

软件质量与评测技术

Software Quality & Evaluation Technology

计算机学院 单纯
sherryshan@bit.edu.cn
2025年11月

测试设计技术

Test Design Techniques

计算机学院 单纯

sherryshan@bit.edu.cn

2025年12月15日

课后思考 (3)

■ 课内实验 (35%)

– 实验1：单元测试实践

- (一) 程序说明
- (二) 源程序

F. 等价类划分方法设计测试用例 (9)

- 等价类划分的目标是把可能的测试用例组合缩减到仍然足以满足软件测试需求为止。
 - 。选择了不完全测试，就要冒一定的风险，所以必须仔细选择分类

F. 等价类划分方法设计测试用例 (10)

- 科学有时也是一门艺术，等价分配的方法有可能不客观。测试同一复杂程序的两个软件测试员，可能会制定出两组不同的等价区间。只要审查等价区间的人都认为它们足以覆盖测试对象就可以了

3.1.6.2 边界值分析（1）

- 美国陆军（CECOM）对其软件进行研究，令人吃惊地发现，大量缺陷都是边界值缺陷
- 边界值测试的基本原理是错误更可能出现在输入变量的极值附近
- 边界值分析是等价划分的扩展，包括等价类+划分的边界值，边界值通常是等价类的界限

3.1.6.2 边界值分析（2）

- 边界值分析是等价类划分的一种变体和改良，主要有两方面的区别
 - 第一，不是在等价类中选择一个元素作为代表，而是在挑选元素时使得等价类（EC）的边界受到测试
 - 第二，不只关注输入条件，也注意输出条件。可以输出什么？输出的类是什么？我们应该提供什么样的输入才能确保获得一个有用的类组能代表符合要求的输出？

3.1.6.2 边界值分析（3）

- 边界值分析法是基于可靠性理论中称为“单故障”的假设，即有两个或两个以上故障同时出现而导致软件失效的情况很少，也就是说，软件失效基本上是由单故障引起的
 - 边界值分析利用输入变量的最小值（min）、略大于最小值（min+）、输入值域内的任意值（nom）、略小于最大值（max-）和最大值（max）来设计测试用例

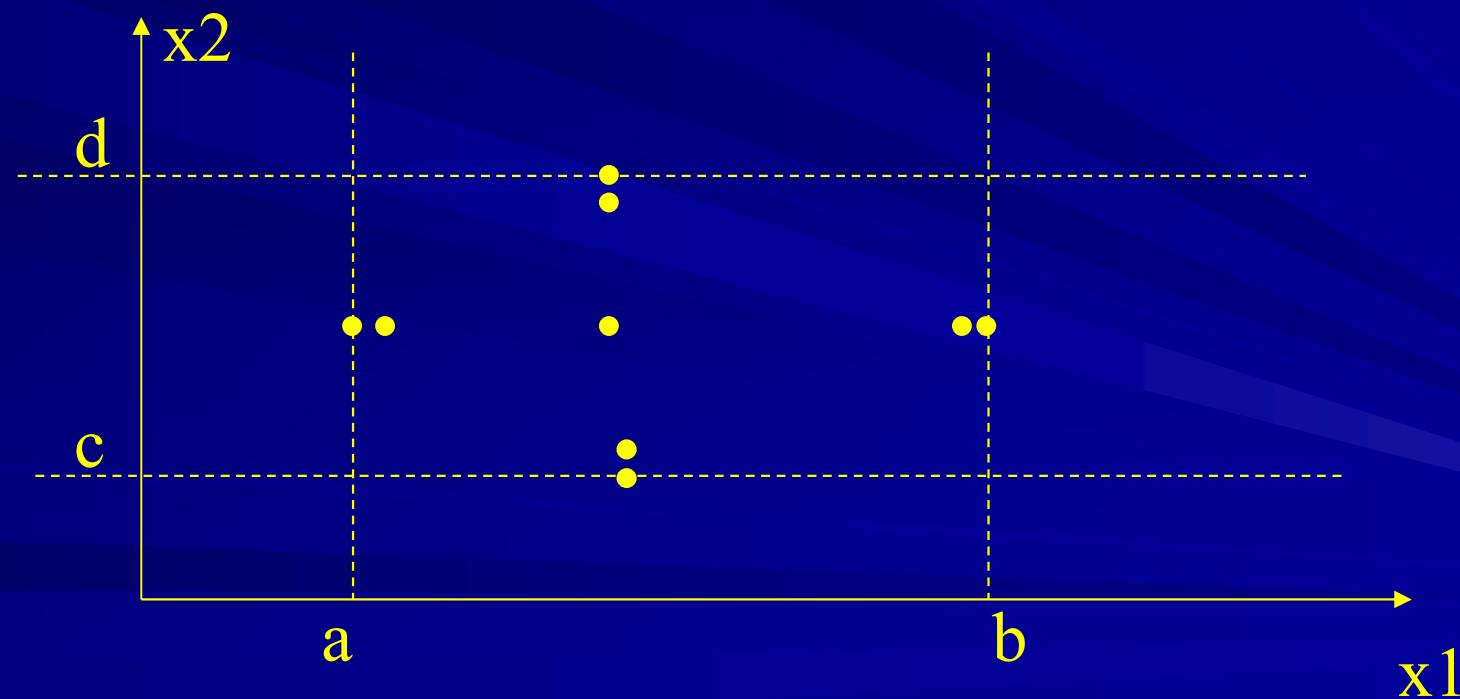
3.1.6.2 边界值分析（4）

■ 有两个输入变量的程序F的边界值分析测试用例

- { $\langle x_{1\text{nom}}, x_{2\text{min}} \rangle, \langle x_{1\text{nom}}, x_{2\text{min+}} \rangle,$
 $\langle x_{1\text{nom}}, x_{2\text{nom}} \rangle, \langle x_{1\text{nom}}, x_{2\text{max-}} \rangle,$
 $\langle x_{1\text{nom}}, x_{2\text{max}} \rangle, \langle x_{1\text{min}}, x_{2\text{nom}} \rangle,$
 $\langle x_{1\text{min+}}, x_{2\text{nom}} \rangle, \langle x_{1\text{max-}}, x_{2\text{nom}} \rangle,$
 $\langle x_{1\text{max}}, x_{2\text{nom}} \rangle$ }

3.1.6.2 边界值分析（5）

- 有两个输入变量的程序F的边界值分析测试用例（续）
 - 如下图所示



3.1.6.2 边界值分析（6）

- 对于一个含有n个变量的程序，保留其中一个变量，让其余的变量取正常值，被保留的变量依次取min、min+、nom、max-、max值，对每个变量都重复进行，这样对于一个n变量的程序，边界值分析测试程序会产生 $4n+1$ 个测试用例

3.1.6.2 边界值分析（7）

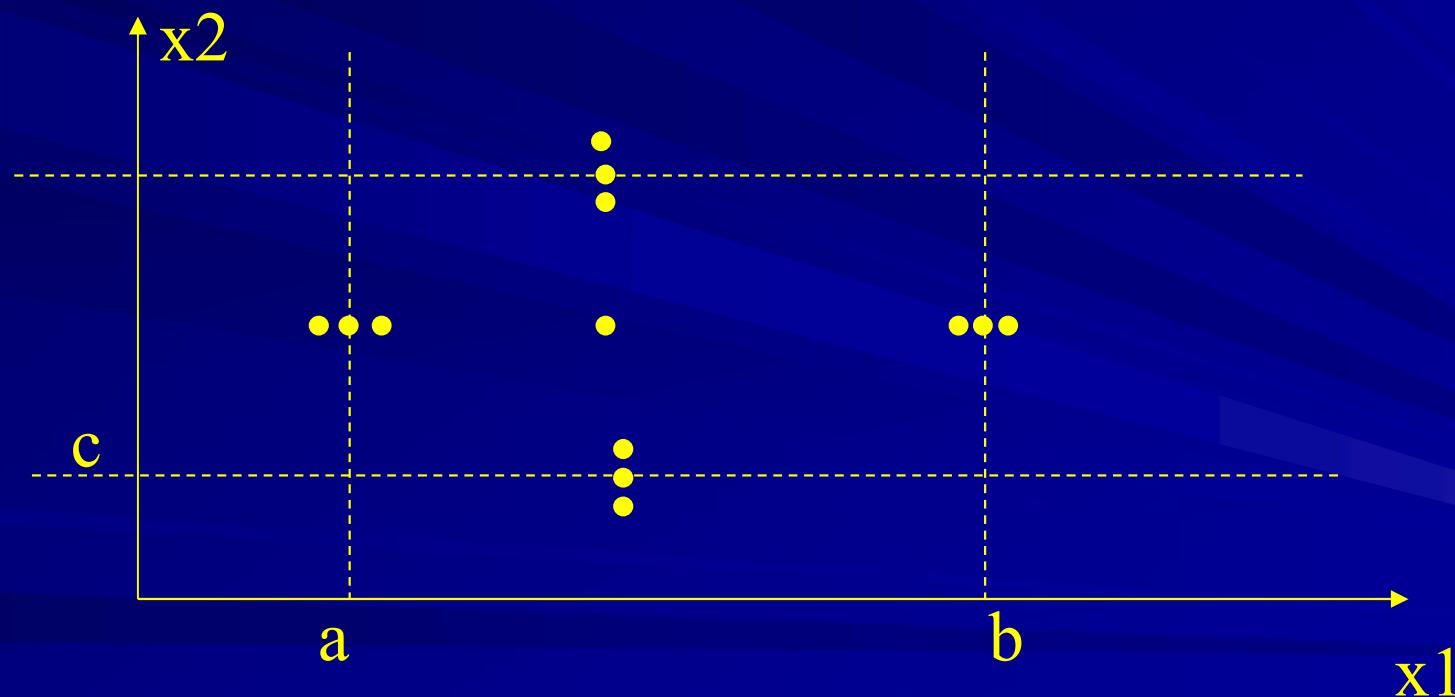
■ 健壮性测试

- 健壮性测试是边界值分析测试的一种扩展，除了取5个边界值外，还需要考虑采用一个略超过最大值（**max+**）以及略小于最小值（**min-**）的取值，检查超过极限值时系统的情况
- 健壮性测试最有意义的部分不是输入，而是预期的输出。要观察例外情况如何处理，比如某个部分的负载能力超过其最大值时可能出现的情形

3.1.6.2 边界值分析 (8)

■ 健壮性测试（续）

– 有两个输入变量的程序F的健壮性测试用例



3.1.6.2 边界值分析（9）

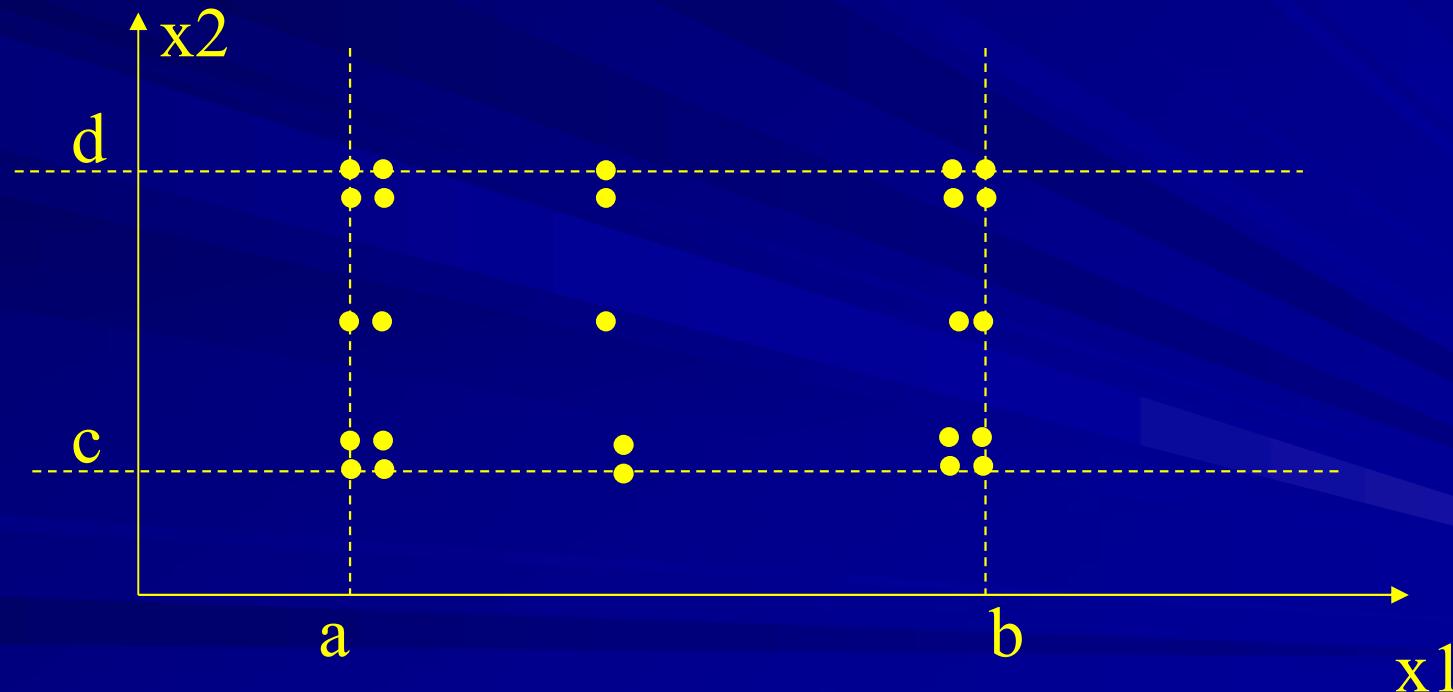
■ 最坏情况测试

- 边界值分析采用了可靠性理论的单缺陷假设。拒绝这种假设，关心当多个变量取极值时会出现什么情况
- 对每个变量，首先进行包含最小值、略高于最小值、正常值、略低于最大值和最大值五元素集合的测试，然后对这些集合进行笛卡尔积计算，以生成测试用例

3.1.6.2 边界值分析 (10)

■ 最坏情况测试 (续)

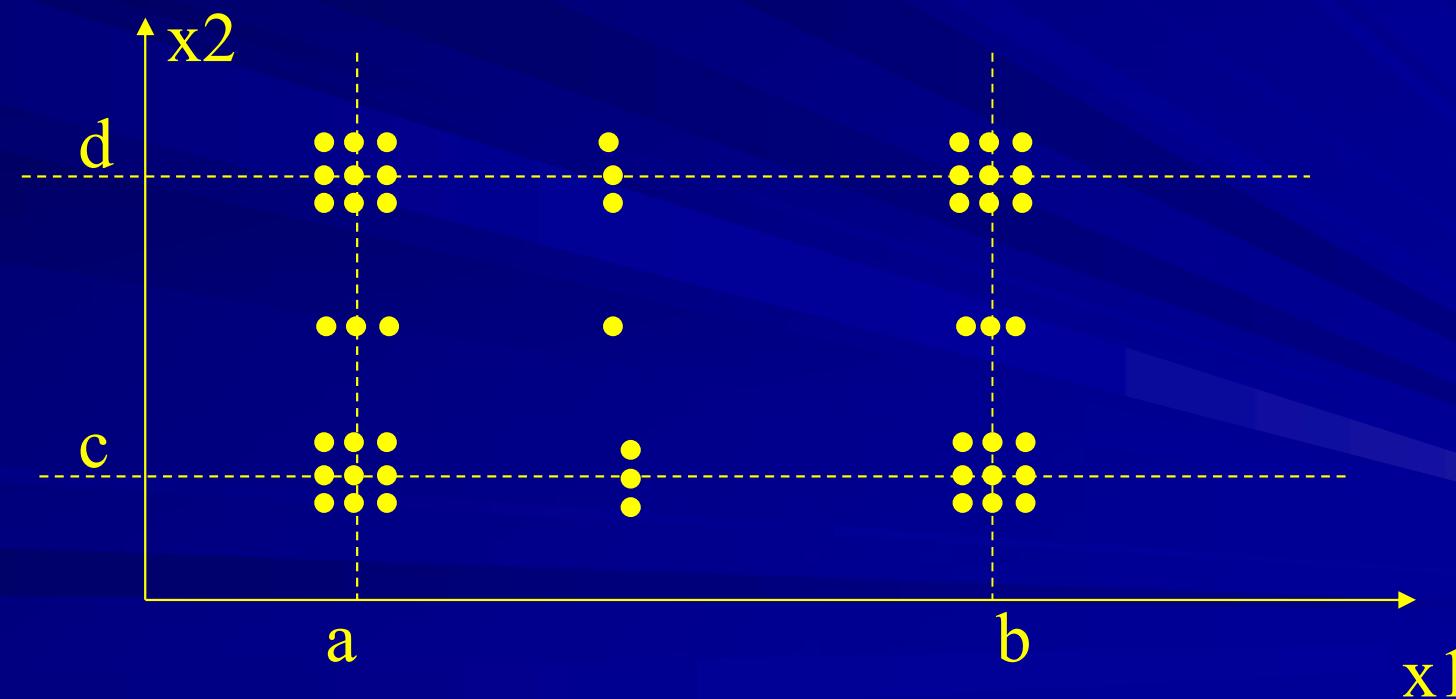
– 有两个输入变量的程序F的最坏情况测试用例



3.1.6.2 边界值分析 (11)

■ 健壮最坏情况测试

- 使用健壮性测试的七元素集合的笛卡尔积生成测试用例



实例（1）

■ 判断三角形问题，问题的提法是：程序接受 3 个整数 a 、 b 和 c 作为输入，用做三角形的边。整数 a 、 b 和 c 必须满足以下条件

- C1. $1 \leq a \leq 200$
- C2. $1 \leq b \leq 200$
- C3. $1 \leq c \leq 200$
- C4. $a < b + c$
- C5. $b < a + c$
- C6. $c < a + b$

实例（2）

■ 按照构成三角形的条件

- C1、C2和C3，整数a、b、c的边界值为1和200，稍超出边界的值为0和201
- C4、C5和C6的边界值分别是 $b+c=a+1$ 、 $a+c=b+1$ 和 $a+b=c+1$

实例 (3)

■ 可取的测试用例如下表所示

测试用例	a	b	c
a, b, c 达到最小边界	1	1	1
a, b, c 达到最大边界	200	200	200
a 超出最小边界	0	1	1
b 超出最小边界	1	0	1
c 超出最小边界	1	1	0
a 超出最大边界	201	200	200
b 超出最大边界	200	201	200
c 超出最大边界	200	200	201

实例 (4)

■ 可取的测试用例如下表所示（续）

测试用例	a	b	c
$b+c=a+1$, 到达边界	4	2	3
$a+c=b+1$, 到达边界	2	4	3
$a+b=c+1$, 到达边界	2	3	4
$b+c=a$, 小于边界	3	1	2
$a+c=b$, 小于边界	1	3	2
$b+a=c$, 小于边界	1	2	3

实例 (5)

- 到达边界的值是合理的输入，只要最少的测试数据就可以
- 超出边界的值是不合理的输入，必须逐个检测，在检查某一个数据时让其他的数据为边界或边界内合理的值
- 分析规格说明，找出其他可能的边界条件

边界条件类型 (1)

- 数值、速度、字符、地址、位置、尺寸、数量等等
- 第一个 / 最后一个，最小值 / 最大值，开始 / 完成，超过 / 在内，空 / 满，最短 / 最长，最慢 / 最快，最早 / 最迟，最大 / 最小，最高 / 最低，相邻 / 最远等等

边界条件类型 (2)

- 越界测试通常是简单地加1或者很小的数（对于最大值）和减少1或者很小的数（对于最小值）

常见的边界值

- 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
- C++ 语言中，`int A[10]: A[0]...A[9]`
- 循环的第 0 次、第 1 次和倒数第 2 次、最后一次

3.1.6.2 边界值分析 (12)

■ 应用边界值分析法进行测试用例设计时，应遵循以下一些原则

- （1）如果输入条件对取值范围进行了限定，则应以边界内部以及刚超出范围边界外的值作为测试用例
- （2）如果对取值的个数进行了界定，则应分别以最大、稍小于最大、稍大于最大、最小、稍小于最小、稍大于最小个数作为测试用例

3.1.6.2 边界值分析（13）

■ 应用边界值分析法进行测试用例设计时，应遵循以下一些原则（续）

- （3）对于输出条件，同样可以应用上面提到的两条原则来进行测试用例设计
- （4）如果程序规格说明书中指明输入或者输出域是一个有序的集合，如顺序文件、表格等，则应注意选取有序集合中的第一个和最后一个元素作为测试用例

3.1.6.3 错误猜测法（1）

- 错误猜测是基于经验和其他一些测试技术（如边界值测试）的。测试人员凭经验猜测错误的类型及在特定的软件中错误发生的位置，并设计测试用例去发现它们
 - 例如，如果所有的资源都需要动态申请，最容易出错的地方就是资源释放的语句

3.1.6.3 错误猜测法（2）

- 错误推測法的基本想法是：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，根据它们选择测试用例
- 可以利用不同测试阶段的经验和对软件系统的认识来设计测试用例
 - 例如，在单元测试中某程序模块已经遇到错误，在系统测试中可以在这些可能出现问题的地方再组织测试用例
 - 在前一个版本中发现的常见错误在下一个版本的测试中有针对性地设计测试用例

3.1.6.3 错误猜测法（3）

- 根据以上想法，测试用例的设计原则
 - 客观因素：产品以前版本已出现的问题
 - 已知因素：语言、操作系统、浏览器的限制可能带来的问题
 - 经验：由模块之间关联所联想到的测试；由修复软件的错误可能会带来的问题
- 对于一个有经验的工程师，错误猜测可能是一个最有效的设计测试用例发现问题的方法

3.1.6.3 错误猜测法（4）

- 一个好的错误猜测可能发现被别的测试方法很容易地遗漏的错误。但反过来，错误猜测也可能是白白浪费时间
- 为了最大限度地利用有效的经验并逐步丰富测试用例设计技术，可以建立一个错误类型的列表（常见错误检查表）。这个列表是总结了早期的单元测试的经验而建立的，可以帮助猜测单元中的错误会在哪里。这个列表应通过不断维护来提高错误猜测的有效性

实例（1）

■ EDGE4.0中文版开发

– EDGE是英文**Effective Data Gathering Expertise**的缩写，它是由美国IMA（**Information Management Associates**）公司开发的一种基于呼叫中心（**Call Center**）的电话综合服务系统。它通过呼叫中心以及顾客交互活动集线器（**Hub**）支持销售、市场调查和顾客服务管理等活动

实例（2）

■ EDGE4.0中文版开发（续）

- 如果说等价划分法和边界值分析法均有线索可寻，那么错误猜测法则更多地依赖于测试人员的直觉与经验
- 对于一个汉化软件而言，在哪些地方出现中文特有问题也许是无法预知的，但我们至少有这样的一些方面可供考虑，那就是：中英文习惯差异，如日期表示法

实例（3）

■ EDGE4.0中文版开发（续）

— 英文中的日期常用表示法一般有如下几种：月-日-年，日-月-年，有时月份还多用月份的英文单词缩写来表示，而中文的日期表示法则多为年-月-日的格式。在软件的汉化过程中能否解决好中英文转换是一个不可忽视的重要问题。基于这种考虑，对于涉及到中英文日期转换的地方应是我们重点的测试之处

实例（4）

■ EDGE4.0中文版开发（续）

– 在信息量激增的今天，对数据库的有效管理是数据库管理系统（DBMS）的基本功能，但如何实现数据库的转储，无差错地实现数据库之间的移植，提高原有数据库的利用率，避免重复劳动必将是各种数据库管理软件的竞争点。EDGE软件包在EDGE level和Project level的Database中都提供了数据库的Import和Export功能，以实现数据库的转储备份功能，实现EDGE程序数据与非EDGE程序数据间的转换

实例（5）

■ EDGE4.0中文版开发（续）

- 经过设计测试用例发现，汉化软件仍然保留有英文习惯的日期格式，这不能不说是一处很严重的汉化遗漏，而且，由此而引发了更严重的导入导出中的日期转换错误

3.1.6.4 判定表法（1）

- 判定表是一种用来表示和分析复杂逻辑关系的工具，最适合描述在多个逻辑条件取值的组合所构成的复杂情况下，分别要执行哪些不同的动作
- 判定表可以把复杂的逻辑关系和多种条件组合的情况表达得既明确又具体

3.1.6.4 判定表法（2）

■ 判定表的组成



3.1.6.4 判定表法（3）

- 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（Action 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（Condition Entry）：列出针对它所列条件的取值，在所有可能情况下的真假值
- 动作项（Action Entry）：列出在条件项的各种取值情况下应该采取的动作

3.1.6.4 判定表法（4）

- 规则：任何一个条件组合的特定取值及其相应要执行的操作。在判定表中贯穿条件项和动作项的一列就是一条规则。显然，判定表中列出多少组条件取值，也就有多少条规则，条件项和动作项就有多少列

3.1.6.4 判定表法（5）

■ 判定表建立应依据软件规格说明，步骤如下

- 确定规则的个数。假如有n个条件，每个条件有两个取值（0, 1），故有 2^n 种规则
- 列出所有的条件桩和动作桩
- 填入条件项
- 填入动作项，制定初始判定表
- 简化，合并相似规则或者相同动作

3.1.6.4 判定表法 (6)

■ Beizer指出了适合使用判定表设计测试用例的条件

- 规格说明以判定表的形式给出，或很容易转换成判定表
- 条件的排列顺序不影响执行哪些操作
- 规则的排列顺序不影响执行哪些操作
- 当某一规则的条件已经满足，并确定要执行的操作后，不必检验别的规则
- 如果某一规则要执行多个操作，这些操作的执行顺序无关紧要

实例1 (1)

■ 维修机器问题

- “.....对于功率大于50马力的机器，并且维修记录不全或已运行10年以上的机器，应给予优先的维修处理.....”

实例1（2）

■ A.列出所有的条件桩和动作桩

– 条件桩

- C1：功率大于50马力吗？
- C2：维修记录不全吗？
- C3：运行超过10年吗？

– 动作桩

- A1：进行优先处理
- A2：做其他处理

实例1 (3)

■ B. 确定规则个数

- 输入条件个数：3
- 每个条件的取值：“是”或“否”
- 规则个数： $2*2*2 = 8$

- ◆ 功率大于50马力吗
- ◆ 维修记录不全吗
- ◆ 运行超过10年吗

实例1 (4)

■ C.填入条件项

		1	2	3	4	5	6	7	8
条件	功率大于50马力吗?	Y	Y	Y	Y	N	N	N	N
	维修记录不全吗?	Y	Y	N	N	Y	Y	N	N
	运行超过10年吗?	Y	N	Y	N	Y	N	Y	N
动作	进行优先处理								
	作其他处理								

– 利用集合的笛卡尔积计算条件项的取值

实例1 (5)

■ (4) 填入动作项

		1	2	3	4	5	6	7	8
条件	功率大于50马力吗?	Y	Y	Y	Y	N	N	N	N
	维修记录不全吗?	Y	Y	N	N	Y	Y	N	N
	运行超过10年吗?	Y	N	Y	N	Y	N	Y	N
动作	进行优先处理	✓	✓	✓		✓		✓	
	作其他处理				✓		✓		✓

– 1, 2合并, 5, 7合并, 6, 8合并

实例1 (6)

■ (5) 简化

		(1)	(2)	(3)	(4)	(5)
条件	功率大于50马力吗？	Y	Y	Y	N	N
	维修记录不全吗？	Y	N	N	—	—
	运行超过10年吗？	—	Y	N	Y	N
动作	进行优先处理	✓	✓		✓	
	作其他处理			✓		✓

实例2 (1)

■ 下表给出了一个用于三角形问题的判定表的例子

- 在表中 ci 是条件, ai 是判定结果, 由 (ci, ai) 构成的规则可视为测试用例
- 在某些条件取值中出现了 “—”, 表示这个条件在相应规则中被忽略

实例2 (2)

■ 三角形问题的判定表

规则	1	2	3	4	5	6	7	8	9
$c_1: a, b, c$ 构成三角形?	F	T	T	T	T	T	T	T	T
$c_2: a = b?$	—	T	T	T	T	F	F	F	F
$c_3: a = c?$	—	T	T	F	F	T	T	F	F
$c_4: b = c?$	—	T	F	T	F	T	F	T	F
$a_1:$ 非三角形	Y								
$a_2:$ 一般三角形									Y
$a_3:$ 等腰三角形					Y		Y	Y	
$a_4:$ 等边三角形		Y							
$a_5:$ 不可能			Y	Y		Y			

实例2 (3)

■ 细化后的判定表

规则	1	2	3	4	5	6	7	8	9	10	11
$c_1: a < b+c?$	F	T	T	T	T	T	T	T	T	T	T
$c_2: b < a+c?$	—	F	T	T	T	T	T	T	T	T	T
$c_3: c < a+b?$	—	—	F	T	T	T	T	T	T	T	T
$c_4: a = b?$	—	—	—	T	T	T	T	F	F	F	F
$c_5: a = c?$	—	—	—	T	T	F	F	T	T	F	F
$c_6: b = c?$	—	—	—	T	F	T	F	T	F	T	F
$a_1: \text{非三角形}$	Y	Y	Y								
$a_2: \text{一般三角形}$											Y
$a_3: \text{等腰三角形}$							Y		Y	Y	
$a_4: \text{等边三角形}$				Y							
$a_5: \text{不可能}$					Y	Y		Y			

实例2 (4)

■ 三角形问题的测试用例

测试用例号	a	b	c	预期输出	备注
1	4	1	2	非三角形	$c_1: b+c < a$
2	1	4	2	非三角形	$c_2: a+c < b$
3	1	2	4	非三角形	$c_3: a+b < c$
4	2	2	2	等边三角形	$c_4: a=b=c$
5	2	2	3	等腰三角形	$c_7: a=b \neq c$
6	2	3	2	等腰三角形	$c_9: a=c \neq b$
7	3	2	2	等腰三角形	$c_{10}: b=c \neq a$
8	3	4	5	一般三角形	$c_{11}: a \neq b \neq c$

实例2 (5)

■ 注意，有 3 个结果为“不可能”的规则不能做测试用例。如果考虑边界条件，即“两边之和等于第三边”，还可补充 3 个测试用例

测试用例号	a	b	c	预期输出	规则
9	2	1	1	非三角形	$c_{12}: a = b + c$
10	1	2	1	非三角形	$c_{13}: b = a + c$
11	1	1	2	非三角形	$c_{14}: c = a + b$

3.1.6.5 因果图法（1）

- 因果图是一种挑选高效测试用例以检查组合输入条件的系统方法，它是将自然语言规格说明转化成形式语言规格说明的一种严格的方法，它揭露规格说明中的不完整性和二义性

3.1.6.5 因果图法（2）

- 等价类划分方法和边界值分析法都是着重考虑输入条件，并没有考虑到输入情况的各种组合，也没考虑到各个输入情况之间的相互制约关系

3.1.6.5 因果图法（3）

- 因果图方法的思路是：从用自然语言书写的程序规格说明的描述中找出因（输入条件）和果（输出或程序状态的改变），通过因果图转换为判定表

3.1.6.5 因果图法（4）

■ 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

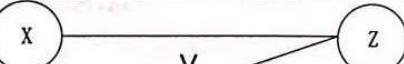
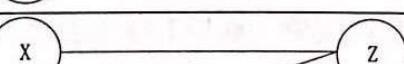
3.1.6.5 因果图法（5）

■ 用因果图生成测试用例的基本步骤（续）

- (3) 由于语法或环境限制，有些原因与原因之间，结果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标明约束或限制条件
- (4) 把因果图转换成判定表
- (5) 把判定表的每一列拿出来作为依据，设计测试用例

3.1.6.5 因果图法 (6)

■ 因果图的关系符号

符号名	图例	释义
恒等		<ul style="list-style-type: none">• 若原因出现，则结果出现• 若原因不出现，则结果也不出现
非		<ul style="list-style-type: none">• 若原因出现，则结果不出现• 若原因不出现，则结果出现
与		<ul style="list-style-type: none">• 若几个原因都出现，结果才出现• 若其中有1个或更多原因不出现，则结果不出现
或		<ul style="list-style-type: none">• 若几个原因中有1个或更多出现，则结果出现• 若几个原因都不出现，则结果不出现
与非		<ul style="list-style-type: none">• 若几个原因中有1个或更多不出现，则结果出现• 若所有原因均出现，则结果不出现
或非		<ul style="list-style-type: none">• 若所有原因全部不出现，则结果出现• 若其中有1个或更多原因出现，则结果不出现

Thank You