Fits Black Box Testing Example

Software Testing and Quality Assurance

1

Program Description

- A plane has 120 seats. A ight can accommodate up to that number of passengers. If the passengers require extra comfort, the effective number of seats is reduced by 40, to ensure that every passenger has an empty seat next to them. The method fits() indicates whether a particular number of passengers can be accommodated with or without extra comfort.

2

Specification

- Status fits(int passengers, boolean comfortFlag)
- Inputs
  - **passengers**: the number of passengers to be carried
  - **comfortFlag**: ag to indicate whether extra comfort is required
- Outputs
  - **return value**:
    - SUCCESS if passengers120 and !comfortFlag
    - SUCCESS if passengers80 and comfortFlag
    - FAILURE if passengers>120, or if passengers>80 and comfortFlag
    - ERROR if any inputs are invalid (e.g. passengers<1)
- Status is defined as follows:
  - enum Status { SUCCESS, FAILURE, ERROR };

3

Equivalence Partitioning - Passengers

- There are four types of processing defined:
  - 1. Enough seats are always available (1..80)
  - 2. Enough seats are conditionally available (81..120)
  - 3. Enough seats are never available (121..Integer.MAX VALUE)
  - 4. An error in the input value (0)
- Note that combinations of input parameters are not considered in Equivalence Partitioning.

4

Processing for comfortFlag

- There are two types of processing defined:
  - 1. Don't leave extra space for comfort (false)
  - 2. Leave extra space for comfort (true)

5

Processing for returnValue

- There are three types of processing defined:
  - 1. Enough seats are available (SUCCESS)
  - 2. Not enough seats are available (FAILURE)
  - 3. An error in the input parameters (ERROR)

6

Natural Ranges

Parameter	Natural Range
passengers	Integer.MIN_VALUE..Integer.MAX_VALUE
comfortFlag	false true
Return Value	SUCCESS FAILURE ERROR

Note that passengers has one natural range, with 2^{32} values; comfortFlag has two natural ranges, each with one value; the return value has three natural ranges, each with one value. In testing, it is useful to treat enumerated and boolean types in this manner.

7

Input Partitions for fits()

Parameter	Range
passengers	Integer.MIN_VALUE..0 1..80 81..120 121..Integer.MAX_VALUE
comfortFlag	false true

8

Output Partitions for fits()

Parameter	Range
Return Value	SUCCESS FAILURE ERROR

9

Test Cases

Case	Parameter	Range	Test
EP1* EP2 EP3 EP4	passengers	Integer.MIN_VALUE..0 1..80 81..120 121..Integer.MAX_VALUE	
EP5 EP6	comfortFlag	false true	
EP7 EP8 EP9*	Return Value	SUCCESS FAILURE ERROR	

10

Notes

- 1. The test cases are specific to a particular technique{it is useful to prefix the identifier with the technique name (EP is used as an abbreviation for Equivalence Partitioning in this example).
- 2. An asterisk (*) indicates an error case, which must be tested separately.
- 3. The Test column is completed after the test data is completed. It confirms that there is at least one Test for each test case.

11

Test Data

- Test input data is selected from arbitrary values within the equivalence partitions:
- Normally a central value is selected. Start with the first normal test case (i.e. not an error case). Then complete the tests for all the other normal test cases - for each additional test, data is selected to cover as many additional normal test cases as possible.
- Finally, complete the error cases: each input error case must have its own unique test {there can only be one input error case covered by any one test.

12

EP data for fits()

ID	Test Cases Covered	Inputs		Exp. Output
		passengers	comfortFlag	return value
T1.1	EP2,5,7	40	false	SUCCESS
T1.2	EP3,6,8	101	true	FAILURE
T1.3	EP4 [5,8]	200	false	FAILURE
T1.4*	EP1*,9*	-100	false	ERROR

13

Note

- Each input error case is tested separately- otherwise multiple simultaneous errors may hide correct error processing for one of the errors
- Note that as an abbreviation *EP*2,5,7" is used to mean *EP*2 and *EP*5 and *EP*7".
- Error cases are marked with an asterisk "*".

14

Boundary Value Analysis

- Boundary Values are the upper and lower values for each Equivalence Partition.
- Having identified the partitions, identifying the boundary values is straightforward.

15

Input and output Boundary values for fits()

Case	Parameter	Boundary Value	Test
BV1*	passengers	Integer.MIN_VALUE	T2.7
BV2*		0	T2.8
BV3		1	T2.1
BV4		80	T2.2
BV5		81	T2.3
BV6		120	T2.4
BV7		121	T2.5
BV8		Integer.MAX_VALUE	T2.6
BV9	comfortFlag	false	T2.1
BV10		true	T2.2

Case	Parameter	Boundary Value	Test
BV11	Return Value	SUCCESS	T2.1
BV12		FAILURE	T2.4
BV13*		ERROR	T2.7

16

Test Data

ID	Test Cases Covered	Inputs		Exp. Output
		passengers	comfortFlag	return value
T2.1	BV3,9,11	1	false	SUCCESS
T2.2	BV4,10 [11]	80	true	SUCCESS
T2.3	BV5 [9,11]	81	false	SUCCESS
T2.4	BV6 [10] 12	120	true	FAILURE
T2.5	BV7 [10,12]	121	true	FAILURE
T2.6	BV8 [10,12]	Integer.MAX_VALUE	true	FAILURE
T2.7	BV1*,13*	Integer.MIN_VALUE	false	ERROR
T2.8	BV2* [13*]	0	false	ERROR

17

Comment

- At the expense of approximately twice the number of tests, the minimum and maximum value of each Equivalence Partition has been tested at least once, using a minimum number of tests. However, combinations of different boundary values have not been exhaustively tested

18

Combinational Testing/Truth Tables

- The (non-error) **causes** for this program, taken from the specification, can be expressed as follows:
  - passengers 80
  - passengers 120
  - comfortFlag
- It takes practice to identify reasonable causes: the partitions provide a good starting point. There is no one right answer { typically the causes can be stated in a large number of different (but equivalent) ways.
- Note: comfortFlag is boolean, so it is redundant to state “comfortFlag==true”.

19

Combinational Testing/Truth Tables

- When considering the non-error combinations for testing, only the non-error effects need be considered:
  - return value == SUCCESS
  - return value == FAILURE

20

Creating the Truth Table

- The rules must be (a) complete, and (b) independent. One rule, and exactly one rule, must be selected by any combination of the input causes. Do not include impossible combinations of causes.
- Develop the rules systematically, starting with all F at the left-hand side of the table, and ending with all T at the right-hand side (or, alternatively, starting with all T, and ending with all F).
- Use don't care conditions (indicated by a '*') when the value of a cause has no impact on the effect of the rule.

21

Candidate Truth Table

		Candidate Rules							
		a	b	c	d	e	f	g	h
Causes									
	passengers ≤ 80	F	F	F	F	T	T	T	T
	passengers ≤ 120	F	F	T	T	F	F	T	T
	comfortFlag	F	T	F	T	F	T	F	T
Effects									
	return value == SUCCESS	F	F	T	F	-	-	T	T
	return value == FAILURE	T	T	F	T	-	-	F	F

22

Reducing the Truth Table

1. Candidate rules a and b have the same effect, and only one different cause value (for comfortFlag), so can be merged using a don't care for comfortFlag

2. Rules c and d must be retained, they have different effects

3. Rules e and f are impossible, and can be removed

4. Rules g and h have the same effect, and only one different cause value (for comfortFlag), so can be merged using a don't care for comfortFlag

23

Final Truth Table

		Rules			
		1	2	3	4
Causes					
	passengers ≤ 80	F	F	F	T
	passengers ≤ 120	F	T	T	T
	comfortFlag	*	F	T	*
Effects					
	return value == SUCCESS	F	T	F	T
	return value == FAILURE	T	F	T	F

24

Test Cases

Case	Rule	Test
TT1	1	T3.1
TT2	2	T3.2
TT3	3	T3.3
TT4	4	T3.4

Each Rule is a Test Case

25

Test Data

ID	Test Cases Covered	Inputs		Exp. Output
		passengers	comfortFlag	return value
T3.1	TT1	200	false	FAILURE
T3.2	TT2	100	false	SUCCESS
T3.3	TT3	100	true	FAILURE
T3.4	TT4	40	false	SUCCESS

26

Random Testing – Test Data

- The random values shown here have been generated (using a Java program and the Random class) as follows:

1. Select the result at random (SUCCESS, FAILURE, or ERROR)

2. Select the comfortFlag at random

3. Select a value for passengers from the range of values that will cause the required result:

ERROR: from Integer.MIN VALUE to 0
SUCCESS and false: from 1 to 120
SUCCESS and true: from 1 to 80
FAILURE and false: from 121 to Integer.MAX VALUE
FAILURE and true: from 81 to Integer.MAX VALUE

27

Random Test Data

ID	Inputs		Exp. Output
	passengers	comfortFlag	return value
T4.1	16	false	SUCCESS
T4.2	46	true	SUCCESS
T4.3	-1974596984	true	ERROR
T4.4	-122368221	false	ERROR
T4.5	10	true	SUCCESS
T4.6	40	false	SUCCESS
T4.7	112	false	SUCCESS
T4.8	1950430522	true	FAILURE
T4.9	74	false	SUCCESS
T4.10	-1942749054	true	ERROR

28

Duplicate Test elimination

- The complete set of test data required for black-box testing is shown in the next table
- Duplicate tests are highlighted with a grey background and can be eliminated
- The use of 'standard' values helps when identifying identical tests

29

Final Test set with duplicates identified

ID	Test Cases Covered	Inputs		Exp. Output	
		passengers	comfortFlag	return value	
T1.1	EP2,5,7	40	false	SUCCESS	
T1.2	EP3,6,8	101	true	FAILURE	
T1.3	EP4 [5,8]	200	false	FAILURE	
T1.4	EP1*,9*	-100	false	ERROR	
T2.1	BV3,9,11	1	false	SUCCESS	
T2.2	BV4,10 [11]	80	true	SUCCESS	
T2.3	BV5 [9,11]	81	false	SUCCESS	
T2.4	BV6 [0] 12	120	true	FAILURE	
T2.5	BV7 [10,12]	121	true	FAILURE	
T2.6	BV8 [10,12]	Integer.MAX_VALUE	true	FAILURE	
T2.7	BV1* 13*	Integer.MIN_VALUE	false	ERROR	
T2.8	BV2* [13*]	0	false	ERROR	
T3.1	TT1	200	false	FAILURE	T1.3
T3.2	TT2	100	false	SUCCESS	
T3.3	TT3	100	true	FAILURE	
T3.4	TT4	40	false	SUCCESS	T1.1
T4.1		16	false	SUCCESS	
T4.2		46	true	SUCCESS	
T4.3		-1974596984	true	ERROR	
T4.4		-122368221	false	ERROR	
T4.5		10	true	SUCCESS	
T4.6		40	false	SUCCESS	
T4.7		112	false	SUCCESS	
T4.8		1950430522	true	FAILURE	
T4.9		74	false	SUCCESS	
T4.10		-1942749054	true	ERROR	

30