

Black Box Software Testing

Spring 2005

MULTI-VARIABLE TESTING

by

Cem Kaner, J.D., Ph.D.

Professor of Software Engineering

Florida Institute of Technology

and

James Bach

Principal, Satisfice Inc.

Copyright (c) Cem Kaner & James Bach, 2000-2005

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

These notes are partially based on research that was supported by NSF Grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers." Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Combination Chart

	Var 1	Var 2	Var 3	Var 4	Var 5
Test 1	Value 11	Value 12	Value 13	Value 14	Value 15
Test 2	Value 21	Value 22	Value 23	Value 24	Value 25
Test 3	Value 31	Value 32	Value 33	Value 34	Value 35
Test 4	Value 41	Value 42	Value 43	Value 44	Value 45
Test 5	Value 51	Value 52	Value 53	Value 54	Value 55
Test 6	Value 61	Value 62	Value 63	Value 64	Value 65

In a combination test, we test several variables together. Each test explicitly sets values for each of the variables under test.

Challenges of multivariable testing

1. The space of possible tests is enormous
2. How to decide which variables to combine?
3. What IS the fault model?
4. How to figure out what the relationships among the variables actually are, in detail?

Combination Testing

- There are several approaches to combination testing:
 - *Mechanical (or procedural)*. The tester uses a routine procedure to determine a good set of tests
 - *Risk-based*. The tester combines test values (the values of each variable) based on perceived risks associated with noteworthy combinations
 - *Scenario-based*. The tester combines test values on the basis of interesting stories created for the combinations

Domain testing

- In 1-dimensional testing, we run two tests for every boundary:
 - Test the boundary-valid value
 - Test the boundary-invalid value
- For example if $X < 24$ defines the boundary, we use $X=24$ (invalid) and $X=24$ -delta (valid). We choose the smallest workable delta to minimize the possibility of an error hiding within the gap between 24 and 24-delta.
- In multi-dimensional testing, we start by testing each dimension on its own, reasonably thoroughly.
- Then we reduce the set of values to test per dimension, probably to the boundaries:
 - Too-low (TL), valid lowest (VL), valid biggest (VB), too big (TB)
 - where

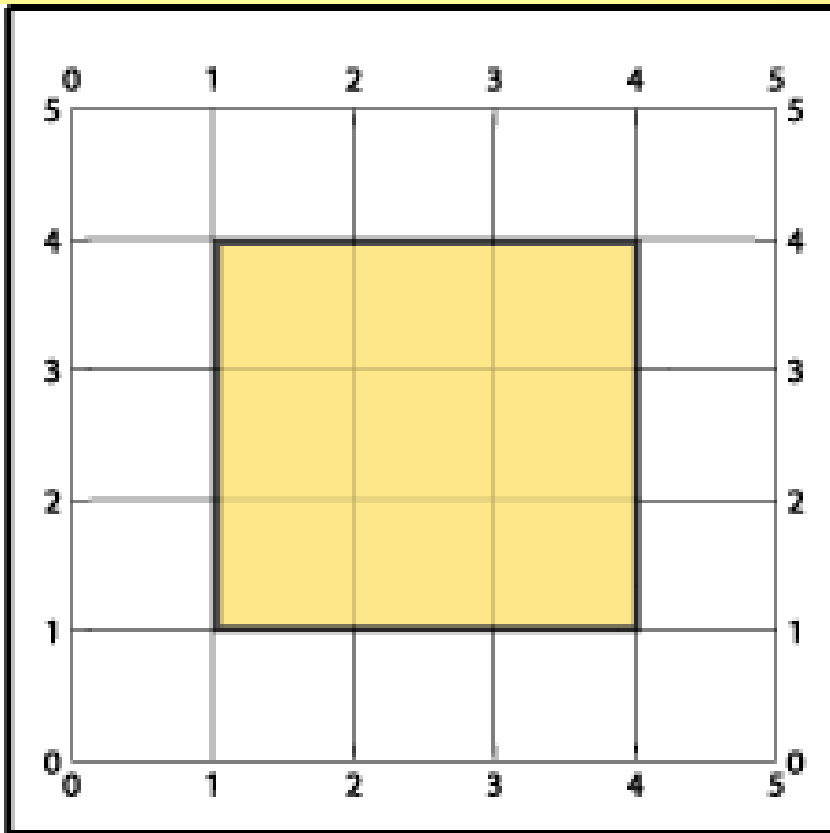
$$TL = VL - \text{delta and } TB = VB + \text{delta}$$

Defining the domain:

2 variables

- Suppose we have two numeric variables, V1 and V2.
- We analyze each variable in terms of its subdomains and boundaries. Thus we have for each variable:
 - V1: Too-low (TL), valid lowest (VL), valid biggest (VB), too big (TB)
 - V2: Too-low (TL), valid lowest (VL), valid biggest (VB), too big (TB)
 - Where we set
 - $TL = VL - \text{delta}$ (smallest available difference between two numbers)
 - $TH = VB + \text{delta}$

Example: 2 variables



Consider the following domain definition:

$$1 \leq V1 < 4$$

$$1 \leq V2 < 4$$

Store data to 3 digits precision:

$$TL = 0.999$$

$$VL = 1.00$$

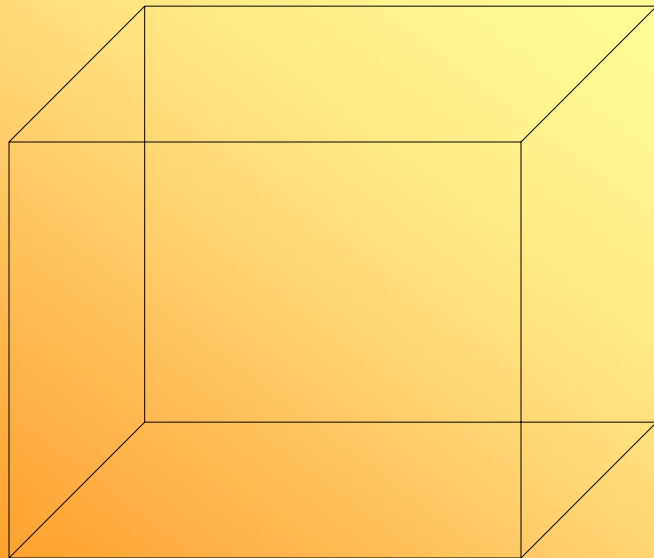
$$VB = 3.99$$

$$TB = 4.00$$

Defining the domain

3 independent variables

- Suppose we have 3 numeric variables, V1, V2, V3.
- We analyze each variable in terms of its subdomains and boundaries. Thus we might have for each variable:
 - V1: Too-low (TL), valid lowest (VL), valid biggest (VB), too big (TB)
 - V2: Too-low (TL), valid lowest (VL), valid biggest (VB), too big (TB)
 - V3: Too-low (TL), valid lowest (VL), valid biggest (VB), too big (TB)



In this simple model, anything inside the box is a valid value, and anything outside the box is not.

(When we restrict ourselves to valid values, we are thinking inside the box.)

Mechanical approach #1

"Weak testing" version 1

	V1	V2	V3
Test 1	VL	VL	VL
Test 2	VB	VB	VB
Test 3	TL	TL	TL
Test 4	TB	TB	TB

- We create enough tests to cover every value of every variable, once. If the largest number of values is N, we need only N tests
- Note the collisions of error cases. If Test 3 fails, is it because of the bad value of V1, V2, V3, or some combination of them?
- What bug do we expect to find in Test 3 that we would not find in a test of single dimension, with a bad value? Why do we need a combination?

Too-low (TL), lowest valid (VL), biggest valid (VB), too big (TB)

"Weak testing" version 2

	V1	V2	V3
Test 1	VL	VL	VL
Test 2	VB	VB	VB
Test 3	TL	VL	VB
Test 4	VB	TL	VL
Test 5	VL	VL	TL
Test 6	TB	VB	VL
Test 7	VL	TB	VB
Test 8	VB	VL	TB

- In this second version, we treat error cases specially:
 - Generate a core set of tests for "valid" (non-error) inputs
 - Generate additional tests in which one error case is allowed per test case. (Jorgensen calls this "weak robust equivalence class testing.")
 - We might also add a few market-critical combinations

Too-low (TL), lowest valid (VL), biggest valid (VB), too big (TB)

"Weak testing" version 3

- *All Singles* -

	V1	V2	V3
Test 1	LV	LV	LV
Test 2	BV	BV	BV

- Drop the error cases
 - test them in single-variable tests.
- Create tests only for valid values
 - Jorgensen calls this “weak normal equivalence class testing”
- Note the coverage that we do and do not achieve:
 - We have a test for every valid value of interest of every variable
 - We are not set up to detect interactions among variables.
 - Here, for example, we check all minima together and all maxima.
 - Should we worry about Low-High combinations?

Mechanical approach #2

"Strong testing" version 1

Test every combination of values of interest:

Jorgensen calls this "strong robust equivalence class testing"

	V1	V2	V3
Test 1	TL	TL	TL
Test 2	TL	TL	LV
Test 3	TL	TL	BV
Test 4	TL	TL	TB
Test 5	TL	LV	TL
Test 6	TL	LV	LV
Test 7	TL	LV	BV
Test 8	TL	LV	TB
Test 9	TL	BV	TL
Test 10	TL	BV	LV

This is part of the table. The complete table has

$4 * 4 * 4$ tests.

In general, if N is the number of variables we test together and they have $k_1, k_2 \dots k_N$ values, strong testing requires

$k_1 \times k_2 \times \dots \times k_N$ tests

Too-low (TL), lowest valid (VL), biggest valid (VB), too big (TB)

"Strong testing" version 2

All n-tuples

- Start with strong testing
- But restrict the values of interest to valid values.
 - Jorgensen calls this “strong normal equivalence class analysis”
- Cover error cases in the one-variable tests.
- If there are N independent dimensions, and we test only LV and BV for each, there are **2^N** tests

More “strong testing”

- Another variation includes all valid-value combinations plus a separate set of combination tests in which one, some, or all variables have an error value.
- Tests that include several errors are of interest only if we think that multiple errors might have some type of cumulative effect.

Mechanical approach #3

Combinatorial testing

- We have N variables
- We assume the variables are independent
 - A value of one variable does not change the effects or validity of values of other variables
- We consider only valid values of interest
 - An invalid value stops the test.
 - (V1, V2, Bad, V4, V5) → what do we learn about V1, V2, V4 or V5?
 - Anything in this test of interest other than “Bad” will be masked
- Our goal is to sample from the space of possible N-tuples in way that assures a minimum level of combination coverage:
 - All N-tuples → all combinations of valid values
 - All singles → all individual valid values
 - All pairs → all pairs of valid values
 - All triples → all triplets of valid values

Combinatorial Example

Find

Find what: lowercase

Match case

Direction

Up Down

Find Next

Cancel

- Here is a simple Find dialog. It takes three inputs:
 - Find what: a text string
 - Match case: yes or no
 - Direction: up or down
- Simplify this by considering only three values for the text string, “lowercase” and “Mixed Cases” and “CAPITALS”.

Combinations Example

- 1 How many combinations of these three variables are possible?
- 2 List ALL the combinations of these three variables.
- 3 Now create combination tests that cover all possible pairs of values, but don't try to cover all possible triplets. List one such set.
- 4 How many test cases are in this set?

Combinations Example

1. How many combinations of these three variables are possible?

- **Find what** has 3 values (lowercase, mixed, caps) (L M C)
- **Match case** has 2 values (Yes / No) (Y N)
- **Direction** has 2 values (Up / Down) (U D)

So there will be $3 \times 2 \times 2 = 12$ tests

2. List ALL the combinations of these three variables.

L Y U	M Y U	C Y U
L Y D	M Y D	C Y D
L N U	M N U	C N U
L N D	M N D	C N D

3. By the way, a more complete analysis will also consider whether the string is in the document or not. We'll add a 4th binary variable to the analysis soon.

Building an all-pairs table

- Label the columns with the variable names.
- List variables in descending order (of number of possible values)
- Each column will have repetition.
 - To determine how many times (rows in which) to repeat the first value before creating a row for the second multiply the number of variable values in column 1 x the number that will be in column 2
- In our example,
 - Find What has 3 values
 - Match Case has 2 values
 - So there will be at least 6 rows

Combination Testing

- Building an all-pairs combination table:
 - In the second column, list all the values of the second variable, skip the line, list the values again, etc. In our example, variable 2's possible values are U,D so the table looks like this so far

Find (LMC)	Match (YN)
L	Y
L	N
M	Y
M	N
C	Y
C	N

Combination Testing

Building an all-pairs combination table:

- Each section of the third column (think of LL as defining a section, MM as defining another) will have to contain every value of variable 3. Order the values such that the variables also make all pairs with variable 2.
- Our variable 3 has two values, U and D
- The third section can be filled in either way, and you might highlight it so that you can reverse it later. The decision (say D, U) is arbitrary.

Find (LMC)	Match (YN)	Direct (UD)
L	Y	U
L	N	D
M	Y	D
M	N	U
C	Y	D
C	N	U

Combination Testing

Now that we've solved the 3-column exercise, let's try adding more variables. Each will have two values.

Let's start by making this look a little more general

A	D	F
A	E	G
B	D	G
B	E	F
C	D	G
C	E	F

The 4th column goes in easily:

- We start by making sure we hit all pairs of values of column 4 and column 2
- then all pairs of column 4 and column 3.

A	D	F	H
A	E	G	I
B	D	G	I
B	E	F	H
C	D	G	H
C	E	F	I

Combination Testing

Watch this first attempt on column 5. We achieve all pairs of JK with columns 1, 2, and 3, but miss it for column 4.

The most recent arbitrary choice was KJ in the 2nd section. (Once that was determined, we had to pick JK for the third in order to pair K with an F in the 3rd column.)

So we will erase the last choice and try again:

A	D	F	H	J
A	E	G	I	K
B	D	G	I	K
B	E	F	H	J
C	D	G	H	J
C	E	F	I	K

Combination Testing

- We flipped the last arbitrary choice (column 5, section 2, to JK from KJ) and erased the JK in section 3.
- We then fill in section 3 by checking for missing pairs.
- JK, JK, JK gives us three DJ, DJ, DJ pairs (2nd and 5th columns) so we have to flip to KJ for the third section.
- Now everything works

A	D	F	H	J
A	E	G	I	K
B	D	G	I	J
B	E	F	H	K
C	D	G	H	K
C	E	F	I	J

Combination Testing

But when we add the next column, we see that we just can't achieve all pairs with 6 values. The first one works up to column 4 but then fails to get pair KL or JM. The next fails on HM and IL

A	D	F	H	J	L
A	E	G	I	K	M
B	D	G	I	J	L
B	E	F	H	K	M
C	D	G	H	K	M
C	E	F	I	J	L

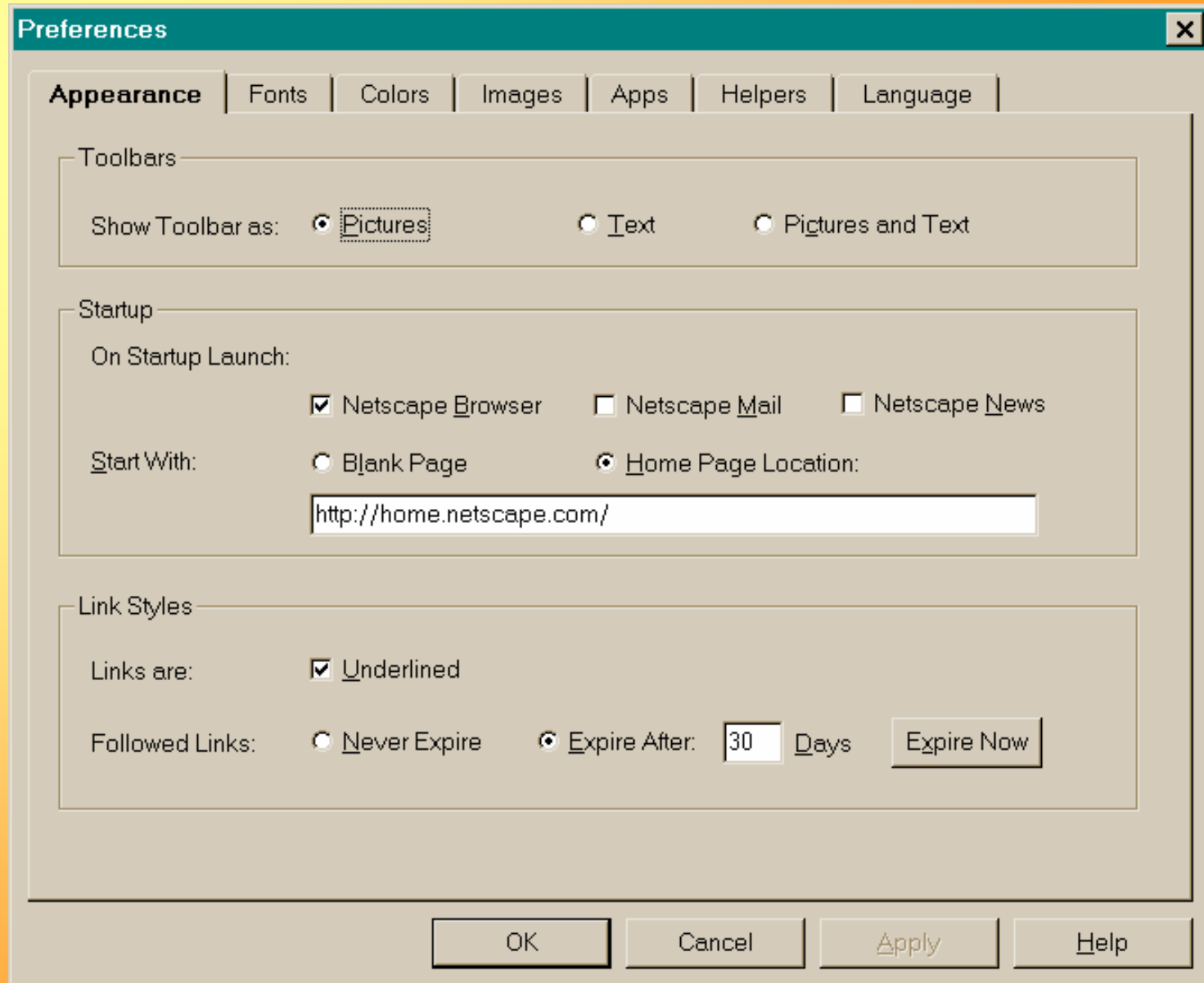
A	D	F	H	J	L
A	E	G	I	K	M
B	D	G	I	J	M
B	E	F	H	K	L
C	D	G	H	K	L
C	E	F	I	J	M

Combination Testing

- When all else fails, add rows. We need one for HM and one for IL, so add two rows. In general, we would need as many rows as the last column has values.
- The other values in the two rows are arbitrary, leave them blank and fill them in as needed when you add new columns. At the very end, fill the remaining blank ones with arbitrary values
- We have 8 tests instead of $3 \times 2 \times 2 \times 2 \times 2 = 96$

A	D	F	H	J	L
A	E	G	I	K	M
			H		M
B	D	G	I	J	M
B	E	F	H	K	L
			I		L
C	D	G	H	K	L
C	E	F	I	J	M

*Let's try
this again
on an old
Netscape
preference
dialog*



The Netscape example

- If we just look at the Appearance tab of the Netscape Preferences dialog, we see the following variables:
 - Toolbars -- 3 choices (P, T, B)
(*pictures, text* or *both*)
 - On Startup Launch --(*browser, mail, news*). Each is an independent binary.
 - Browser (Y, N)
 - Mail (Y, N)
 - News (Y, N)
 - Start With -- 3 choices (B,V,E)
(*blank* page, *valid* existing file, *error* (syntax) in the URL)
(Many more cases are possible)
 - Links -- 2 choices (D,U)
(*don't* underline, *underlined*)
 - Followed Links -- 2 choices (N,E)
(*never* expire, *expire* after 30 days) (Many more cases are possible)

The Netscape example

- I simplified the combinations by simplifying the choices for two fields.
- In the **Start With** field, I used either a valid home page name or a blank. Some other tests for this field are:
 - Link to a different type of file, such as pdf
 - Link to a nonexistent file
 - Abbreviated URL, such as name.htm instead of http://
 - File on the local drive, the local network drive, or the remote drive
 - maximum length file names, maximum length paths
 - Note that a bad URL won't stop Netscape from starting, so we should be able to use an error case here without blocking testing of the other variables
- For combination testing, select a few of these that look like they might interact with other variables. Test the rest independently.
- Similarly for the **Expire After** field. This lets you enter the number of days to store links. If you use more than one value, use boundary cases, not all the numbers in the range.
- *In multi-variable testing, use partition analysis or other special values instead of testing all values in combination with all other variables' all values.*

All N-tuples

- We can create $3 \times 2 \times 2 \times 2 \times 3 \times 2 \times 2 = 288$ different test cases by testing these variables in combination. Here are some examples, from the combination table.
- This is what Jorgensen would call “strong normal” testing.
- Strong because we test for faults triggered by a combination of conditions.
- Normal because we omit error cases.

	Toolbars PTB	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Start With BVE	Links DU	Followed NE
Test # 1	P	Y	Y	Y	B	D	N
2	P	Y	Y	N	B	D	E
3	P	Y	N	Y	B	U	N
4	P	Y	N	N	B	U	E
5	P	Y	Y	Y	V	D	N
6	P	Y	Y	N	V	D	E
7	P	N	N	Y	V	U	N
8	P	N	N	N	V	U	E
9	P	N	Y	Y	E	D	N
10	P	N	Y	N	E	D	E
11	P	N	N	Y	E	U	N
12	P	N	N	N	E	U	E

Here are the 288 test cases. Every value of every variable is combined with every combination of the other variables.

1	PYYYBDN	PNY YBDN	TYYYBDN	TNYYBDN	BYYYBDN	BNYYBDN
2	PYYYVDE	PNY YVDE	TYYYVDE	TNYYVDE	BYYYVDE	BNYYVDE
3	PYYYEDN	PNY YEDN	TYYYEDN	TNYYEDN	BYYYEDN	BNYYEDN
4	PYYYBUE	PNY YBUE	TYYYBUE	TNYYBUE	BYYYBUE	BNYYBUE
5	PYYYVUN	PNY YVUN	TYYYVUN	TNYYVUN	BYYYVUN	BNYYVUN
6	PYYYEUE	PNY YEUE	TYYYEUE	TNYYEUE	BYYYEUE	BNYYEUE
7	PY YNBDN	PNY NBDN	TY YNBDN	TNY NBDN	BY YNBDN	BN YNBDN
8	PY YNVDE	PNY NVDE	TY YNVDE	TNY NVDE	BY YNVDE	BN YNVDE
9	PY YNEDN	PNY NEDN	TY YNEDN	TNY NEDN	BY YNEDN	BN YNEDN
10	PY YNBUE	PNY NBUE	TY YNBUE	TNY NBUE	BY YNBUE	BN YNBUE
11	PY YNVUN	PNY NVUN	TY YNVUN	TNY NVUN	BY YNVUN	BN YNVUN
12	PY YNEUE	PNY NEUE	TY YNEUE	TNY NEUE	BY YNEUE	BN YNEUE
13	PYYYBDE	PNY YBDE	TYYYBDE	TNYYBDE	BYYYBDE	BNYYBDE
14	PYYYVDN	PNY YVDN	TYYYVDN	TNYYVDN	BYYYVDN	BNYYVDN
15	PYYYEDE	PNY YEDE	TYYYEDE	TNYYEDE	BYYYEDE	BNYYEDE
16	PYYYBUN	PNY YBUN	TYYYBUN	TNYYBUN	BYYYBUN	BNYYBUN
17	PYYYVUE	PNY YVUE	TYYYVUE	TNYYVUE	BYYYVUE	BNYYVUE
18	PYYYEUN	PNY YEUN	TYYYEUN	TNYYEUN	BYYYEUN	BNYYEUN
19	PY YNBDE	PNY NBDE	TY YNBDE	TNY NBDE	BY YNBDE	BN YNBDE
20	PY YNVDN	PNY NVDN	TY YNVDN	TNY NVDN	BY YNVDN	BN YNVDN
21	PY YNEDE	PNY NEDE	TY YNEDE	TNY NEDE	BY YNEDE	BN YNEDE
22	PY YNBUN	PNY NBUN	TY YNBUN	TNY NBUN	BY YNBUN	BN YNBUN
23	PY YNVUE	PNY NVUE	TY YNVUE	TNY NVUE	BY YNVUE	BN YNVUE
24	PY YNEUN	PNY NEUN	TY YNEUN	TNY NEUN	BY YNEUN	BN YNEUN
25	PY NYBDE	PNN YBDE	TY NYBDE	TNN YBDE	BY NYBDE	BN NYBDE
26	PY NYVDN	PNN YVDN	TY NYVDN	TNN YVDN	BY NYVDN	BN NYVDN
27	PY NYEDE	PNN YEDE	TY NYEDE	TNN YEDE	BY NYEDE	BN NYEDE
28	PY NYBUN	PNN YBUN	TY NYBUN	TNN YBUN	BY NYBUN	BN NYBUN
29	PY NYVUE	PNN YVUE	TY NYVUE	TNN YVUE	BY NYVUE	BN NYVUE
30	PY NYEUN	PNN YEUN	TY NYEUN	TNN YEUN	BY NYEUN	BN NYEUN
31	PY NNBDE	PNN NBDE	TY NNBDE	TNN NBDE	BY NNBDE	BN NNBDE
32	PY NNVDN	PNN NVDN	TY NNVDN	TNN NVDN	BY NNVDN	BN NNVDN
33	PY NNEDE	PNN NEDE	TY NNEDE	TNN NEDE	BY NNEDE	BN NNEDE
34	PY NNBUN	PNN NBUN	TY NNBUN	TNN NBUN	BY NNBUN	BN NNBUN
35	PY NNVUE	PNN NVUE	TY NNVUE	TNN NVUE	BY NNVUE	BN NNVUE
36	PY NNEUN	PNN NEUN	TY NNEUN	TNN NEUN	BY NNEUN	BN NNEUN
37	PY NYBDN	PNN YBDN	TY NYBDN	TNN YBDN	BY NYBDN	BN NYBDN
38	PY NYVDE	PNN YVDE	TY NYVDE	TNN YVDE	BY NYVDE	BN NYVDE
39	PY NYEDN	PNN YEDN	TY NYEDN	TNN YEDN	BY NYEDN	BN NYEDN
40	PY NYBUE	PNN YBUE	TY NYBUE	TNN YBUE	BY NYBUE	BN NYBUE
41	PY NYVUN	PNN YVUN	TY NYVUN	TNN YVUN	BY NYVUN	BN NYVUN
42	PY NYEUE	PNN YEUE	TY NYEUE	TNN YEUE	BY NYEUE	BN NYEUE
43	PY NNBBDN	PNN NBBDN	TY NNBBDN	TNN NBBDN	BY NNBBDN	BN NNBBDN
44	PY NNVDE	PNN NVDE	TY NNVDE	TNN NVDE	BY NNVDE	BN NNVDE
45	PY NNEEDN	PNN NEEDN	TY NNEEDN	TNN NEEDN	BY NNEEDN	BN NNEEDN
46	PY NNBUE	PNN NBUE	TY NNBUE	TNN NBUE	BY NNBUE	BN NNBUE
47	PY NNVUN	PNN NVUN	TY NNVUN	TNN NVUN	BY NNVUN	BN NNVUN
48	PY NNEUE	PNN NEUE	TY NNEUE	TNN NEUE	BY NNEUE	BN NNEUE

All N-tuples

When creating a combination table, I strongly recommend that you order the columns from the variable with the most values to the variable with the least.

	Toolbars PTB	Start With BVE	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Links DU	Followed NE
Test #1	P	B	Y	Y	Y	D	N
2	P	B	Y	Y	N	D	E
3	P	B	Y	N	Y	U	N
4	P	B	Y	N	N	U	E
5	P	V	Y	Y	Y	D	N
6	P	V	Y	Y	N	D	E
7	P	V	N	N	Y	U	N
8	P	V	N	N	N	U	E
9	P	E	N	Y	Y	D	N
10	P	E	N	Y	N	D	E
11	P	E	N	N	Y	U	N
12	P	E	N	N	N	U	E

All Singles

- There are $3+3+2+2+2+2+2=16$ different individual (single) values of interest.
- We can cover them in 3 tests

	Toolbars PTB	Start With BVE	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Links DU	Followed NE	
Test #1	P	B	Y	Y	Y	D	N	
2	T	V	N	N	N	U	E	
3	B	E	Y	Y	Y	D	N	

What about pairs?

- To simplify this, many testers would test variables in pairs, each test involving only 2 values.
- There are 109 pairs in our example.
- Testing only 2 variables at once is an inefficient form of combination testing.
- One test that combines 7 variables incorporates 21 tests of pairs of variables.

	Toolbars PTB	Start With BVE	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Links DU	Followed NE
Combo	P	B	Y	Y	Y	D	N
Pair 1	P	B					
2	P		Y				
3	P			Y			
4	P				Y		
5	P					D	
6	P						N
7		B	Y				
8		B		Y			
9		B			Y		
10		B				D	
11		B					N
12			Y	Y			
13			Y		Y		
14			Y			D	
15			Y				N
16				Y	Y		
17				Y		D	
18				Y			N
19					Y	D	
20					Y		N
21						D	N

Combinatorics

“Combinatorics is, loosely, the science of counting. This is the area in mathematic in which we study families of sets (usually) finite with certain characteristic arrangements of their elements or subsets, and ask what combinations are possible, and how many there are. This includes numerous quite elementary topics, such as enumerating all possible permutations or combinations of a finite set.”

www.albany.edu/faculty/tangr/isp602/notes/terms.htm

- In combinatorial testing, we test many variables together as an efficient way of testing many of the combinations of those variables (e.g. testing 7 variables together in one test captures ${}^7C_2 = 21$ tests of the pairs).
- So how many tests would we have to run to cover **all** the pairs?

All pairs for Netscape

We can cover all 109 pairs inside 9 tests

	Toolbars PTB	Start With BVE	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Links DU	Followed NE
Test #1	P	B	Y	Y	Y	D	N
2	P	V	Y	N	N	D	E
3	P	E	N	Y	Y	U	E
4	T	E	Y	N	Y	U	N
5	T	B	N	Y	N	D	E
6	T	V	N	N	Y	D	N
7	B	B	Y	N	N	U	E
8	B	V	N	Y	Y	U	E
9	B	E	N	N	N	D	N

All pairs for Netscape

- Let's work it through.
- We start with the first two variables (biggest and second biggest number of values of interest.)
- Here are all the pairs of those two variables. There are $3 \times 3 = 9$ of them

	Toolbars	PTB	Start With BVE
Test #1	P	B	B
2	P	V	V
3	P	E	E
4	T	E	E
5	T	B	B
6	T	V	V
7	B	B	B
8	B	V	V
9	B	E	E

All pairs for Netscape

Add the next variable

- We need all the pairs of the
 - 1st and 2nd variables
 - 1st and 3rd variables
 - 2nd and 3rd variables
- We already have the pairs for 1st & second variables
- For the 1st and 3rd, we need
 - a Y with a P, an N with a P,
 - a Y with a T, an N with a T,
 - a Y with a B and an N with a B.
- The values of the 3rd variable for the other cases don't matter for 1st & 3rd, but they might matter for 2nd & 3rd.

	Toolbars PTB	Start With BVE	On Startup, Browser Y/N
Test #1	P	B	Y
2	P	V	N
3	P	E	
4	T	B	N
5	T	V	Y
6	T	E	
7	B	B	
8	B	V	
9	B	E	

All pairs for Netscape

Add the 4th variable. We have the pairs for the 1st 3, we just have to work in the 4th.

	Toolbars PTB	Start With BVE	On Startup, Browser Y/N	On Startup, Mail Y/N
Test #1	P	B	Y	Y
2	P	V	N	N
3	P	E	Y	Y
4	T	B	N	N
5	T	V	Y	Y
6	T	E	N	N
7	B	B	Y	N
8	B	V	N	Y
9	B	E	Y	N

All pairs for Netscape

- Keep going, through the 7th variable.

	Toolbars PTB	Start With BVE	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Links DU	Followed NE
Test #1	P	B	Y	Y	Y	D	N
2	P	V	N	N	N	D	E
3	P	E	Y	Y	N	U	E
4	T	B	N	N	N	U	N
5	T	V	Y	Y	N	D	E
6	T	E	N	N	Y	D	N
7	B	B	Y	N	Y	U	E
8	B	V	N	Y	Y	U	N
9	B	E	Y	N	N	D	N

All pairs

	01	01	01
Test #1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

A	01	01	01	01	B	01	01	01	01	C	01	01	01	01	D	01	01	01	01
Test #1	0	0	0	0	Test #1	0	0	0	0	Test #1	0	0	0	1	Test #1	0	0	0	1
2	0	1	1	1	2	0	1	1	1	2	0	1	1	0	2	0	1	1	0
3	1	0	1	0	3	1	0	1	1	3	1	0	1	0	3	1	0	1	1
4	1	1	0	1	4	1	1	0	0	4	1	1	0	1	4	1	1	0	0

Reminder of a common misconception.

The lower bound on the number of rows is the number of values of column 1 times column 2 but we often need more than that.

	01	01	01	01
Test #1	0	0	0	0
2	0	1	1	1
3	1	0	1	0
4	1	1	0	1
5	0	0	0	1
6	1	1	0	0

Combinatorial testing

- At one of the LAWST meetings, we were advised that Microsoft often uses a modified all-singles in configuration testing:
 - All singles, plus
 - All other combinations designated by marketing (or by error history) as particularly interesting
- Similarly if we use all pairs, we might add to the set of tests:
 - Special cases (marketing)
 - Special cases (identified risks of higher-order interactions)

Combinatorial testing

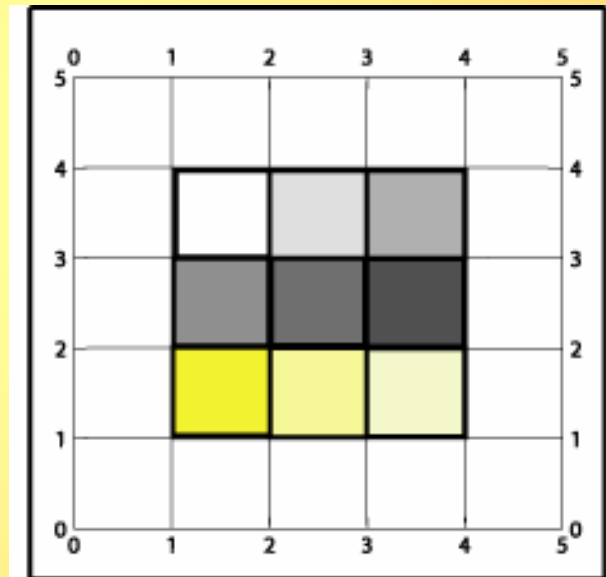
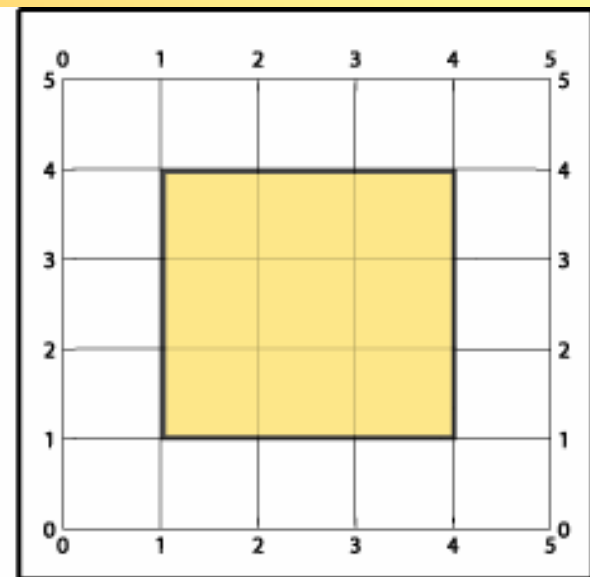
- www.pairwise.org has been collecting references and links to tools, including free tools
- Another free tool is at <http://www.satisfice.com/tools/pairs.zip>
- Rob Vanderwall has developed VPTAG, which allows you to specify some constraints (a given value of X makes a range of values of Y impossible).

Let's add some complications

- So far, we've assumed
 - Independent variables
 - All valid values are equivalent
- What if we have multiple valid equivalence classes?
- Let's assume fixed precision, to 1 digit after decimal
 - Invalid: $X < -100$ boundary -100.1
 - Valid 1 $-100 \leq X < 0$ bounds -100.0, -0.1
 - Valid 2 $0 \leq X \leq 5$ bounds 0.0, 5.0
 - Valid 3 $5 < X < 10$ bounds 5.1, 9.9
 - Invalid $10 \leq X$ bounds 10.0
- These all become values of interest in combinatorial tests

Let's add more complications

- So far, we've assumed
 - Independent variables
 - All valid values are equivalent
- What if the values of one variable affect the validity or effect of the values of another?



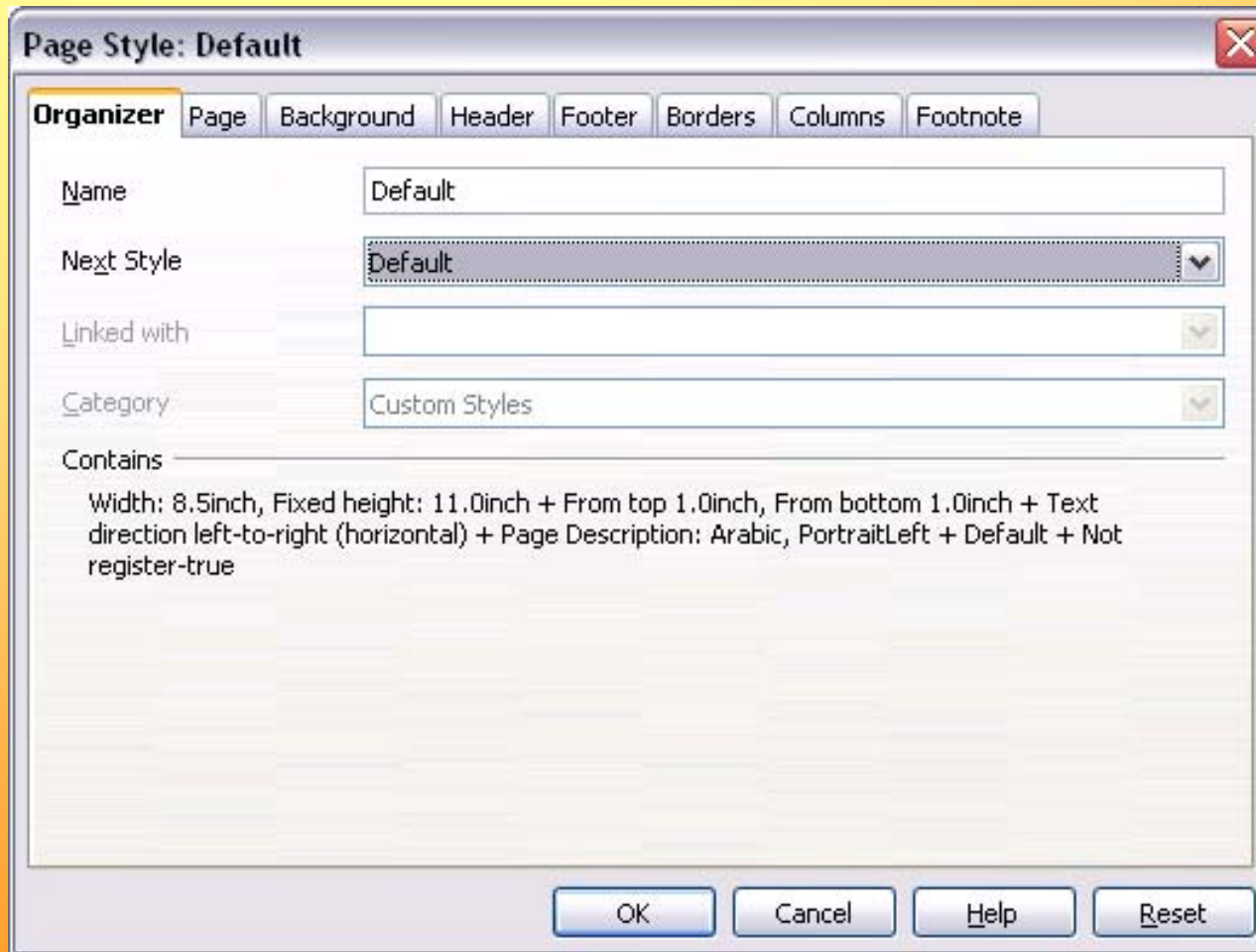
A common example: Testing a date field

- $0 < \text{day} < \text{it depends}$
- $1 \leq \text{month} \leq 12$
- $2000 < \text{year} < 3000$ (whatever limits you choose)

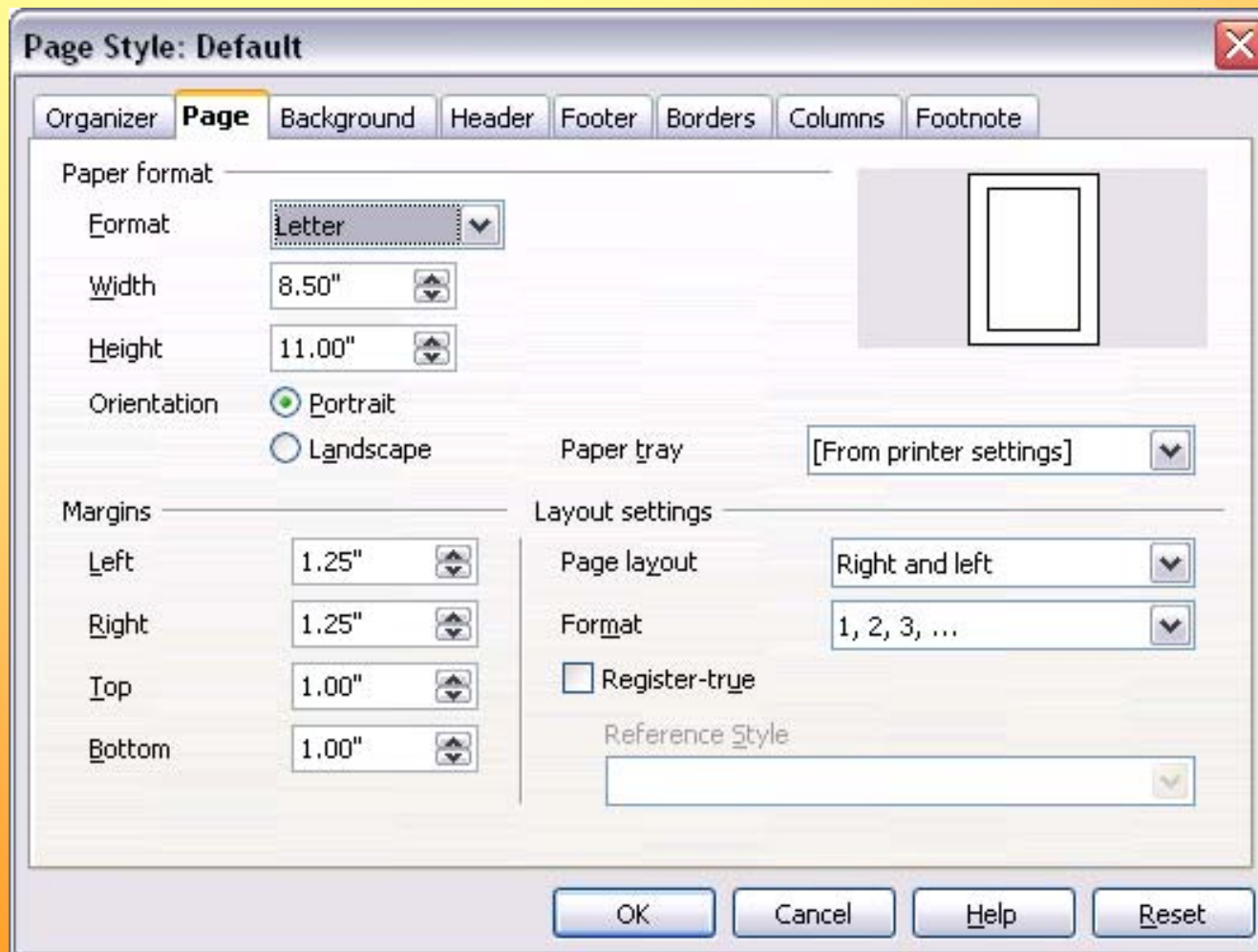
- For month 2 $1 \leq \text{day} \leq 28 \text{ or } 29$
- For months 4, 6, 9, 11 $1 \leq \text{day} \leq 30$
- For months 1,3,5,7,8,10,12 $1 \leq \text{day} \leq 31$

- See Jorgensen for a thorough analysis

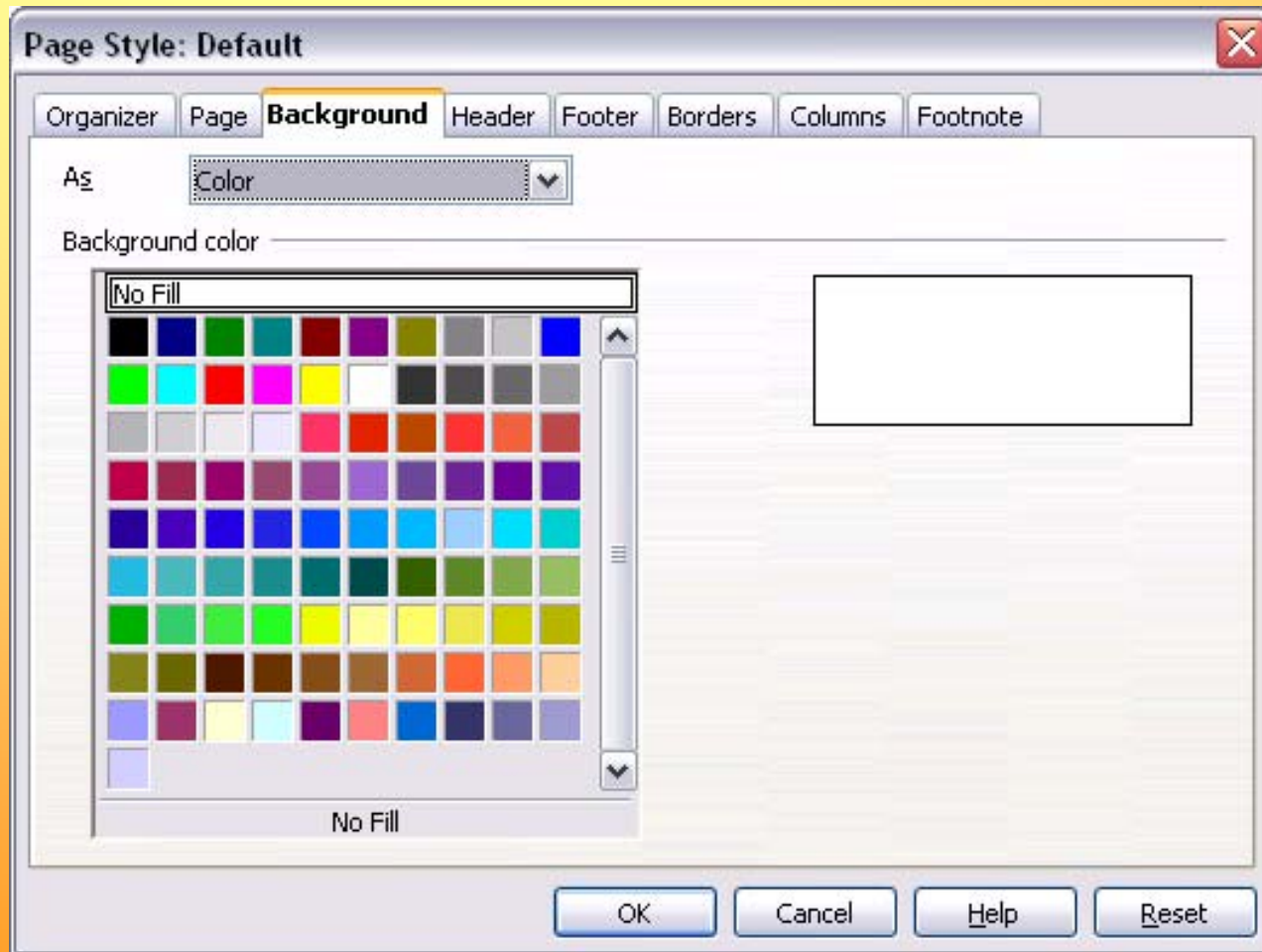
Let's make this more challenging.
The next slides present the Open
Office Writer page style dialog.



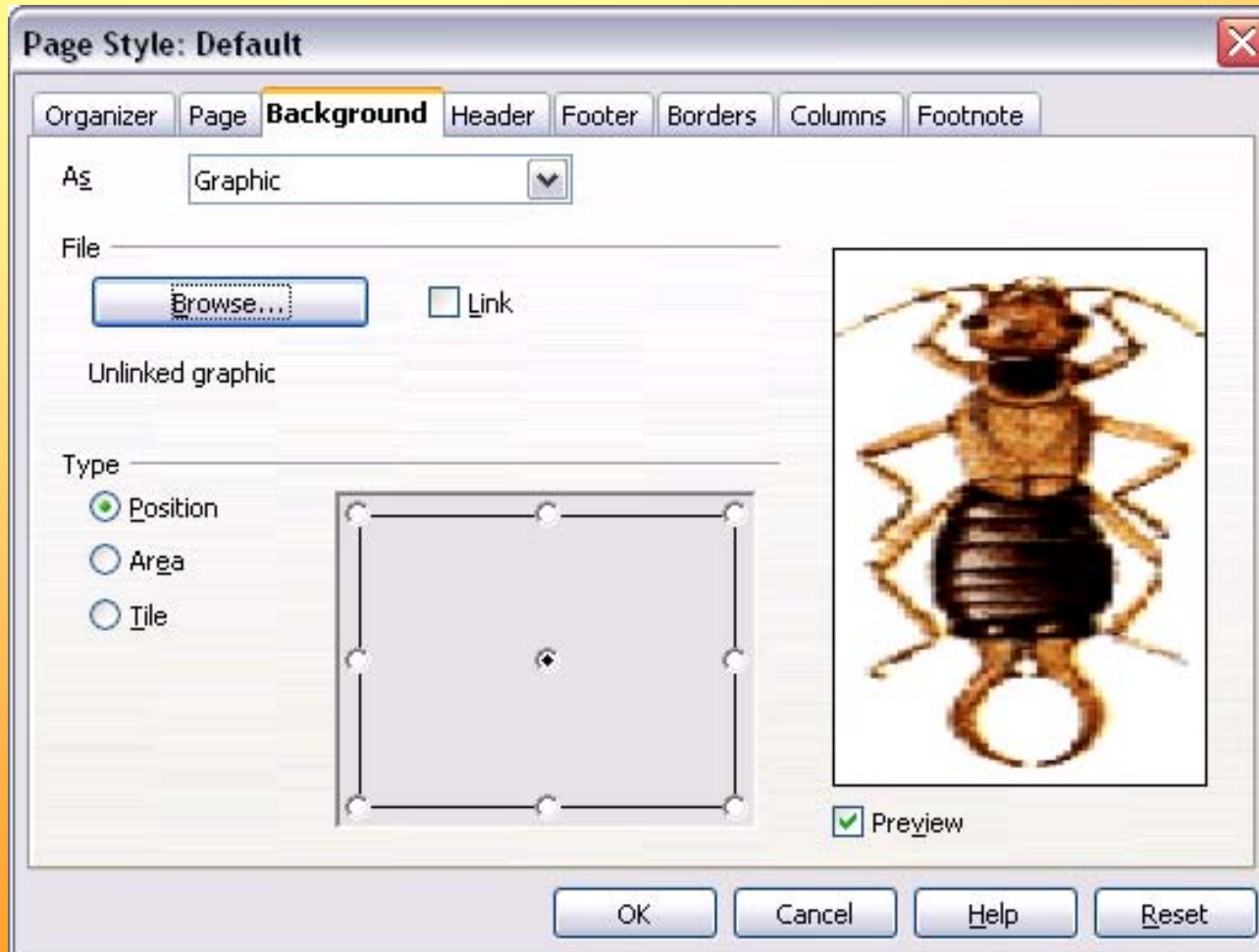
These are interesting as a group because they all interact.



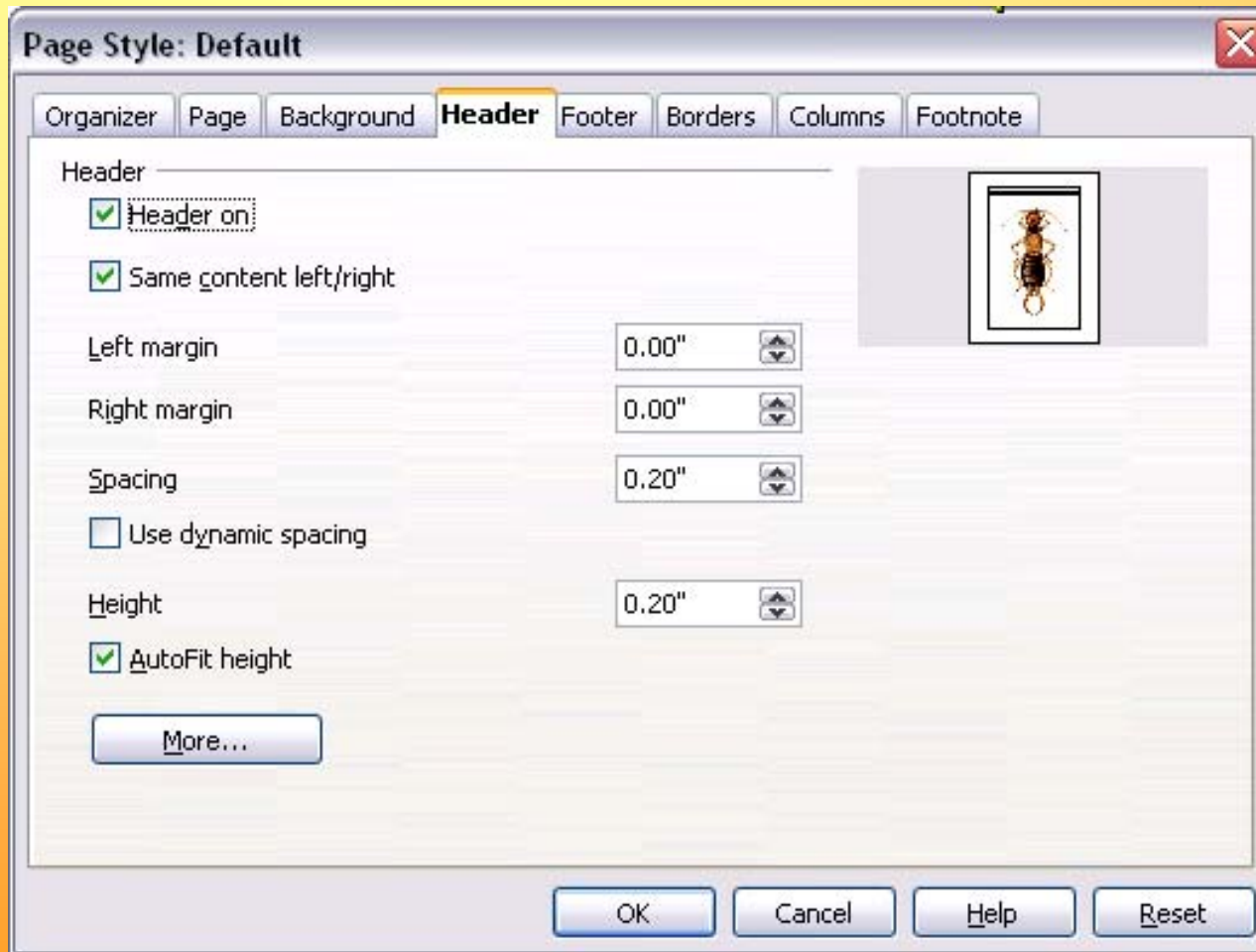
Just print a page. Its layout is jointly determined by all of these



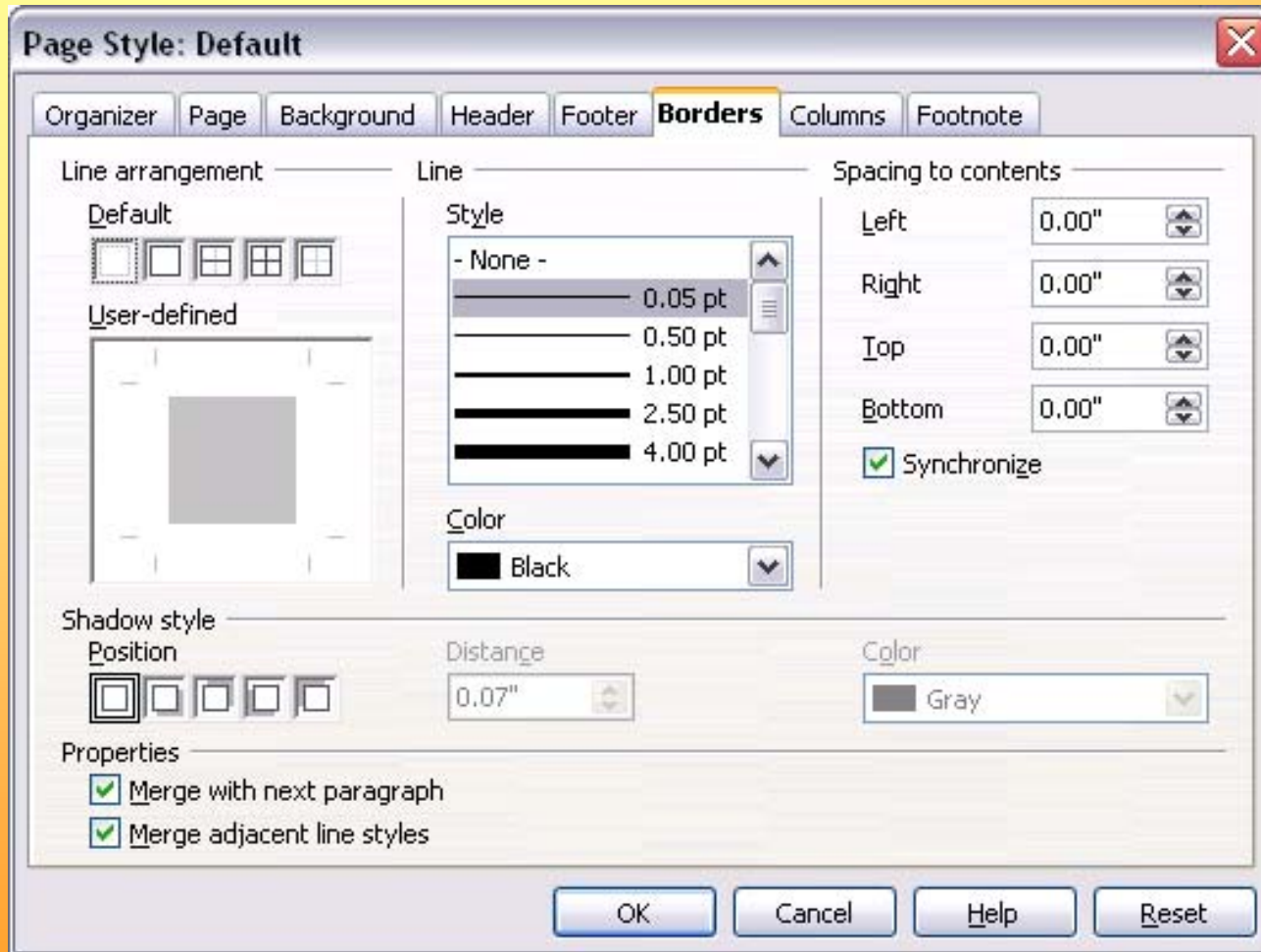
So how do we test all of these?



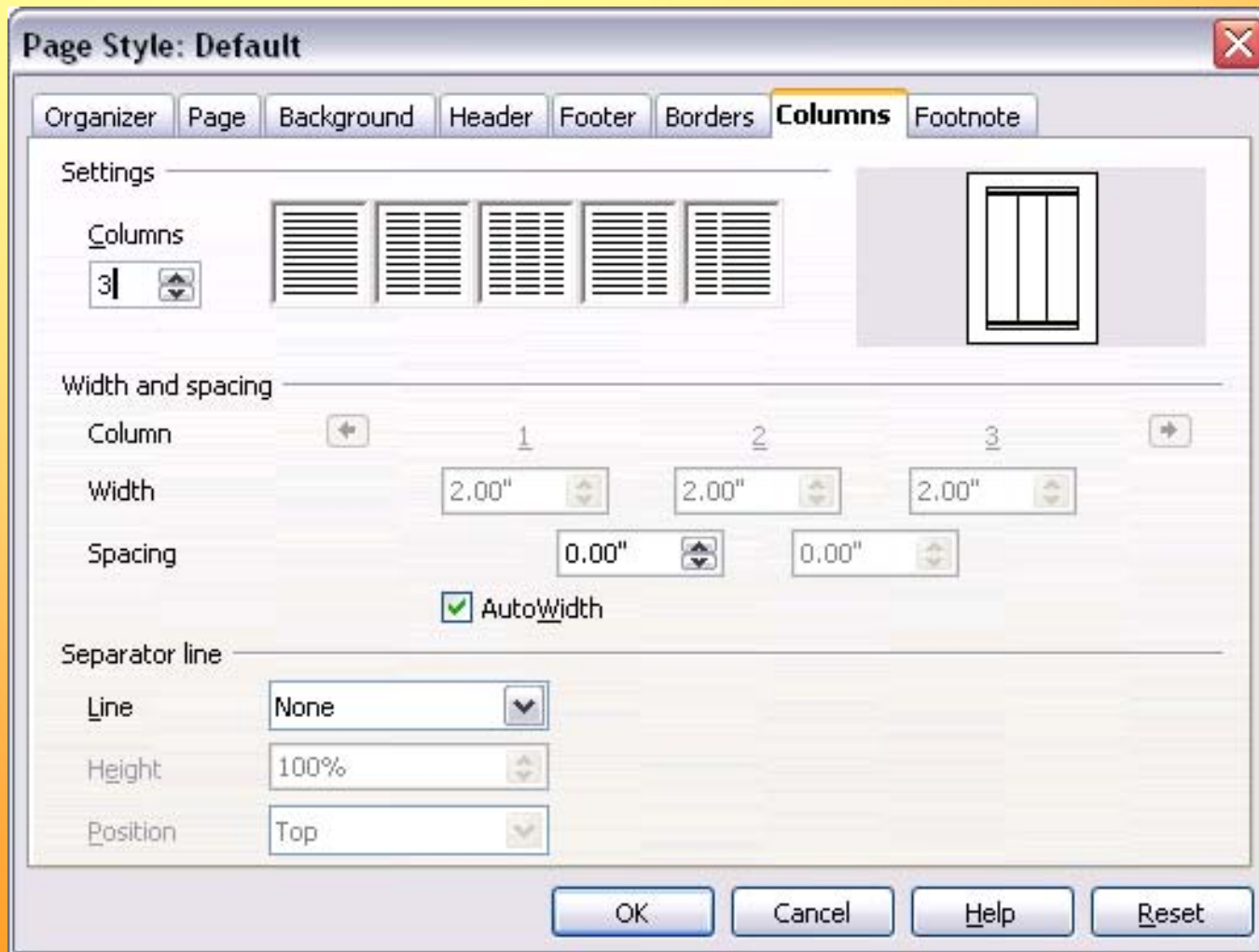
Can you list the relevant variables?



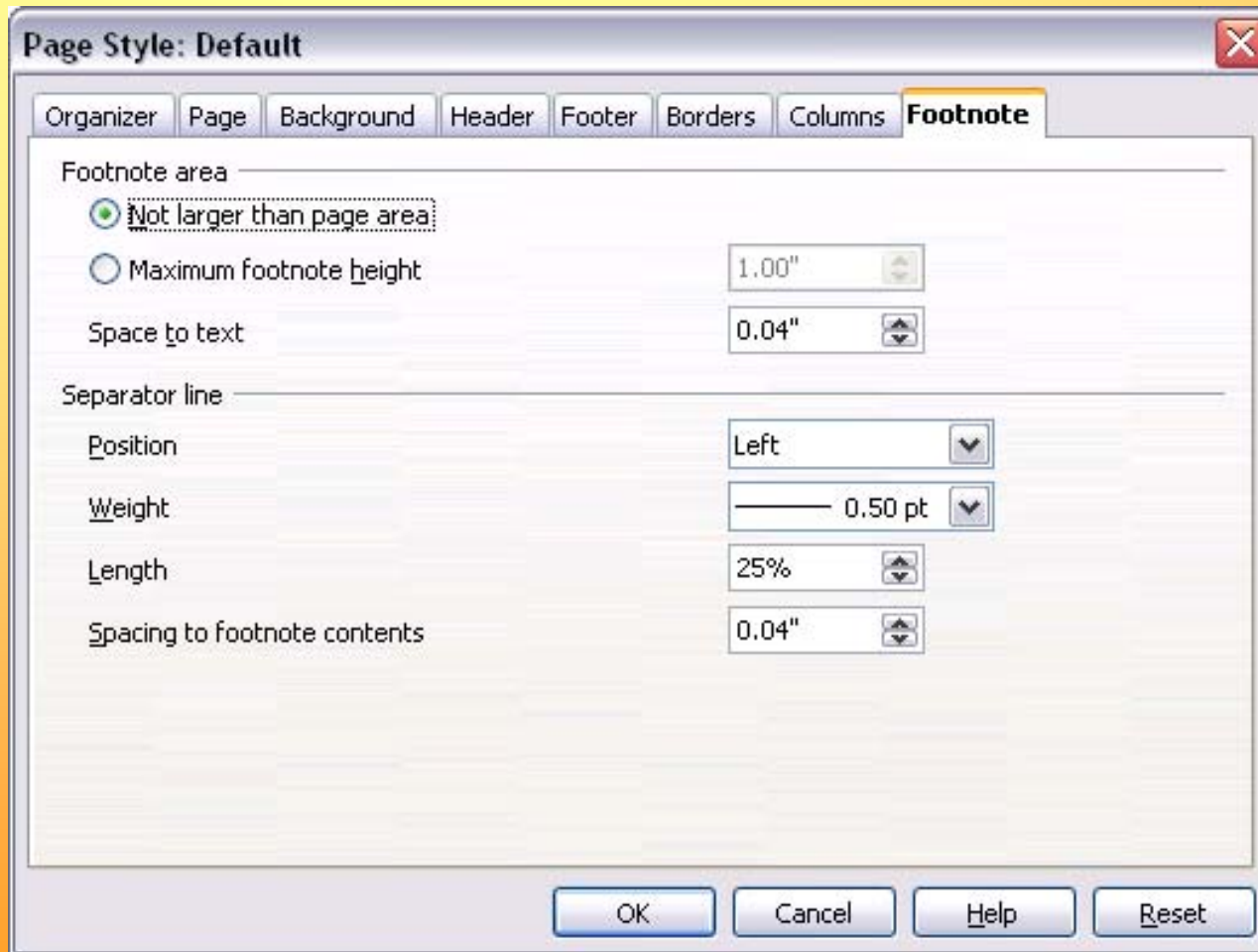
How many variables are on this page?



The number of variables on this page depends on how many columns you choose.



At last, we're through this one (1) dialog.
You can see why people would give up and
do all singles or random combination.



Thoughts on all pairs

- All pairs is ideal for independent variables.
 - A classic use is configuration testing.
 - But if they're independent, why test them in combination?
 - We are managing a type of coverage here.
 - We are rarely working from a theory of error.
 - Schroeder & Bach argue that in this case, we are probably as well off using a random combination algorithm. The set of tests will approximate all pairs
 - If we combination-test the program several times, randomness creates variation in the testing
- All pairs is adaptable when the number of constraints is small
 - Whenever a test has an invalid pair, substitute two tests, identical except that you substitute a valid value for the first (second) value of the pair. All other pairs in the test stay intact and are tested

Thoughts on all pairs

All pairs is inefficient when the number of constraints is significant.

- Beizer (Black Box Testing) discusses the general case in which there are several levels for each variable and the program behaves differently as a joint function of the settings of several variables. His presentation of a domain testing approach to this problem is interesting but as described, I find it challenging to apply.
- In electrical engineering, this situation is analyzed as *Combination Circuit Testing*. Given a set of values of interest for several variables (you arrive at them through domain analysis or in some other way), the question is whether the program behaves correctly for each combination.
- In software testing, the analysis is called *Cause-Effect Graphing*

Alternative approaches

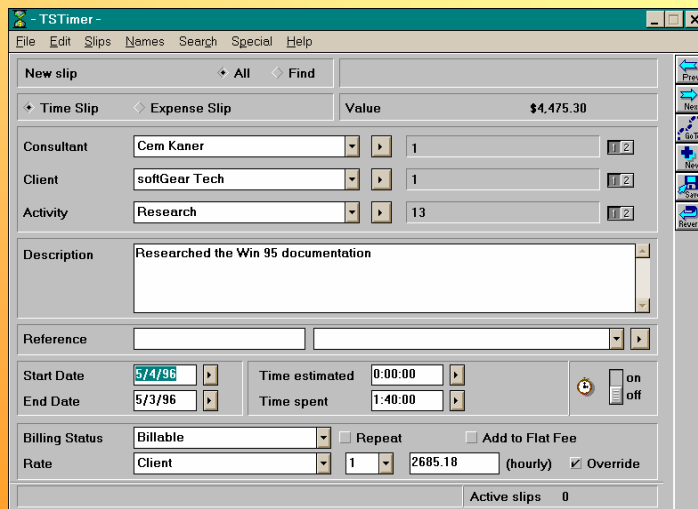
There are several approaches to combination testing

- ***Mechanical (or procedural)***. The tester uses a routine procedure to determine a good set of tests
- ***Risk-based***. The tester combines test values (the values of each variable) based on perceived risks associated with noteworthy combinations
- ***Scenario-based***. The tester combines test values on the basis of interesting stories created for the combinations.

- Some groups of variables involve too complex a set of relationships for you to analyze (given your skills, tools and the time available) or are not well enough specified for you to analyze.
- If you believe that you need to test combinations anyway, and want to consciously control the design of the tests, you might want a technique that helps you explore relationships and make sense of them.

Exploring relationships

- Look at this record (bigger on the next slide), from the Timeslips Deluxe time and billing database. In this dialog box, click the arrow next to the **Consultant** field to edit the **Consultant** record (my name, billing info, etc.) or enter a new one.
- If I edit it here, will the changes carry over to every other display of this **Consultant** record?
- Also, note that the **End Date** for this task is before the **Start Date**. *That's not possible.*



The screenshot shows the TSTimer software interface. The window title is "- TSTimer -". The menu bar includes File, Edit, Slips, Names, Search, Special, and Help. The main area is divided into several sections:

- New slip:** Includes a dropdown for "All" and a "Find" button.
- Time Slip / Expense Slip:** Shows a "Value" of \$4,475.30.
- Fields:** Consultant (Cem Kaner), Client (softGear Tech), Activity (Research), and Description (Researched the Win 95 documentation).
- Reference:** A field for entering a reference number.
- Start Date:** 5/4/96
- End Date:** 5/3/96
- Time estimated:** 0:00:00
- Time spent:** 1:40:00
- Billing Status:** Billable
- Rate:** Client, 1, 2685.18 (hourly), with an "Override" checkbox checked.
- Repeat:** A checkbox that is unchecked.
- Add to Flat Fee:** A checkbox that is unchecked.
- Active slips:** 0

Exploring relationships

- TSTimer -

File Edit Slips Names Search Special Help

New slip ◆ All ◆ Find


◆ Time Slip ◆ Expense Slip Value **\$4,475.30**

Consultant	Cem Kaner	▶	1	▶	1 2
Client	softGear Tech	▶	1	▶	1 2
Activity	Research	▶	13	▶	1 2

Description
Researched the Win 95 documentation

Reference

Start Date **5/4/96** ▶ Time estimated **0:00:00** ▶

End Date **5/3/96** ▶ Time spent **1:40:00** ▶  on off

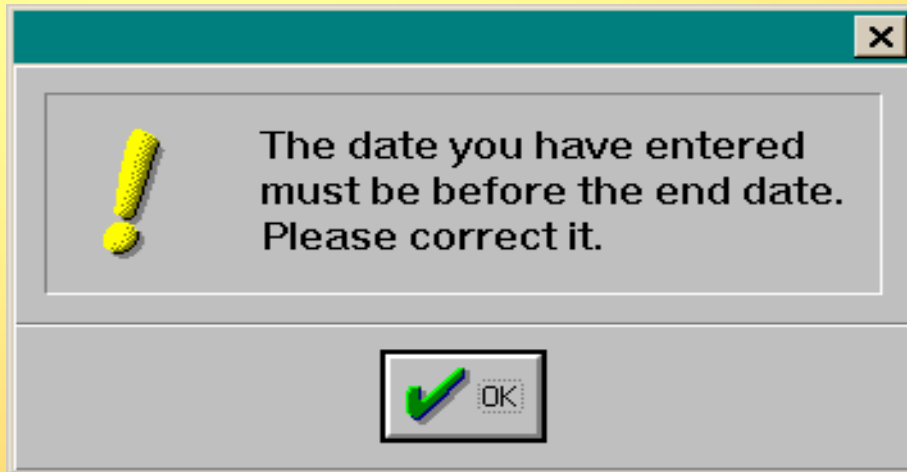
Billing Status **Billable** Repeat Add to Flat Fee

Rate **Client** **1** **2685.18** (hourly) Override

Active slips 0

Prev Next Go To New Save Revert

Exploring relationships



The program checks the **End Date** against the **Start Date** and rejects this pair as impossible because the task can't end before it starts.

*The value of **End Date** is constrained by **Start Date**, because **End Date** can't be earlier than **Start Date**.*

*The value of **Start Date** constrains **End Date**, because **End Date** can't be earlier than **Start Date**.*

Exploring relationships

A relationship table

<i>Field</i>	<i>Entry Source</i>	<i>Display</i>	<i>Print</i>	<i>Related Variable</i>	<i>Relationship</i>
Variable 1 <i>Start date</i>	Any way you can change values in V1	After V1 & V2 are brought to incompatible values, what are all the ways to display them?	After V1 & V2 are brought to incompatible values, what are all the ways to display or use them?	Variable 2 <i>End date</i>	<i>Constraint to a range</i>
Variable 2 <i>End date</i>	Any way you can change values in V2			Variable 1 <i>Start date</i>	<i>Constraint to a range</i>

Relationship Table

THE TABLE'S FIELDS

- Field: Create a row for each field (Consultant, End Date, and Start Date are examples of fields.)
- Entry Source: What dialog boxes can you use to enter data into this field? Can you import data into this field? Can data be calculated into this field? List every way to fill the field -- every screen, etc.
- Display: List every dialog box, error message window, etc., that can display the value of this field. When you re-enter a value into this field, will the new entry show up in each screen that displays the field? (Not always -- sometimes the program makes local copies of variables and fails to update them.)
- Print: List all the reports that print the value of this field (and any other functions that print the value).
- Related to: List every variable that is related to this variable. (What if you enter a legal value into this variable, then change the value of a constraining variable to something that is incompatible with this variable's value?)
- Relationship: Identify the relationship to the related variable.

Exploring relationships

- Given the relationship,
 - Try to enter relationship-breaking values everywhere that you can enter V1 and V2.
 - Pay attention to unusual entry options, such as editing in a display field, import, revision using a different component or program
- Once you achieve a mismatch between V1 and V2,
 - the program's data no longer obey rules the programmer expected would be obeyed, so anything that assumes the rules hold is vulnerable.
 - Do follow-up testing to discover serious side effects of the mismatch

Many relationships among data

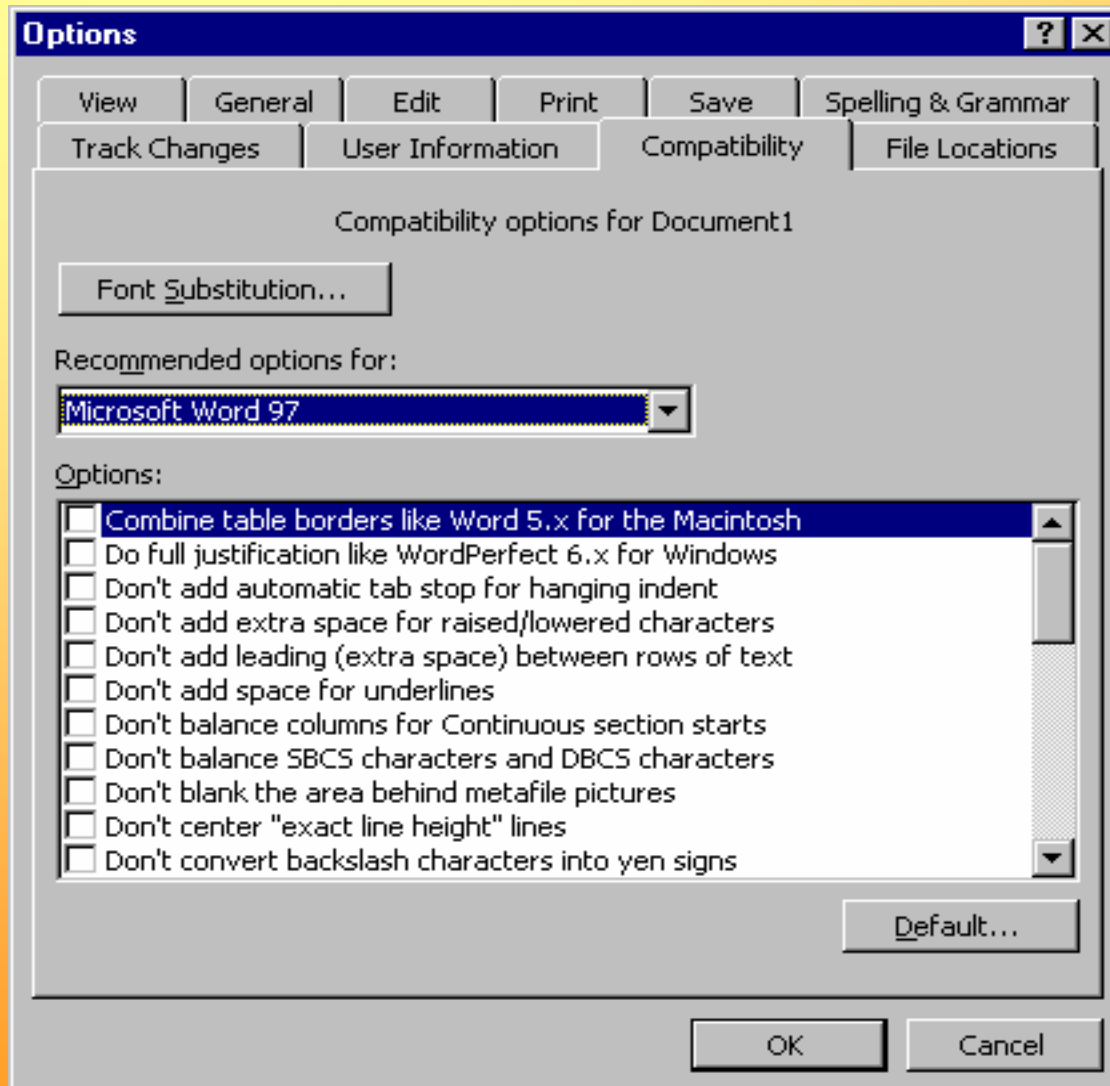
- Independence
 - Varying one has no effect on the permissible values of the other or on how the computer responds to a value of the other variable.
- Causal determination
 - By changing the value of one, we determine the value of the other. For example, in selecting page layout, if you select “Letter” the page becomes 8.5x11.
- Constrained to a range
 - For example, width of a line must be less than the width of the page.
 - In a date field, the max day is determined by the month
- Selection of rules
 - Example, hyphenation rules depend on the language you choose.
- Relations are often reciprocal, so if V2 constrains V1, then V1 might constrain V2 (try to change V2 after setting V1)

Many relationships

- Logical selection from a list
 - processes the value you entered and then figures out what value to use for the next variable. Example: timeouts in phone dialing:
 - 0 seconds on complete call 555-1212 but 95551212?
 - 10 seconds on ambiguous completion 955-5121
 - 30 seconds on incomplete 555-121
- Logical selection *of* a list:
 - For example, in printer setup, choose:
 - OfficeJet
 - get Graphics Quality, Paper Type, and Color Options
 - LaserJet 4
 - get Economode, Resolution, and Half-toning.

» Marick (Craft of Software Testing) discusses catalogs of tests for data relationships.

Complex Relationships



Data Relationship Table

- Looking at the Word options, you see the real value of the data relationships table. Many of these options have a lot of repercussions (they impact many features).
- You might analyze all of the details of all of the relationships later, but for now, it is challenging just to find out what all the relationships ARE.
- The table guides exploration and will surface a lot of bugs.

PROBLEM

- Works great for *this* release. Next release, what is your support for more exploration?

Let's sum up

There are several approaches to combination testing

- ***Mechanical (or procedural)***. The tester uses a routine procedure to determine a good set of tests
- ***Risk-based***. The tester combines test values (the values of each variable) based on perceived risks associated with noteworthy combinations
- ***Scenario-based***. The tester combines test values on the basis of interesting stories created for the combinations.

Mechanical approaches:

- Give you a handle on some complex problems
- Provide easy justification for management. *The number of tests needed is driven by theory and computed by the tool. Doesn't appear discretionary. This is an important difference from random testing.*
- Provide an intuitively appealing coverage model
- Appeal to the mathematically inclined
- Are rarely based on a plausible theory of risk. *(They're wasteful, however, if and only if a risk-based model would generate substantially different or fewer tests.)*