Software Product Line Testing Part III : Interactions

Myra Cohen

Matthew Dwyer

Laboratory for Empirically-based Software Quality Research
Department of Computer Science
University of Nebraska - Lincoln

Work supported by NSF CCF through awards 0429149 and 0444167, by the U.S. Army Research Office through award DAAD190110564 and by an NSF EPSCoR FIRST award.

e2 ESQuaReD

University of Nebraska Lincoln

Outline

- Software Product Lines : What and Why?
- Modeling Variability in Software Product Lines
- Validating Product Lines
- A Framework for Variability Coverage
- Toward Product Line Driven Test Processes

Outline

Validating Product Lines

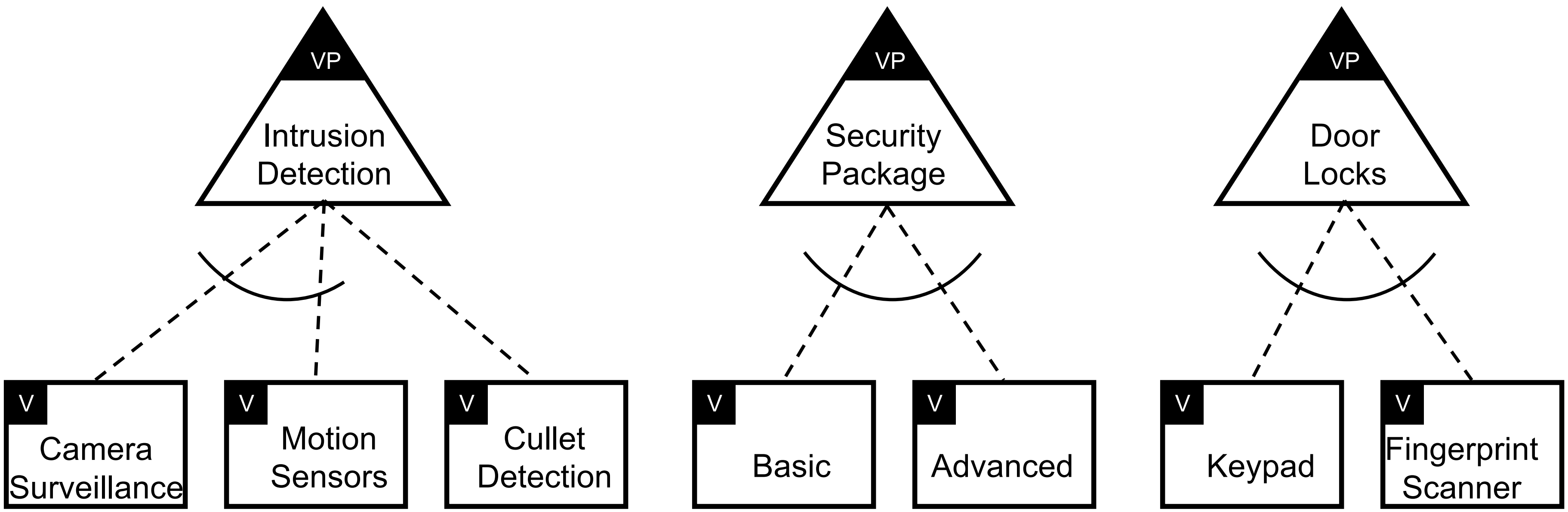
1. Introduction
2. A Motivating Example
3. Combinatorial Interaction Testing

The Meaning of Validation

A program is validated if we have confidence that it will operate correctly.

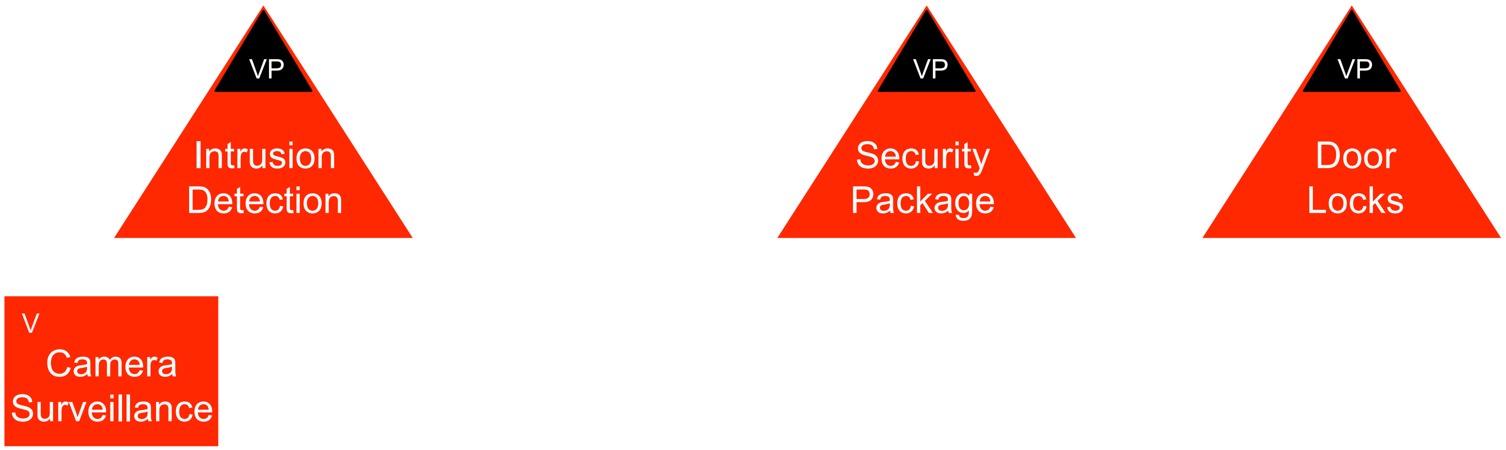
A software product line is validated if we have confidence that any instance of that produce line will operate correctly.

Validating the Instance

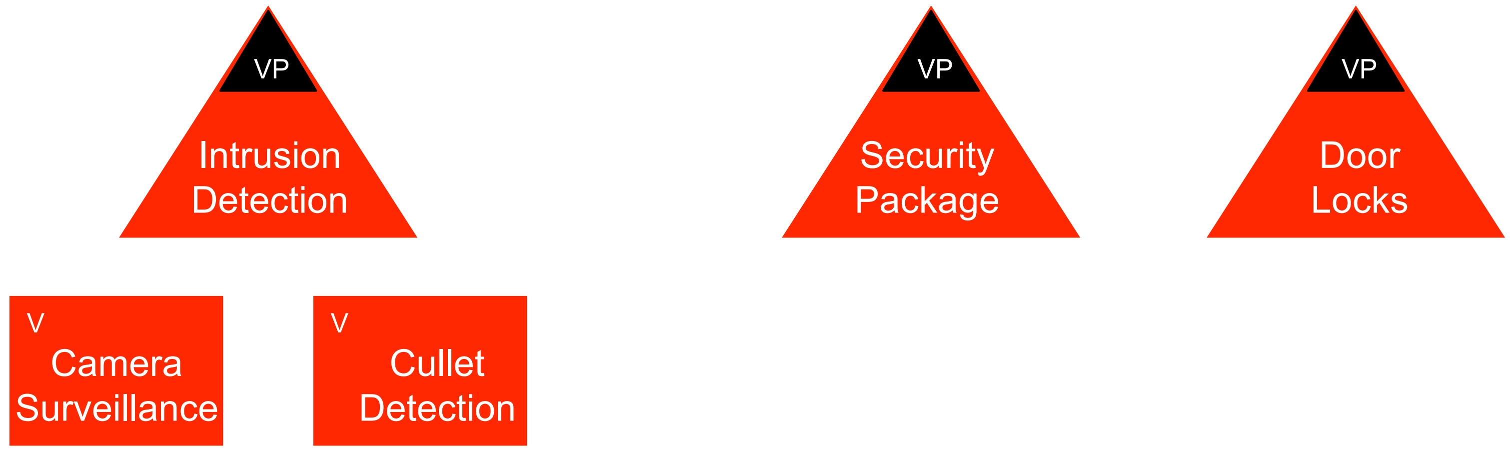


Validating the Instance

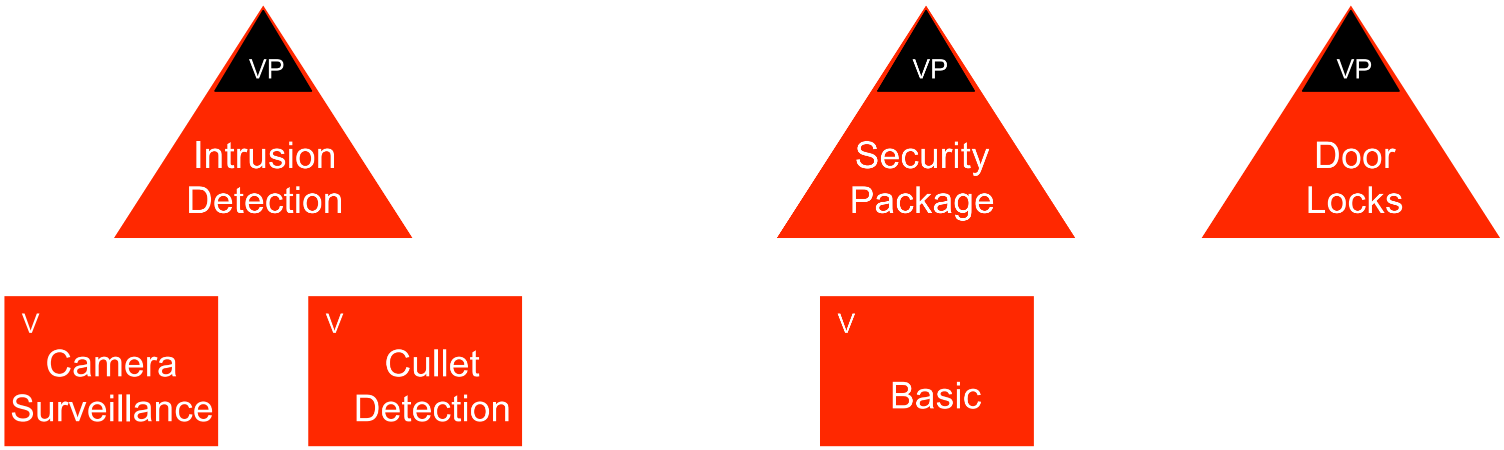
Validating the Instance



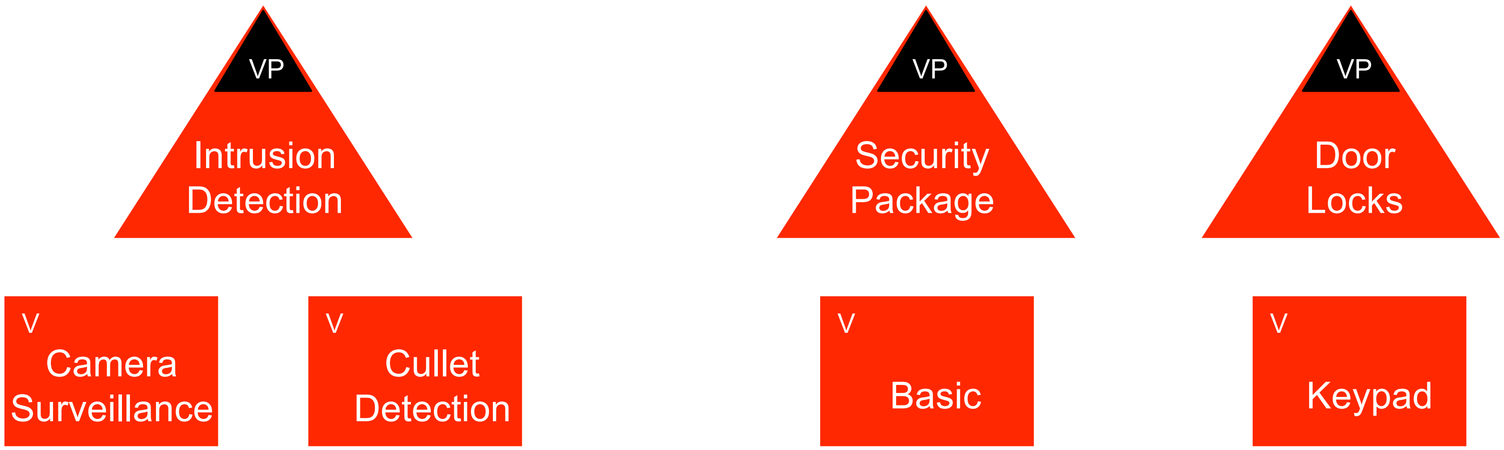
Validating the Instance



Validating the Instance

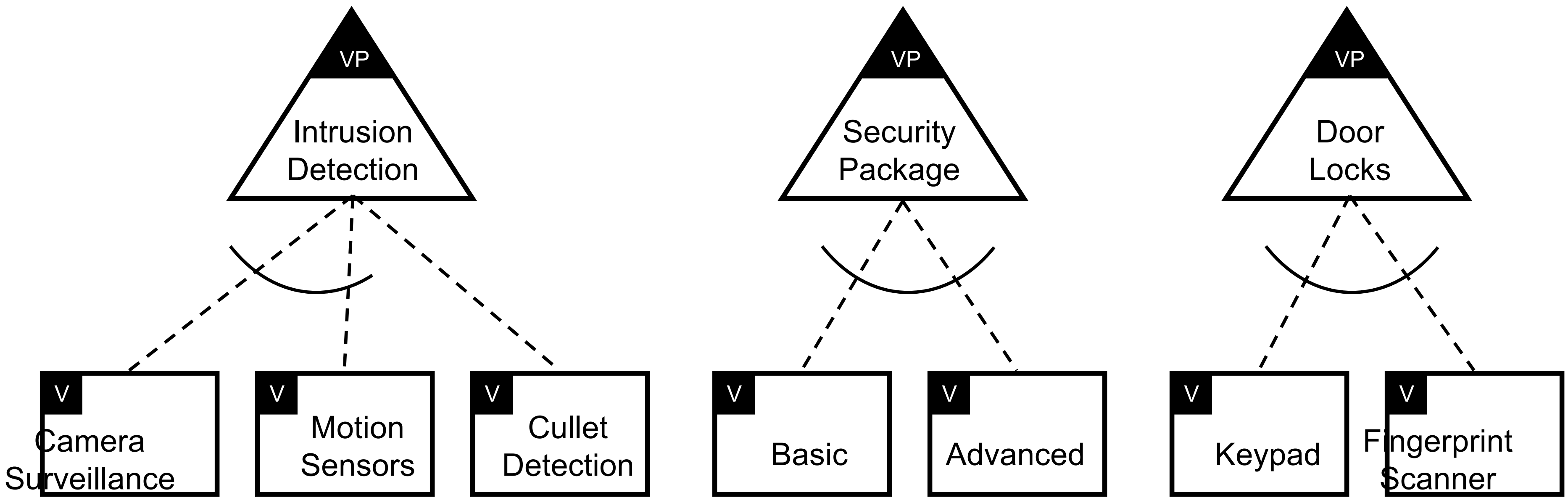


Validating the Instance



- Can we re-use tests across different instances?

Validating the Product Line



- Focus is on testing the product line as a whole.

Testing An SPL

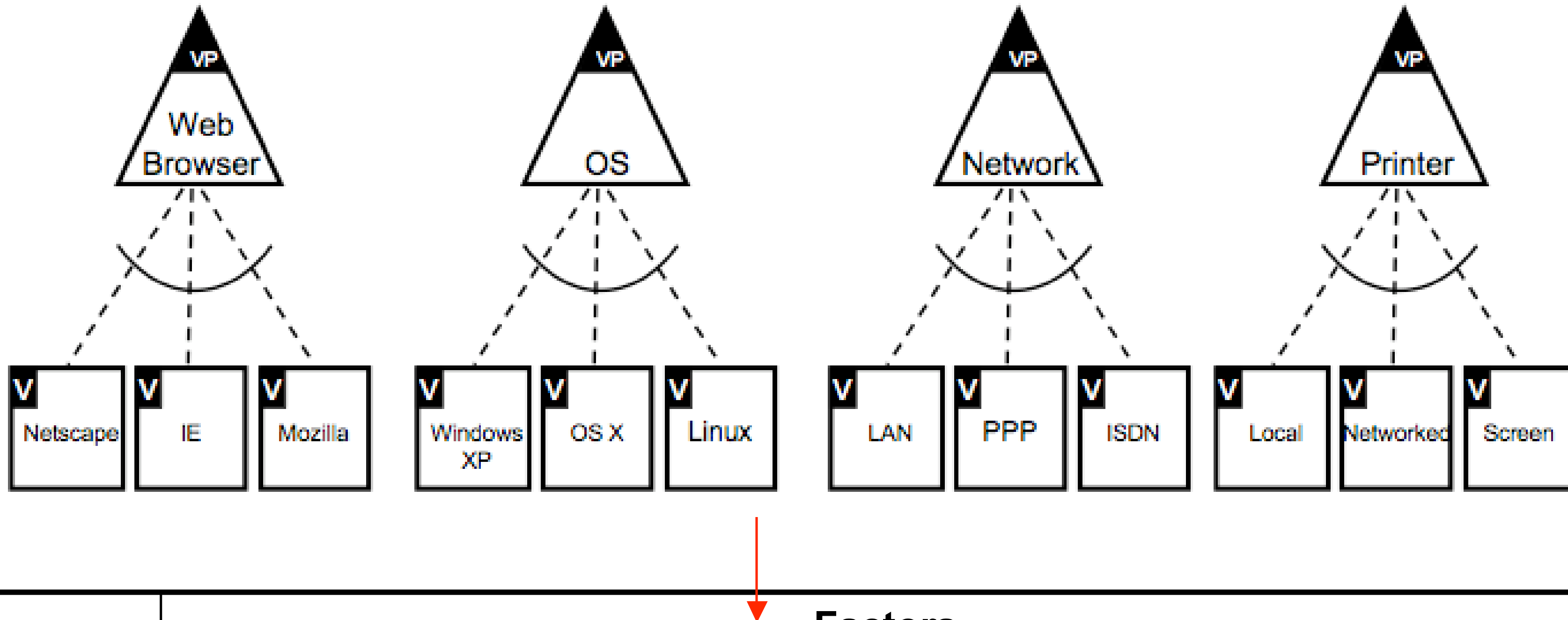
- Much of the current research on testing SPLs focuses on testing individual instances and reuse of specific test cases.
- Our assumption is that this problem has been solved and good test cases have been developed.
- We add a second layer of complexity and focus on the entire product line.

Outline : Interactions

Validating Product Lines

1. Introduction
2. A Motivating Example
3. Combinatorial Interaction Testing

An Example Variability Model



	Factors			
	Web Browser	**Operating System**	**Connection Type**	**Printer Type**
Value	Netscape	Windows XP	LAN	Local
	IE	OS X	PPP	Networked
	Mozilla	Linux	ISDN	Screen

Testing This Model

	Factors			
	Web Browser	**Operating System**	**Connection Type**	**Printer Type**
Value	Netscape	Windows XP	LAN	Local
	IE	OS X	PPP	Networked
	Mozilla	Linux	ISDN	Screen

In this example we have
- 4 factors
- 3 values each

Testing This Model

	Factors			
	Web Browser	**Operating System**	**Connection Type**	**Printer Type**
Value	Netscape	Windows XP	LAN	Local
	IE	OS X	PPP	Networked
	Mozilla	Linux	ISDN	Screen

In this example we have
• 4 factors
• 3 values each

There are 3^4 or 81 possible instances of this variability model

.

Testing This Model

	Factors			
	Web Browser	**Operating System**	**Connection Type**	**Printer Type**
Value	Netscape	Windows XP	LAN	Local
	IE	OS X	PPP	Networked
	Mozilla	Linux	ISDN	Screen

In this example we have
- 4 factors
- 3 values each

There are 3^4 or 81 possible instances of this variability model

Suppose we have 15 factors with 5 values each:

5^{15} = 30,517,578,125 possible instances!

We cannot realistically test all of these.

Some Real Software Systems

- SQL Server 7.0:
  - 47 configuration options
    - 10 are binary, the rest have a range of values
- Oracle 9:
  - 211 initialization parameters
    - ? Options/per parameter
- Apache HTTP Server Version 1.3
  - 85 core configuration options
    - 15 binary
- GCC-3.3.1 compiler
  - over 1000 command line flags
    - These control 14 options.
    - More than 50 flags are used to control optimization alone
- Czarnecki:
    - E-commerce software product line with 350 variation points

Another Example: ACE+TAO

(Memon et al., ICSE 2004)

Middleware for distributed software applications

Over 1 million lines of code, runs on multiple operating systems and multiple compilers.

- **Static configurations**:
  - The static configuration space has over 82,000 potential configurations.
  - Compiling the full system requires 4 hours.
  - A simplified model was used that examined less than 100 static configurations. Of these only 29 compiled successfully.
- **Dynamic configurations**:
  - This includes 6 runtime options ranging from 2-4 values each.
  - 648 possible combinations of CORBA runtime policies, each of which has to be tested with *all valid* static configurations (29).

ACE/TAO (Cont.)

- Tests Provided
  - A set of 96 tests has to compiled and run for each system configuration
  - Compilation of these test cases requires an additional 3.5 hours
  - running this set of tests requires 30 minutes.

- Total time to compile/run tests for each configurations
  8 hours
- Testing the partial variation space includes compiling and testing 18,792 configurations which requires 9,400 hours (1 year) of computer time!

Mappings

Static Configs

TAO_HAS_AMI
TAO_HAS_AMI_CALLBACK
TAO_HAS_AMI_POLLER
TAO_HAS_CORBA_MESSAGING
TAO_HAS_DIOP
TAO_HAS_INTERCEPTORS
TAO_HAS_MINIMUM_CORBA
TAO_HAS_MINIMUM_POA
TAO_HAS_MINIMUM_POA_MAPS
TAO_HAS_NAMED_RT_MUTEXES

Some Runtime

1- ORBCollocation
- global
- per-orb
- NO

2- ORBConnectionPurgingStrategy
- lru
- lfu
- fifo
- null

Instances

1101100001	per-orb	lfu	reactive	thread-per-connection	MT	LF
1001110001	per-orb	fifo	reactive	reactive	RW	LF

Sampling the Variability Space

- One solution used for functional software testing is to sample a systematic subset of input combinations.

- Want to guarantee certain properties are met.

- A balanced property is to select a sample that includes all pairs or three way combinations of factors.

Pair Wise Coverage of the SPL

	Factors			
	Web Browser	**Operating System**	**Connection Type**	**PrinterType**
Value	Netscape	**Windows XP**	LAN	**Local**
	IE	OS X	PPP	Networked
	Mozilla	Linux	**ISDN**	Screen

Pair Wise Coverage of the SPL

	Factors			
	Web Browser	**Operating System**	**Connection Type**	**PrinterType**
Value	Netscape	**Windows XP**	LAN	**Local**
	IE	OS X	PPP	Networked
	Mozilla	Linux	**ISDN**	Screen

Test Case	**Browser**	**OS**	**Connection**	**Printer**
1	Netscape	**Windows XP**	LAN	**Local**
2	Netscape	Linux	**ISDN**	Networked
3	Netscape	OS X	PPP	Screen
4	**IE**	**Windows XP**	**ISDN**	Screen
5	**IE**	OS X	LAN	Networked
6	**IE**	Linux	PPP	**Local**
7	Mozilla	**Windows XP**	PPP	Networked
8	Mozilla	Linux	LAN	Screen
9	Mozilla	OS X	**ISDN**	**Local**

Outline : Interactions

Validating Product Lines

1. Introduction
2. A Motivating Example
3. Combinatorial Interaction Testing

Combinatorial Interaction Testing

- Based on statistical design of experiments (DOE)
  - Manufacturing
  - Drug test interactions
  - Chemical interactions
- For software testing
  - Mandl – compiler testing
  - Brownlie, Prowse, Phadke – OATS system
  - D. Cohen, Dalal, Fredman, Patton, Parelius – AETG
  - Williams, Probert – network node interfaces
  - Yilmaz, Cohen, Porter- ACE/TAO

Combinatorial Structures Used

- Mandl (1985) uses Mutually Orthogonal Latin Squares
- Browlie *et al.* (1992) uses Orthogonal Arrays
- D. Cohen, Dalal, Fredman, Patton, Parelius (1996) uses Covering Arrays

Mutually Orthogonal Latin Squares (MOLS)

0	2	**3**	1
3	1	0	2
1	**3**	2	0
2	0	1	**3**

0	2	**3**	1
1	3	2	0
2	**0**	1	3
3	1	0	**2**

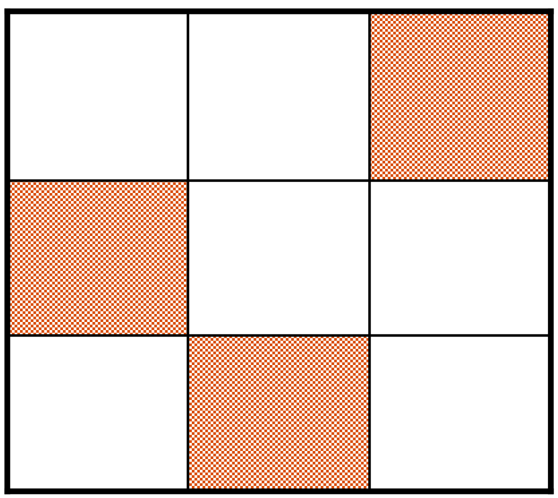
0	2	**3**	1
2	0	1	3
3	**1**	0	2
1	3	2	**0**

- Each row and each column contains all symbols (0…s-1) exactly once.
- Each pair of squares covers all s^2 ordered pairs
  {(0,0), (0,1),(0,2),…,(s-1,s-1)}
- We can use n MOLS to test a system with n+2 factors, each with s values.

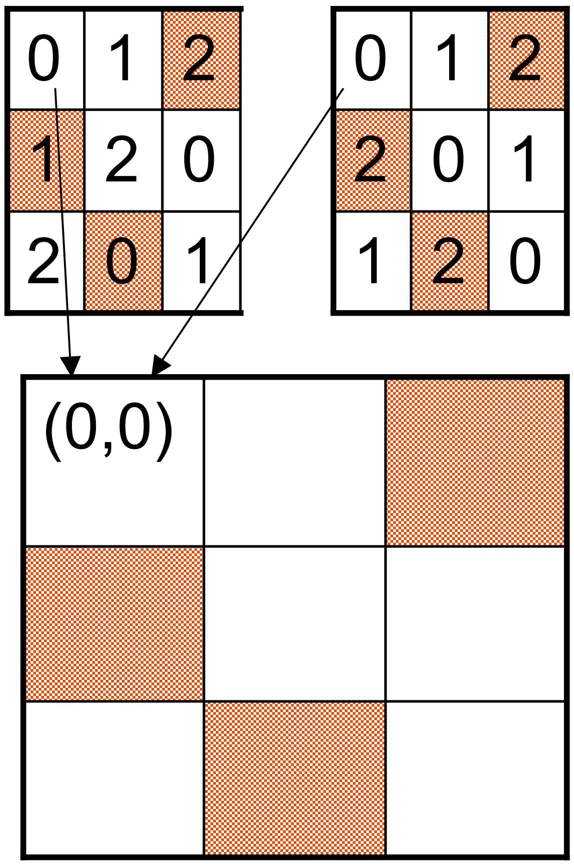
Example

0	1	2
1	2	0
2	0	1

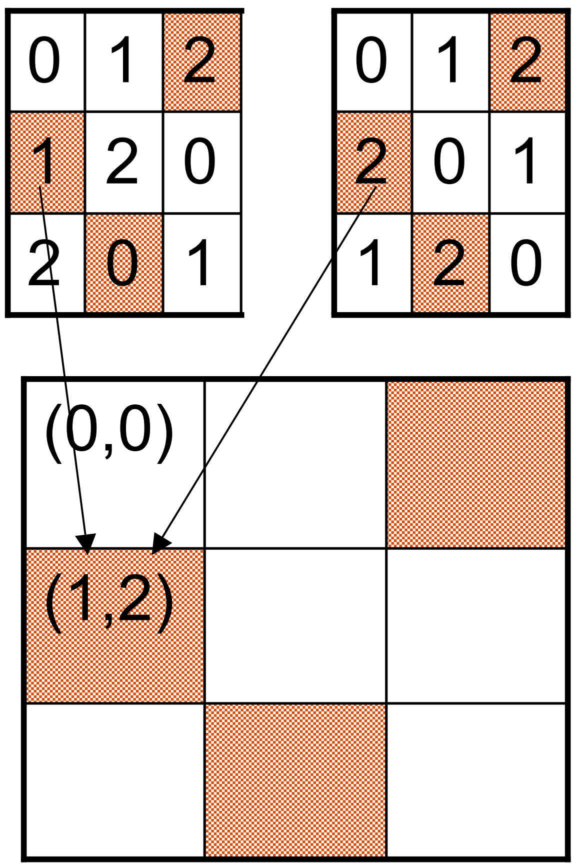
0	1	2
2	0	1
1	2	0



Example



Example



Example

0	1	2
1	2	0
2	0	1

0	1	2
2	0	1
1	2	0

(0,0)	(1,1)	(2,2)
(1,2)	(2,0)	(0,1)
(2,1)	(0,2)	(1,0)

Col Index	row index	square 1 cell	square 2 cell
0	0	0	0
0	1	1	1
0	2	2	2
.			
.			
.			

Example

0	1	2
1	2	0
2	0	1

0	1	2
2	0	1
1	2	0

(0,0)	(1,1)	(2,2)
(1,2)	(2,0)	(0,1)
(2,1)	(0,2)	(1,0)

Browser (row)	**OS (col)**	**Connection (latin Sq 1)**	**Printer (latin Sq 2)**
Netscape	Windows XP	LAN	Local
Netscape	OS X	PPP	Networked
Netscape 0	Linux 2	ISDN 2	Screen 2
IE	Windows XP	PPP	Screen
IE	OS X	ISDN	Local
IE	Linux	LAN	Networked
Mozilla	Windows XP	ISDN	Networked
Mozilla	OS X	LAN	Screen
Mozilla	Linux	PPP	Local

Mappings

Netscape → 0, IE → 1, Mozilla → 2
Win XP→ 0, OS X → 1, Linux→ 2
LAN → 0, PPP → 1, ISDN → 2
Local → 0, Networked → 1, Screen→ 2

Method

- We create an s^2 x (n+2) array.
  (Each row will be a product line instance)

- The first two columns are the row and column indices of the squares.

- For each row we fill the next n columns with the cell entries of the n corresponding latin squares.

Orthogonal Arrays

$OA_\lambda(t,k,v)$

- A v^t *x k* array on v symbols where each *N x t* sub-array contains all ordered *t-sets* *exactly* λ times.

OA(3,4,2)

0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**1**
0	**1**	**1**	**0**
1	**0**	**0**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

Orthogonal Arrays

$OA_\lambda(t,k,v)$

– A v^t *x k* array on v symbols where each *N x t* sub-array contains all ordered *t-sets* *exactly* λ times.

OA(3,4,2)

0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**1**
0	**1**	**1**	**0**
1	**0**	**0**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

Orthogonal Arrays

$OA_\lambda(t,k,v)$

- A v^t *x k* array on v symbols where each *N x t* sub-array contains all ordered *t-sets exactly* λ times.

OA(3,4,2)

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

Orthogonal Arrays

- Orthogonal arrays are used in statistical testing for determining “main effects” because they are balanced.

But:

- They do not exist for all values of t,k,v.
- They have the property that all t-sets occur exactly once.

This property (exactly once) is more restrictive than is needed for testing software.

Covering Arrays

$CA_\lambda(N;t,k,v)$

– An $N \times k$ on v symbols array where each $N \times t$ sub-array contains all ordered t-sets at least λ times.

(we can drop the λ when $\lambda=1$)

t is the strength of the array

$CA(6;2,5,2)$

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

Covering Arrays

$CA_{\lambda}(N;t,k,v)$

– An N x k on v symbols array where each N x t sub-array contains all ordered t-sets at least λ times.

(we can drop the λ when λ=1)

t is the strength of the array

CA(6;2,5,2)

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

Covering Arrays

$CA_{\lambda}(N;t,k,v)$

- An N x k on v symbols array where each N x t sub-array contains all ordered *t-sets* at least λ times.

(we can drop the λ when λ=1)

t is the strength of the array

CA(6;2,5,2)

Networked

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

Local

Win XP

OS X

Covering Arrays

t strength (t-wise coverage)
k degree (number of factors)
v order (number of values)

A covering array is optimal if it contains the minimum possible number of rows. We call this the *covering array number:*

$$CAN(t,k,v)$$

The covering array number is not known for all covering arrays.

The Original Problem

	Factor			
	Web Browser	**Operating System**	**Connection Type**	**Printer Type**
Value	Netscape	Windows XP	LAN	Local
	IE	OS X	PPP	Networked
	Mozilla	Linux	ISDN	Screen

The product line has 4 factors, each with 3 values.

For pair-wise coverage: k=4, v=3, t=2

a CA(N;2,4,3)

CA(9;2,4,3)

(also an OA(2,4,3))

A set of product line instances that covers all pair-wise interactions.

Config	Browser	OS	Connection	Printer
1	Netscape	Windows XP	LAN	Local
2	Netscape	Linux	ISDN	Networked
3	Netscape	OS X	PPP	Screen
4	IE	Windows XP	ISDN	Screen
5	IE	OS X	LAN	Networked
6	IE	Linux	PPP	Local
7	Mozilla	Windows	PPP	Networked
8	Mozilla	Linux	LAN	Screen
9	Mozilla	OS X	ISDN	Local

Another CA(N;2,4,3)
Is this Optimal?

Config	Browser	OS	Connection	Printer
1	Netscape	Windows	LAN	Local
2	Netscape	Linux	ISDN	Networked
3	Netscape	OS X	PPP	Screen
4	IE	Windows	ISDN	Screen
5	IE	OS X	LAN	Networked
6	IE	Linux	PPP	Local
7	Mozilla	Windows	PPP	Networked
8	Mozilla	Linux	LAN	Screen
9	Mozilla	Windows	PPP	Local
10	Mozilla	Linux	PPP	Screen
11	Mozilla	OS X	ISDN	Local

References

R.Brownlie, J.Prowse, and M.S. Phadke, Robust testing of AT&T PMX/StarMAIL using OATS,*AT& T Technical Journal*, vol.71 no. 3, pp. 41--47, 1992.

D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, The AETG system: an approach to testing based on combinatorial design, *IEEE Transactions on Software Engineering*, vol. 23, no.7, pp. 437--444, 1997.

D.M. Cohen, S. R. Dalal, M. L. Fredman, and G.~C. Patton, Method and system for automatically generating efficient test cases for systems having interacting elements,1996, United States Patent, Number 5,542,043.

A. Hedayat, N.Sloane, and J. Stufken, *Orthogonal Arrays*,New York: Springer-Verlag, 1999.

References

D.R. Kuhn, D.R. Wallace and A.M. Gallo, Software fault interactions and implications for software testing, *IEEE Trans. Software Engineering*, 30(6), 2004, pp. 418--421.

R.Mandl, Orthogonal Latin squares: an application of experiment design to compiler testing, *Communications of the ACM,* vol.28, no.10, pp. 1054--1058, 1985.

A.W. Williams and R.L. Probert, A practical strategy for testing pair-wise coverage of network interfaces, In *Proc. Intl. Symp. on Software Reliability Engineering*,(ISSRE), 1996 pp.~246--54.

C. Yilmaz, M .B. Cohen and A. Porter, Covering arrays for efficient fault characterization in complex configuration spaces, *IEEE Transactions on Software Engineering*, 31(1), 2006, pp. 20-34.