National Institute of Standards and Technology

Combinatorial Methods in Software Testing

Rick Kuhn

National Institute of
Standards and Technology
Gaithersburg, MD

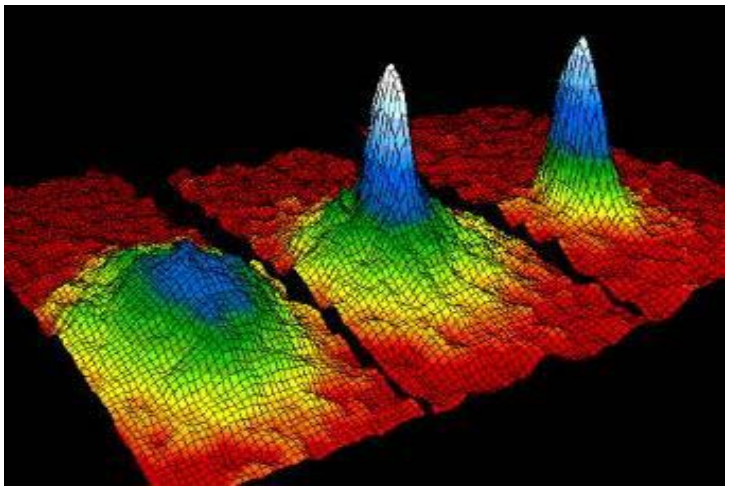
Federal Computer Security Managers Forum, Dec. 6, 2011

NIST Combinatorial Testing project

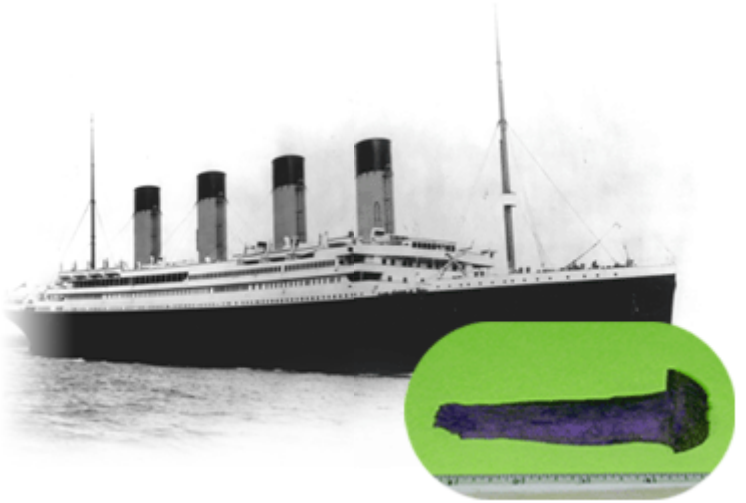
- Goals – reduce testing cost, improve cost-benefit ratio for testing
  - Merge automated test generation with combinatorial methods
  - New algorithms to make large-scale combinatorial testing practical
- Accomplishments – huge increase in performance, scalability
  + widespread use in real-world applications
- Joint research with many organizations

What is NIST and why are we doing this?

- A US Government agency
- The nation's measurement and testing laboratory – 3,000 scientists, engineers, and support staff including 3 Nobel laureates



Research in physics, chemistry, materials, manufacturing, computer science



Analysis of engineering failures, including buildings, materials, **and ...**

NIST
National Institute of Standards and Technology

Software Failure Analysis

- We studied software failures in a variety of fields including 15 years of FDA medical device recall data
- What causes software failures?
  - logic errors?
  - calculation errors?
  - interaction faults?
  - inadequate input checking? Etc.
- What testing and analysis would have prevented failures?
- Would statement coverage, branch coverage, all-values, all-pairs etc. testing find the errors?

Interaction faults: e.g., failure occurs if
`pressure < 10 && volume > 300` (2-way interaction <= all-pairs testing catches)

NIST National Institute of Standards and Technology

Software Failure Internals

How does an interaction fault manifest itself in code?

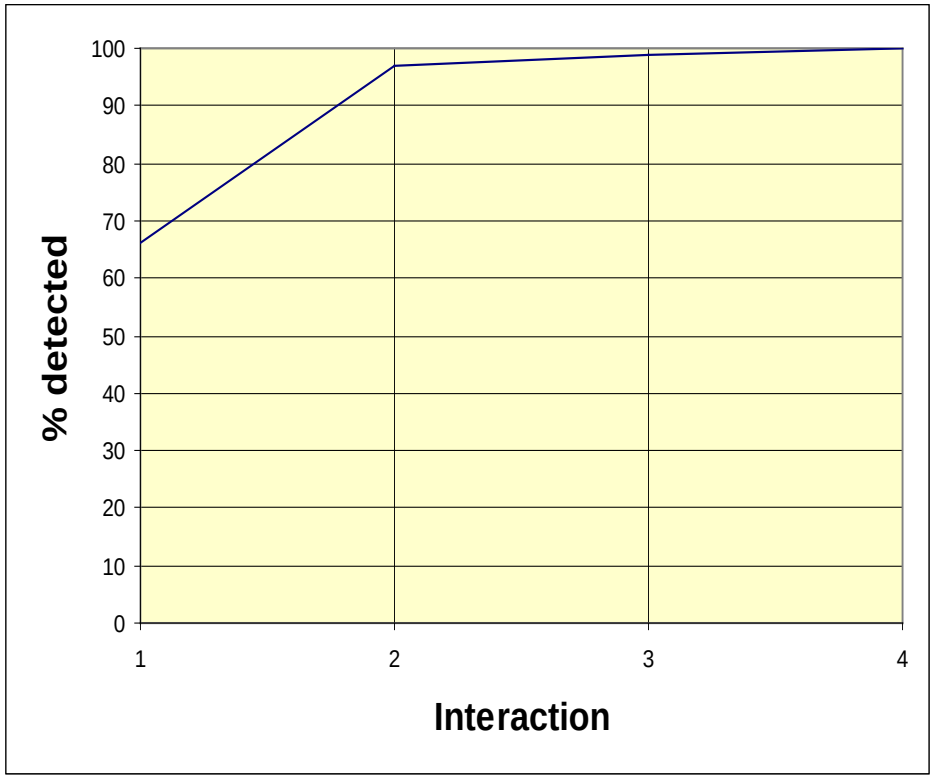
Example: pressure < 10 && volume > 300 (2-way interaction)

```
if (pressure < 10) {
    // do something
    if (volume > 300)  { faulty code!  BOOM! }
    else { good code, no problem}
} else {
    // do something else
}
```

A test that included pressure = 5 and volume = 400 would trigger this failure

How about flaws that are harder to find ?

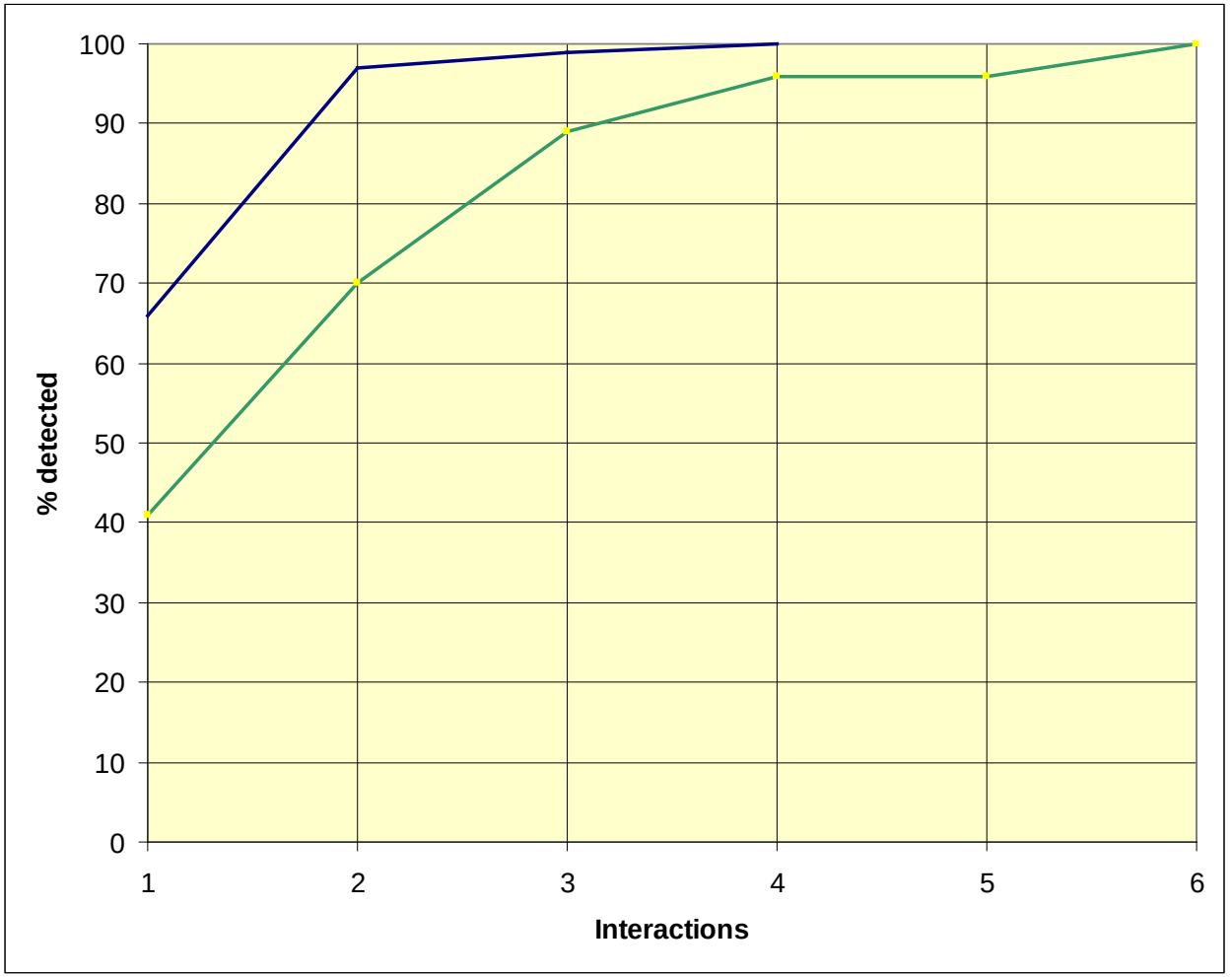
- Interactions e.g., failure occurs if
- pressure < 10 (1-way interaction)
- pressure < 10 & volume > 300 (2-way interaction)
- pressure < 10 & volume > 300 & velocity = 5 (3-way interaction)
- The most complex failure reported required 4-way interaction to trigger



NIST National Institute of Standards and Technology

What about other applications?

Server (green)

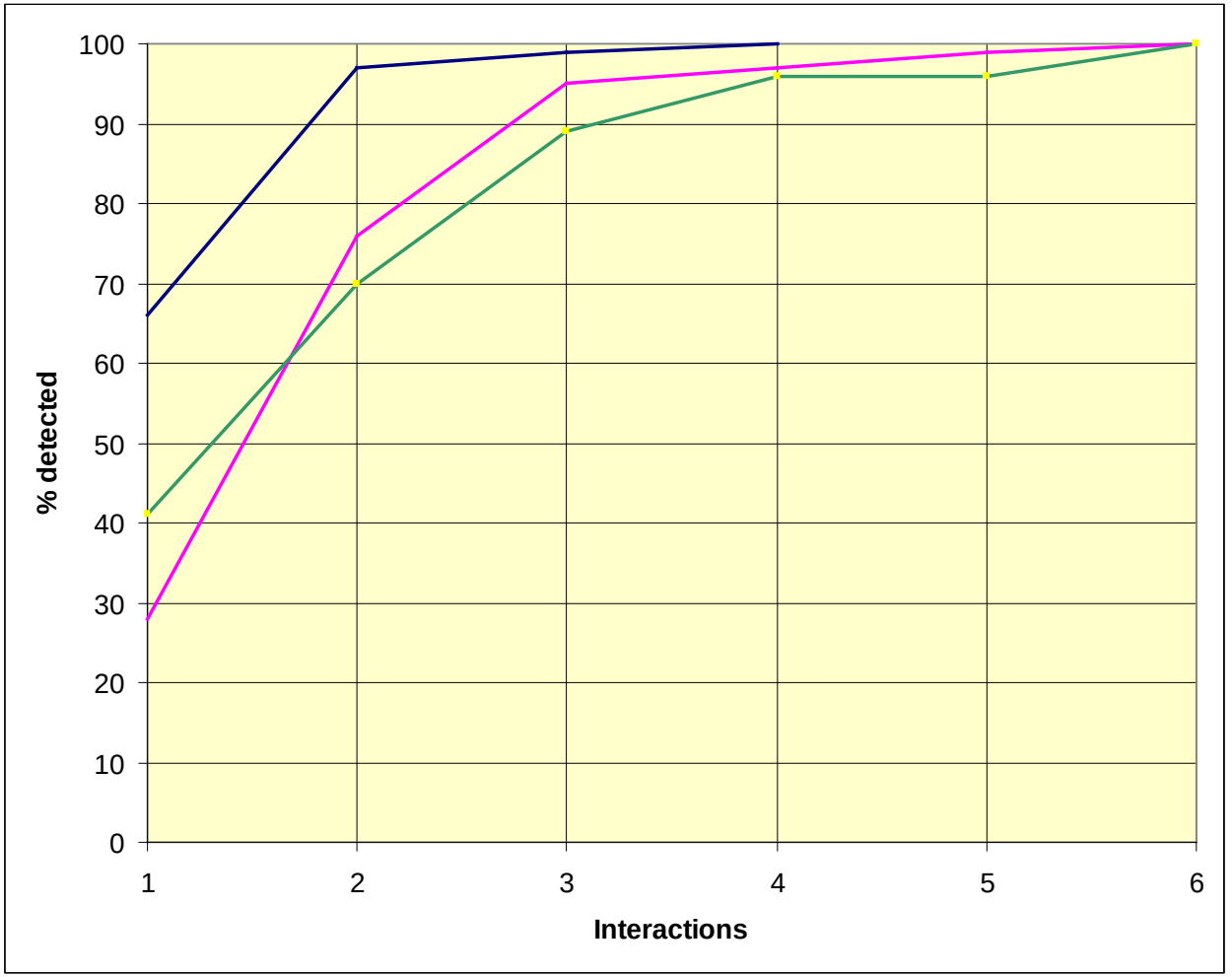


These faults more complex than medical device software!!

Why?

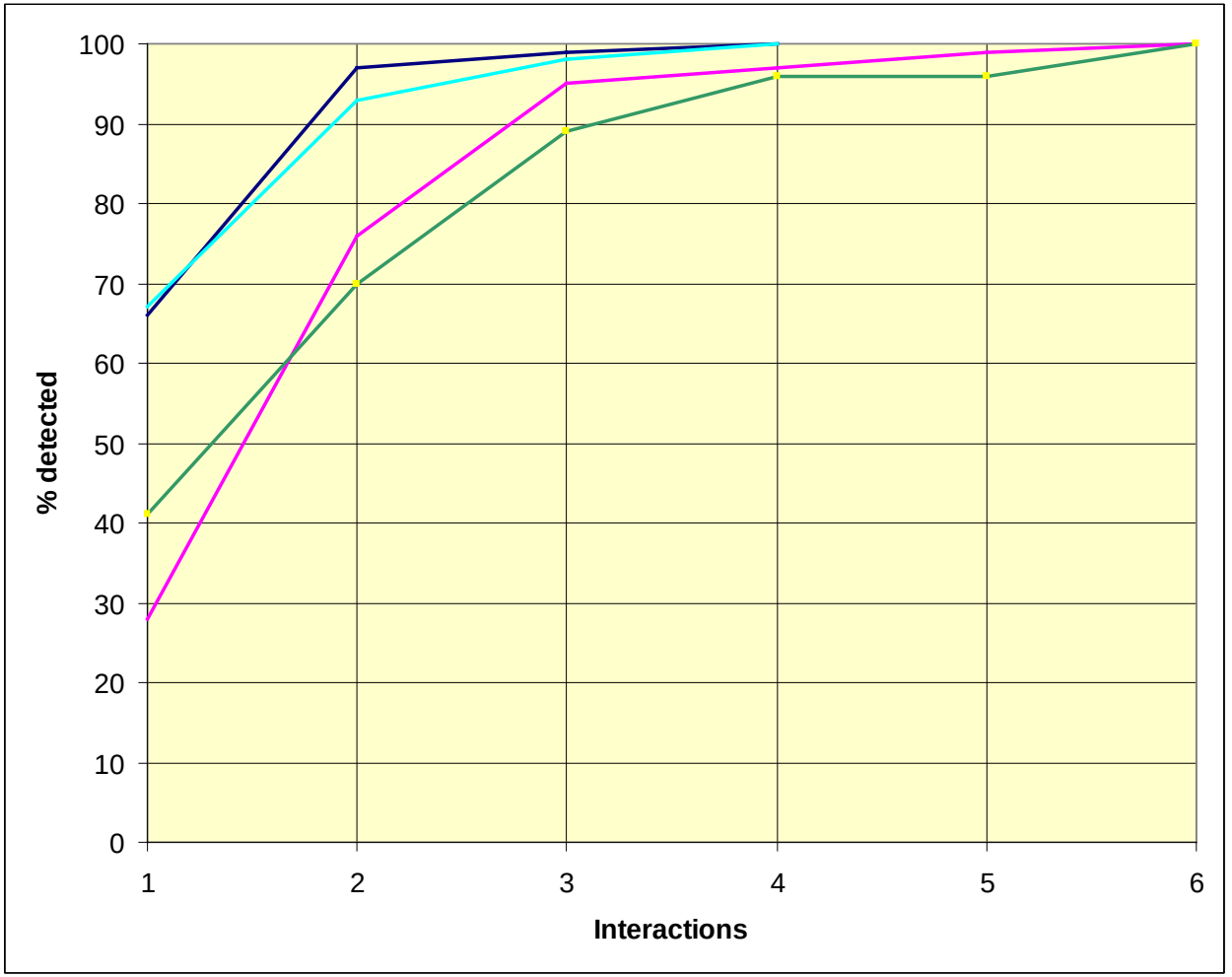
Others?

Browser (magenta)



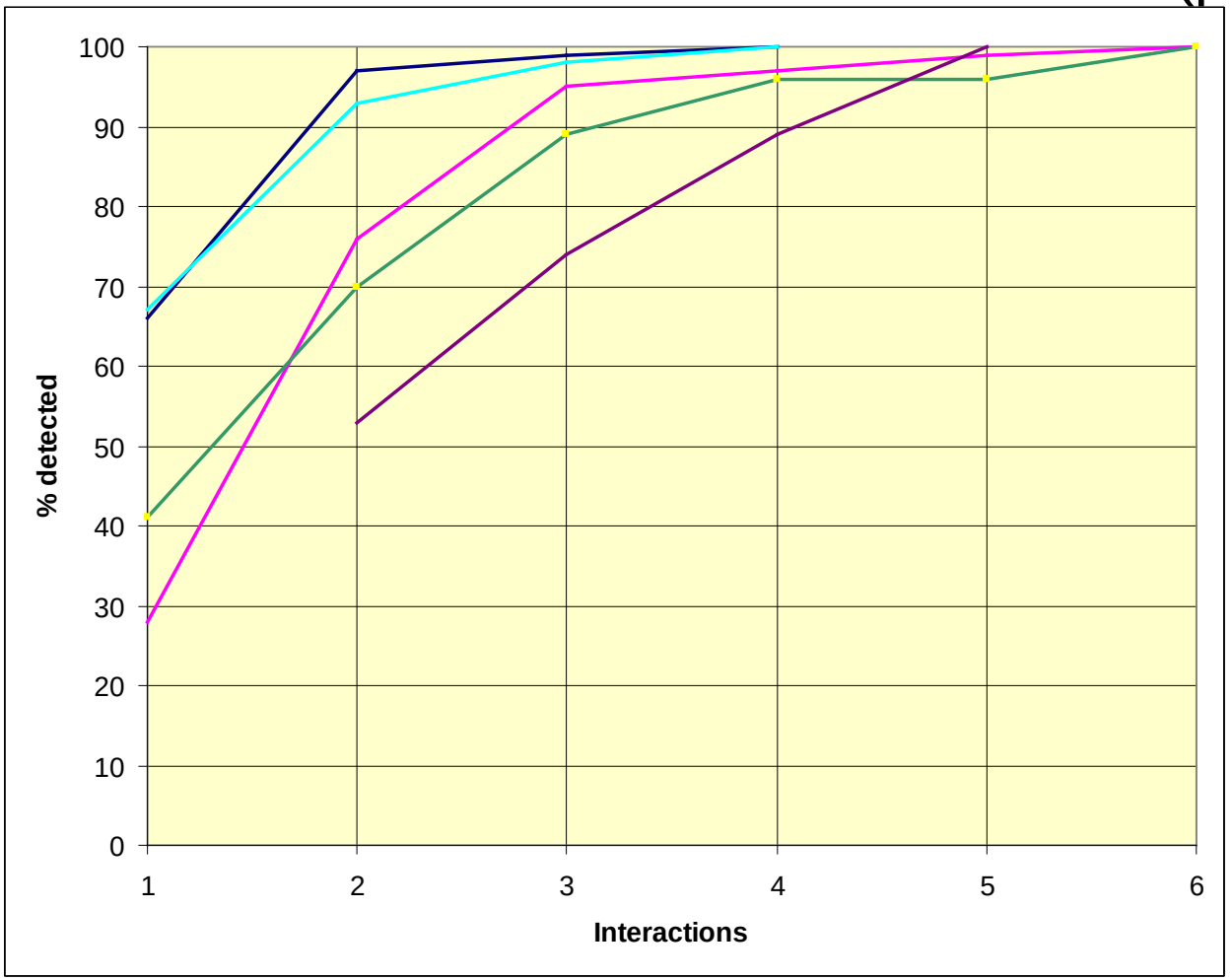
Still more?

NASA Goddard distributed database (light blue)



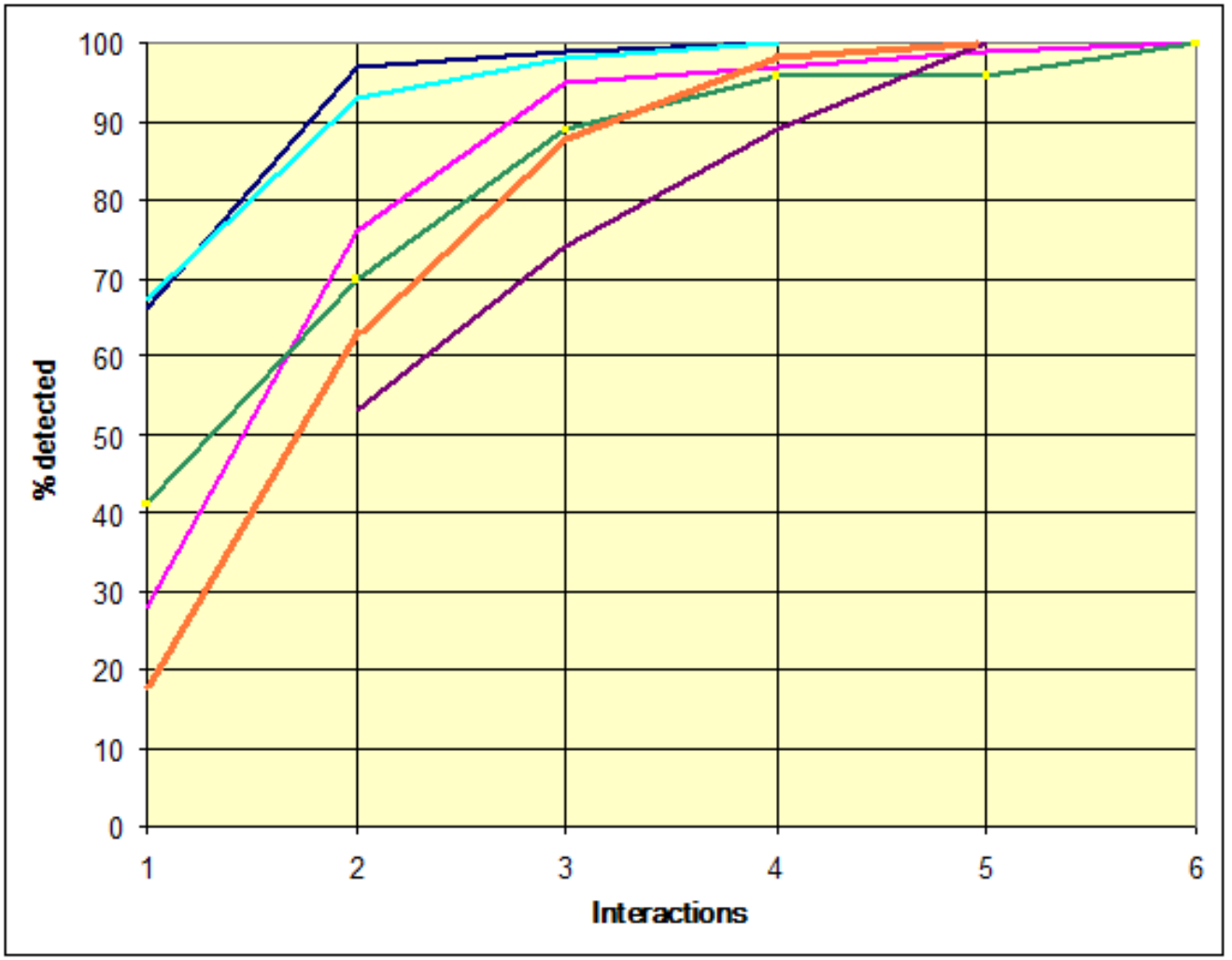
Even more?

FAA Traffic Collision Avoidance System module (seeded errors) (purple)



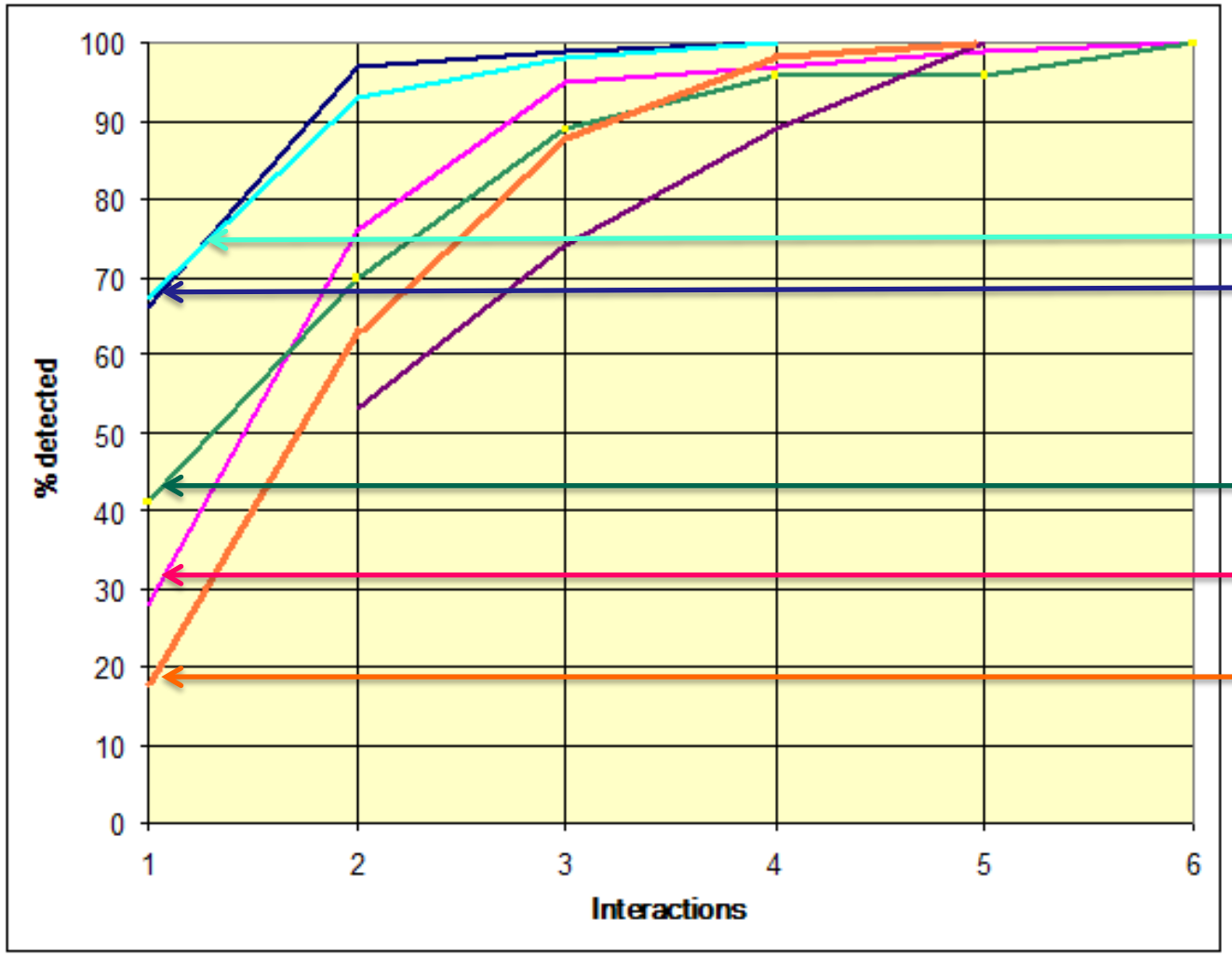
Finally

Network security (Bell, 2006) (orange)



Curves appear to be similar across a variety of application domains.

Why this distribution?



App / users / SLOC

NASA 0 ≈ 10^6

Med. 1000s ≈ 10^3 – 10^4

Server 10s of mill. ≈ 10^5

Browser 10s of mill. ≈ 10^6

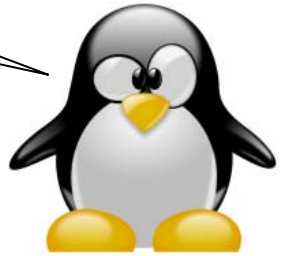
TCP/IP 100s of mill. ≈ 10^3

NIST National Institute of Standards and Technology

So, how many parameters are involved in really tricky faults?

- The ***interaction rule***: most failures are triggered by one or two parameters, and progressively fewer by three, four, or more parameters, and the maximum interaction degree is small.
- Maximum interactions for fault triggering was 6
- Popular "pairwise testing" not enough
- More empirical work needed
- Reasonable evidence that maximum interaction strength for fault triggering is relatively small

How does it help me to know this?



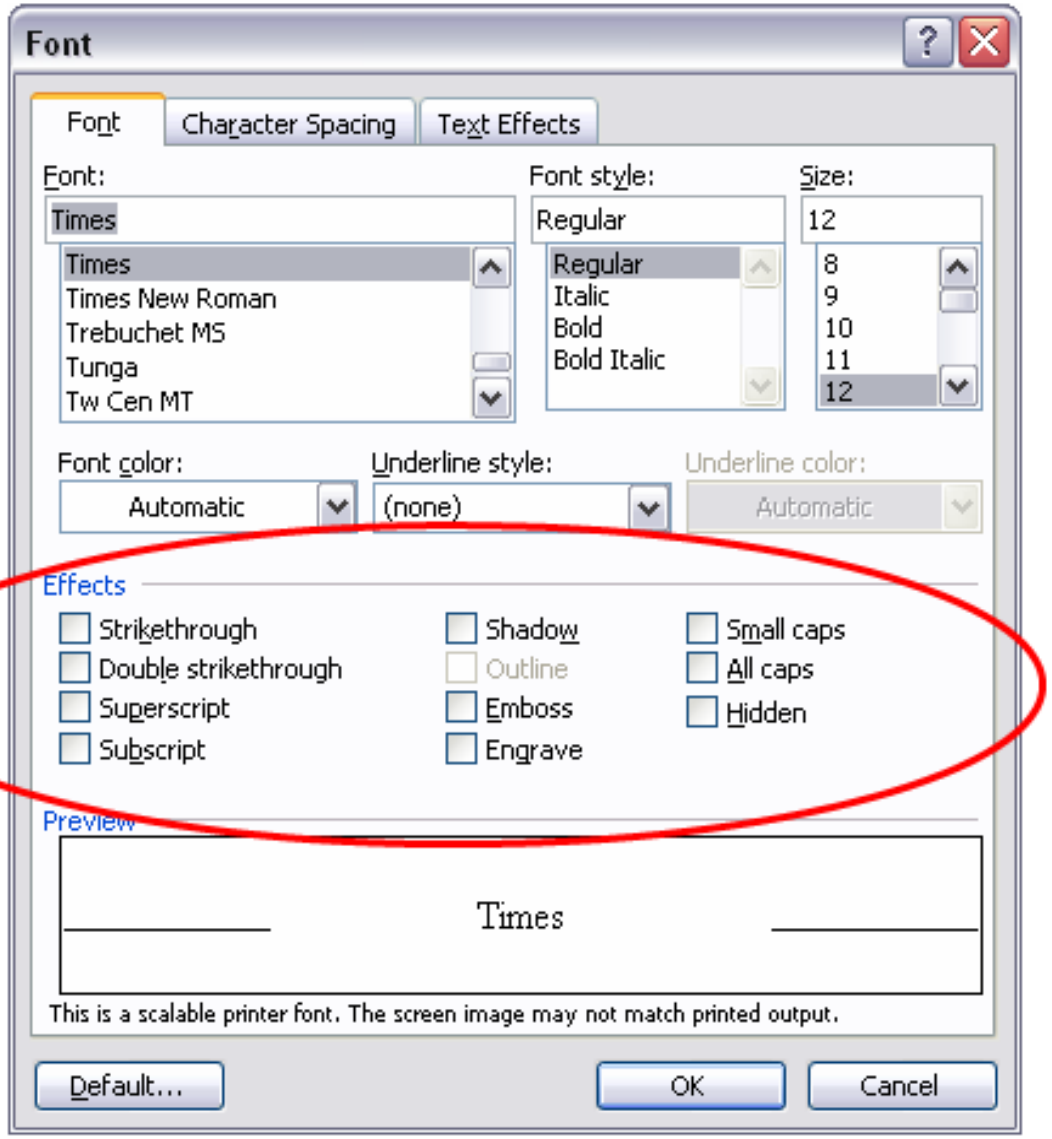
National Institute of Standards and Technology

How does this knowledge help?

If all faults are triggered by the interaction of *t* or fewer variables, then testing all *t*-way combinations can provide strong assurance.

(taking into account: value propagation issues, equivalence partitioning, timing issues, more complex interactions, . . .)

How do we use this knowledge in testing?
A simple example



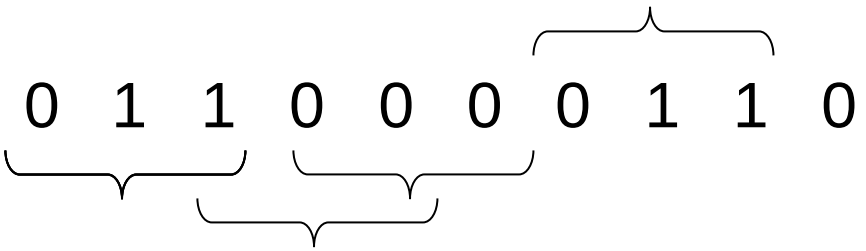
NIST
National Institute of Standards and Technology

How Many Tests Would It Take?

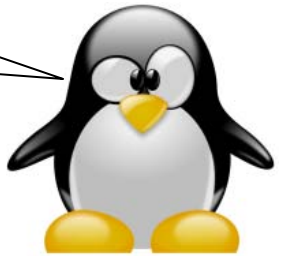
- There are 10 effects, each can be on or off
- All combinations is 2^{10} = 1,024 tests
- What if our budget is too limited for these tests?
- Instead, let's look at all 3-way interactions …

Now How Many Would It Take?

- There are $\binom{10}{3}$ = 120 3-way interactions.
- Naively 120 x 2^3 = 960 tests.
- Since we can pack 3 triples into each test, we need no more than 320 tests.
- Each test exercises many triples:



OK, OK, what's the smallest number of tests we need?



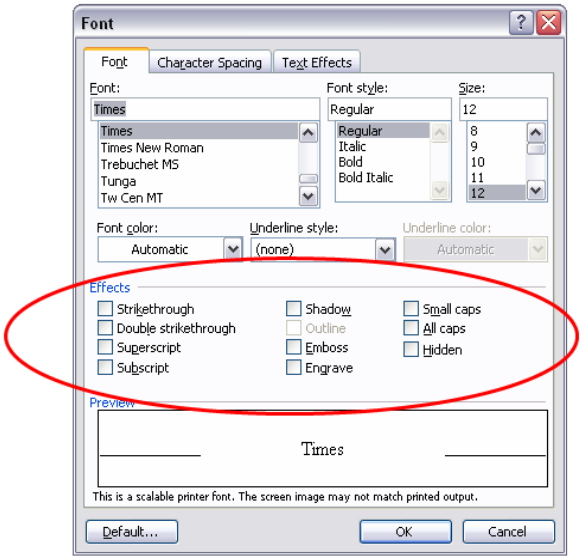
A covering array

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:



- Developed 1990s
- Extends Design of Experiments concept
- NP hard problem but good algorithms now

NIST National Institute of Standards and Technology

A larger example

Suppose we have a system with on-off switches. Software must produce the right response for any combination of switch settings:

How do we test this?

34 switches = 2^{34} = 1.7 x 10^{10} possible inputs = 1.7 x 10^{10} tests

What if we knew no failure involves more than 3 switch settings interacting?

- 34 switches = 2^{34} = 1.7 x 10^{10} possible inputs = **1.7 x 10^{10}** tests
- If only 3-way interactions, need only **33** tests
- For 4-way interactions, need only **85** tests

Two ways of using combinatorial testing

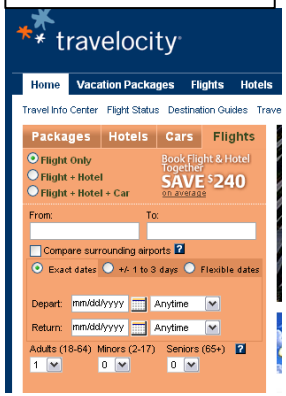
Use combinations here

or here

Test case	OS	CPU	Protocol
1	Windows	Intel	IPv4
2	Windows	AMD	IPv6
3	Linux	Intel	IPv6
4	Linux	AMD	IPv4

Configuration

Test data inputs



System under test

Testing Configurations

- Example: app must run on any configuration of OS, browser, protocol, CPU, and DBMS
- Very effective for interoperability testing, being used by NIST for DoD Android phone testing

Test	OS	Browser	Protocol	CPU	DBMS
1	XP	IE	IPv4	Intel	MySQL
2	XP	Firefox	IPv6	AMD	Sybase
3	XP	IE	IPv6	Intel	Oracle
4	OS X	Firefox	IPv4	AMD	MySQL
5	OS X	IE	IPv4	Intel	Sybase
6	OS X	Firefox	IPv4	Intel	Oracle
7	RHL	IE	IPv6	AMD	MySQL
8	RHL	Firefox	IPv4	Intel	Sybase
9	RHL	Firefox	IPv4	AMD	Oracle
10	OS X	Firefox	IPv6	AMD	Oracle

Testing Smartphone Configurations

Some Android configuration options:

int HARDKEYBOARDHIDDEN_NO;
int HARDKEYBOARDHIDDEN_UNDEFINED;
int HARDKEYBOARDHIDDEN_YES;
int KEYBOARDHIDDEN_NO;
int KEYBOARDHIDDEN_UNDEFINED;
int KEYBOARDHIDDEN_YES;
int KEYBOARD_12KEY;
int KEYBOARD_NOKEYS;
int KEYBOARD_QWERTY;
int KEYBOARD_UNDEFINED;
int NAVIGATIONHIDDEN_NO;
int NAVIGATIONHIDDEN_UNDEFINED;
int NAVIGATIONHIDDEN_YES;
int NAVIGATION_DPAD;
int NAVIGATION_NONAV;
int NAVIGATION_TRACKBALL;
int NAVIGATION_UNDEFINED;
int NAVIGATION_WHEEL;
int ORIENTATION_LANDSCAPE;
int ORIENTATION_PORTRAIT;
int ORIENTATION_SQUARE;
int ORIENTATION_UNDEFINED;
int SCREENLAYOUT_LONG_MASK;
int SCREENLAYOUT_LONG_NO;
int SCREENLAYOUT_LONG_UNDEFINED;
int SCREENLAYOUT_LONG_YES;
int SCREENLAYOUT_SIZE_LARGE;
int SCREENLAYOUT_SIZE_MASK;
int SCREENLAYOUT_SIZE_NORMAL;
int SCREENLAYOUT_SIZE_SMALL;
int SCREENLAYOUT_SIZE_UNDEFINED;
int TOUCHSCREEN_FINGER;
int TOUCHSCREEN_NOTOUCH;
int TOUCHSCREEN_STYLUS;
int TOUCHSCREEN_UNDEFINED;

Configuration option values

Parameter Name	Values	# Values
HARDKEYBOARDHIDDEN	NO, UNDEFINED, YES	3
KEYBOARDHIDDEN	NO, UNDEFINED, YES	3
KEYBOARD	12KEY, NOKEYS, QWERTY, UNDEFINED	4
NAVIGATIONHIDDEN	NO, UNDEFINED, YES	3
NAVIGATION	DPAD, NONAV, TRACKBALL, UNDEFINED, WHEEL	5
ORIENTATION	LANDSCAPE, PORTRAIT, SQUARE, UNDEFINED	4
SCREENLAYOUT_LONG	MASK, NO, UNDEFINED, YES	4
SCREENLAYOUT_SIZE	LARGE, MASK, NORMAL, SMALL, UNDEFINED	5
TOUCHSCREEN	FINGER, NOTOUCH, STYLUS, UNDEFINED	4

Total possible configurations:

3 x 3 x 4 x 3 x 5 x 4 x 4 x 5 x 4 = 172,800

NIST National Institute of Standards and Technology

Number of configurations generated for *t*-way interaction testing, *t* = 2..6

t	# Configs	% of Exhaustive
2	29	0.02
3	137	0.08
4	625	0.4
5	2532	1.5
6	9168	5.3

New algorithms

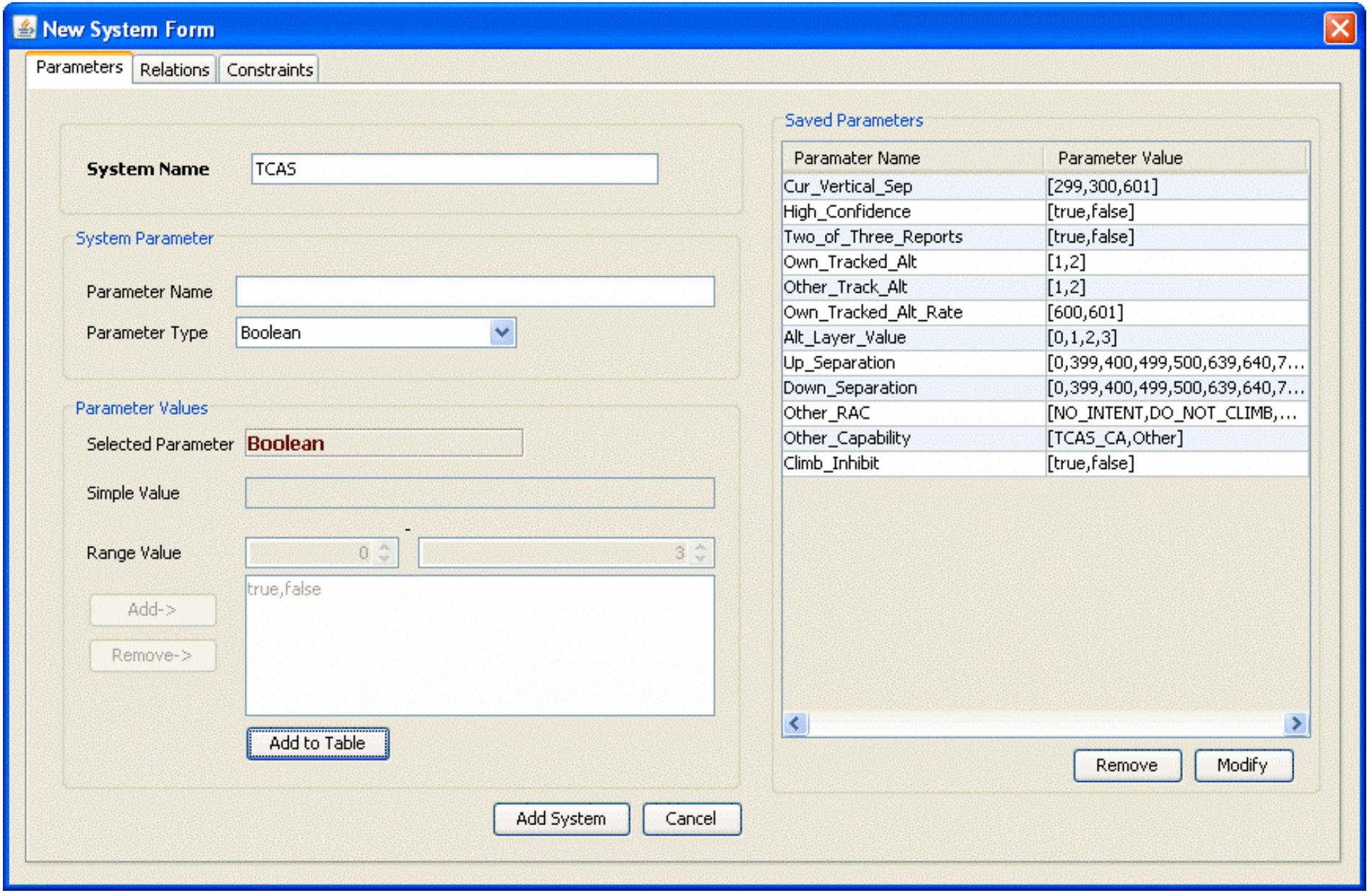
- Smaller test sets faster, with a more advanced user interface
- First parallelized covering array algorithm
- More information per test

T-Way	IPOG		ITCH (IBM)		Jenny (Open Source)		TConfig (U. of Ottawa)		TVG (Open Source)	
	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
2	100	0.8	120	0.73	108	0.001	108	>1 hour	101	2.75
3	400	0.36	2388	1020	413	0.71	472	>12 hour	9158	3.07
4	1363	3.05	1484	5400	1536	3.54	1476	>21 hour	64696	127
5	4226	18s	NA	>1 day	4580	43.54	NA	>1 day	313056	1549
6	10941	65.03	NA	>1 day	11625	470	NA	>1 day	1070048	12600

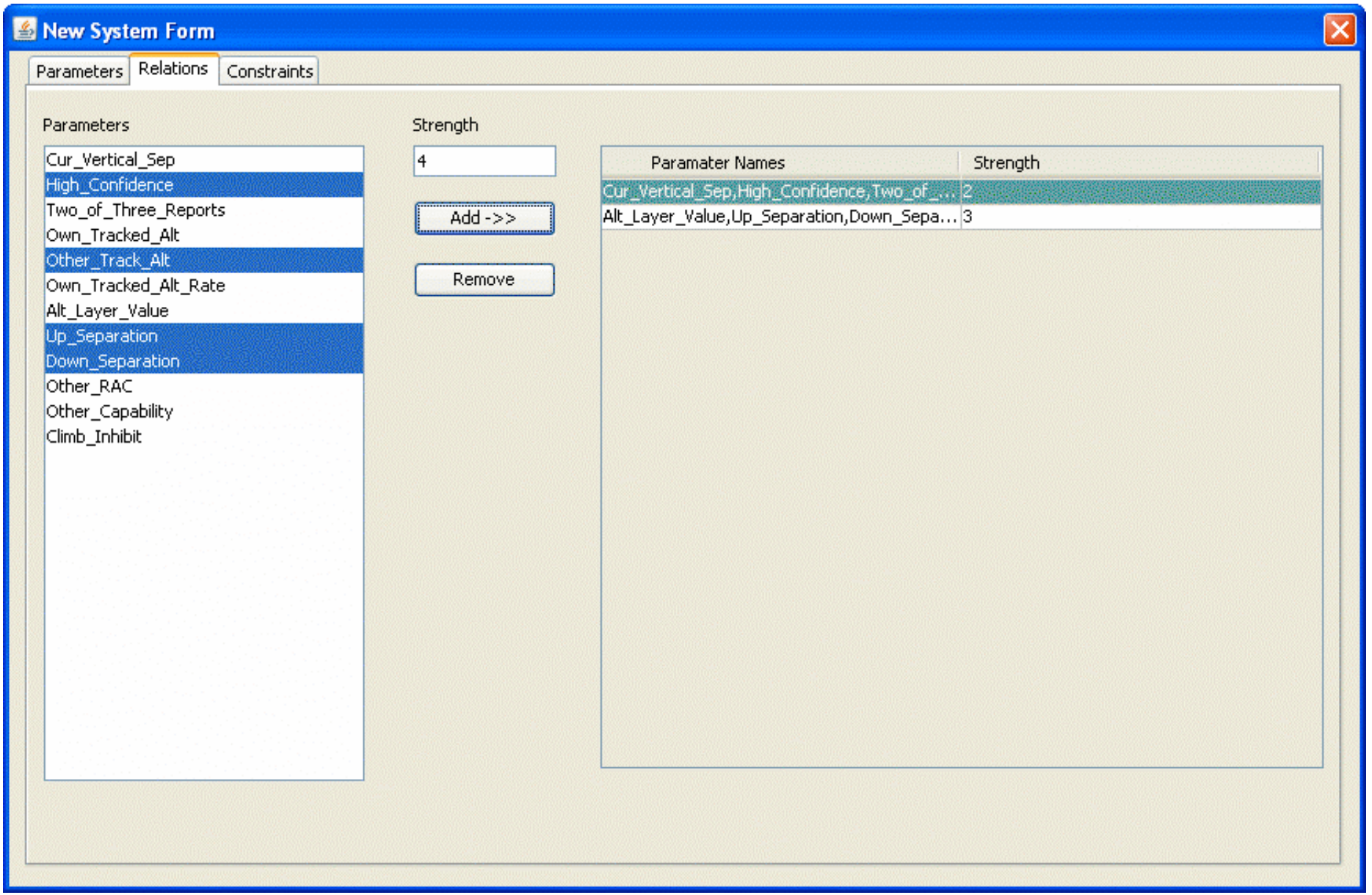
Traffic Collision Avoidance System (TCAS): $2^7 3^2 4^1 10^2$

Times in seconds

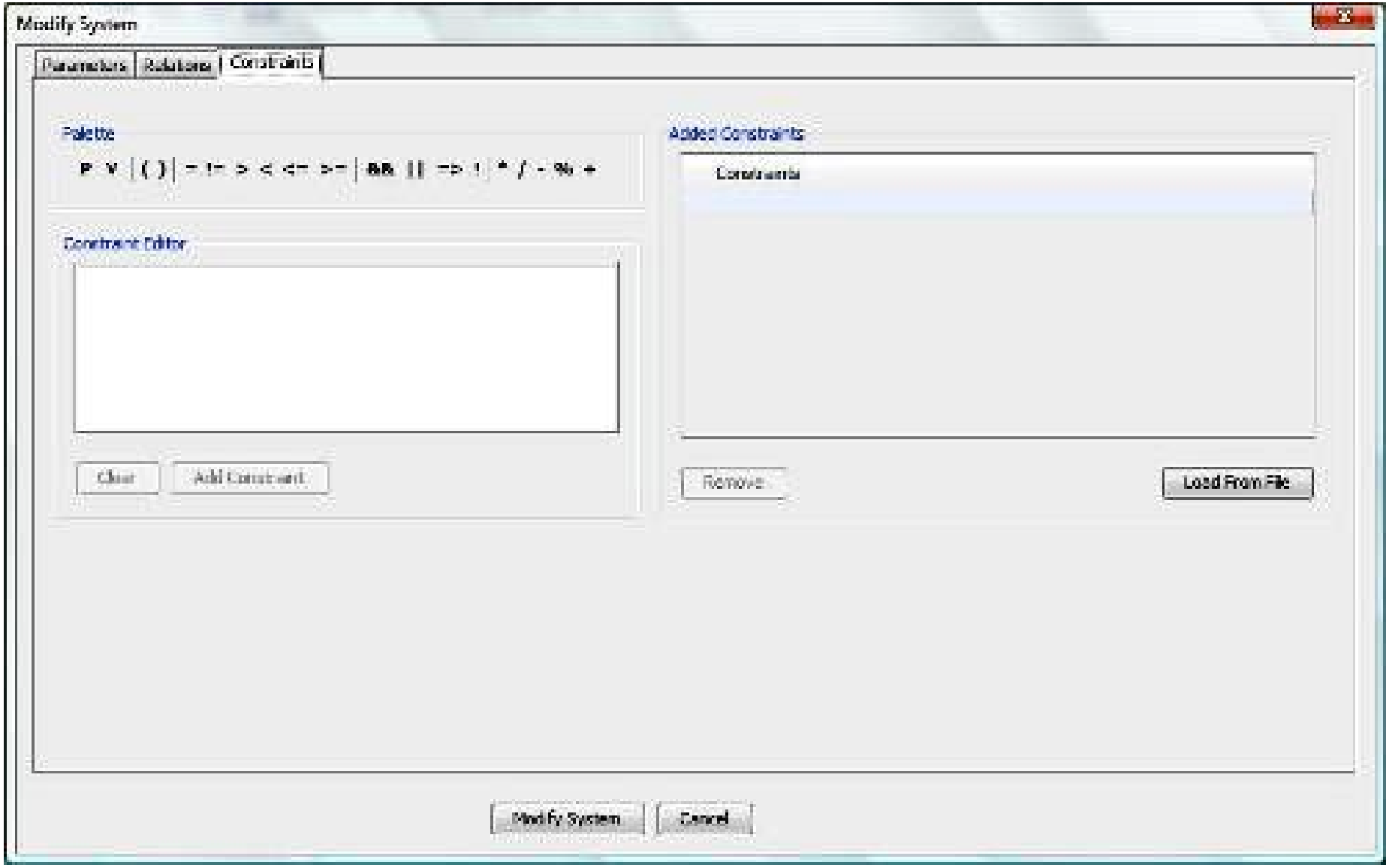
ACTS - Defining a new system



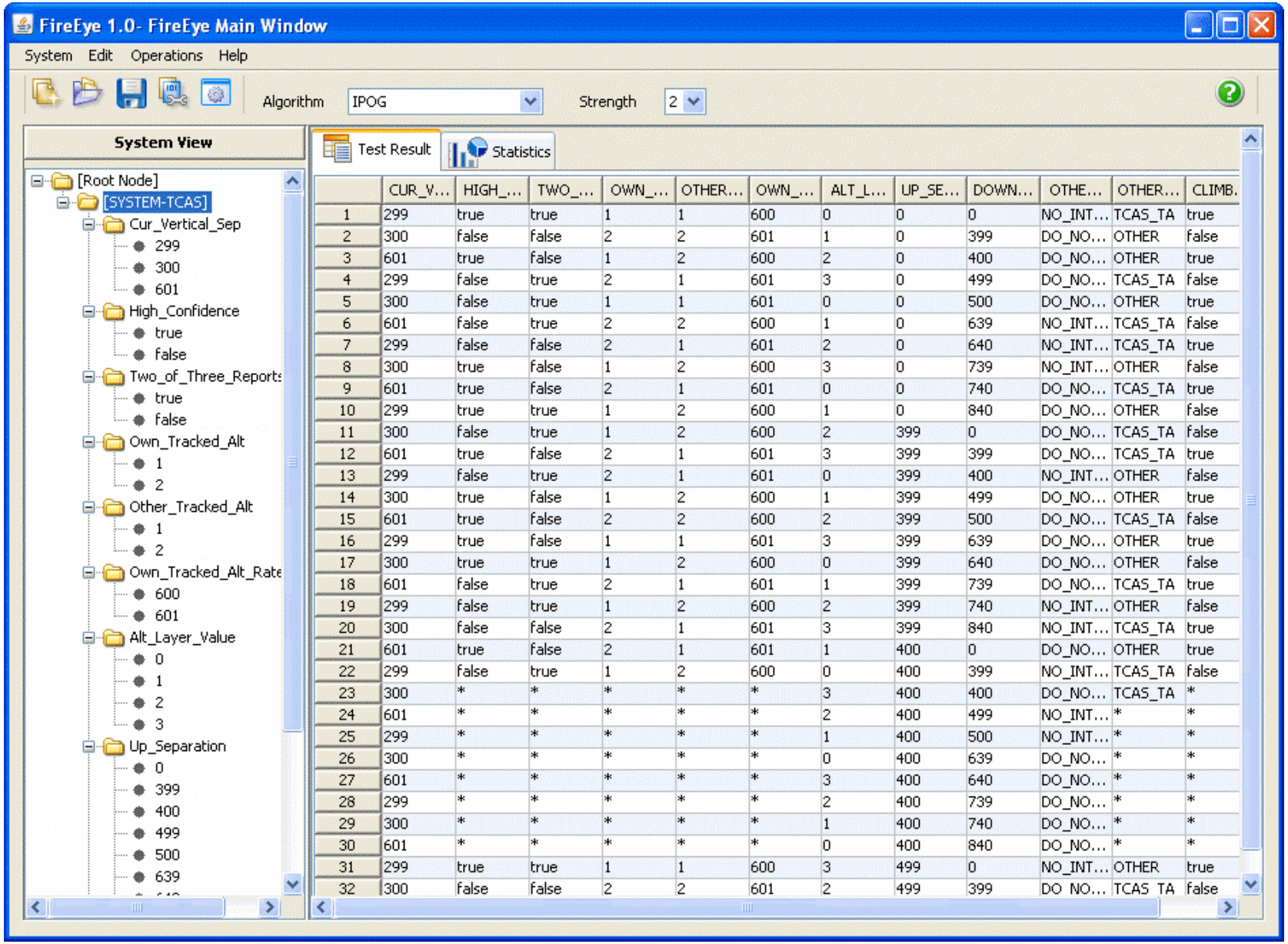
Variable interaction strength



Constraints



Covering array output



	CUR_V...	HIGH_...	TWO_...	OWN_...	OTHER...	OWN_...	ALT_L...	UP_SE...	DOWN...	OTHE...	OTHER...	CLIMB.
1	299	true	true	1	1	600	0	0	0	NO_INT...	TCAS_TA	true
2	300	false	false	2	2	601	1	0	399	DO_NO...	OTHER	false
3	601	true	false	1	2	600	2	0	400	DO_NO...	OTHER	true
4	299	false	true	2	1	601	3	0	499	DO_NO...	TCAS_TA	false
5	300	false	true	1	1	601	0	0	500	DO_NO...	OTHER	true
6	601	false	true	2	2	600	1	0	639	NO_INT...	TCAS_TA	false
7	299	false	false	2	1	601	2	0	640	NO_INT...	TCAS_TA	true
8	300	true	false	1	2	600	3	0	739	NO_INT...	OTHER	false
9	601	true	false	2	1	601	0	0	740	DO_NO...	TCAS_TA	true
10	299	true	true	1	2	600	1	0	840	DO_NO...	OTHER	false
11	300	false	true	1	2	600	2	399	0	DO_NO...	TCAS_TA	false
12	601	true	false	2	1	601	3	399	399	DO_NO...	TCAS_TA	true
13	299	false	true	2	1	601	0	399	400	NO_INT...	OTHER	false
14	300	true	false	1	2	600	1	399	499	DO_NO...	OTHER	true
15	601	true	false	2	2	600	2	399	500	DO_NO...	TCAS_TA	false
16	299	true	false	1	1	601	3	399	639	DO_NO...	OTHER	true
17	300	true	true	1	2	600	0	399	640	DO_NO...	OTHER	false
18	601	false	true	2	1	601	1	399	739	DO_NO...	TCAS_TA	true
19	299	false	true	1	2	600	2	399	740	NO_INT...	OTHER	false
20	300	false	false	2	1	601	3	399	840	NO_INT...	TCAS_TA	true
21	601	true	false	2	1	601	1	400	0	DO_NO...	OTHER	true
22	299	false	true	1	2	600	0	400	399	NO_INT...	TCAS_TA	false
23	300	*	*	*	*	*	3	400	400	DO_NO...	TCAS_TA	*
24	601	*	*	*	*	*	2	400	499	NO_INT...	*	*
25	299	*	*	*	*	*	1	400	500	NO_INT...	*	*
26	300	*	*	*	*	*	0	400	639	DO_NO...	*	*
27	601	*	*	*	*	*	3	400	640	DO_NO...	*	*
28	299	*	*	*	*	*	2	400	739	DO_NO...	*	*
29	300	*	*	*	*	*	1	400	740	DO_NO...	*	*
30	601	*	*	*	*	*	0	400	840	DO_NO...	*	*
31	299	true	true	1	1	600	3	499	0	NO_INT...	OTHER	true
32	300	false	false	2	2	601	2	499	399	DO_NO...	TCAS_TA	false

NIST
National Institute of Standards and Technology

Output options

Mappable values

```
Degree of interaction
coverage: 2
Number of parameters: 12
Number of tests: 100

-------------------------------

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

Etc.

Human readable

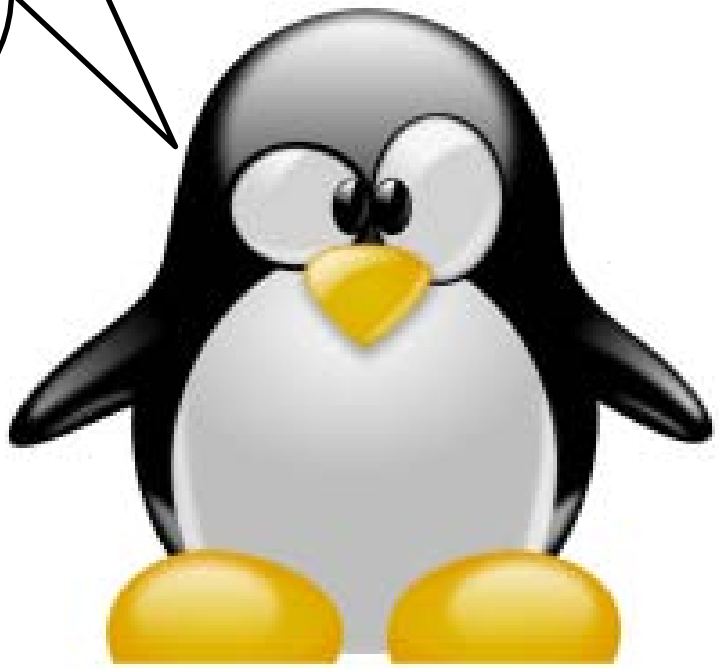
```
Degree of interaction coverage: 2
Number of parameters: 12
Maximum number of values per
parameter: 10
Number of configurations: 100
-----------------------------------
Configuration #1:

1 = Cur_Vertical_Sep=299
2 = High_Confidence=true
3 = Two_of_Three_Reports=true
4 = Own_Tracked_Alt=1
5 = Other_Tracked_Alt=1
6 = Own_Tracked_Alt_Rate=600
7 = Alt_Layer_Value=0
8 = Up_Separation=0
9 = Down_Separation=0
10 = Other_RAC=NO_INTENT
11 = Other_Capability=TCAS_CA
12 = Climb_Inhibit=true
```

Available Tools

- **Covering array generator** – basic tool for test input or configurations;
- **Sequence covering array generator** – new concept; applies combinatorial methods to event sequence testing
- **Combinatorial coverage measurement** – detailed analysis of combination coverage; automated generation of supplemental tests; helpful for integrating c/t with existing test methods
- **Domain/application specific tools**:
  - Access control policy tester
  - .NET config file generator

Is this stuff really useful in the real world??



Real world use - Document Object Model Events

Original test set:

Event Name	Param.	Tests
Abort	3	12
Blur	5	24
Click	15	4352
Change	3	12
dblClick	15	4352
DOMActivate	5	24
DOMAttrModified	8	16
DOMCharacterDataModified	8	64
DOMElementNameChanged	6	8
DOMFocusIn	5	24
DOMFocusOut	5	24
DOMNodeInserted	8	128
DOMNodeInsertedIntoDocument	8	128
DOMNodeRemoved	8	128
DOMNodeRemovedFromDocument	8	128
DOMSubTreeModified	8	64
Error	3	12
Focus	5	24
KeyDown	1	17
KeyUp	1	17
Load	3	24
MouseDown	15	4352
MouseMove	15	4352
MouseOut	15	4352
MouseOver	15	4352
MouseUp	15	4352
MouseWheel	14	1024
Reset	3	12
Resize	5	48
Scroll	5	48
Select	3	12
Submit	3	12
TextInput	5	8
Unload	3	24
Wheel	15	4096
Total Tests		36626

Exhaustive testing of equivalence class values

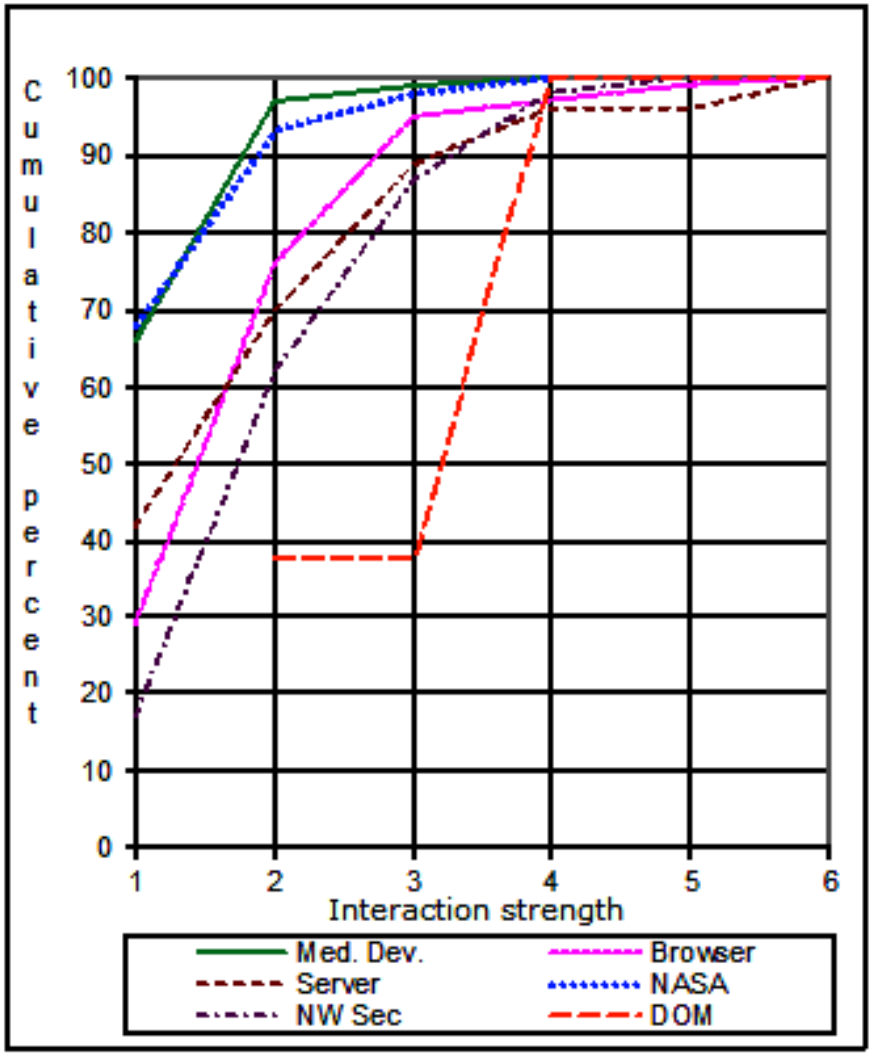
NIST National Institute of Standards and Technology

Document Object Model Events

Combinatorial test set:

t	Tests	% of Orig.	Test Results		
			Pass	Fail	Not Run
2	702	1.92%	202	27	473
3	1342	3.67%	786	27	529
4	1818	4.96%	437	72	1309
5	2742	7.49%	908	72	1762
6	4227	11.54 %	1803	72	2352

All failures found using < 5% of original exhaustive test set



NIST National Institute of Standards and Technology

Modeling & Simulation for Network Security

- "Simured" network simulator
  - Kernel of ~ 5,000 lines of C++ (not including GUI)
- Objective: detect configurations that can produce deadlock:
  - Prevent connectivity loss when changing network
  - Attacks that could lock up network
- Compare effectiveness of random vs. combinatorial inputs
- Deadlock combinations discovered
- Crashes in >6% of tests w/ valid values (Win32 version only)

NIST
National Institute of Standards and Technology

Simulation Input Parameters

Parameter		Values
1	DIMENSIONS	1,2,4,6,8
2	NODOSDIM	2,4,6
3	NUMVIRT	1,2,3,8
4	NUMVIRTINJ	1,2,3,8
5	NUMVIRTEJE	1,2,3,8
6	LONBUFFER	1,2,4,6
7	NUMDIR	1,2
8	FORWARDING	0,1
9	PHYSICAL	true, false
10	ROUTING	0,1,2,3
11	DELFIFO	1,2,4,6
12	DELCROSS	1,2,4,6
13	DELCHANNEL	1,2,4,6
14	DELSWITCH	1,2,4,6

5x3x4x4x4x4x2x2
x2x4x4x4x4x4
= 31,457,280
configurations

Are any of them dangerous?

If so, how many?

Which ones?

NIST
National Institute of Standards and Technology

Network Deadlock Detection

Deadlocks Detected: combinatorial

t	Tests	500 pkts	1000 pkts	2000 pkts	4000 pkts	8000 pkts
2	28	0	0	0	0	0
3	161	2	3	2	3	3
4	752	14	14	14	14	14

Average Deadlocks Detected: random

t	Tests	500 pkts	1000 pkts	2000 pkts	4000 pkts	8000 pkts
2	28	0.63	0.25	0.75	0. 50	0. 75
3	161	3	3	3	3	3
4	752	10.13	11.75	10.38	13	13.25

NIST
National Institute of Standards and Technology

Network Deadlock Detection

Detected 14 configurations that can cause deadlock:

$14/ 31{,}457{,}280 = 4.4 \times 10^{-7}$

Combinatorial testing found more deadlocks than random, including some that might never have been found with random testing

Why do this testing? Risks:

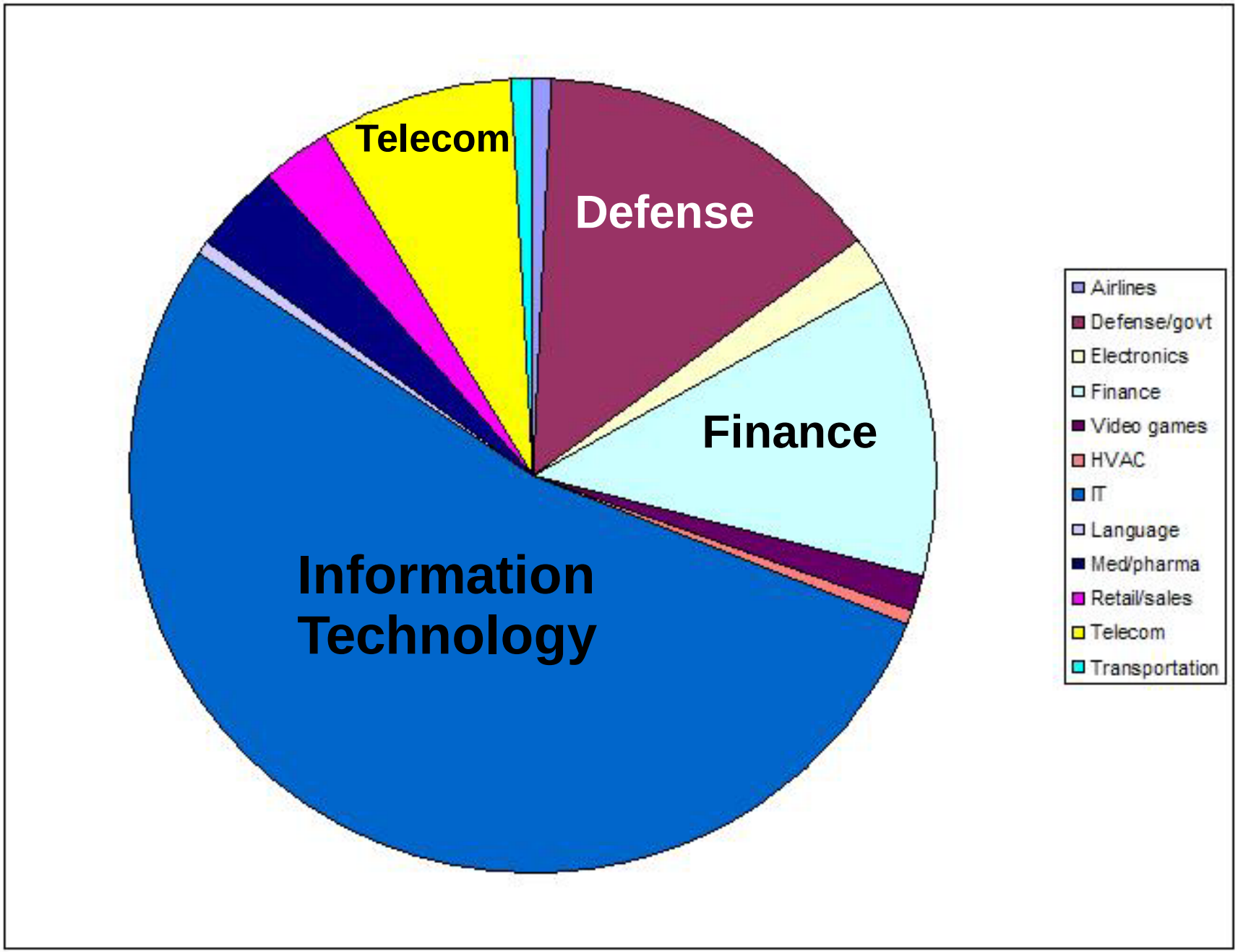
- accidental deadlock configuration: low
- deadlock config discovered by attacker: **much higher**

(because they are looking for it)

Buffer Overflows

- Empirical data from the National Vulnerability Database
  - Investigated > 3,000 denial-of-service vulnerabilities reported in the NIST NVD for period of 10/06 – 3/07
  - Vulnerabilities triggered by:
    - **Single variable – 94.7%**
      example: *Heap-based buffer overflow in the SFTP protocol handler for Panic Transmit … allows remote attackers to execute arbitrary code via a long ftps:// URL.*
    - **2-way interaction – 4.9%**
      example: *single character search string in conjunction with a single character replacement string, which causes an "off by one overflow"*
    - **3-way interaction – 0.4%**
      example: *Directory traversal vulnerability when register_globals is enabled and magic_quotes is disabled and .. (dot dot) in the page parameter*

ACTS User Distribution

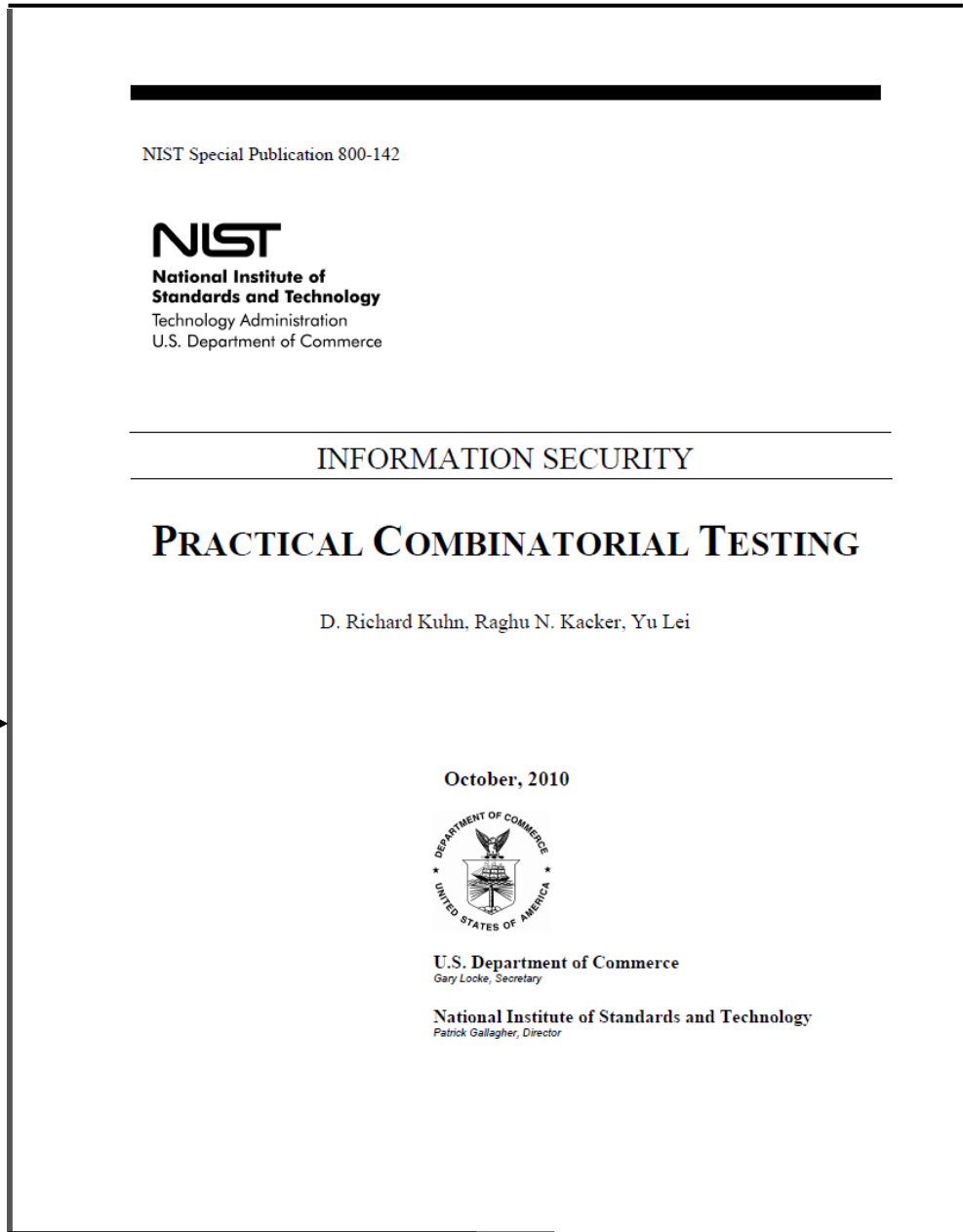


Some our users

Integrating into Testing Program

- Test suite development
  - Generate covering arrays for tests OR
  - Measure coverage of existing tests and supplement

- Training
  - Testing textbooks – Ammann & Offutt, Mathur
  - Combinatorial testing "textbook" on ACTS site
  - User manuals
  - Worked examples



NIST Special Publication 800-142

NIST
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce

INFORMATION SECURITY

PRACTICAL COMBINATORIAL TESTING

D. Richard Kuhn, Raghu N. Kacker, Yu Lei

October, 2010

U.S. Department of Commerce
Gary Locke, Secretary

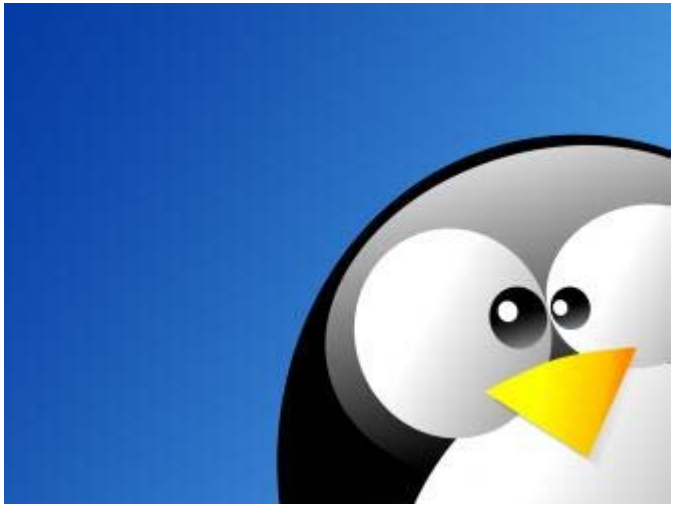
National Institute of Standards and Technology
Patrick Gallagher, Director

Industrial Usage Reports

- Cooperative Research & Development Agreement with Lockheed Martin - report 2011
- Previously reported:
  Johns Hopkins Applied Physics Lab, US Air Force, Accenture, DOM Level 3 events conformance test
- New work with NASA IV&V

NIST
National Institute of Standards and Technology

Please contact us if you are interested.



Rick Kuhn
kuhn@nist.gov

Raghu Kacker
raghu.kacker@nist.gov

http://csrc.nist.gov/acts

NIST
National Institute of Standards and Technology