EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 8a: Combinatorial Testing Introduction

Creative thinkers made here.

Lecture Summary

The learning goal for this week is to familiarize you with Combinatorial Testing

Do you know what combinatorial testing is?

Why its used?

How to use mutually orthogonal Latin squares to derive test cases?

Can you complete the workshop?

Introduction

EDITH COWAN UNIVERSITY

The behavior of software applications may be affected by many factors such as the operating environment, network connection, hardware platform and input parameters.

Windows 10

Mac OS X

X

Z

Y

Introduction

EDITH COWAN UNIVERSITY

Software applications may be affected by many factors such input parameters

Black box techniques help identify possible inputs

- equivalence partitioning (EP) and,
- boundary value analysis (BVA) can be applied for identifying possible values for each factor.

The goal here was to try generate a set of tests that adequately test the input domains for our software undertest

Introduction

Shortcomings of EP:

- Possibility of a large number of sub-domains in the partition
- Lacks guidelines on how to select inputs from the sub-domains
- **What about combinations of input?????**
- **Does not account for faults due to specific interactions amongst values of different input variables**

Shortcomings of BVA:

- Other interactions in the input domain remain untested.
- Doesn’t account for interaction of variables

Combinatorial Testing

We are going to look at techniques that generate small test configurations OR test sets even when the possible configurations or the input domain and the sub domains are large and complex.

EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 8b: Combinatorial Testing Terms

Creative thinkers made here.

More Terminology – Input Space

Your **input space** is your input domain.

The variables being assigned to your software under test.

The **input space** of a program P consists of k-tuples of values that could be input to P during execution.

Example:

- Consider program P that takes two integers $x>0$ and $y>0$ as inputs.
  - The **input space** of P is the set of all pairs of positive non-zero integers.

Input Space

- Another example..
- Date function that outputs the next date
- Day, month, year – Input space (days, months and years)

Day	Month	Year	Expected Output
15	6	1912	16/6/1912

Modeling: Input and configuration space (from slides, Mathur (2014)

The configuration space of P consists of all possible settings of the environment variables under which P could be used.

Example:

- This program is also intended to be executed under the Windows and the MacOS operating system, through the Chrome or Safari browsers, and must be able to print to a local or a networked printer.
  - The configuration space of P consists of triples (X, Y, Z) where X represents an operating system, Y a browser, and Z a local or a networked printer.

Many Combinations

Input or Configurations space leads to many combinations

- Will firefox on mac handle the same as firefox on windows?
  - Better test this!

Should I test the day and month input to a function in isolation or is there a relationship between these variables I should account for?

New Terminology

Levels,

Factors and,

Factor Combinations!

Factors

Consider a program P that takes n inputs corresponding to variables X_1, X_2, ..X_n. We refer to the inputs as factors. The inputs are also referred to as test parameters or as values.

You can think about factors as the input parameters to a program.

If an addition program required two inputs, x and y, these would be the factors

- This program would have 2 factors, being the x and y inputs.

Levels

Let us assume that each factor may be set at any one from a total of c_i, $1 \leq i \leq n$ values…..

The levels of your factors are essentially, your equivalence classes.

Each value assignable to a factor is known as a level. The notation |F| refers to the number of levels for factor F.

Pizza ordering program inputs

Factor	Levels
Size	Large | Small
Topping	Custom | Pre-set
Address	Valid | Invalid

Factor Combination

A set of values, one for each factor, is known as a factor combination.

Let's for a second assume that we have a pizza ordering application that takes in three inputs…

$Levels^{Factors}$ = 2 ^ 3 = 8 factor combinations

Or |Size| x |Toppings| x |Address| = 2 x 2 x 2

(L, C, V), (L, P, V), (L, C, I), (L, P, I),
(S, C, V), (S, P, V), (S, C, I), (S, P, I),

Factor	Levels
Size	Large | Small
Topping	Custom | Pre-set
Address	Valid | Invalid

Factors, Levels and Factor Combination

Example the second!

- Program P
  - two input variables X and Y
  - X from the set {a, b, c}
  - Y from the set {d, e, f}
- Thus we have 2 factors (X AND Y)
- 3 levels for each factor.
- This leads to a total of $3^2=9$ factor combinations (a, d), (a, e), (a, f), (b, d), (b, e), (b, f), (c, d), (c, e), and (c, f).

If each factor combination yields one test case, for many programs, the number of tests generated for exhaustive testing could be exorbitantly large.

If a program has 15 factors with 4 levels each, the total number of tests is 4^{15} OR 1,073,741,824 test cases.

Executing a billion tests might be impractical for many software applications

CSI3105: Software Testing

Module 8c: Combinatorial Testing and Exhaustive Testing

Creative thinkers made here.

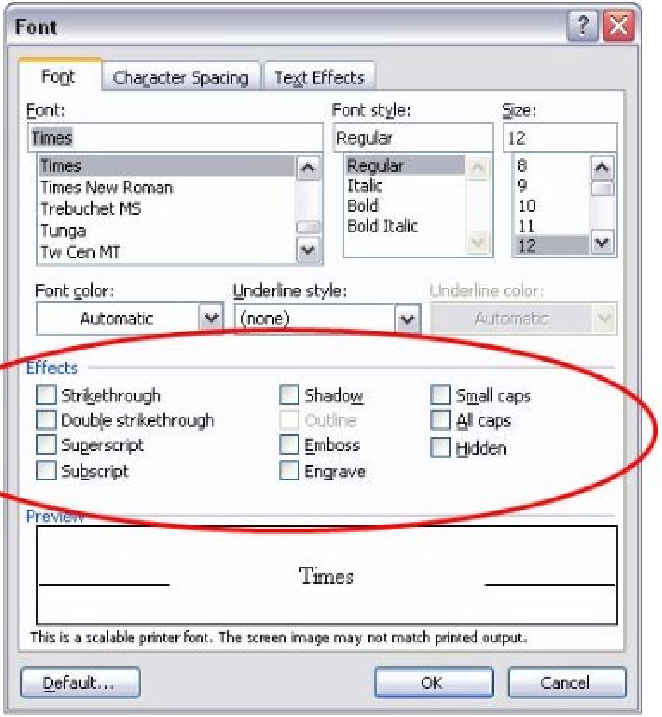
Common Problem

Executing a billion tests might be impractical for many software applications

Exhaustive testing is not possible!

Common Problem

https://csrc.nist.gov/CSRC/media/Presentations/Combinatorial-Methods-for-System-and-Softw



- Microsoft word has ten text effects
- Check button (on/off)
- All combinations 2 ^ 10 = 1024 tests

Common Problem

https://csrc.nist.gov/CSRC/media/Presentations/Combinatorial-Methods-for-System-and-Software-Test/images-media/4_software-testing_kuhn.pdf

- There are $\binom{10}{3}$ = 120 3-way interactions.
- Each triple has 2^3 = 8 settings: 000, 001, 010, 011, ..
- 120 x 8 = 960 combinations
- Each test exercises many triples:

0 1 1 0 0 0 0 1 1 0

Common Problem

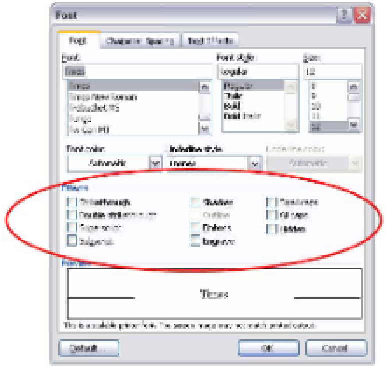
https://csrc.nist.gov/CSRC/media/Presentations/Combinatorial-Methods-for-System-and-Software-Test/images-media/4_software-testing_kuhn.pdf

All triples in only 13 tests, covering $\binom{10}{3} 2^3 = 960$ combinations

Each row is a test:

0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Each column is a parameter:



EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 8d: Test Design Process

Creative thinkers made here.

Combinatorial Test Design Process

- Billions of test cases are not good!
- Instead of testing all possible combinations, find a subset that satisfy some well-defined combination strategies.
- Not every factor contributes to every fault – a fault is often the result of the interactions of a few factors

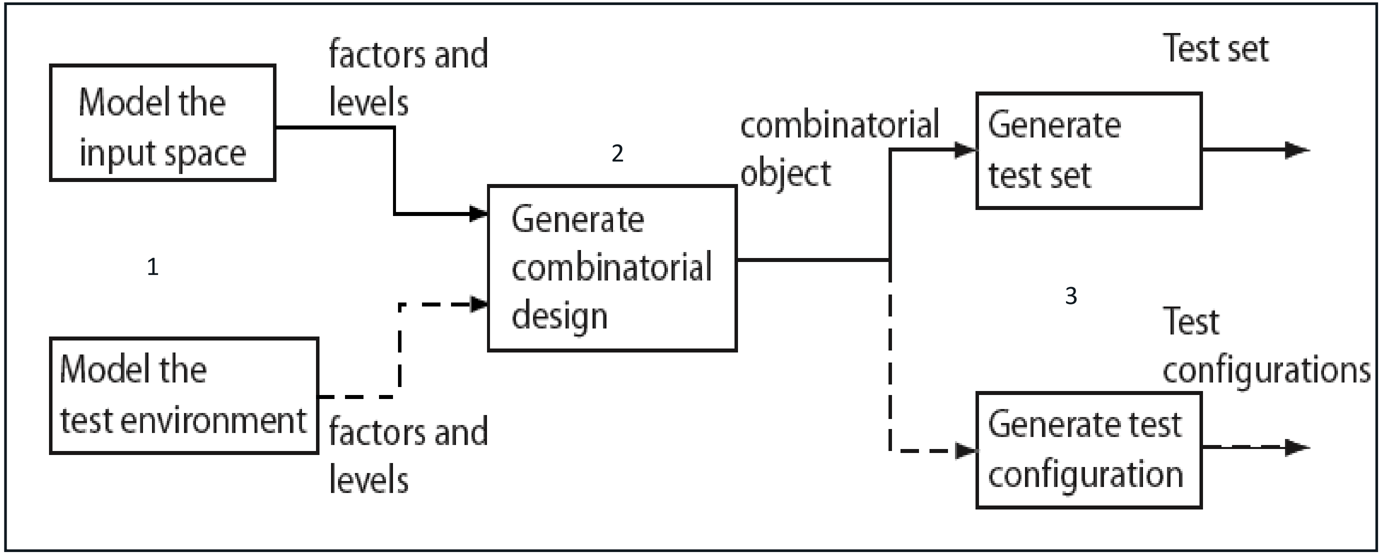


Diagram from Mathur, 2014

CSI3105: Software Testing

Module 8e: Fault Types

Creative thinkers made here.

Interaction Faults

Introduce some more terminology

Revisited the concept of insanely big input spaces/ exhaustive testing and its feasibility

The crux of todays lecture..

Combinatorial design helps find Interaction Faults

Fault Models (from slides, Mathur (2014)

Faults aimed at by the combinatorial design techniques are known as interaction faults.

We say that an interaction fault is triggered when a certain combination of $t \geq 1$ input values causes the program containing the fault to enter an invalid state.

And always

- this invalid state must propagate to a point in the program execution where it is observable and hence is said to reveal the fault.

Fault Models (from slides, Mathur (2014)

Faults triggered by some value of an input variable, i.e. t=1, regardless of the values of other input variables, are known as simple faults.

For t=2, the faults are known as pairwise interaction faults.

In general, for any arbitrary value of t, the faults are known as t-- way interaction faults.

Fault Models (from slides, Mathur (2014)

Interactions e.g., failure occurs

```
if enemy_distance < 10
  {
        // do faulty AI attack code
  } // 1- way interaction

if enemy_distance < 10 AND enemy_health > 80
  {
        // do faulty AI attack code
  } // 2- way interaction

if enemy_distance < 10 AND enemy_health > 80 AND my_health > 80
  {
        // do faulty AI attack code
  } // 3- way interaction
```

Pairwise Interaction Fault - Example from Mathur (2014)

Error due to the interaction between factors X and Y

Program Specification:

Program requires 3 inputs *x*, *y*, *z*.

- *x* is assigned a value from the set $\{x_1, x_2, x_3\}$,
- variable *y* a value from the set $\{y_1, y_2, y_3\}$ and,
- variable *z* a value from the set $\{z_1, z_2\}$.

The program outputs:

- the function *f(x, y, z)* when $x = x_1$ and $y = y_2$,
- function *g(x, y)* when $x = x_2$ and $y = y_{1,,}$
- **function *f(x, y, z)* - *g(x, y)* when $x = x_1$ and $y = y_1$**
- and function *f(x, y, z)* + *g(x, y) when x = x2 and y = y2*

```
begin
  int x, y, z;
  input (x, y, z);
  if(x==x1 and y==y2)
    output (f(x, y, z));
  else if(x==x2 and y==y1)
    output (g(x, y));
  else
    output (f(x, y, z)+g(x, y))   ← This statement is not protected correctly.
end
```

Pairwise Interaction Fault - Example from Mathur (2014)

Error due to the interaction between factors X a

```
begin
  int x, y, z;
  input (x, y, z);
  if(x==x1 and y==y2)
    output (f(x, y, z));
  else if(x==x2 and y==y1)
    output (g(x, y));
  else
    output (f(x, y, z)+g(x, y))    ← This statement is not protected correctly.
end
```

Program Specification:

The program outputs:

- the function *f(x, y, z)* when $x = x_1$ and $y = y_2$,
- function *g(x, y)* when $x = x_2$ and $y = y_{1,,}$
- **function *f(x, y, z)* - *g(x, y)* *when* $x = x_1$ and $y = y_1$**
- and function *f(x, y, z)* + *g(x, y) when x = x2 and y = y2*

Program contains one error.

It must output *f(x, y, z)* - *g(x, y)* when $x = x_1$ and $y = y_1$ and *f(x, y, z)* + *g(x, y)* when $x = x_2$ and $y = y_2$

Error is revealed when value of *f(x, y, z)* + *g(x, y) not equal to f(x, y, z)* - *g(x, y) when* $x = x_1$ and $y = y_1$ for any value of *z*

3-way Interaction Fault - Example from Mathur (2014)

```
begin
  int x, y, z, p;
  input (x, y, z);
  p=(x+y)*z;    ← This statement must be p=(x−y)*z
   if(p≥0)
    output (f(x, y, z));
  else
    output (g(x, y));
end
```

Possible inputs
-1 ,-1, 0
-1, -1, 1
-1, 0, 0
-1, 0, 1
1, -1, 0
1, -1, 1
1, 0, 0
1, 0, 1

Program Specification:

- The program takes 3 variables: x ϵ { -1, 1}, y ϵ { -1, 0}
- and z ϵ { 0, 1}.

The fault is triggered by all inputs such that **$x+y \neq x-y$ and $z \neq 0$.**

However, the fault is revealed only by the following two of the eight possible input combinations: x=1, y=-1, z=1 and x=-1, y=-1, z=1.

CSI3105:
Software Testing
Module 8f: Pairwise approach

Creative
thinkers
made here.

Pairwise approach

We now know what interaction faults are

How do we target them?

- Pairwise approach!

Pairwise approach – Algorithms and Tools

Several algorithms for generating a combinatorial object for test cases

- Latin Squares
- Orthogonal arrays
- In-parameter Order (IPO) procedure

Example Tools

- CATS (Constrained Array Test Systems)
- AETG (Telecordia Web-based system)
- CTS (Combinatorial Test Services) – IBM
- Test Vector Generator

Latin Squares

Let S be a finite set of n symbols. A Latin square of order n is an n x n matrix such that no symbol appears more than once in a row and column. The term ``Latin square'' arises from the fact that the early versions used letters from the Latin alphabet A, B, C, etc. in a square arrangement.

$$\begin{matrix} A & B \\ B & A \end{matrix} \qquad \begin{matrix} B & A \\ A & B \end{matrix}$$

S={A, B}. Latin squares of order 2.

$$\begin{matrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{matrix} \qquad \begin{matrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{matrix} \qquad \begin{matrix} 2 & 1 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{matrix}$$

S={1, 2, 3}. Latin squares of order 3.

Larger Latin Squares

Larger Latin squares of order n can be constructed by creating a row of n distinct symbols. Additional rows can be created by permuting the first row.

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

2	3	4	1
3	4	1	2
4	1	2	3
1	2	3	4

For example, here is a Latin square M of order 4 constructed by cyclically rotating the first row and placing successive rotations in subsequent rows.

Modulo arithmetic and Latin Squares

A Latin square of order n>2 can also be constructed easily by doing modulo arithmetic. For example, the Latin square M of order 4 given below is constructed such that $M(i, j)=i+j \pmod 4$, $1\leq (i, j) \leq 4$.

	1	2	3	4
1	2	3	0	1
2	3	0	1	2
3	0	1	2	3
4	1	2	3	0

Mutually orthogonal Latin squares

Up next…

Mutually Orthogonal Latin Squares (MOLS)

Let M1 and M2 be two Latin squares, each of order n. Let M1(i, j) and M2(i, j) denote, respectively, the elements in the ith row and jth column of M1 and M2.

We now create an n x n matrix L from M1 and M2 such that the M(i, j) is M1(i, j)M2(i, j), i.e. we simply juxtapose the corresponding elements of M1 and M2.

If each element of L is unique, i.e. it appears exactly once in L, then M1 and M2 are said to be mutually orthogonal Latin squares of order n.

MOLS: Example

There are no MOLS of order 2. MOLS of order 3 follow.

$$M_1 = \begin{matrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{matrix} \qquad M_2 = \begin{matrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{matrix}$$

$$L = \begin{matrix} 1\,2 & 2\,3 & 3\,1 \\ 2\,1 & 3\,2 & 1\,3 \\ 3\,3 & 1\,1 & 2\,2 \end{matrix}$$

Juxtaposing the corresponding elements gives us L. Its elements are unique and hence M1 and M2 are MOLS.

MOLS: How many of a given order?

MOLS(n) is the set of MOLS of order n. When n is prime, or a power of prime, MOLS(n) contains n-1 mutually orthogonal Latin squares. Such a set of MOLS is a complete set.

MOLS do not exist for n=2 and n=6 but they do exist for all other values of n>2. Numbers 2 and 6 are known as Eulerian numbers after the famous mathematician Leonhard Euler (1707-1783). The number of MOLS of order n is denoted by N(n). When n is prime or a power of prime, N(n)=n-1.

MOLS: How many of a given order?

- We can use MOLS to help us in pair wise test case design!
- If the number of levels of a factor is greater than 2, brilliant..
- What happens if the number of levels is 2 and no MOLs exist?

EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 8g: SAMNA technique

Creative thinkers made here.

Pairwise designs

In this video we will look at a simple technique to generate a subset of factor combinations from the complete set. Each combination selected generates at least one test input or test configuration for the program under test.

Only 2-valued, or binary, factors are considered. Each factor can be at one of two levels. This assumption will be relaxed later.

Pairwise designs: Example

Suppose that a program to be tested requires 3 inputs, one corresponding to each input variable. Each variable can take only one of two distinct values.

Considering each input variable as a factor, the total number of factor combinations is 2^3. Let X, Y, and Z denote the three input variables and {X1, X2}, {Y1, Y2}, {Z1, Z2} their respective sets of values. All possible combinations of these three factors follow.

$$
\begin{array}{ll}
(X_1, Y_1, Z_1) & (X_1, Y_1, Z_2) \\
(X_1, Y_2, Z_1) & (X_1, Y_2, Z_2) \\
(X_2, Y_1, Z_1) & (X_2, Y_1, Z_2) \\
(X_2, Y_2, Z_1) & (X_2, Y_2, Z_2)
\end{array}
$$

Pairwise designs: Reducing the combinations

Now suppose we want to generate tests such that each pair appears in at least one test. There are 12 such pairs: (X1, Y1), (X1, Y2), (X1, Z1), (X1, Z2), (X2, Y1), (X2, Y2), (X2, Z1), (X2, Z2), (Y1, Z1), (Y1, Z2), (Y2, Z1), and (Y2, Z2). The following four combinations cover all pairs:

$$\begin{array}{ll} (X_1, Y_1, Z_2) & (X_1, Y_2, Z_1) \\ (X_2, Y_1, Z_1) & (X_2, Y_2, Z_2) \end{array}$$

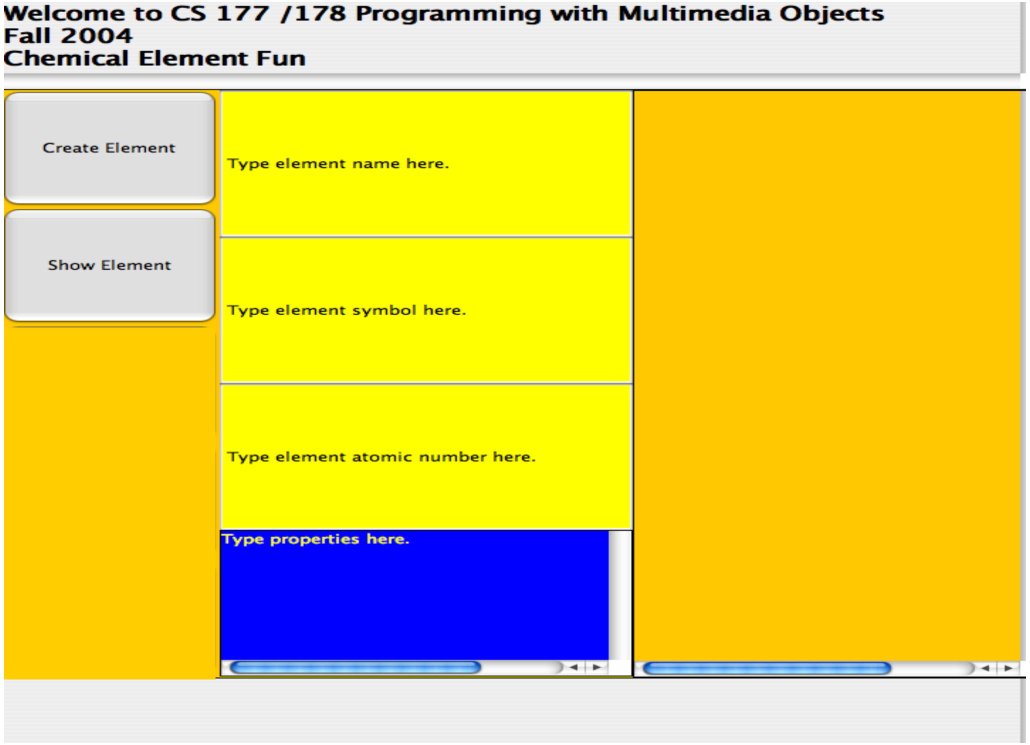
The above design is also known as a pairwise design. It is a balanced design because each value occurs exactly the same number of times. *There are several sets of four combinations that cover all 12 pairs.*

Example: ChemFun applet

A Java applet ChemFun allows its user to create an in-memory database of chemical elements and search for an element. The applet has 5 inputs listed after the next slide with their possible values.

We refer to the inputs as factors. For simplicity we assume that each input has exactly two possible values.

Example: ChemFun applet



Example: ChemFun applet: Factor identification

Factor	Name	Levels	Comments
1	Operation	{Create, Show}	Two buttons
2	Name	{Empty, Non-empty}	Data field, string expected
3	Symbol	{Empty, Non-empty}	Data field, string expected
4	Atomic number	{Invalid, Valid}	Data field, data typed > 0
5	Properties	{Empty, Non-empty}	Data field, string expected

5 factors, each with 2 levels.
$2^5 = 32$ tests

We can reduce this to 6 tests

Example: ChemFun applet: Factor identification

Factor	Name	Levels	Comments
1	Operation	{Create, Show}	Two buttons
2	Name	{Empty, Non-empty}	Data field, string expected
3	Symbol	{Empty, Non-empty}	Data field, string expected
4	Atomic number	{Invalid, Valid}	Data field, data typed > 0
5	Properties	{Empty, Non-empty}	Data field, string expected

Map to levels to either 0 or 1's

A test string could be 0, 0, 0, 1, 1

Test case input = Create, Empty, Empty, Valid, Non-empty

ChemFun applet: Input/Output

Input: n=5 factors

Output: A set of factor combinations such that all pairs of input values are covered.

ChemFun applet: Step 1

Variable **k** represents the number of 1’s in our test string

Compute the smallest integer k such that $n \leq {}_{2k-1}$ Input: n=5 factor

If k = 2

2k-1 = 3

3 isn’t greater or equal too n (n = 5)

If k = 3

2k-1 = 5

5 is greater or equal too n (n = 5)

ChemFun applet: Step 1

This formula shows us how many binary strings of length n , can have k ones

k = 3 (3 ones in our test string) n = 5 factors (how many inputs)

$$\binom{n}{k} = \frac{!n}{!(n-k)\ !k}$$

k = 3, n = 5

$$\frac{!\,5}{!\,(5-3)!\,3}$$

$$\frac{120}{12} = 10$$

For k=3 we have S_5= 10

ChemFun applet: Step 2

Select any subset of **n** strings from S_{2k-1}. We have, k=3 and we have the following strings in the set S_5 (sk – 1 = 5.) AND n = 5

	1	2	3	4	5
1	0	0	1	1	1
2	0	1	1	1	0
3	1	1	1	0	0
4	1	0	1	1	0
5	0	1	1	0	1
6	1	1	0	1	0
7	1	0	1	0	1
8	0	1	0	1	1
9	1	1	0	0	1
10	1	0	0	1	1

We select first five of the 10 strings in. S_5.

	1	2	3	4	5
1	0	0	1	1	1
2	0	1	1	1	0
3	1	1	1	0	0
4	1	0	1	1	0
5	0	1	1	0	1

ChemFun applet: Step 3

Append 0's to the end of each selected string. This will increase the size of each string from 2k-1 to 2k.

	1	2	3	4	5
1	0	0	1	1	1
2	0	1	1	1	0
3	1	1	1	0	0
4	1	0	1	1	0
5	0	1	1	0	1

$\longrightarrow$

	1	2	3	4	5	6
1	0	0	1	1	1	0
2	0	1	1	1	0	0
3	1	1	1	0	0	0
4	1	0	1	1	0	0
5	0	1	1	0	1	0

ChemFun applet: Step 4

Each combination is of the kind (X_1, X_2,…, X_n), where the value of each variable is selected depending on whether the bit in column i, $1 \leq i \leq n$, is a 0 or a 1.

	1	2	3	4	5	6
1	0	0	1	1	1	0
2	0	1	1	1	0	0
3	1	1	1	0	0	0
4	1	0	1	1	0	0
5	0	1	1	0	1	0

ChemFun applet: Step 4 (contd.)

The following factor combinations by replacing the 0s and 1s in each column by the corresponding values of each factor.

Factor	Name	Levels	Comments
1	Operation	{Create, Show}	Two buttons
2	Name	{Empty, Non-empty}	Data field, string expected
3	Symbol	{Empty, Non-empty}	Data field, string expected
4	Atomic number	{Invalid, Valid}	Data field, data typed > 0
5	Properties	{Empty, Non-empty}	Data field, string expected

	1	2	3	4	5	6
1	0	0	1	1	1	0
2	0	1	1	1	0	0
3	1	1	1	0	0	0
4	1	0	1	1	0	0
5	0	1	1	0	1	0

	1	2	3	4	5	6
1	Create	Create	Show	Show	Show	Create
2	Empty	Non-empty	Non-empty	Non-empty	Empty	Empty
3	Non-empty	Non-empty	Non-empty	Empty	Empty	Empty
4	Valid	Invalid	Valid	Valid	Invalid	Invalid
5	Empty	Non-empty	Non-Empty	Empty	Non-empty	Empty

ChemFun applet: tests

	1	2	3	4	5	6
1	Create	Create	Show	Show	Show	Create
2	Empty	Non-empty	Non-empty	Non-empty	Empty	Empty
3	Non-empty	Non-empty	Non-empty	Empty	Empty	Empty
4	Valid	Invalid	Valid	Valid	Invalid	Invalid
5	Empty	Non-empty	Non-Empty	Empty	Non-empty	Empty

$$t_1 : \; < \; \text{Button} = \text{Create}, \text{Name} = \text{“”}, \text{Symbol} = \text{‘C”}, \text{Atomic number} = 6, \text{Properties} = \text{“”} \; >$$

ChemFun applet: All tests

$$
\begin{array}{lll}
T = \{ & t_1: & < \text{Button} = \text{Create}, \text{Name} = \text{``''}, \text{Symbol} = \text{`C'}, \\
& & \text{Atomic number} = 6, \text{Properties} = \text{``''} > \\
& t_2: & < \text{Button} = \text{Create}, \text{Name} = \text{``Carbon''}, \text{Symbol} = \text{`C'}, \\
& & \text{Atomic number} = -6, \text{Properties} = \text{``Non-metal''} > \\
& t_3: & < \text{Button} = \text{Show}, \text{Name} = \text{``Hydrogen''}, \text{Symbol} = \text{`C'}, \\
& & \text{Atomic number} = 1, \text{Properties} = \text{``Non-metal''} > \\
& t_4: & < \text{Button} = \text{Show}, \text{Name} = \text{``Carbon''}, \text{Symbol} = \text{`C'}, \\
& & \text{Atomic number} = 6, \text{Properties} = \text{``''} > \\
& t_5: & < \text{Button} = \text{Show}, \text{Name} = \text{``''}, \text{Symbol} = \text{``''}, \\
& & \text{Atomic number} = -6, \text{Properties} = \text{``Non-metal''} > \\
& t_6: & < \text{Button} = \text{Create}, \text{Name} = \text{``''}, \text{Symbol} = \text{``''}, \\
& & \text{Atomic number} = -6, \text{Properties} = \text{``''} > \\
\} & &
\end{array}
$$

Recall that the total number of combinations is 32. Requiring only pairwise coverage reduces the tests to 6.

CSI3105: Software Testing

Module 8h: MOLs Approach

Creative thinkers made here.

MOLS: How many of a given order?

- What happens if the number of levels is greater that 2?
- MOLs!

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

4 levels? Map to 1 - 4

Combinatorial Testing : PD MOLS

- Example 1:
  - Program with three input parameters as follows:
  - X1 = {p, q, r}, X2 = {a, b} and X3 = {1,2,3,4},
  - all combinations would involve 3*2*4 = 24 test cases.
  - |X3| = 4, |X1| = 3, |X2| = 2

Combinatorial Testing : PD MOLS

X1 = {p, q, r}, X2 = {a, b} and X3 = {1,2,3,4},

Block	**X3**	**X1**	**X2**
1			
2			
3			
4			

Combinatorial Testing : PD MOLS

X1 = {p, q, r}, X2 = {a, b} and X3 = {1,2,3,4},

Block	**X3**	**X1**	**X2**
1	1		
	1		
	1		
	1		
2	2		
	2		
	2		
	2		
3	3		
	3		
	3		
	3		
4	4		
	4		
	4		
	4		

Combinatorial Testing : PD MOLS

X1 = {p, q, r}, X2 = {a, b} and X3 = {1,2,3,4},

Block	**X3**	**X1**	**X2**
1	1	1	
	1	2	
	1	3	
	1	4	
2	2	1	
	2	2	
	2	3	
	2	4	
3	3	1	
	3	2	
	3	3	
	3	4	
4	4	1	
	4	2	
	4	3	
	4	4	

Combinatorial Testing : PD MOLS

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

X1 = {p, q, r}, X2 = {a, b} and X3 = {1,2,3,4},

Block	**X3**	**X1**	**X2**
1	1	1	1
	1	2	3
	1	3	4
	1	4	2
2	2	1	2
	2	2	4
	2	3	3
	2	4	1
3	3	1	3
	3	2	1
	3	3	2
	3	4	4
4	4	1	4
	4	2	2
	4	3	1
	4	4	3

Combinatorial Testing : PD MOLS

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

X1 = {p, q, r}, X2 = {a, b} and X3 = {1,2,3,4},

Block	X3	X1	X2
1	1	1	1
	1	2	3
	1	3	4
	1	4	2
2	2	1	2
	2	2	4
	2	3	3
	2	4	1
3	3	1	3
	3	2	1
	3	3	2
	3	4	4
4	4	1	4
	4	2	2
	4	3	1
	4	4	3

Combinatorial Testing : PD MOLS

X1 = {p, q, r}, X2 = {a, b} and X3 = {1,2,3,4},

Block	**X3**	**X1**	**X2**
1	1	1	1
	1	2	1
	1	3	2
	1	1	2
2	2	1	2
	2	2	1
	2	3	2
	2	1	1
3	3	1	1
	3	2	1
	3	3	2
	3	1	2
4	4	1	2
	4	2	2
	4	3	1
	4	1	1

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

Combinatorial Testing : PD MOLS

X1 = {p, q, r}, X2 = {a, b} and X3 = {1,2,3,4},

Block	**X3**	**X1**	**X2**
1	1	p	a
	1	q	a
	1	r	b
	1	p	b
2	2	p	b
	2	q	a
	2	r	b
	2	p	a
3	3	p	a
	3	q	a
	3	r	b
	3	p	b
4	4	p	b
	4	q	b
	4	r	a
	4	p	a

Combinatorial Testing : PD MOLS

X1 = {p, q, r}, X2 = {a, b} and X3 = {1,2,3,4},

Block	**X3**	**X1**	**X2**
1	1	p	a
	1	q	a
	1	r	b
2	2	p	b
	2	q	a
	2	r	b
3	3	p	a
	3	q	a
	3	r	b
4	4	p	b
	4	q	b
	4	r	a

all combinations would involve 3*2*4 = 24 test cases
We now have 12 test

Combinatorial Testing : PD MOLS

Connection = {nbn, adsl, 4g}, Printer = {local, network} and OS= {XP, win 7, win 8, win 10},

Block	**OS**	**Con**	**Printer**
1	XP	nbn	local
	XP	adsl	local
	XP	4g	network
2	win 7	nbn	network
	win 7	adsl	local
	win 7	4g	network
3	win 8,	nbn	local
	win 8,	adsl	local
	win 8,	4g	network
4	win 10	nbn	network
	win 10	adsl	network
	win 10	4g	local

Factors, Levels and Factor Combination (from slides,

Factor	Levels
Size	Large | Medium | Small
Topping	Chicken | Veg | Vegan | Beef
Address	Valid | Invalid

Factor	Levels
Size	1 | 2 | 3
Topping	1 | 2 | 3 | 4
Address	1 | 2

3 * 4 * 2= 24 combinations

EDITH COWAN UNIVERSITY

Combinatorial Testing : PD MOLS

Size **= {1, 2, 3},** Address **= {1, 2} and Size = {1,2,3,4},**

Block	Topping	Size	Address
1			
2			
3			
4			

Combinatorial Testing : PD MOLS

Size = {p, q, r}, Address = {a, b} and Topping = {1,2,3,4},

Block	Topping	Size	Address
1	1		
	1		
	1		
	1		
2	2		
	2		
	2		
	2		
3	3		
	3		
	3		
	3		
4	4		
	4		
	4		
	4		

Combinatorial Testing : PD MOLS

Size = {p, q, r}, Address = {a, b} and Topping = {1,2,3,4},

Block	Topping	Size	Address
1	1	1	
	1	2	
	1	3	
	1	4	
2	2	1	
	2	2	
	2	3	
	2	4	
3	3	1	
	3	2	
	3	3	
	3	4	
4	4	1	
	4	2	
	4	3	
	4	4	

Combinatorial Testing : PD MOLS

Size = {p, q, r}, Address = {a, b} and Topping = {1,2,3,4},

Block	Topping	Size	Address
1	1	1	1
	1	2	3
	1	3	4
	1	4	2
2	2	1	2
	2	2	4
	2	3	3
	2	4	1
3	3	1	3
	3	2	1
	3	3	2
	3	4	4
4	4	1	4
	4	2	2
	4	3	1
	4	4	3

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

Combinatorial Testing : PD MOLS

Size = {1, 2, 3}, Address = {1, 2} and Topping = {1,2,3,4}

Block	Topping	Size	Address 1
	1	1	1
	1	2	3
	1	3	4
	1	4	2
2	2	1	2
	2	2	4
	2	3	3
	2	4	1
3	3	1	3
	3	2	1
	3	3	2
	3	4	4
4	4	1	4
	4	2	2
	4	3	1
	4	4	3

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

Combinatorial Testing : PD MOLS

Size = {1, 2, 3}, Address = {1, 2} and Topping = {1,2,3,4},

Block	Topping	Size	Address
1	1	1	1
	1	2	1
	1	3	2
	1	1	2
2	2	1	2
	2	2	1
	2	3	2
	2	1	1
3	3	1	1
	3	2	1
	3	3	2
	3	1	2
4	4	1	2
	4	2	2
	4	3	1
	4	1	1

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

Combinatorial Testing : PD MOLS

Size = {1, 2, 3}, Address = {1, 2} and Topping = {1,2,3,4},

Block	**Topping**	**Size**	**Address**
1	Chicken	Large	valid
	Chicken	Medium	valid
	Chicken	Small	invalid
	Chicken	Large	invalid
2	Veg	Large	invalid
	Veg	Medium	valid
	Veg	Small	invalid
	Veg	Large	valid
3	Vegan	Large	valid
	Vegan	Medium	valid
	Vegan	Small	invalid
	Vegan	Large	invalid
4	Beef	Large	invalid
	Beef	Medium	invalid
	Beef	Small	valid
	Beef	Large	valid

Combinatorial Testing : PD MOLS

Size = {1, 2, 3}, Address = {1, 2} and Topping = {1,2,3,4},

Block	**Topping**	**Size**	**Address**
1	Chicken	Large	valid
	Chicken	Medium	valid
	Chicken	Small	invalid
2	Veg	Large	invalid
	Veg	Medium	valid
	Veg	Small	invalid
3	Vegan	Large	valid
	Vegan	Medium	valid
	Vegan	Small	invalid
4	Beef	Large	invalid
	Beef	Medium	invalid
	Beef	Small	valid

Combinatorial Testing : PD MOLS

t1	Chicken	Large	valid
t2	Chicken	Medium	valid
t3	Chicken	Small	invalid
t4	Veg	Large	invalid
t5	Veg	Medium	valid
t6	Veg	Small	invalid
t7	Vegan	Large	valid
t8	Vegan	Medium	valid
t9	Vegan	Small	invalid
t10	Beef	Large	invalid
t11	Beef	Medium	invalid
t12	Beef	Small	valid

Combinatorial Testing : PD MOLS

T1<Chicken, Large, “14 Walter RD, Morley 6052”> expected = $14.95
T2 <Chicken, Medium, “14 Walter RD, Morley 6052”> expected = $12.95
T3<Chicken , small, invalid> expected = No sale

T4<Veg, Large, invalid> expected = No sale
T5<Veg, Medium, “14 Walter RD, Morley 6052”> expected = $11.95
T6<Veg,   Small, invalid> expected = No sale

t7<Vegan, Large, “14 Walter RD, Morley 6052”> > expected = $17.95
T8< Vegan, Medium, “14 Walter RD, Morley 6052”> expected = $13.95
T9< Vegan , Small, invalid> expected = No sale

T10< Beef , Large, invalid> expected = No sale
T11< Beef , Medium , invalid> expected = No sale
T12< Beef , Small, “14 Walter RD, Morley 6052”> expected = $9.95

CSI3105: Software Testing

Module 8i: Test Coverage

Creative thinkers made here.

Combinatorial Testing : Coverage

All combinations

- Test cases where every possible combinations of values of the parameters must be covered.
- **Example 1**: Program with three input parameters as follows: X_1 = {p, q, r}, X_2 = {a, b} and X_3 = {1,2,3,4}, all combinations would involve 3*2*4 = 24 test cases.

Each Choice

- At least one test case for each value associated with the parameter.
- Using Example 1, a test set this criterion would be: {(p,a,1) (q,b,2) (r,a,3)(p,b,4)}

Pairwise

- Given any two parameters, at least one test case for every combinations of their respective values
- Covered in previous slides

t-wise

- Given any t parameters, at least one test case for every combinations of values of the t parameters.

Combinatorial Testing : Coverage

Base Choice Coverage

- One of the possible values of each parameter is picked as the *base choice*
- A base test is defined by using the base choice for each parameter
- Hold all base choices constant, except for one which is substituted then with non-base choice of that respective parameter
- **Example 1**: Program with three input parameters as follows: X_1 = {p, q, r}, X_2 = {a, b} and X_3 = {1,2,3,4}, all combinations would involve 3*2*4 = 24 test cases
- Using Example 1 and base choices for: X_1 = {p}, X_2 = {a } and X_3 = {1 }, test cases would be: {(p,a,1) (q,a,1) (r,a,1) {(p,b,1) (p,a,2) (p,a,3) (p,a,4)}

Multiple Base Choices Coverage

- At least one or more base choices are designated for each of the parameters.
- Generation of base test cases same as before