
INTRODUCTION TO SOFTWARE ENGINEERING

Structural Testing

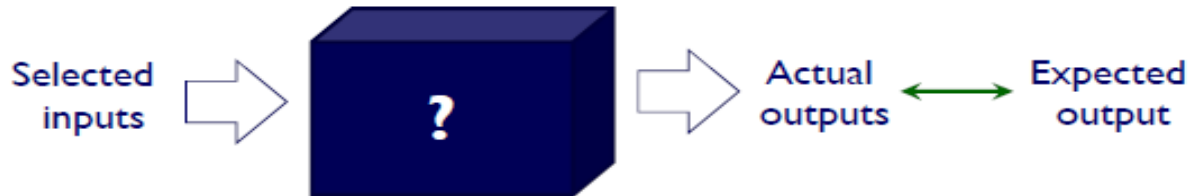
Daniel Sinnig, PhD
d_sinnig@cs.concordia.ca

Department for Computer Science
and Software Engineering

29-July-14



Functional Testing



- A program can be considered as a function that has inputs as a domain and outputs as a range
- Historically functional testing emphasized the input domain but it can be apply for range based test cases as well
- Is not based on the structure of the program (which is unknown)

Types of functional testing

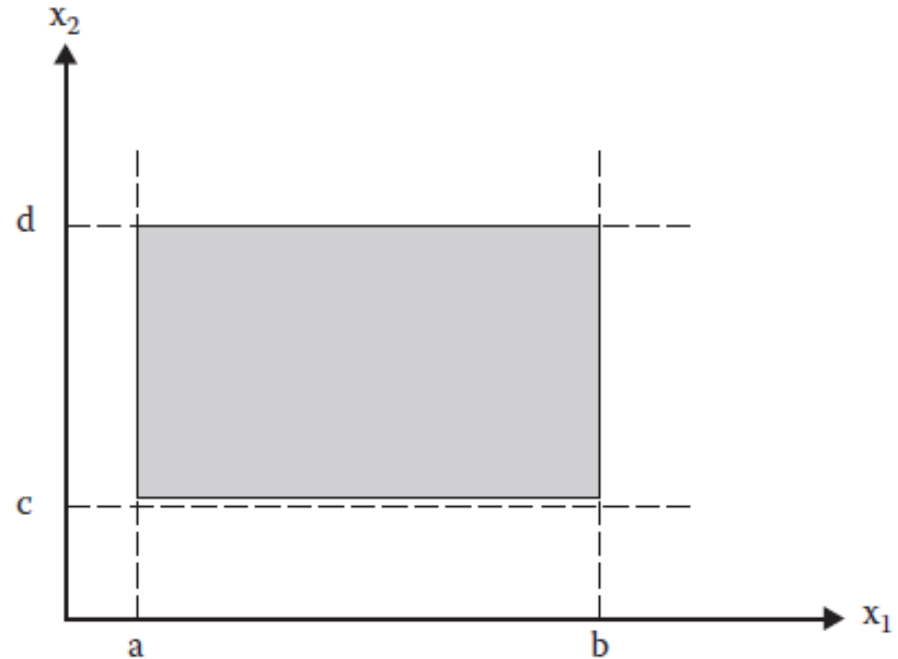
- Boundary testing
- Equivalence class testing
- Decision table testing

Types of functional testing

- **Boundary testing**
- Equivalence class testing
- Decision table testing

Program view for boundary analysis

- Many programs can be viewed as a function F that maps values from a set A (its domain) to values in another set B (its range)
- The input variables of F will have some (possibly unstated) boundaries: $F(x_1, x_2): A \rightarrow B$, with $a \leq x_1 \leq b$ and $c \leq x_2 \leq d$



Boundary Value Testing

- Boundary Value Analysis
- Robustness Testing
- Worst-Case Testing
- Special Value Testing
- Random Testing

Boundary Value Testing

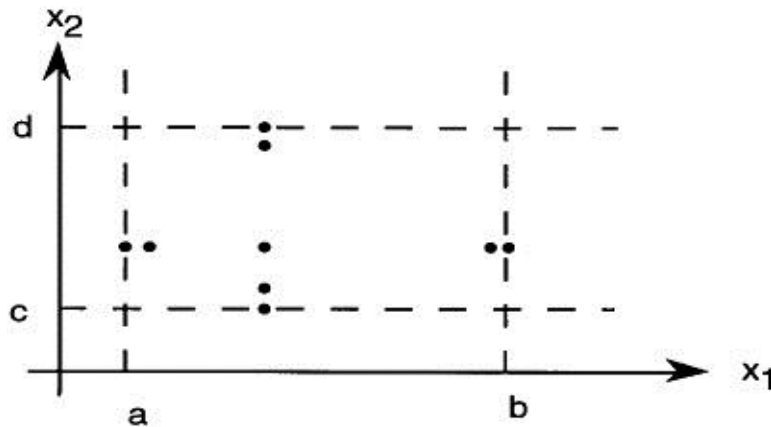
- **Boundary Value Analysis**
- Robustness Testing
- Worst-Case Testing
- Special Value Testing
- Random Testing

Boundary Value Analysis

- Based on “single fault” assumption in reliability theory
 - Failures are only rarely the result of the simultaneous occurrence of two (or more) faults
- The input variable values are used at their min, min+, nom, max-, max
- Text case generation:
 - For all i
 - Values of all but one variable x_i at nominal
 - x_i assumes all 5 values from above
- n variables yield $4n+1$ test cases

Boundary Value Analysis (two variables)

- For two variables x_1 , x_2 , the boundary value analysis test cases will be:
 - $T = \{ \langle x1nom, x2min \rangle, \langle x1nom, x2min+ \rangle, \langle x1nom, x2nom \rangle, \langle x1nom, x2max- \rangle, \langle x1nom, x2max \rangle, \langle x1min, x2nom \rangle, \langle x1min+, x2nom \rangle, \langle x1max-, x2nom \rangle, \langle x1max, x2nom \rangle \}$



Boundary Value Analysis(remarks)

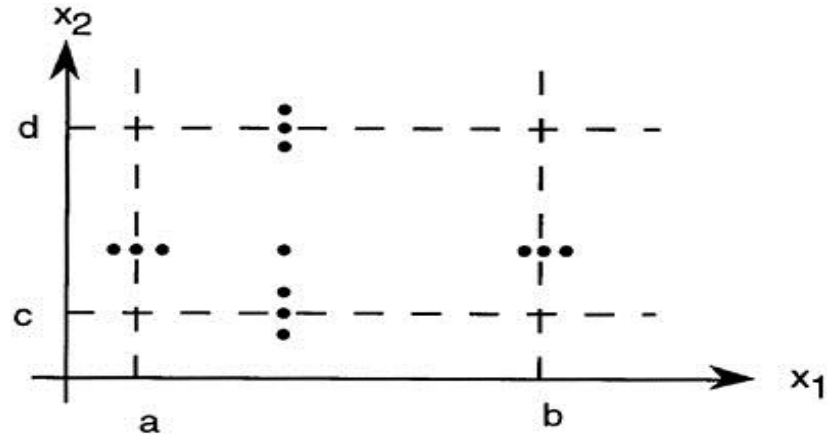
- Can be applied to loop control variables, indices, pointers
- Choose also inputs that invoke *output boundary values* (values on the boundary of output classes)
- Limitations:
 - Works well for functions of several independent variables that represent bounded physical quantities
 - No consideration of the nature of the function nor the semantic meaning of the variables
 - Does not work well for Boolean variables
 - Does not work well for logical variables (PIN, transaction type)

Boundary Value Testing

- Boundary Value Analysis
- **Robustness Testing**
- Worst-Case Testing
- Special Value Testing
- Random Testing

Robustness Testing

- Explicit range checking and exception handling
- In addition to the five values of the variable we add two more:
 - slightly greater than the maximum (**max+**)
 - slightly less than the minimum (**min-**)



Robustness testing (cont.)

- Help to test that:
 - Error cases are properly handled,
 - Error messages are properly displayed
 - System is able to recover from the error cases

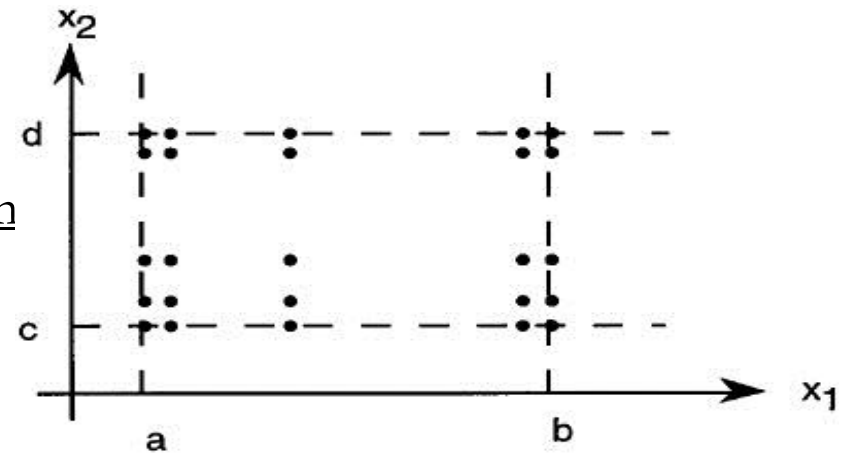
- n variables yield $6n+1$ test cases

Boundary Value Testing

- Boundary Value Analysis
- Robustness Testing
- **Worst-Case Testing**
- Special Value Testing
- Random Testing

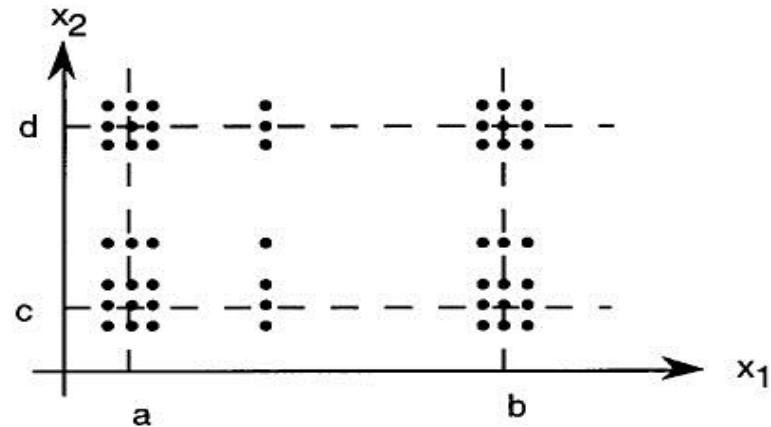
Worst-Case Testing

- **Rejection of the single fault assumption**
- For each variable, we have min, min+, max-, max and then we take the Cartesian product of these sets to generate test cases
- For n input variables the number of test cases is 5^n



Robust Worst-Case Testing

- Add min-, max+
- For n input variables the number of test cases is 7^n



Boundary Value Testing

- Boundary Value Analysis
- Robustness Testing
- Worst-Case Testing
- **Special Value Testing**
- Random Testing

Special Value Testing

- Tester uses his domain knowledge, experience with same programs
- No guidelines other than engineer's judgment
- Also called “ad-hoc” or “seat-of-the pants” testing
- Despite all the apparent negatives, it can be useful

Boundary Value Testing

- Boundary Value Analysis
- Robustness Testing
- Worst-Case Testing
- Special Value Testing
- **Random Testing**

Random Testing

- The most widely practiced way of functional testing
- Basic idea:
 - Use a random number generator to produce random values (instead of **max**, **min**, ...) for test cases
 - Test cases are generated randomly until at least one of each output occurs

Review: Boundary Value Testing

- A testing technique in which test data are chosen to lie along “boundaries” of the input domain [or output range] classes, data structures, procedure parameters, etc.
- Problems:
 - Serious gaps (non-completeness)
 - Massive redundancy
- Avoiding these problems is the motivation for equivalence class testing

Types of functional testing

- Boundary testing
- **Equivalence class testing**
- Decision table testing

Equivalence class testing

- A testing technique in which test data is derived by partitioning the input domain into disjoint sub-sets (equivalence classes).
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
 - where all members of a class are related (by an equivalence relation)
- Test cases are identified by using one element from each equivalence class.
- Underlying technique: **Set partitioning**

Set Partitioning and Equivalence Relation

- Given a set A , and a set of subsets A_1, A_2, \dots, A_n of A , the subsets are a partition of A iff:

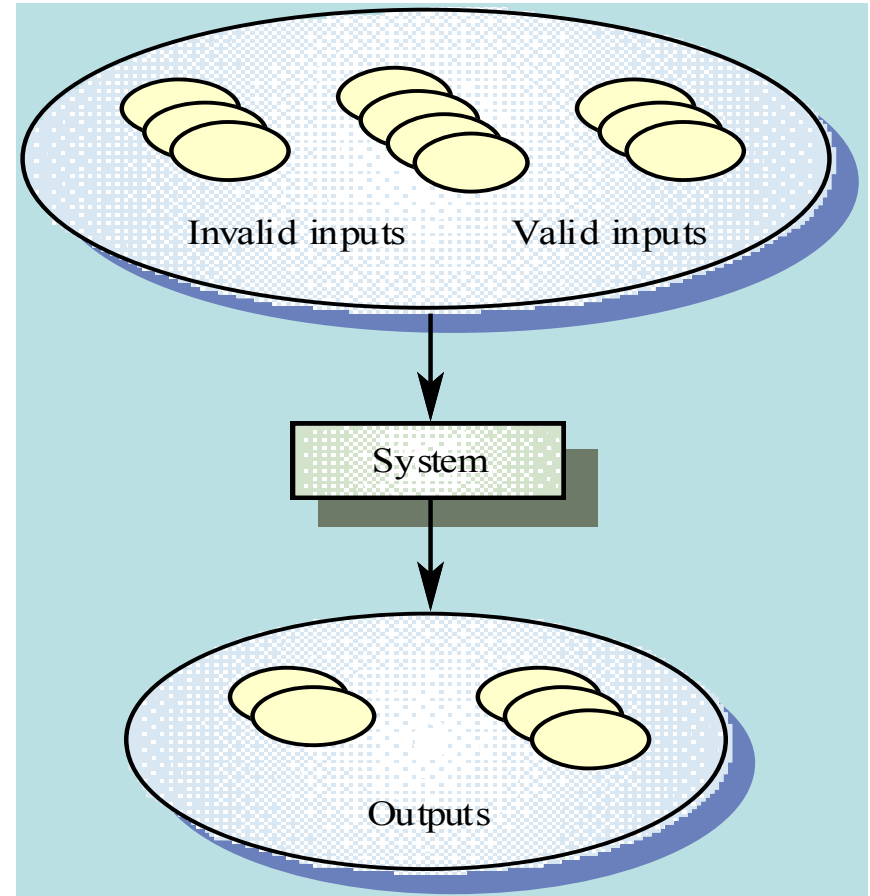
$$A_1 \cup A_2 \cup \dots \cup A_n = A \text{ and } i \neq j \Rightarrow A_i \cap A_j = \emptyset$$

- A relation $R \subseteq A \times A$ is an equivalence relation if R is reflexive, symmetric, and transitive.
 - Reflexive: $\forall a \in A. (a, a) \in R$
 - Symmetric: $\forall a, b \in A. (a, b) \in R \Rightarrow (b, a) \in R$
 - Transitive: $\forall a, b, c \in A. (a, b) \in R \wedge (b, c) \in R \Rightarrow (a, c) \in R$
- Each partition forms an equivalence class

Implication for testing

- Testing based on partitioning provides a form of completeness as the entire set of represented
- The disjointness of the sets assures a form of non-redundancy

Equivalence partitioning



Choosing the equivalence relation

- Choosing the right equivalence relation is crucial.
- Often the equivalence relation is chosen based on:
 - The range of the input variables (e.g., $a \leq x \leq b$)
 - The expected program output (e.g., “No triangle”, “Isosceles”, etc)
 - The required calculation (e.g., increasing a regulate date is computed differently than increasing the end of a month)
 - Etc.

Variations of equivalence class testing

- Same two orthogonal dimensions as in boundary value analysis
- **Robustness**
 - Robust-normal distinguishes valid data from invalid data
- **Single/Multiple Fault Assumption**
 - Weak-strong distinguishes single from multiple fault
- Combinations give four variations.

Equivalence Class Testing

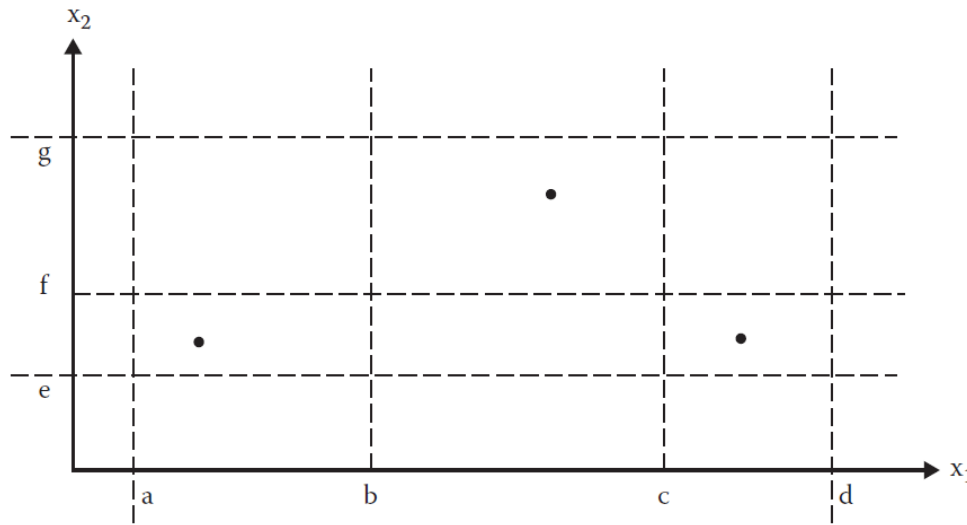
- Weak Normal Equivalence Class Testing
- Strong Normal Equivalence Class Testing
- Weak Robust Equivalence Class Testing
- Strong Robust Equivalence Class Testing

Running example

- Consider a two-variable function $F(x_1, x_2)$ where the equivalence classes are defined by the following intervals for x_1 and x_2 .
 - $a \leq x_1 \leq d$, with intervals $[a, b)$, $[b, c)$, $[c, d]$
 - $e \leq x_2 \leq g$, with intervals $[e, f)$, $[f, g]$

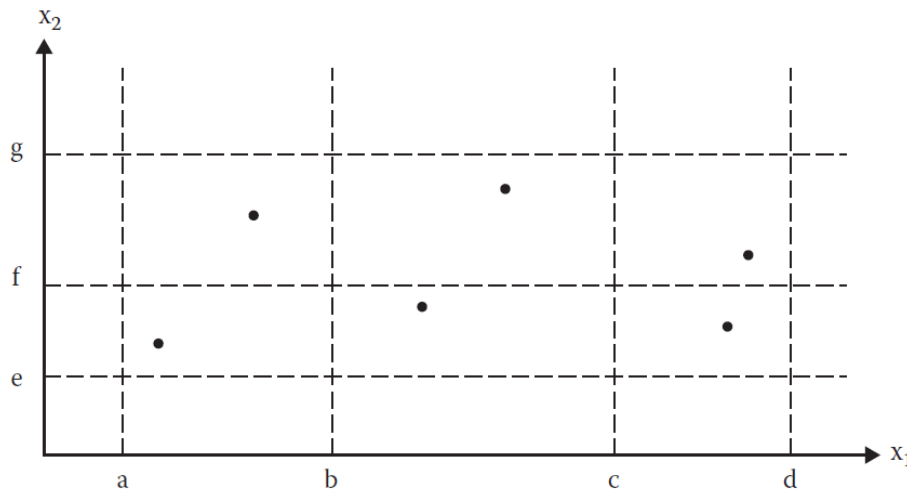
Weak Normal Equivalence Class Testing

- Accomplished by using one value from each (valid) equivalence class
- A valid equivalence class is within valid input range.
- Single fault assumption



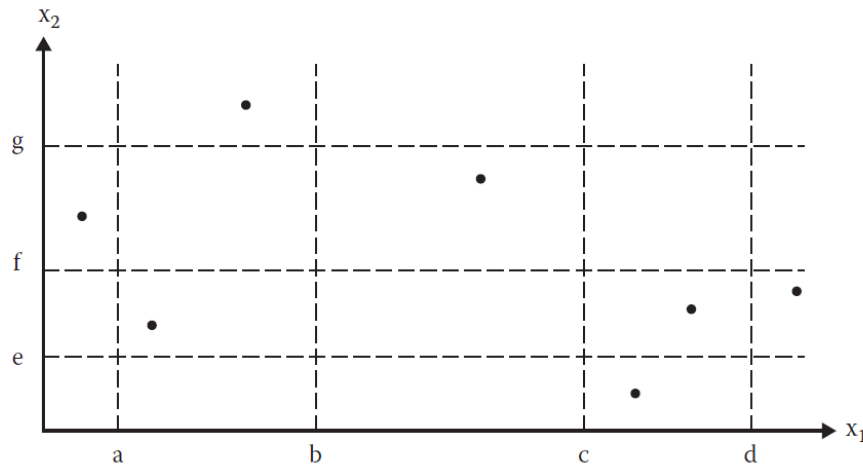
Strong Normal Equivalence Class Testing

- Based on multiple fault assumption
- Requires test cases for each element of the Cartesian product of the “valid” equivalent classes.
 - Covers all equivalence classes and requires one test case for each possible combination of (valid) inputs



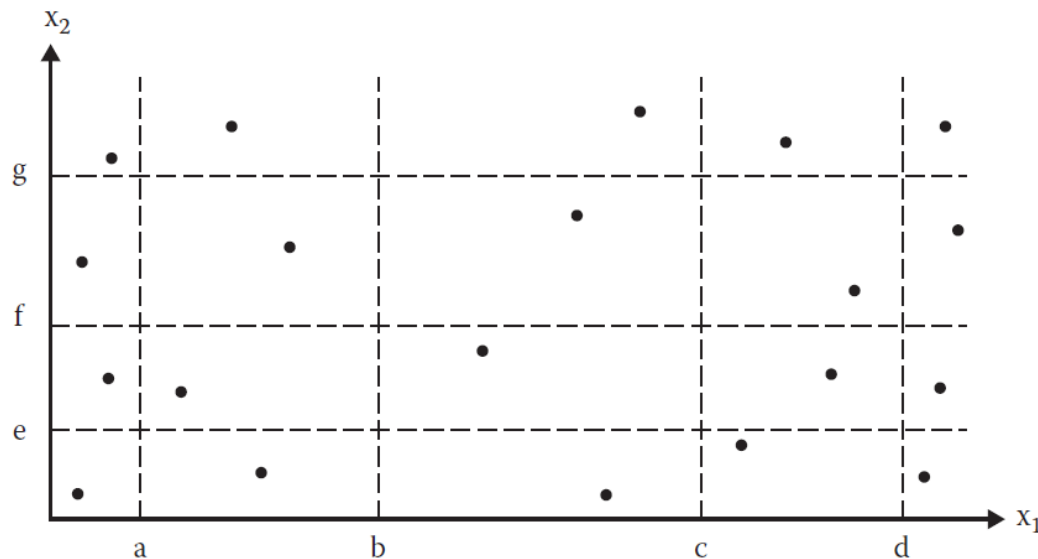
Weak Robust Equivalence Class Testing

- Based on single fault assumption
- Weak normal equivalence testing + testing of invalid values such that each invalid “range” was exercised ones.



Strong Robust Equivalence Class Testing

- Based on multiple fault assumption
- Requires test cases covering each element of the Cartesian product of all equivalence classes (valid and invalid)



Limitations

- Same as those for boundary value testing
 - Does not work well for Boolean variables
 - Does not work well for logical variables
 - When variables are not independent – i.e. are dependent
- For robust variations same as for boundary value testing
 - Difficult or impossible to determine expected values for invalid variable values

Guidelines for equivalence classes

- Equivalence Class Testing is appropriate when input data is defined in terms of intervals and sets of discrete values.
- Equivalence Class Testing is strengthened when combined with Boundary Value Testing
- Strong equivalence makes the presumption that variables are independent.
 - If that is not the case, redundant test cases may be generated
- The key point in equivalence class testing is the choice of the equivalence relation that determines the classes
 - Several tries may be required before the “right” equivalence relation is discovered
 - If the equivalence classes are chosen wisely, the potential redundancy among test cases is greatly reduced.

Guidelines for equivalence classes (cont.)

1. If an input condition specifies **range**,
 - one valid and two invalid equivalence classes are needed
2. If a condition requires **a specific value**,
 - then one valid and two invalid equivalence classes are needed
3. If an input condition specifies **a member of a set**,
 - one valid and one invalid equivalence class are needed
4. If an input condition is **Boolean**,
 - one valid and one invalid class are needed

Exercise

The function “generate_grading” is specified as follows:

The function takes two arguments examMark (max. 75) and courseworkMark (max. 25) as input. Based on the sum of the marks submitted the function calculates the overall course grade as follows:

- ‘A’ – sum is greater or equal than 70
 - ‘B’ – sum is greater or equal than 50 and less than 70
 - ‘C’ – sum is greater or equal than 30 and less than 50
 - ‘D’ – sum is less than 30
-
- Define suitable equivalence classes for:
 - Strong normal equivalence class testing
 - Strong robust equivalence class testing

Types of functional testing

- Boundary testing
- Equivalence class testing
- **Decision table testing**

Decision Table testing

- A testing technique in which test data is derived from a **decision table**.
- Decision table uses a tabular format to capture the logic behind a decision-making process
 - Useful for modeling complicated logic
 - Displays a combination of conditions to be met and actions to be taken
 - Each column represents a unique combination (of conditions and actions)
- Example of a single condition/action:
If there is smoke (*condition*) **then** push the fire alarm (*action*)

Decision Table Terminology

- Condition entries restricted to binary values
 - We have a **limited entry table**
- Condition entries have more than two values
 - We have an **extended entry table**

Constructing a Decision Table

- Determines the conditions and values each condition can take
- Find all possible actions/effects that can occur
- List all rules
- Define the actions for each rule
- Simplify the table

Decision Table Structure

Condition labels	Condition entries
Action labels	Action entries

Decision Tables, simple example.

(On any given day.) For leisure, I generally read a good book but I only take some leisure time after I have finished my homework. For leisure on Fridays, I go out to see a repertoire movie, but only if I have completed my homework before 6pm, otherwise (because it is too late for the movie) I just read a good book at home.

[Solution, to be given in class]

Decision Tables, simple example [Solution]

Homework done	Y	Y	Y	Y	N	N	N	N
Before 6pm	N	Y	N	Y	N	Y	N	Y
Friday	N	N	Y	Y	N	N	Y	Y
Read book	✓	✓	✓					
See Movie				✓				
No Leisure					✓	✓	✓	✓

Optimization

- Decision tables grow exponentially, if we add conditions.
 - Number of columns = $2^{\text{number of conditions}}$
- Optimizing size by
 - Using “don’t care” values (*)
 - May occur if a certain condition is irrelevant for an given action
 - Identifying impossible situations (-)
 - May occur if conditions are not independent

Decision Tables, simple example [Optimized Solution]

Homework done	Y	Y	Y	N
Before 6pm	N	Y	Y	*
Friday	*	N	Y	*
Read book	✓	✓		
See Movie			✓	
No Leisure				✓

Exercise

Create a decision table for the following specification:

If there is smoke
and you have given up smoking
and someone else is not smoking
then push the fire alarm
and leave the building
but if there is no smoke
then do nothing

Exercise

Create a decision table for the following specification:

If there is smoke (*condition 1*)
and you have given up smoking (*condition 2*)
and someone else is not smoking (*condition 3*)
then push the fire alarm (*action 1*)
and leave the building (*action 2*)
but if there is no smoke (*condition 1*)
then do nothing (*action 1*)

Condition and Action Labels

There is smoke
You have given up smoking
Someone else is smoking
Push the fire alarm
Leave the building
Do nothing

Condition and Action Entries

There is smoke	Y	Y	Y	N
You have given up smoking	Y	N	Y	*
Someone else is smoking	N	*	Y	*
Push the fire alarm	1			
Leave the building	2			
Do nothing		1	1	1

The numbers appearing in the action entry section tell you which actions are to be taken and in what order.

Exercise 2

Create a decision table for the following specification:

- Three types of customers are defined: a regular, a silver customer, and a gold customer.
- A regular customer receives normal print rates ad delivery.
- A silver customer gets an 8% discount on all quotes and is placed ahead of all regular customers in the job queue.
- A gold customer gets a 15% reduction in quoted prices and is placed ahead of both regular and silver customers in the job queue.
- A special discount of $x\%$ in addition to other discounts can be applied to any customer's quote at the discretion of management.

Exercise 2: Solution

Regular Customer	Y	Y	-	-	-	-
Silver Customer	-	-	Y	Y	-	-
Gold Customer	-	-	-	-	Y	Y
Special Discount	N	Y	N	Y	N	Y
No discount	✓					
Apply 8% discount			✓	✓		
Apply 15% discount					✓	✓
Apply additional x percent discount		✓		✓		✓

Exercise 2: Solution (Extended entry decision table)

Customer	Reg.	Reg.	Silver	Silver	Gold	Gold
Special Discount	N	Y	N	Y	N	Y
No discount	✓					
Apply 8% discount			✓	✓		
Apply 15% discount					✓	✓
Apply additional x percent discount		✓		✓		✓

Rule count

- Limited entry tables with N conditions have 2^N rules
- Don't care entries (*) and impossible situations (-) reduce the number of explicit rules by implying the existence of non-explicitly stated rules.
- Counting rules of a given decision table:
 - Rules with no (*) and (-) count as '1'
 - Roles with (*) and (-) count as 2^i where i is the number of occurrences of (*) and (-)

Why counting rules in a decision table?

Rule count acts as “Sanity check”:

- Less rules than combination rule count
 - Indicates missing rules
- More rules than combination rule count
 - Could indicate redundant rules
 - Could indicate inconsistent table

Rule count example 1

Homework done	Y	Y	Y	Y	N	N	N	N
Before 6pm	N	Y	N	Y	N	Y	N	Y
Friday	N	N	Y	Y	N	N	Y	Y
Rule count	1	1	1	1	1	1	1	1
Read book	✓	✓	✓					
See Movie				✓				
No Leisure					✓	✓	✓	✓

Rule count: $8 * 1 = 2^3$ ✓

Rule count example 2

Homework done	Y	Y	Y	N
Before 6pm	N	Y	Y	*
Friday	*	N	Y	*
Rule count	2	1	1	4
Read book	✓	✓		
See Movie			✓	
No Leisure				✓

Rule count: $2+1+1+4 = 2^3$ ✓

Rule count example 3

There is smoke	Y	Y	Y	N
You have given up smoking	Y	N	Y	*
Someone else is smoking	N	*	Y	*
Rule count	1	2	1	4
Push the fire alarm	1			
Leave the building	2			
Do nothing		1	1	1

Rule count: $1+2+1+4 = 2^3$ ✓

Rule count example 4

Regular Customer	Y	Y	-	-	-	-
Silver Customer	-	-	Y	Y	-	-
Gold Customer	-	-	-	-	Y	Y
Special Discount	N	Y	N	Y	N	Y
Rule count	4	4	4	4	4	4
No discount	✓					
Apply 8% discount			✓	✓		
Apply 15% discount					✓	✓
Apply additional x percent discount		✓		✓		✓

Rule count: $6 \cdot 4 = 24 \neq 2^3$

Where is the problem?

Rule count example 4 (unfolded)

Regular Customer	Y	Y	Y	Y	Y	Y	Y	Y																
Silver Customer									Y	Y	Y	Y	Y	Y	Y	Y								
Gold Customer																Y	Y	Y	Y	Y	Y	Y	Y	
Special Discount	N	N	N	N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y
Rule count	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
No discount																								
Apply 8% discount																								
Apply 15% discount																								
Apply additional x %																								
Impossible																								

Rule count example 4 (unfolded)

Regular Customer	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Silver Customer	Y	Y	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y	N	Y	N
Gold Customer	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
Special Discount	N	N	N	N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y
Rule count	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
No discount				✓																				
Apply 8% discount											✓					✓								
Apply 15% discount																			✓				✓	
Apply additional x%								✓								✓							✓	
Impossible	✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓	

Rule count example 4 (without redundancies)

Regular Customer	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Silver Customer	Y	Y	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y	N	Y	N
Gold Customer	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
Special Discount	N	N	N	N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y
Rule count	1	1	1	1	1	1	1	1			1	1			1	1				1				1
No discount				✓																				
Apply 8% discount											✓				✓									
Apply 15% discount																			✓					✓
Apply additional x%								✓							✓									✓
Impossible	✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓	

Introduction to Software Engineering

Rule count example 4 (with redundancies + missing rules)

Regular Customer	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	N	N
Silver Customer	Y	Y	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y	N	Y	N	N	N
Gold Customer	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	N
Special Discount	N	N	N	N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	N
Rule count	1	1	1	1	1	1	1	1			1	1			1	1				1				1	1	1
No discount				✓																						
Apply 8% discount												✓				✓										
Apply 15% discount																				✓					✓	
Apply additional x%								✓								✓									✓	
Impossible	✓	✓	✓		✓	✓	✓		Y	Y	✓		Y	Y	✓		Y	Y	Y		Y	Y	Y		✓	✓

Decision Tables Advantages

- A precise yet compact way to model complicated logic (while being able to check for completeness)
- Associate conditions with actions to perform
- Can associate many independent conditions with several actions in an elegant way
- Completeness can be verified
- Excellent tool for designing control logic and...**for deriving test cases**

Decision tables vs Test cases

- With a complete decision table we have a complete set of test cases.
- Each possible rule corresponds to a test case.
- Should we also write test cases for impossible rules?

Decision Table-Based Testing

- Used to analyze complex logical relations
- They are suitable for cases where combination of actions are taken under some conditions

Key Points: Decision Table-Based Testing Technique

- To identify test cases, we interpret conditions as input and actions as outputs then the rules are interpreted as test cases
- Steps:
 - Identifying conditions and values
 - Identifying maximum number of rules
 - Identifying actions
 - Encoding rules
 - Assigning the appropriate actions for rules
 - Simplify the rules (using “don’t care”)
 - Validate completeness
 - Derive test cases by assigning values to the variables

Exercise: Decision Table

- **Develop a *decision table*** for the following description of the insurance plan policy at the Everlasting life insurance company:

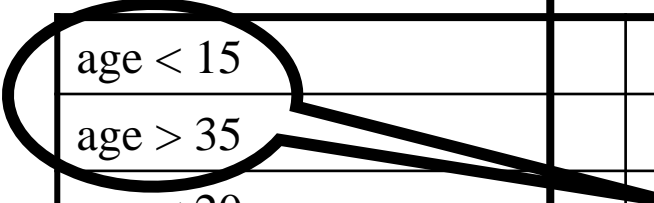
“Males between 15 and 35 years old are assigned to insurance plan A. Females between 20 and 40 years old are assigned to insurance plan B. All males under 15 and females under 20 are assigned to insurance plan C. All males over 35 and all females over 40 years old are assigned to insurance plan D. Clients without a driver's license are given a 10% rebate. Clients that have been insured for more than 20 years and currently part of insurance plan D are given a 25% rebate.”

Introduction to Software Engineering

	male								female							
age < 15																
age > 35																
age < 20																
age > 40																
insured > 20 years																
has driver's license																
Plan A																
Plan B																
Plan C																
Plan D																
10% rebate																
25% rebate																

- Here is an example of the possible decision table setup that you can have.
- Of course, you could have put “gender is male” on the left with the other conditions.

	male						female						
age < 15													
age > 35													
age < 20													
age > 40													
insured > 20 years													
has driver's license													
Plan A													
Plan B													
Plan C													
Plan D													
10% rebate													
25% rebate													



- These age ranges are only relevant for males.
- The next two are only relevant for females.
- In such cases use “*”, e.g., ...

Introduction to Software Engineering

	male						female					
age < 15	F											
age > 35	F											
age < 20	*											
age > 40	*											
insured > 20 years	*											
has driver's license	F											
Plan A	X											
Plan B												
Plan C												
Plan D												
10% rebate	X											
25% rebate												

“*” indicates “don't care” entries.