Software Testing & Quality Assurance

Decision Table-Based Testing

- Background and Definitions
- Related Techniques
- Decision Tables for the Triangle Program
- Decision Tables for the NextDate Program
- Test Case Design Guidelines
- Guidelines and Observations

Credits & Readings

- The material included in these slides are mostly adopted from the following books:
  - *Software Testing: A Craftsman's Approach*, by Paul Jorgensen, CRC PRESS, third edition, ISBN: 0-8493-7475-8
  - Cem Kaner, Jack Falk, Hung Q. Nguyen, *"Testing Computer Software"* Wiley (see also http://www.testingeducation.org/)
  - Paul Ammann and Jeff Offutt, *"Introduction to Software Testing"*, Cambridge University Press
  - Glen Myers, *"The Art of Software Testing"*

Decision Tables

- It's a tabular technique that has been used to represent and analyze complex logical relationships since the early 1960's
- Ideal for describing situations in which a number of combinations of actions are taken under varying sets of conditions

Decision Table Anatomy

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

- Left of the bold vertical line is called the *stub portion*
- The part to the right is called the *entry portion*
  - Columns in the entry portion are rules
  - Rules indicate what actions need to be taken
- The part above the horizontal bold line is called the *condition portion*
- The part below is called the *action portion*

- Tables that have binary conditions are called truth tables
- Tables where all the conditions are binary are called limited entry decision tables
  - If *n* conditions exist, there must be 2^n rules
- Tables that allow conditions to have several values are called extended entry decision tables
  - *If n* conditions exist with *k* number of values each then there must be k^n rules

Redundancy in Decision Tables

conditions	1-4	5	6	7	8	9
c1:	T	F	F	F	F	T
c2:	--	T	T	F	F	F
c3:	--	T	F	T	F	F
a1:	X	X	X	--	--	X
a2:	--	X	X	X	--	--
a3:	X	--	X	X	X	X

Rule 9 is identical to Rule 4 (T, F, F)

•Since the action entries for rules 4 and 9 are identical, there is no ambiguity, just redundancy.

Inconsistency in Decision Tables

conditions	1-4	5	6	7	8	9
c1:	T	F	F	F	F	T
c2:	--	T	T	F	F	F
c3:	--	T	F	T	F	F
a1:	X	X	X	--	--	--
a2:	--	X	X	X	--	X
a3:	X	--	X	X	X	--

Rule 9 is identical to Rule 4 (T, F, F)

- Since the action entries for rules 4 and 9 are different there is ambiguity.
- This table is inconsistent, and the inconsistency implies non-determinism.

A Decision Table for Printer Troubleshooting

Conditions	Printer does not print?	Y	Y	Y	Y	N	N	N	N
	A red light is flashing?	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized?	Y	N	Y	N	Y	N	Y	N
Actions	Heck the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

A Decision Table for a Billing System

When monthly bills are processed, it is noted whether the customer has made a payment. The payment may, or may not, equal what is owed. Also, it is possible that a customer owes money but makes no payment at all. For a customer that sends in money, a receipt must be sent. For a customer that owes money, a bill must be sent. Furthermore, a customer not owing money is to receive a special preferred-customer sales flyer. In other words, a decision has to be made concerning what actions to take and send a bill, a receipt, or a flyer based on certain conditions.

Payment made?	**T**		**F**	
Balance owed?	**T**	**F**	**T**	**F**
Send receipt	●	●		
Send bill	●		●	
Send flyer		●		●

- **k^n** combinations where
  - *n* is the number of conditions and
  - *k* is the number of possible values for each condition
- In this case, n=2 and k=2

Decision Table for a *Toy-Test* Activity

Make a decision table for a *toy-test* activity. Toy-test is an activity designed to test toys in different ways. If the toy is made of plastic, it should be stepped on. If it is made of medal, hit it with a hammer. If it has stickers on it, try to peel them off. If it is metal and painted, scratch the paint with a nail. If it is metal, has wheels, and has stickers, then submerge it in water for one hour.

Decision Table for Toy-test

Made of plastic	T								F							
Has stickers	T				F				T				F			
It is painted	T		F		T		F		T		F		T		F	
Has wheels	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Step on it	X	X	X	X	X	X	X	X								
Hit it									X	X	X	X	X	X	X	X
Peel stickers	X	X	X	X					X	X	X	X				
Scratch paint									X	X			X	X		
Submerge									X		X					

Decision Table for Car Insurance

Create a decision table to assign risk categories and charges to applicants for car insurance. Use the following rules. If the applicant is under 21, apply a surcharge. If the applicant is male and under 26 and married, or male and over 26, assign him to risk category B. If the applicant is a single male under 26 or a female under 21, assign the applicant to risk category C. All other applicants are assigned to risk category A.

Decision Table for Risk Categories

Age	<21				21-25				>26			
Gender	M		F		M		F		M		F	
Married	T	F	T	F	T	F	T	F	T	F	T	F
Surcharge	X	X	X	X								
Risk category A							X	X			X	X
Risk category B	X				X				X	X		
Risk category C		X	X	X		X						

Identifying Test Cases Using Decision Tables

We interpret

- *Conditions* as inputs
  - Conditions refer to equivalence classes of inputs
- *Actions* as outputs
  - Actions refer to functional portions of the item being tested
- *Rules* to test cases
- Since a decision table can be mechanically forced to be complete, it produces a comprehensive set of test cases

Techniques

- Adding an action to show when a rule is logically impossible
  - Sometimes conditions in the decision table refer to mutually exclusive possibilities
- The choice of conditions can greatly expand the size of a decision table

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

A Decision Table for the Triangle

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			

- Conditions C1-C6 represent equivalence input classes
- Actions A1-A5 represent output functionality of the program
- Rules portion represent possible test cases

- Notice the *don't care* entries
  - For instance, if the integers a, b and c don't constitute a triangle (C1-C3), we don't even care about possible equalities (C4-C6)
- Also, notice the *impossible rule* usage
  - For instance, if a=b and b=c then a=c must be true

Derived Test Cases for Triangle

Case ID	a	b	c	Expected Output
DT1	4	1	2	Not a Triangle
DT2	1	4	2	Not a Triangle
DT3	1	2	4	Not a Triangle
DT4	5	5	5	Equilateral
DT5	?	?	?	Impossible
DT6	?	?	?	Impossible
DT7	2	2	3	Isosceles
DT8	?	?	?	Impossible
DT9	2	3	2	Isosceles
DT10	3	2	2	Isosceles
DT11	3	4	5	Scalene

- From the previous decision table we derive 11 test cases
- Each test case corresponds to an action taken (denoted by X in the decision table)
  - 3 impossible cases
  - 3 not valid triangle
  - 1 for an equilateral
  - 1 for scalene
  - 3 for isosceles

The *NextDate* Problem

- The NextDate problem illustrates the problem of dependencies in the input domain
- Decision tables can highlight such dependencies
- Decision tables use the notion of "impossible action" to denote impossible combinations of conditions
- In NextDate, the impossible dates can be clearly marked as a separate action

We will try 3 refinement techniques

- Technique 1: focus is on identifying impossible combinations of inputs
- Technique 2: focuses on the leap year aspect
- Technique 3: focuses on specific days and months and clears up end-of-year considerations

Equivalence Classes

M1= {month | month has 30 days}
M2= {month | month has 31 days}
M3= {month | month is February}

D1= {day | $1 \leq day \leq 28$}
D2= {day | day = 29}
D3= {day | day = 30}
D4= {day | day=31}

Y1= {year | year = 2000}
Y2= {year | year is a leap year}
Y3= {year | year is a common year}

Partial Decision Table for *NextDate*

(Technique 1)

C1: month in M1?	T	T	T	T	T	T	T	T	T	T	T	T
C2: month in M2?												
C3: month in M3?												
C4: day in D1?	T	T	T									
C5: day in D2?				T	T	T						
C6: day in D3?							T	T	T			
C7: day in D4?										T	T	T
C8: year in Y1?	T			T			T			T		
C9: year in Y2?		T			T			T			T	
C10: year in Y3?			T			T			T			T
A1: Impossible										X	X	X
A2: Next Date	X	X	X	X	X	X	X	X	X			

- The goal is to show that many of these rules will be impossible
- Using the equivalence classes in the previous slide we can derive $2^{10}=1024$ rules
- To see why these rules are impossible, we can expand our actions to include:
  - A1: Too many days in a month
  - A2: Cannot happen in a non-leap year
  - A3: Compute the next date

Partial Decision Table for *NextDate*

(Technique 2--Part I)

C1: month in	M1	M1	M1	M1	M2	M2	M2	M2
C2: day in	D1	D2	D3	D4	D1	D2	D3	D4
C3: year in	-	-	-	-	-	-	-	-
A1: Impossible				X				
A2: Increment day	X	X			X	X	X	
A3: Reset day			X					X
A4: Increment month			X					?
A5: Reset month								?
A6: Increment year								?

Rest of table on next slide…

- The goal is to ensure that equivalence classes form a true partition of the input domain (i.e. disjoint subsets)
- We achieve that by creating an extended entry (rule) decision table
- We will consider only five possible manipulations needed to produce the next date:
  - Increment the day
  - Reset the day
  - Increment the month
  - Reset the month
  - Increment the year

- Using the cross product of equivalence classes M1-M3 X D1-D4 X Y1-Y3 we can derive 1+2+3+5+…+10 = 55 rules

Partial Decision Table for *NextDate*

(Technique 2--Part II)

C1: month in	M3	M3	M3	M3	M3	M3	M3	M3
C2: day in	D1	D1	D1	D2	D2	D2	D3	D3
C3: year in	Y1	Y2	Y3	Y1	Y2	Y3	-	-
A1: Impossible				X		X	X	X
A2: Increment day		X						
A3: Reset day	X		X		X			
A4: Increment month	X		X		X			
A5: Reset month								
A6: Increment year								

Partial Decision Table for *NextDate*

(Technique 3--Part I)

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	-	-	-	-	-	-	-	-	-	-
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										

Rest of table on next slide…

Updated Equivalence Classes

M1= {month | month has 30 days}
M2= {month | month has 31 days}
M3= {month | month is December}
M4= {month | month is February}

D1= {day | 1 ≤ day ≤ 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}

- We consider a third set of equivalence classes in order to handle end-of-year issues
- Also, we focus on days and months
- We give no special attention to the year 2000

Partial Decision Table for *NextDate*

(Technique 3--Part II)

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	-	-
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							

Commission Problem

- Test Cases for the Commission Problem
  - Not well served by a decision table
    - Very little decisional logic is used
    - There are no impossible rules that will occur in a decision table in which conditions correspond to the equivalence classes

Test Case Design

- To identify test cases with decision tables, we interpret conditions 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 being tested
- The rules are then interpreted as test cases

Applicability

- The specification is given or can be converted to a decision table
- The order in which the predicates are evaluated does not affect the interpretation of the rules or resulting action
- The order of rule evaluation has no effect on resulting action
- Once a rule is satisfied and the action selected, no other rule need be examined
- The order of executing actions in a satisfied rule is of no consequence
- The restrictions do not in reality eliminate many potential applications
  - In most applications, the order in which the predicates are evaluated is immaterial
  - Some specific ordering may be more efficient than some other but in general the ordering is not inherent in the program's logic

Decision Tables - Issues

- Before deriving test cases, ensure that
  - The rules are complete
    - Every combination of predicate truth values is explicit in the decision table
  - The rules are consistent
    - Every combination of predicate truth values results in only one action or set of actions

Guidelines and Observations

- Decision Table testing is most appropriate for programs where
  - There is a lot of decision making
  - There are important logical relationships among input variables
  - There are calculations involving subsets of input variables
  - There are cause and effect relationships between input and output
  - There is complex computation logic (high cyclomatic complexity)
- Decision tables do not scale up very well
  - May need to
    - Use extended entry decision tables
    - Algebraically simplify tables
- Decision tables can be iteratively refined
  - The first attempt may be far from satisfactory