Software Testing

Tecniche di testing funzionale
(black-box)

1

1

Testing Funzionale

- La definizione del casi di test e dell'oracolo è fondata sulla base della sola conoscenza dei requisiti specificati del sistema e dei suoi componenti
- Attività principale di generazione di casi di test …
- E' scalabile (usabile per i diversi livelli di testing)
- Non può rilevare difetti che dipendono dal codice
- Molte tecniche usabili sia per software tradizionale che OO

2

2

Tecniche di testing funzionale

- ***Funzionalità Esterne***: funzionalità visibili all'utente e definite dai requisiti e dalle specifiche
- ***Funzionalità Interne***: funzionalità non visibili all'utente e definite dal progetto di alto e basso livello
- Metodi
  - ✓ Metodi basati su specifiche formali (specifiche algebriche, macchine a stati finiti, statecharts)
  - ✓ Metodi basate su specifiche semi-formali (es. UML)
  - ✓ Metodi basati su specifiche informali

3

3

Criteri di copertura per testing funzionale

- Eseguire almeno una volta ogni funzionalità
- Per ogni funzionalità effettuare un numero di esecuzioni dedotte dai dati di ingresso e di uscita, da precondizioni e postcondizioni
  - ✓ definito il dominio dei dati di I/O effettuare test case ottenuti selezionando
    - valori in posizione centrale
    - valori ai bordi
    - valori "speciali"
  - ✓ precondizioni e postcondizioni per test case
    - positive
    - negative (invalid input data, invalid output data)
    - neutrali
- … suddivisione dei test case in classi di equivalenza

4

4

Suddivisione in classi di equivalenza

- Il dominio dei dati di ingresso è partizionato in classi di casi di test in modo tale che, se il programma è corretto per un caso di test, si possa dedurre ragionevolmente che è corretto per ogni caso di test in quella classe
- Una classe di equivalenza rappresenta un insieme di stati validi o non validi per una condizione sulle variabili d'ingresso

5

5

Classi di Equivalenza: Criteri di Copertura Deboli (Weak) e Forti (Strong)

- Tre variabili di input dei domini: A, B, C
- Partizioni dei domini
  - ✓ $A = A1 \cup A2 \cup A3$ dove $a_n \in An$
  - ✓ $B = B1 \cup B2 \cup B3 \cup B4$ dove $b_n \in Bn$
  - ✓ $C = C1 \cup C2$ dove $c_n \in Cn$
- Criterio debole (Weak Equivalence Class Testing - WECT): scegli un valore per una variabile da ogni classe
- Criterio forte (Strong Equivalence Class Testing – SECT): verifica tuttle le interazioni tra le classi (basato sul prodotto cartesiano dei sottoinsiemi delle partizioni $A \times B \times C$

6

6

WECT Test Cases

Test Case	a	b	c
WE1	a1	b1	c1
WE2	a2	b2	c2
WE3	a3	b3	c1
WE4	a1	b4	c2

Quasi partitioning …

	B1	B2	B3	B4
A1	(a1, b1, c1)			(a1, b4, c2)
A2		(a2, b2, c2)		
A3			(a3, b3, c1)	

7

SECT Test Cases

Test Case	a	b	c
SE1	a1	b1	c1
SE2	a1	b1	c2
SE3	a1	b2	c1
SE4	a1	b2	c2
SE5	a1	b3	c1
SE6	a1	b3	c2
SE7	a1	b4	c1
SE8	a1	b4	c2
SE9	a2	b1	c1
SE10	a2	b1	c2
SE11	a2	b2	c1
SE12	a2	b2	c2
SE13	a2	b3	c1
SE14	a2	b3	c2
SE15	a2	b4	c1
SE16	a2	b4	c2
SE17	a3	b1	c1
SE18	a3	b1	c2
SE19	a3	b2	c1
SE20	a3	b2	c2
SE21	a3	b3	c1
SE22	a3	b3	c2
SE23	a3	b4	c1
SE24	a3	b4	c2

Total partitioning …

	B1	B2	B3	B4
A1	(a1, b1, c1)	(a1, b2, c1)	(a1, b3, c1)	(a1, b4, c1)
A2	(a2, b1, c1)	(a2, b2, c1)	(a2, b3, c1)	(a2, b4, c1)
A3	(a3, b1, c1)	(a3, b2, c1)	(a3, b3, c1)	(a3, b4, c1)

8

Esempio: `NextDate`

- `NextDate` è una funzione con tre variabili: `month, day, year`. Restituisce la data del giorno successivo alla data di input tra gli anni 1840 e 2040
- Specifica sommaria: se non è l'ultimo giorno del mese, la funzione incrementa semplicemente il giorno. Alla fine del mese il prossimo giorno è 1 e il mese è incrementato. Alla fine dell'anno, giorno e mese diventano 1 e l'anno è incrementato. Considerare il fatto che il numero di giorni del mese varia con il mese e l'anno bisestile

9

9

NextDate: Classi di Equivalenza

- M1 = { mese: il mese ha 30 giorni}
- M2 = {mese: il mese ha 31 giorni}
- M3 = {mese: il mese è Febbraio}
- D1 = {giorno: 1<= giorno <= 28}
- D2 = {giorno : giorno = 29}
- D3 = {giorno : giorno = 30}
- D4 = {giorno : giorno = 31}
- Y1 = {anno: anno = 1900}
- Y2 = {anno: ((anno != 1900) AND (anno mod 4 = 0)}
- Y3 = {anno: anno mod 4 != 0}

10

10

NextDate Test Cases (1)

- WECT: 4 test cases

```
Case ID: Month, Day, Year, Output
WE1: 6, 14, 1900, 6/15/1900
WE2: 7, 29, 1912, 7/30/1912
WE3: 2, 30, 1913, Invalid Input
WE4: 6, 31, 1900, Invalid Input
```

- SECT: 36 test cases >> WECT

11

11

NextDate: SECT test cases

Case ID	Month	Day	Year	Expected Output
SE1	6	14	1900	6/15/1900
SE2	6	14	1912	6/15/1912
SE3	6	14	1913	6/15/1913
SE4	6	29	1900	6/30/1900
SE5	6	29	1912	6/30/1912
SE6	6	29	1913	6/30/1913
SE7	6	30	1900	7/1/1900
SE8	6	30	1912	7/1/1912
SE9	6	30	1913	7/1/1913
SE10	6	31	1900	ERROR
SE11	6	31	1912	ERROR
SE12	6	31	1913	ERROR
SE13	7	14	1900	7/15/1900
SE14	7	14	1912	7/15/1912
SE15	7	14	1913	7/15/1913
SE16	7	29	1900	7/30/1900
SE17	7	29	1912	7/30/1912
SE18	7	29	1913	7/30/1913
SE19	7	30	1900	7/31/1900
SE20	7	30	1912	7/31/1912
SE21	7	30	1913	7/31/1913
SE22	7	31	1900	8/1/1900
SE23	7	31	1912	8/1/1912
SE24	7	31	1913	8/1/1913
SE25	2	14	1900	2/15/1900
SE26	2	14	1912	2/15/1912
SE27	2	14	1913	2/15/1913
SE28	2	29	1900	ERROR
SE29	2	29	1912	3/1/1912
SE30	2	29	1913	ERROR
SE31	2	30	1900	ERROR
SE32	2	30	1912	ERROR
SE33	2	30	1913	ERROR
SE34	2	31	1900	ERROR
SE35	2	31	1912	ERROR
SE36	2	31	1913	ERROR

12

12

Discussione

- Se le condizioni di errore hanno alta priorità, bisogna estendere il criterio forte (SECT) includendo anche classi non valide
  - ✓ Test di robustezza …
- Il metodo delle classi di equivalenza è appropriato quando i dati di input sono definiti in termini di intervalli e insiemi di valori discreti
- Il criterio forte (SECT) assume che le variabili sono indipendenti – le dipendenze generano "error" test cases
  - ✓ E' possibile che ce ne siano molti …
  - ✓ Esistono metodi che analizzano le dipendenze tra le variabili e riducono i casi di test

13

13

Classi non valide e selezione delle classi di equivalenza

- Se la condizione sulle variabili d'ingresso specifica:
  - ✓ intervallo di valori
    - » almeno una classe valida per valori interni all'intervallo, una non valida per valori inferiori al minimo, e una non valida per valori superiori al massimo
  - ✓ elemento di un insieme discreto (enumerazione)
    - » una classe valida per ogni elemento dell'insieme, una non valida per un elemento non appartenente
    - » include il caso di valori specifici
    - » include il caso di valori booleani

14

14

Selezione dei casi di test dalle classi di equivalenza (WECT)

- Ogni classe di equivalenza deve essere coperta da almeno un caso di test
  - ✓Ciascun caso di test per le classi valide deve comprendere il maggior numero di classi valide ancora scoperte
  - ✓Un caso di test per ogni classe non valida

15

15

Intervallo di valori: esempio

Condizioni d'ingresso:

» Il codice cliente può avere un valore compreso tra 1 e 999

Classi di equivalenza:

» Valida

$CE_1 : 1 \leq COD \leq 999$

» Non valide

$CE_2 : COD < 1$

$CE_3 : COD > 999$

16

16

Insieme di valori: esempio

Condizioni di ingresso:

Il tipo di cliente può essere : *Privilegiato, Normale o Potenziale.*

Classi di equivalenza

Valide

CE_1: TIPO CL. = PRIVILEGIATO

CE_2: TIPO CL. = NORMALE

CE_3: TIPO CL. = POTENZIALE

- Non valida

  CE_4: TIPO CL. = DIFFICILE

17

17

Valori specifici (condizioni vincolanti): esempio

Condizione di ingresso

Il tipo di prodotto deve essere uguale a fabbricato

Classi di equivalenza

- Valida

  CE_1: TIPO PROD. = *FABBRICATO*

- Non valida

  CE_2: TIPO PROD. = *ACQUISTATO*

18

18

Progettazione dei casi di test (1)

CLASSI DI EQUIVALENZA VALIDE

Le classi di equivalenza individuate devono essere utilizzate per identificare casi di test che:

- Minimizzino il numero complessivo di test
- Risultino significativi (affidabili)

Si devono individuare tanti casi di test da coprire tutte le classi di equivalenza valide, con il vincolo che ciascun caso di test comprenda il maggior numero possibile di classi valide ancora scoperte.

19

19

Progettazione dei casi di test (2)

CLASSI DI EQUIVALENZA NON VALIDE

Si devono individuare tanti casi di test da coprire tutte le classi di equivalenza non valide, con il vincolo che ciascun caso di test copra una ed una sola delle classi non valide

20

20

Progettazione dei casi di test: esempio

CONDIZIONI ESTERNE	CLASSI DI EQUIVALENZA			
	#CE	Valide	#CE	Non valide
Valore del Cod. cli. tra 001 e 999	CE_1	001 ≤cod. cli ≤ 999	CE_2 CE_3	Cod.cli< 001 Coc.cli> 999
Tipo di cliente: Privilegiato, Normale, Potenziale	CE_4 CE_5 CE_6	Privilegiato Normale Potenziale	CE_7	Difficile
Il tipo deve essere *Fabbricato*	CE_8	*Fabbricato*	CE_9	*Acquistato*

Progettazione dei casi di test: esempio

I casi di test e le classi coperte sono:

Test case	TC_1	TC_2	TC_3
Cod. cli	005	005	005
Tipo cliente	Privilegiato	Normale	Potenziale
Tipo prodotto	Fabbricato	Fabbricato	Fabbricato
Classi coperte	CE_1, CE_4, CE_8	CE_1, CE_5, CE_8	CE_1, CE_6, CE_8

Progettazione dei casi di test: esempio

I casi di test e le classi coperte sono:

Test case	TC_4	TC_5	TC_6	TC_7
Cod. cli	000	1000	008	008
Tipo cliente	Normale	Normale	Difficile	Potenziale
Tipo prodotto	Fabbricato	Fabbricato	Fabbricato	Acquistato
Classi coperte	CE_2, CE_5, CE_8	CE_3, CE_5, CE_8	CE_1, CE_7, CE_8	CE_1, CE_6, CE_9

23

23

Testing dei valori limite (boundary values)

- Il metodo delle classi di equivalenza partiziona il dominio di ingresso assumendo che il comportamento del programma su input della stessa classe è simile
- Tipici errori di programmazione capitano al limite tra classi diverse
- Il testing dei valori limiti si focalizza su questo aspetto
- E' più semplice e serve a complementare la tecnica precedente

24

24

Analisi dei valori limite

- Consideriamo una funzione F, con due variabili x1 e x2
- Limiti (possibilmente non definiti): a <= x1 <= b, c <= x2 <= d
- In alcuni linguaggi di programmazione con strong typing, è possibile specificare tali intervalli
- Il metodo si focalizza sui limiti dello spazio di input per individuare i casi di test
- Il razionale (supportato da studi empirici) è che gli errori tendono a capitare vicino ai valori limite delle variabili di input

25

25

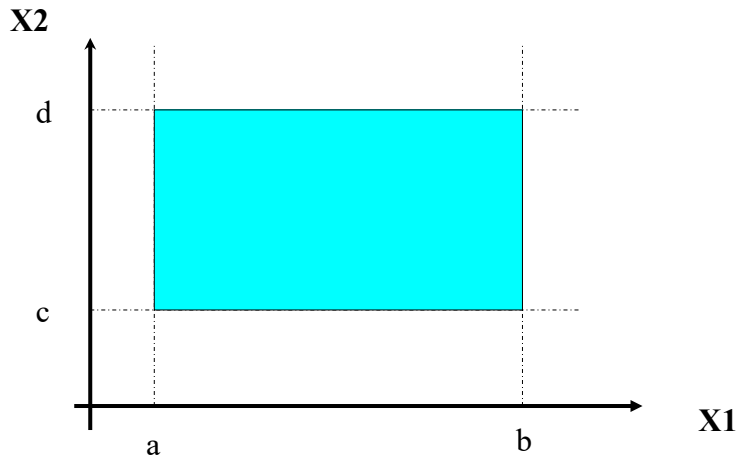
Idee di base

- Verificare I valori della variabile di input al minimo, immediatamente sopra il minimo, un valore intermedio (nominale), immediatamente sotto il massimo e al massimo
- Convenzione: min, min+, nom, max-, max
- Mantenere tutti i valori delle variabili, eccetto una, al loro valore nominale, mentre l'altra assume I valori estremi

26

26

Dominio di input della Funzione F

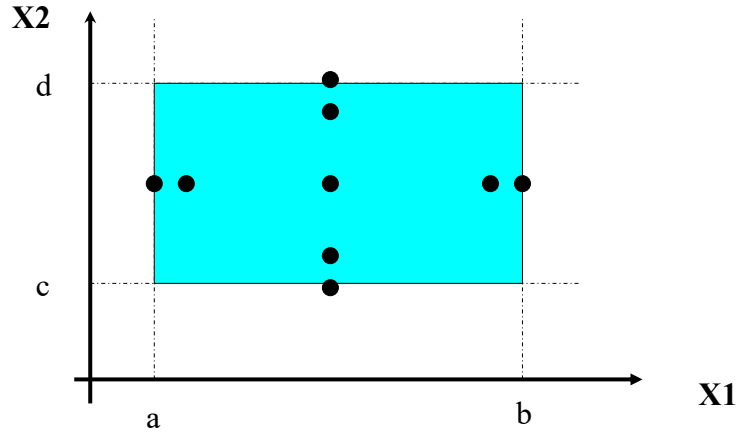


27

27

Boundary Analysis Test Cases

- Test set ={<$x1_{nom}$, $x2_{min}$>, <$x1_{nom}$, $x2_{min+}$>, <$x1_{nom}$, $x2_{nom}$>, <$x1_{nom}$, $x2_{max-}$>, <$x1_{nom}$, $x2_{max}$>, <$x1_{min}$, $x2_{nom}$,>, <$x1_{min+}$, $x2_{nom}$,>, <$x1_{max-}$, $x2_{nom}$>, <$x1_{max}$, $x2_{nom}$>}



28

28

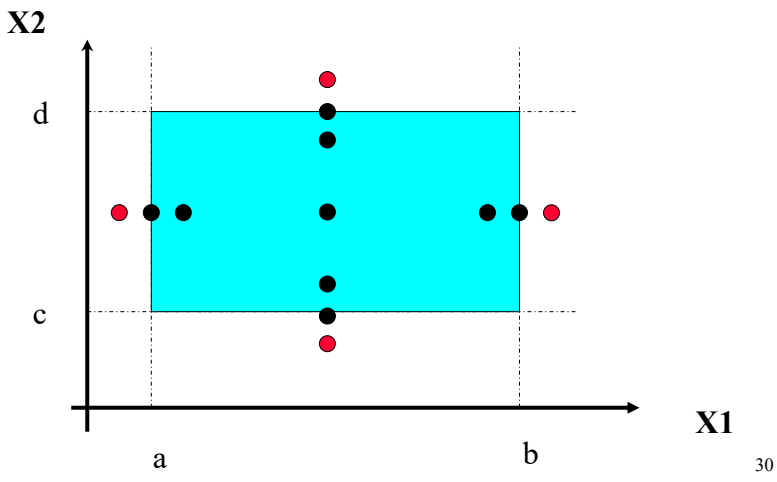
Caso Generale e Limitazioni

- Una funzione con n variabili richiede 4n + 1 casi di test
- Funziona bene con variabili che rappresentano quantità fisiche limitate
- Non considera la natura della funzione e il significato delle variabili
- Tecnica rudimentale che tende al test di robustezza

29

29

Test di Robustezza



30

30

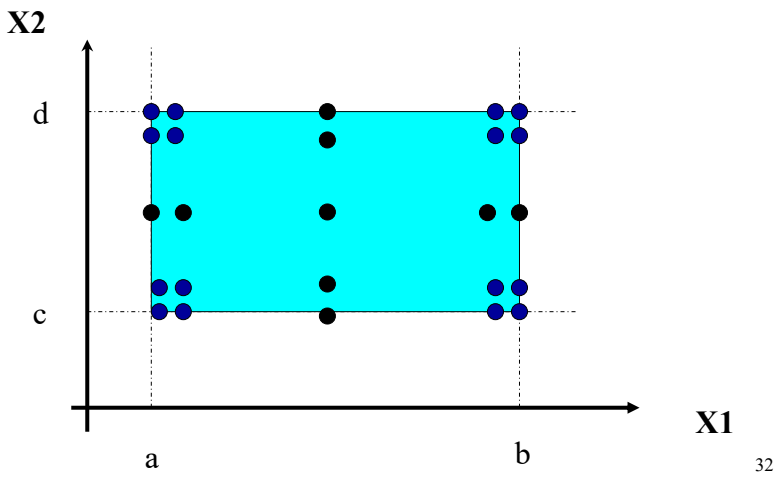
Worst Case Testing (WCT)

- La tecnica dei valori limite assume che la maggior parte delle failure sono originate da un solo fault
- Cosa succede quando più di una variabile ha un valore estremo ?
- L'idea viene dall'analisi dei circuiti in elettronica
- Prodotto cartesiano di {min, min+, nom, max-, max}
- Chiaramente più profonda dell'analisi dei valori limite, ma molto più costosa: 5^n test cases
- Buona strategia quando le variabili fisiche hanno numerose interazioni e quando le failure sono costose
- E in più: Robust Worst Case Testing

31

31

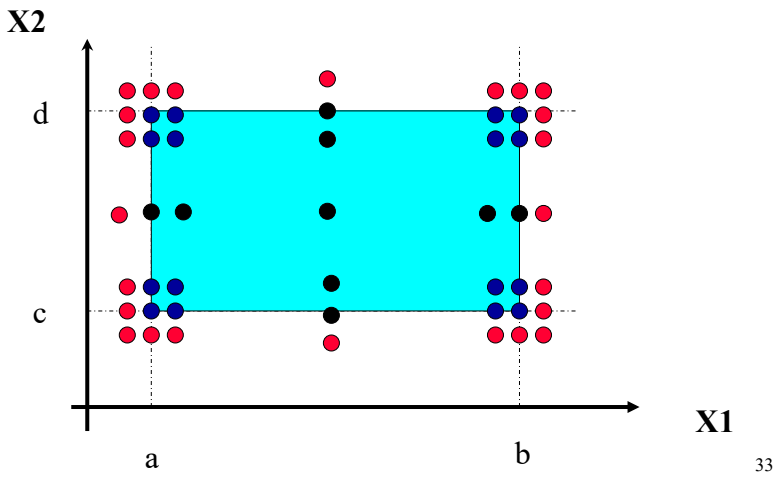
WCT per 2 variabili



32

32

Robust WCT per 2 variabili



33

33

Esercizio

- L'utente può chiamare la banca usando il proprio computer collegato via modem, digitare una **password** di 6 cifre, e digitare vari **comandi** che consentono di accedere alle varie funzioni bancarie (trasferimento fondi, estratto conto, saldo, fine sessione). L'utente non può **trasferire** meno di 100.000 lire e più di un milione per telefonata
- Selezionare i casi di test mediante partizione in classi di equivalenza con criterio di copertura debole

34

34

Metodi per ridurre i casi di prova

- Obiettivo: ridurre il numero (esponenziale) di combinazioni considerando solo i casi più significativi
  - ✓ partizione delle categorie
    - » basato sull' identificazione di categorie di input e relazioni di incompatibilità
  - ✓ tabelle delle decisioni
    - » basato sulla costruzione di tabelle per identificare i casi più significativi
  - ✓ grafo causa effetto
    - » basato sulla costruzione di grafi per ridurre il numero di combinazioni in base agli effetti delle diverse combinazioni
  - ✓ Pairwaise combinatorial testing

35

35

Partizione delle categorie: Passi (1)

- Il sistema è diviso in funzioni che possono essere testate indipendentemente
- Il metodo individua i *parametri* di ogni "funzione" e per ogni parametro individua *categorie* distinte
  - ✓ Oltre ai parametri, possono essere considerati anche *oggetti dell'ambiente*
  - ✓ Le categorie sono le principali proprietà o caratteristiche
- Le categorie sono ulteriormente suddivise in scelte allo stesso modo in cui si applica la partizione in classi di equivalenze (possibili valori)
  - ✓ normal values, boundary values, special values, error values

36

36

Partizione delle categorie: Passi (2)

- Vengono individuati i vincoli che esistono tra le scelte, ossie in che modo l'occorrenza di una scelta può influenzare l'esistenza di un'altra scelta
- Vengono generati *Test frames* che consistono di combinazioni valide di scelte nelle categorie
- I Test frames sono quindi convertiti in *test data*

37

37

Esempio (parametri, categorie e scelte)

- Una funzione che prende in ingresso un array di lunghezza variabile di qualunque tipo e restituisce l'array ordinato (in accordo a qualche criterio) e i valori massimo e minimo
- Categorie per il parametro array: dimensione dell'array, tipo degli elementi, massimo valore, minimo valore, posizioni dei valori massimo e minimo nell'array
- Scelte per la dimensione dell'array: array di dimensione 0, di dimensione 1 e di dimensione da 2 a n (dove n è la dimensione massima) e un tentativo con n+1 (classe non valida)
- Scelte per posizione: il valore massimo in prima posizione, posizione centrale e ultima posizione dell'array
- Etc.

38

38

Vincoli

- *Poprietà e Selettori* associati alle scelte

Categoria A
- SceltaA1 [property X, Z]
- SceltaA2 [property Y, Z]

Categoria B
- SceltaB1
- SceltaB2 [if X and Z]

Annotazioni speciali: [Error], [Single]

39

39

Example (borrowed from original article)

Command: find

Syntax: find <pattern> <filename>

Function: The find command is used to locate one or more instance of a given pattern in a text file. All lines in the file that contain the pattern are written to standard output. A line containing the pattern is written only once, regardless of the number of times the pattern occurs in it.

The pattern is any sequence of characters whose length does not exceed the maximum length of a line in the file. To include a blank in the pattern, the entire pattern must be enclosed in quotes (“). To include quotation mark in the pattern, two quotes in a row (“ “) must be used.

40

Examples:

find john myfile
display lines in the file **myfile** which contain **john**

find “john smith” in myfile
display lines in the file **myfile** which contain **john smith**

find “john”” smith” in myfile
display lines in the file **myfile** which contain **john” smith**

41

Parameters and environment objects

Parameters:

pattern
filename

Environment objects:

the content of the file *filename*

42

Test specification: categories and choices for the parameters

Pattern size:
- empty
- single character
- many characters
- longer than any line in the file

Quoting:
- pattern is quoted
- pattern is not quoted
- pattern is improperly quoted

Embedded blanks:
- no embedded blank
- one embedded blank
- several embedded blanks

Embedded quotes:
- no embedded quotes
- one embedded quotes
- several embedded quotes

File name:
- good file name
- no file with this name

43

Test specification: categories and choices for the parameters

Number of occurrence of pattern in file:
- none
- exactly one
- more than one

Pattern occurrences on target line:
- one
- more than one

44

Test Frame Example:

Pattern size: empty
Quoting: pattern is quoted
Embedded blanks: several embedded blanks
Embedded quotes: no embedded quote
File name: good file name
Number of occurrence of pattern in file: none
Pattern occurrence on target line: one

Is this a suitable test frame?

45

How many test frames?

Un-restricted test specification

1944
Total Tests frames

but many combinations are incompatible …

46

Property and selector expression (1)

Pattern size:

empty	[property Empty]
single character	[property NonEmpty]
many character	[property NonEmpty]
longer than any line in the file	[property NonEmpty]

Quoting:

pattern is quoted	[property quoted]
pattern is not quoted	[if NonEmpty]
pattern is improperly quoted	[if NonEmpty]

Embedded blanks:

no embedded blank	[if NonEmpty]
one embedded blank	[if NonEmpty and Quoted]
several embedded blanks	[if NonEmpty and Quoted]

To eliminate contradictory frame: *constraints* are indicated with ***properties*** that can be assigned to certain choices, and tested for by other choices.

When the generator encounters a choice with ***a selector expression***, it omits that choice from any partial test frame that does not already have the properties specified in the selector expression.

47

Property and selector expression (2)

Embedded quotes:

no embedded quotes	[if NonEmpty]
one embedded quotes	[if NonEmpty]
several embedded quotes	[if NonEmpty]

File name:

good file name
no file with this name

Number of occurrence of pattern in file:

none	[if NonEmpty]
exactly one	[if NonEmpty] [property Match]
more than one	[if NonEmpty] [property Match]

Pattern occurrences on target line:

one	[if Match]
more than one	[if Match]

48

How many test frames after constraints?

Restricted test specification

678
Total Tests frames

Can we do better?

- Many test frame are **redundant**; they test the same essential situation.
- They differ only in varying parameters that have no effect on the command's outcome.
- This occurs frequently when *some parameter or environment condition causes an error*.

- **[error]** tests are designed to test a particular feature which will cause an exception or other error state.
- A choice marked with **[error]** is not combined with choices in the other categories to create test frames.

49

Using Error property (1)

Pattern size:

empty	[property Empty]
single character	[property NonEmpty]
many character	[property NonEmpty]
longer than any line in the file	[error]

Quoting:

pattern is quoted	[property quoted]
pattern is not quoted	[if NonEmpty]
pattern is improperly quoted	[error]

Embedded blanks:

no embedded blank	[if NonEmpty]
one embedded blank	[if NonEmpty and Quoted]
several embedded blanks	[if NonEmpty and Quoted]

50

Using Error property (2)

Embedded quotes:

no embedded quotes	[if NonEmpty]
one embedded quotes	[if NonEmpty]
several embedded quotes	[if NonEmpty]

Total Tests frames: 125

File name:

good file name	
no file with this name	[error]

Environments:

Number of occurrence of pattern in file:

none	[if NonEmpty]
exactly one	[if NonEmpty] [property Match]
more than one	[if NonEmpty] [property Match]

Pattern occurrences on target line:

one	[if Match]
more than one	[if Match]

51

What else ?

- The annotation [single] is intended to describe special, unusual, or redundant conditions that do not have to be combined with all possible choices.
- As with [error] choices, the generator does not combine a [single] choice with any choices from other categories.

52

Using Single property

Embedded quotes:
- no embedded quotes [if NonEmpty]
- one embedded quotes [if NonEmpty]
- several embedded quotes [if NonEmpty] [single]

Total Tests frames: 40

File name:
- good file name
- no file with this name [error]

Number of occurrence of pattern in file:
- none [if NonEmpty] [single]
- exactly one [if NonEmpty] [property Match]
- more than one [if NonEmpty] [property Match]

Pattern occurrences on target line:
- one [if Match]
- more than one [if Match] [single]

53

Test Frame :

Test case 28 : (Key = 3.1.3.2.1.2.1.)
- **Pattern size** : many character
- **Quoting** : pattern is quoted
- **Embedded blanks** : several embedded blanks
- **Embedded quotes** : one embedded quote
- **File name** : good file name
- **Number of occurrence of pattern in file** : exactly none
- **Pattern occurrence on target line** : one

Command to set up the test:

```
copy/testing/sources/case_28 testfile
```

find command to perform the test:

find “has ”“ one quote” testfile

Instruction for checking the test :

the following line should be display:

```
This line has “ one quote on it
```

54

Altro Esempio

- Specifica
- Il programma chiede all'utente un intero positivo nell'intervallo 1-20 e quindi una stringa di caratteri di quella lunghezza. Il programma chiede all'utente un carattere e restituisce la prima posizione nella stringa in cui il carattere viene trovato o un messaggio che indica che il carattere non è presente nella stringa. L'utente ha l'opzione di cercare più caratteri.

Questo esempio mostrerà come sia facile fare confusione tra categorie e scelte

55

55

Parametri e Categorie

- Tre parametri: l'intero x (lunghezza), la stringa a e il carattere c
- Categorie per x: "in-range" (1-20) o "out-of-range"
- Categorie per a: lunghezza minima, massima, intermedia
- Categorie per c: il carattere appare all'inizio, al centro o alla fine della stringa, oppure non appare nella stringa

56

56

Scelte

- Intero x, out-of-range: 0, 21
- Intero x, in-range: 1, 2-19, 20
- Stringa a: 1, 2-19, 20
- Carattere c: prima posizione, ultima posizione, posizione centrale, non in stringa
- Note: a volte è possibile che ci sia solo una scelta per una categoria

57

57

Specifiche Formali dei Test

x:		
1)	0	[error]
2)	1	[property stringok, length1]
3)	2-19	[property stringok, midlength]
4)	20	[property stringok, length20]
5)	21	[error]
a:		
1)	Lunghezza 1	[if stringok and length1]
2)	Lunghezza 2-19	[if stringok and midlength]
3)	Lunghezza 20	[if stringok and length20]
c:		
1)	Prima posizione	[if stringok]
2)	Ultima posizione	[if stringok and not length1]
3)	Posizione centrale	[if stringok and not length1]
4)	Non in stringa	[if stringok]

58

58

Test Frames and Casi di test

x1	x = 0
x2a1c1	x = 1, a = ‘A’, c = ‘A’
x2a1c4	x = 1, a = ‘A’, c = ‘B’
x3a2c1	x = 7, a = ‘ABCDEFG’, c = ‘A’
x3a2c2	x = 7, a = ‘ABCDEFG’, c = ‘G’
x3a2c3	x = 7, a = ‘ABCDEFG’, c = ‘D’
x3a2c4	x = 7, a = ‘ABCDEFG’, c = ‘X’
x4a3c1	x = 20, a = ‘ABCDEFGHIJKLMNOPQRST’, c = ‘A’
x4a3c2	x = 20, a = ‘ABCDEFGHIJKLMNOPQRST’, c = ‘T’
x4a3c3	x = 20, a = ‘ABCDEFGHIJKLMNOPQRST’, c = ‘J’
x4a3c4	x = 20, a = ‘ABCDEFGHIJKLMNOPQRST’, c = ‘X’
x5	x = 21

59

59

Esempio: funzione *Check Configuration*

- Controlla la validità di una configurazione di un computer
- I parametri di check-configuration sono:
  - ✓ Model
  - ✓ Set of components

Esempio tratto da libro di Pezzé e Young

60

60

Parametro *model*

- A *model* identifies a specific product and determines a set of constraints on available components. Models are characterized by logical slots for components, which may or may not be implemented by physical slots on a bus. Slots may be required or optional. Required slots must be assigned with a suitable component to obtain a legal configuration, while optional slots may be left empty or filled depending on the customers' needs
- *Esempio: The required "slots" of the Chipmunk C20 laptop computer include a screen, a processor, a hard disk, memory, and an operating system. (Of these, only the hard disk and memory are implemented using actual hardware slots on a bus.) The optional slots include external storage devices such as a CD/DVD writer.*

61

61

Parametro *set of components*

- A set of *(slot, selection)* pairs, corresponding to the required and optional slots of the model. A *component* is a choice that can be varied within a model, and which is not designed to be replaced by the end user. Available components and a default for each slot is determined by the model. The special value *empty* is allowed (and may be the default selection) for optional slots. In addition to being compatible or incompatible with a particular model and slot, individual components may be compatible or incompatible with each other.
- *Esempio: The default configuration of the Chipmunk C20 includes 20 gigabytes of hard disk; 30 and 40 gigabyte disks are also available. (Since the hard disk is a required slot, empty is not an allowed choice.) The default operating system is RodentOS 3.2, personal edition, but RodentOS 3.2 mobile server edition may also be selected. The mobile server edition requires at least 30 gigabytes of hard disk.*

62

62

Step 1: Individuazione delle categorie

- Selezione delle categorie
  - ✓ no hard-and-fast rules
  - ✓ not a trivial task!
- Le categorie riflettono il giudizio del test designer
  - ✓ Riguardano le classi di valori che possono essere considerate in maniera diversa da una implementazione
- Una buona selezione richiede esperienza e conoscenza
  - ✓ del dominio applicativo e dell'architettura del prodotto.
  - ✓ Il test designer deve guardare sotto la superficie della specifica e individuare caratteristiche nascoste

63

63

Step1: Individuazione delle categorie

Parametro *Model*
- ✓ Model number
- ✓ Number of required slots for selected model (#SMRS)
- ✓ Number of optional slots for selected model (#SMOS)

Parametro *Components*
- ✓ Correspondence of selection with model slots
- ✓ Number of required components with selection ≠ empty
- ✓ Required component selection
- ✓ Number of optional components with selection ≠ empty
- ✓ Optional component selection

Oggetto d'ambiente: *Product database*
- ✓ Number of models in database (#DBM)
- ✓ Number of components in database (#DBC)

64

64

Step2: Individuazione delle scelte

- Individuazione di classi rappresentative di valori per ogni categoria
  - ✓ Ignorare le interazioni tra i valori di diverse categorie (considerate nel prossimo passo)
- Possono essere individuati applicando
  - ✓ Boundary value testing
    - » Selezionare valori estremi nella classe
    - » Selezionare valori fuori, ma vicini, alla classe
    - » Selezionare valori interni (non estremi) alla classe
  - ✓ Erroneous condition testing
    - » Selezionare valori al di fuori del dominio (classi non valide)

65

65

Step2 - Scelte: *Model*

Model number
- Malformed
- Not in database
- Valid

Number of required slots for selected model (#SMRS)
- 0
- 1
- Many

Number of optional slots for selected model (#SMOS)
- 0
- 1
- Many

66

66

Step2 - Scelte: *Components*

- Correspondence of selection with model slots
  - Omitted slots
  - Extra slots
  - Mismatched slots
  - Complete correspondence
- Number of required components with non empty selection
  - 0
  - < #SMRS
  - = #SMRS
- Required component selection
  - Some defaults
  - All valid
  - ≥ 1 incompatible with slots
  - ≥ 1 incompatible with another selection
  - ≥ 1 incompatible with model
  - ≥ 1 not in database
- Number of optional components with non empty selection
  - 0
  - < #SMOS
  - = #SMOS
- Optional component selection
  - Some defaults
  - All valid
  - ≥ 1 incompatible with slots
  - ≥ 1 incompatible with another selection
  - ≥ 1 incompatible with model
  - ≥ 1 not in database

67

67

Step2 - Scelte: *Database*

- Number of models in database (#DBM)
  - 0
  - 1
  - Many
- Number of components in database (#DBC)
  - 0
  - 1
  - Many

NB*:* 0 and 1 sono valori (speciali) insoliti. Possono causare comportamenti non anticipati da soli o in combinazione con particolari valori di altri parametri.

68

68

Step 3: Introduzione dei vincoli

- Una combinazione di valori per ogni categoria corrisponde ad una specifica di un caso di test
  - ✓Nell'esempio abbiamo 314.928 test cases
  - ✓Molti dei quali sono impossibili !
    - » Esempio
      zero slots e ***at least one incompatible slot***
- I vincoli si introducono per
  - ✓eliminare combinazioni impossibili
  - ✓ridurre la dimensione della test suite (se troppo grande)

69

69

Step 3: error constraint

[error] indica una classe di valori che
- ✓corrisponde ad un valore erroneo
- ✓deve essere provato una sola volta

Esempio
Model number: Malformed and Not in database

Classi di valori *error*
- ✓Non è necessario testare tutte le possibili combinazioni di errors
- ✓Un test è abbastanza (assumiamo che gestire un caso di errore by-passa il resto della logica del programma)

70

70

Esempio - Step 3: error constraint

Model number
- Malformed [error]
- Not in database [error]

Correspondence of selection with model slots
- Omitted slots [error]
- Extra slots [error]
- Mismatched slots [error]

Number of required comp. with non empty selection
- 0 [error]
- < #SMRS [error]

Required comp. selection
- ≥ 1 not in database [error]

Number of models in database (#DBM)
- 0 [error]

Number of components in database (#DBC)
- 0 [error]

Error constraints riducono la test suite da 314.928 a 2.711 test cases

71

71

Step 3: property constraints

constraint [property] [if-property] eliminano combinazioni non valide di valori (scelte)

[property] raggruppa i valori di un singolo parametro per individuare sottoinsiemi di valori con proprietà comuni

[if-property] limita le scelte di valori di una categoria che possono essere combinate con un particolare valore selezionato per una diversa categoria

Esempio: combina

Number of required comp. with non empty selection = number required slots [if RSMANY]

solo con

Number of required slots for selected model (#SMRS) = Many [property RSMANY]

72

72

Esempio - Step 3: property constraints

Number of required slots for selected model (#SMRS)

1	[property RSNE]
Many	[property RSNE] [property RSMANY]

Number of optional slots for selected model (#SMOS)

1	[property OSNE]
Many	[property OSNE] [property OSMANY]

Number of required comp. with non empty selection

0	[if RSNE] [error]
< #SMRS	[if RSNE] [error]
= #SMRS	[if RSMANY]

Number of optional comp. with non empty selection

< #SMOS	[if OSNE]
= #SMOS	[if OSMANY]

da 2.711 a 908 test cases

73

73

Step 3 (cont): single constraints

[single] indica una classe di valori che i test designer scelgono di testare solo una volta per ridurre il numero di casi di test

Esempio

value some default for required component selection and optional component selection may be tested only once despite not being an erroneous condition

Nota -

single e error hanno lo stesso effetto ma differiscono nel razionale. Lasciarli distinti è importante ai fini della documentazione e per il testing di regressione

74

74

Example - Step 3: single constraints

Number of required slots for selected model (#SMRS)

0 [single]

1 [property RSNE] [single]

Number of optional slots for selected model (#SMOS)

0 [single]

1 [single] [property OSNE]

Required component selection

Some default [single]

Optional component selection

Some default [single]

da 908 a 69 test cases

Number of models in database (#DBM)

1 [single]

Number of components in database (#DBC)

1 [single]

75

75

Check configuration – Summary

Parameter Model

- Model number
  - ✓ Malformed [error]
  - ✓ Not in database [error]
  - ✓ Valid
- Number of required slots for selected model (#SMRS)
  - ✓ 0 [single]
  - ✓ 1 [property RSNE] [single]
  - ✓ Many [property RSNE] [property RSMANY]
- Number of optional slots for selected model (#SMOS)
  - ✓ 0 [single]
  - ✓ 1 [property OSNE] [single]
  - ✓ Many [property OSNE] [property OSMANY]

Environment Product data base

- Number of models in database (#DBM)
  - ✓ 0 [error]
  - ✓ 1 [single]
  - ✓ Many
- Number of components in database (#DBC)
  - ✓ 0 [error]
  - ✓ 1 [single]
  - ✓ Many

Parameter Component

- Correspondence of selection with model slots
  - ✓ Omitted slots [error]
  - ✓ Extra slots [error]
  - ✓ Mismatched slots [error]
  - ✓ Complete correspondence
- # of required components (selection ≠ empty)
  - ✓ 0 [if RSNE] [error]
  - ✓ < < #SMRS [if RSNE] [error]
  - ✓ = < #SMRS [if RSMANY]
- Required component selection
  - ✓ Some defaults [single]
  - ✓ All valid
  - ✓ ≥ 1 incompatible with slots
  - ✓ ≥ 1 incompatible with another selection
  - ✓ ≥ 1 incompatible with model
  - ✓ ≥ 1 not in database [error]
- # of optional components (selection ≠ empty)
  - ✓ 0
  - ✓ < #SMOS [if OSNE]
  - ✓ = #SMOS [if OSMANY]
- Optional component selection
  - ✓ Some defaults [single]
  - ✓ All valid
  - ✓ ≥ 1 incompatible with slots
  - ✓ ≥ 1 incompatible with another selection
  - ✓ ≥ 1 incompatible with model
  - ✓ ≥ 1 not in database [error]

76

76

Conclusioni

- L'individuazione di parametri, condizioni di ambiente e categorie dipende fortemente dall'esperienza del tester
- Rende le decisioni di testing esplicite (e.g., vincoli) e aperte a revisioni
- Una volta completati i primo passo, la tecnica è semplice e può essere automatizzata
- Riducendo i casi di test, la tecnica è utile e rende il testing sistematico più praticabile

77

77

Tabelle di decisione: Motivazioni

- Aiuta ad esprimere *test requirements* in una forma direttamente comprensibile e usabile
- Supporta la generazione automatica o manuale di casi di test
- Una particolare risposta o un sottinsieme di risposte deve essere selezionato valutando molte condizioni corrispondenti
- Ideale per descrivere situazioni in cui un numero di combinazioni di azioni sono prese in corrispondenza di insiemi di condizioni variabili, come ad esempio nei sistemi di controllo

78

78

Tabelle di decisione: Struttura

- La sezione delle condizioni contiene *condizioni* e combinazioni di queste
- Una condizione esprime relazioni tra *variabili di decisione*
- La sezione delle azioni mostra le risposte che devono essere prodotte quando le corrispondenti combinazioni di condizioni sono vere
- Limiti: Le azioni risultanti sono determinate dai valori *correnti* delle variabili di decisione
- Le azioni sono *indipendenti* dall'ordine di input e dall'ordine in cui le condizioni sono valutate
- Le azioni possono apparire più di una volta, ma ogni combinazione di condizioni è unica.

79

79

Struttura della Tabella

Condizione	c1	True			False			
	c2	True		False	True		False	
	c3	T	F	–	T	F	–	
Azione	a1	X	X		X			
	a2	X				X		
	a3		X		X			
	a4			X			X	

80

80

Esempio di Tabella

c1: a, b, c triangle?	N	Y									
c2: a = b?	–		Y		N						
c3: a = c?	–			Y		N		Y		N	
c4: b = c?	–	Y	N	Y	N	Y	N	Y	N	Y	N
a1: not a triangle	X										
a2: Scalene											X
a3: Isosceles						X		X	X		
a4: Equilateral		X									
a5: Impossible			X	X			X				

Tabella di Verità

conditions											
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**			

Casi di Test

Case ID	a	b	c	Expected Output
TC1	4	1	2	Not a triangle
TC 2	1	4	2	Not a triangle
TC 3	1	2	4	Not a triangle
TC 4	5	5	5	Equilateral
TC 5	?	?	?	Impossible
TC 6	?	?	?	Impossible
TC 7	2	2	3	Isosceles
TC 8	?	?	?	Impossible
TC 9	2	3	2	Isosceles
TC 10	3	2	2	Isosceles
TC 11	3	4	5	Scalene

83

83

Condizioni di uso ideali

- Una tra molte risposte distinte deve essere selezionata in accordo a casi distinti delle variabili di input
- Questi casi possono essere modellati da espressioni booleane mutualmente esclusive sulle variabili di input
- La risposta che deve essere prodotta non dipende dall'ordine in cui le variabili di input sono definite o valutate
- La risposta non dipende da input o output precedenti

84

84

Tabelle di decisione: Scalabilità

- Per n condizioni, ci possono essere al più 2^n *varianti* (combinazioni uniche di condizioni e azioni)
- Ma per fortuna ci sono di solito molto meno varianti *esplicite* …
- Valori "Don't care" nelle tabelle di decisione aiutano a ridurre il numero di varianti
- "Don't care" possono corrispondere a diversi casi:
  - ✓ Gli input sono necessari ma non hanno effetto
  - ✓ Gli input possono essere omessi
  - ✓ Casi mutualmente esclusivi

85

85

Casi speciali

- "non può succedere" : riflette qualche assunzione che alcuni input sono mutualmente esclusivi, o che essi non possono essere prodotti nell'ambiente
- Una sorgente cronica di bugs, e.g. Ariane 5
- "non può succedere" succede a causa di errori di programmazione e effetti di cambiamenti inattesi
- La condizione "non so" riflette un modello incompleto, ad esempio dovuto a documentazione incompleta
- La maggior parte delle volte sono errori di specifica

86

86

Metodo dei Grafi Causa-Effetto (IEEE Std 982.2-1988)

- Una tecnica grafica che aiuta a derivare tabelle di decisione
- Mira a creare combinazioni "interessanti" di test data
- Individua cause (condizioni su input, stimoli) e effetti (output, cambiamenti di stato del sistema)
- Le cause devono essere stabilite in modo tale da essere o vere o false (espressioni booleane)
- Specifica esplicitamente vincoli (ambientali, esterni) su cause ed effetti
- Aiuta a selezionare i più significativi sottoinsiemi di combinazioni input-output e costruisce tabelle di decisione più piccole

87

87

Struttura di un grafo causa-effetto

- Associazione di un nodo ad ogni causa e ad ogni effetto
  - ✓ Nodi causa ed effetto piazzati su lati opposti di un foglio
- Interconnessione di nodi causa ed effetto e produzione di un grafo booleano
  - ✓ Una linea da una causa ad un effetto indica che la causa è una condizione necessaria per l'effetto
  - ✓ Una singola causa può essere necessaria per diversi effetti; un singolo effetto può avere molte cause necessarie
  - ✓ Se un effetto ha due o più cause, la relazione logica tra le cause è espressa inserendo operatori *and* e *or* logici tra le due linee
  - ✓ Una causa la cui negazione è necessaria per un effetto è espressa etichettando la linea con l'operatore *not* logico
- Possono essere usati nodi intermedi per semplificare il grafo e la sua costruzione

88

88

Esempio (1)

Un data base management system organizza i file nel db in modo che il nome di ciacuno di essi sia listato in un indice. L'indice è organizzato in 10 sezioni. Il programma da testare consente all'utente di immettere comandi per ottenere il display di una sezione dell'indice.

Requisiti:

Per ottenere il display di una delle 10 sezioni dell'indice, l'utente deve fornire in input un comando costituito da una lettera ed una cifra. La lettera deve essere "D" per Display o "L" per List e deve trovarsi in colonna 1. La cifra deve essere in (0,...,9) e deve trovarsi in colonna 2. A fronte di un comando corretto viene realizzato il display della sezione desiderata. Se il primo carattere è scorretto viene stampato il messaggio A "INVALID COMMAND". Se è scorretto il secondo carattere viene stampato il messaggio B "INVALID INDEX NUMBER".

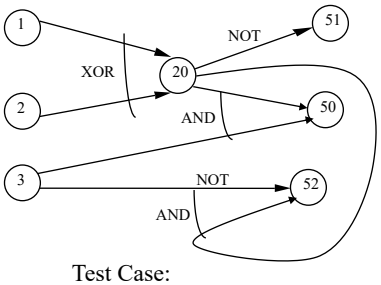
89

89

Esempio (2)

Cause
(1) Il carattere in colonna 1 è "D"
(2) Il carattere in colonna 1 è "L"
(3) Il carattere in colonna 2 è cifra

Effetti
(50) Display dell'indice della sezione
(51) Display Messaggio A
(52) Display Messaggio B



	effetti				
cause	**50a**	**50b**	**51**	**52a**	**52b.**
1	**1**	**0**	**0**	**1**	**0 .**
2	**0**	**1**	**0**	**0**	**1 .**
3	**1**	**1**	**?**	**0**	**0 .**

Test Case:
1: D5; 2:L4; 3:B2; 4: DA; 5:LB

90

90

Derivazione di Test Case

- produzione di una tabella delle decisioni
  - ✓ cause sulle righe
  - ✓ effetti colonne
- Per ogni effetto percorrere il grafo alla ricerca delle cause che rendono true l'effetto definendo conseguentemente una colonna della tabella
  - Possibili più colonne per uno stesso effetto
- Ad ogni colonna della tabella delle decisioni corrisponde un Test Case

91

91

Esempio: gestione clienti di un albergo

Dato un cliente, il tipo di stanza occupata ed il periodo di permanenza, il programma calcola il conto. Per le stanze esistono 3 tariffe (corrispondenti al tipo di stanza): singole, doppie e matrimoniali; i clienti che hanno soggiornato meno di 15 giorni pagano tariffa piena; i clienti che hanno soggiornato almeno 15 giorni hanno diritto ad uno sconto del 10%; i clienti di riguardo, infine, hanno diritto ad uno sconto del 30%. Lo sconto del 30% non permette di beneficiare dello sconto del 10% Le tariffe dipendono dalla stagione: sono considerati alta stagione i fine settimana, il periodo di Natale, dal 24 dicembre al 6 gennaio e il mese di agosto; media stagione i mesi di luglio e settembre, bassa stagione, infine, il resto dell' anno. Nel caso di conto inferiore a 1.500.000 lire il pagamento può essere fatto solo in contanti; nel caso di conto superiore o uguale a 1.500.000 lire il pagamento può essere effettuato in contanti e/o carta di credito; un cliente di riguardo può pagare con assegni o carta di credito e/o contanti..

92

92

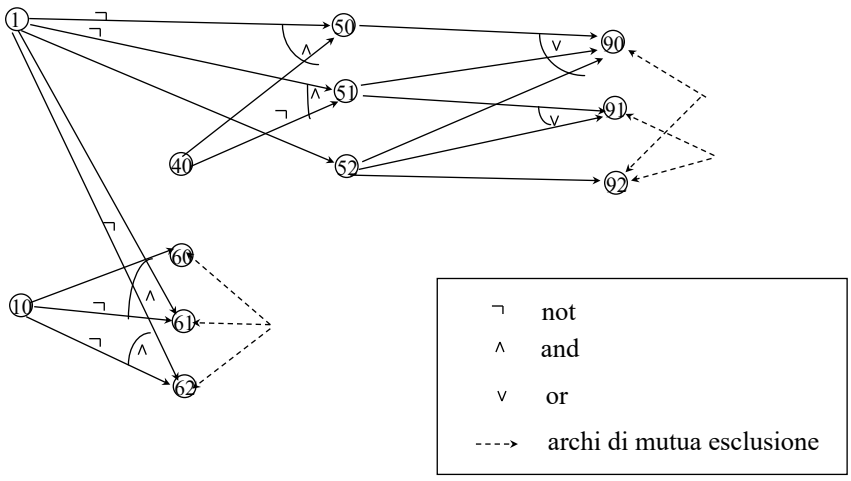
Esempio: cause e effetti

1. cliente di riguardo
10. permanenza inferiore a 15 giorni
20. stanza singola
21. stanza doppia
22. stanza matrimoniale
30. alta stagione
31. media stagione
32. bassa stagione
40. conto totale inferiore a 1.500.000 lire
60. nessuno sconto
61. sconto del 30%
62. sconto del 10%
90. pagamento in contanti
91. pagamento con carta di credito
92. pagamento con assegni

93

93

Esempio: grafo causa-effetto



94

94

Esempio: tabella

	1	2	3	4	5	6	7	8	9
1	0	0	1	0	1	1		1	0
10							1	0	0
40	1	0		0					
60							1		
61								1	
62									1
90	1	1	1						
91				1	1	1			
92									

0: indica che la condizione deve essere falsa
1: indica che la condizione deve essere vera

95

95

Pairwise Combinatorial Testing

- Pairwise (and n-way) combinatorial testing
  - ✓ Combina valori sistematicamente ma non esaustivamente
  - ✓ Razionale: Molte interazioni avvengono solo tra due (o comunque pochi) parametri o caratteristiche
- Pairwise combination (invece che esaustive)
  - ✓ Genera combinazioni che efficientemente comprono tutte le coppie (triple, …) di classi
  - ✓ Razionale: la maggior parte delle failure sono causate da valori singoli o combinazioni di pochi valori. Coprendo coppie (triple, …) riduce il numero di casi di test, ma consente di rivelare la maggior parte dei difetti

96

96

Esempio: Display Control

Display Mode	Language	Fonts	Color	Screen size
full-graphics	English	Minimal	Monochrome	Hand-held
text-only	French	Standard	Color-map	Laptop
limited-bandwidth	Spanish	Document-loaded	16-bit	Full-size
	Portuguese		True-color	

- Nessun vincolo riduce il numero totale di combinazioni
- 432 (3x4x3x4x3) test case

97

97

Pairwise combinations: 16 test cases

Language	Color	Display Mode	Fonts	Screen Size
English	Monochrome	full-graphics	Minimal	Hand-held
English	Color-map	text-only	Standard	Laptop
English	16-bit	limited-bandwidth	Document-loaded	Full-size
English	True-color	---	---	---
French	Monochrome	limited-bandwidth	Document-loaded	Laptop
French	Color-map	full-graphics	Document-loaded	Full-size
French	16-bit	text-only	Standard	Hand-held
French	True-color	limited-bandwidth	Minimal	---
Spanish	Monochrome	limited-bandwidth	Standard	Full-size
Spanish	Color-map	text-only	Minimal	---
Spanish	16-bit	full-graphics	---	Laptop
Spanish	True-color	text-only	Document-loaded	Hand-held
Portuguese	Monochrome	text-only	Document-loaded	---
Portuguese	Color-map	limited-bandwidth	---	Hand-held
Portuguese	16-bit	text-only	Minimal	Full-size
Portuguese	True-color	full-graphics	Standard	Laptop

98

E se si aggiungono i vincoli ?

Semplici vincoli come ad esempio:

Color *monochrome* non compatibile con Screen size *laptop* e *full size*

possono essere gestiti considerando tabelle separate

99

99

Esempio: Monochrome solo con hand-held

Display Mode	Language	Fonts	Color	Screen size
full-graphics	English	Minimal	Monochrome	Hand-held
text-only	French	Standard	Color-map	
limited-bandwidth	Spanish	Document-loaded	16-bit	
	Portuguese		True-color	

Display Mode	Language	Fonts	Color	Screen size
full-graphics	English	Minimal		
text-only	French	Standard	Color-map	Laptop
limited-bandwidth	Spanish	Document-loaded	16-bit	Full-size
	Portuguese		True-color	

100