Testing Interactions Among Software Components

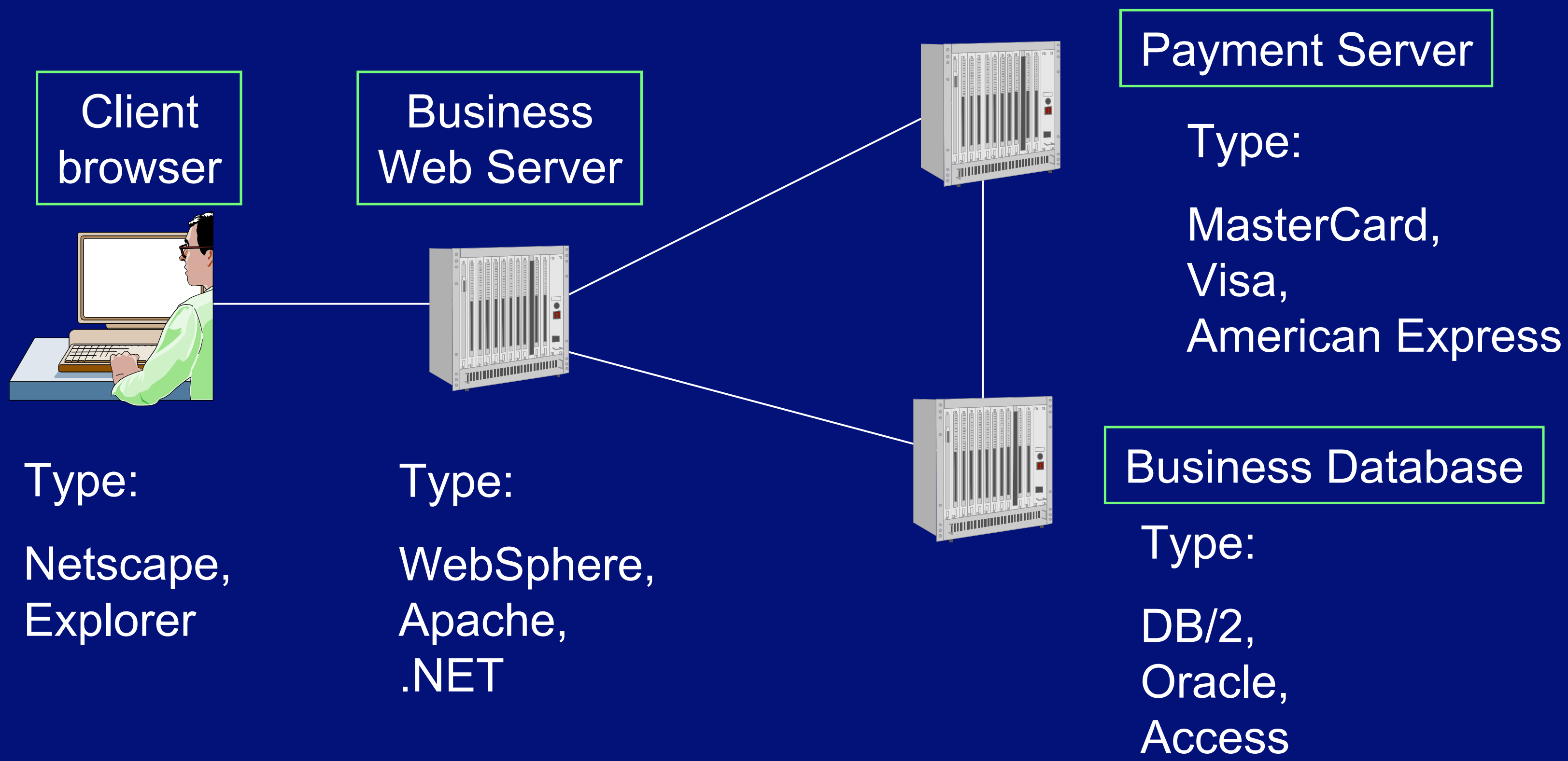
Alan Williams

School of Information Technology and Engineering,
University of Ottawa

awilliam@site.uottawa.ca

www.site.uottawa.ca/~awilliam

Component-Based Systems



- The goal: verify reliability and interoperability by testing as many system configurations as possible, given time and budget constraints.

Issues for Developers

- Can one software model be used for all deployment configurations?
- We could use configuration management, with a different version for each deployment configurations.
  - Drawback: maintenance of multiple models.
- Software modelling techniques do not take deployment to multiple environments into account, at the model level.
- Ideal: one model could generate code (or even be executable) for any deployment configuration
  - UML virtual machines?

Issues for Testers

- Assumption: Suppose we already have a “sufficient” test suite for a single configuration.
- If we do not have the resources to test all configurations, which ones should be selected for testing?
- If test cases are automated, they will need modification for a particular execution environment.

Selecting Test Configurations

- A well-known source of problems is components that function correctly on their own, but cause problems when interacting with other components.

- Strategy for selecting test configurations:
  - Maximize coverage of potential interactions

Objectives

1. Develop a measure that shows how well potential interactions are covered by a set of test configurations.

2. Determine how to achieve the highest interaction coverage with the fewest number of configurations.

Objectives

1. Develop a measure that shows how well potential interactions are covered by a set of test configurations.

2. Determine how to achieve the highest interaction coverage with the fewest number of configurations.

A test configuration is:

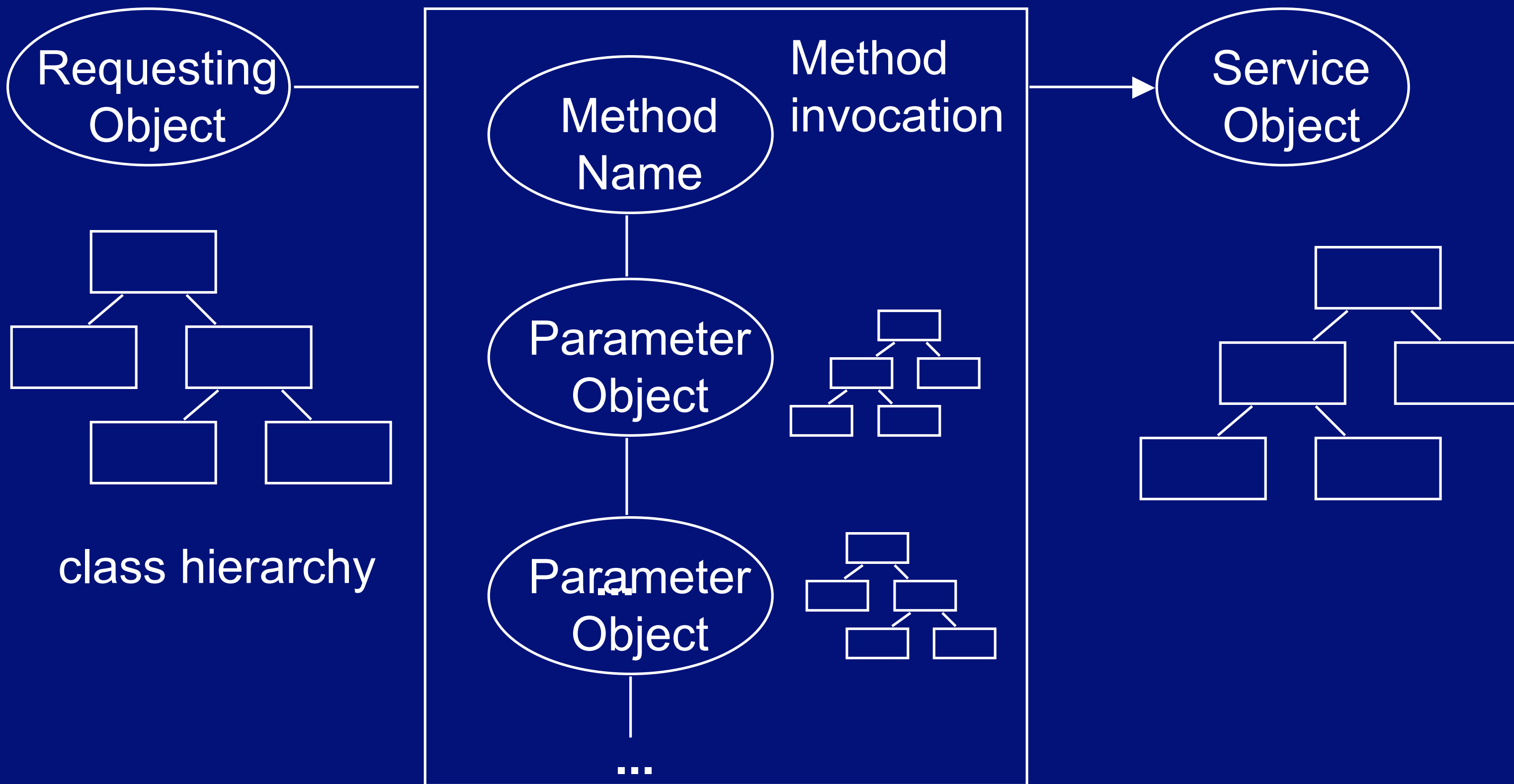
Browser:	Explorer
Server:	WebSphere
Payment:	Visa
Database:	DB/2

- For each component, one of the available types of that component is selected.
- The entire e-commerce application test suite is run for each configuration that is selected.

Formal Definition of General Problem

- Let p be the number of parameters (components).
  - Parameters are indexed $1,\ldots, p$.
- For each parameter i, suppose that there are n_i possible values (component types).
  - Each parameter can take a value v_i, where $1 \leq v_i \leq n_i$
- Assumption: parameters are independent.
  - The choice of values for any parameter does not affect the choice of values for any other parameter.
  - Dependencies among parameters can be resolved by creating a hybrid parameter that enumerates all legal combinations.

Testing in the Presence of Polymorphism and Inheritance



- Requester, service, and message parameter objects could be instances of various classes within the class hierarchy.

Testing of Parameter Equivalence Classes

- Suppose we have:

  `method( a, b, c, d, e )`

- Determine equivalence classes for each parameter
  - Example: for **`a`**, we might have:

    `[-∞, -4][-3,-1][0][1,12][13,+∞]`

  - Select a representative value from each equivalence class
- To test if parameters affect each other, we need to select combinations of equivalence classes
  - Desirable if decision conditions involve more than one parameter

An interaction element is:

Browser:	Explorer
Server:	
Payment:	Visa
Database:	

- Choose a subset of the parameters:
  - The size of the subset is the interaction degree.
- Choose specific values for the parameters.

Example

- Suppose that we have three parameters.
- For each parameter, there are two possible values.
  - Values are :
    - A, B for parameter 1.
    - J, K for parameter 2.
    - Y, Z for parameter 3.
- Degree of interaction coverage is 2.
  - We want to cover all potential 2-way interactions among parameter values.

Set of all possible test configurations

Three parameters, two values for each.

There are $2^3 = 8$ possible test configurations.

A	J	Y
A	J	Z
A	K	Y
A	K	Z
B	J	Y
B	J	Z
B	K	Y
B	K	Z

Set of all possible degree 2 interaction elements

There are $\binom{3}{2} \times 2^{2} = 12$ possible interaction elements.

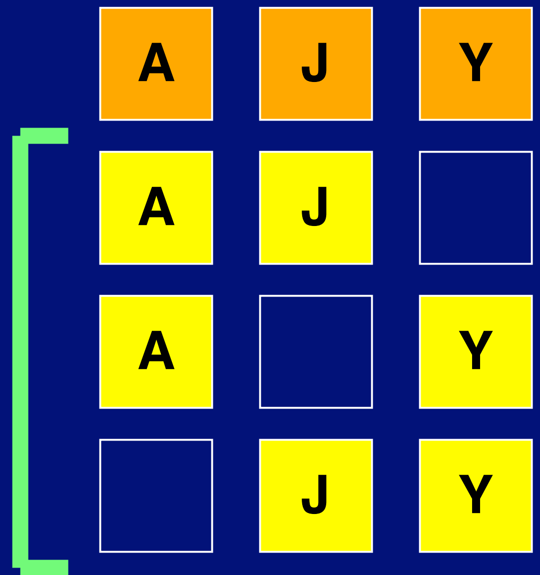
A	J		A		Y		J	Y
A	K		A		Z		J	Z
B	J		B		Y		K	Y
B	K		B		Z		K	Z

- Coverage measure:
  - Percentage of interaction elements covered.

Test configurations as sets of interactions

One test configuration...

… covers 3 possible interaction elements.



Interaction test coverage goal

A	J			A		Y			J	Y
A	K			A		Z			J	Z
B	J			B		Y			K	Y
B	K			B		Z			K	Z

A	J	Y
A	J	Z
A	K	Y
A	K	Z
B	J	Y
B	J	Z
B	K	Y
B	K	Z

Goal: cover all interaction elements…

…using a subset of all test configurations.

Selection of test configurations for coverage of interaction elements

Interaction elements

A	J		A		Y		J	Y
A	K		A		Z		J	Z
B	J		B		Y		K	Y
B	K		B		Z		K	Z

Degree 2 coverage: 3 / 12 = 25%

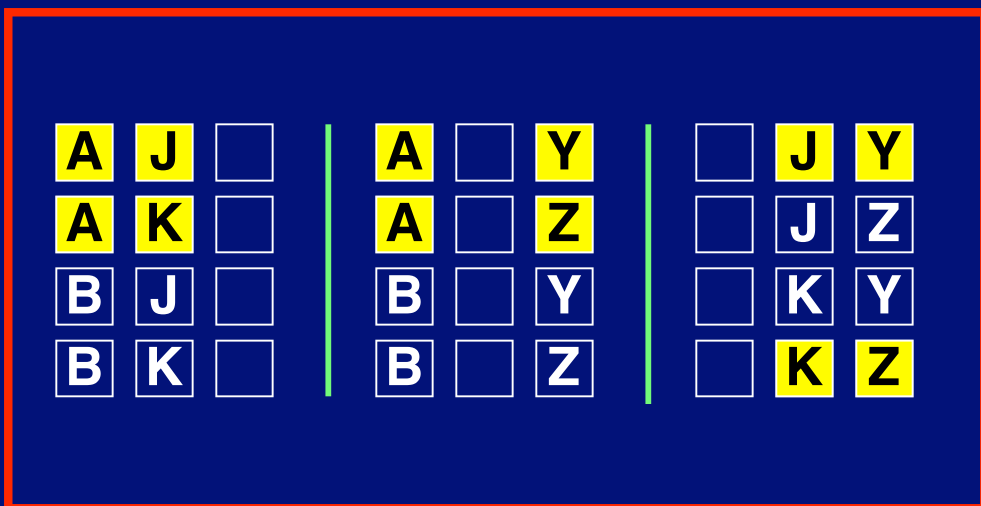
Degree 3 coverage: 1 / 8 = 12.5%

Test configurations

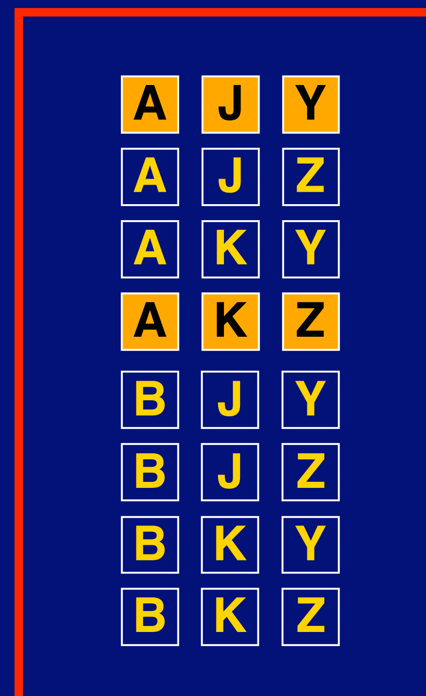
A	J	Y
A	J	Z
A	K	Y
A	K	Z
B	J	Y
B	J	Z
B	K	Y
B	K	Z

Selection of test configurations for coverage of interaction elements

Interaction elements



Test configurations



Degree 2 coverage: 6 / 12 = 50%

Degree 3 coverage: 2 / 8 = 25%

Selection of test configurations for coverage of interaction elements

Interaction elements

A	J		A		Y		J	Y
A	K		A		Z		J	Z
B	J		B		Y		K	Y
B	K		B		Z		K	Z

Test configurations

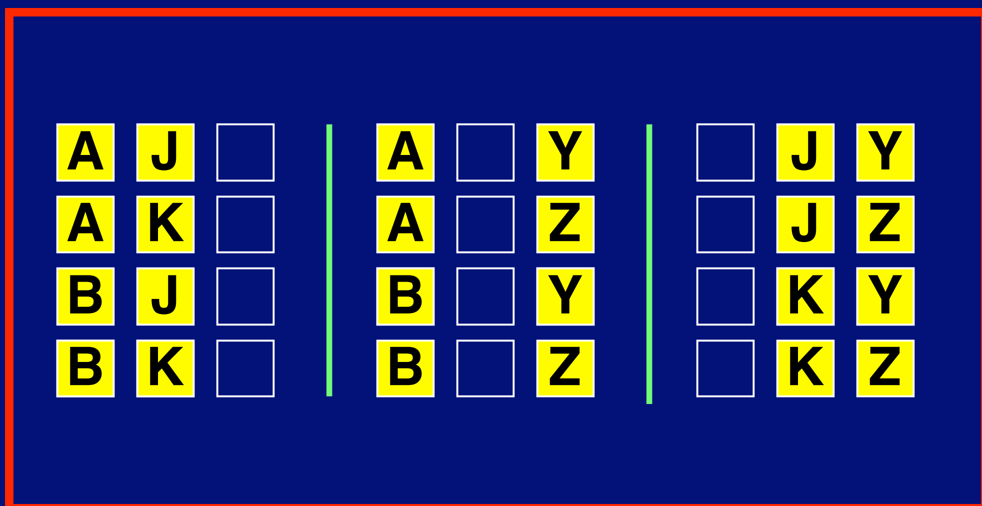
A	J	Y
A	J	Z
A	K	Y
A	K	Z
B	J	Y
B	J	Z
B	K	Y
B	K	Z

Degree 2 coverage: 9 / 12 = 75%

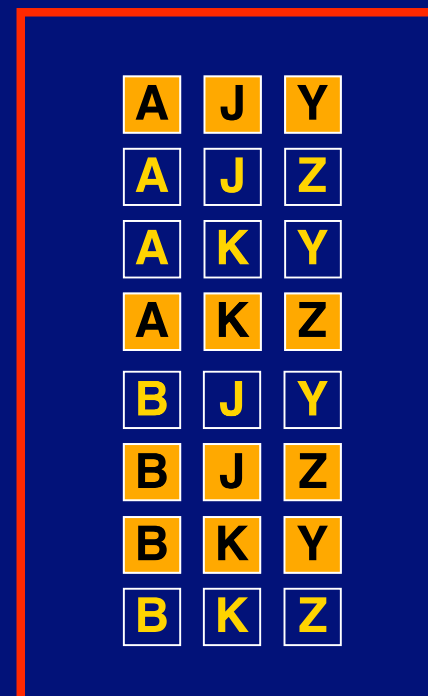
Degree 3 coverage: 3 / 8 = 37.5%

Selection of test configurations for coverage of interaction elements

Interaction elements



Test configurations



Degree 2 coverage: 12 / 12 = 100%

Degree 3 coverage: 4 / 8 = 50%

Choosing the degree of coverage

- In one experiment, covering 2 way interactions resulted in the following average code coverage:
  - 93% block coverage.
  - 83% decision coverage.
  - 76% c-use coverage.
  - 73% p-use coverage.
  - Source: Cohen, et al, "The combinatorial design approach to automatic test generation", *IEEE Software*, Sept. 1996.
- Another experience report investigating interactions among 2-4 components:
  - Dunietz, et al, "Applying design of experiments to software testing", *Proc. Of ICSE '97*.

Section summary

- We have defined how to measure coverage of potential system interactions.
- Strategy for choosing test configurations:
  - Maximize coverage of interaction elements for a given degree.
- Choose interaction degree based on:
  - Degree of interaction risk that can be tolerated.
  - Test budget constraints.

Objectives

1. Develop a measure that shows how well potential interactions are covered by a set of test configurations.

2. Determine how to achieve the highest interaction coverage with the fewest number of configurations.

Constraint-based approach

			AJY	AJZ	AKY	AKZ	BJY	BJZ	BKY	BKZ	
A	J		x_1	$+ x_2$							? 1
A		Y	x_1		$+ x_3$						? 1
	J	Y	x_1				$+ x_5$				? 1
A	K				x_3	$+ x_4$					? 1
A		Z		x_2		$+ x_4$					? 1
	J	Z		x_2				$+ x_6$			? 1
B	J						x_5	$+ x_6$			? 1
B		Y					x_5		$+ x_7$		? 1
	K	Y			x_3				$+ x_7$		? 1
B	K								x_7	$+ x_8$	? 1
B		Z						x_6		$+ x_8$	? 1
	K	Z				x_4				$+ x_8$	? 1

- Minimize:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8; \qquad x_i \in \{0,1\}$$

Running the problem as a {0,1} integer program

# parms	# values per parm	# constraints	# variables	Result: # configs	Run time (s)
3	2	12	8	4	<0.01
4	2	24	16	5	0.01
5	2	40	32	6	0.70
6	2	60	64	6	16.57
7	2	84	128	6	441.21
4	3	54	81	9	0.08
5	3	90	243	13*	*

* process killed after running for 6.5 hours

- Solution of {0,1} integer programs is an NP-complete problem

A linear programming approximation

```
Value of objective function: 9
x043      0.530351
x047      0.401344
x117      0.389934
x201      0.323851
x069      0.302597
x241      0.301478
x213      0.286215
x003      0.279507
x176      0.258343
x224      0.25652
x181      0.249479
x107      0.248555
x159      0.247746
x166      0.247169
x130      0.244542
x233      0.243054
x056      0.238081
x113      0.235705
x136      0.234124
x092      0.228301
x099      0.226595
x009      0.173559
x026      0.172155
x013      0.167052
x194      0.165232
x145      0.155093
x058      0.153223
x149      0.152299
x228      0.146957
x156      0.135725
x190      0.129007
x081      0.123935
x100      0.123432
x024      0.120402
x094      0.119709
x119      0.118003
x165      0.11396
x077      0.101096
x005      0.087325
x189      0.0859768
x073      0.0810688
x142      0.0750125
x031      0.0683057
x215      0.0678995
x088      0.0480873
x169      0.0450714
x222      0.04304
x199      0.0277951
x116      0.0118162
x236      0.00895104
x083      0.00532082
```

- 5 parameters, 3 values for each.
- Value of objective function can be achieved by setting $\{x_{43}, x_{47}, x_{117}, x_{201}, x_{69}, x_{241}, x_{213}, x_3, x_{176}, x_{224}, x_{181}, x_{107}, x_{159}, x_{166}, x_{130}, x_{56}, x_{136}, x_{92}\}$ to 1, and the rest to 0.
- However, this results in 18 configurations instead of the (fewest known) 11.

Statistical Experimental Design

- Used in many fields other than computer science.
- Objective:
  - Create an experiment to test several factors at once.
    - Individual effect of each factor.
    - Interactions among factors.
  - Minimize the number of experiments needed.
  - Facilitate result analysis.
- Application to software system testing:
  - Can be used in any situation where there are a set of parameters, each of which have a set of (discrete) values.

Orthogonal Arrays

#	Caller	Market	Type	Called
1	Regular	Canada	Local	Regular
2	Regu			

- Orthogonal arrays are a standard construction used for statistical experiments.
- Strength 2: select any 2 columns and all ordered pairs occur the same number of times.
  - Covers all 2-way interactions.
- Orthogonal arrays can be found in statistical tables, or can be calculated from algebraic finite fields.
  - Many existence restrictions.

Adaptation to Software Testing

- If we are testing strictly for software interactions, we can use a smaller experimental design.
- Why?
  - If each component has been tested on its own, we can eliminate the need for testing for the effect of a single parameter.
  - Software testing yields a discrete test result ("pass" or "fail"), rather than requiring result analysis of real valued results.
- The result:
  - Each interaction needs to be covered at least once, instead of the same number of times.
  - Fewer configurations are required.
  - The construction for this purpose is called a covering array.

Covering Arrays

- Definition of covering array:
  - If we select d columns, all possible ordered d-tuples occur at least once.
- A covering array of strength d will ensure than any consistent interaction problem caused by a particular combination of two elements is detected.
- Problems caused by an interaction of $d+1$ (or more) elements may not be detected.
- Choosing the degree of coverage defines the trade-off in risk we are making:
  - Fewer test configurations versus potential uncovered interactions.

Recursive Covering Array Construction

- Problem:
  - If the range of values is $1, \ldots, n$, then an orthogonal array can handle at most $n + 1$ parameters.
  - Existence of suitable orthogonal arrays.
- Goal:
  - Generate covering arrays for problems of arbitrary size.
- Method:
  - Assemble larger covering array from smaller building blocks.
  - No heuristics.

Constructing Large Covering Arrays

1	1	1	1
1	2	2	2
1	3	3	3
2	1	2	3
2	2	3	1
2	3	1	2
3	1	3	2
3	2	1	3
3	3	2	1

With 3 values per parameter, an orthogonal array can handle up to 4 parameters.

Constructing Large Covering Arrays

1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	2	2	2	1	2	2	2
1	3	3	3	1	3	3	3	1	3	3	3
2	1	2	3	2	1	2	3	2	1	2	3
2	2	3	1	2	2	3	1	2	2	3	1
2	3	1	2	2	3	1	2	2	3	1	2
3	1	3	2	3	1	3	2	3	1	3	2
3	2	1	3	3	2	1	3	3	2	1	3
3	3	2	1	3	3	2	1	3	3	2	1

Duplicate orthogonal array three times
for 12 parameters …

Constructing Large Covering Arrays

1	1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	2	2	2	1	2	2	2	2
1	3	3	3	1	3	3	3	1	3	3	3	3
2	1	2	3	2	1	2	3	2	1	2	3	3
2	2	3	1	2	2	3	1	2	2	3	1	1
2	3	1	2	2	3	1	2	2	3	1	2	2
3	1	3	2	3	1	3	2	3	1	3	2	2
3	2	1	3	3	2	1	3	3	2	1	3	3
3	3	2	1	3	3	2	1	3	3	2	1	1

Check coverage so far:

For the first column...

Constructing Large Covering Arrays

1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	2	2	2	1	2	2	2
1	3	3	3	1	3	3	3	1	3	3	3
2	1	2	3	2	1	2	3	2	1	2	3
2	2	3	1	2	2	3	1	2	2	3	1
2	3	1	2	2	3	1	2	2	3	1	2
3	1	3	2	3	1	3	2	3	1	3	2
3	2	1	3	3	2	1	3	3	2	1	3
3	3	2	1	3	3	2	1	3	3	2	1

… we have pair-wise coverage
with the rest of the orthogonal array
(by definition) …

Constructing Large Covering Arrays

1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	2	2	2	1	2	2	2
1	3	3	3	1	3	3	3	1	3	3	3
2	1	2	3	2	1	2	3	2	1	2	3
2	2	3	1	2	2	3	1	2	2	3	1
2	3	1	2	2	3	1	2	2	3	1	2
3	1	3	2	3	1	3	2	3	1	3	2
3	2	1	3	3	2	1	3	3	2	1	3
3	3	2	1	3	3	2	1	3	3	2	1

… but we also have pair-wise coverage with the corresponding columns in the duplicate orthogonal arrays.

Constructing Large Covering Arrays

1	1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	2	2	2	1	2	2	2	2
1	3	3	3	1	3	3	3	1	3	3	3	3
2	1	2	3	2	1	2	3	2	1	2	3	3
2	2	3	1	2	2	3	1	2	2	3	1	1
2	3	1	2	2	3	1	2	2	3	1	2	2
3	1	3	2	3	1	3	2	3	1	3	2	2
3	2	1	3	3	2	1	3	3	2	1	3	3
3	3	2	1	3	3	2	1	3	3	2	1	1

We have also covered the
(x,x) combinations in the identical columns,
but not the (x,y) combinations.

Constructing Large Covering Arrays

1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	2	2	2	1	2	2	2
1	3	3	3	1	3	3	3	1	3	3	3
2	1	2	3	2	1	2	3	2	1	2	3
2	2	3	1	2	2	3	1	2	2	3	1
2	3	1	2	2	3	1	2	2	3	1	2
3	1	3	2	3	1	3	2	3	1	3	2
3	2	1	3	3	2	1	3	3	2	1	3
3	3	2	1	3	3	2	1	3	3	2	1

Use reduced array, which covers only the (x,y) combinations

This is the original orthogonal array, with the first 3 rows and first column removed.

1	2	3
2	3	1
3	1	2
1	3	2
2	1	3
3	2	1

Constructing Larger Covering Arrays

1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	2	2	2	1	2	2	2
1	3	3	3	1	3	3	3	1	3	3	3
2	1	2	3	2	1	2	3	2	1	2	3
2	2	3	1	2	2	3	1	2	2	3	1
2	3	1	2	2	3	1	2	2	3	1	2
3	1	3	2	3	1	3	2	3	1	3	2
3	2	1	3	3	2	1	3	3	2	1	3
3	3	2	1	3	3	2	1	3	3	2	1
1				2				3			
2				3				1			
3				1				2			
1				3				2			
2				1				3			
3				2				1			

… and add new configurations to cover missing combinations.

Constructing Large Covering Arrays

1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	2	2	2	1	2	2	2
1	3	3	3	1	3	3	3	1	3	3	3
2	1	2	3	2	1	2	3	2	1	2	3
2	2	3	1	2	2	3	1	2	2	3	1
2	3	1	2	2	3	1	2	2	3	1	2
3	1	3	2	3	1	3	2	3	1	3	2
3	2	1	3	3	2	1	3	3	2	1	3
3	3	2	1	3	3	2	1	3	3	2	1
1				2				3			
2				3				1			
3				1				2			
1				3				2			
2				1				3			
3				2				1			

This covers the remaining combinations for the first column.

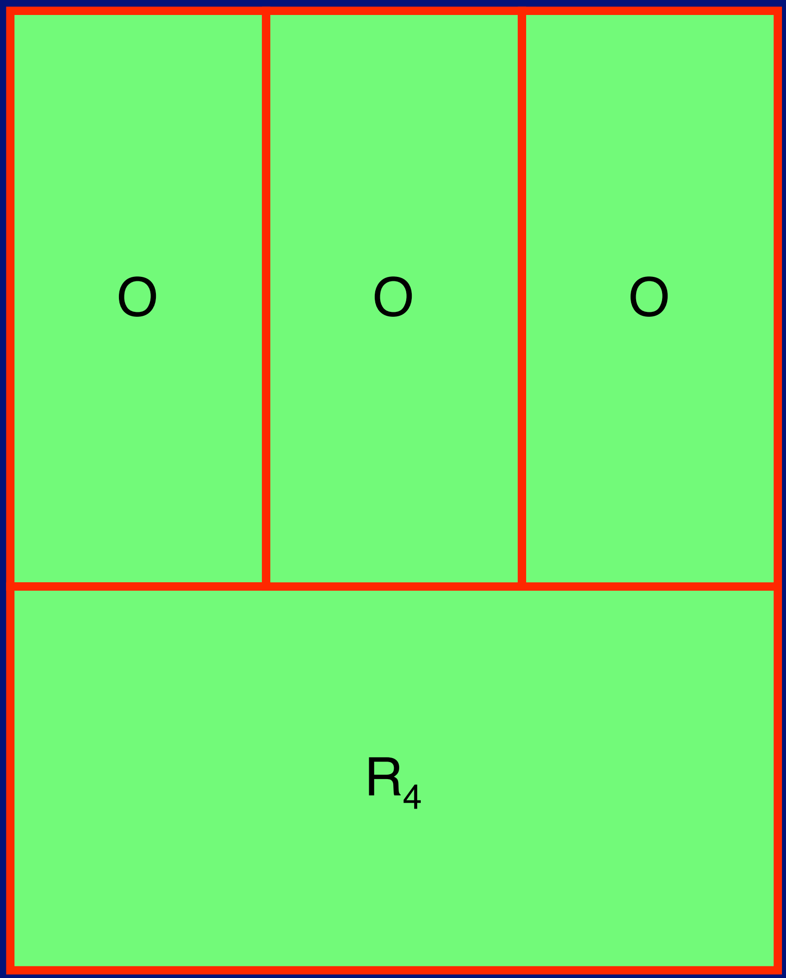
Constructing Larger Covering Arrays

1	1	1	1	1	1	1	1	1	1	1	1
1	2	2	2	1	2	2	2	1	2	2	2
1	3	3	3	1	3	3	3	1	3	3	3
2	1	2	3	2	1	2	3	2	1	2	3
2	2	3	1	2	2	3	1	2	2	3	1
2	3	1	2	2	3	1	2	2	3	1	2
3	1	3	2	3	1	3	2	3	1	3	2
3	2	1	3	3	2	1	3	3	2	1	3
3	3	2	1	3	3	2	1	3	3	2	1
1	1	1	1	2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3	1	1	1	1
3	3	3	3	1	1	1	1	2	2	2	2
1	1	1	1	3	3	3	3	2	2	2	2
2	2	2	2	1	1	1	1	3	3	3	3
3	3	3	3	2	2	2	2	1	1	1	1

The same scheme applies to other columns.

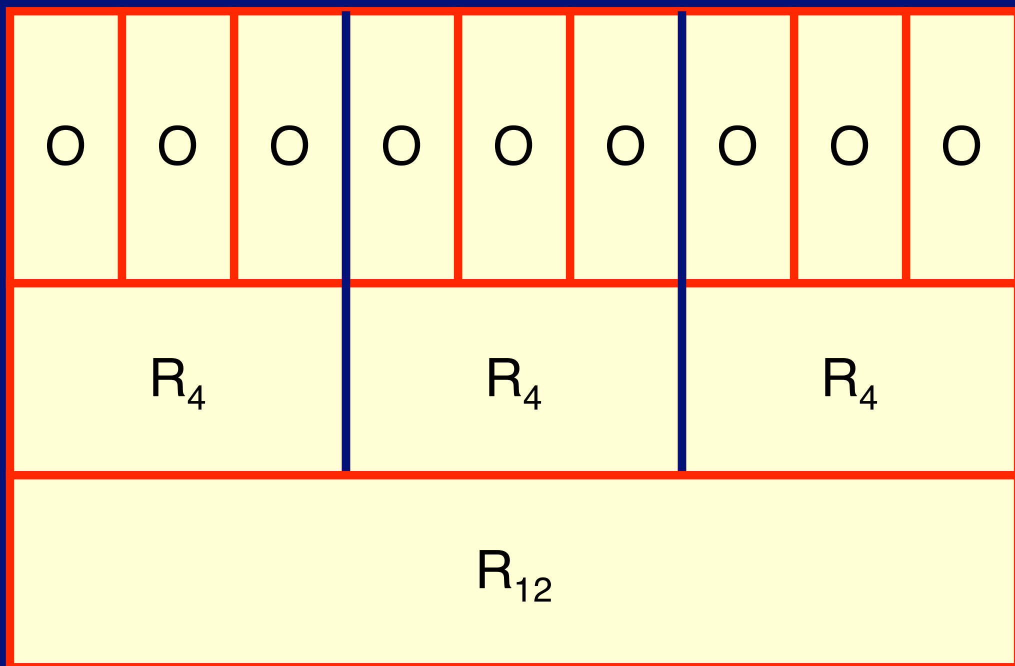
All pair-wise combinations have now been covered

Constructing Larger Covering Arrays



- R_4: columns are duplicated 4 times consecutively

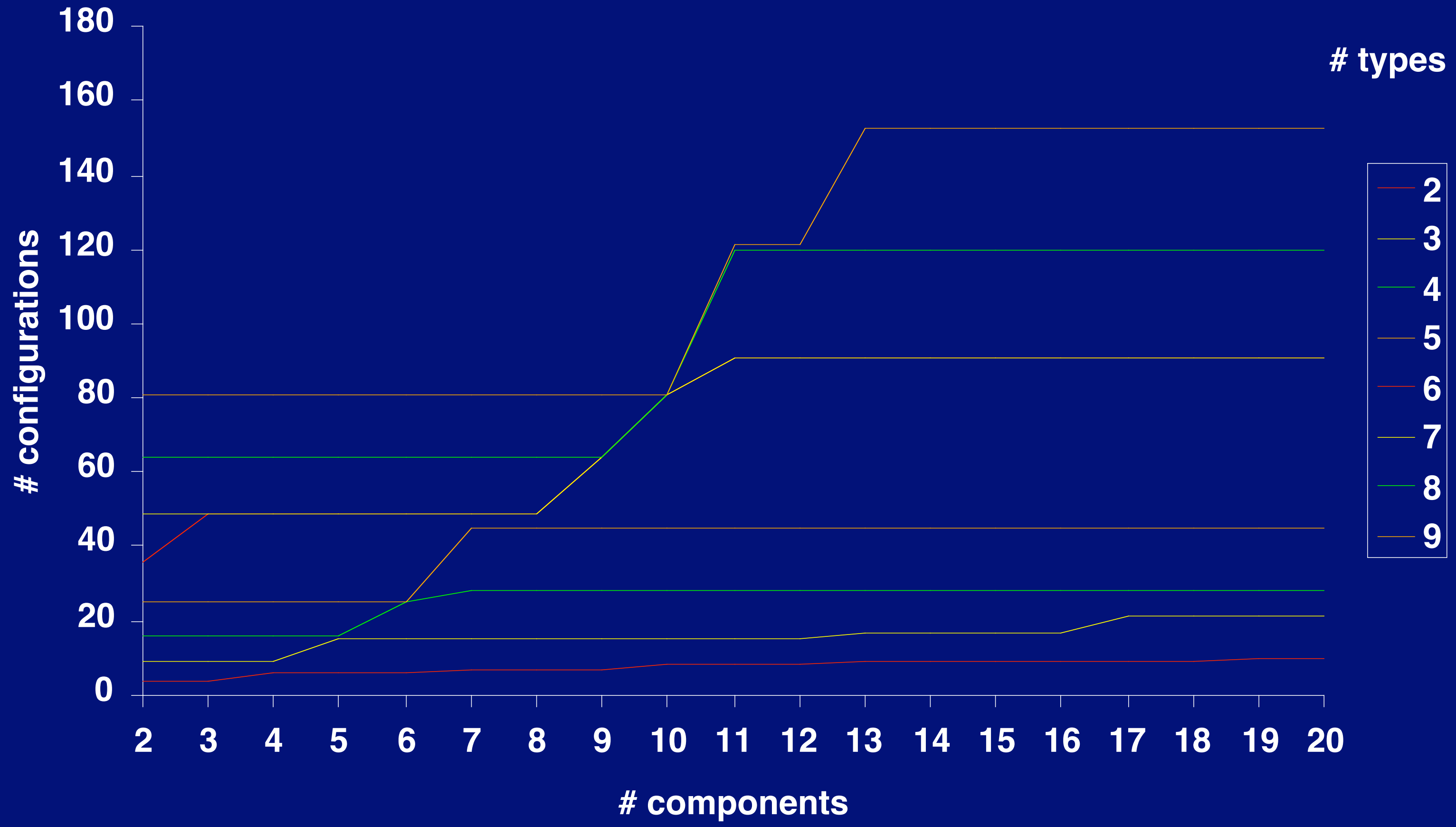
Multistage Covering Arrays



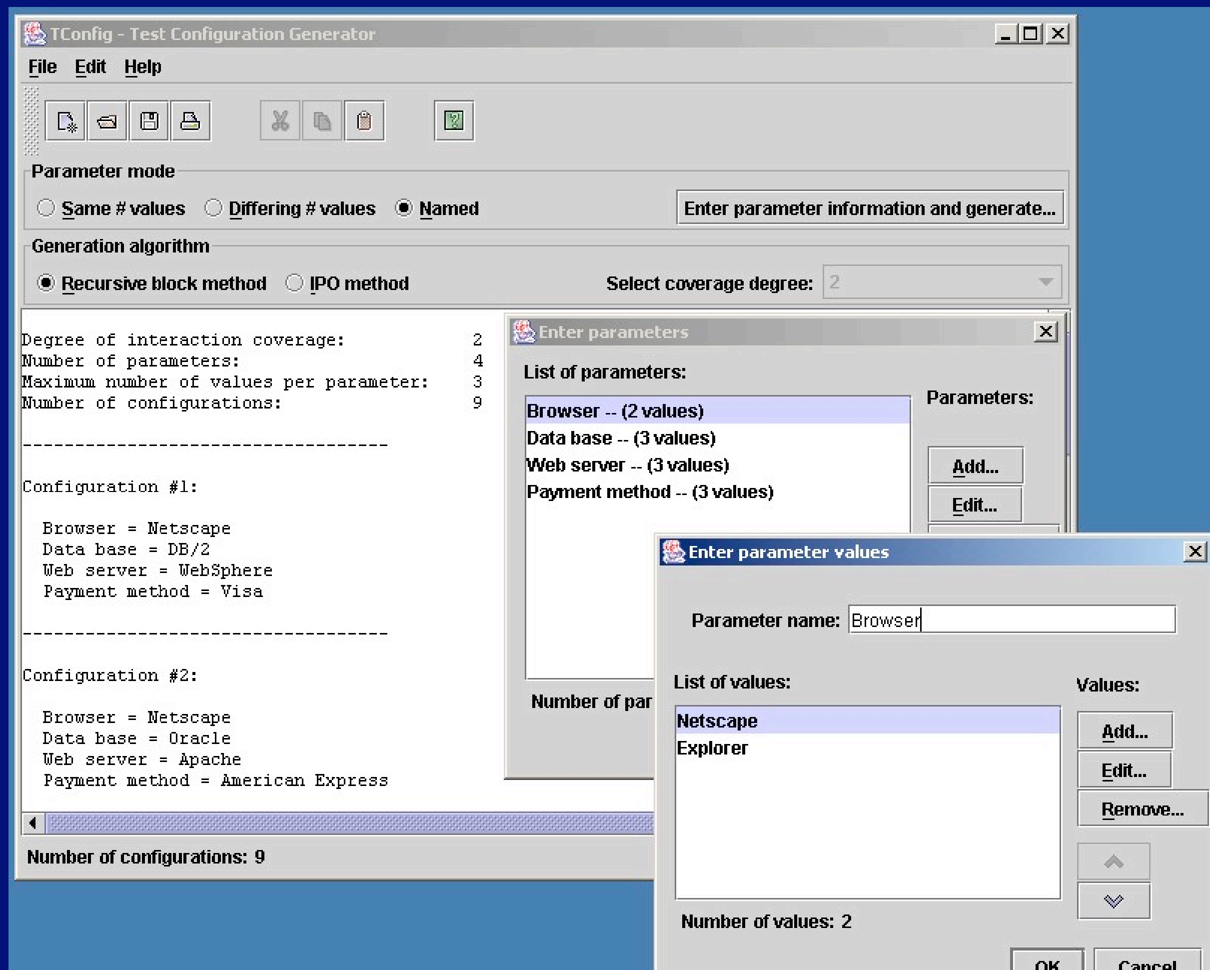
Some results

- Results from the recursive construction example:
  - 13 components, 3 types for each component.
  - Number of potential test configurations: 1,594,323.
  - Number of degree 2 interaction elements: 702.
  - Minimum number of configurations for 100% coverage of degree 2 interaction elements: 15.
- Achieving coverage of interaction elements results in a number of test configurations that is proportional to.
  - The logarithm of the number of components.
  - The maximum number of types for any component, raised to the power of the interaction coverage degree.

Number of configurations to cover interaction elements of degree 2

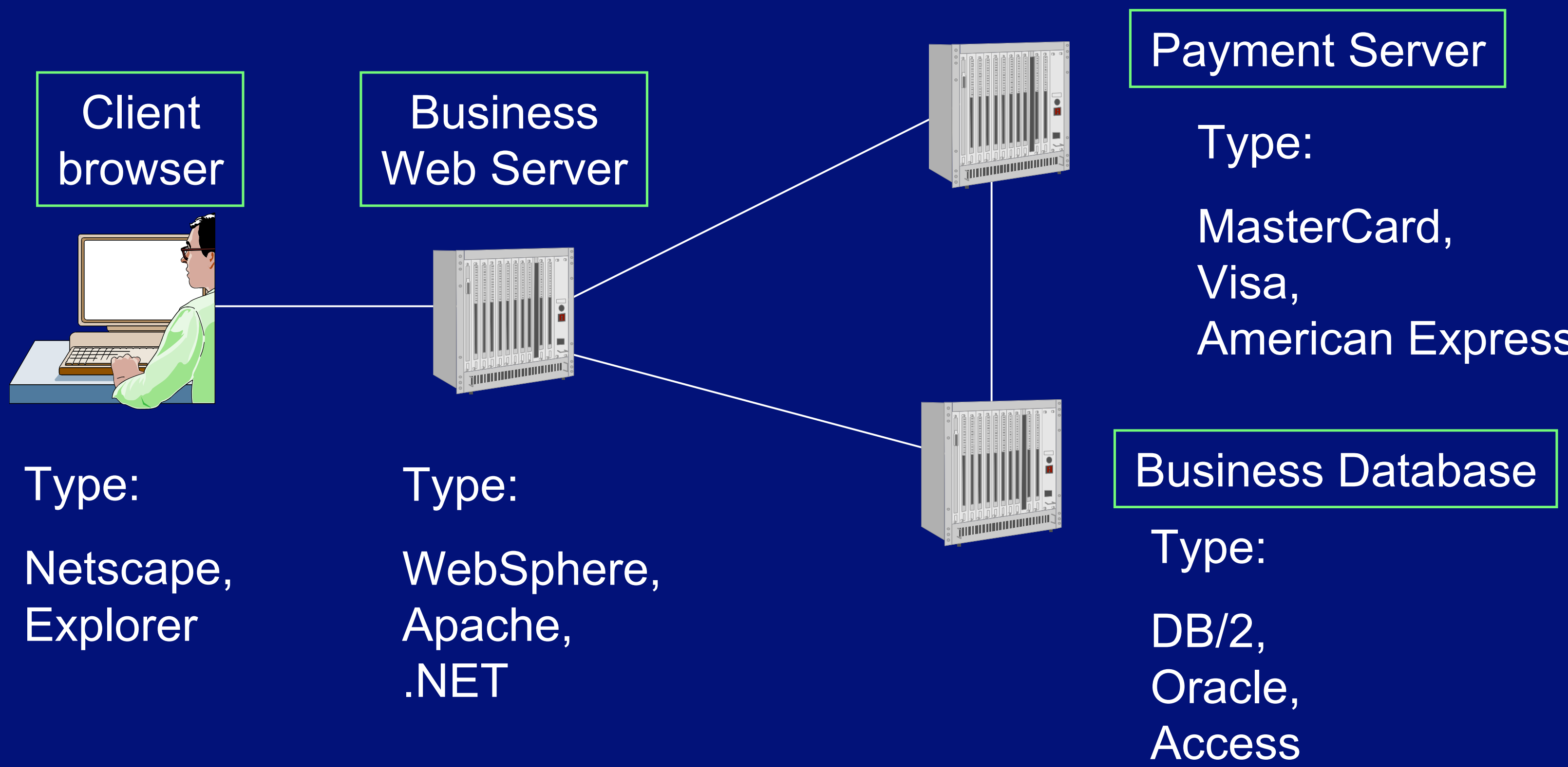


TConfig: Test configuration generator



Try it: www.site.uottawa.ca/~awilliam/TConfig.jar

Our example again...



Test configurations for degree 2 coverage

Configuration	Browser	Web Server	Payment	Data Base
1	Netscape	WebSphere	MasterCard	DB/2
2	Netscape	Apache	Visa	Oracle
3	Netscape	.NET	AmEx	Access
4	Explorer	WebSphere	Visa	Access
5	Explorer	Apache	AmEx	DB/2
6	Explorer	.NET	MasterCard	Oracle
7	(don't care)	WebSphere	AmEx	Oracle
8	(don't care)	Apache	MasterCard	Access
9	(don't care)	.NET	Visa	DB/2

Comments from testers:

- Pre-existing regression test suites:
  - "I already have a collection of tests that are working fine, and have been developed at great expense. How do I determine which additional tests need to be added to bring the test suite to a certain level of interaction coverage?"
- Ensuring that desired test configurations are included by the generation method:
  - "A specific set of test configurations are recommended to customers. We want to make sure those configurations are covered."
- Changes in set of allowed parameters and values:
  - "What additional configurations are required if…
    - … a new component is added to the system?"
    - … a new version of an existing component becomes available?"

The road ahead

- Now that we know which test configurations to select, is there a way to automatically modify test scripts for each configuration?
- Design model to support deployment to multiple environments…
  - Build deployment into modelling notations?
  - Virtual machines for execution?

Thank you!

- This presentation is available at:
  - http://www.site.uottawa.ca/~awilliam