NIST
National Institute of Standards and Technology

Combinatorial Testing: Rationale and Impact

Rick Kuhn

National Institute of
Standards and Technology
Gaithersburg, MD

IEEE Seventh International Conference on Software Testing, Verification, and Validation

April 2, 2014

What is NIST?

- US Government agency, whose mission is to support US industry through developing better measurement and test methods
- 3,000 scientists, engineers, and support staff including 4 Nobel laureates
- Research in physics, chemistry, materials, manufacturing, computer science
- Trivia: NIST is one of the only federal agencies chartered in the Constitution

Outline

- What is combinatorial testing?
- Why does it work?
- How does it work?
- Does it work in the real world?
- What are some research issues?

Where did these testing ideas come from?

Scottish physician James Lind determined cure of scurvy

Ship HM Bark Salisbury in 1747

12 sailors "were as similar as I could have them"

6 treatments 2 sailors for each – dilute sulfuric acid, seawater, vinegar, cider, orange/lemon juice, barley water

Principles used (blocking, replication, randomization)

Did not consider interactions, but otherwise used basic **Design of Experiments** principles

Father of DOE:
R A Fisher, 1890-1962, British geneticist

Key features of Design of Experiments

- Blocking
- Replication
- Randomization
- Orthogonal arrays test interactions between <u>two</u> factors

Test	P1	P2	P3
1	1	1	3
2	1	2	2
3	1	3	1
4	2	1	2
5	2	2	1
6	2	3	3
7	3	1	1
8	3	2	3
9	3	3	2

Each combination occurs same number of times, usually once.

Example: P1, P2 = 1,2

National Institute of Standards and Technology

Four eras in evolution of DOE

Era 1:(1920's …): Beginning in agricultural then animal science, clinical trials, medicine

Era 2:(1940's …): Industrial productivity – new field, same basics

Era 3:(1980's …): Designing robust products – new field, same basics

Then things begin to change . . .

Era 4:(2000's …): Combinatorial Testing of Software

National Institute of Standards and Technology

Agriculture and biological investigations

System under investigation

- Crop growing, effectiveness of drugs or other treatments

Variable Types

- Primary test factors (things farmer can adjust, drug dosages, etc.)
- Relatively few, held constant

Numbers of treatments

- Generally less than 10

Objectives: compare treatments to find better

Treatments: qualitative or discrete levels of continuous

Manufactured products

Scope of investigation:

- Optimum levels of control factors at which variation from noise factors is minimum

Key principles

- Variation from noise factors
- Efficiency in testing; accommodate constraints

Designs: Based on Orthogonal arrays (OAs)

- Taguchi designs (strength 2 orthogonal arrays)

Orthogonal Arrays for Software Interaction Testing

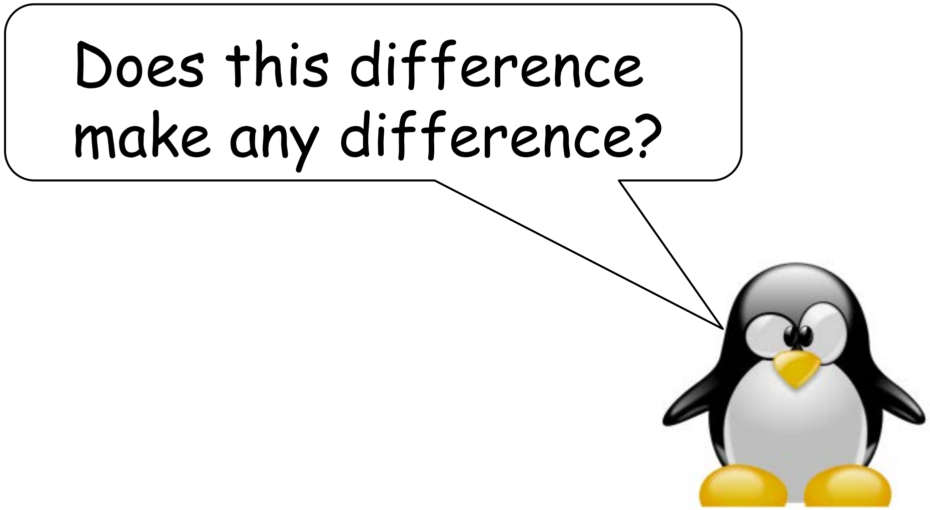
- Functional (black-box) testing
  - Hardware-software systems
  - Identify single and 2-way combination faults
- Early papers
  - Taguchi followers (mid1980's)
  - Mandl (1985) Compiler testing
  - Tatsumi et al (1987) Fujitsu
  - Sacks et al (1989) Computer experiments
  - Brownlie et al (1992) AT&T
- Generation of test suites using orthogonal arrays
  - OATS (Phadke, AT&T-BL)

NIST National Institute of Standards and Technology

What's different about software?

NIST
National Institute of Standards and Technology

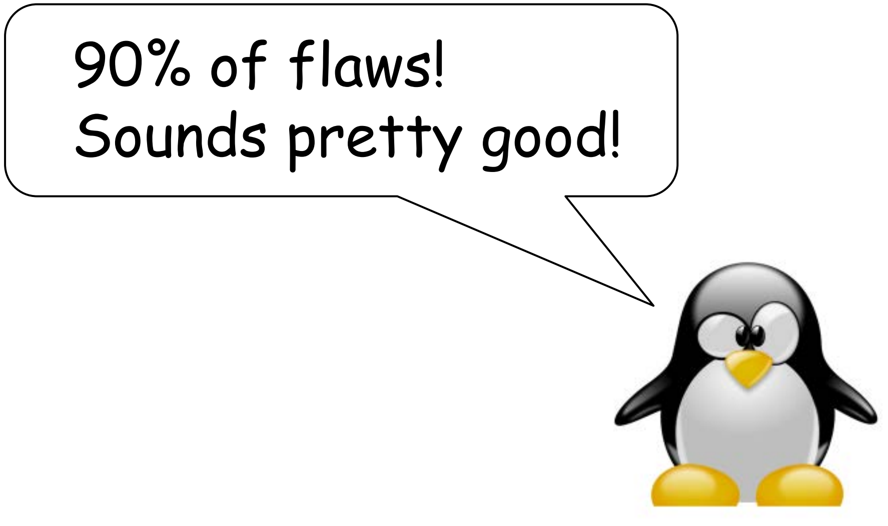
Traditional DoE	DoE for Software
• Continuous variable results	• Binary result (pass or fail)
• Few parameters	• Many parameters
• Interactions typically increase or decrease output	• Interactions affect path through program
• Statistical model requires balance	• Balance not needed



So how did testing interactions work in practice for software?

NIST
National Institute of Standards and Technology

- Pairwise, 2-factor, testing 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.

Maybe two factors are not enough?

- NIST studied software failures in 15 years of FDA medical device recall data
- What causes software failures?
  - logic errors? calculation errors?
- What testing would have detected errors?

Also found **interaction faults**: e.g., failure occurs if

`pressure < 10 && volume > 300` (2 factors)

Example from FDA failure analysis: Failure when "altitude adjustment set on 0 meters and total flow volume set at delivery rate of less than 2.2 liters per minute."

What does an interaction fault look like?

How does an interaction fault manifest itself in code?

Example: altitude_adj == 0 && volume < 2.2 (2-way interaction)

```
if (altitude_adj == 0 ) {
    // do something
    if (volume < 2.2)  { faulty code!  BOOM! }
    else { good code, no problem}
} else {
    // do something else
}
```

A test with altitude_adj == 0 and volume = 1 would find this

Again, ~ 90% of the FDA failures were 2-way or 1-way

… but some involved more than 2 factors

How are interaction faults distributed?

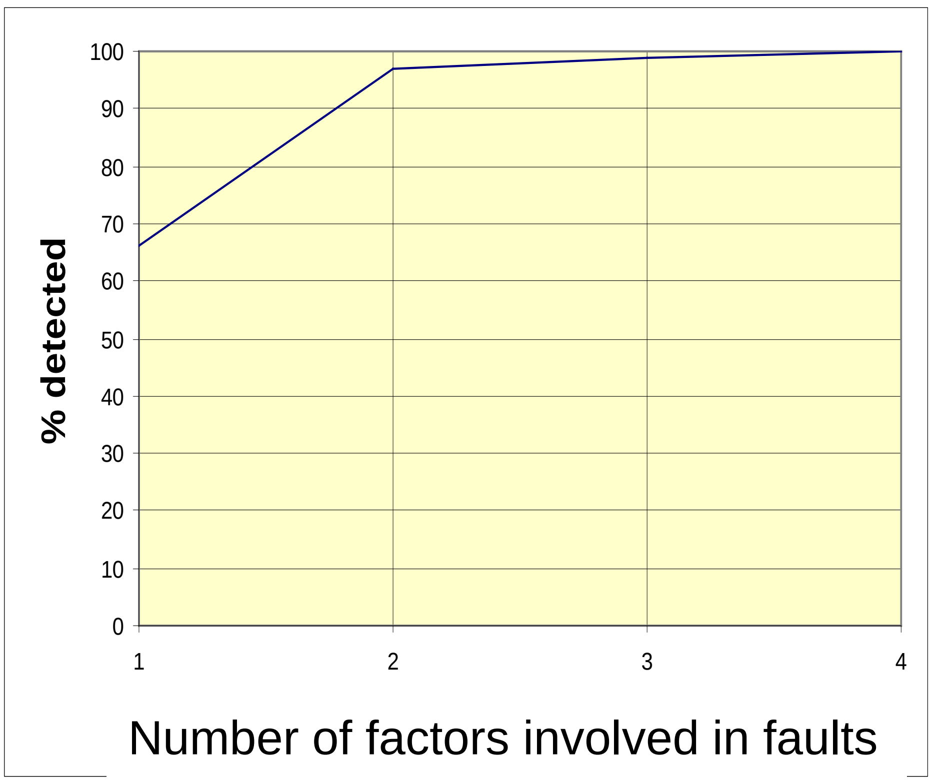
- Interactions e.g., failure occurs if

  pressure < 10 (1-way interaction)

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

  pressure < 10 & volume > 300 & velocity = 5 (3-way interaction)

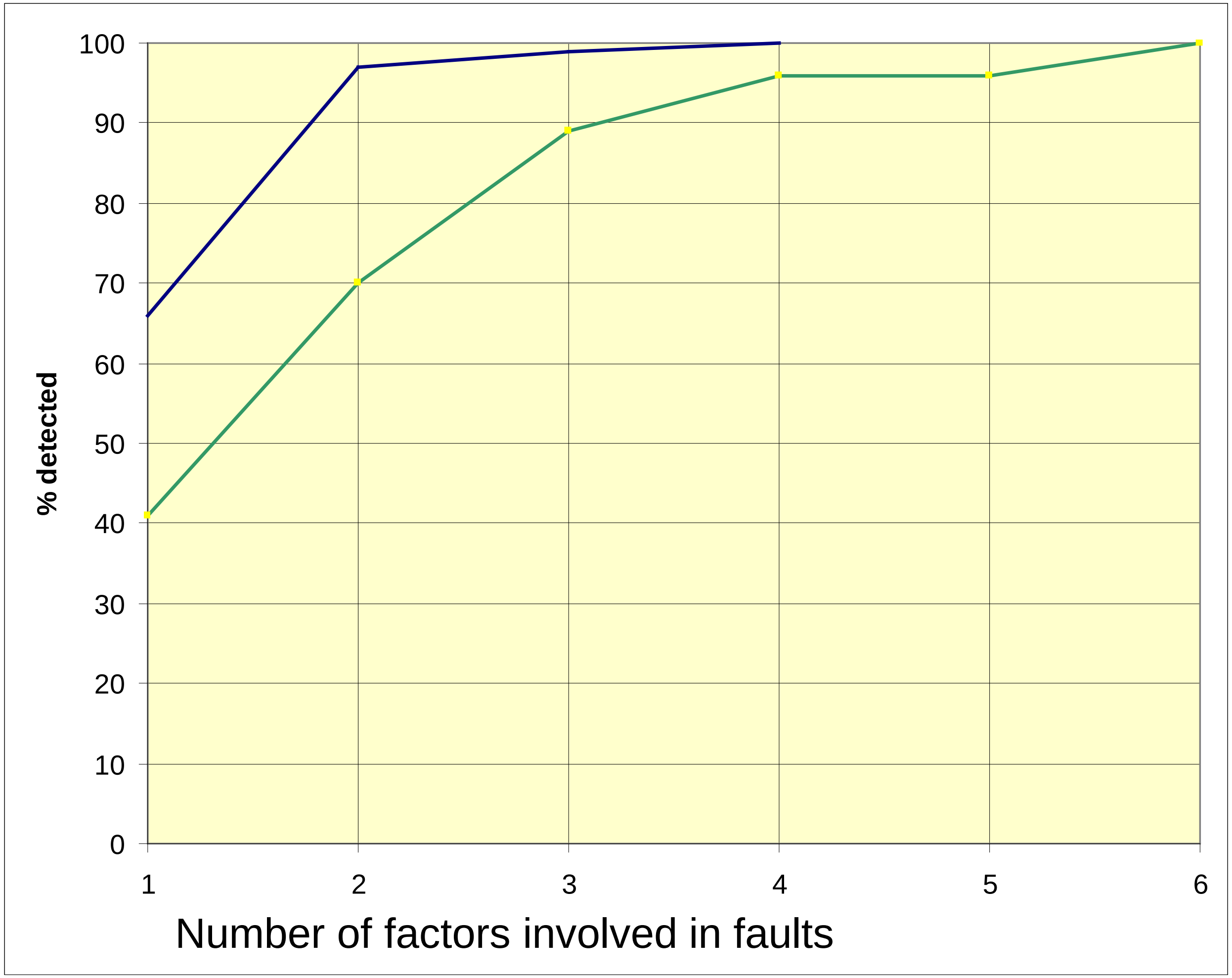
- Surprisingly, no one had looked at interactions beyond 2-way before
- The most complex medical device failure reported required 4-way interaction to trigger.



National Institute of Standards and Technology

What about other applications?

Server (green)

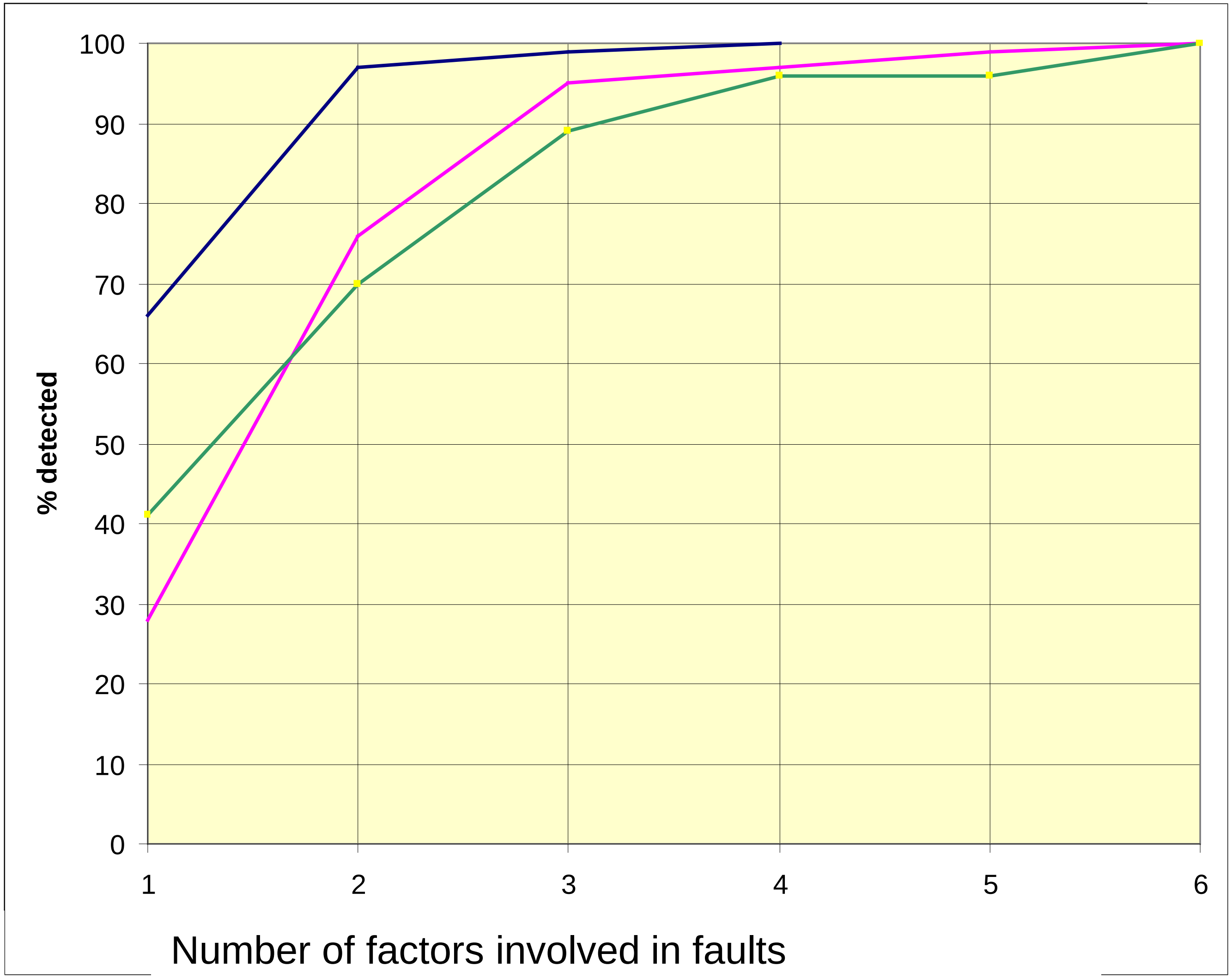


These faults more complex than medical device software!!

Why?

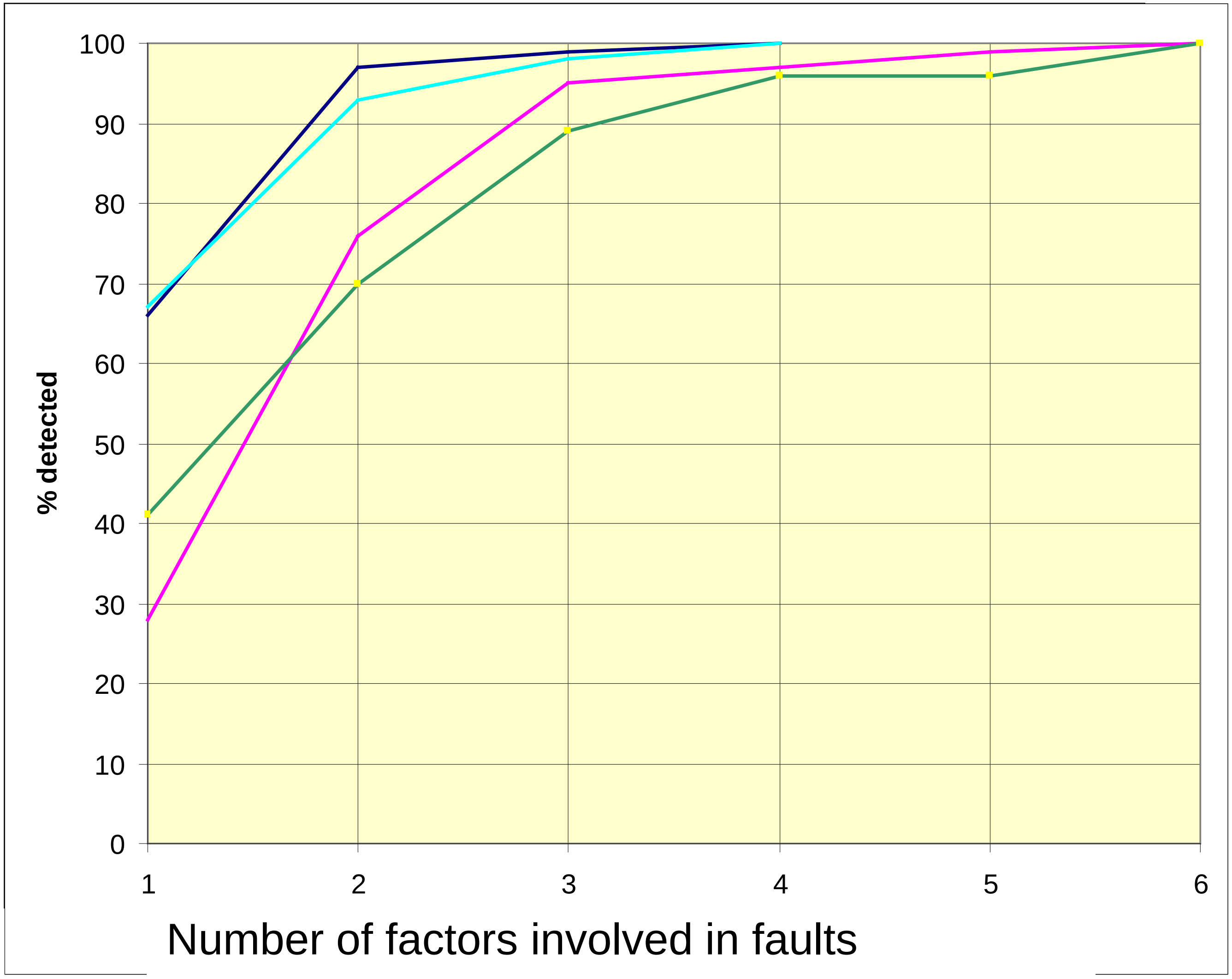
Others?

Browser (magenta)



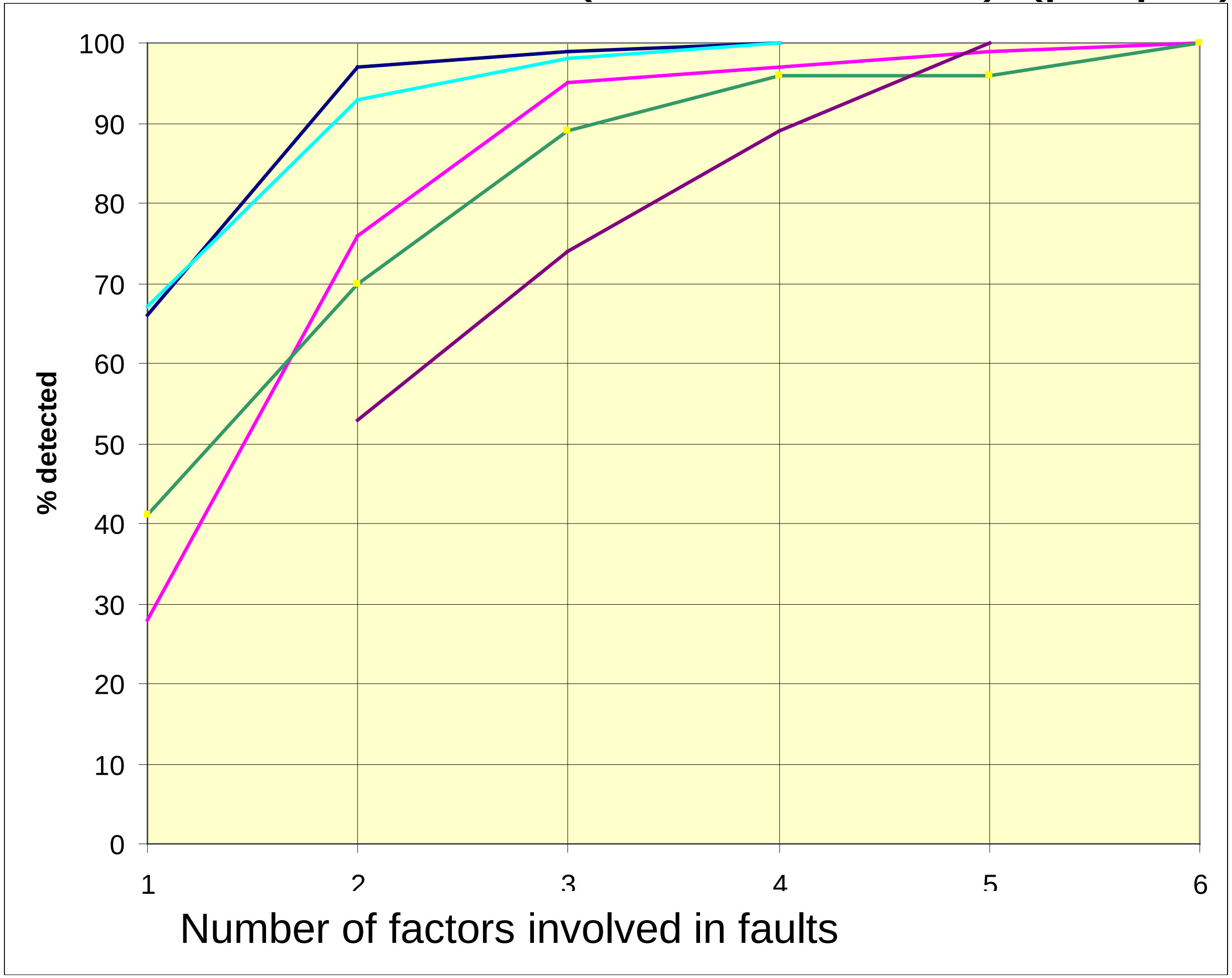
Still more?

NASA Goddard distributed database (light blue)



Even more?

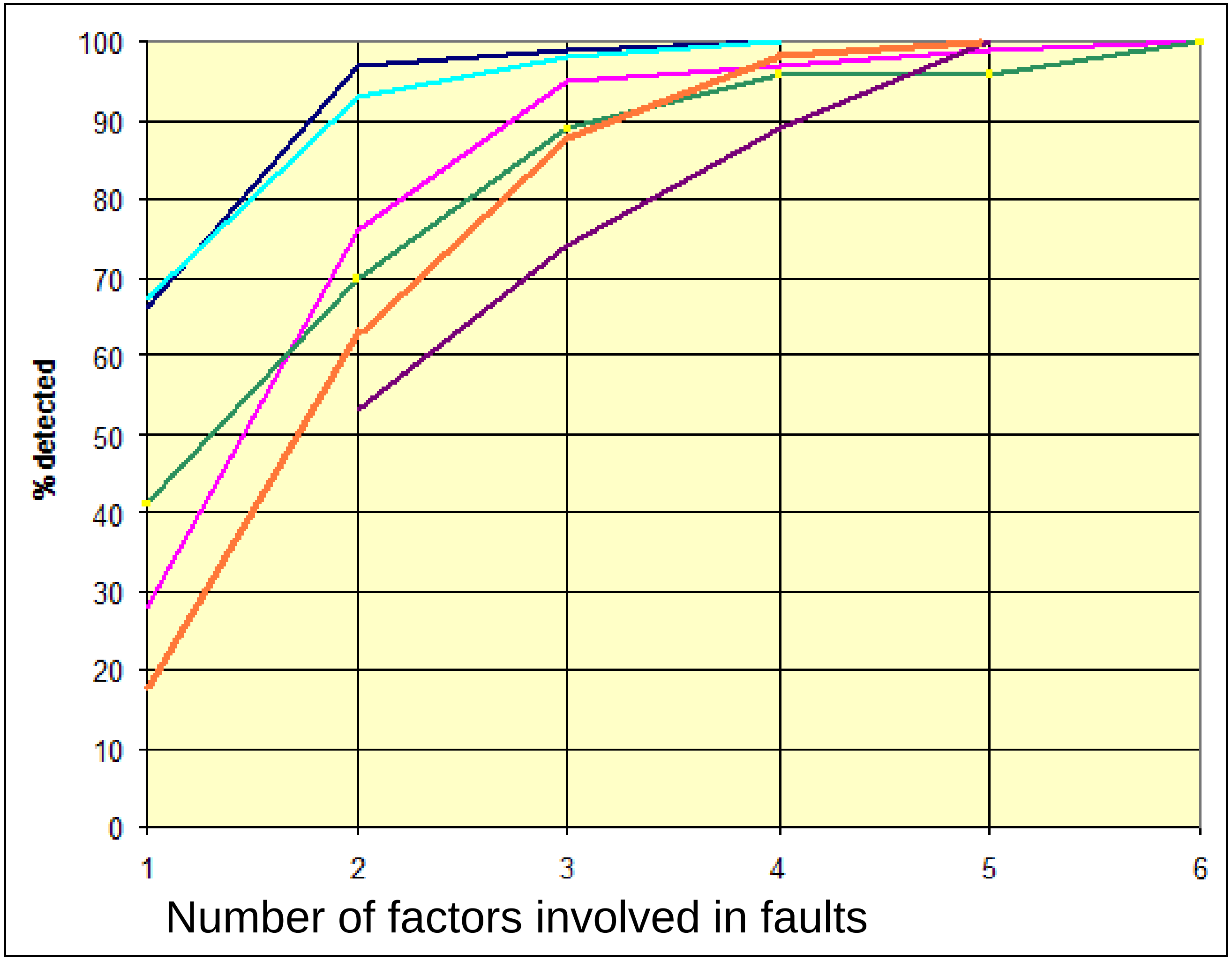
FAA Traffic Collision Avoidance System module (seeded errors) (purple)



National Institute of Standards and Technology

Finally

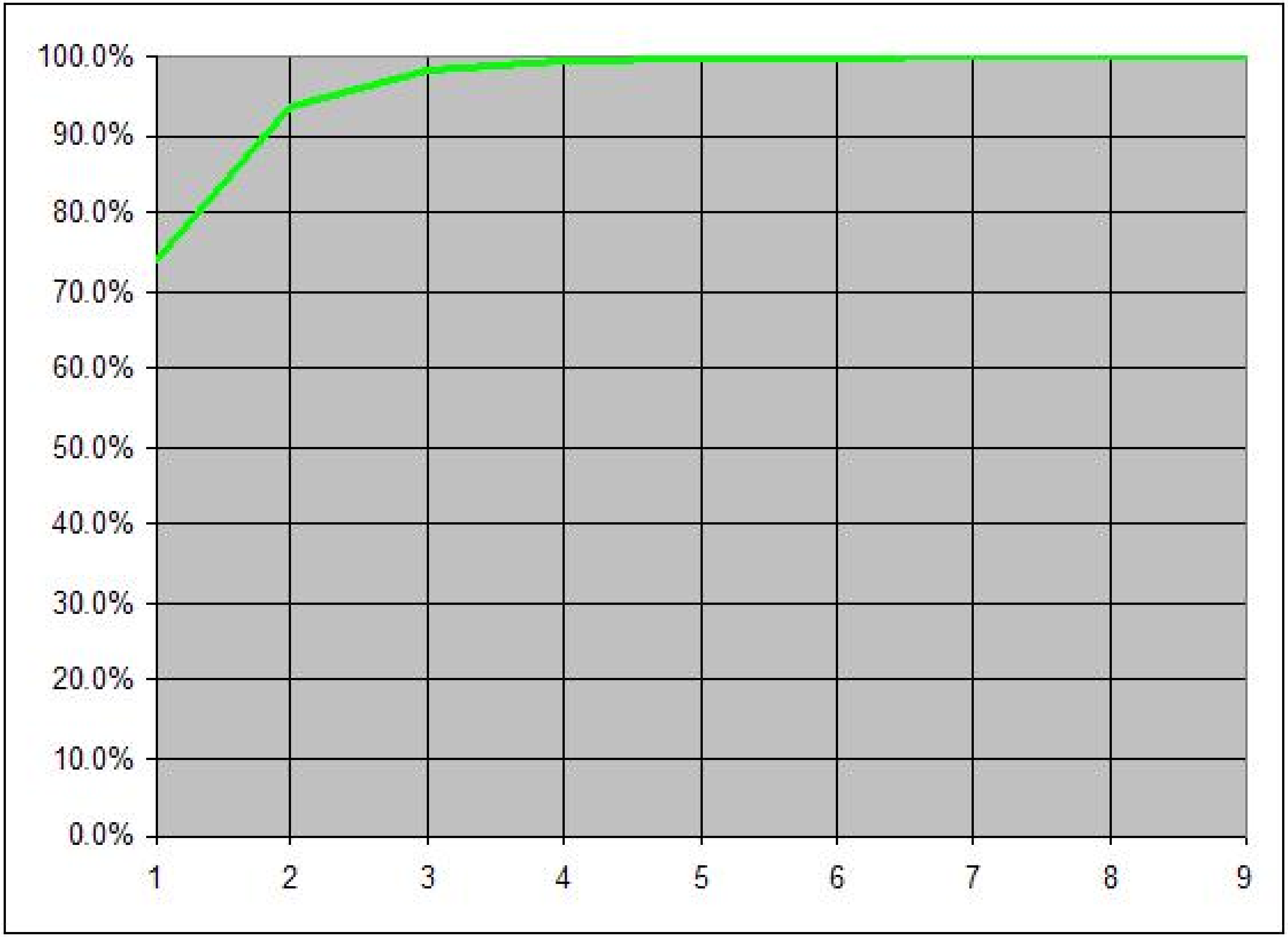
Network security (Bell, 2006) (orange)



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

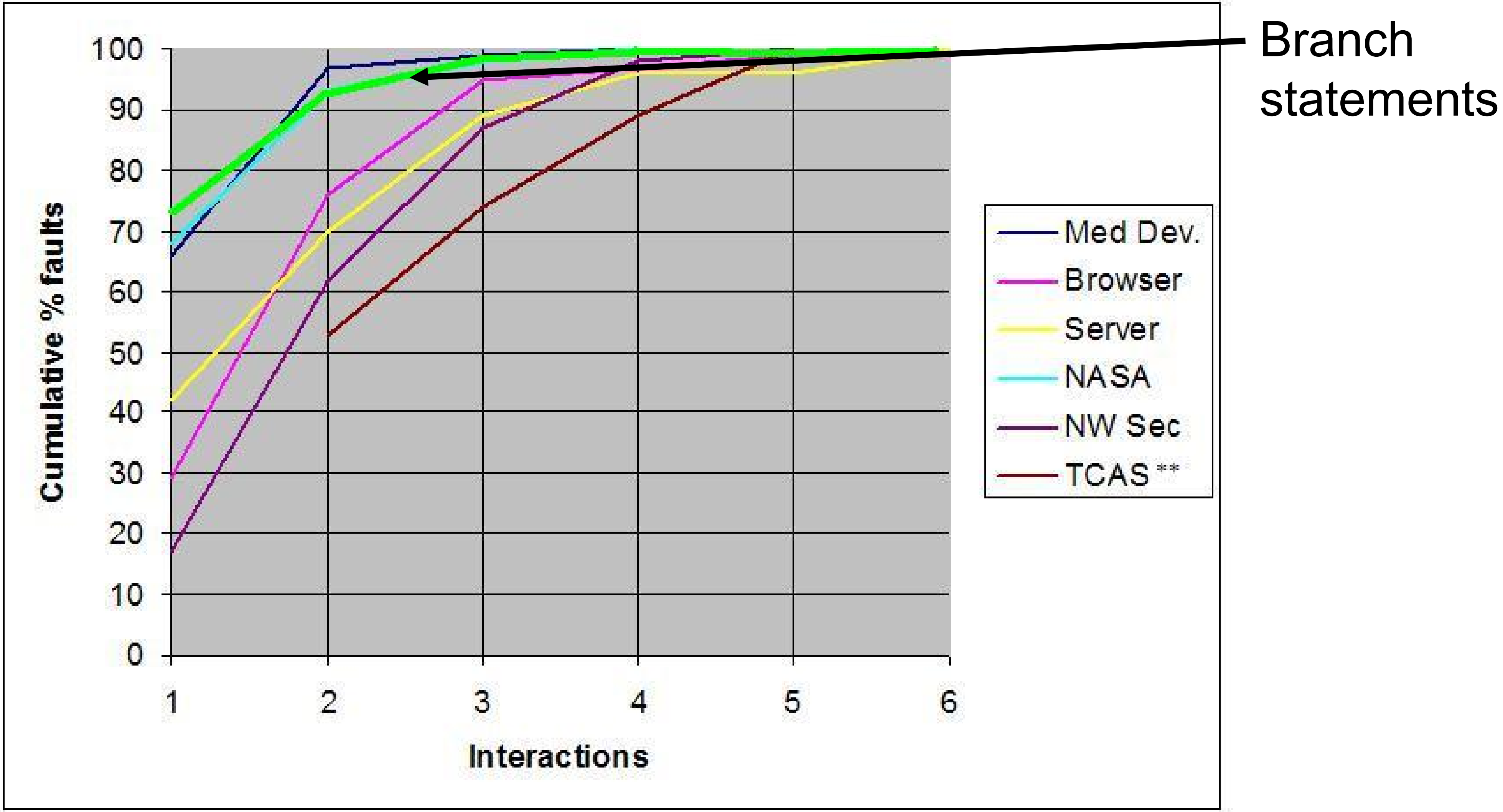
National Institute of Standards and Technology

What causes this distribution?

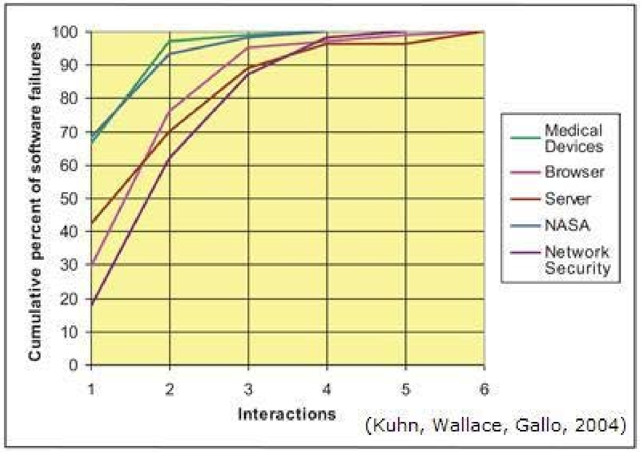


One clue: branches in avionics software.
7,685 expressions from *if* and *while* statements

Comparing with Failure Data



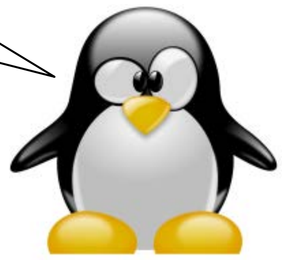
Interaction Rule



- Refers to how many parameters are involved in faults:
  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
- Two-factor testing not enough
- Reasonable evidence that maximum interaction strength for fault triggering is relatively small

How does it help me to know this?

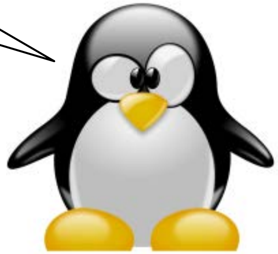


How does this knowledge help?

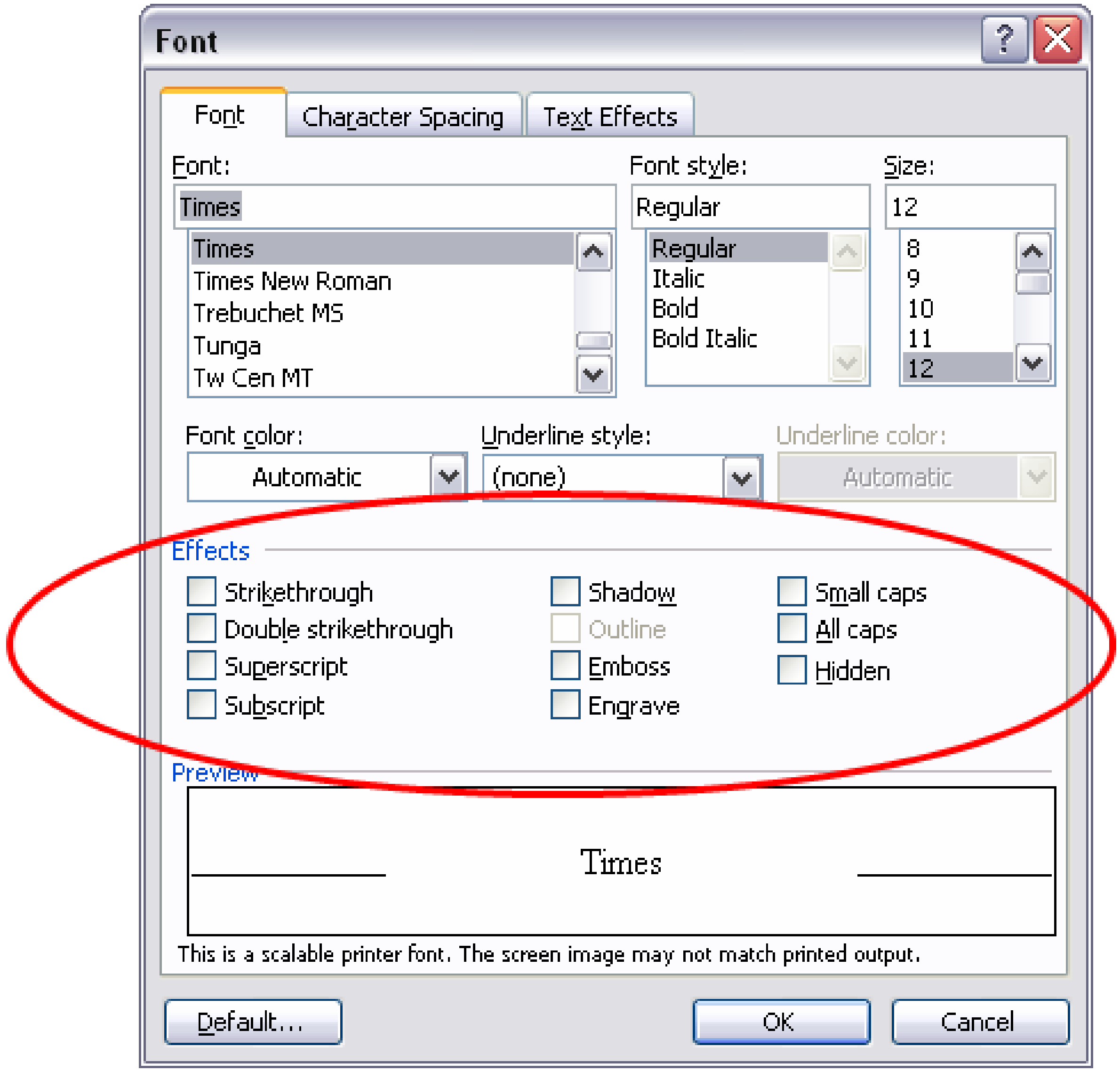
If all faults are triggered by the interaction of *t* or fewer variables, then testing all *t*-way combinations can provide strong assurance.

- New algorithms make it practical to test these combinations
- (however: value propagation issues, equivalence partitioning, timing issues, more complex interactions,…)

Still no silver bullet. Rats!



Let’s see how to use this knowledge in testing.   A simple example:



How Many Tests Would It Take?

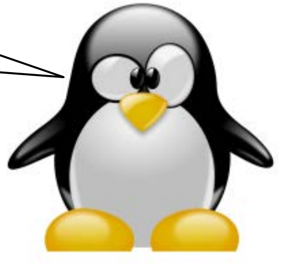
- There are 10 effects, each can be on or off
- All combinations is 2^{10} = 1,024 tests
- What if our budget is too limited for these tests?
- Instead, let's look at all 3-way interactions …

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

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



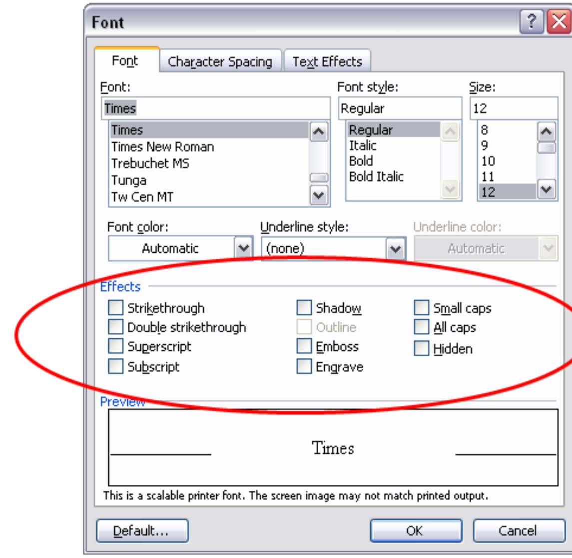
A covering array

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
- Difficult mathematically but good algorithms now

NIST National Institute of Standards and Technology

Design of Experiments for Software Testing

Not orthogonal arrays, but Covering arrays: Fixed-value CA(N, v^k, t) has four parameters N, k, v, t : It is a matrix covers every t-way combination at least once

Early developments by Tatsumi, Sherwood

Key differences

orthogonal arrays:

- Combinations occur same number of times
- Only exist for a certain configurations

covering arrays:

- Combinations occur at least once
- Always possible to find for any configuration
- Always smaller than orthogonal array (or same)

A larger example

Suppose we have a system with 34 on-off switches. Software must produce the right response for any combination of switch settings:

NIST
National Institute of Standards and Technology

How do we test this?

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

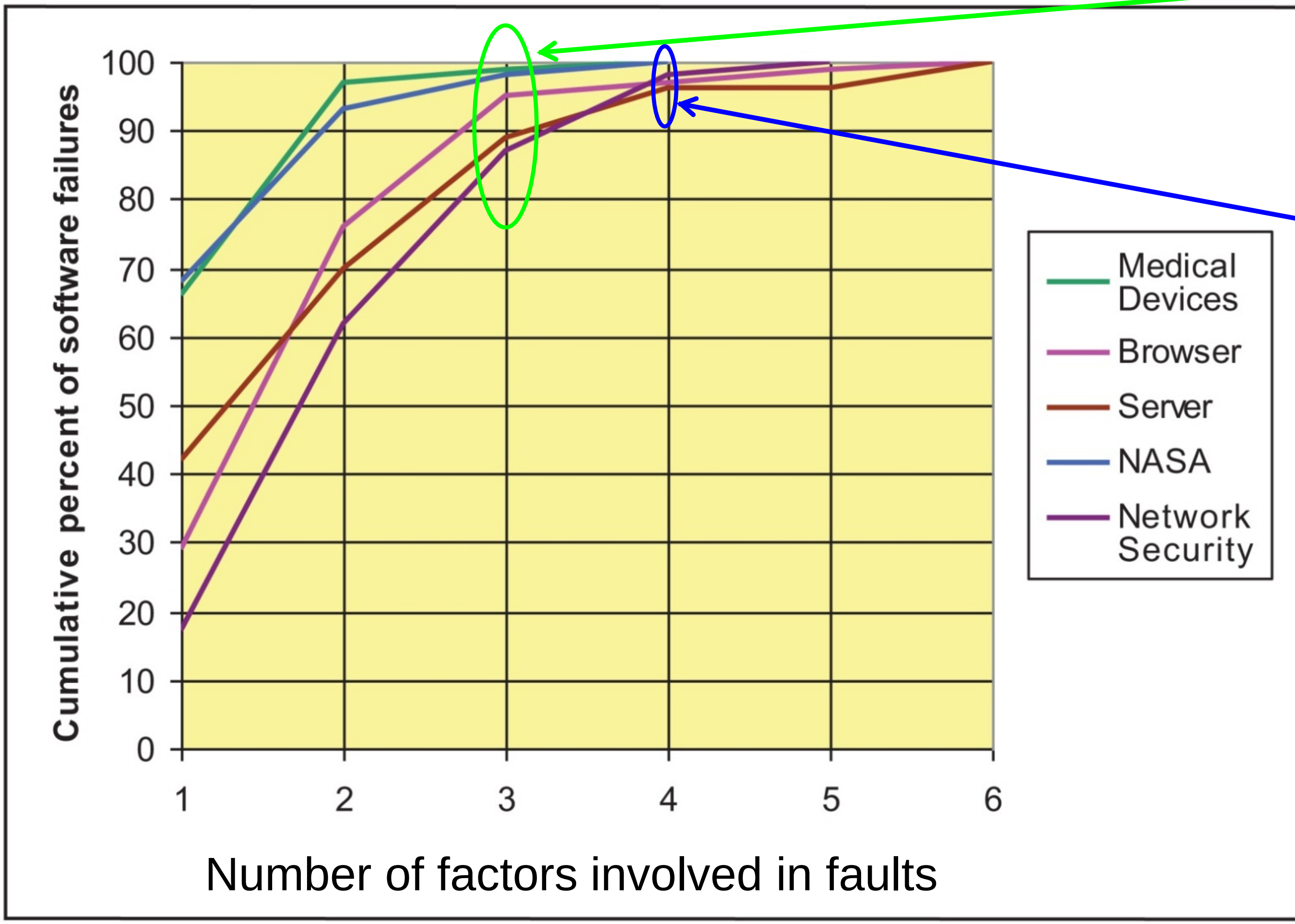
National Institute of Standards and Technology

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

National Institute of Standards and Technology

Do we have enough tests?



33 tests for this range of fault detection

85 tests for this range of fault detection

That’s way better than 17 billion!

We can also use it on configurations

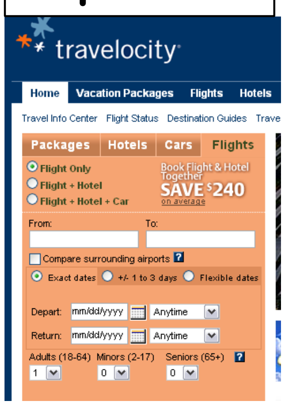
Use combinations here

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

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

- But something is wrong here … there is no Linux & IE configuration
- Covering array tools can avoid such invalid combinations

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:

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

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

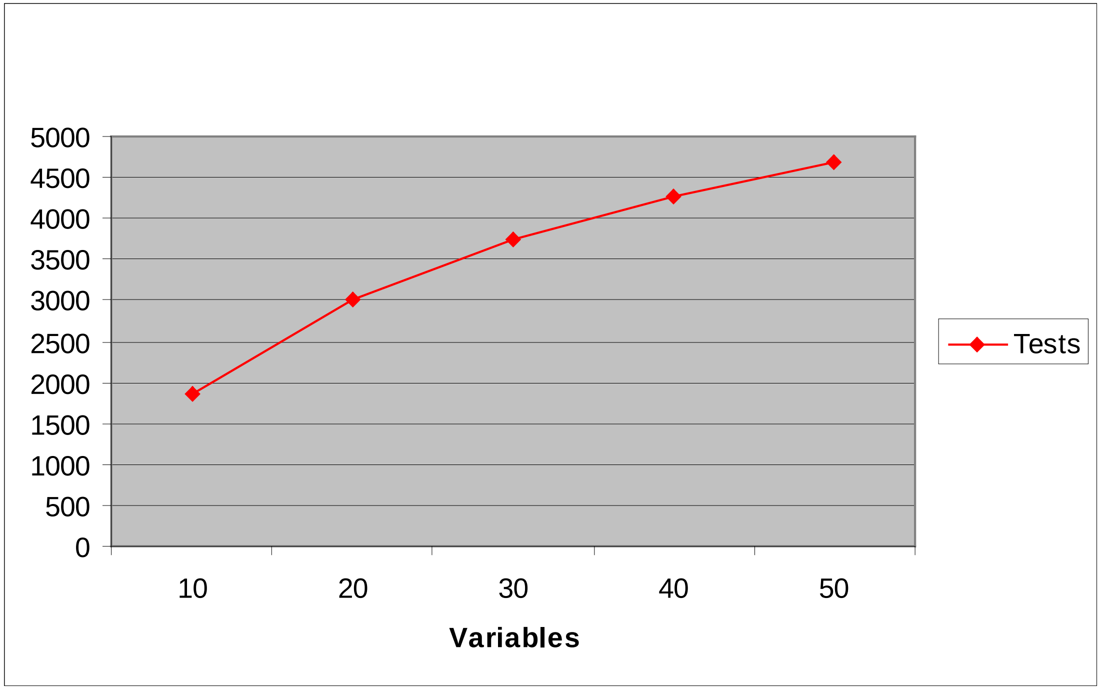
t	# Configs	% of Exhaustive
2	29	0.02
3	137	0.08
4	625	0.4
5	2532	1.5
6	9168	5.3

NIST
National Institute of Standards and Technology

How many tests are needed?

- 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*
  - But *logarithmically* with the number of parameters

- Example: suppose we want all 4-way combinations of *n* parameters, 5 values each:



How do we automate checking correctness of output?

- **Creating test data is the easy part!**
- How do we check that the code worked correctly on the test input?
  - **Crash testing** server or other code to ensure it does not crash for any test input (like ‘fuzz testing’)
    - Easy but limited value
  - **Built-in self test with embedded assertions** – incorporate assertions in code to check critical states at different points in the code, or print out important values during execution
  - **Full scale model-checking** using mathematical model of system and model checker to generate expected results for each input - expensive but tractable

NIST
National Institute of Standards and Technology

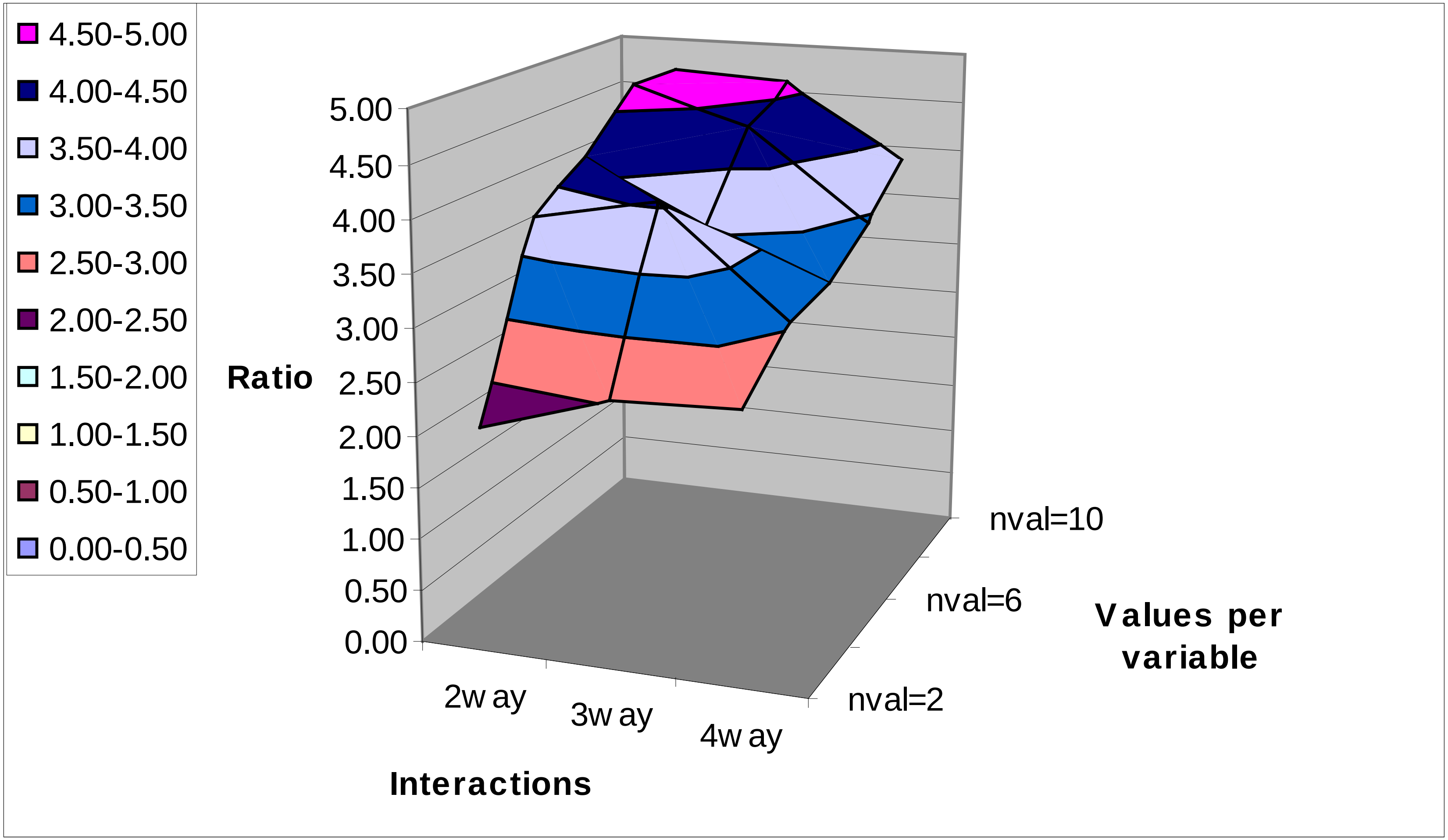
Crash Testing

- Like “fuzz testing” - send packets or other input to application, watch for crashes
- Unlike fuzz testing, input is non-random; cover all t-way combinations
- May be more efficient - random input generation requires several times as many tests to cover the t-way combinations in a covering array

Limited utility, but can detect high-risk problems such as:

- buffer overflows
- server crashes

Ratio of Random/Combinatorial Test Set Required to Provide t-way Coverage



Embedded Assertions

Assertions check properties of expected result:

```
ensures balance  == \old(balance) - amount
 &&  \result == balance;
```

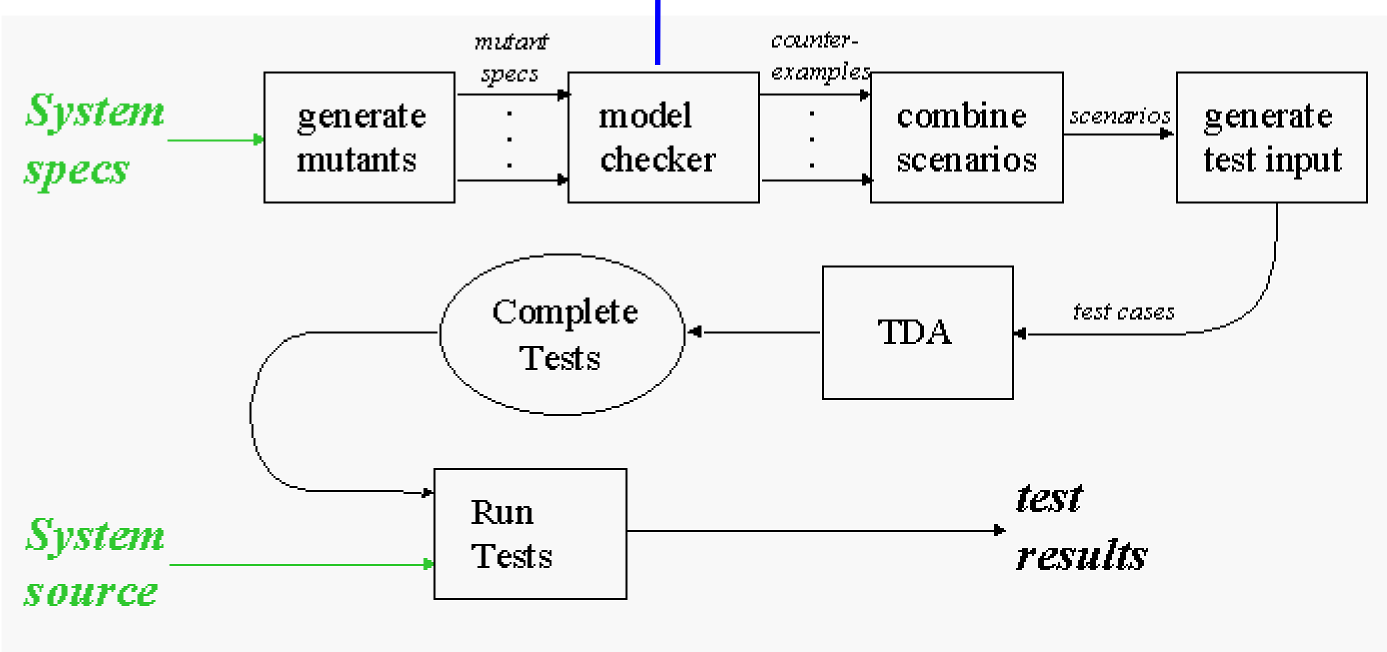
•Reasonable assurance that code works correctly across the range of expected inputs

•May identify problems with handling unanticipated inputs

•Example: Smart card testing

- Used Java Modeling Language (JML) assertions
- Detected 80% to 90% of flaws

Using model checking to produce tests



- Model-checker test production: if assertion is not true, then a counterexample is generated.

- This can be converted to a test case.

Black & Ammann, 1999

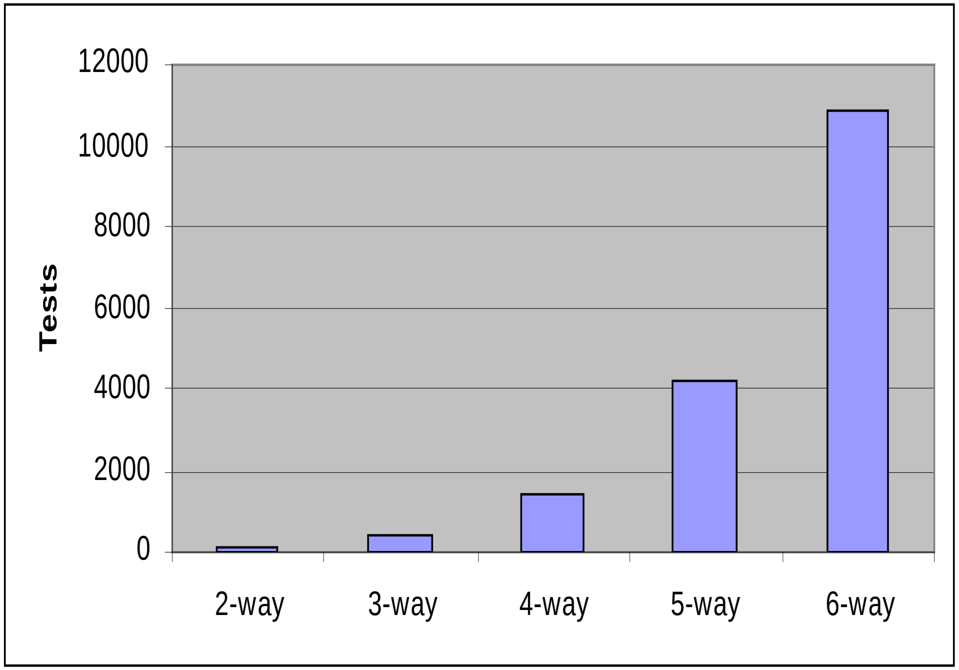
Testing inputs

- 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

NIST National Institute of Standards and Technology

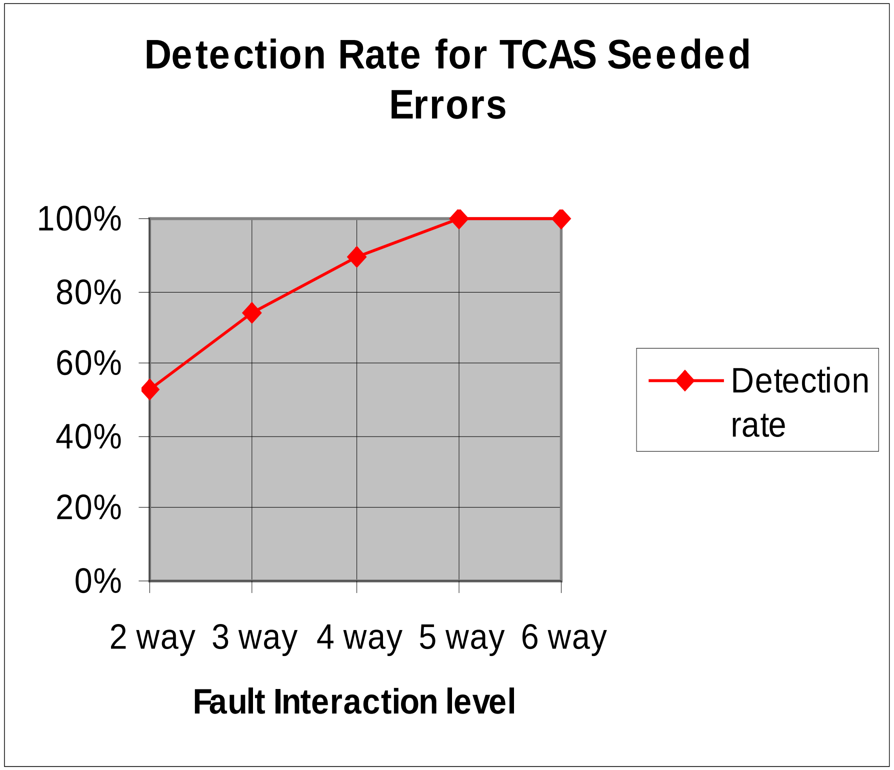
Tests generated

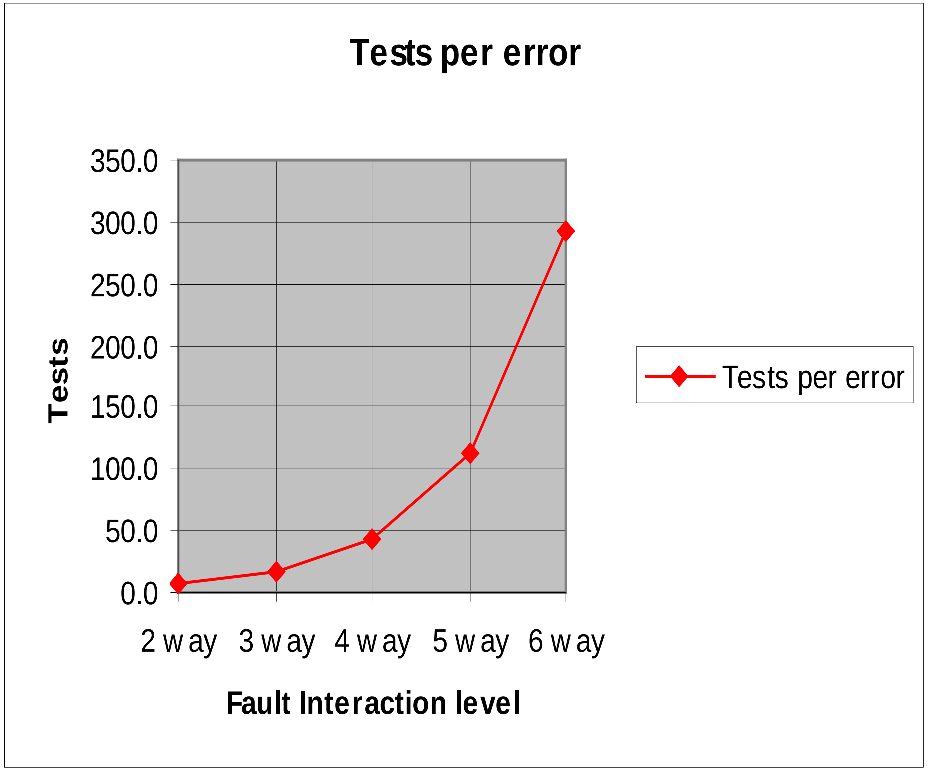
t	Test cases
2-way:	156
3-way:	461
4-way:	1,450
5-way:	4,309
6-way:	11,094



Results

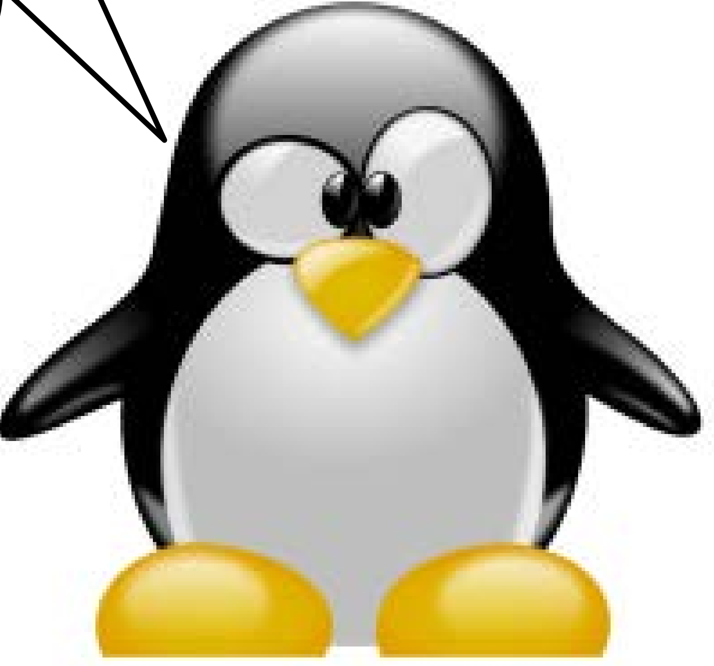
- Roughly consistent with data on large systems
- But errors harder to detect than real-world examples





Bottom line for model checking based combinatorial testing:
Requires more technical skill but can be highly effective

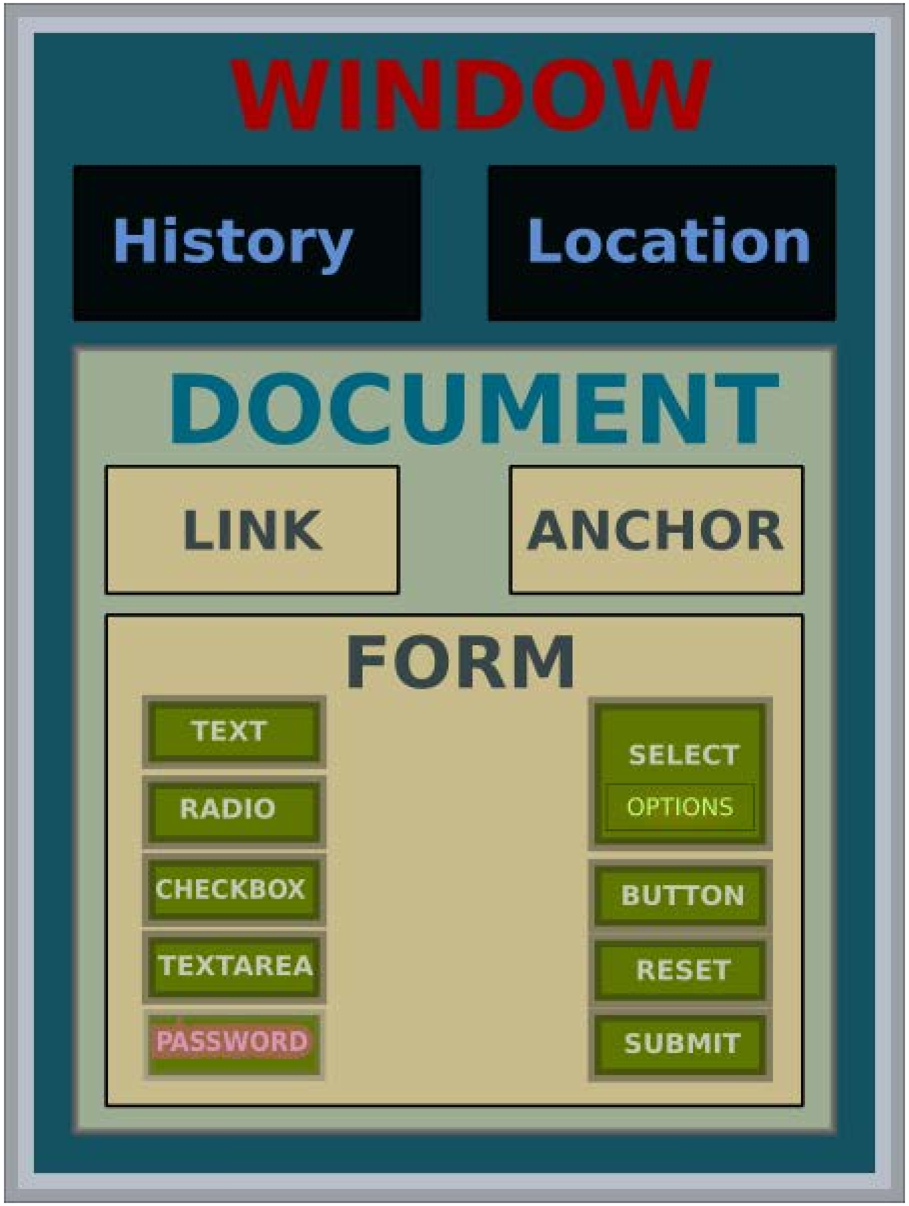
Is this stuff useful in the real world ??



What good is combinatorial testing?

- 2.5 year Lockheed Martin study, 8 pilot projects in aerospace software
- Results: "Lockheed Martin's initial estimate is that this method supported by the technology can save up to 20% of test planning/design costs if done early on a program while increasing test coverage by 20% to 50%."

Example 1: Document Object Model Events



- DOM is a World Wide Web Consortium standard for representing and interacting with browser objects
- NIST developed conformance tests for DOM
- Tests covered all possible combinations of discretized values, >36,000 tests

- Question: can we use the Interaction Rule to increase test effectiveness the way we claim?

NIST National Institute of Standards and Technology

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

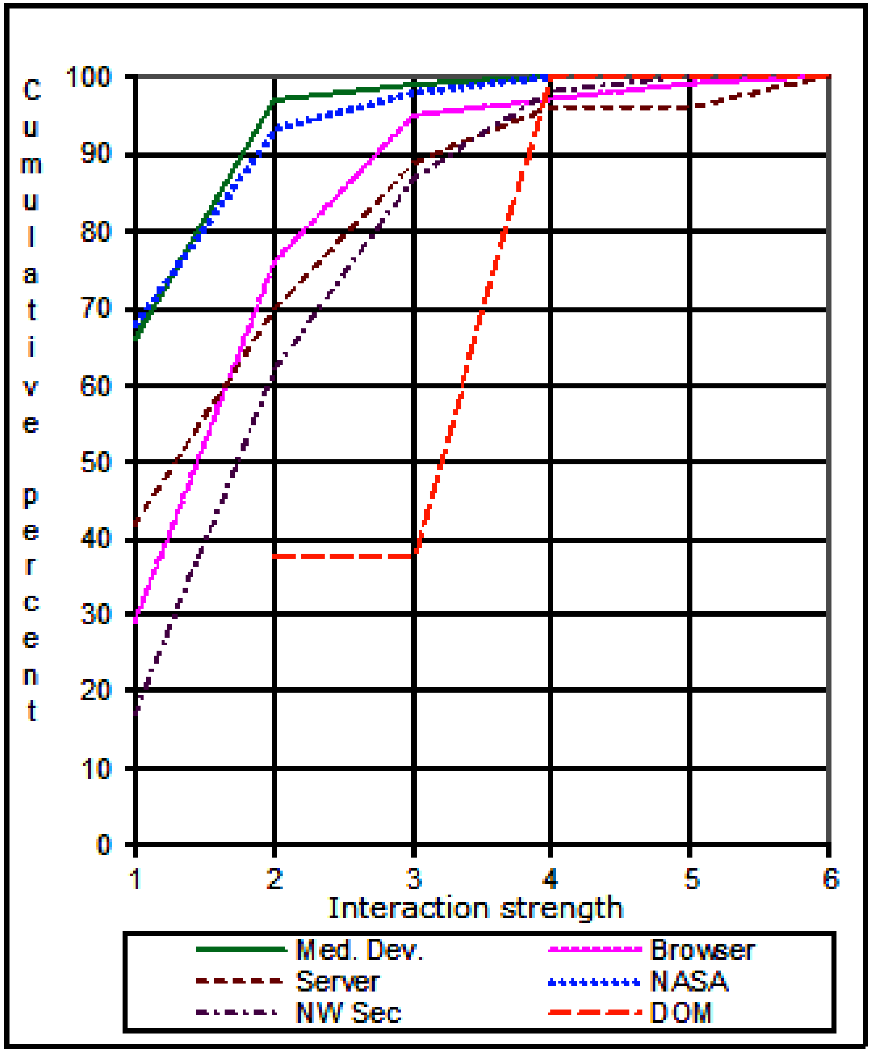
National Institute of Standards and Technology

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



National Institute of Standards and Technology

Example 2: Laptop application testing

Connection Sequences

1	Boot	P-1 (USB-RIGHT)	P-2 (USB-BACK)	P-3 (USB-LEFT)	P-4	P-5	App	Scan
2	Boot	App	Scan	P-5	P-4	P-3 (USB-RIGHT)	P-2 (USB-BACK)	P-1 (USB-LEFT)
3	Boot	P-3 (USB-RIGHT)	P-2 (USB-LEFT)	P-1 (USB-BACK)	App	Scan	P-5	P-4
	etc...							

Event Sequence Testing

- Suppose we want to see if a system works correctly regardless of the order of events. How can this be done efficiently?
- Failure reports often say something like: 'failure occurred when A started if B is not already connected'.
- Can we produce compact tests such that all t-way sequences covered (possibly with interleaving events)?

Event	Description
a	connect video
b	connect range finder
c	connect satellite link
d	connect navigation
e	start comm link
f	boot system

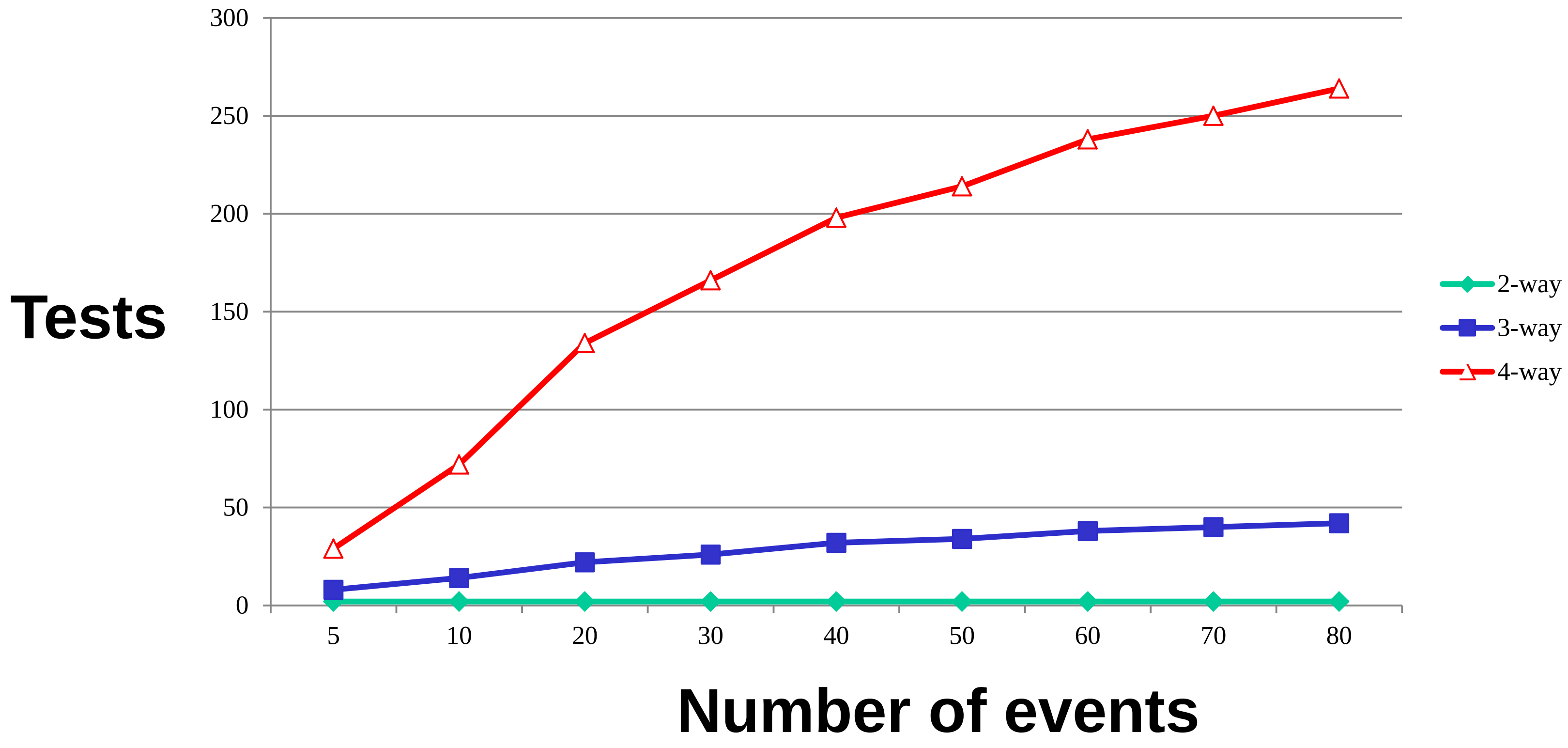
Sequence Covering Array

- With 6 events, all sequences = 6! = 720 tests
- Only 10 tests needed for all 3-way sequences, results <u>even better for larger numbers of events</u>
- Example: .*c.*f.*b.* covered. Any such 3-way seq covered.

Test	Sequence					
1	a	b	c	d	e	f
2	f	e	d	c	b	a
3	d	e	f	a	b	c
4	c	b	a	f	e	d
5	b	f	a	d	c	e
6	e	c	d	a	f	b
7	a	e	f	c	b	d
8	d	b	c	f	e	a
9	c	e	a	d	b	f
10	f	b	d	a	e	c

Sequence Covering Array Properties

- 2-way sequences require only 2 tests
  (write events in any order, then reverse)
- For > 2-way, number of tests grows with log *n*, for *n* events
- Simple greedy algorithm produces compact test set
- Not previously described in CS literature (but pure math paper 1970s!)



Example 3: Existing Test Sets

- Will this method disrupt my test process?
- What if I already have a large set of tests? Does this approach add anything?

- NASA spacecraft software test set, approx 7,500 tests
- Does it already provide 2-way, 3-way, 4-way coverage?

Measuring Combinatorial Coverage

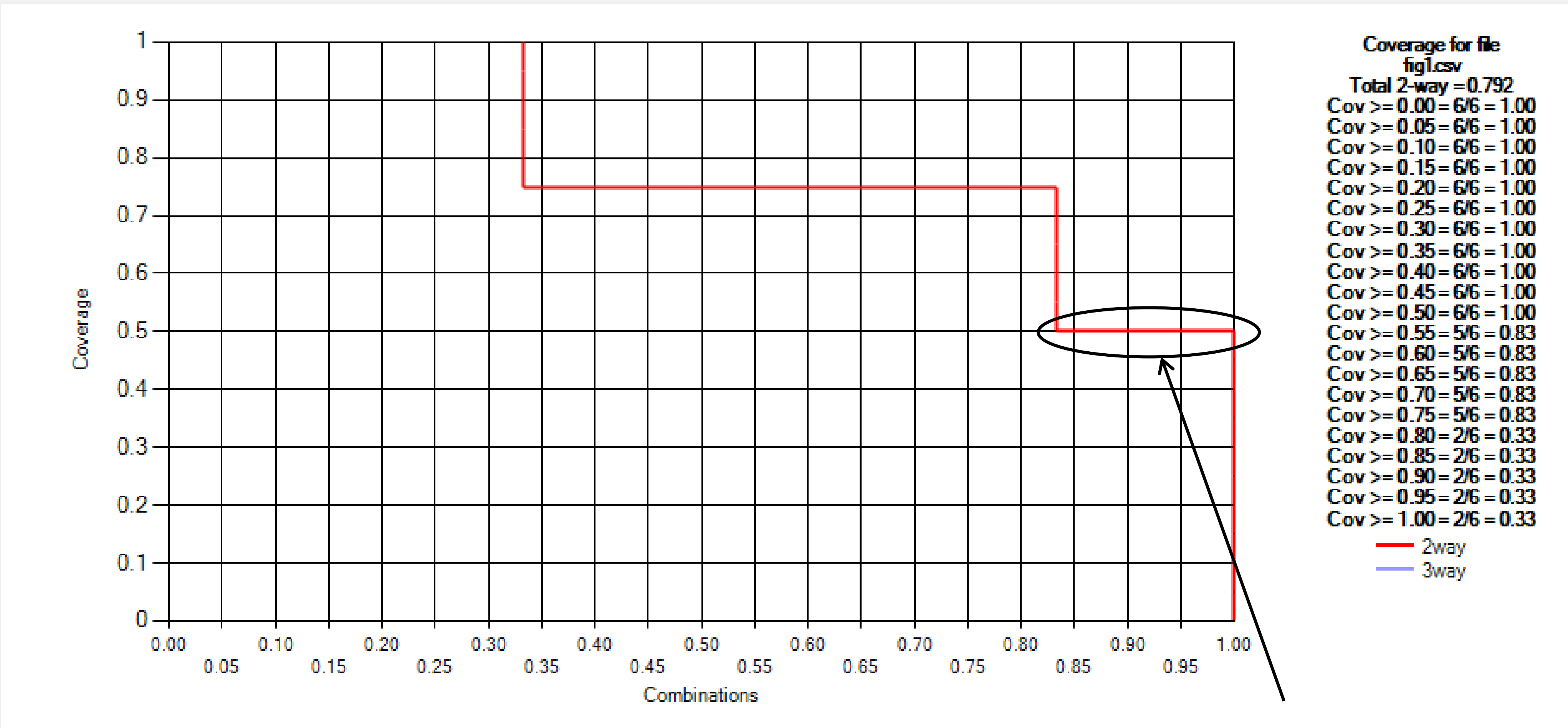
Tests	Variables			
	a	b	c	d
1	0	0	0	0
2	0	1	1	0
3	1	0	0	1
4	0	1	1	1

Variable pairs	Variable-value combinations covered	Coverage
ab	00, 01, 10	.75
ac	00, 01, 10	.75
ad	00, 01, 11	.75
bc	00, 11	.50
bd	00, 01, 10, 11	1.0
cd	00, 01, 10, 11	1.0

100% coverage of 33% of combinations
75% coverage of half of combinations
50% coverage of 16% of combinations

NIST
National Institute of Standards and Technology

Graphing Coverage Measurement

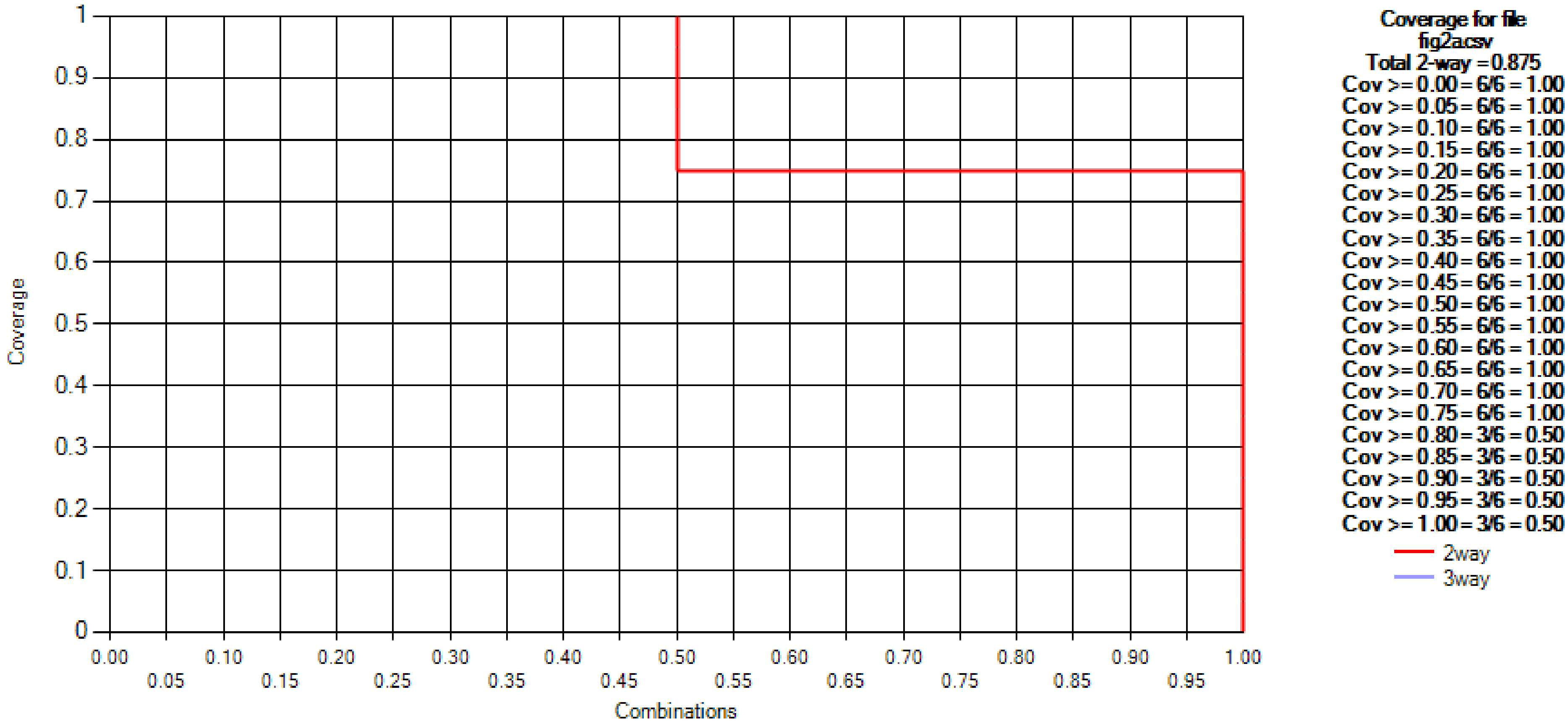


100% coverage of 33% of combinations
75% coverage of half of combinations
50% coverage of 16% of combinations

Bottom line:
All combinations
covered to at least 50%

National Institute of Standards and Technology

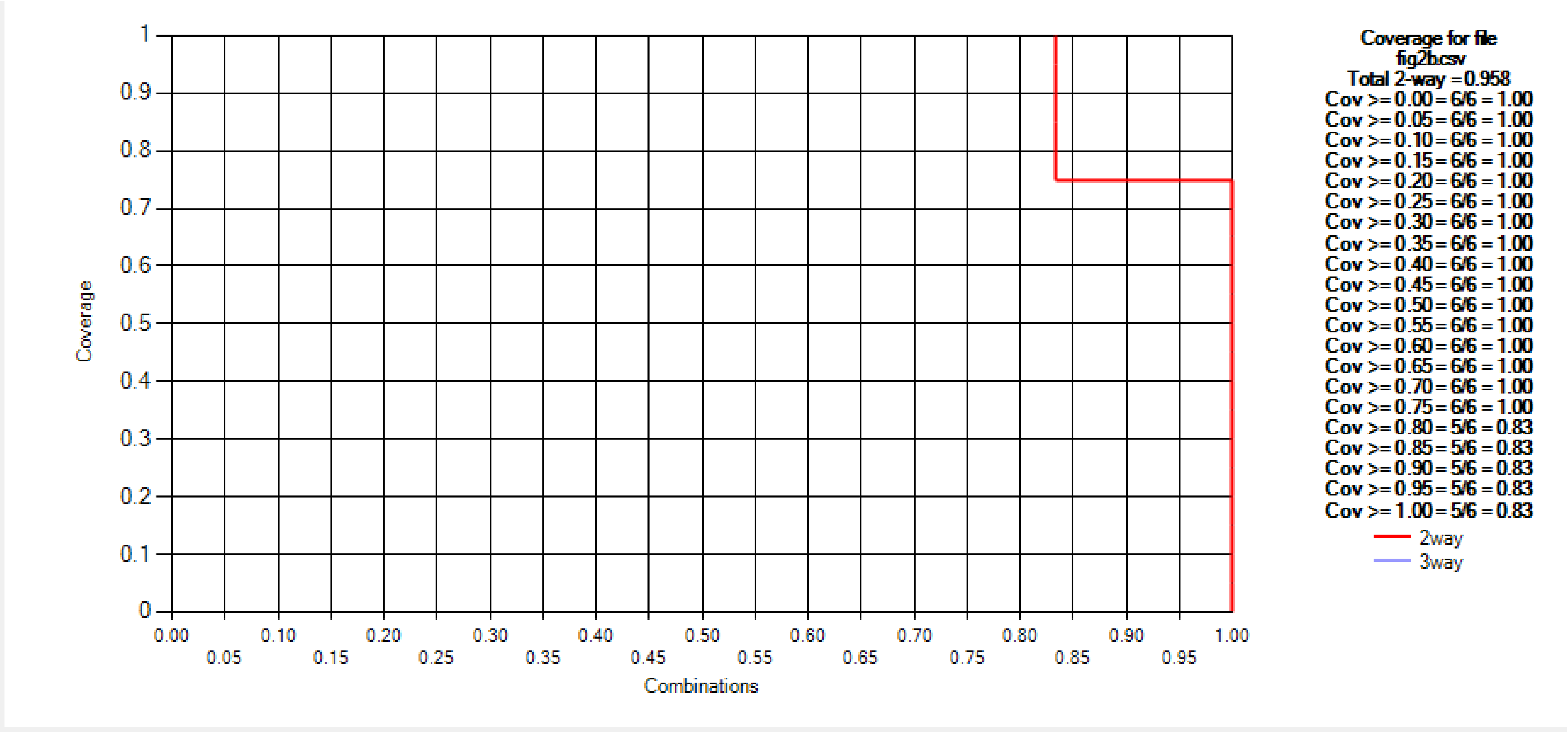
Adding a test



Coverage after adding test [1,1,0,1]

NIST National Institute of Standards and Technology

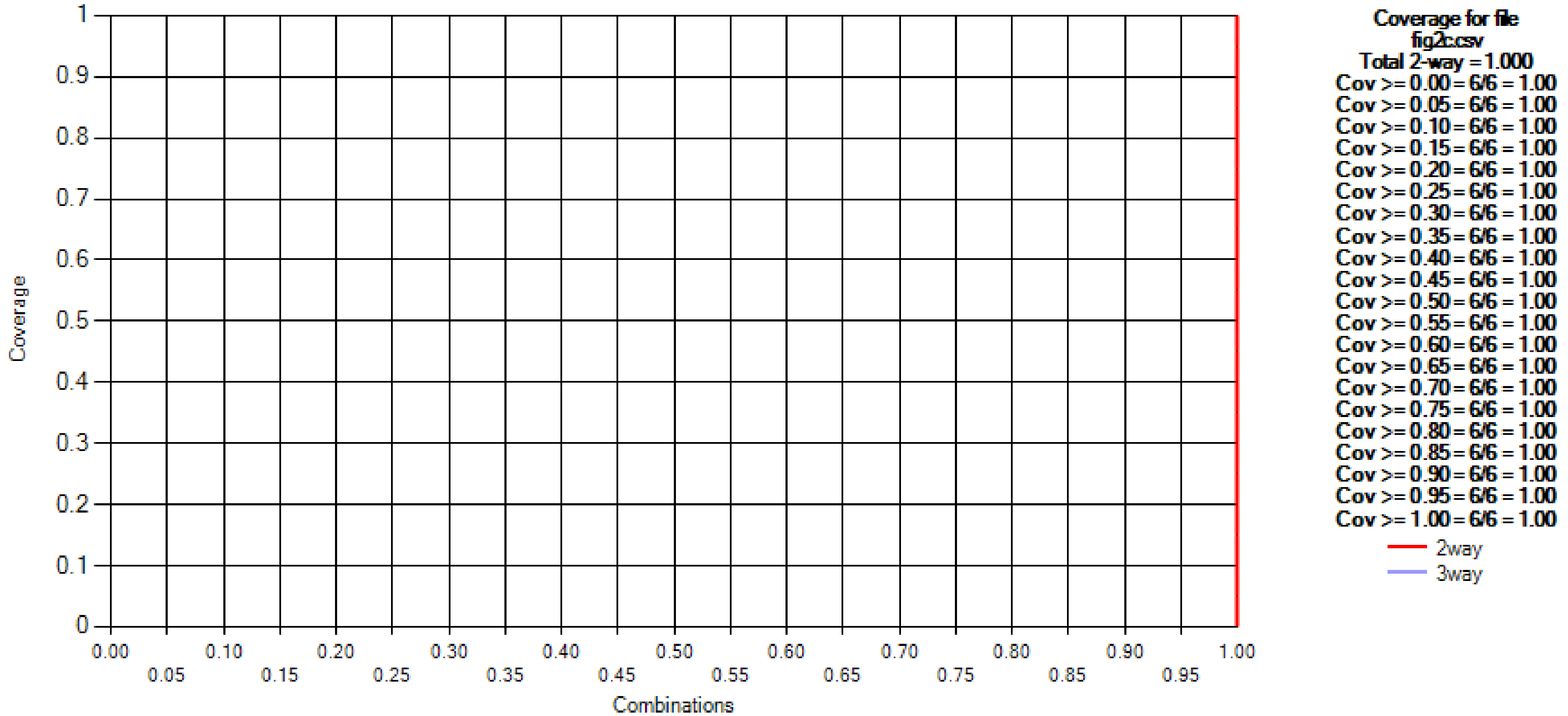
Adding another test



Coverage after adding test [1,0,1,1]

NIST National Institute of Standards and Technology

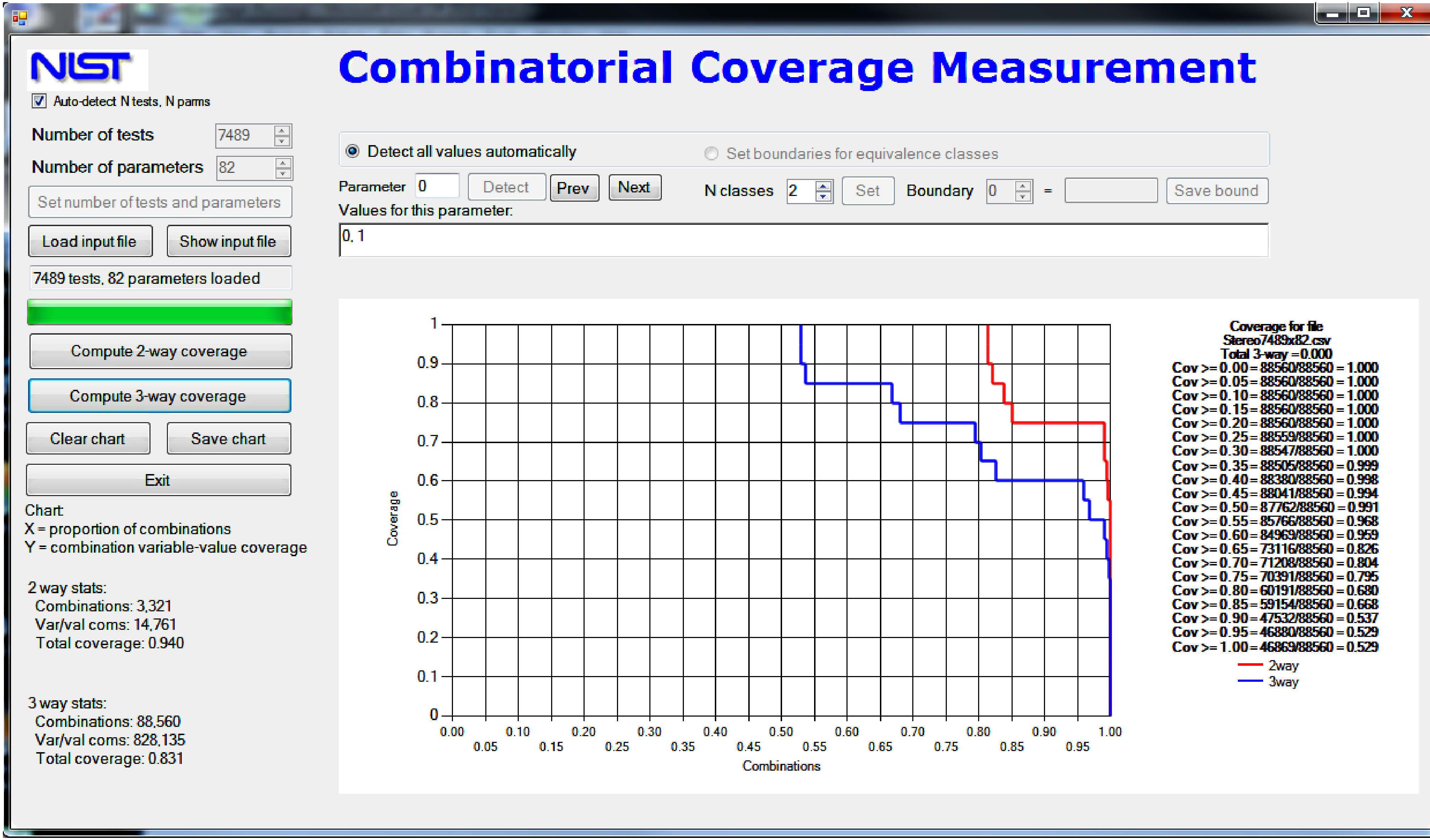
Additional test completes coverage



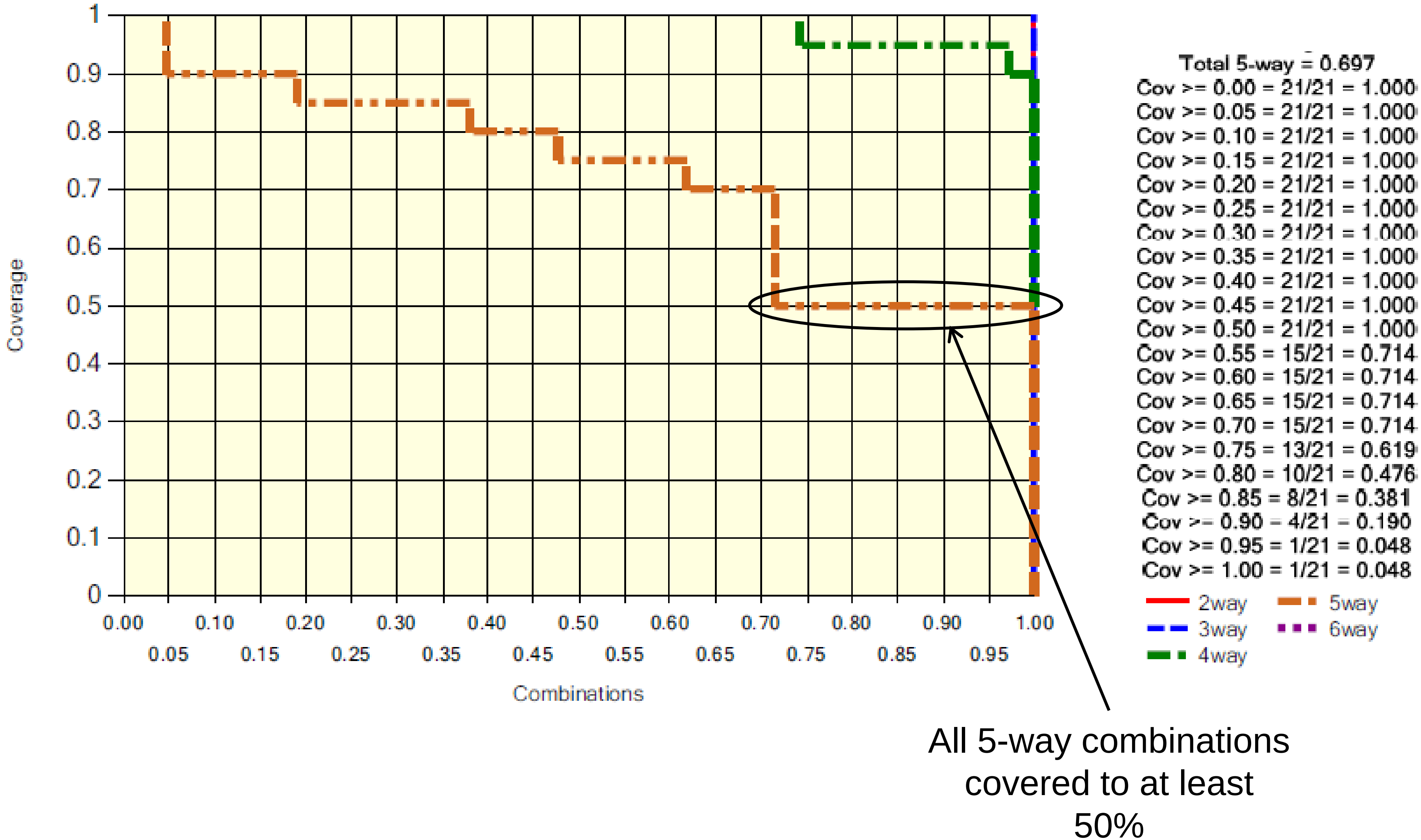
Coverage after adding test [1,0,1,0]
All combinations covered to 100% level,
so this is a covering array.

National Institute of Standards and Technology

Combinatorial Coverage Measurement



USAF test plan coverage



Where do we go next?

- "Internet of things" – testing problem enormous
  - Vast number of interacting components
  - Combinatorial testing is a natural fit
- Cyber-physical systems
  - Safety aspects
  - Another natural fit with combinatorial methods
- Test development environment
  - Define the data model – critical for testing
  - Classification tree tool project at CMU
- **Many research questions**

NIST
National Institute of
Standards and Technology

Algorithms

- Highly effective now, but room for improvements
- Algebraic
  - Very compact arrays, but some configurations can't be computed, problems with constraints
- Computational – (greedy, simulated annealing, hill climbing, genetic)
  - May produce more tests, but always possible, good constraint handling
- Post optimization and array reduction
  advances announced at this conference! (CAS, Nanjing)
- Current best known sizes:

http://www.public.asu.edu/~ccolbou/src/tabby/catable.html

Test Prioritization

Given a set of tests, what permutation finds faults fastest?

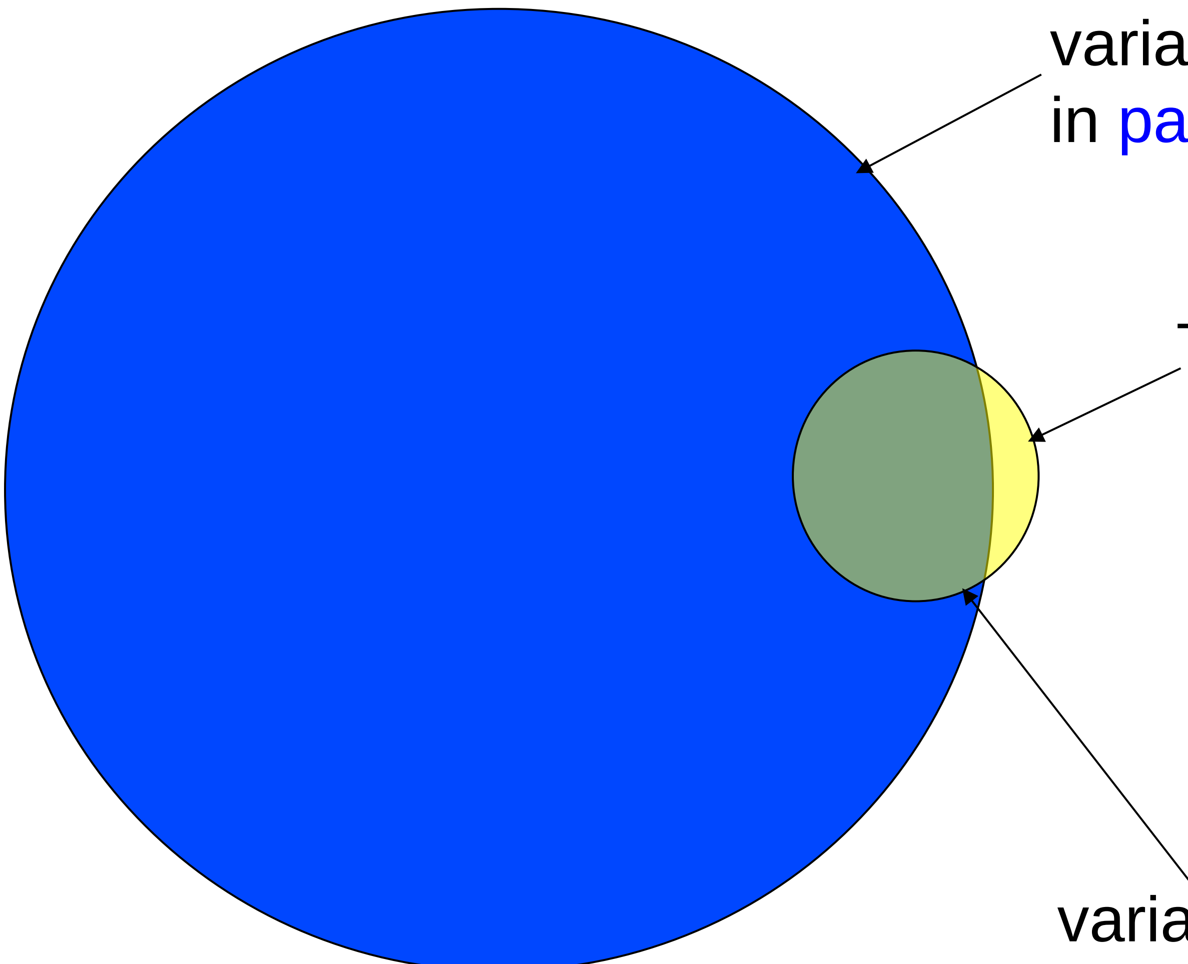
Tests can be ordered by various criteria:

- Combination coverage (equiv to greedy algo)
- Hamming distance
- Random
- Application-specific criterion

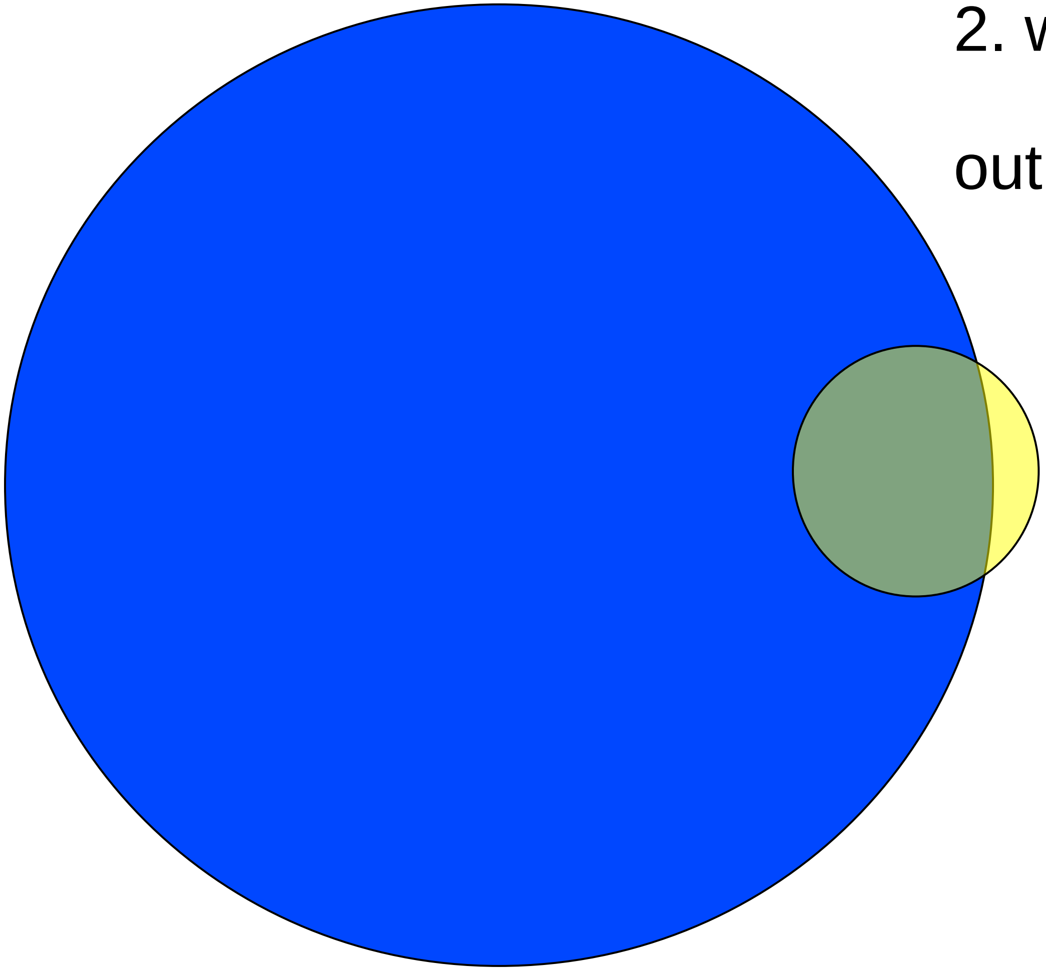
Especially important for GUI testing

Fault location

Given: a set of tests that the SUT fails, which combinations of variables/values triggered the failure?



Fault location – what's the problem?



If they're in failing set but not in passing set:

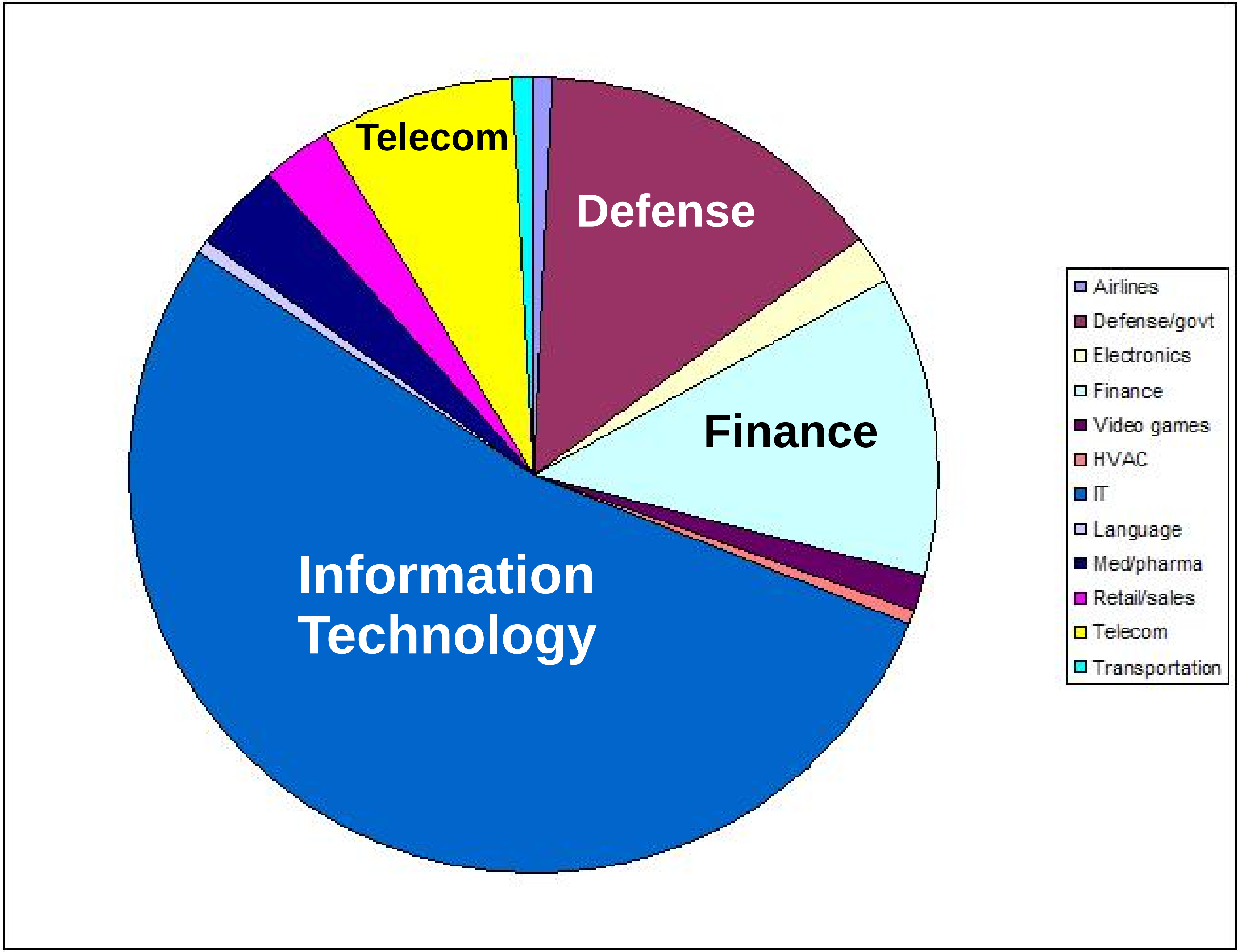
1. which ones triggered the failure?
2. which ones don't matter?

out of $v^t \binom{n}{t}$ combinations

Example:
30 variables, 5 values each
= 445,331,250
5-way combinations

142,506 combinations
in each test

Users and Tech Transfer



Lessons Learned and Needs

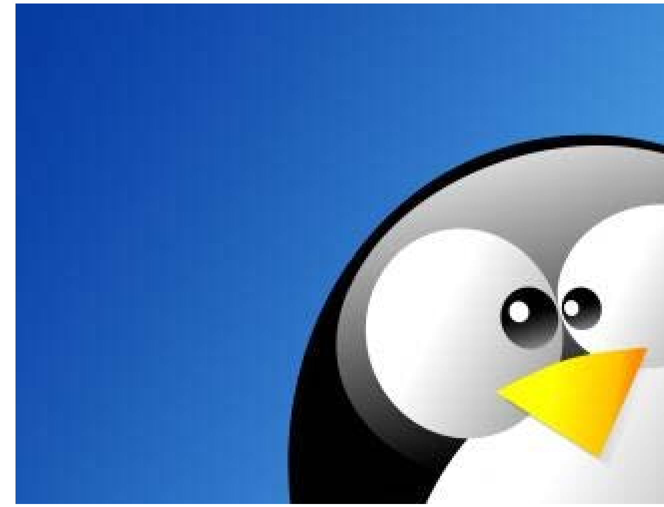
- Education and training materials – tutorial, textbook
- Greater availability of tools to support combinatorial testing – see pairwise.org
- Modify approaches to using combinatorial testing – integrating combinatorial testing with other test practices; ability to adopt CT partially or gradually – measurement tool
- Incorporate combinatorial methods into DoD guidance and industry standards; develop a community of practice

Review

video by

Tyler Mesch, Bose Corp.

Thank you for listening!



NIST
National Institute of Standards and Technology

BACKUP SLIDES FOR ADDITIONAL DISCUSSION

Examples from the National Vulnerability Database

Single variable, 1-way interaction

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

example: ***single character search string*** *in conjunction with a* ***single character replacement string****, which causes an "off by one overflow"*

3-way interaction

example: *Directory traversal vulnerability when* ***register_globals is enabled*** *and* ***magic_quotes is disabled*** *and* ***.. (dot dot) in the page parameter***

Example 2: Problem: unknown factors causing failures of F-16 ventral fin

Figure 1. LANTIRN pod carriage on the F-16.

LANTIRN = Low Altitude Navigation & Targeting Infrared for Night

It’s not supposed to look like this:

Figure 2. F-16 ventral fin damage on flight with LANTIRN

Can the problem factors be found efficiently?

Original solution: Lockheed Martin engineers spent many months with wind tunnel tests and expert analysis to consider interactions that could cause the problem

Combinatorial testing solution: modeling and simulation using ACTS

Parameter	Values
Aircraft	15, 40
Altitude	5k, 10k, 15k, 20k, 30k, 40k, 50k
Maneuver	hi-speed throttle, slow accel/dwell, L/R 5 deg side slip, L/R 360 roll, R/L 5 deg side slip, Med accel/dwell, R-L-R-L banking, Hi-speed to Low, 360 nose roll
Mach (100th)	40, 50, 60, 70, 80, 90, 100, 110, 120

Results

- Interactions causing problem included Mach points .95 and .97; multiple side-slip and rolling maneuvers
- Solution analysis tested interactions of Mach points, maneuvers, and multiple fin designs
- Problem could have been found much more efficiently and quickly
- Less expert time required

- Spreading use of combinatorial testing in the corporation:
    - Community of practice of 200 engineers
    - Tutorials and guidebooks
    - Internal web site and information forum

What tools are available?

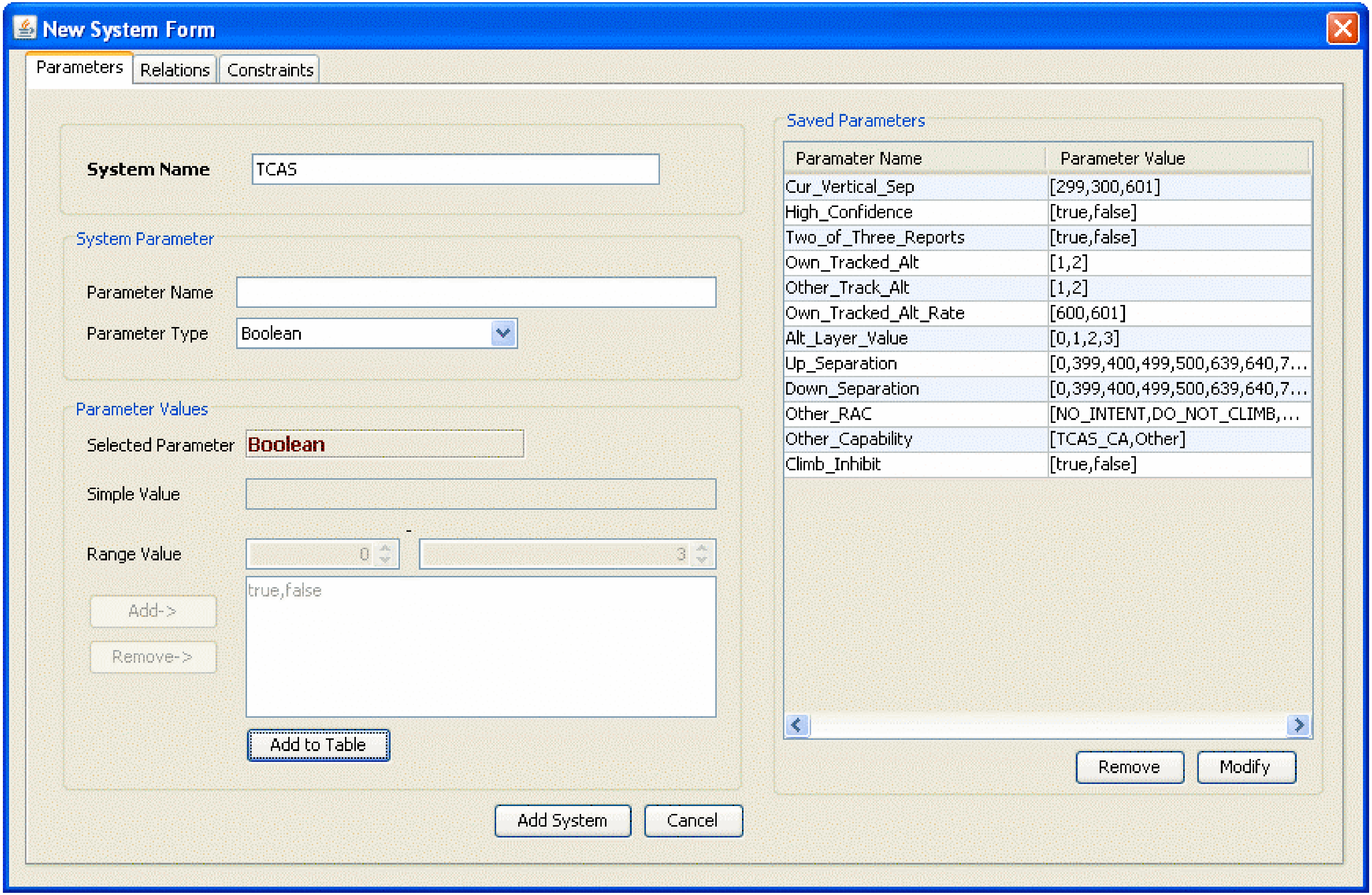
- **Covering array generator** – basic tool for test input or configurations;

- **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

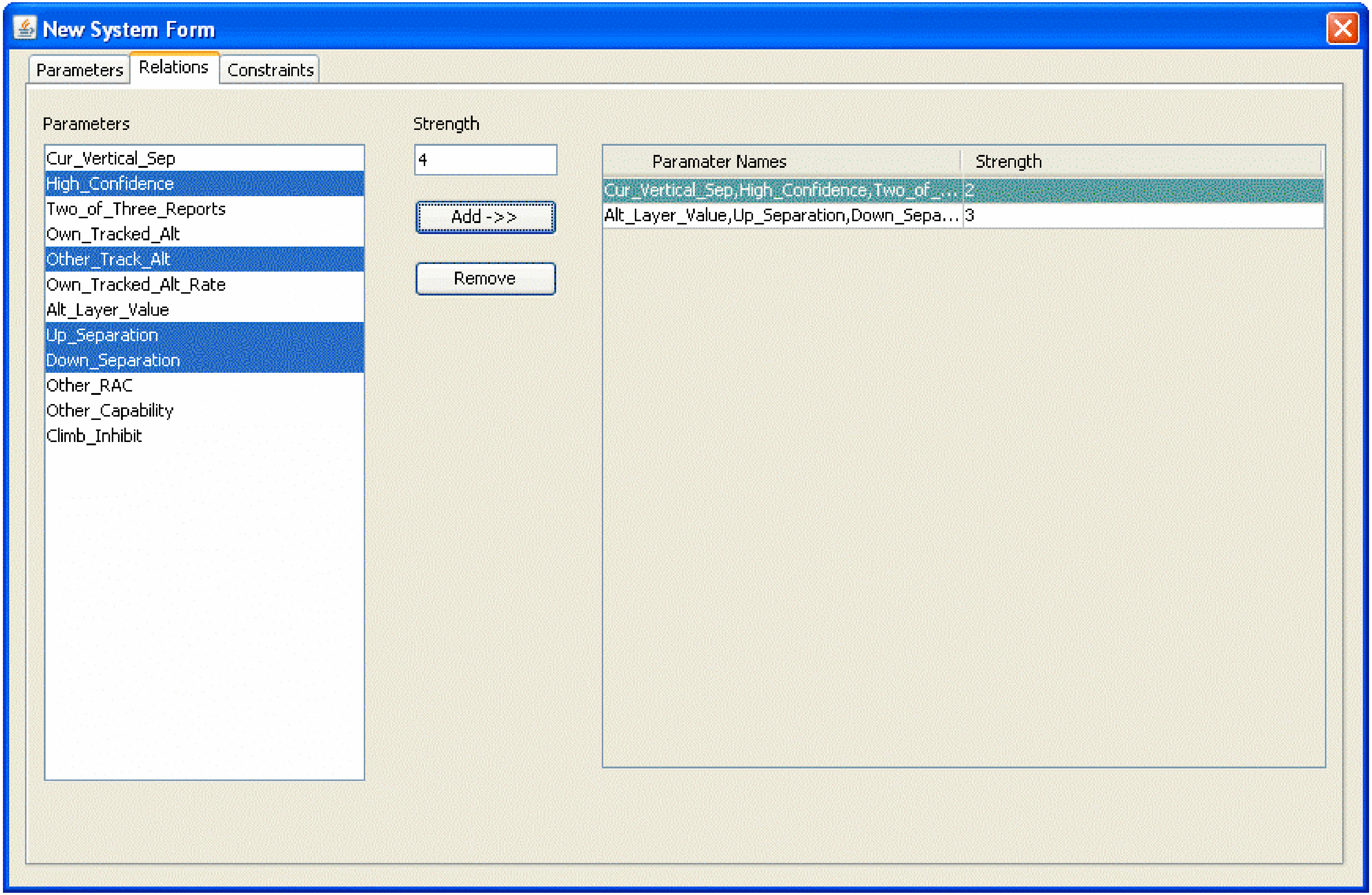
- **Domain/application specific tools**:
    - Access control policy tester
    - .NET config file generator

ACTS - Defining a new system



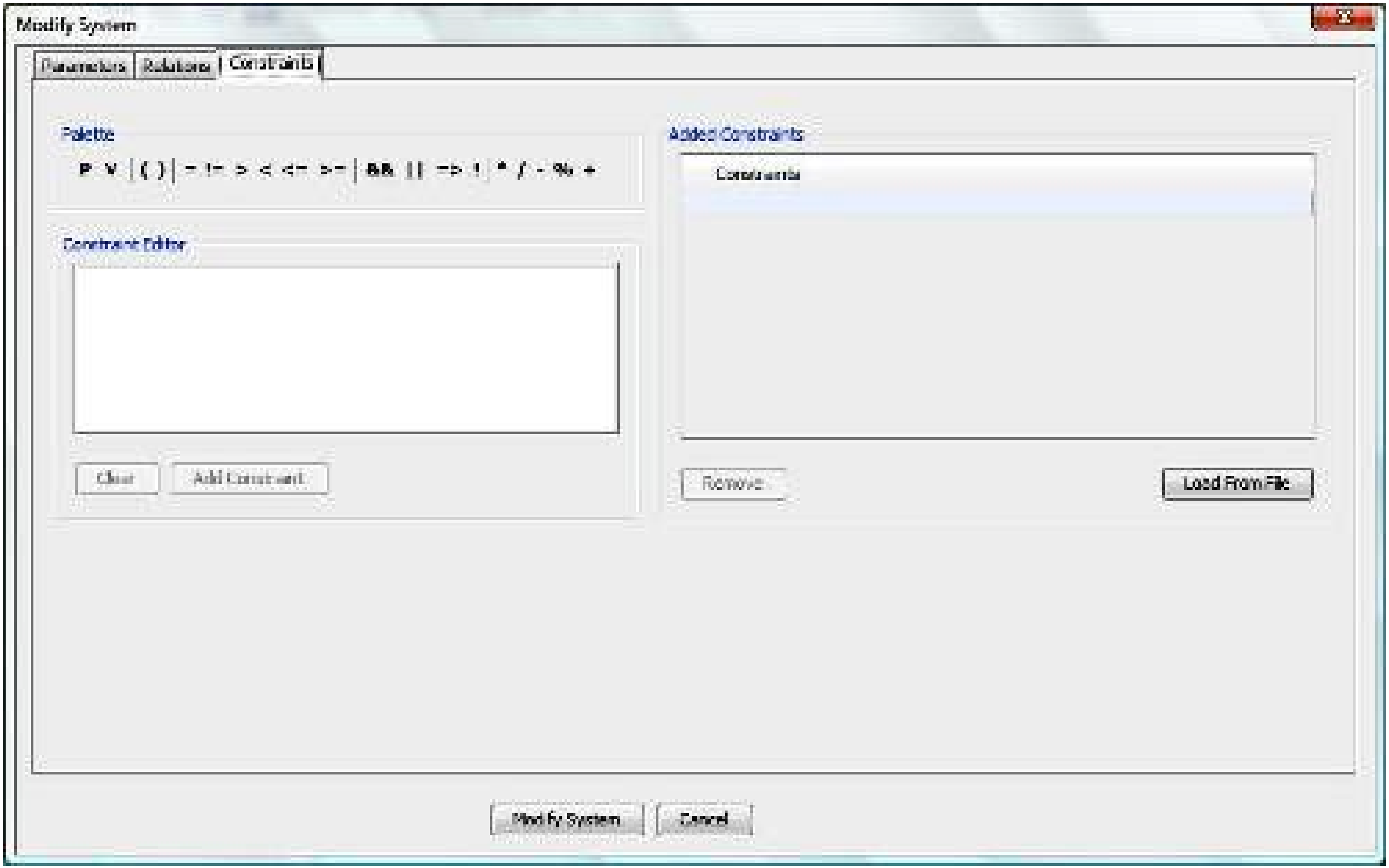
National Institute of Standards and Technology

Variable interaction strength

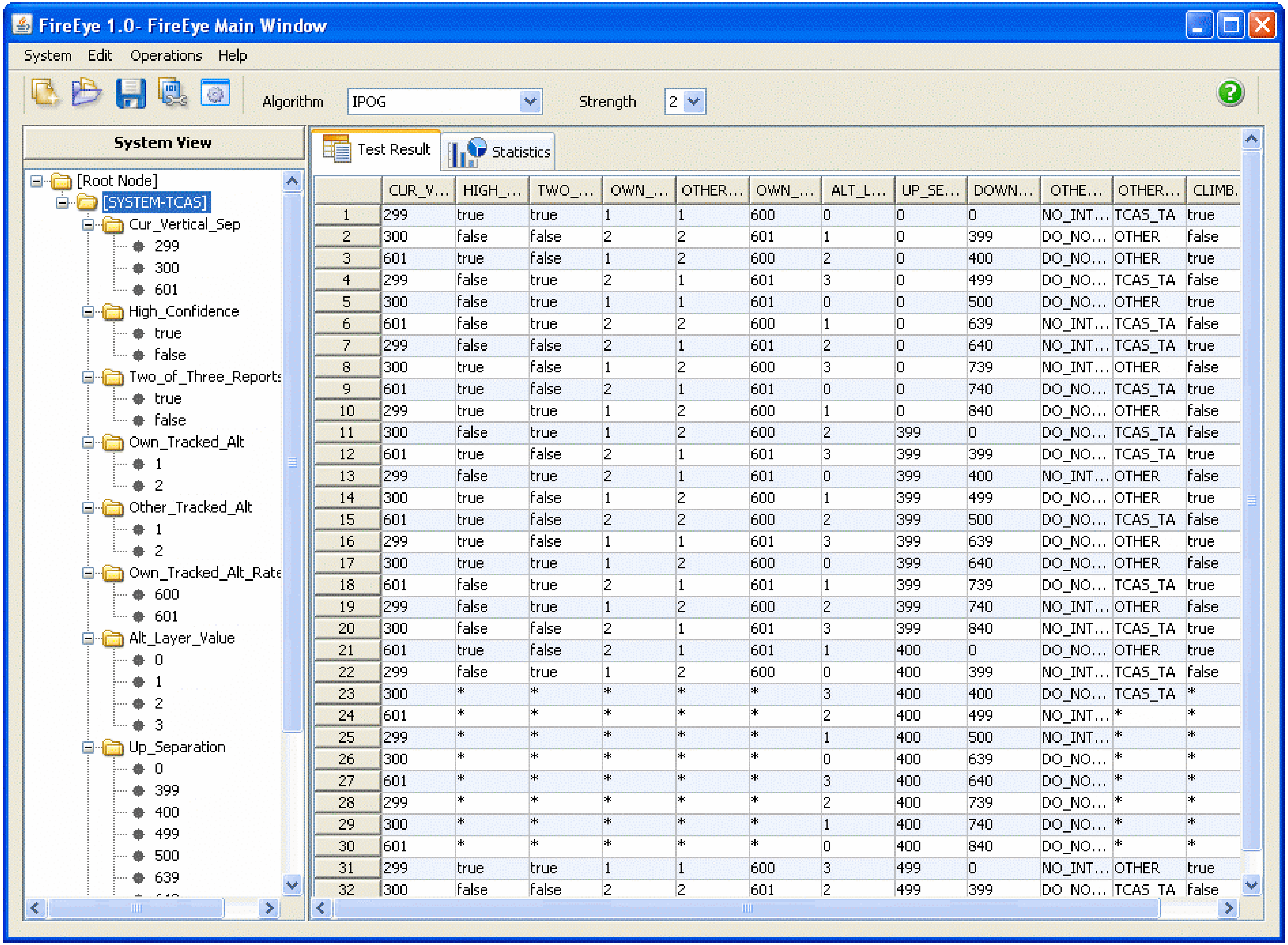


National Institute of Standards and Technology

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

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
```

NIST National Institute of Standards and Technology

Model checking example

```
-- specification for a portion of tcas - altitude separation.
-- The corresponding C code is originally from Siemens Corp. Research
-- Vadim Okun 02/2002
MODULE main
VAR
  Cur_Vertical_Sep : { 299, 300, 601 };
  High_Confidence : boolean;
...
init(alt_sep) := START_;
  next(alt_sep) := case
    enabled & (intent_not_known | !tcas_equipped) : case
      need_upward_RA & need_downward_RA : UNRESOLVED;
      need_upward_RA : UPWARD_RA;
      need_downward_RA : DOWNWARD_RA;
      1 : UNRESOLVED;
    esac;
    1 : UNRESOLVED;
  esac;
...
SPEC AG ((enabled & (intent_not_known | !tcas_equipped) &
!need_downward_RA & need_upward_RA) -> AX (alt_sep = UPWARD_RA))
-- "FOR ALL executions,
-- IF enabled & (intent_not_known ....
-- THEN in the next state alt_sep = UPWARD_RA"
```

Computation Tree Logic

National Institute of Standards and Technology

The usual logic operators,plus temporal:

A φ - All: φ holds on all paths starting from the current state.

E φ - Exists: φ holds on some paths starting from the current state.

G φ - Globally: φ has to hold on the entire subsequent path.

F φ - Finally: φ eventually has to hold

X φ - Next: φ has to hold at the next state

[others not listed]

execution paths

states on the execution paths

```
SPEC AG ((enabled & (intent_not_known |
!tcas_equipped) & !need_downward_RA & need_upward_RA)
-> AX (alt_sep = UPWARD_RA))
```

"FOR ALL executions,
IF enabled & (intent_not_known ....
THEN in the next state alt_sep = UPWARD_RA"

What is the most effective way to integrate combinatorial testing with model checking?

- Given **AG(P -> AX(R))**
  "for all paths, in every state,
  if P then in the next state, R holds"
- For k-way variable combinations, **v1 & v2 & ... & vk**
- vi abbreviates "var1 = val1"
- Now combine this constraint with assertion to produce counterexamples. Some possibilities:

1. **AG(v1 & v2 & ... & vk & P -> AX !(R))**
2. **AG(v1 & v2 & ... & vk -> AX !(1))**
3. **AG(v1 & v2 & ... & vk -> AX !(R))**

What happens with these assertions?

1. `AG(v1 & v2 & ... & vk & P -> AX !(R))`

P may have a negation of one of the v_i, so we get

`0 -> AX !(R))`

always true, so no counterexample, no test.

This is too restrictive!

1. `AG(v1 & v2 & ... & vk -> AX !(1))`

   The model checker makes non-deterministic choices for variables not in v1..vk, so all R values may not be covered by a counterexample.

   This is too loose!

2. `AG(v1 & v2 & ... & vk -> AX !(R))`

   Forces production of a counterexample for each R.

   This is just right!

NIST
National Institute of Standards and Technology

Modeling & Simulation

1. **Aerospace - Lockheed Martin – analyze structural failures for aircraft design**

2. **Network defense/offense operations - NIST – analyze network configuration for vulnerability to deadlock**

Example 3: Network Simulation

- "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)

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?

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

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)

Example 4: 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*

NIST
National Institute of Standards and Technology

Finding Buffer Overflows

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
        ……
conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024, sizeof(char));
                ……
        pPostData=conn[sid].PostData;
        do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
                  ……
                pPostData+=rc;
                x+=rc;
        } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

Interaction: request-method=”POST”, content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
        ……
conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024, sizeof(char));
                ……
        pPostData=conn[sid].PostData;
        do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
                  ……
                pPostData+=rc;
                x+=rc;
        } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

National Institute of Standards and Technology

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

true branch

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
        ……
conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024, sizeof(char));
                ……
        pPostData=conn[sid].PostData;
        do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
                  ……
                pPostData+=rc;
                x+=rc;
        } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
        ......
        conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024, sizeof(char));
              ......
       pPostData=conn[sid].PostData;
       do {
              rc=recv(conn[sid].socket, pPostData, 1024, 0);
                ......
              pPostData+=rc;
              x+=rc;
       } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

true branch

Interaction: request-method="POST", content-length = -1000, data= a string > 24 bytes

National Institute of Standards and Technology

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
        ……
        conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024,
sizeof(char));
                ……
        pPostData=conn[sid].PostData;
        do {
                rc=recv(conn[sid].socket, pPostData, 1024, 0);
                  ……
                pPostData+=rc;
                x+=rc;
        } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

true branch

Allocate -1000 + 1024 bytes = 24 bytes

Interaction: request-method=”POST”, content-length = -1000, data= a string > 24 bytes

```
if (strcmp(conn[sid].dat->in_RequestMethod, "POST")==0) {
        if (conn[sid].dat->in_ContentLength<MAX_POSTSIZE) {
        ……
        conn[sid].PostData=calloc(conn[sid].dat->in_ContentLength+1024, sizeof(char));
              ……
       pPostData=conn[sid].PostData;
       do {
               rc=recv(conn[sid].socket, pPostData, 1024, 0);
                 ……
               pPostData+=rc;
               x+=rc;
       } while ((rc==1024)||(x<conn[sid].dat->in_ContentLength));
conn[sid].PostData[conn[sid].dat->in_ContentLength]='\0';
 }
```

true branch

Allocate -1000 + 1024 bytes = 24 bytes

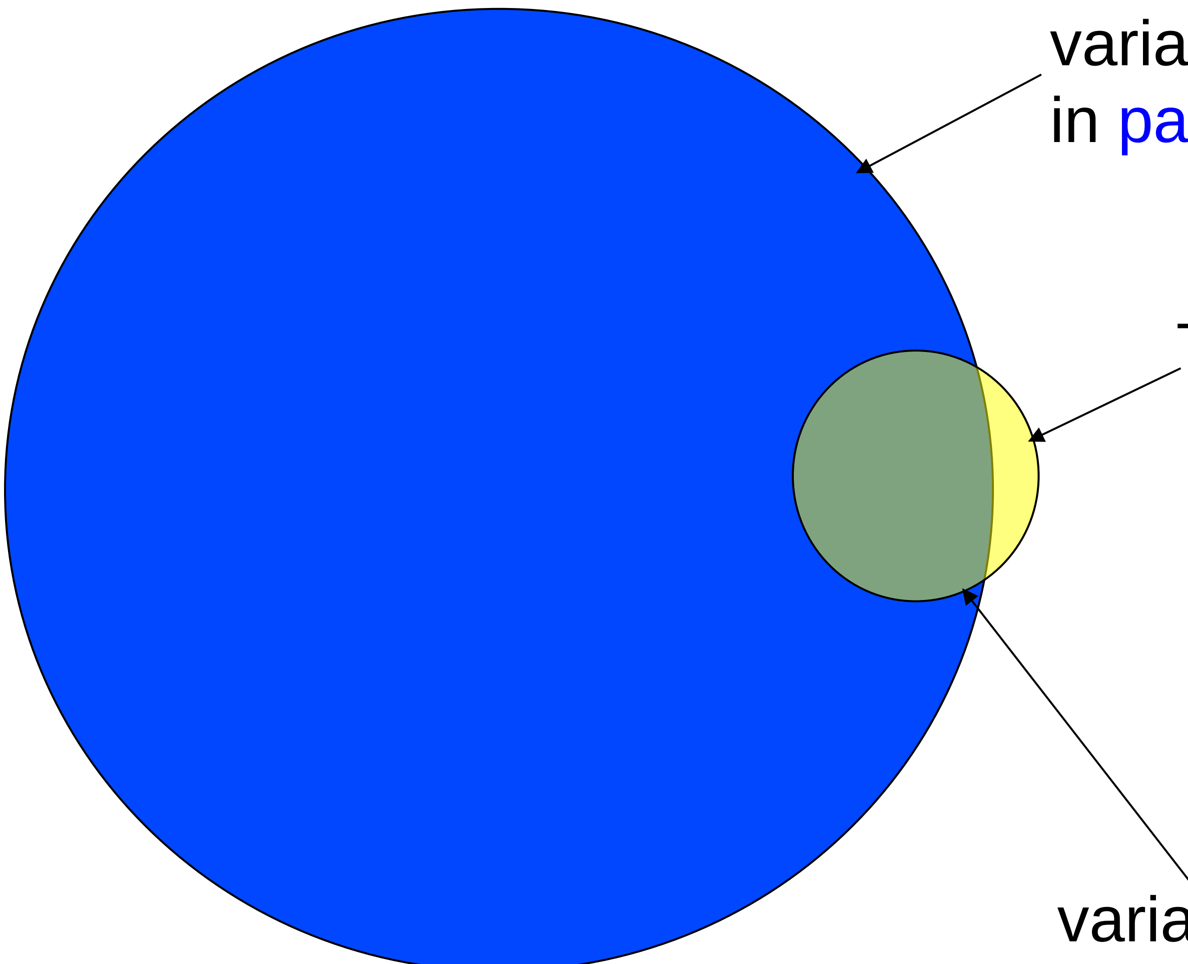
Boom!

Tutorial Overview

1. Why are we doing this?
2. What is combinatorial testing?
3. What tools are available?
4. Is this stuff really useful in the real world?
5. **What's next?**

Fault location

Given: a set of tests that the SUT fails, which combinations of variables/values triggered the failure?

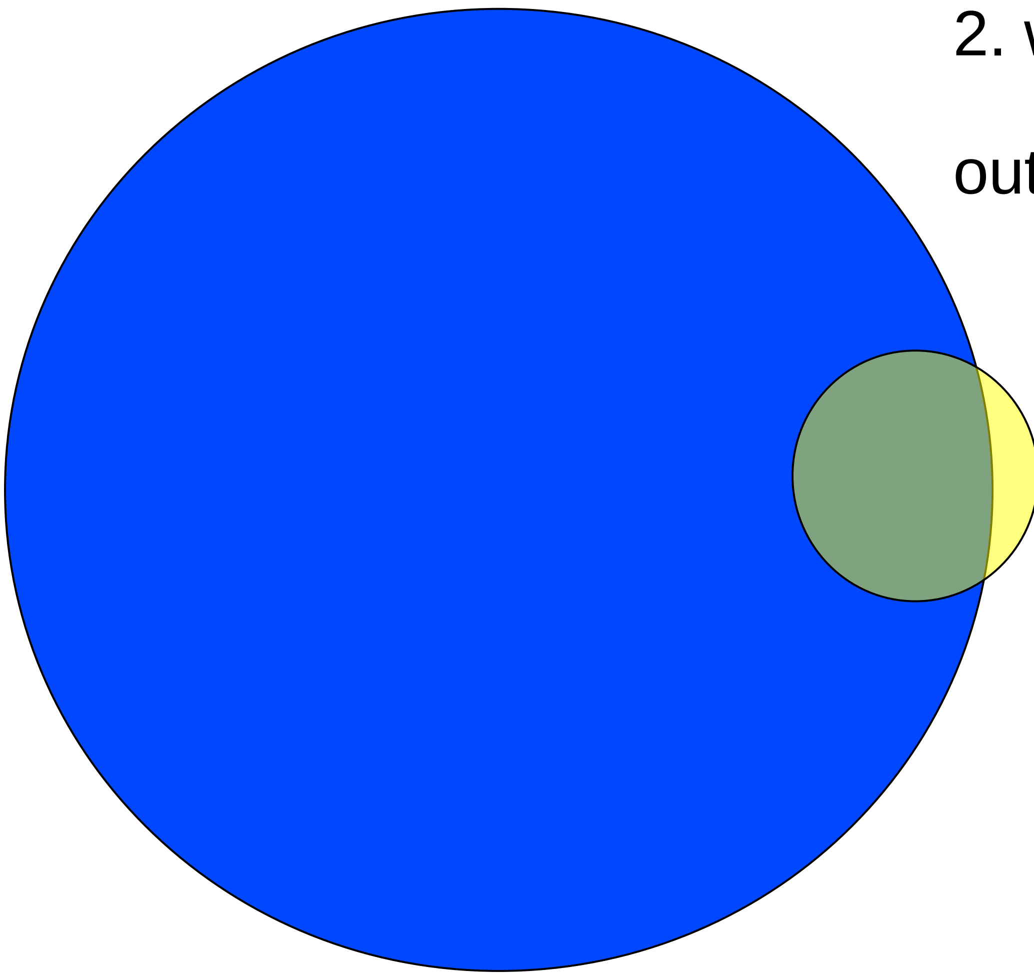


Fault location – what's the problem?

If they're in failing set but not in passing set:

1. which ones triggered the failure?
2. which ones don't matter?

out of $v^t\binom{n}{t}$ combinations



Example:
30 variables, 5 values each
= 445,331,250
5-way combinations

142,506 combinations
in each test

Tutorial Overview

1. Why are we doing this?
2. What is combinatorial testing?
3. What tools are available?
4. Is this stuff really useful in the real world?
5. What's next?

Tutorial Overview

1. Why are we doing this?
2. **What is combinatorial testing?**
3. What tools are available?
4. Is this stuff really useful in the real world?
5. What's next?

Tutorial Overview

1. Why are we doing this?
2. What is combinatorial testing?
3. **What tools are available?**
4. Is this stuff really useful in the real world?
5. What's next?

Tradeoffs

- Advantages
  - Tests rare conditions
  - Produces high code coverage
  - Finds faults faster
  - May be lower overall testing cost
- Disadvantages
  - Expensive at higher strength interactions (>4-way)
  - May require high skill level in some cases (if formal models are being used)