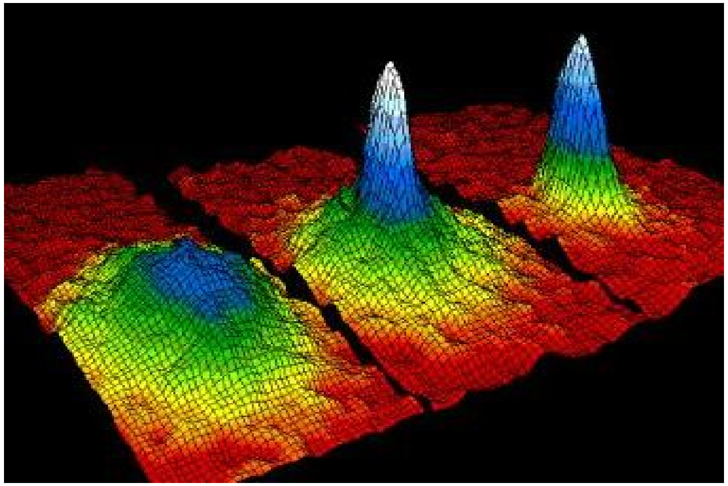
Introduction to Combinatorial Testing

Rick Kuhn
National Institute of
Standards and Technology
Gaithersburg, MD

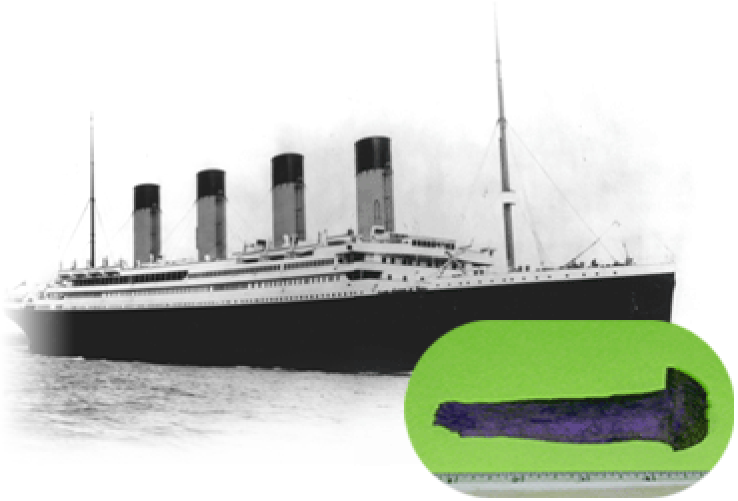
Carnegie-Mellon University, 7 June 2011

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 (1-way interaction <= all-values testing catches)
pressure < 10 & volume > 300 (2-way interaction <= all-pairs testing catches)

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

Pairwise testing is popular, but is it enough?

- Pairwise testing commonly applied to software
- Intuition: some problems only occur as the result of an interaction between parameters/components
- Tests all pairs (2-way combinations) of variable values
- Pairwise testing finds about 50% to 90% of flaws

Finding 90% of flaws is pretty good, right?

"Relax, our engineers found 90 percent of the flaws."

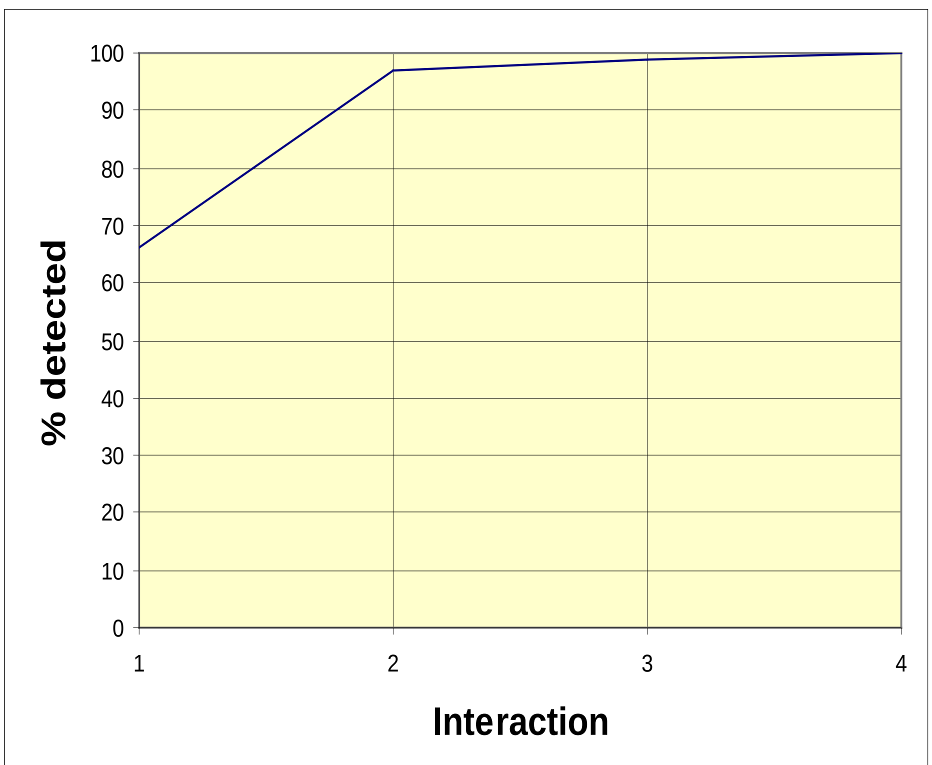
I don't think I want to get on that plane.

National Institute of Standards and Technology

How about hard-to-find flaws?

NIST
National Institute of Standards and Technology

- 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



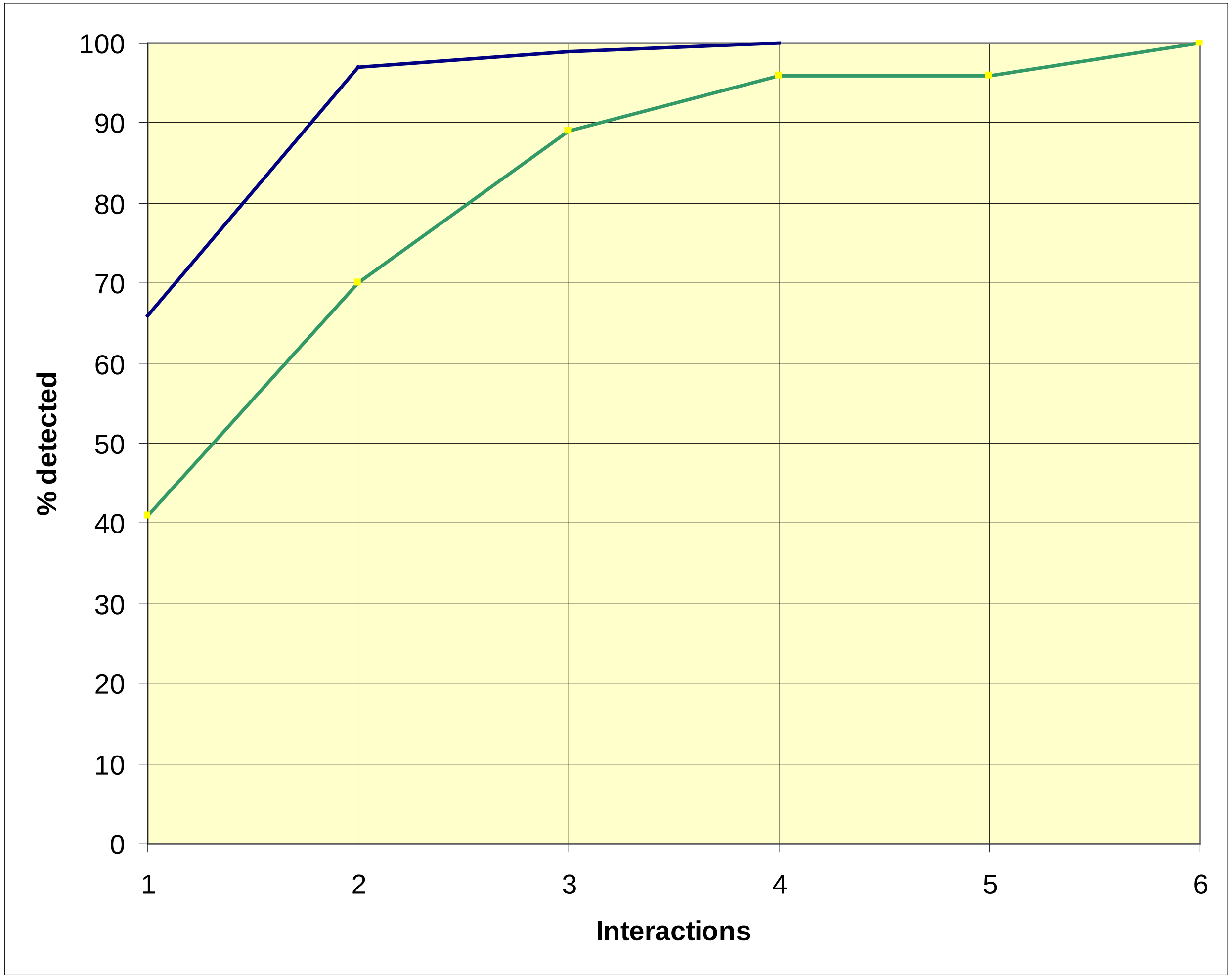
NIST study of 15 years of FDA medical device recall data

Interesting, but that's just one kind of application.

How about other applications?

NIST National Institute of Standards and Technology

Browser (green)

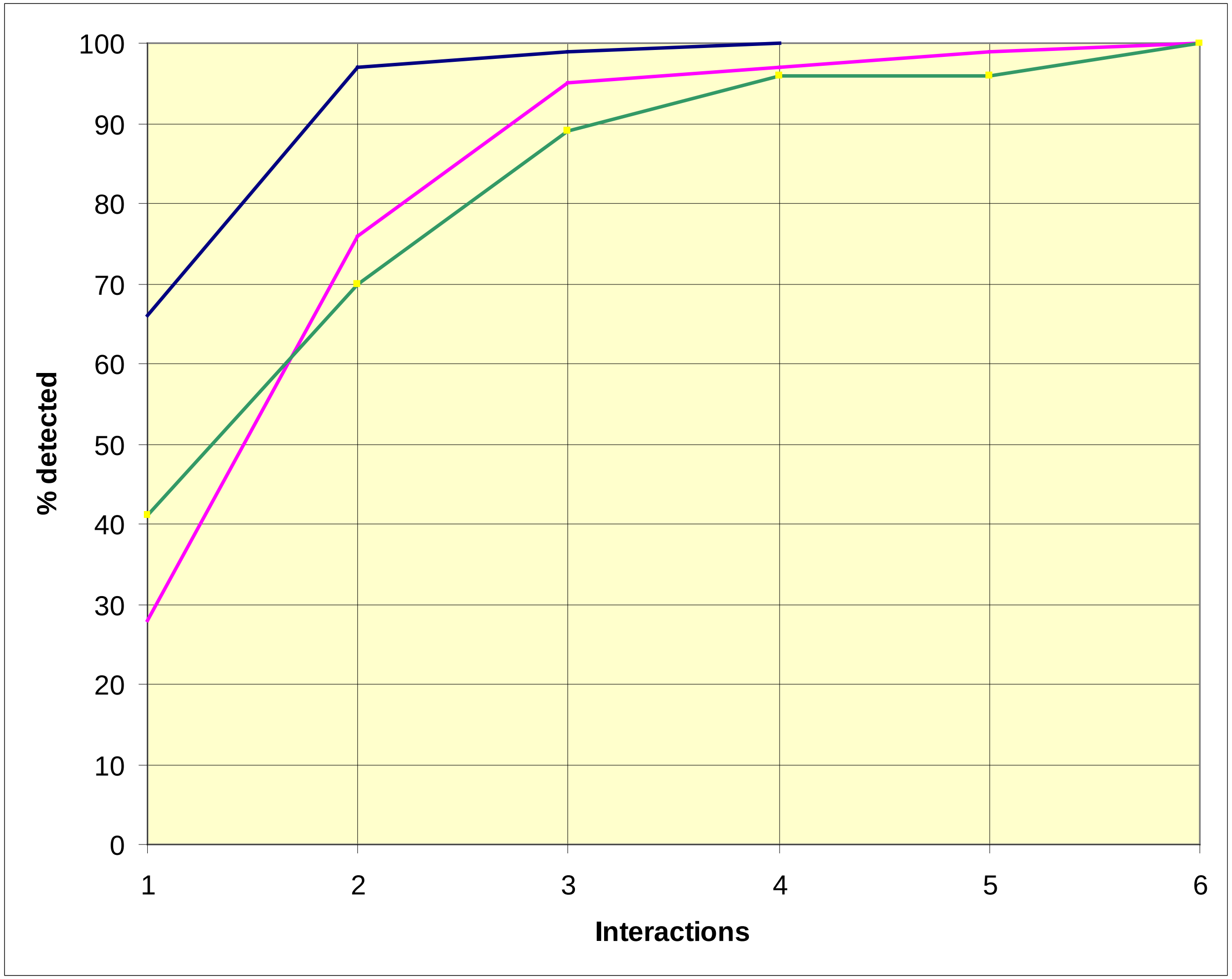


These faults more complex than medical device software!!

Why?

And other applications?

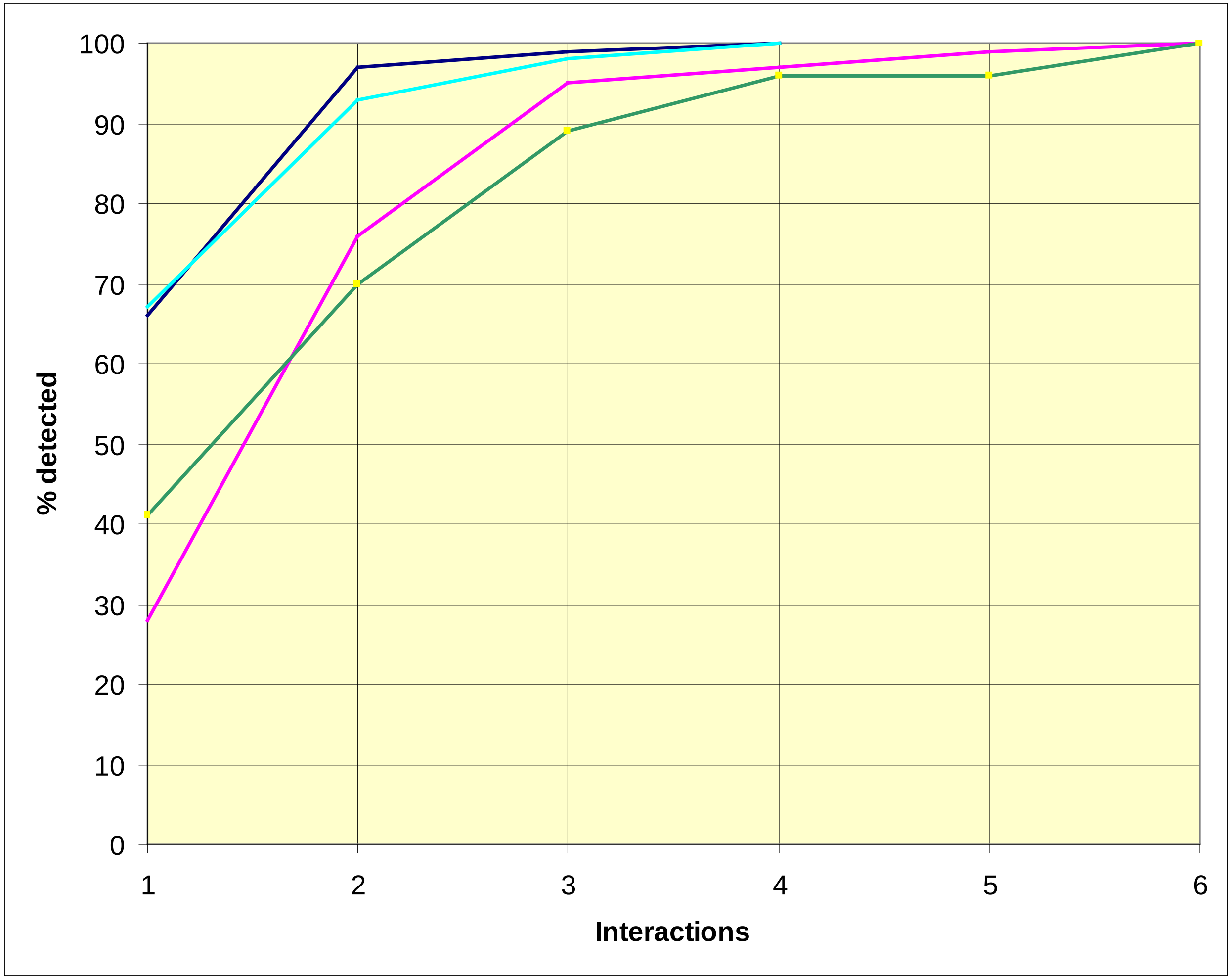
Server (magenta)



Still more?

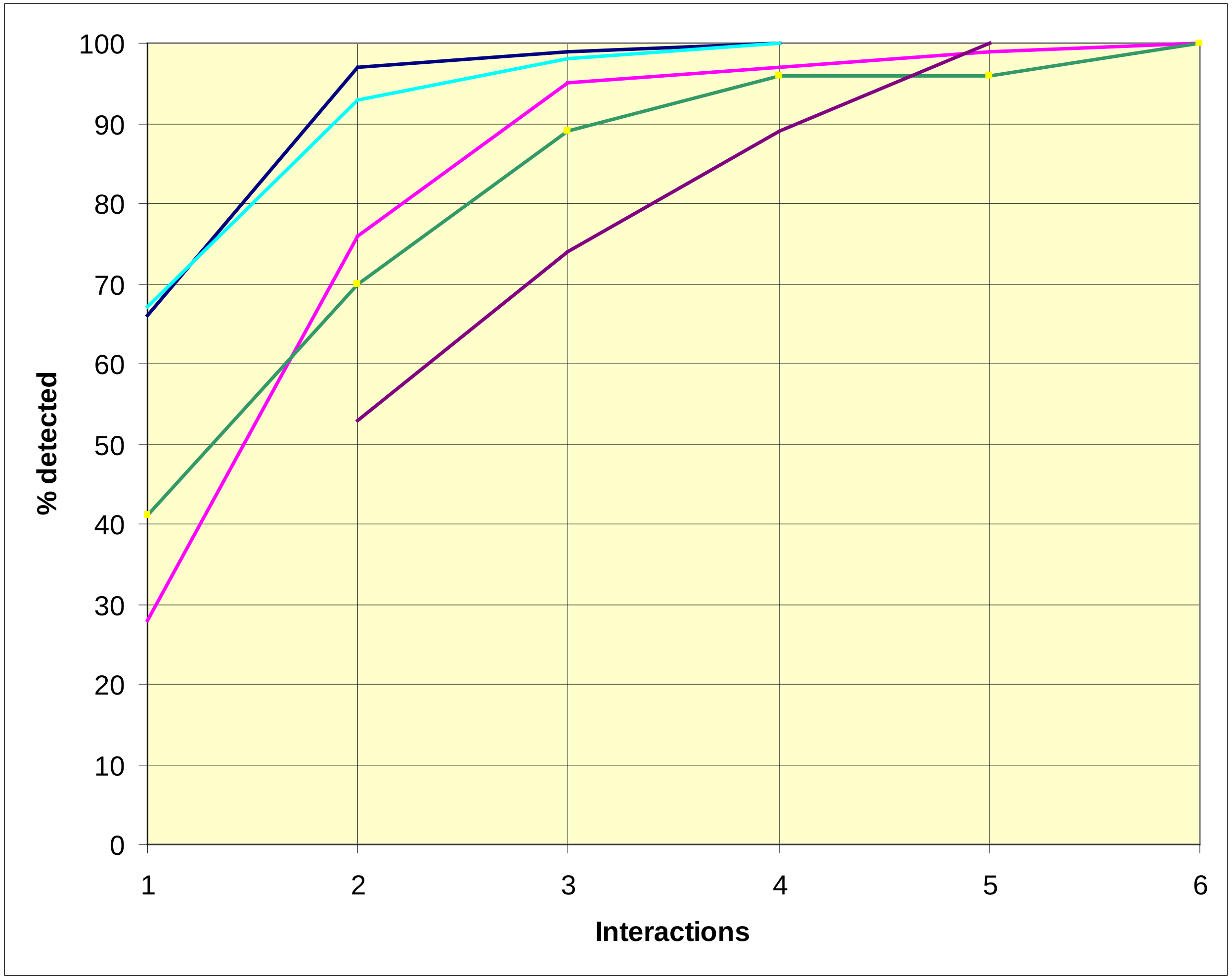
NASA distributed database

(light blue)



Even more?

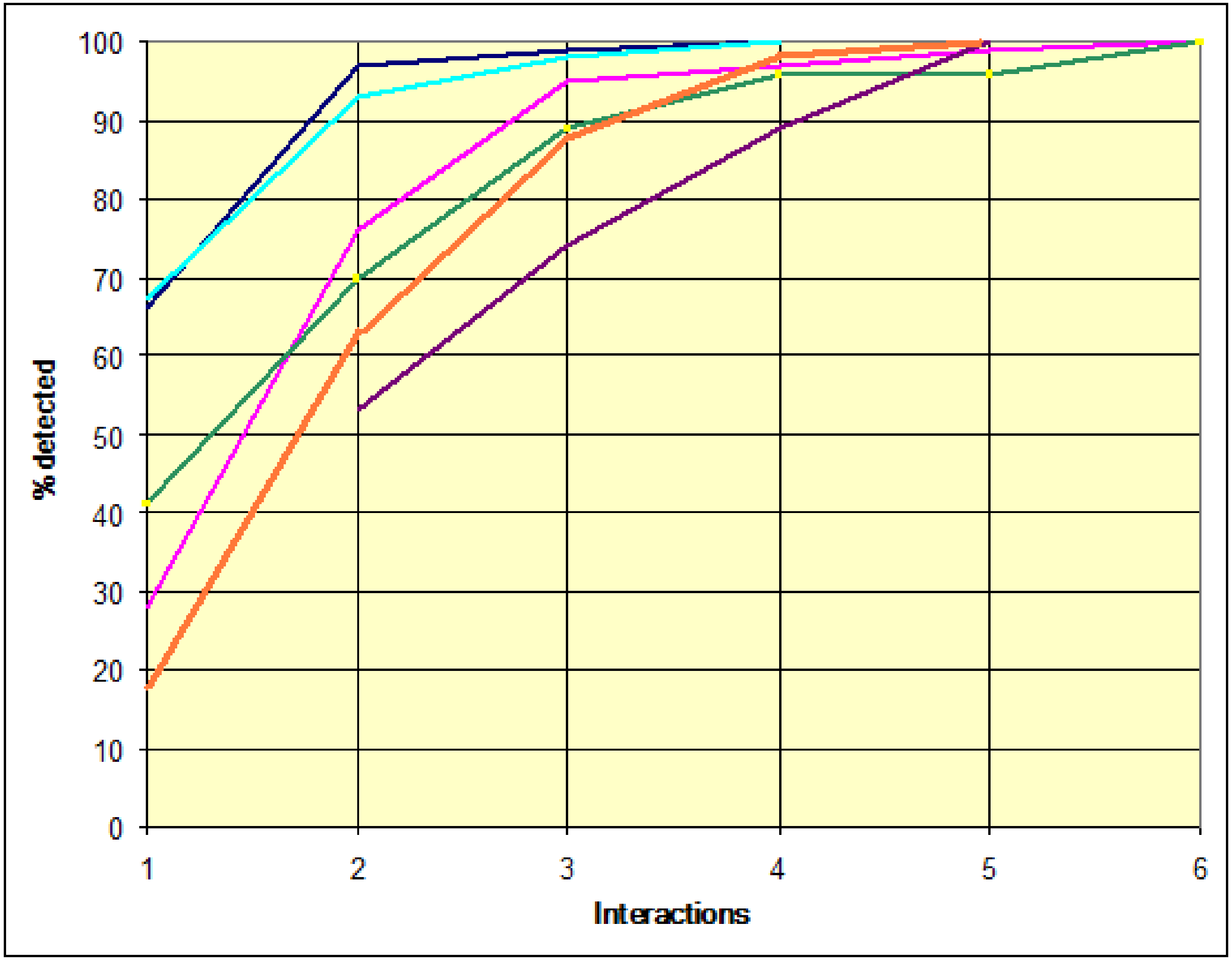
Traffic Collision Avoidance System module (seeded errors) (purple)



Finally

Network security (Bell, 2006)

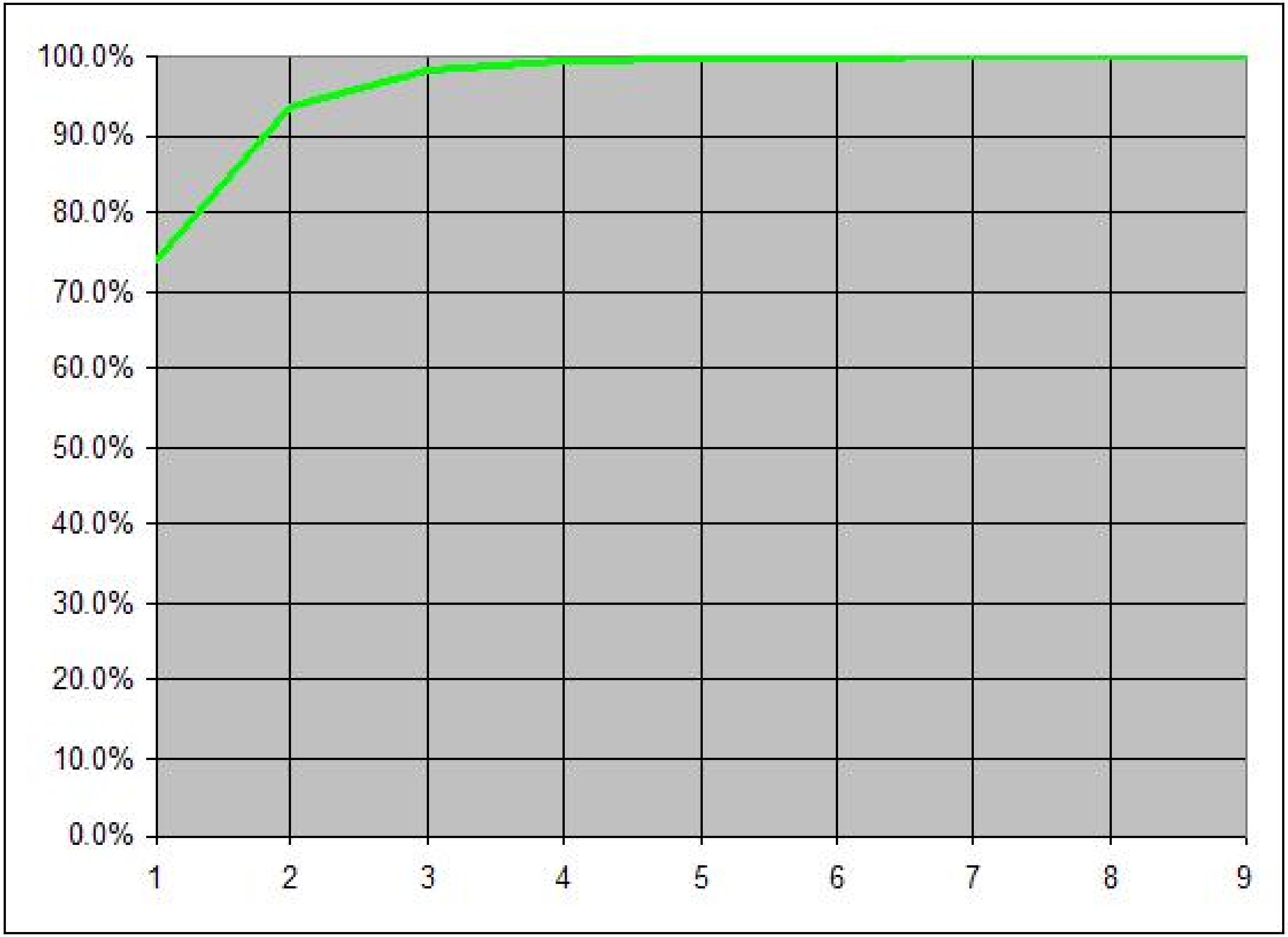
(orange)



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

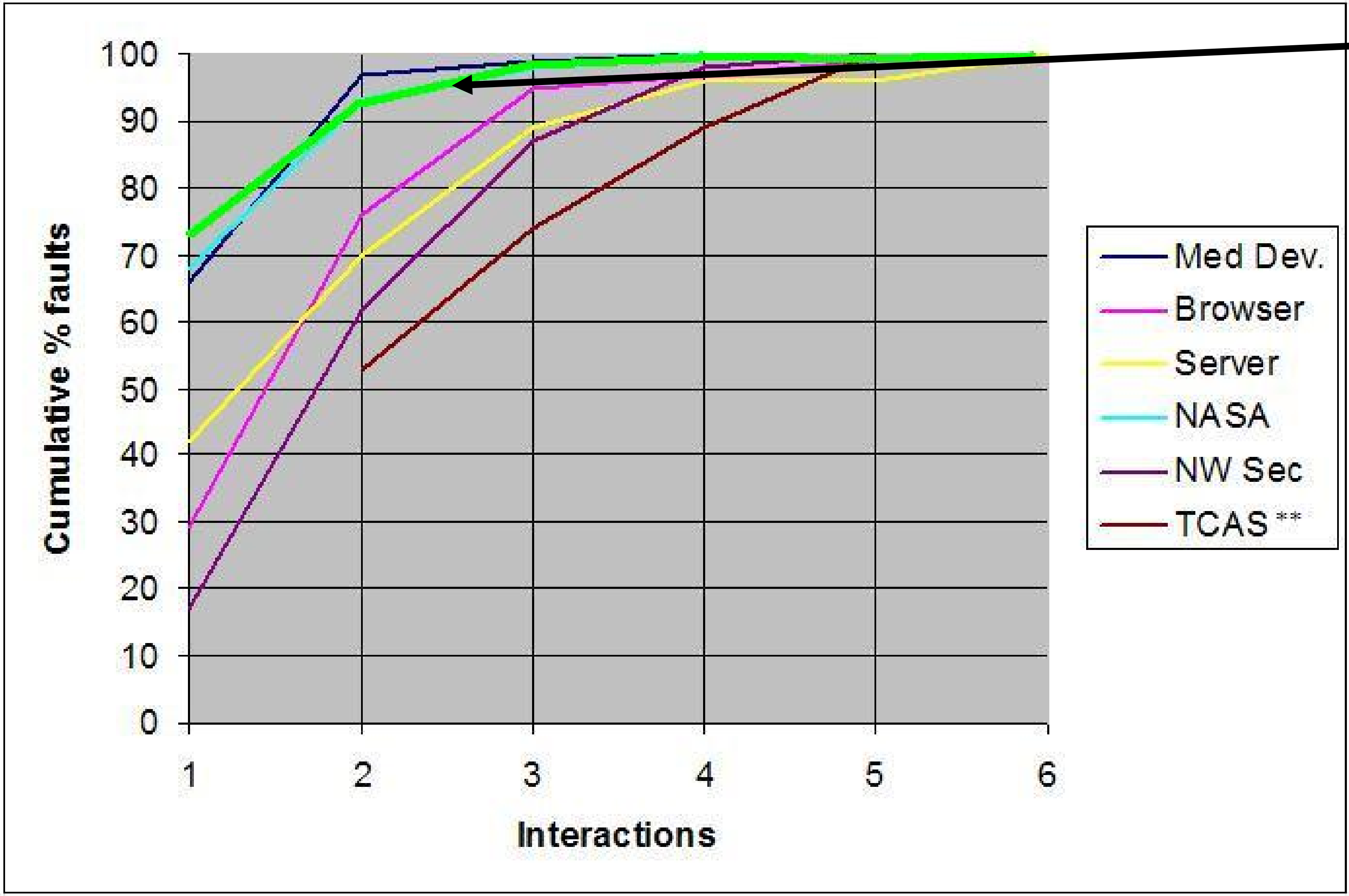
Why this distribution?

What causes this distribution?



One clue: branches in avionics software.
7,685 expressions from *if* and *while* statements

Comparing with Failure Data



Branch statements

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

- Maximum interactions for fault triggering for these applications was 6
- Much more empirical work needed
- Reasonable evidence that maximum interaction strength for fault triggering is relatively small

How does it help me to know this?

How does this knowledge help?

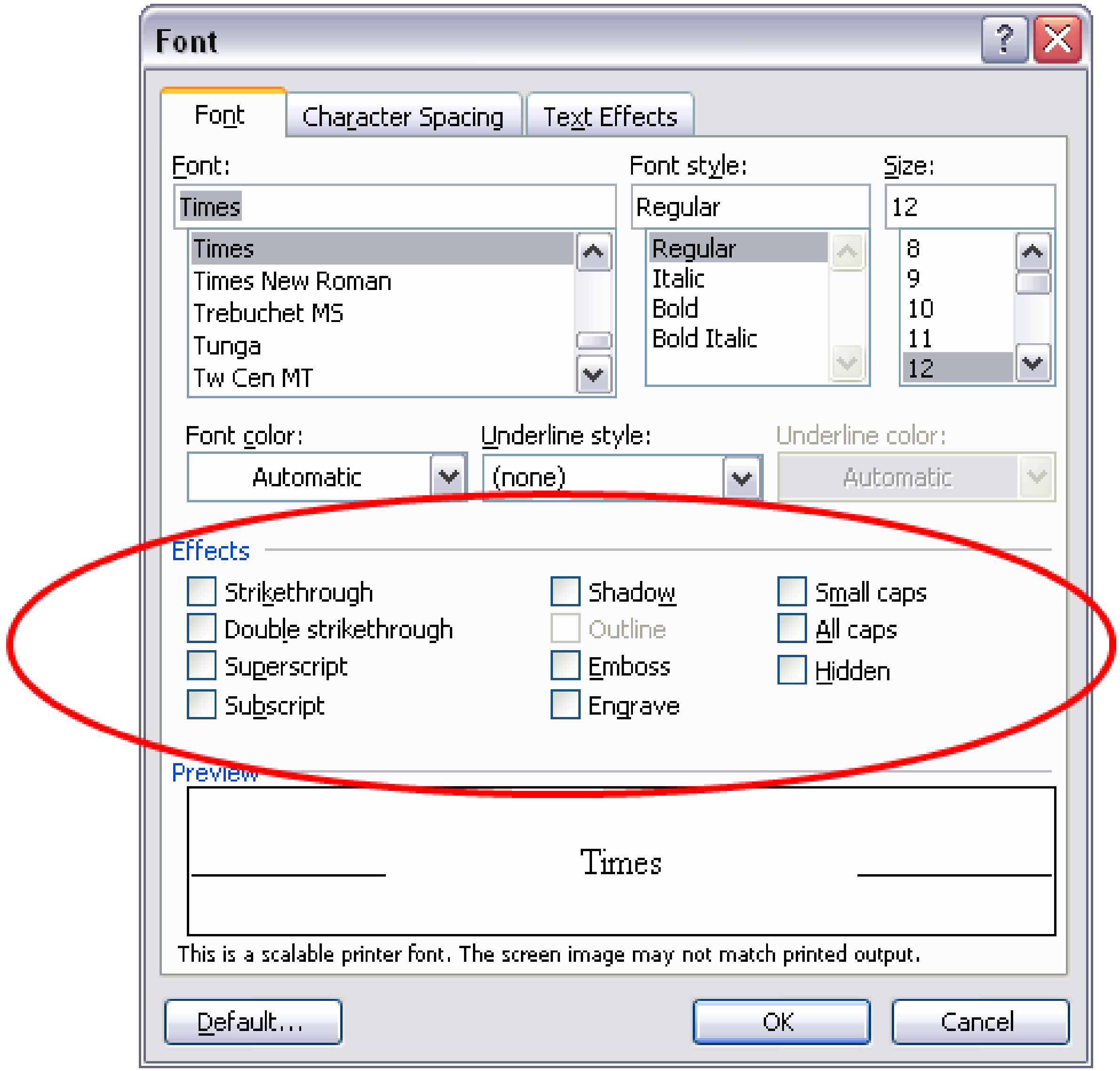
Biologists have a "central dogma", and so do we:

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, . . .)

Still no silver bullet. Rats!

What is combinatorial testing? A simple example



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:

0 1 1 0 0 0 0 1 1 0

We can pack a lot into one test, so what's the smallest number of tests we need?

National Institute of Standards and Technology

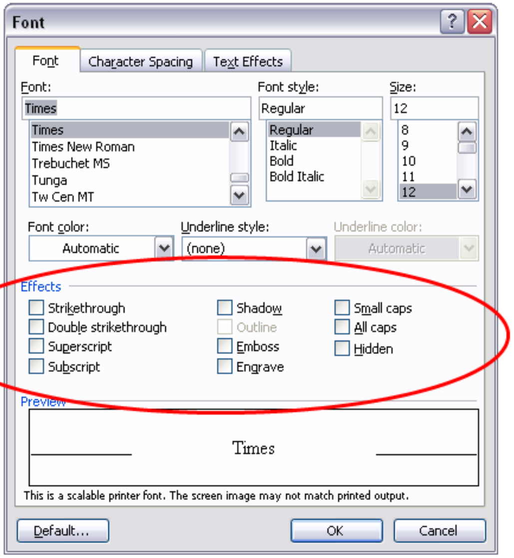
A covering array

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:



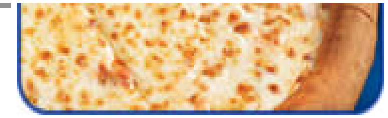
Each test covers $\binom{10}{3} = 120$ 3-way combinations

<u>Finding covering arrays is NP hard</u>

National Institute of Standards and Technology

Ordering Pizza

NIST
National Institute of Standards and Technology



Step 1 *Select your favorite size and pizza crust.*

Large Original Crust

Step 2

Select your favorite pizza toppings from the pull down. Whole toppings cover the entire pizza. First ½ and second ½ toppings cover half the pizza. For a regular cheese pizza, do not add toppings.

☑ I want to add or remove toppings on this pizza -- add on whole or half pizza.

Add toppings whole pizza — Extra Cheese (Remove) — Bacon (Remove) — Black Olives (Remove)

Add toppings 1st half

Add toppings 2nd half

$6 \times 2^{17} \times 2^{17} \times 2^{17} \times 4 \times 3 \times 2 \times 2 \times 5 \times 2$

= WAY TOO MUCH TO TEST

Simplified pizza ordering:

$6 \times 4 \times 4 \times 4 \times 4 \times 3 \times 2 \times 2 \times 5 \times 2$

= 184,320 possibilities

Step 3 *Select your pizza instructions.*

☑ I want to add special instructions for this pizza -- light, extra or no sauce; light or no cheese; well done bake

Regular Sauce | Normal Cheese | Normal Bake | Normal Cut

Step 4 *Add to order.*

Quantity 1

Add To Order | Add To Order & Checkout

Ordering Pizza Combinatorially

Simplified pizza ordering:

6x4x4x4x4x3x2x2x5x2
= 184,320 possibilities

2-way tests: 32

3-way tests: 150

4-way tests: 570

5-way tests: 2,413

6-way tests: 8,330

If all failures involve 5 or fewer parameters, then we can have confidence after running all 5-way tests.

A larger example

- Suppose we have a system with on-off switches:

National Institute of Standards and Technology

How do we test this?

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

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

- 34 switches = 2^{34} = 1.7×10^{10} possible inputs = **1.7×10^{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

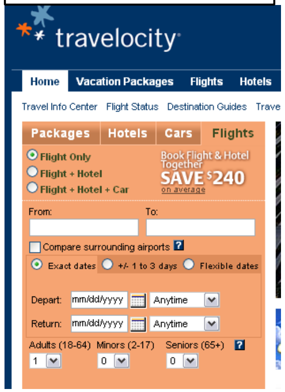
Use combinations here

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

NIST National Institute of Standards and Technology

Testing Configurations

- Example: app must run on any configuration of OS, browser, protocol, CPU, and DBMS
- Very effective for interoperability 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

NIST
National Institute of Standards and Technology

Configurations to Test

Degree of interaction coverage: 2
Number of parameters: 5
Maximum number of values per parameter: 3
Number of configurations: 10

Configuration #1:
1 = OS=XP
2 = Browser=IE
3 = Protocol=IPv4
4 = CPU=Intel
5 = DBMS=MySQL

Configuration #2:
1 = OS=XP
2 = Browser=Firefox
3 = Protocol=IPv6
4 = CPU=AMD
5 = DBMS=Sybase

Configuration #3:
1 = OS=XP
2 = Browser=IE
3 = Protocol=IPv6
4 = CPU=Intel
5 = DBMS=Oracle
. . . etc.

t	# Configs	% of Exhaustive
2	10	14
3	18	25
4	36	50
5	72	100

NIST
National Institute of Standards and Technology

Testing Smartphone Configurations

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:

$$3 \times 3 \times 4 \times 3 \times 5 \times 4 \times 4 \times 5 \times 4 = 172{,}800$$

Number of configurations generated

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

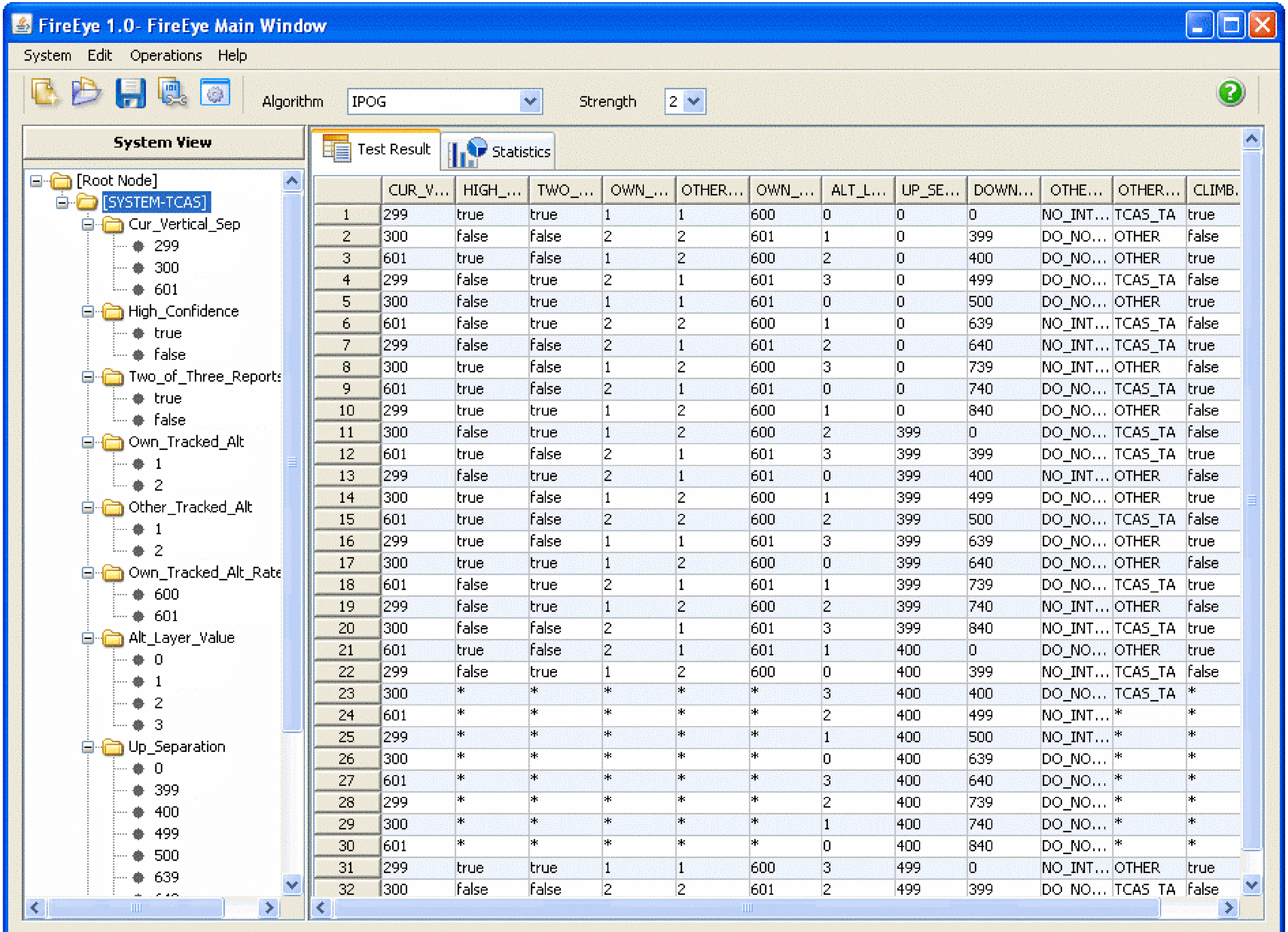
Traffic Collision Avoidance System (TCAS): $2^7 3^2 4^1 10^2$

Times in seconds

That's fast!

Unlike diet plans,
results ARE typical!

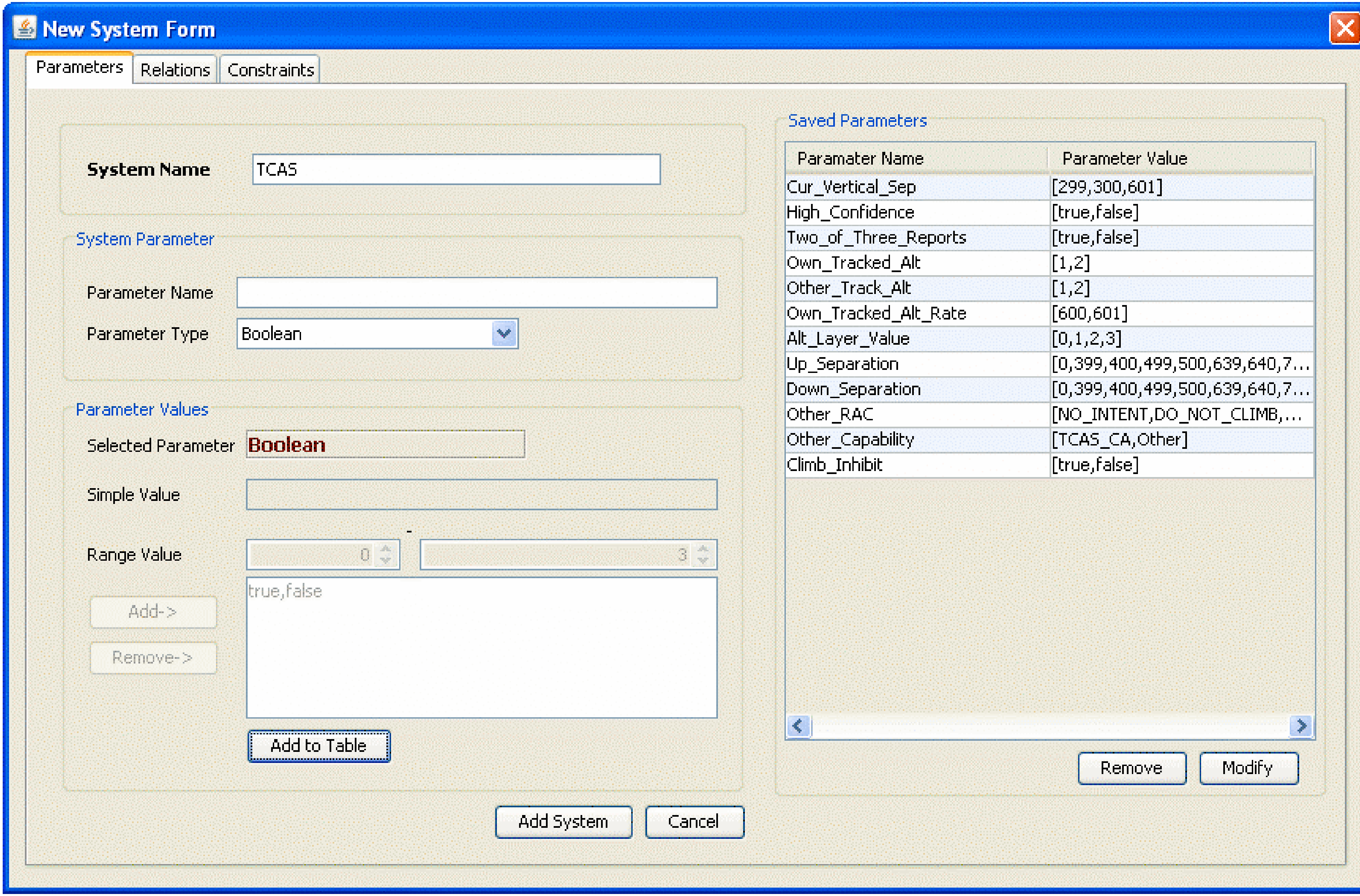
ACTS Tool



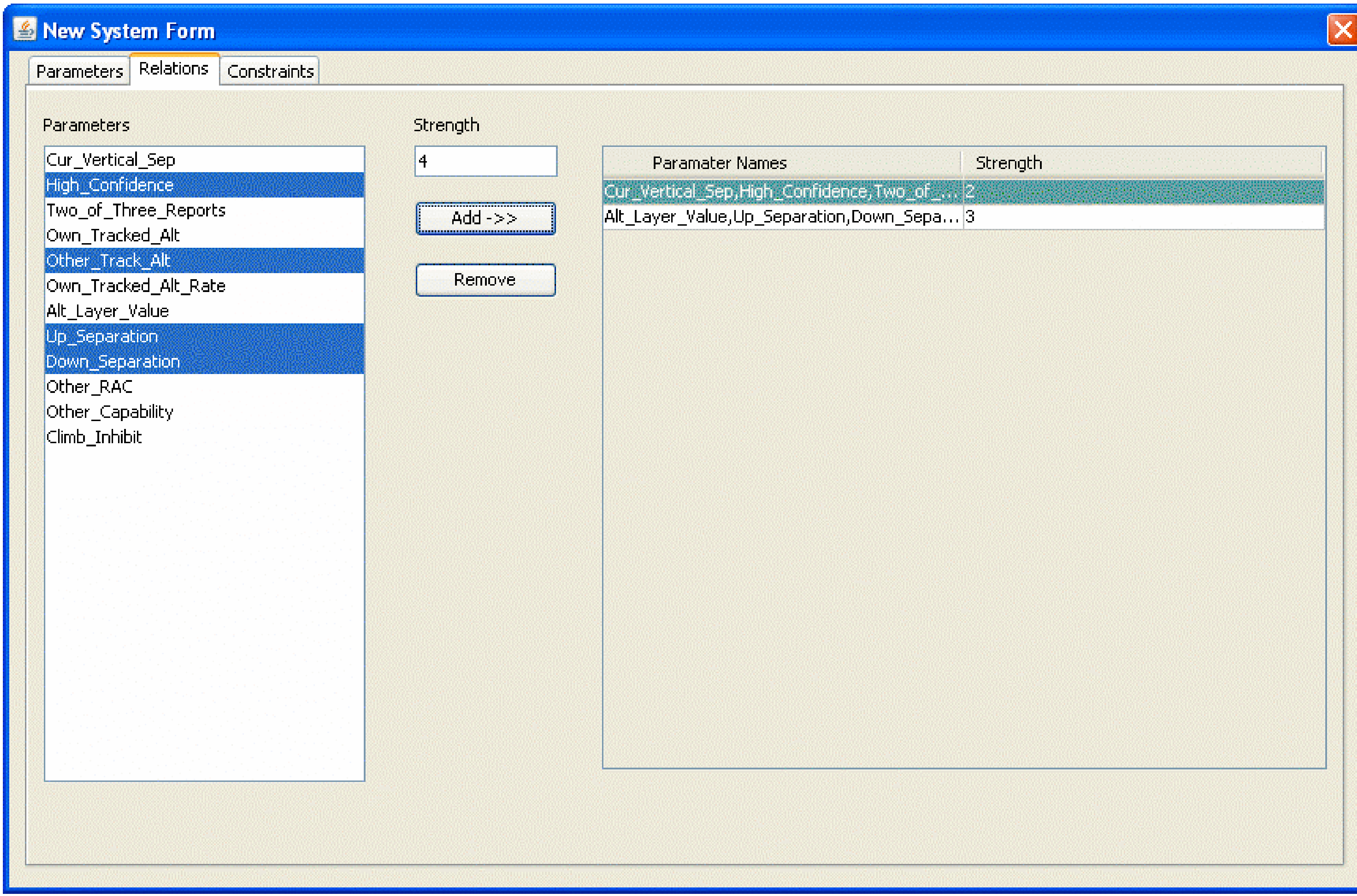
FireEye 1.0- FireEye Main Window
System
Edit
Operations
Help
Algorithm
IPOG
Strength
2
System View
[Root Node]
[SYSTEM-TCAS]
Cur_Vertical_Sep
299
300
601
High_Confidence
true
false
Two_of_Three_Reports
true
false
Own_Tracked_Alt
1
2
Other_Tracked_Alt
1
2
Own_Tracked_Alt_Rate
600
601
Alt_Layer_Value
0
1
2
3
Up_Separation
0
399
400
499
500
639
Test Result
Statistics

	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

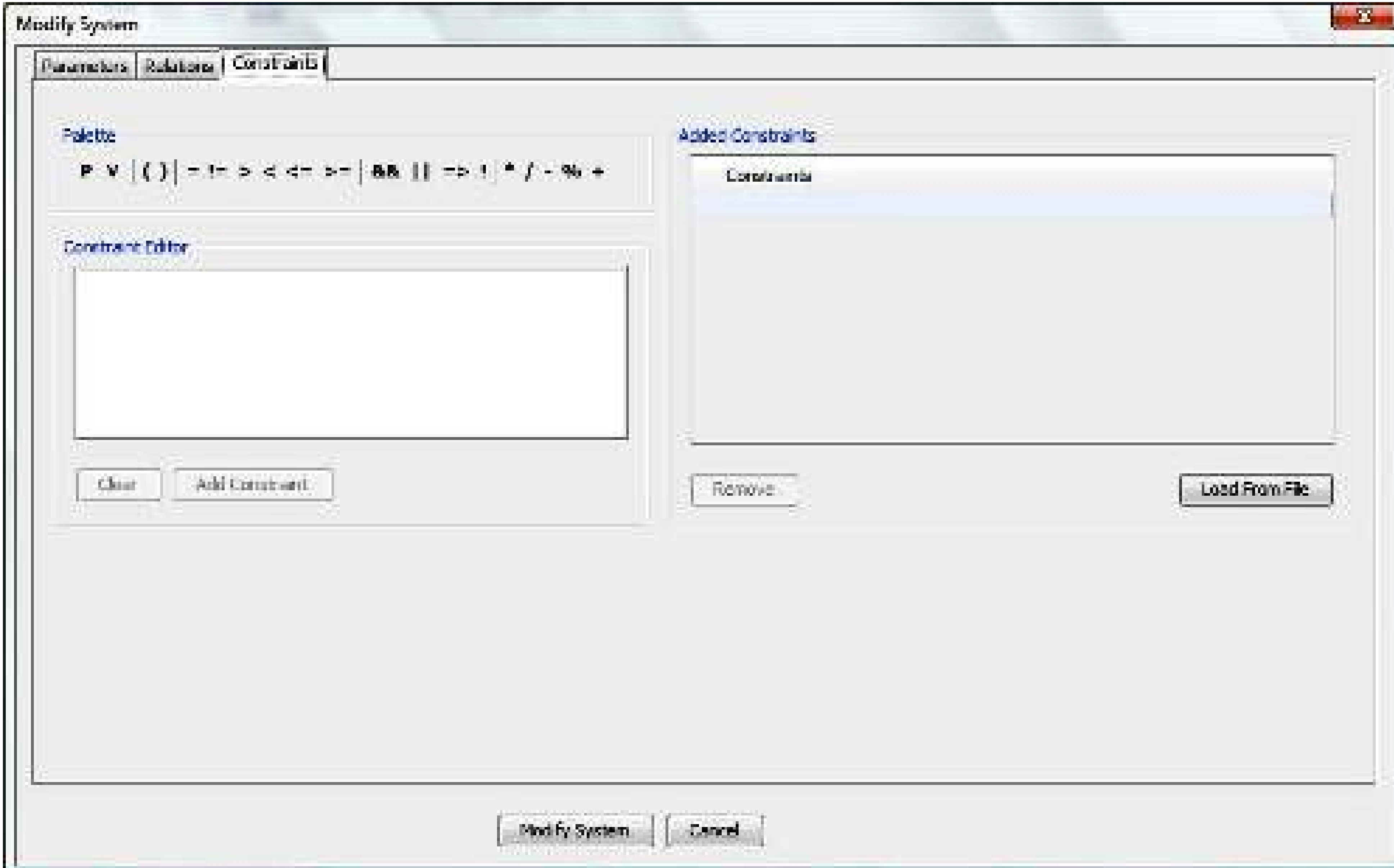
Defining a new system



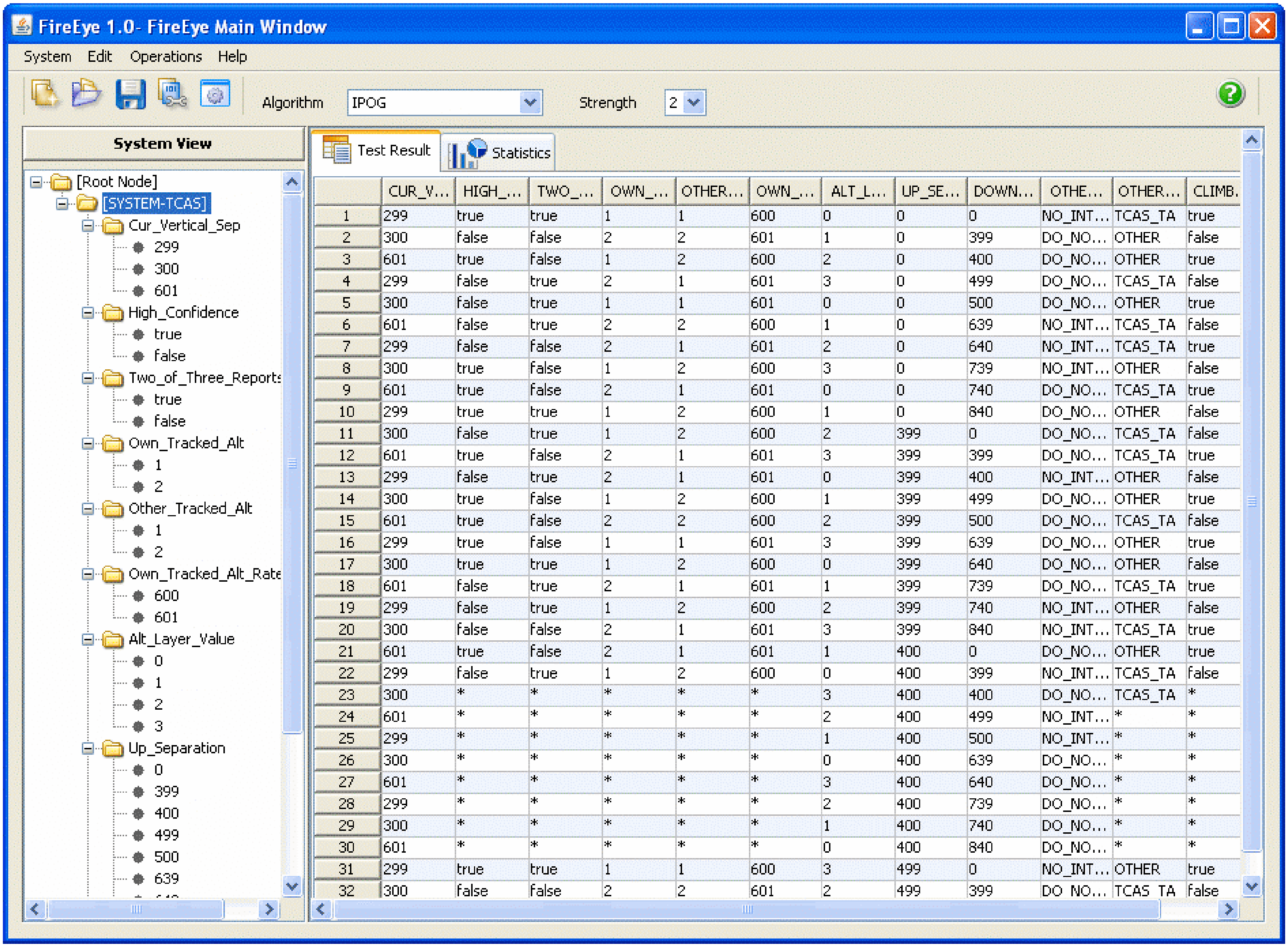
Variable interaction strength



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

Output

- Variety of output formats:
  - XML
  - Numeric
  - CSV
  - Excel
- Separate tool to generate .NET configuration files from ACTS output
- Post-process output using Perl scripts, etc.

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

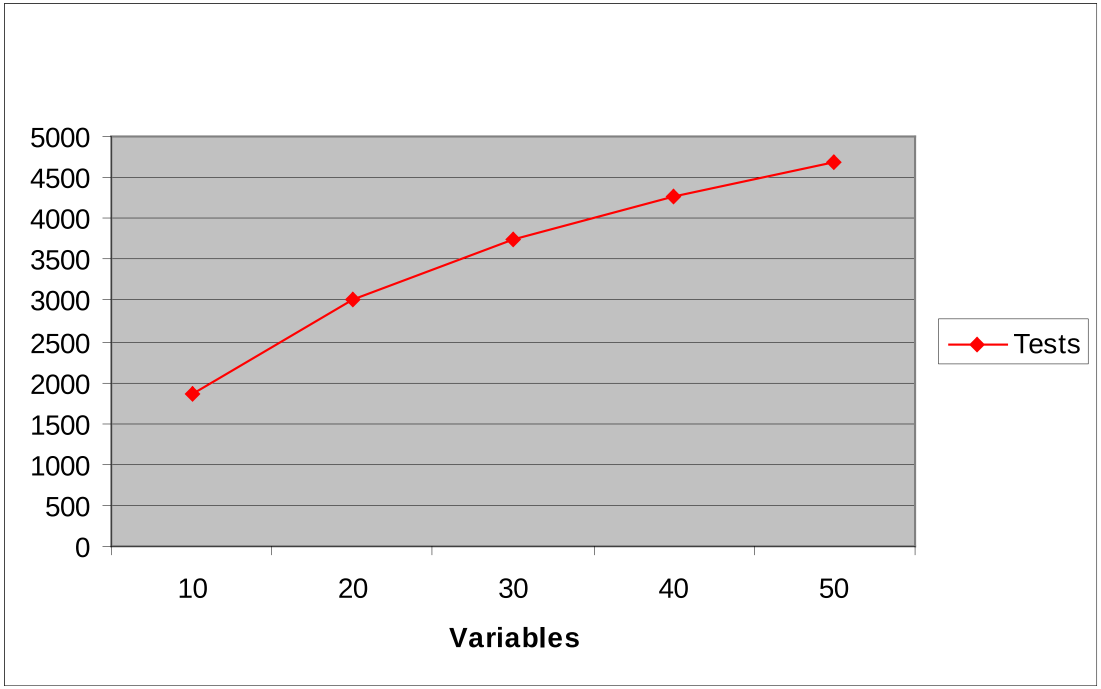
```
Degree of interaction coverage: 2
Number of parameters: 12
Maximum number of values per
parameter: 10
Number of configurations: 100
-----------------------------------
Configuration #1:

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
```

Using ACTS

Cost and Volume of Tests

- Number of tests: proportional to $v^t \log n$
  for v values, n variables, t-way interactions
- *Thus:*
  - Tests increase exponentially with interaction strength *t* : BAD, but unavoidable
  - But only logarithmically with the number of parameters : GOOD!
- Example: suppose we want all 4-way combinations of *n* parameters, 5 values each:



National Institute of Standards and Technology

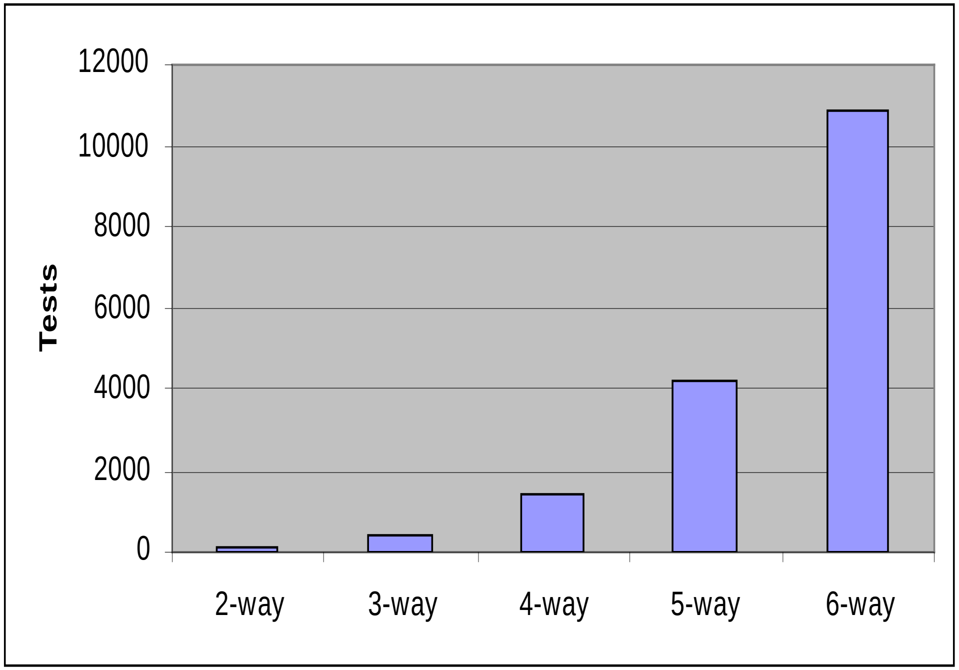
Example 1: Traffic Collision Avoidance System (TCAS) module

- Used in previous testing research
- 41 versions seeded with errors
- 12 variables: 7 boolean, two 3-value, one 4-value, two 10-value
- All flaws found with 5-way coverage
- Thousands of tests - generated by model checker in a few minutes

National Institute of Standards and Technology

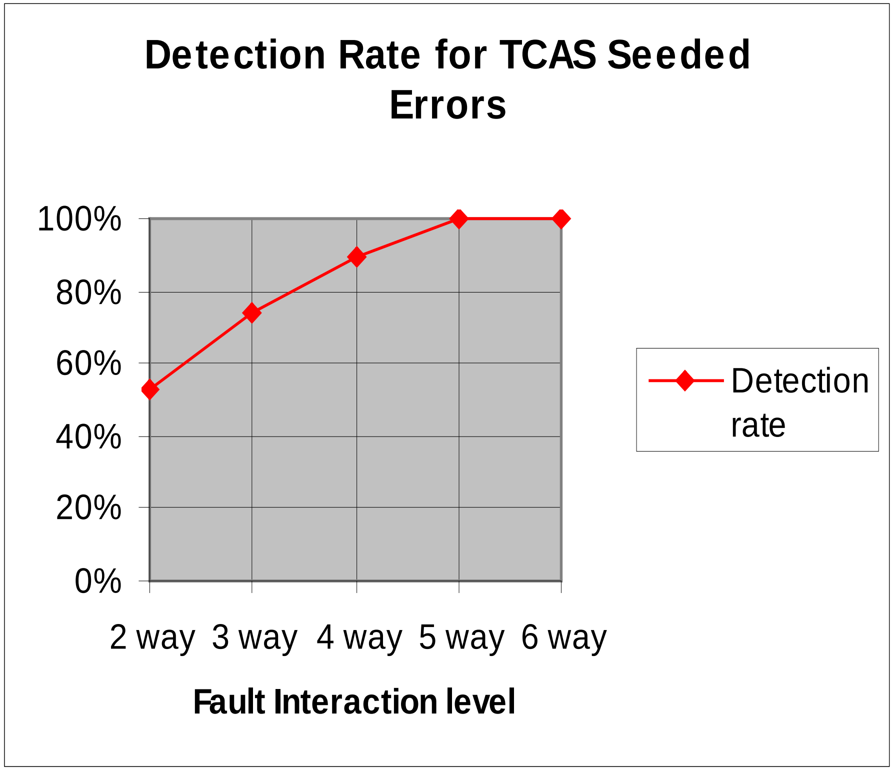
Tests generated

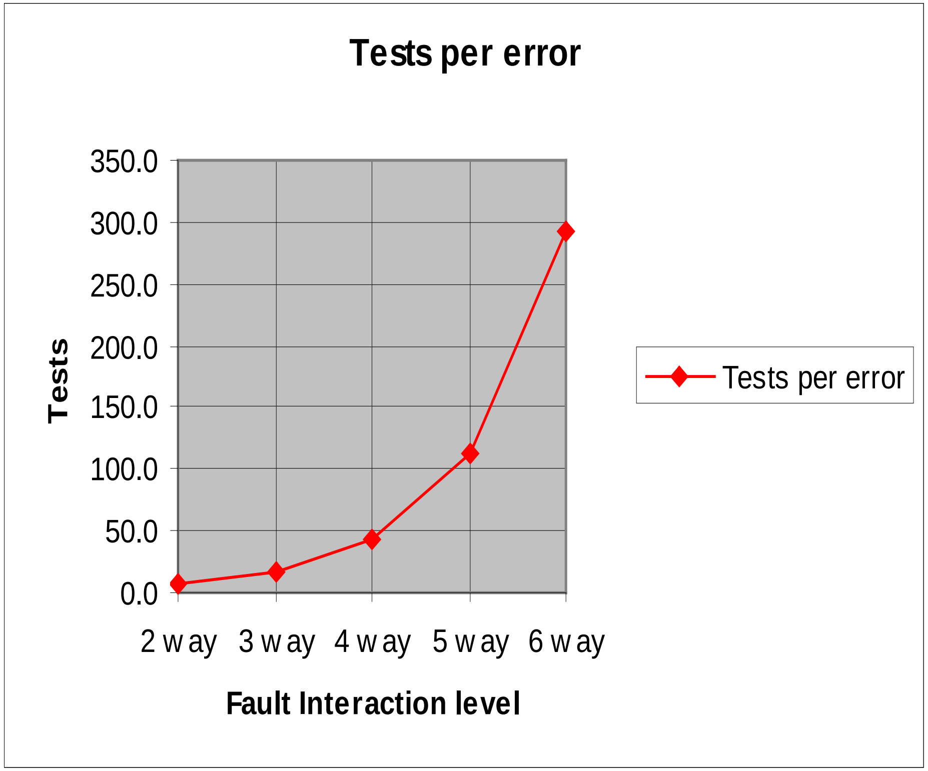
t	Test cases
2-way:	156
3-way:	461
4-way:	1,450
5-way:	4,309
6-way:	11,094



Results

- Roughly consistent with data on large systems
- But errors harder to detect than real-world examples





Bottom line for model checking based combinatorial testing:
Expensive but can be highly effective

Example 2: Document Object Model Events

- DOM is a World Wide Web Consortium standard incorporated into web browsers

- NIST Systems and Software division develops tests for standards such as DOM

- DOM testing problem:
  - large number of events handled by separate functions
  - functions have 3 to 15 parameters
  - parameters have many, often continuous, values
  - verification requires human interaction (viewing screen)
  - testing takes a *long* time

DOM Functions

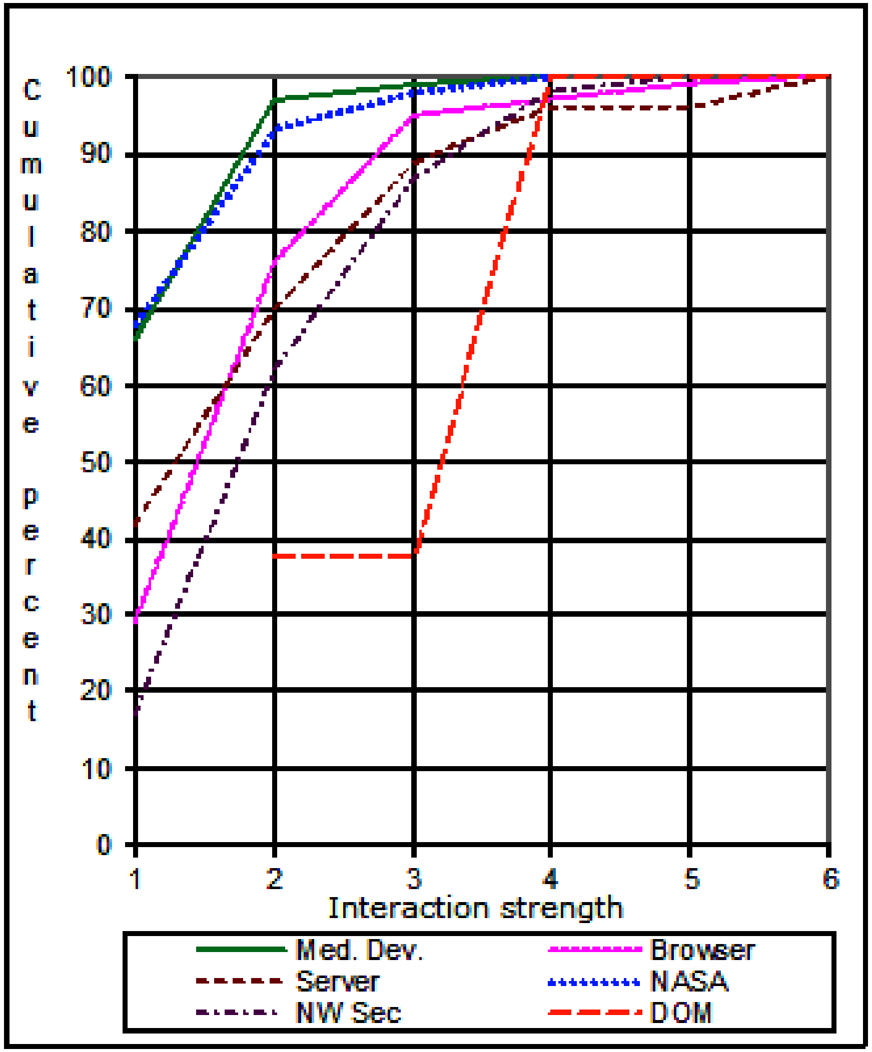
Event Name	Param.	Tests
Abort	3	12
Blur	5	24
Click	15	4352
Change	3	12
dblClick	15	4352
DOMActivate	5	24
DOMAttrModified	8	16
DOMCharacterDataModified	8	64
DOMElementNameChanged	6	8
DOMFocusIn	5	24
DOMFocusOut	5	24
DOMNodeInserted	8	128
DOMNodeInsertedIntoDocument	8	128
DOMNodeRemoved	8	128
DOMNodeRemovedFromDocument	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

World Wide Web Consortium Document Object Model Events

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 discretized test set



SUMMARY

- Combinatorial testing is now a practical approach that produces high quality testing at lower cost

- Good algorithms and user-friendly tools are available – no cost tools from NIST, Microsoft, others

- Basic combinatorial testing can be used in two ways:
  - combinations of configuration values
  - combinations of input values
  - these can be used separately or at the same time

- Case studies are beginning to appear

- All tools and materials available at NIST web site csrc.nist.gov/acts

National Institute of Standards and Technology