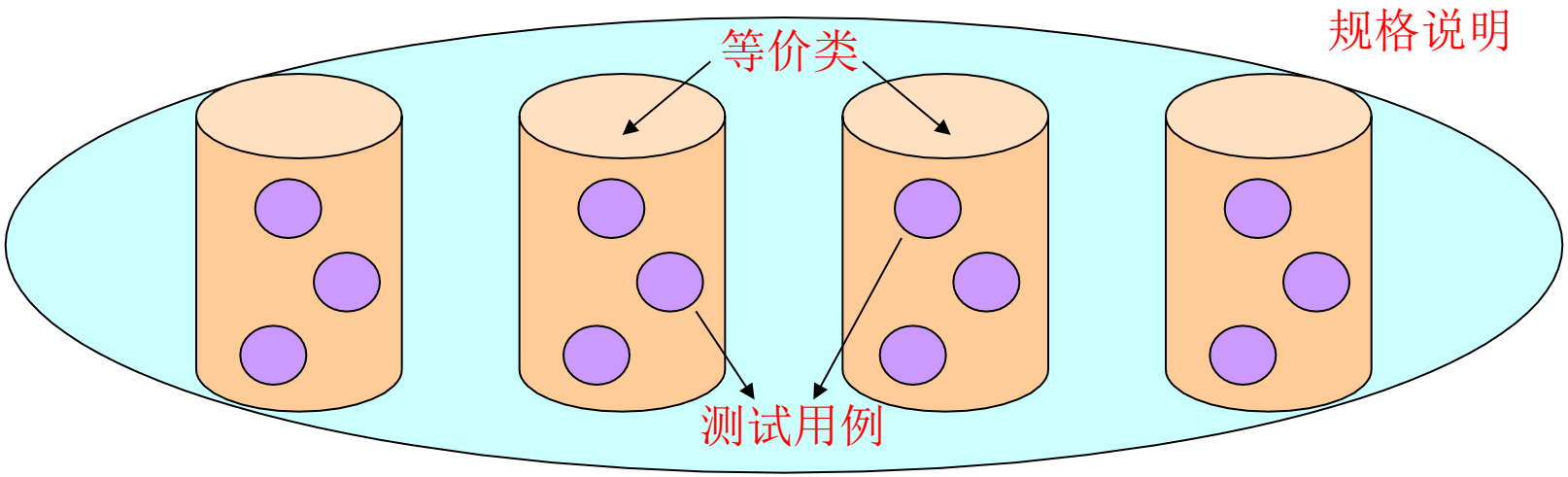
测试用例的设计技术

- **等价类划分**
- **边界值分析**
- 错误推测法
- 因果图法
- 随机测试
- 决策树方法
- 判定表驱动分析方法
- 正交实验设计方法
- 功能图分析方法

2. 等价类划分方法

等价类划分

- **等价类：**输入数据的某个子集，在该子集中的各个输入数据对于揭露程序中的错误都是等效的。并合理地假定“**测试某等价类的代表值就等于对这一类中其它值的测试**”。



- **在每一个等价类中选取少量有代表性的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，并取得较好的测试结果。**

等价类划分(Equivalence partitioning)

- **关键步骤：确定等价类和选择测试用例**
- **基本原则：**
  - 每个可能的输入属于某一个等价类
  - 任何输入都不会属于多个等价类
  - 用等价类的某个成员作为输入时，如果证明执行存在误差，那么用该类的任何其他成员作为输入，也能检查到同样的误差。

有效/无效等价类

- **有效等价类**
  - 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**
  - 对程序的规格说明是不合理的或无意义的输入数据所构成的集合。
  - 无效等价类至少应有一个，也可能有多个。
- **设计测试用例时，要同时考虑这两种等价类。**
  - 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。

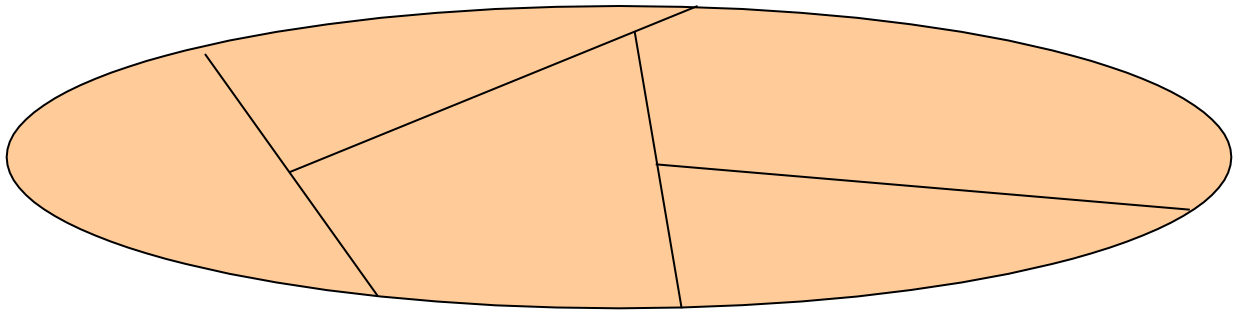
划分等价类的标准

- 划分等价类的标准：“完备测试、避免冗余”

- 将输入数据的集合(P)划分为一组子集($E_1, E_2, \ldots, E_n$)，并尽可能满足：

$$E_1 \cup E_2 \cup \ldots \cup E_n = P$$

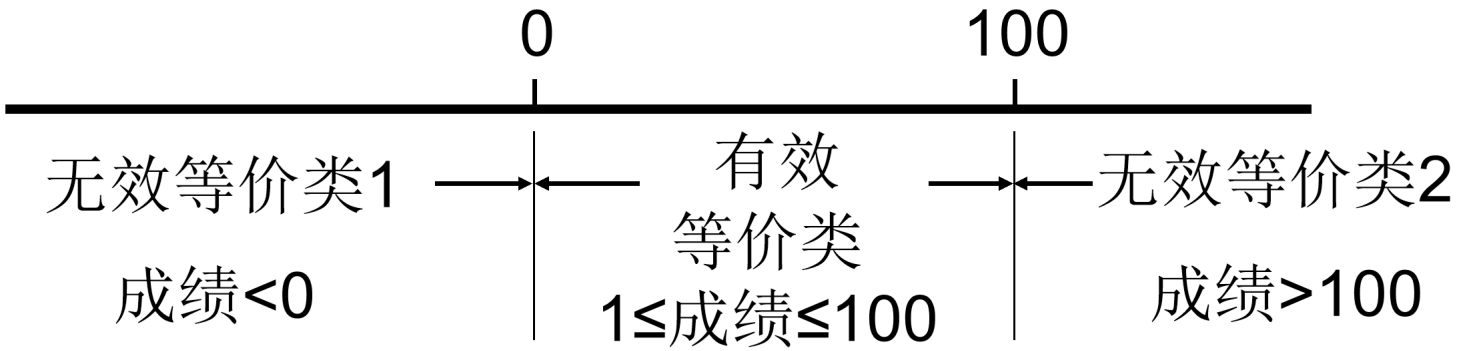
$$E_i \cap E_j = \varnothing$$



确定等价类的六大原则

- **原则1：在输入条件规定了取值范围或值的个数的情况下，则可以确立一个有效等价类和两个无效等价类。**

- **例如：输入值是学生成绩，范围是0～100**



确定等价类的六大原则

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

- 原则3：在输入条件是一个布尔量的情况下，可确定一个有效等价类和一个无效等价类。

确定等价类的六大原则

- **原则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。**

- 例如，输入条件说明学历可为{专科, 本科, 硕士, 博士}四种之一，则分别取这四种值作为四个有效等价类，另外把四种学历之外的任何学历作为无效等价类。

确定等价类的六大原则

- 原则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)。

- 原则6：在确知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，则应再将该等价类进一步的划分为更小的等价类。

设计测试用例

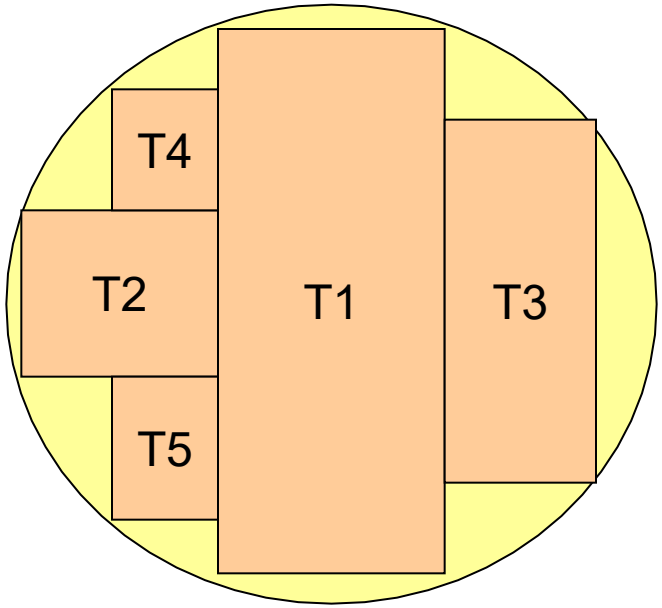
- 测试用例 = {测试数据+期望结果}
- 测试结果 = {测试数据+期望结果+实际结果}

- 在确立了等价类后，可建立等价类表，列出所有划分出的等价类输入数据+期望结果：

输入条件	有效等价类	无效等价类
…	…	…
…	…	…

设计测试用例

- 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，使其尽可能多的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



例1：数据录入问题

- **[例1]某一程序要求输入数据满足以下条件：**
  - 可输入1个或多个数据，每个数据由1-8个字母或数字构成，且第一个字符必须为字母；
  - 如果满足上述条件，则判断为“合法数据”；否则，判断为“非法数据”；

- **用等价类划分方法为该程序进行测试用例设计。**

例1：输入变量问题

- **[例1]某一程序要求输入数据满足以下条件：**
  - 可输入1个或多个数据，每个数据由1-8个字母或数字构成，且第一个字符必须为字母；
  - 如果满足上述条件，则判断为“合法数据”；否则，判断为“非法数据”；

- **划分等价类：**

输入条件	有效等价类	号码	无效等价类	号码
标识符个数	1个	(1)	0个	(6)
	多个	(2)		
标识符字符数	1~8个	(3)	0个	(7)
			>8个	(8)
标识符组成	数字与字母	(4)	含有非“字母或数字”	(9)
第一个标识符	字母	(5)	非字母	(10)

例1：输入变量问题

- **设计测试用例：**

输入条件	有效等价类	号码	无效等价类	号码
标识符个数	1个	(1)	0个	(6)
	多个	(2)		
标识符字符数	1~8个	(3)	0个	(7)
			>8个	(8)
标识符组成	数字与字母	(4)	含有非“字母或数字”	(9)
第一个标识符	字母	(5)	非字母	(10)

测试用例编号	测试用例内容	覆盖的等价类
1	a2ku83t	(1)(3)(4)(5)
2	a2ku83t, fta, b2	(2)(3)(4)(5)
3		(6)
4	a2ku83t, , b2	(7)
5	a2ku83t29, fta	(8)
6	a2ku8$t	(9)
7	92ku83t	(10)

例2：日期检查

- [例2] 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。

- 用等价类划分法设计测试用例，来测试程序的“日期检查功能”。

例2：日期检查

- **[例2] 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。**

- **划分等价类：**

输入等价类	有效等价类	号码	无效等价类	号码
日期的类型及长度	6位数字字符	①	有非数字字符 少于6位数字字符 多于6位数字字符	② ③ ④
年份范围	在1990~2049之间	⑤	小于1990 大于2049	⑥ ⑦
月份范围	在01~12之间	⑧	等于00 大于12	⑨ ⑩

例2：日期检查

输入等价类	有效等价类	号码	无效等价类	号码
日期的类型及长度	6位数字字符	①	有非数字字符 少于6位数字字符 多于6位数字字符	② ③ ④
年份范围	在1990~2049之间	⑤	小于1990 大于2049	⑥ ⑦
月份范围	在01~12之间	⑧	等于00 大于12	⑨ ⑩

- **设计有效等价类的测试用例：**

测试用例编号	测试用例内容	覆盖的等价类
1	200711	(1)(5)(8)

- **设计无效等价类的测试用例：**

测试用例编号	测试用例内容	覆盖的等价类
1	07June	(2)
2	20076	(3)
3	2007011	(4)
4	198912	(6)
5	205401	(7)
6	200700	(8)
7	200713	(10)

3. 边界值分析法

边界值分析

- 边界值分析是等价类测试的补充，主要是考虑等价类的边界条件，在等价类的“边缘”选择元素。

- 长期的测试经验表明：大量的错误是发生在输入或输出范围的边界上，而不是发生在输入输出范围的内部。

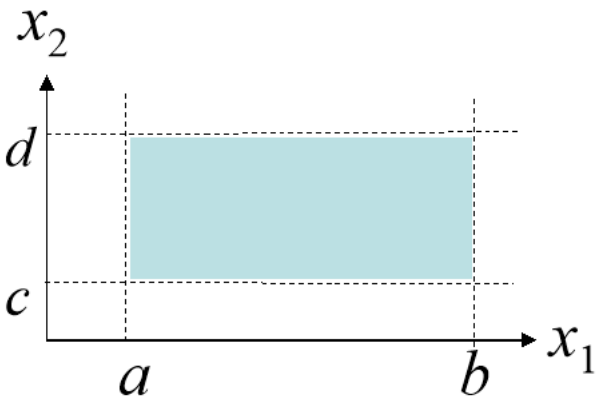
- 因此针对各种边界情况设计测试用例，可以查出更多的错误。

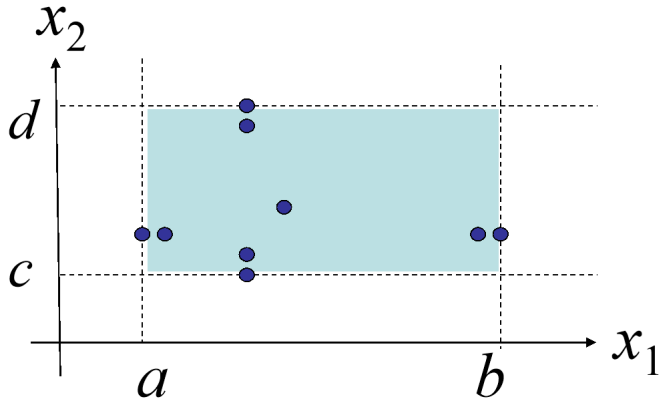
确定边界

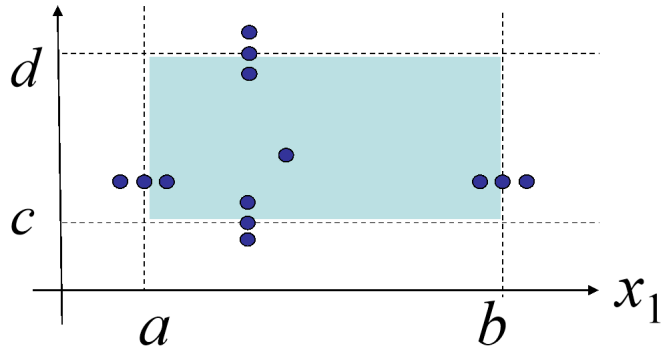
- **使用边界值分析方法设计测试用例，首先应确定边界情况：**
  - 通常输入和输出等价类的边界，就是应着重测试的边界情况。
  - **选取正好等于、刚刚大于、刚刚小于边界的值作为测试数据，**而不是选取等价类中的典型值或任意值作为测试数据。

- **例：在[R1, R2] 的取值区间中，应如何选择？**

根据边界确定测试用例







5个测试用例:
最小值；略高于最小值；
正常值；
略低于最大值；最大值；

n个变量，有4n+1个测试用例

在5个测试用例的基础上增如:
略高于最大值；
略低于最小值；

常见的边界值

- **通常情况下，软件测试所包含的边界检验有几种类型：数字、字符、位置、重量、大小、速度、方位、尺寸、空间等。**
- **相应地，以上类型的边界值应该在：最大/最小、首位/末位、上/下、最快/最慢、最高/最低、 最短/最长、 空/满等情况下。**
  - 对16-bit 的整数而言，32767和-32768是边界
  - 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
  - 数组元素的第一个和最后一个
  - 循环的第0次、第1次和倒数第2次、最后1次

边界值分析的原则

- **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，应取10、50、10.01、9.99 、49.99及50.01等

边界值分析的原则

- **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1，比最大个数多1的数据作为测试数据。**

- 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。

边界值分析的原则

- **原则3：将原则1和原则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。**

- 例如，某程序的规格说明要求计算出“每月保险金扣除额为**0**至**1165.25**元”，其测试用例设计为可使每月保险金扣除额为**0.00**及**1165.25**、**-0.01**、**0.01**及**1165.26**、**1165.24**等。

- 再如，某程序要求每次“最少显示**1**条、最多显示**4**条查询结果”，这时应考虑的测试用例包括**1**和**4**，还应包括**0**和**5**等。

边界值分析的原则

- 原则4：如果程序的规格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。
- 原则5：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。
- 原则6：分析规格说明，找出其它可能的边界条件。

[例1] 三角形

- 假定三角形每边边长的取范围值设值为**[1, 100]**

测试用例	a	b	c	预期输出
Test1	**60**	**60**	**1**	**等腰三角形**
Test2	**60**	**60**	**2**	**等腰三角形**
Test3	**60**	**60**	**60**	**等边三角形**
Test4	**50**	**50**	**99**	**等腰三角形**
Test5	**50**	**50**	**100**	**非三角形**
Test6	**60**	**1**	**60**	**等腰三角形**
Test7	**60**	**2**	**60**	**等腰三角形**
Test8	**50**	**99**	**50**	**等腰三角形**
Test9	**50**	**100**	**50**	**非三角形**
Test10	**1**	**60**	**60**	**等腰三角形**
Test11	**2**	**60**	**60**	**等腰三角形**
Test12	**99**	**50**	**50**	**等腰三角形**
Test13	**100**	**50**	**50**	**非三角形**

[例2] NextDate()

- 在**NextDate()**函数中：
  - 1≤month≤12
  - 1≤day≤31
  - 1912≤year≤2050

测试用例	mouth	day	year	预期输出
Test1	6	15	1911	1911.6.16
Test2	6	15	1912	1912.6.16
Test3	6	15	1913	1913.6.16
Test4	6	15	1975	1975.6.16
Test5	6	15	2049	2049.6.16
Test6	6	15	2050	2050.6.16
Test7	6	15	2051	2051.6.16
Test8	6	-1	2001	day超出[1…31]
Test9	6	1	2001	2001.6.2
Test10	6	2	2001	2001.6.3
Test11	6	30	2001	2001.7.1
Test12	6	31	2001	输入日期超界
Test13	6	32	2001	day超出[1…31]
Test14	-1	15	2001	Mouth超出[1…12]
Test15	1	15	2001	2001.1.16
Test16	2	15	2001	2001.2.16
Test17	11	15	2001	2001.11.16
Test18	12	15	2001	2001.12.16
Test19	13	15	2001	Mouth超出[1…12]

[例2] NextDate()

- 另一种更详尽的划分方法
  - D1 = { 1≤date＜last day of the month }
  - D2 = { last day of the month }
  - D3 = { Dec. 31 }
  - M1 = { 30 - day months }
  - M2 = { 31 - day months }
  - M3 = { Feb. }
  - Y1 = { 2000 }
  - Y2 = { leap year }
  - Y3 = { not leap year }

[例2] NextDate()

序号	Month
1	2
2	2
3	2
4	2
5	2
6	2
7	2
8	2
9	2
10	2
11	2
12	2

序号	Month
13	6
14	6
15	6
16	6
17	6
18	6
19	6
20	6
21	6
22	6
23	6
24	6

序号	Month
25	8
26	8
27	8
28	8
29	8
30	8
31	8
32	8
33	8
34	8
35	8
36	8

序号	Month	Day	Year	期望输出
37	12	31	2000	2001-1-1
38	12	31	1996	1997-1-1
39	12	31	2002	2003-1-1
40	13	14	2000	无效的输入日期
41	13	30	1996	无效的输入日期
42	13	31	2002	无效的输入日期
43	3	0	2000	无效的输入日期
44	3	32	1996	无效的输入日期
45	9	14	0	无效的输入日期
46	0	0	0	无效的输入日期
47	-1	14	2000	无效的输入日期
48	11	-1	2002	无效的输入日期

4. 黑盒测试 vs 白盒测试

黑盒测试的优点

- 对于较大的代码单元来说(子系统甚至系统级)，黑盒测试比白盒测试效率要高；
- 测试人员不需要了解实现的细节，包括特定的编程语言；
- 测试人员和编码人员是彼此独立的；
- 从用户的视角进行测试，很容易被理解和接受；
- 有助于暴露任何规格不一致或有歧义的问题；
- 测试用例可以在规格完成之后马上进行。

黑盒测试的缺点

- 只有一小部分可能的输入被测试到，要测试每个可能的输入流几乎是不可能的；
- 没有清晰的和简明的规格，测试用例是很难设计的；
- 如果测试人员不被告知开发人员已经执行过的用例，在测试数据上会存在不必要的重复；
- 会有很多程序路径没有被测试到；
- 不能直接针对特定程序段测试，这些程序段可能很复杂(因此可能隐藏更多的问题)；
- 大部分和研究相关的测试都是直接针对白盒测试的。

白盒测试的优缺点

- **优点**
  - 迫使测试人员去仔细思考软件的实现；
  - 可以检测代码中的每条分支和路径；
  - 揭示隐藏在代码中的错误；
  - 对代码的测试比较彻底；
  - 最优化。
- **缺点**
  - 昂贵；
  - 无法检测代码中遗漏的路径和数据敏感性错误；
  - 不验证规格的正确性。

黑盒测试 vs. 白盒测试

黑盒(功能)测试	白盒(结构)测试
只利用规格说明标识测试用例	只利用程序源代码标识测试用例
如果程序实现了未描述的行为，功能测试无法意识到。	如果已描述的行为未能实现，结构性测试无法意识到。
冗余度大，可能会有漏洞	具有覆盖率指标

二者的对比

- 白盒测试只考虑测试软件产品，它不保证完整的需求规格是否被满足。
- 黑盒测试只考虑测试需求规格，它不保证实现的所有部分是否被测试到。
- 黑盒测试会发现遗漏的缺陷，指出规格的哪些部分没有被完成。而白盒测试会发现代码方面缺陷，指出哪些实现部分是错误的。
- 白盒测试比黑盒测试成本要高得多。它需要在测试可被计划前产生源代码，并且在确定合适的数据和决定软件是否正确方面需要花费更多的工作量。
- 一个白盒测试的失败会导致一次修改，这需要所有的黑盒测试被重复执行并且重新决定白盒测试路径。

有了黑盒测试为何还需要白盒测试

- 黑盒测试只能观察软件的外部表现，即使软件的输入输出都是正确的，却并不能说明软件就是正确的。因为程序有可能用错误的运算方式得出正确的结果，例如“负负得正，错错得对”，只有白盒测试才能发现真正的原因。
- 白盒测试能发现程序里的隐患，例如内存泄漏、误差累计问题。在这方面，黑盒测试存在严重的不足。

有了白盒测试为何还需要黑盒测试

- 通过了白盒测试只能说明程序代码符合设计需求，并不能说明程序的功能符合用户的需求。如果程序的系统设计偏离了用户需求，即使100%正确编码的程序也不是用户所要的。

课外阅读：Pressman教材第14-17章

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

结束

2024年6月2日