SOFTWARE TESTING

Chapter 5

Black Box and Gray Box Testing

School of Data & Computer Science
Sun Yat-sen University

Approaches & Technologies

OUTLINE

判定表法 Decision Table-Based Testing

- What's Decision Table
  - Decision tables (判定表/决策表) have been used to represent and analyze complex logical relationships since the early 1960s. They are ideal for describing situations in which a number of combinations of actions are taken under varying sets of conditions.
  - A decision table has four portions as illustrated: the part to the left of the bold vertical line is the stub (桩) portion; to the right is the entry (项) portion. The part above the bold horizontal line is the condition portion, and below is the action portion. Thus, we can refer to the condition stub, the condition entries, the action stub, and the action entries.
  - A column in the entry portion is a rule. Rules indicate which actions, if any, are taken for the circumstances indicated in the condition portion of the rule.

判定表法 Decision Table-Based Testing

- Example of Decision Table

	Stub Portion	Entry Portion					
	Stub	*Rule 1*	*Rule 2*	*Rules 3, 4*	*Rule 5*	*Rule 6*	*Rules 7, 8*
Condition Stub	c1	T	T	T	F	F	F
	c2	T	T	F	T	T	F
	c3	T	F	–	T	F	–
Action Stub	a1	X	X		X		
	a2	X				X	
	a3		X		X		
	a4			X			X

Condition Entries

Action Entries

判定表法 Decision Table-Based Testing

- Example of Decision Table
  - When conditions c1, c2, and c3 are all true, actions a1 and a2 occur.
  - When c1 and c2 are both true and c3 is false, then actions a1 and a3 occur.
  - When c1 is true and c2 is false, the entry for c3 in the rule is called a “don’t care” entry. This means the condition is irrelevant, or the condition does not apply. Sometimes people will enter the “n/a” symbol for this latter interpretation.

Stub	*Rule 1*	*Rule 2*	*Rules 3, 4*	*Rule 5*	*Rule 6*	*Rules 7, 8*
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	–	T	F	–
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

判定表法 Decision Table-Based Testing

- Test Cases identified with a Decision Table
  - To identify test cases with decision tables, we interpret conditions in a decision table as inputs and actions as outputs. Sometimes conditions end up referring to equivalence classes of inputs, and actions refer to major functional processing portions of the item tested.
  - The rules are then interpreted as test cases. Because the decision table can mechanically be forced to be complete, we have some assurance that we will have a comprehensive set of test cases.
  - Decision tables in which all the conditions are binary are called Limited Entry Decision Tables (LEDTs). For a limited entry decision table with *n* conditions, there must be 2^n independent rules.

判定表法 Decision Table-Based Testing

Decision Table for Triangle Problem

Stub	*R1*	*R2*	*R3*	*R4*	*R5*	*R6*	*R7*	*R8*	*R9*	*R10*	*R11*
c1: a < b + c ?	F	T	T	T	T	T	T	T	T	T	T
c2: b < a + c ?	–	F	T	T	T	T	T	T	T	T	T
c3: c < a + b ?	–	–	F	T	T	T	T	T	T	T	T
c4: a = b ?	–	–	–	T	T	T	T	F	F	F	F
c5: a = c ?	–	–	–	T	T	F	F	T	T	F	F
c6: b = c ?	–	–	–	T	F	T	F	T	F	T	F
a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			
Rule Count=64:	**32**	**16**	**8**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

判定表法 Decision Table-Based Testing

- Test Cases from Decision Table for the Triangle Problem

Case ID	*a*	*b*	*c*	*Expected Output*
ID1	4	1	2	Not a triangle
ID2	1	4	2	Not a triangle
ID3	1	2	4	Not a triangle
ID4	5	5	5	Equilateral
ID5	?	?	?	Impossible
ID6	?	?	?	Impossible
ID7	2	2	3	Isosceles
ID8	?	?	?	Impossible
ID9	2	3	2	Isosceles
ID10	3	2	2	Isosceles
ID11	3	4	5	Scalene

判定表法 Decision Table-Based Testing

- NextDate Problem
  - The complexity in the triangle program is due to the relationships between inputs and correct outputs. The NextDate function is a good example to illustrate a different kind of complexity—logical relationships among the input variables.
  - NextDate is a function of three variables: month, date, and year. It returns the date of the day after the input date. The month, date, and year variables have integer values subject to these conditions:
    - C1: $1 \leq month \leq 12$
    - C2: $1 \leq day \leq 31$
    - C3: $1812 \leq year \leq 2012$
  - By the *Gregorian* calendar, a year is a leap year (which has 29 days in February) if it is divisible by 4, unless it is a century year. Century years are leap years only if they are multiples of 400; thus, 1992, 1996, and 2000 are leap years, while the year 1900 is not a leap year.

判定表法 Decision Table-Based Testing

- NextDate Problem
  - A set of equivalence classes is as follows
    - M1 = {month: month has 30 days}
    - M2 = {month: month has 31 days except December}
    - M3 = {month: month is December}
    - M4 = {month: month is February}
    - D1 = {day: $1 \le day \le 27$}
    - D2 = {day: day = 28}
    - D3 = {day: day = 29}
    - D4 = {day: day = 30}
    - D5 = {day: day = 31}
    - Y1 = {year: year is a leap year}
    - Y2 = {year: year is a common year}
  - The *Cartesian* product of these contains $4 \times 5 \times 2 = 40$ elements. The result of combining rules with "don't care" entries is given in the table next slide; it has 22 rules.

判定表法 Decision Table-Based Testing

NextDate Problem

Decision Table for NextDate Function (1)

Stub	*R1*	*R2*	*R3*	*R4*	*R5*	*R6*	*R7*	*R8*	*R9*	*R10*		
Condition												
c1: Month in	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2		
c2: Day in	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5		
c3: Year in	–	–	–	–	–	–	–	–	–	–		
Action												
a1: Impossible					X							
a2: Increment day	X	X	X			X	X	X	X			
a3: Reset day				X						X		
a4: Increment month				X						X		
a5: Reset month												
a6: Increment year												

判定表法 Decision Table-Based Testing

NextDate Problem

Decision Table for NextDate Function (2)

Stub	*R11*	*R12*	*R13*	*R14*	*R15*	*R16*	*R17*	*R18*	*R19*	*R20*	*R21*	*R22*
Condition												
c1: Month in	M3	M3	M3	M3	M3	M4	M4	M4	M4	M4	M4	M4
c2: Day in	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5
c3: Year in	–	–	–	–	–	–	Y1	Y2	Y1	Y2	–	–
Action												
a1: Impossible										X	X	X
a2: Increment day	X	X	X	X		X	X					
a3: Reset day					X			X	X			
a4: Increment month								X	X			
a5: Reset month					X							
a6: Increment year					X							

判定表法 Decision Table-Based Testing

NextDate Problem

Decision Table for NextDate Function (1)

Don't care

Identical actions

Stub	*R1*	*R2*	*R3*	*R4*	*R5*	*R6*	*R7*	*R8*	*R9*	*R10*		
Condition												
c1: Month in	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2		
c2: Day in	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5		
c3: Year in	–	–	–	–	–	–	–	–	–	–		
Action												
a1: Impossible					X							
a2: Increment day	X	X	X			X	X	X	X			
a3: Reset day				X						X		
a4: Increment month				X						X		
a5: Reset month												
a6: Increment year												

判定表法 Decision Table-Based Testing

NextDate Problem

Reduced Decision Table for NextDate Function (1)

Stub	*R1-R3*	*R4*	*R5*	*R6-R9*	*R10*		
Condition							
c1: Month in	M1	M1	M1	M2	M2		
c2: Day in	D1, D2, D3	D4	D5	D1, D2, D3, D4	D5		
c3: Year in	–	–	–	–	–		
Action							
a1: Impossible			X				
a2: Increment day	X			X			
a3: Reset day		X			X		
a4: Increment month		X			X		
a5: Reset month							
a6: Increment year							

判定表法 Decision Table-Based Testing

NextDate Problem

Decision Table for NextDate Function (2)

Stub	*R11*	*R12*	*R13*	*R14*	*R15*	*R16*	*R17*	*R18*	*R19*	*R20*	*R21*	*R22*
Condition												
c1: Month in	M3	M3	M3	M3	M3	M4	M4	M4	M4	M4	M4	M4
c2: Day in	D1	D2	D3	D4	D5	D1	D2	D2	D3	D3	D4	D5
c3: Year in	–	–	–	–	–	–	Y1	Y2	Y1	Y2	–	–
Action												
a1: Impossible										X	X	X
a2: Increment day	X	X	X	X		X	X					
a3: Reset day					X			X	X			
a4: Increment month								X	X			
a5: Reset month					X							
a6: Increment year					X							

判定表法 Decision Table-Based Testing

NextDate Problem

Reduced Decision Table for NextDate Function (2)

Stub	*R11-R14*	*R15*	*R16*	*R17*	*R18*	*R19*	*R20*	*R21-R22*
Condition								
c1: Month in	M3	M3	M4	M4	M4	M4	M4	M4
c2: Day in	D1, D2, D3, D4	D5	D1	D2	D2	D3	D3	D4, D5
c3: Year in	–	–	–	Y1	Y2	Y1	Y2	–
Action								
a1: Impossible							X	X
a2: Increment day	X		X	X				
a3: Reset day		X			X	X		
a4: Increment month					X	X		
a5: Reset month		X						
a6: Increment year		X						

判定表法 Decision Table-Based Testing

NextDate Problem

Decision Table Test Cases for NextDate

Case ID	*1-3*	*4*	*5*	*6-9*	*10*
Month	4	4	4	1	1
Day	15	30	31	15	31
Year	2001	2001	2001	2001	2001
Expect Output	4/16/2001	5/1/2001	Invalid input data	1/16/2001	2/1/2001

Case ID	*11-14*	*15*	*16*	*17*	*18*	*19*	*20*	*21-22*
Month	12	12	2	2	2	2	2	2
Day	15	31	15	28	28	29	29	30
Year	2001	2001	2001	2004	2001	2004	2001	2001
Expect Output	12/16/2001	1/1/2002	2/16/2001	2/29/2004	3/1/2001	3/1/2004	Invalid input data	Invalid input data

- 判定表法 Decision Table-Based Testing
  - 判定表小结
    - 判定表技术适用于具有以下特征的应用程序：
      - If-then-else 逻辑突出
      - 输入变量之间存在逻辑关系
      - 涉及输入变量子集的计算
      - 输入与输出之间存在因果关系
      - 很高的圈复杂度
    - 判定表的精简
      - *n* 个条件的 LEDT 有 2^n 个规则 (每个条件分别取真、假值)。实际使用判定表时，常常需要先将它简化。判定表的简化以合并相似规则为目标，即若判定表中有两条以上的规则具有相同的动作，并且在条件项之间存在极为相似的关系，便可以设法将其合并。

判定表法 Decision Table-Based Testing

判定表小结

构造判定表的步骤：

(1) 确定规则的个数。例如 n 个条件的 LEDT 有 2^n 个规则。

(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) 化简：合并相似规则后得到最终判定表。

判定表法 Decision Table-Based Testing

判定表小结

- 使用判定表设计测试用例，可以把条件解释为输入，把行动解释为输出。适合使用判定表设计测试用例的情况有：
  - 规格说明以判定表形式给出，或容易转换成判定表。
  - 条件的排列顺序不会也不应影响执行的操作。
  - 规则的排列顺序不会也不应影响执行的操作。
  - 每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。
  - 如果某一规则得到满足要执行多个操作，这些操作的执行顺序无关紧要。

因果图法 Cause-and-Effect Graphing

- 概述
  - 因果图方法考虑被测程序输入条件之间的联系和组合关系，描述由多种条件的组合产生多个动作的结构形式，最终生成判定表 (Decision Table)，方便设计测试用例。
  - 因果图方法有助于指出程序规格说明中存在的问题。

因果图法

- 因果图法生成测试用例的基本步骤

(1) 对软件规格说明进行描述

- 给每个原因 (即输入条件或输入条件的等价类) 和结果 (即输出条件) 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、或者原因与原因之间的对应关系；
- 根据这些关系，画出因果图。

(3) 分析有关的语法或环境限制条件

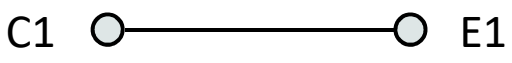
- 由于语法或环境限制，有些原因和结果因素的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。

(4) 把因果图转换成判定表

(5) 判定表的每一列构成测试用例设计的依据

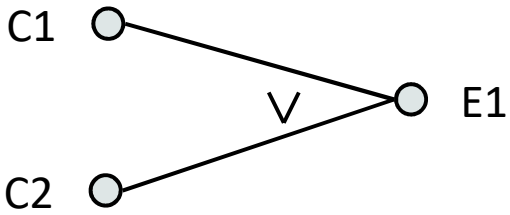
因果图法

- 因果图中使用的基本符号
  - 用 Ci 表示原因 (Cause)，i 为原因的编号
  - 用 Ej 表示结果 (Effect)，j 为结果的编号
  - 用结点表示原因或结果的状态，可取值 “0” 或 “1”
    - “1” 表示该状态出现，“0” 表示该状态不出现。
  - 原因和结果之间的关系图解主要有：

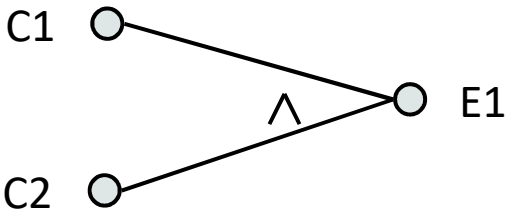


恒等：C1 ↔ E1

非：C1 ↔ ¬ E1



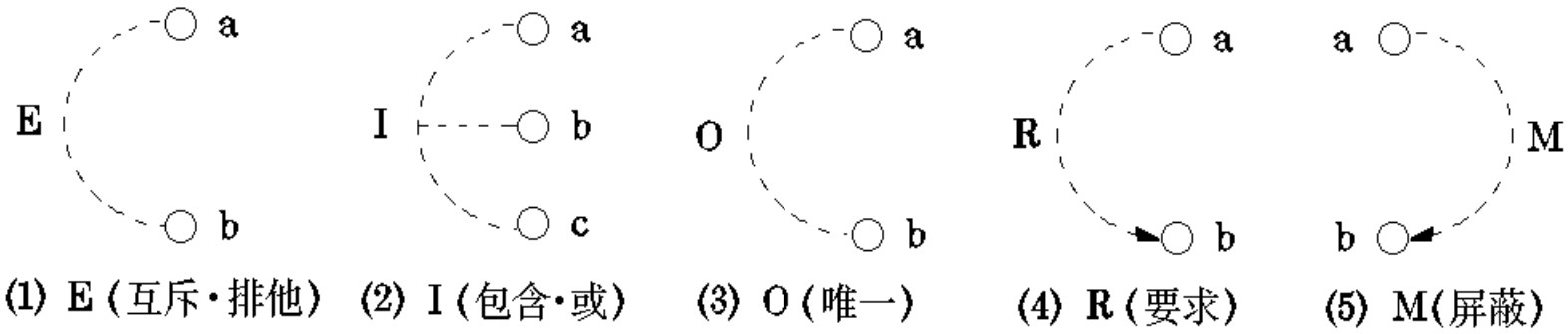
或：C1∨C2 → E1



与：C1∧C2 → E1

因果图法

- 因果图中使用的表示约束条件的符号
  - 为表示原因与原因之间、结果与结果之间可能存在的有关状态的约束，在因果图中可以附加一些表示约束条件的符号：



(1) E（互斥·排他） (2) I（包含·或） (3) O（唯一） (4) R（要求） (5) M(屏蔽)

  - 互斥 E：a, b 不会同时出现，最多有一个可能出现；
  - 包含 I：a, b, c 中至少一个必须出现；
  - 唯一 O：a 和 b 当中必须有一个，且仅有一个出现；
  - 要求 R：当 a 出现时，b 必须也出现；
  - 屏蔽 M：用于对结果的约束。当 a 出现时，b 必不出现；当 a 不出现时，b 值不定。

因果图法

- 例：自动售货机软件的测试用例。
  - 设计处理单价为5角的饮料的自动售货机软件的测试用例。软件规格说明如下：
    (1) 操作者投入5角或1元的硬币，按下 橙汁 或 啤酒 的按钮，售货机送出相应的饮料 (不考虑饮料不足的情况)。
    (2) 若售货机没有零钱找，则一个显示 零钱找完 的红灯亮。
    - 此时操作者投入1元硬币并按下按钮后，不送出饮料，而是退还1元硬币。

    (3) 若售货机有零钱找，则显示 零钱找完 的红灯灭。
    - 此时操作者投入1元硬币并按下按钮后，售货机送出饮料，退还5角硬币。

因果图法

- 例：自动售货机软件的测试用例。

(1) 分析需求说明，列出原因和结果清单

- 原因清单 (输入条件)
  - C1 售货机可找零
  - C2 投入1元硬币
  - C3 投入5角硬币
  - C4 按下 橙汁 按钮
  - C5 按下 啤酒 按钮
- 结果清单 (输出结果)
  - E21 零钱找完 灯亮
  - E22 退还1元硬币
  - E23 退还5角硬币
  - E24 送出橙汁饮料
  - E25 送出啤酒饮料

因果图法

- 例：自动售货机软件的测试用例。

(1) 分析需求说明，列出原因和结果清单 (续)

- 建立中间结点，表示处理的中间状态
  - T11 投入1元硬币且按下饮料按钮
  - T12 按下 橙汁 或 啤酒 按钮
  - T13 应当找5角零钱并且售货机有零钱找
  - T14 钱已付清

因果图法

- 例：自动售货机软件的测试用例。

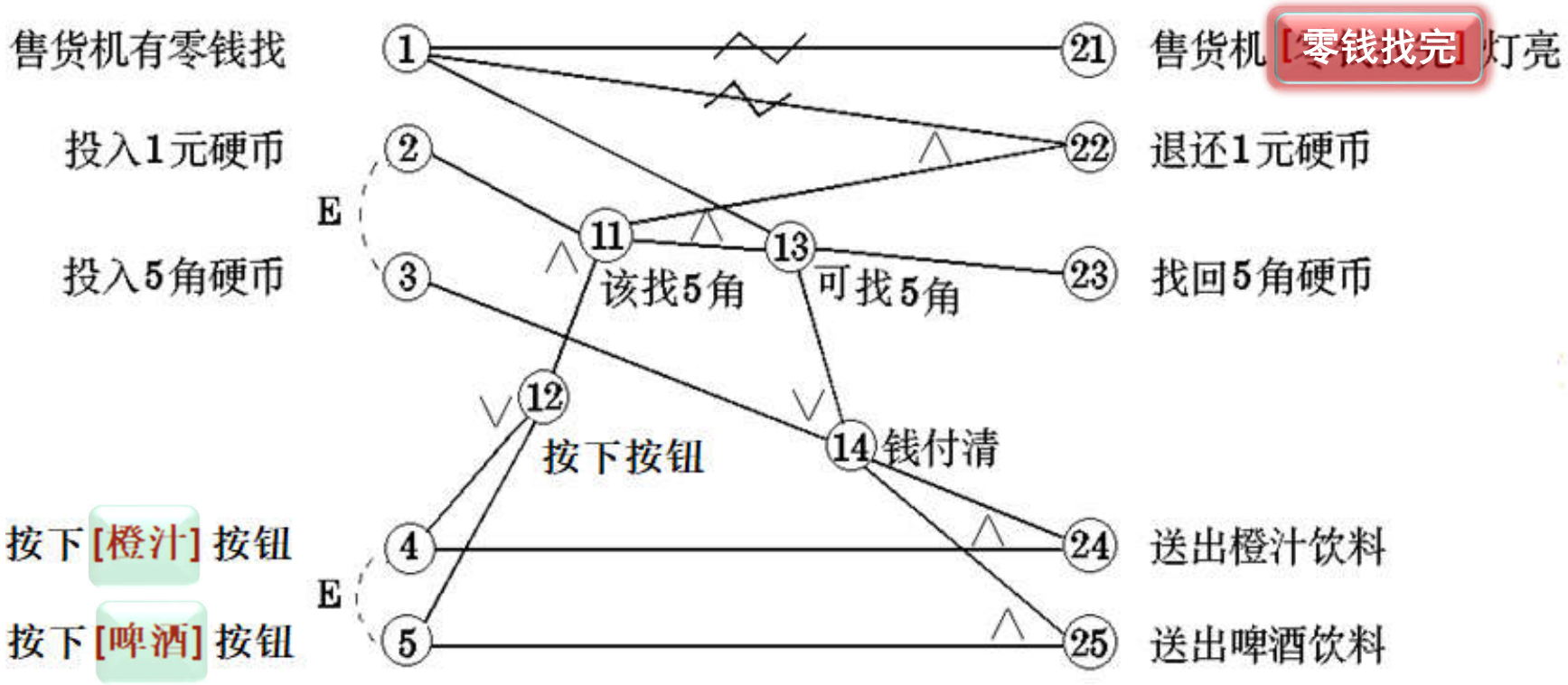
(2) 画出因果图

- 所有原因结点列在左边
- 所有结果结点列在右边
- 所有中间结点列在中间
- 所有因果关系表示为连接图解
- 加上必要的互斥约束条件 E
  - C2 与 C3、C4 与 C5 不能同时发生。

因果图法

- 例：自动售货机软件的测试用例。

(2) 画出因果图 (续)



因果图法

- 例：自动售货机软件的测试用例。

(3) 因果图转换成判定表

- 按照因果图建立规则库，对输入条件 C1-C5 的全部解释计算输出结果，得到 2^5=32 列的判定表。

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

因果图法

- 例：自动售货机软件的测试用例。

(3) 因果图转换成判定表 (续)

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

因果图法

- 例：自动售货机软件的测试用例。

(3) 因果图转换成判定表 (续)

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

- 判定表中可以删去的列：阴影部分表示违反约束条件的不可能出现的情况；第16列与第32列对应的输入条件 C2、C3、C4、C5 为0 (黄色部分)，表示操作者没有动作。

因果图法

- 例：自动售货机软件的测试用例。

(3) 因果图转换成判定表 (续)

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

- 余下的16列用绿色标示，作为确定测试用例的依据。

因果图法

- 例：自动售货机软件的测试用例。

(4) 判定表的分析

- 以判定表的第7列为例：
  - 输入 11001，表示 C1 售货机可找零、C2 投入1元硬币、C5 按下啤酒按钮。输出 00101，表示 E23 退还5角硬币、E25 送出啤酒饮料。
  - 实现上述输入-输出过程的规则描述：

$C1 \wedge T11 \rightarrow T13$

$C2 \wedge T12 \rightarrow T11$

$C4 \vee C5 \rightarrow T12$

$C3 \vee T13 \rightarrow T14$

$C5 \wedge T14 \rightarrow E25$

$T13 \rightarrow E23$

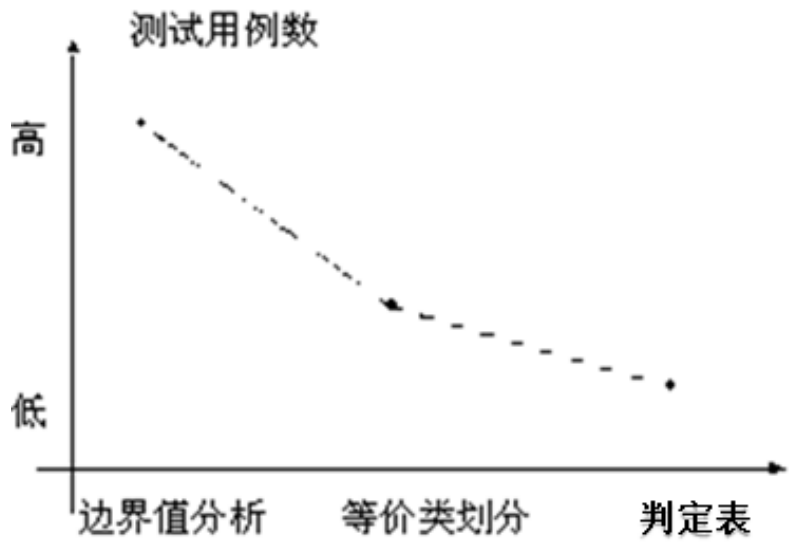
  - 以 C1, C2, C5 为前提，应用上述规则，可以证明逻辑结论 E23 和 E25。

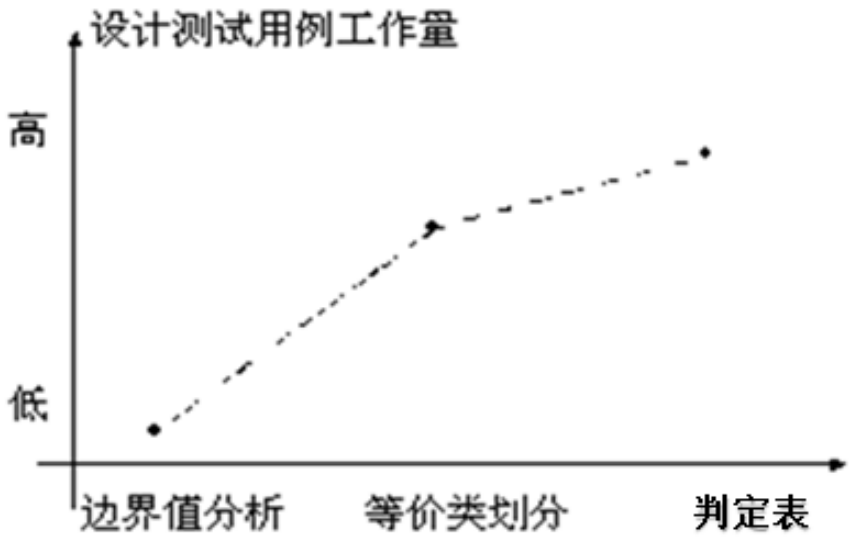
序号		1	2	3	4	5	6	7	8
条件	①	1	1	1	1	1	1	1	1
	②	1	1	1	1	1	1	1	1
	③	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0
	⑫						1	1	0
	⑬						1	1	0
	⑭						1	1	0
结果	㉑						0	0	0
	㉒						0	0	0
	㉓						1	1	0
	㉔						1	0	0
	㉕						0	1	0

- 因果图法
  - 因果图法小结
    - 因果图方法是一种非常有效的黑盒测试方法。
      - 能够生成没有重复性且发现错误能力强的测试用例；
      - 对输入、输出同时进行了分析。
    - 从因果图生成的测试用例包括了被测程序所有输入数据的“取真”与“取假”的情况。
      - 构成相对较少的测试用例数目；
      - 测试用例数目随被测程序输入数据数目的增加而非线性地增加。
    - 如果项目在设计阶段就采用了判定表技术，则可以直接利用已有的判定表设计测试用例。

黑盒测试方法的比较

- 测试工作量
  - 以边界值分析、等价类划分和判定表测试方法来讨论它们的测试工作量，即生成测试用例的数量与设计开发这些测试用例所需的工作量。





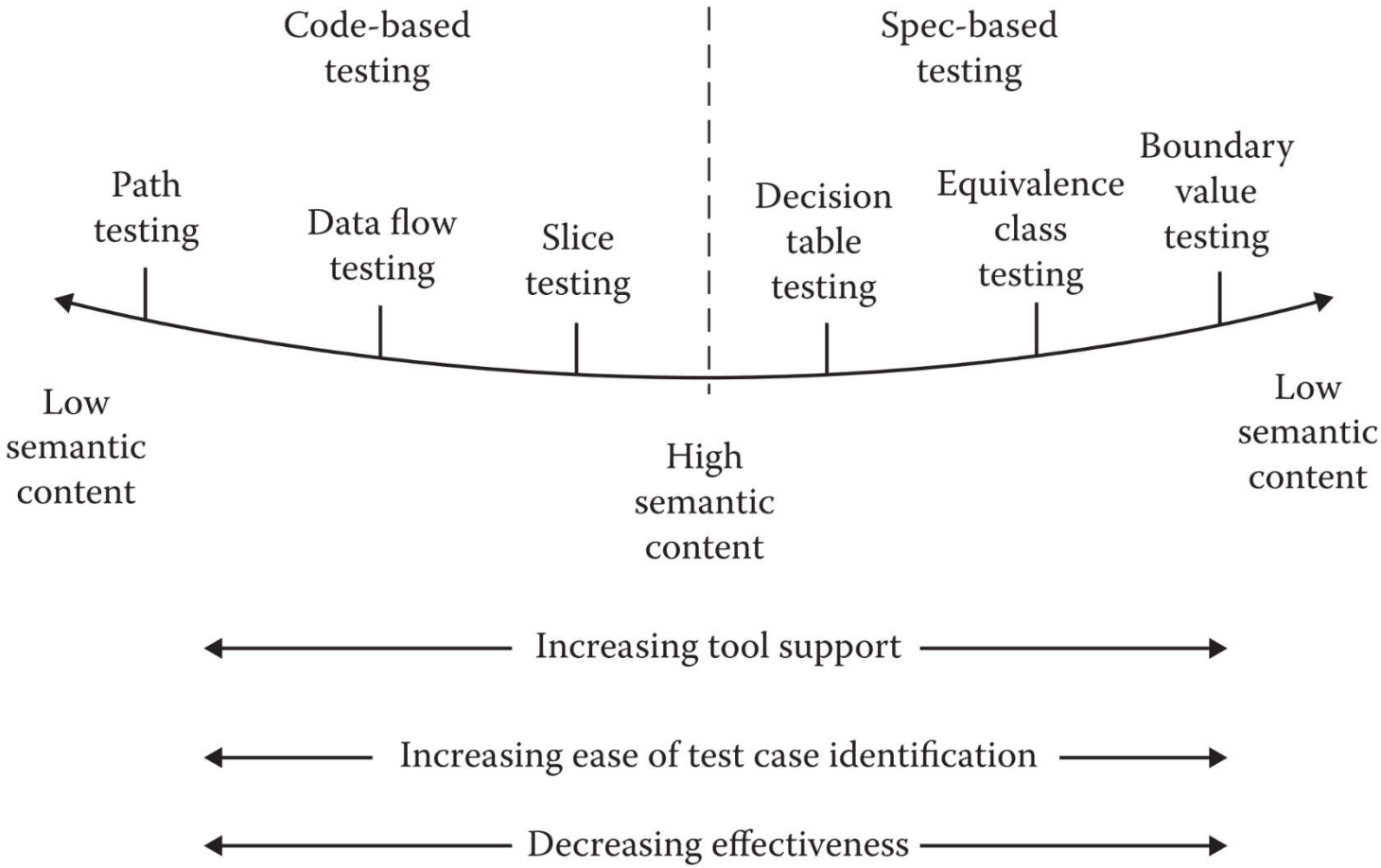
黑盒测试方法的比较

- 测试有效性
  - 解释测试有效性是很困难的。因为我们不知道程序中的所有故障，因此也不可能知道给定方法所产生的测试用例是否能够发现这些故障。测试人员能够做的，只是根据不同类型的故障，选择最有可能发现这种缺陷的测试方法 (包括白盒测试)。根据最可能出现的故障种类，分析得到可提高测试有效性的实用方法。通过跟踪所开发软件中的故障的种类和密度，也可以改进这些方法。

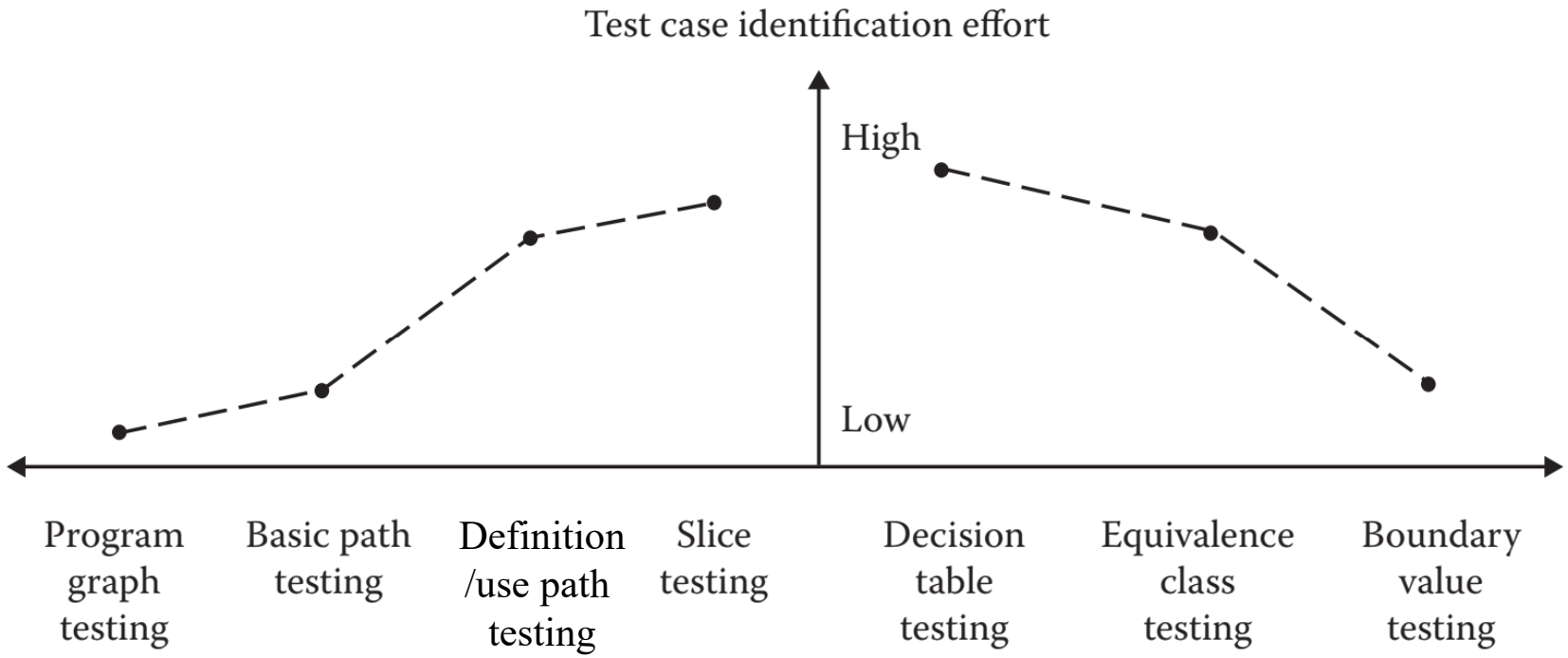
黑盒测试方法的选择

- 利用程序的一些已知属性，选择处理这种属性的测试方法。在选择黑盒测试方法时一些经常用到的属性有：
  - 变量表示物理量还是逻辑量?
  - 在变量之间是否存在依赖关系?
  - 是否有大量的例外处理?
- 黑盒测试方法选取的建议原则：
  - 变量引用的是物理量：采用边界值分析测试和等价类测试。
  - 变量是独立的：采用边界值分析测试和等价类测试。
  - 变量不是独立的：采用判定表测试。
  - 可保证是单缺陷假设：采用边界值分析和健壮性测试。
  - 可保证是多缺陷假设：采用边界值分析测试和判定表测试。
  - 程序包含大量例外处理：采用健壮性测试和判定表测试。
  - 变量引用的是逻辑量：采用等价类测试用例和判定表测试。

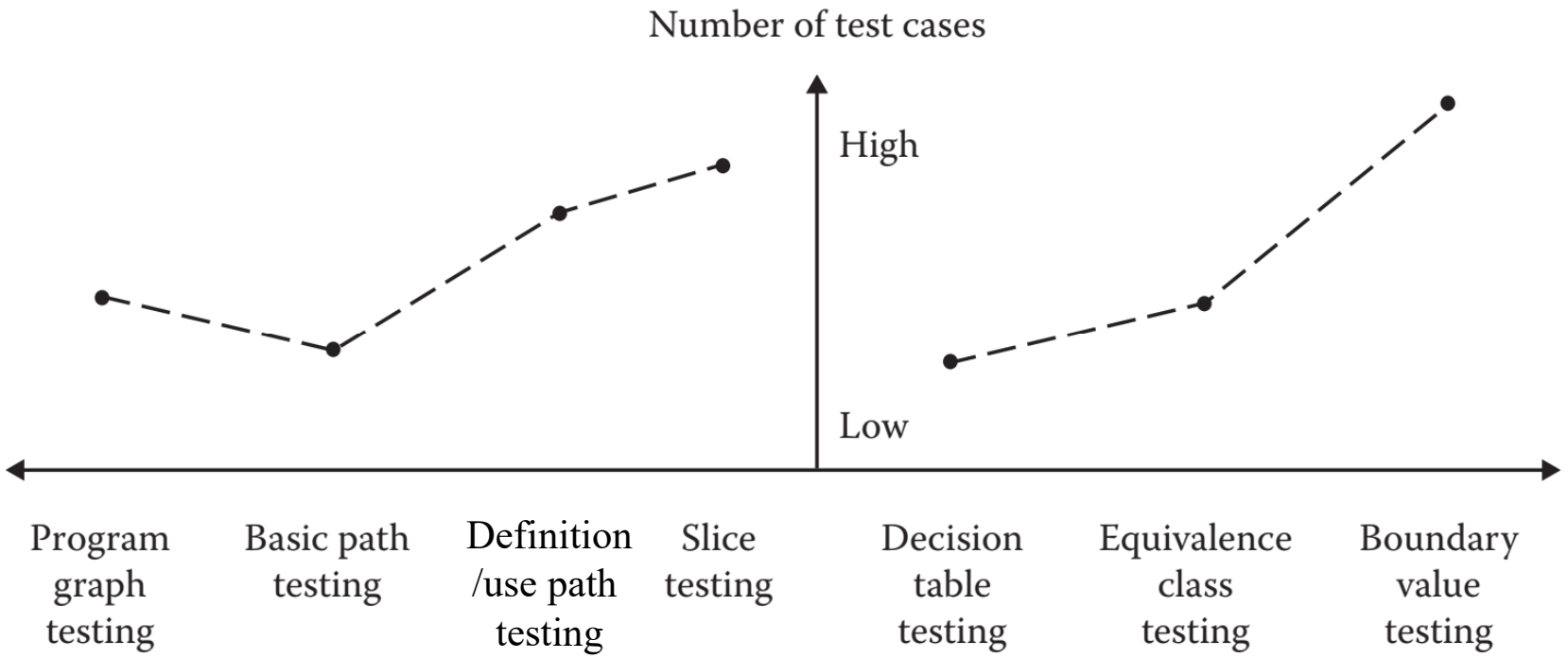
The Test Method Pendulum



The Test Method Pendulum



The Test Method Pendulum



灰盒测试概述

- 灰盒测试方法提出的基础
  - 白盒测试过程中测试者拥有被测程序源代码，可以根据程序的内部结构设计测试用例。
    - 理想的白盒测试应该使选取的测试用例覆盖被测程序的所有执行路径，但通常有难度。
    - 白盒测试不关注被测程序的外部功能。
  - 黑盒测试是在完全不考虑被测程序内部结构和内部特征的情况下，测试者根据被测程序的需求规格说明书设计测试用例，推断测试结果的正确性。
    - 黑盒测试用例的选择只考虑被测程序的输入以及相应的的输出，并不考虑程序的内部结构。
    - 被测程序的内部结构是否规范、结构化程度的好坏、系统的性能如何等都得不到测试。

灰盒测试概述

- 灰盒测试方法提出的基础 (续)
  - 因此，要进行较全面的测试，可以考虑对被测程序同时采用白盒测试方法和黑盒测试方法。
  - 灰盒测试是一种将黑盒测试、白盒测试、回归测试和变异测试结合在一起的综合测试方法，也是一种软件全生命周期测试方法。

灰盒测试概述

- 灰盒测试的基本思路
  - 基于被测程序运行时的外部表现，同时结合被测程序内部逻辑结构来设计测试用例。
  - 验证被测程序满足外部指标，而且被测程序的所有通道或路径都进行了检验。
  - 以被测程序的主要功能和主要性能为测试依据。
    - 根据程序流程图、需求规格说明书以及测试者的实践经验来设计测试用例。
  - 现代测试工程中，最常见的灰盒测试是集成测试。
    - 重点关注被测软件系统的各个模块之间的相互关联，即模块之间的互相调用、数据传递、同步/互斥等等。

灰盒测试概述

灰盒测试的特点

- 灰盒测试根据被测程序的需求规格说明文档进行测试用例的设计，这点类似于黑盒测试；但它需要深入到被测程序内部的特殊点来进行功能测试和结构测试。
  - 例：单元接口测试。将接口参数传递到被测单元中，检验软件在测试执行环境控制下的执行情况。
- 灰盒测试通常在程序员做完白盒测试之后，功能测试人员进行大规模集成测试之前进行，并由专门测试人员实施。
- 灰盒测试通过类似白盒测试的方法进行。
  - 需要了解代码工程的实现；
  - 通过编写代码，调用函数或者封装好的接口进行测试，但无需关心被测程序模块内部的实现细节，依然可把它当成一个黑盒。

灰盒测试的优缺点

- 灰盒测试的优点
  - 能够进行基于需求的覆盖测试和基于程序路径覆盖的测试；
  - 测试结果对应于程序内部路径，便于定位、分析和解决缺陷；
  - 能够保证所设计的黑盒测试用例的完整性，防止遗漏软件的一些不常用的功能或功能组合；
  - 能够减少需求或设计不详细或不完整对测试造成的影响。
- 灰盒测试的不足
  - 投入的时间比单纯的黑盒测试多大约 20-40%；
  - 对测试人员的要求较高；
    - 灰盒测试要求测试人员清楚系统内部的模块构成，以及模块之间的协作关系。
  - 不如白盒测试深入；
  - 不适用于简单的系统。

Thank you!