# CS608
# Software Testing

Dr. Stephen Brown

Room Eolas 116

stephen.brown@mu.ie

# CS608

## Testing Combinations with Decision Tables

(Essentials of Software Testing, Chapter 4)

# Introduction

- The techniques presented previously have not considered different combinations of input values which can lead to undetected faults in the code
- Next: the black-box technique of decision table (DT) testing

# Testing Combinations with Decision Tables

- There are a number of techniques for identifying combinations of inputs for testing software
- Decision tables provide a systematic approach for identifying all the possible combinations of input values based on equivalence partitions

**Definition**:
a decision table is a model of the functional requirements that maps combinations of input values (causes) to their matching output values (effects) through rules.

# Example

- Continue testing OnlineSales.giveDiscount()
- Summary – the method returns:

FULLPRICE if bonusPoints≤120 and not a goldCustomer

FULLPRICE if bonusPoints≤80 and a goldCustomer

DISCOUNT if bonusPoints>120

DISCOUNT if bonusPoints>80 and a goldCustomer

ERROR if any inputs are invalid (bonusPoints<1)

# Approach

- Identify causes and effects
- Identify all combinations (of causes)
- Identify possible combinations (of causes)
- Complete the decision table
- Each rule in the decision table is a Test Coverage Item
- Each rule needs its own Test Case (by definition)

# Step 1. Analysis

1. Restate the specification in terms of **causes** and **effects** (boolean expressions for each partition)
2. Identify the feasible combinations of causes
3. Generate a decision table – this relates the causes (inputs) to the effects (outputs) through rules

- Provides a systematic way to find all the combinations
- This example only considers normal inputs and not error inputs
- If different combinations of error inputs result in different outputs, then we use a separate table for these (next week)

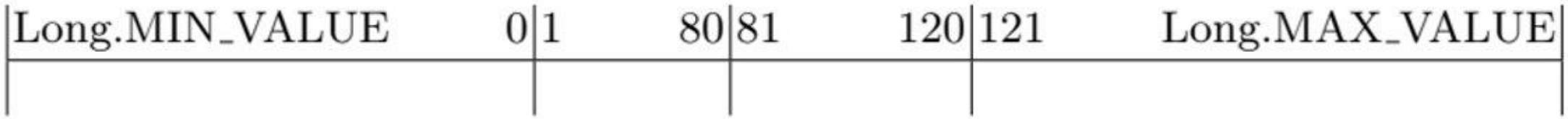
# Input Equivalence Partitions for giveDiscount()

| Parameter | Equivalence Partition |
|---|---|
| bonusPoints | (*) Long.MIN_VALUE..0<br>1..80<br>81..120<br>121..Long.MAX_VALUE |
| goldCustomer | true<br>false |

- Develop boolean expressions that define the **non-error causes**
- Consider the input parameters in order, examining the partitions from left to right (i.e. increasing values)
- Systematic approach – limits mistakes

# Non-Error Causes for bonusPoints

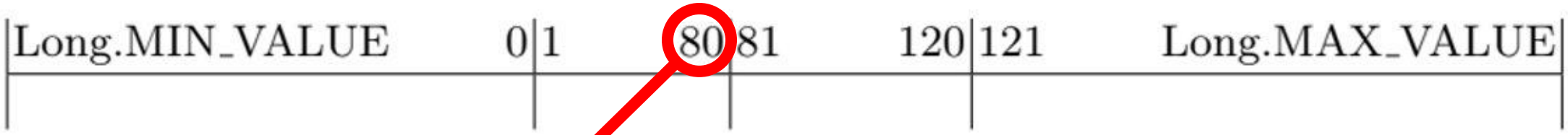
| Parameter | Equivalence Partition |
|---|---|
| bonusPoints | (*) Long.MIN_VALUE..0 |
| | 1..80 |
| | 81..120 |
| | 121..Long.MAX_VALUE |




- [Long.MIN_VALUE..0] is an error partition – not used

# Non-Error Causes for bonusPoints

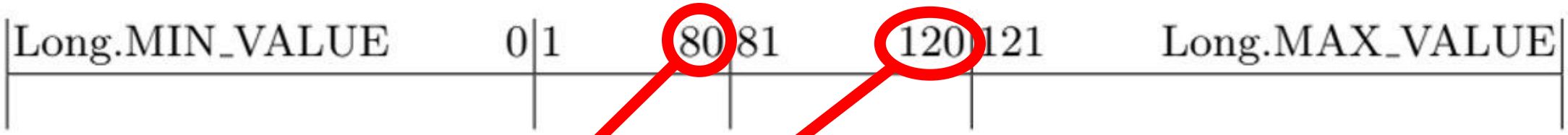
| Parameter | Equivalence Partition |
|---|---|
| bonusPoints | (*) Long.MIN_VALUE..0 |
| | 1..80 |
| | 81..120 |
| | 121..Long.MAX_VALUE |




- [1..80] is a normal partition, identified by:
  - **bonusPoints<=80** being true (as we disallow bonusPoints<1)

# Non-Error Causes for bonusPoints

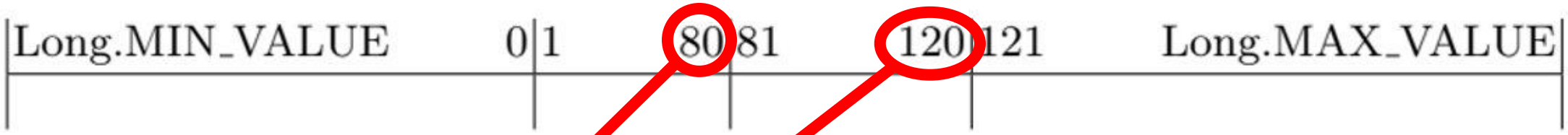
| Parameter | Equivalence Partition |
|---|---|
| bonusPoints | (*) Long.MIN_VALUE..0 |
| | 1..80 |
| | 81..120 |
| | 121..Long.MAX_VALUE |




- [81..120] identified by:
  - **bonusPoints<=80** being false and
  - **bonusPoints<=120** being true.

# Non-Error Causes for bonusPoints

| Parameter | Equivalence Partition |
|---|---|
| bonusPoints | (*) Long.MIN_VALUE..0 |
| | 1..80 |
| | 81..120 |
| | 121..Long.MAX_VALUE |




- [121..Long.MAX VALUE] identified by:
  - **bonusPoints<=80** being false
    and
  - **bonusPoints<=120** being false

# Non-Error Causes for bonusPoints

| Parameter | Equivalence Partition |
|---|---|
| bonusPoints | (*) Long.MIN_VALUE..0 |
| | 1..80 |
| | 81..120 |
| | 121..Long.MAX_VALUE |

- [Long.MIN VALUE..0] is an error partition – not used
- [1..80] is a normal partition, identified by:
  - **bonusPoints<=80** being true (as we disallow bonusPoints<1)
- [81..120] identified by:
  - **bonusPoints<=80** being false
    and
  - **bonusPoints<=120** being true.
- [121..Long.MAX VALUE] identified by:
  - **bonusPoints<=80** being false
    and
  - **bonusPoints<=120** being false

# Non-Error Causes for bonusPoints

| Parameter | Equivalence Partition |
|---|---|
| bonusPoints | (*) Long.MIN_VALUE..0 |
| | 1..80 |
| | 81..120 |
| | 121..Long.MAX_VALUE |

- [Long.MIN VALUE..0] is an error partition – not used
- [1..80] is a normal partition, identified by:
  - **bonusPoints<=80** being true (as we disallow bonusPoints<1)
- [81..120] identified by:
  - **bonusPoints<=80** being false
    and
  - **bonusPoints<=120** being true.
- [121..Long.MAX VALUE] identified by:
  - **bonusPoints<=80** being false
    and
  - **bonusPoints<=120** being false

# Notes

- Fewer causes is better: reduces size of the decision table
- N partitions should lead to no more than Log2(N) expressions:
- for example, 10 partitions could realistically be turned into about 3 or 4 expressions
- For the example above, a third expression **bonusPoints>120** is not required – just negate **bonusPoints<=120**
- A consistent approach to developing the boolean expressions:
  - Reduces mistakes
  - Makes review easier
- In our example, used the <= operator for a consistent logical style

# Causes for goldCustomer

- The partition true is a normal partition, and can be identified by the expression:

  **goldCustomer**

  being true

- The partition false is a normal partition, and can be identified by the expression:

  **goldCustomer**

  being false

# Notes

- **goldCustomer** is already a boolean expression, so it is redundant to state goldCustomer==true and goldCustomer==false as separate causes
- Avoid double negatives: it is much easier to work with positive expressions
  - The expression “goldCustomer” being true is much easier to understand
  - Than the expression “not goldCustomer” being false
- Handling boolean and enum types is generally very straightforward

# Causes for giveDiscount()

We have developed three causes:

1. bonusPoints ≤ 80
2. bonusPoints ≤ 120
3. goldCustomer

- Following this approach
- Leads to similar results every time
- Other approaches may lead to a logically equivalent set of causes which may look quite different
- It takes experience to develop causes that are easy to use and review
- It is bad practice to include logical operators in the causes

# Output Partitions for giveDiscount()

| Parameter | Equivalence Partition |
|---|---|
| Return Value | FULLPRICE<br>DISCOUNT<br>ERROR |

- Normal Effects:
  - Return value==FULLPRICE
  - Return value==DISCOUNT
- Error Effects (don’t use):
  - Return value==ERROR

# Notes

- For output enum values it is clearer to identify each explicitly, as shown here
- It would be equally valid to identify the partition DISCOUNT by the expression return value==FULLPRICE being false, but this does not reduce the size of the table, and makes further development of the test much more difficult
- Minimising the number of **effects** is not important – minimising the number of **causes** is

# Feasible Combinations of Causes

- In order to build the decision table, all the combinations of causes must be identified
- Often all the combinations of the causes are not logically possible
- In this example, it is not possible for both
  bonusPoints ≤ 80 to be true
  and
  bonusPoints ≤ 120 be false
- First identify all combinations in a table (with $2^N$ columns)
- Then remove the impossible combinations

# Empty Combinations Table, bonusPoints>0

| Causes | Combinations | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | | | | | | | | |
| bonusPoints $\leq$ 120 | | | | | | | | |
| goldCustomer | | | | | | | | |

- We have 3 causes, therefore 8 columns ($8=2^3$)

# Fill The Table Systematically from True to False

- Complete the combination columns systematically, starting with all T (for true) at the left-hand side of the table, and ending with all F (for false) at the right-hand side

| | |
|---|---|
| 1 cause produces 2 combinations: | T, F |

# Fill The Table Systematically from True to False

- Complete the combination columns systematically, starting with all T (for true) at the left-hand side of the table, and ending with all F (for false) at the right-hand side

| | |
|---|---|
| 1 cause produces 2 combinations: | T, F |
| 2 causes produce 4 combinations: | TT, TF, FT, FF |

# Fill The Table Systematically from True to False

- Complete the combination columns systematically, starting with all T (for true) at the left-hand side of the table, and ending with all F (for false) at the right-hand side

| | |
|---|---|
| 1 cause produces 2 combinations: | T, F |
| 2 causes produce 4 combinations: | TT, TF, FT, FF |
| 3 causes produce 8 combinations: | TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF |

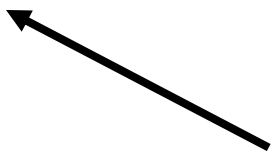
# Fill The Table Systematically from True to False

- Complete the combination columns systematically, starting with all T (for true) at the left-hand side of the table, and ending with all F (for false) at the right-hand side

| | |
|---|---|
| 1 cause produces 2 combinations: | T, F |
| 2 causes produce 4 combinations: | TT, TF, FT, FF |
| 3 causes produce 8 combinations: | TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF |
| 4 causes produce 16 combinations: | TTTT, TTTF, TTFT, TTFF TFTT, TFTF, TFFT, TFFF, FTTT, FTTF, FTFT, FTFF, FFTT, FFTF, FFFT, FFFF |

# Combinations Table where bonusPoints>0

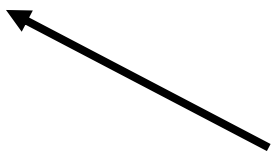
| Causes | Combinations |
| --- | --- |
| bonusPoints $\leq$ 80 | |
| bonusPoints $\leq$ 120 | |
| goldCustomer | |



3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

# Combinations Table where bonusPoints>0

| Causes | Combinations | |
|---|---|---|
| bonusPoints $\leq$ 80 | T | |
| bonusPoints $\leq$ 120 | T | |
| goldCustomer | T | |



3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

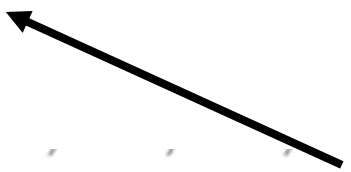
# Combinations Table where bonusPoints>0

| Causes | Combinations | | |
|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | |
| bonusPoints $\leq$ 120 | T | T | |
| goldCustomer | T | F | |

3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

# Combinations Table where bonusPoints>0

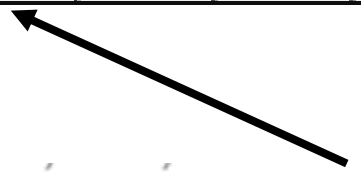
| **Causes** | **Combinations** | | | |
|---|---|---|---|---|
| bonusPoints $\leq 80$ | T | T | T | |
| bonusPoints $\leq 120$ | T | T | F | |
| goldCustomer | T | F | T | |



3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

# Combinations Table where bonusPoints>0

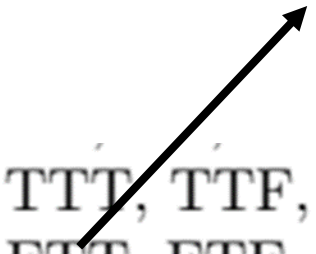
| **Causes** | **Combinations** | | | | |
|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | |
| bonusPoints $\leq$ 120 | T | T | F | F | |
| goldCustomer | T | F | T | F | |



3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

# Combinations Table where bonusPoints>0

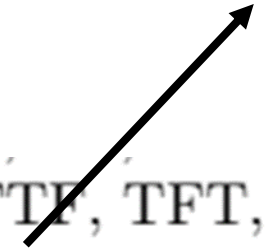
| **Causes** | **Combinations** | | | | | |
|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | F | |
| bonusPoints $\leq$ 120 | T | T | F | F | T | |
| goldCustomer | T | F | T | F | T | |



3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

# Combinations Table where bonusPoints>0

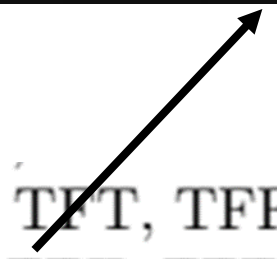
| **Causes** | **Combinations** | | | | | | |
|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | F | F | |
| bonusPoints $\leq$ 120 | T | T | F | F | T | T | |
| goldCustomer | T | F | T | F | T | F | |



3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

# Combinations Table where bonusPoints>0

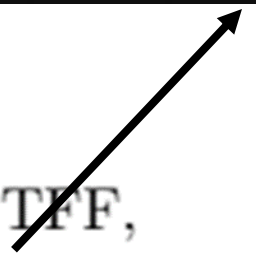
| **Causes** | **Combinations** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | F | F | F | |
| bonusPoints $\leq$ 120 | T | T | F | F | T | T | F | |
| goldCustomer | T | F | T | F | T | F | T | |



3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

# Combinations Table where bonusPoints>0

| **Causes** | **Combinations** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | F | F | T | T | F | F |
| goldCustomer | T | F | T | F | T | F | T | F |



3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

# Completed Combinations Table where bonusPoints>0

| **Causes** | **Combinations** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | F | F | T | T | F | F |
| goldCustomer | T | F | T | F | T | F | T | F |

3 causes produce 8 combinations: TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF

# Identify Impossible Combinations

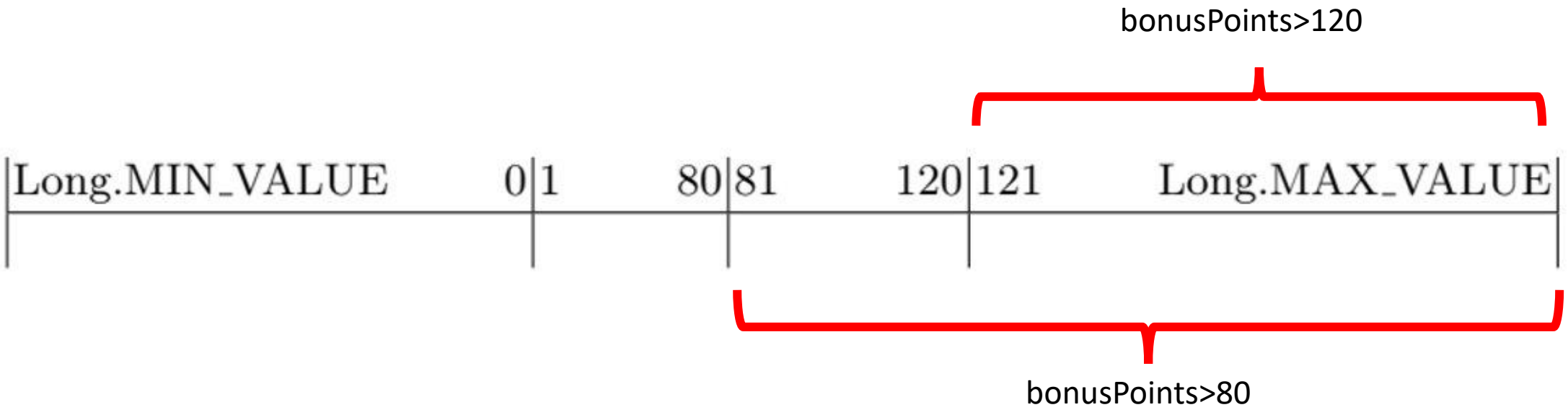
| **Causes** | **Combinations** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | F | F | T | T | F | F |
| goldCustomer | T | F | T | F | T | F | T | F |

# Identify Impossible Combinations

| **Causes** | **Combinations** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | F | F | T | T | F | F |
| goldCustomer | T | F | T | F | T | F | T | F |

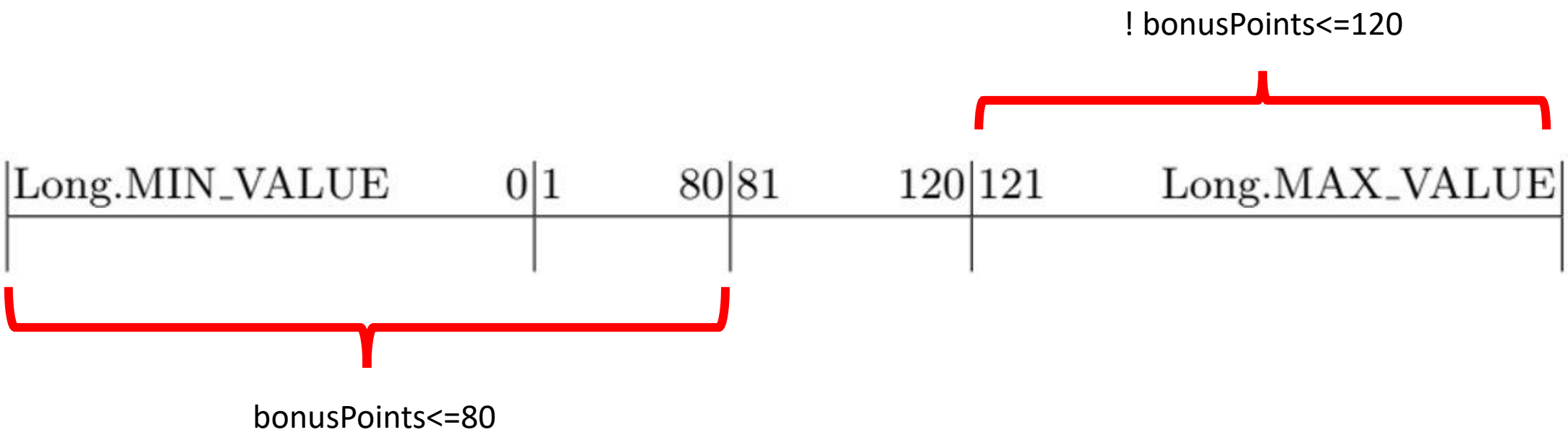
# Identify Impossible Combinations

- If bonusPoints is greater than 120 it must also be greater than 80



# Identify Impossible Combinations

- If bonusPoints <= 120 is false
- Then bonusPoints <= 80 can't be true



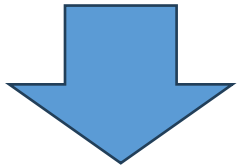
# Identify Impossible Combinations

- The following combination is infeasible
  - bonusPoints<=80, !bonusPoints<=120
- If bonusPoints is greater than 120 it must also be greater than 80

| **Causes** | **Combinations** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | F | F | T | T | F | F |
| goldCustomer | T | F | T | F | T | F | T | F |

# Feasible Combinations of Causes for giveDiscount() where bonusPoints>0

| **Causes** | **Combinations** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | F | F | T | T | F | F |
| goldCustomer | T | F | T | F | T | F | T | F |



| **Causes** | **Combinations** | | | | | |
|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |

# In Practice

- Only one table needs to be used, and developed step-by-step
- The infeasible columns can be crossed out, instead of being removed

| **Causes** | **Combinations** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | ~~T~~ | ~~T~~ | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | ~~F~~ | ~~F~~ | T | T | F | F |
| goldCustomer | T | F | ~~T~~ | ~~F~~ | T | F | T | F |

# Decision Table

- A software test decision table maps each combination of causes to its specified effect
- Using all feasible combinations of causes as a basis, a decision table is initialised with all causes and effects
- Columns are numbered for the rules
- The effects rows are initially left blank

| Causes | Combinations | | | | | |
|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |

# Initial Decision Table giveDiscount() where bonusPoints>0

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| **Effects** | | | | | | |

# Initial Decision Table giveDiscount() where bonusPoints>0

| | | Rules | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | | |
| | bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| | bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| | goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | | |

# Initial Decision Table giveDiscount() where bonusPoints>0

| | | Rules | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | | |
| | bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| | bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| | goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | | |
| return value == FULLPRICE | | | | | | | |
| return value == DISCOUNT | | | | | | | |

| Causes | Combinations | | | | | |
|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |

# Initial Decision Table giveDiscount() where bonusPoints>0

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | | | | | | |
| return value == DISCOUNT | | | | | | |

| Causes | Combinations | | | | | |
|---|---|---|---|---|---|---|
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |

# Initial Decision Table giveDiscount() where bonusPoints>0

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | | | | | | |
| return value == DISCOUNT | | | | | | |

Note the test item: **giveDiscount()**
And the condition: **bonusPoints>0**

# Adding Rules

- The rules must be (a) complete, and (b) independent
- Exactly one rule, must be selected by any combination of the input causes
- The effects for each rule can now be completed from the specification

# Adding Rules

| | 1 |
|---|---|
| **Causes** | |
| bonusPoints ≤ 80 | T |
| bonusPoints ≤ 120 | T |
| goldCustomer | T |
| **Effects** | |
| return value == FULLPRICE | T |
| return value == DISCOUNT | F |

- Rule 1 applies when:
  - bonusPoints ≤ 80 is true
  - bonusPoints ≤ 120 is true
  - goldCustomer is true
- bonusPoints is less than or equal to 80 (but greater than 0), and goldCustomer is true
- According to the specification, the expected result is FULLPRICE

# Notes

- It is good practice to complete both the T and F values for the effects
- Here we have used T for FULLPRICE and F for DISCOUNT

| **Effects** | | | | | | | |
|---|---|---|---|---|---|---|---|
| | return value == FULLPRICE | T | | | | | |
| | return value == DISCOUNT | F | | | | | |

- In more complex examples, a rule may have multiple effects
- Also, when reviewing the table later, blank entries can be confused with an incomplete table

# Decision Table in Progress giveDiscount() where bonusPoints>0

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | | | | | |
| return value == DISCOUNT | F | | | | | |

# Decision Table in Progress giveDiscount() where bonusPoints>0

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | | | | | |
| return value == DISCOUNT | F | | | | | |

- Complete the other rules in the same way

# Decision Table in Progress giveDiscount() where bonusPoints>0

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | | | | |
| return value == DISCOUNT | F | F | | | | |

- Rule 2

# Decision Table in Progress giveDiscount() where bonusPoints>0

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | | | |
| return value == DISCOUNT | F | F | T | | | |

- Rule 3

# Decision Table in Progress giveDiscount() where bonusPoints>0

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | T | | |
| return value == DISCOUNT | F | F | T | F | | |

- Rule 4

# Decision Table in Progress giveDiscount() where bonusPoints>0

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | T | F | |
| return value == DISCOUNT | F | F | T | F | T | |

- Rule 5

# Decision Table in Progress giveDiscount() where bonusPoints>0

| | **Rules** | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | T | F | F |
| return value == DISCOUNT | F | F | T | F | T | T |

- Rule 6

# Final Decision Table for giveDiscount() where bonusPoints>0

| | **Rules** | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | T | F | F |
| return value == DISCOUNT | F | F | T | F | T | T |

# BREAK

# Final Decision Table for giveDiscount() where bonusPoints>0

| | **Rules** | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | T | F | F |
| return value == DISCOUNT | F | F | T | F | T | T |

# Large Decision Tables

- For small decision tables, no optimisation is required
- Optimisations to reduce the size of large tables are discussed in “Decision Tables in More Detail” next week

# Verifying the Decision Table

| | 1 |
|---|---|
| **Causes** | |
| bonusPoints ≤ 80 | T |
| bonusPoints ≤ 120 | T |
| goldCustomer | T |
| **Effects** | |
| return value == FULLPRICE | T |
| return value == DISCOUNT | F |

- Note the common condition: bonusPoints is greater than 0
- **Rule 1** states that when bonusPoints is less than or equal to 80 and goldCustomer is true then the expected return value is FULLPRICE

FULLPRICE if bonusPoints≤120 and not a goldCustomer

FULLPRICE if bonusPoints≤80 and a goldCustomer

DISCOUNT if bonusPoints>120

DISCOUNT if bonusPoints>80 and a goldCustomer

ERROR if any inputs are invalid (bonusPoints<1)

- This is correct

# Verifying the Decision Table

| | | 1 | 2 |
|---|---|---|---|
| **Causes** | | | |
| | bonusPoints ≤ 80 | T | T |
| | bonusPoints ≤ 120 | T | T |
| | goldCustomer | T | F |
| **Effects** | | | |
| | return value == FULLPRICE | T | T |
| | return value == DISCOUNT | F | F |

- **Rule 2** states that when bonusPoints is less than or equal to 80 and goldCustomer is false then the expected return value is FULLPRICE

FULLPRICE if bonusPoints≤120 and not a goldCustomer

FULLPRICE if bonusPoints≤80 and a goldCustomer

DISCOUNT if bonusPoints>120

DISCOUNT if bonusPoints>80 and a goldCustomer

ERROR if any inputs are invalid (bonusPoints<1)

- This is correct

# Verifying the Decision Table

| | | Ru | | |
|---|---|---|---|---|
| | | 1 | 2 | 3 |
| **Causes** | | | | |
| | bonusPoints ≤ 80 | T | T | F |
| | bonusPoints ≤ 120 | T | T | T |
| | goldCustomer | T | F | T |
| **Effects** | | | | |
| | return value == FULLPRICE | T | T | F |
| | return value == DISCOUNT | F | F | T |

- **Rule 3** states that when bonusPoints is greater than 80 and less than or equal to 120, and goldCustomer is true, then the expected return value is DISCOUNT

FULLPRICE if bonusPoints≤120 and not a goldCustomer

FULLPRICE if bonusPoints≤80 and a goldCustomer

DISCOUNT if bonusPoints>120

DISCOUNT if bonusPoints>80 and a goldCustomer

ERROR if any inputs are invalid (bonusPoints<1)

- This is correct

# Verifying the Decision Table

| | | Rules | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| **Causes** | | | | | |
| | bonusPoints $\leq$ 80 | T | T | F | F |
| | bonusPoints $\leq$ 120 | T | T | T | T |
| | goldCustomer | T | F | T | F |
| **Effects** | | | | | |
| | return value == FULLPRICE | T | T | F | T |
| | return value == DISCOUNT | F | F | T | F |

- **Rule 4** states that when bonusPoints is greater than 80 and less than or equal to 120, and goldCustomer is false, then the expected return value is FULLPRICE

FULLPRICE if bonusPoints≤120 and not a goldCustomer

FULLPRICE if bonusPoints≤80 and a goldCustomer

DISCOUNT if bonusPoints>120

DISCOUNT if bonusPoints>80 and a goldCustomer

ERROR if any inputs are invalid (bonusPoints<1)

- This is correct

# Verifying the Decision Table

| | Rules | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| **Causes** | | | | | |
| bonusPoints ≤ 80 | T | T | F | F | F |
| bonusPoints ≤ 120 | T | T | T | T | F |
| goldCustomer | T | F | T | F | T |
| **Effects** | | | | | |
| return value == FULLPRICE | T | T | F | T | F |
| return value == DISCOUNT | F | F | T | F | T |

- **Rule 5** states that when bonusPoints is greater than 120, and goldCustomer is true, then the expected return value is DISCOUNT

FULLPRICE if bonusPoints≤120 and not a goldCustomer

FULLPRICE if bonusPoints≤80 and a goldCustomer

DISCOUNT if bonusPoints>120

DISCOUNT if bonusPoints>80 and a goldCustomer

ERROR if any inputs are invalid (bonusPoints<1)

- This is correct

# Verifying the Decision Table

| | | Rules | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | | |
| | bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| | bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| | goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | | |
| | return value == FULLPRICE | T | T | F | T | F | F |
| | return value == DISCOUNT | F | F | T | F | T | T |

- **Rule 6** states that when bonusPoints is greater than 120, and goldCustomer is false, then the expected return value is DISCOUNT

FULLPRICE if bonusPoints≤120 and not a goldCustomer

FULLPRICE if bonusPoints≤80 and a goldCustomer

DISCOUNT if bonusPoints>120

DISCOUNT if bonusPoints>80 and a goldCustomer

ERROR if any inputs are invalid (bonusPoints<1)

- This is correct

# 2. Test Coverage Items

- Each rule is a TCI

Table 4.11: DT Test Coverage Items for giveDiscount() where bonusPoints>0

| TCI | Rule | Test Case |
|---|---|---|
| DT1 | 1 | To be completed later |
| DT2 | 2 | |
| DT3 | 3 | |
| DT4 | 4 | |
| DT5 | 5 | |
| DT6 | 6 | |

# 3. Test Cases

- In DT testing, each test coverage item covered by a separate test case
- Not possible to have multiple combinations in the same test case
- Test data is selected by picking input values that satisfy the causes, and output values that match the effects, for each rule to be tested
- Use the previously selected **equivalence partition values** for the input parameters
  - Allows duplicate tests to be easily identified and removed
  - Makes the task of reviewing the tests easier
- Start with **candidate** test cases (X3.1, X3.2, etc.) until duplicates removed

# 3. Candidate Test Cases (X3.1)

| | 1 |
|---|---|
| **Causes** | |
| bonusPoints $\leq$ 80 | T |
| bonusPoints $\leq$ 120 | T |
| goldCustomer | T |
| **Effects** | |
| return value == FULLPRICE | T |
| return value == DISCOUNT | F |

| **ID** | **TCI Covered** | **Inputs** | | **Exp. Results** |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |

# 3. Candidate Test Cases (X3.2)

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | T | F | F |
| return value == DISCOUNT | F | F | T | F | T | T |

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | | | | |
| X3.4 | | | | |
| X3.5 | | | | |
| X3.6 | | | | |

# 3. Candidate Test Cases (X3.3)

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | T | F | F |
| return value == DISCOUNT | F | F | T | F | T | T |

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | DT3 | 100 | true | DISCOUNT |
| X3.4 | | | | |
| X3.5 | | | | |
| X3.6 | | | | |

# 3. Candidate Test Cases (X3.4)

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | T | F | F |
| return value == DISCOUNT | F | F | T | F | T | T |

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | DT3 | 100 | true | DISCOUNT |
| X3.4 | DT4 | 100 | false | FULLPRICE |
| X3.5 | | | | |
| X3.6 | | | | |

# 3. Candidate Test Cases (X3.5)

| | | Rules | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | | |
| | bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| | bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| | goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | | |
| | return value == FULLPRICE | T | T | F | T | F | F |
| | return value == DISCOUNT | F | F | T | F | T | T |

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | DT3 | 100 | true | DISCOUNT |
| X3.4 | DT4 | 100 | false | FULLPRICE |
| X3.5 | DT5 | 200 | true | DISCOUNT |
| X3.6 | | | | |

# 3. Candidate Test Cases (X3.6)

| | Rules | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | |
| bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | |
| return value == FULLPRICE | T | T | F | T | F | F |
| return value == DISCOUNT | F | F | T | F | T | T |

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | DT3 | 100 | true | DISCOUNT |
| X3.4 | DT4 | 100 | false | FULLPRICE |
| X3.5 | DT5 | 200 | true | DISCOUNT |
| X3.6 | DT6 | 200 | false | DISCOUNT |

# 3. Candidate Test Cases (complete)

| | | Rules | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| **Causes** | | | | | | | |
| | bonusPoints $\leq$ 80 | T | T | F | F | F | F |
| | bonusPoints $\leq$ 120 | T | T | T | T | F | F |
| | goldCustomer | T | F | T | F | T | F |
| **Effects** | | | | | | | |
| | return value == FULLPRICE | T | T | F | T | F | F |
| | return value == DISCOUNT | F | F | T | F | T | T |

| **ID** | **TCI Covered** | **Inputs** | | **Exp. Results** |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | DT3 | 100 | true | DISCOUNT |
| X3.4 | DT4 | 100 | false | FULLPRICE |
| X3.5 | DT5 | 200 | true | DISCOUNT |
| X3.6 | DT6 | 200 | false | DISCOUNT |

# Removing Duplicate Test Cases

- From the candidate test cases, shown in Table 4.14, we can identify a few duplicates – new test cases that have equivalent test data to previously defined test cases
- They can be removed
- We are able to identify a number of such duplicate test cases:
  - X3.1 duplicates test case T1.1

# X3.1 Duplicates T1.1

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | DT3 | 100 | true | DISCOUNT |
| X3.4 | DT4 | 100 | false | FULLPRICE |
| X3.5 | DT5 | 200 | true | DISCOUNT |
| X3.6 | DT6 | 200 | false | DISCOUNT |

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| T1.1 | EP2,5,7 | 40 | true | FULLPRICE |

# X3.2 Duplicates T2.2

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | DT3 | 100 | true | DISCOUNT |
| X3.4 | DT4 | 100 | false | FULLPRICE |
| X3.5 | DT5 | 200 | true | DISCOUNT |
| X3.6 | DT6 | 200 | false | DISCOUNT |

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| T2.1 | BV3,9,11 | 1 | true | FULLPRICE |
| T2.2 | BV4,10,\|11\| | 80 | false | FULLPRICE |

# Why?

| | | 1 | 2 |
|---|---|---|---|
| **Causes** | | | |
| | bonusPoints ≤ 80 | T | T |
| | bonusPoints ≤ 120 | T | T |
| | goldCustomer | T | F |
| **Effects** | | | |
| | return value == FULLPRICE | T | T |
| | return value == DISCOUNT | F | F |

- The data values are different, as X3.2 is based on equivalence partition values (40, false) and T2.2 is based on BVA values (80,false)
- However, T2.2 matches the same rule in the decision table:
  - Rule 2 (bonusPoints ≤ 80 and !goldCustomer)
- So X3.2 would be a duplicate test case for decision table testing

# X3.4 Duplicates T1.2

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | DT3 | 100 | true | DISCOUNT |
| X3.4 | DT4 | 100 | false | FULLPRICE |
| X3.5 | DT5 | 200 | true | DISCOUNT |
| X3.6 | DT6 | 200 | false | DISCOUNT |

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| T1.1 | EP2,5,7 | 40 | true | FULLPRICE |
| T1.2 | EP3,6,[7] | 100 | false | FULLPRICE |

# X3.6 Duplicates T1.3

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| X3.1 | DT1 | 40 | true | FULLPRICE |
| X3.2 | DT2 | 40 | false | FULLPRICE |
| X3.3 | DT3 | 100 | true | DISCOUNT |
| X3.4 | DT4 | 100 | false | FULLPRICE |
| X3.5 | DT5 | 200 | true | DISCOUNT |
| X3.6 | DT6 | 200 | false | DISCOUNT |

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| T1.1 | EP2,5,7 | 40 | true | FULLPRICE |
| T1.2 | EP3,6,[7] | 100 | false | FULLPRICE |
| T1.3 | EP4,[6],8 | 200 | false | DISCOUNT |

# Final DT Test Cases for giveDiscount() where bonusPoints>0

| ID | TCI Covered | Inputs | | Exp. Results |
|---|---|---|---|---|
| | | bonusPoints | goldCustomer | return value |
| T3.1 | DT3 | 100 | true | DISCOUNT |
| T3.2 | DT5 | 200 | true | DISCOUNT |

- By selecting the same values as before for each partition, we have made the task of identifying duplicate tests easier
- Alternatively, testers may remove duplicates only during the test implementation, or even refrain from removing them at all
- After removing duplicates, we can now add the decision tables tests to create the complete set of test cases with test case IDs (e.g. T3.1)

# 4. Test Design Verification

- Two parts:
    1. Complete the test coverage item table
    2. Review your work

# Completed Test Coverage Item Table

| TCI | Rule | Test Case |
|---|---|---|
| DT1 | 1 | T1.1 |
| DT2 | 2 | T2.2 |
| DT3 | 3 | T3.1 |
| DT4 | 4 | T1.2 |
| DT5 | 5 | T3.2 |
| DT6 | 6 | T1.3 |

# Reviewing Your Work

1. Every TCI is covered by a test case: the tests are complete
2. Every decision table test case covers a different TCI:
   - no unnecessary tests
3. No duplicate tests (including previously developed tests)

# 5. Implementation

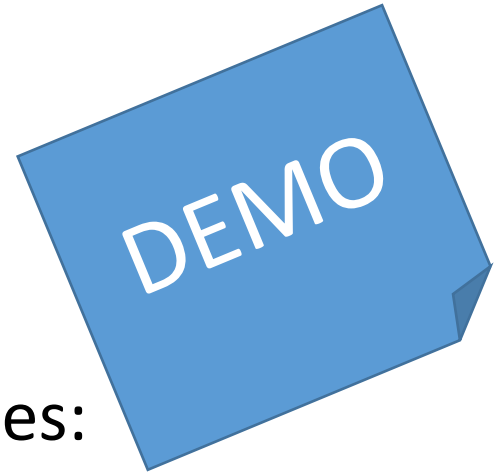
- The full test implementation includes the previously developed equivalence partition and boundary value analysis tests
- Full decision table testing is not achieved unless these previously developed tests are included!!

# OnlineSalesTest with DT Coverage

testData1[] is extended

```
private static Object[][] testData1 = new Ob
    //  test, bonuspoints, goldCustomer, expected output
    { "T1.1",          40L,          true,     FULLPRICE },
    { "T1.2",         100L,         false,     FULLPRICE },
    { "T1.3",         200L,         false,      DISCOUNT },
    { "T1.4",        -100L,         false,         ERROR },
    { "T2.1",           1L,          true,     FULLPRICE },
    { "T2.2",          80L,         false,     FULLPRICE },
    { "T2.3",          81L,         false,     FULLPRICE },
    { "T2.4",         120L,         false,     FULLPRICE },
    { "T2.5",         121L,         false,      DISCOUNT },
    { "T2.6", Long.MAX_VALUE,       false,      DISCOUNT },
    { "T2.7", Long.MIN_VALUE,       false,         ERROR },
    { "T2.8",           0L,         false,         ERROR },
    { "T3.1",         100L,          true,       DISCOUNT },
    { "T3.2",         200L,          true,       DISCOUNT },
};
```

# 6. Test Execution




- Running these tests against the class OnlineSales produces:

```
PASSED: test_giveDiscount("T1.1", 40, true, FULLPRICE)
PASSED: test_giveDiscount("T1.2", 100, false, FULLPRICE)
PASSED: test_giveDiscount("T1.3", 200, false, DISCOUNT)
PASSED: test_giveDiscount("T1.4", -100, false, ERROR)
PASSED: test_giveDiscount("T2.1", 1, true, FULLPRICE)
PASSED: test_giveDiscount("T2.2", 80, false, FULLPRICE)
PASSED: test_giveDiscount("T2.3", 81, false, FULLPRICE)
PASSED: test_giveDiscount("T2.4", 120, false, FULLPRICE)
PASSED: test_giveDiscount("T2.5", 121, false, DISCOUNT)
PASSED: test_giveDiscount("T2.6", 92233720368547758O7, false, DISCOUNT)
PASSED: test_giveDiscount("T2.7", -92233720368547758O8, false, ERROR)
PASSED: test_giveDiscount("T2.8", 0, false, ERROR)
PASSED: test_giveDiscount("T3.1", 100, true, DISCOUNT)
PASSED: test_giveDiscount("T3.2", 200, true, DISCOUNT)
===============================================
Command line suite
Total tests run: 14, Passes: 14, Failures: 0, Skips: 0
===============================================
```

# 7. Test Results

```
PASSED: test_giveDiscount("T1.1", 40, true, FULLPRICE)
PASSED: test_giveDiscount("T1.2", 100, false, FULLPRICE)
PASSED: test_giveDiscount("T1.3", 200, false, DISCOUNT)
PASSED: test_giveDiscount("T1.4", -100, false, ERROR)
PASSED: test_giveDiscount("T2.1", 1, true, FULLPRICE)
PASSED: test_giveDiscount("T2.2", 80, false, FULLPRICE)
PASSED: test_giveDiscount("T2.3", 81, false, FULLPRICE)
PASSED: test_giveDiscount("T2.4", 120, false, FULLPRICE)
PASSED: test_giveDiscount("T2.5", 121, false, DISCOUNT)
PASSED: test_giveDiscount("T2.6", 92233720368547758O7, false, DISCOUNT)
PASSED: test_giveDiscount("T2.7", -92233720368547758O8, false, ERROR)
PASSED: test_giveDiscount("T2.8", 0, false, ERROR)
PASSED: test_giveDiscount("T3.1", 100, true, DISCOUNT)
PASSED: test_giveDiscount("T3.2", 200, true, DISCOUNT)
===============================================
Command line suite
Total tests run: 14, Passes: 14, Failures: 0, Skips: 0
===============================================
```

- All the tests have passed

# Next Week

- DT in More Detail
  - Fault Model
  - Description, Analysis, Test Coverage Items, Test Cases
  - Pitfalls
- Evaluation
  - Limitations: injected faults into the code
  - Strengths & Weaknesses
- Key Points & Notes for Experienced Testers

# This Afternoon/Lab 4

- DT Analysis, Test Coverage Items, Test Cases, Review, Implement
- Quiz
- Deadline: next Monday 13:00 (but try and get it done today)

# This Afternoon's Lab

- Written work:
  - Cause and effects
  - Combinations
  - Decision Table
  - TCI
  - Test Cases (Clearly show the removal of duplicates)
  - Review
- Edit InsuranceTest.java (**add extra tests)**
- Run tests and collect output (on correct and faulty implementations 3 & 4)
- **Quiz**
  - Make sure to use your exact written results in the quiz
  - Enter (copy-and-paste) your test results

# Independent Study

- Read Chapters 3 (more detail) and 4 of the book
- Run the test for the book example (chapter 4) and make sure you understand the test code
  - Layout (for clarity)
  - Matching the identified test cases (for correctness)
- Review the faults (ch03/fault3 and ch04/fault4) in Insurance.java, run the DT tests against them, and make sure you understand why fault 3 is found and fault 4 is not found by the DT tests