National Institute of Standards and Technology

Combinatorial Methods for System and Software Testing

Rick Kuhn

National Institute of
Standards and Technology
Gaithersburg, MD

Information Security & Privacy Advisory Board

June 16, 2016

Applications

Software testing – primary application of these methods

- functionality testing and security vulnerabilities
- approx 2/3 of vulnerabilities from implementation faults

Modeling and simulation – ensure coverage of complex cases

- measure coverage of traditional Monte Carlo sim
- faster coverage of input space than randomized input

Performance tuning – determine most effective combination of configuration settings among a large set of factors

>> systems with a large number of factors that interact <<

Why combinatorial testing? - examples

- Cooperative R&D Agreement w/ Lockheed Martin
  - 2.5 year study, 8 Lockheed Martin pilot projects in aerospace software
  - Results: save 20% of test costs; increase test coverage by 20% to 50%
- Rockwell Collins applied NIST method and tools on testing to FAA life-critical standards
  - Found practical for industrial use
  - Enormous cost reduction

Average software: testing typically 50% of total dev cost
Civil aviation: testing >85% of total dev cost (NASA rpt)

Research areas

- Empirical data on software failures
- Covering array generation
- Combinatorial coverage measurement of input space
- Sequence covering arrays
- Oracle-free testing

What is the empirical basis?

- NIST studied software failures in 15 years of FDA medical device recall data
- What causes software failures?
  - logic errors? calculation errors? inadequate input checking? interaction faults? Etc.

Interaction faults: e.g., failure occurs if

```
altitude = 0 && volume < 2.2
```

(interaction between 2 factors)

So this is a **2-way interaction**

=> testing all pairs of values can find this fault

How are interaction faults distributed?

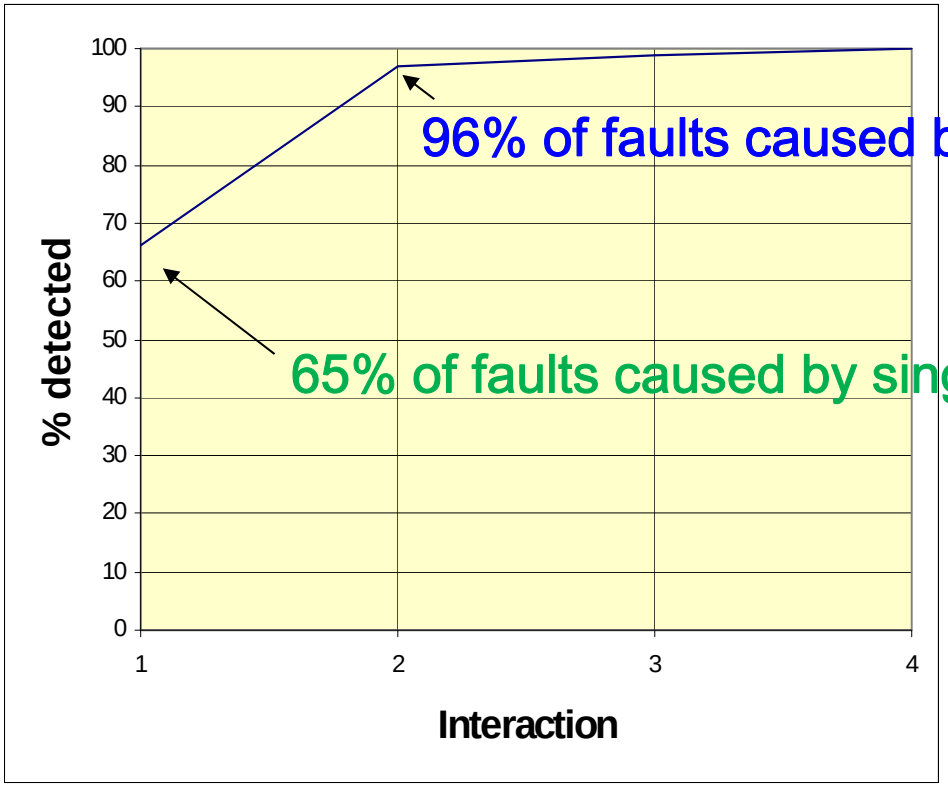
- Interactions e.g., failure occurs if

  pressure < 10 (1-way interaction)

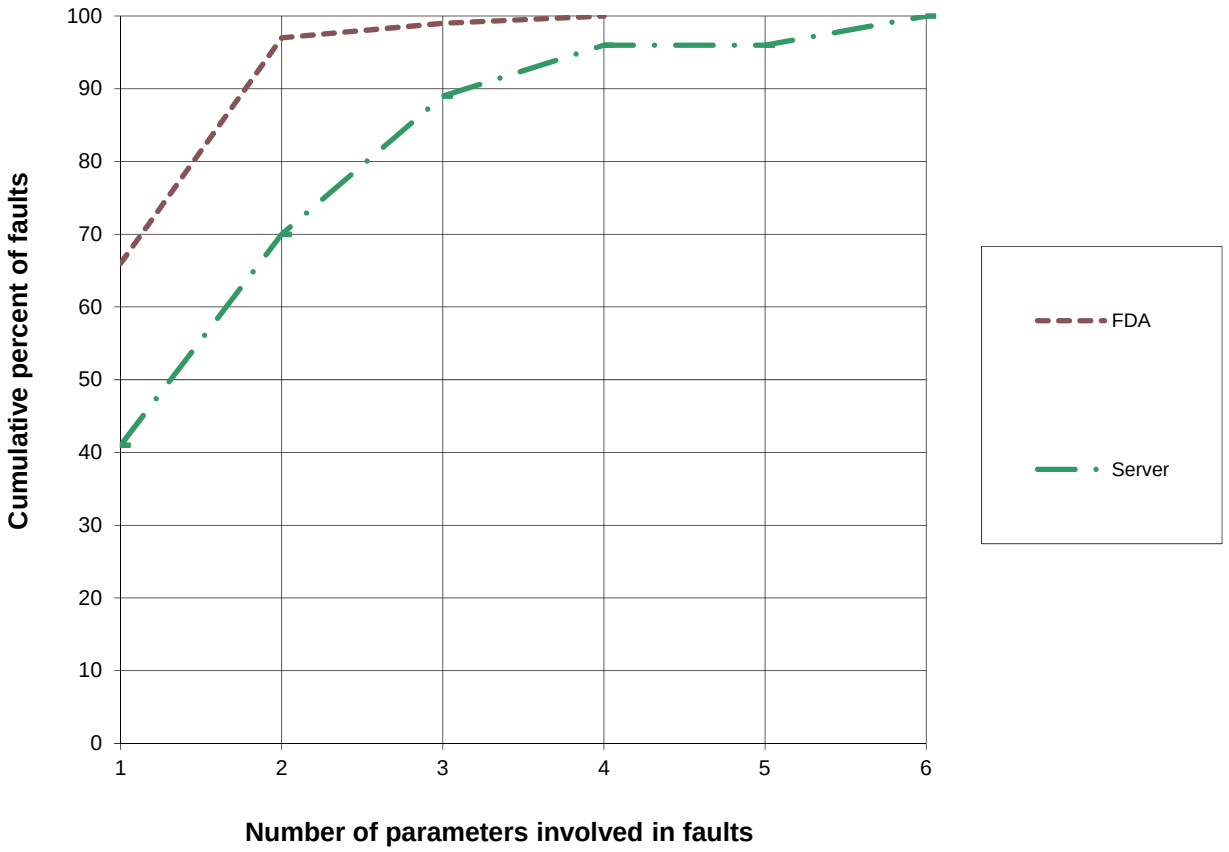
  pressure < 10 & volume > 300 (2-way interaction)

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

- Surprisingly, no one had looked at interactions > 2-way before



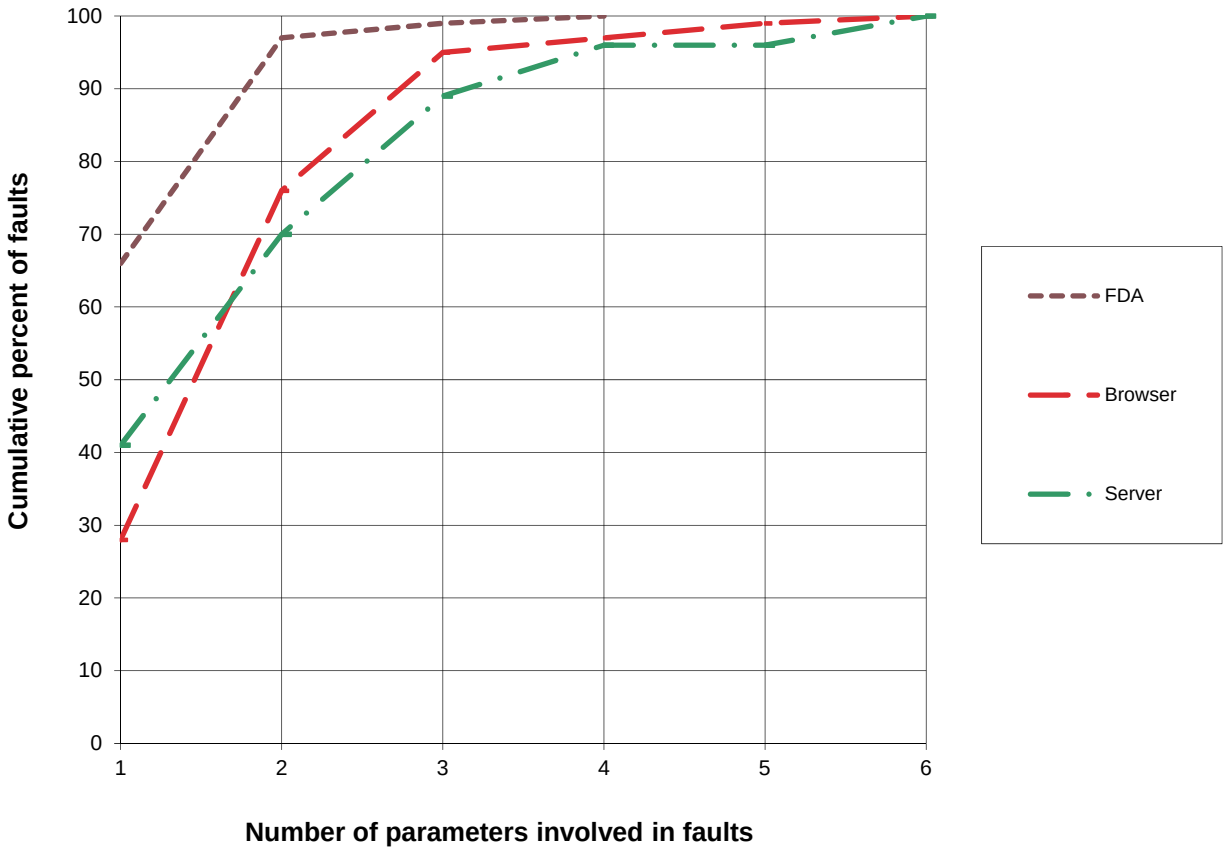
Server



These faults more complex than medical device software!!

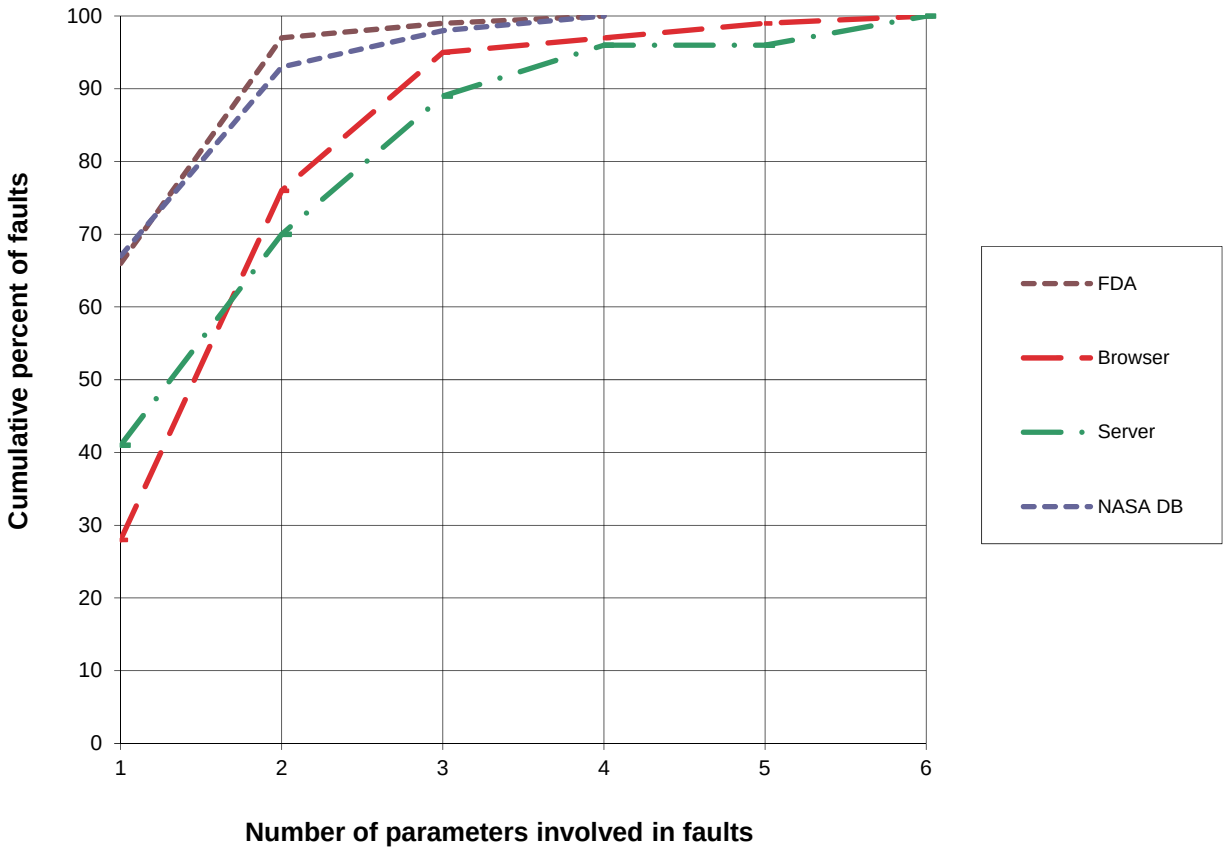
Why?

Browser



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

NASA distributed database

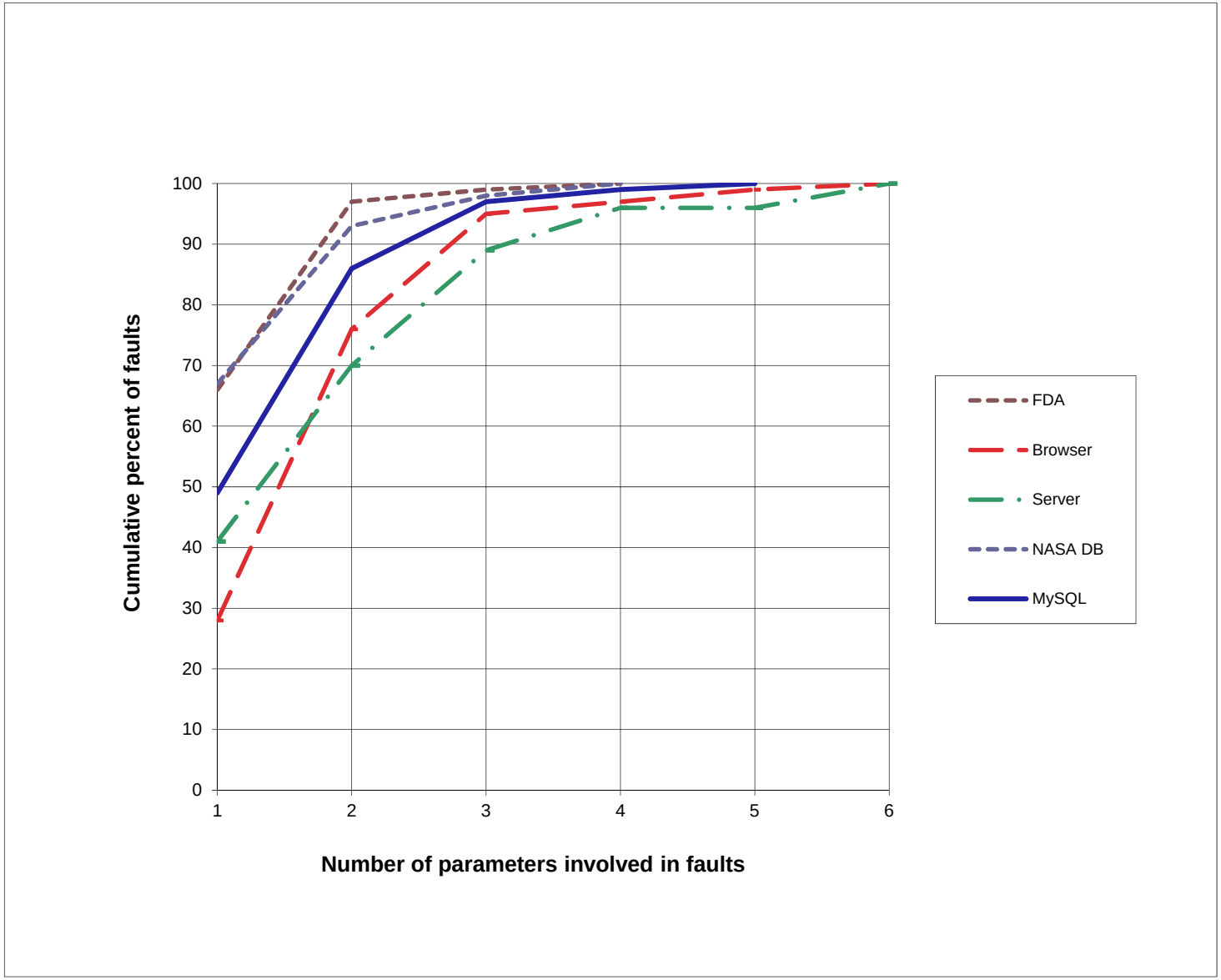


Note: initial testing

but ….

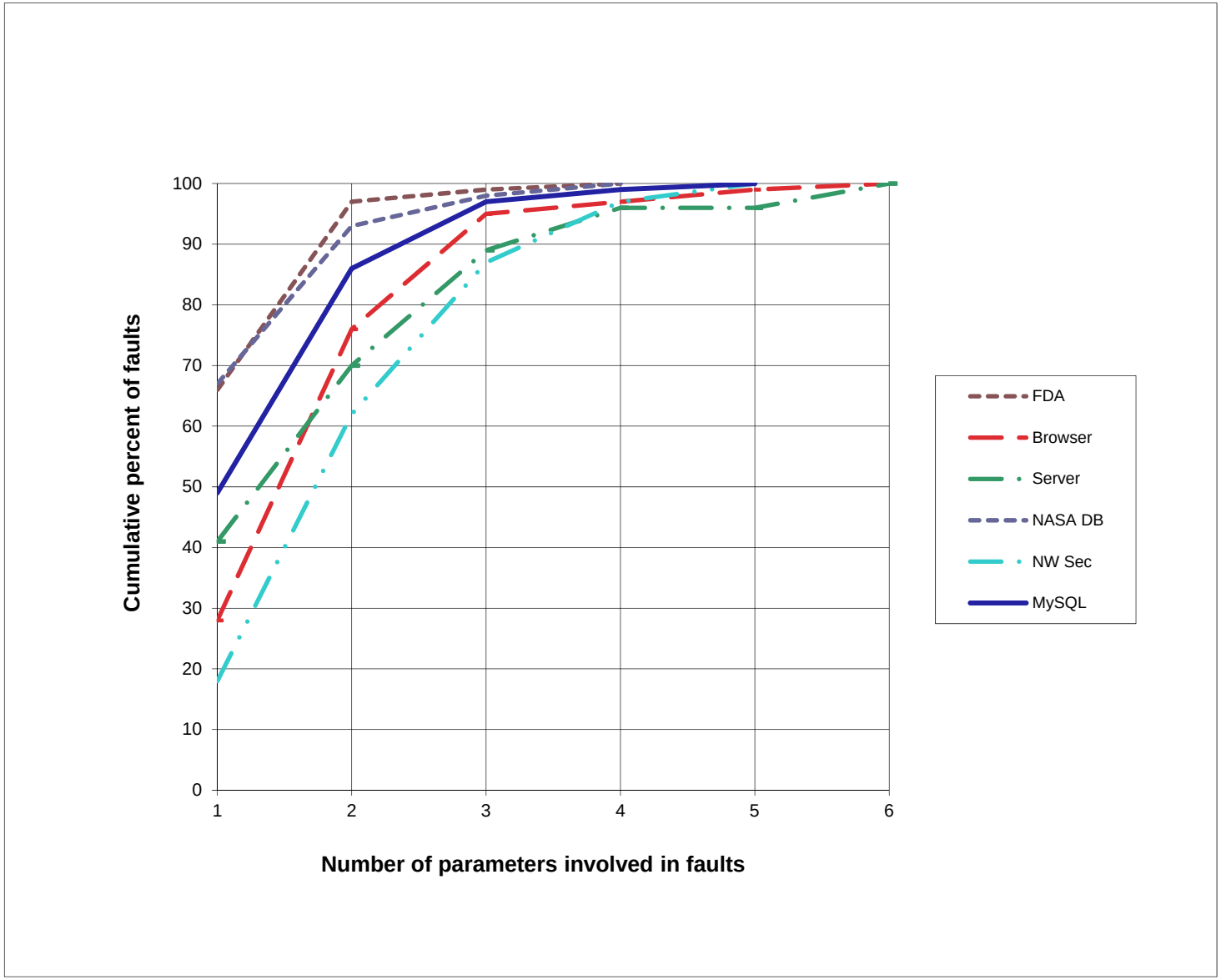
Fault profile better than medical devices!

MySQL



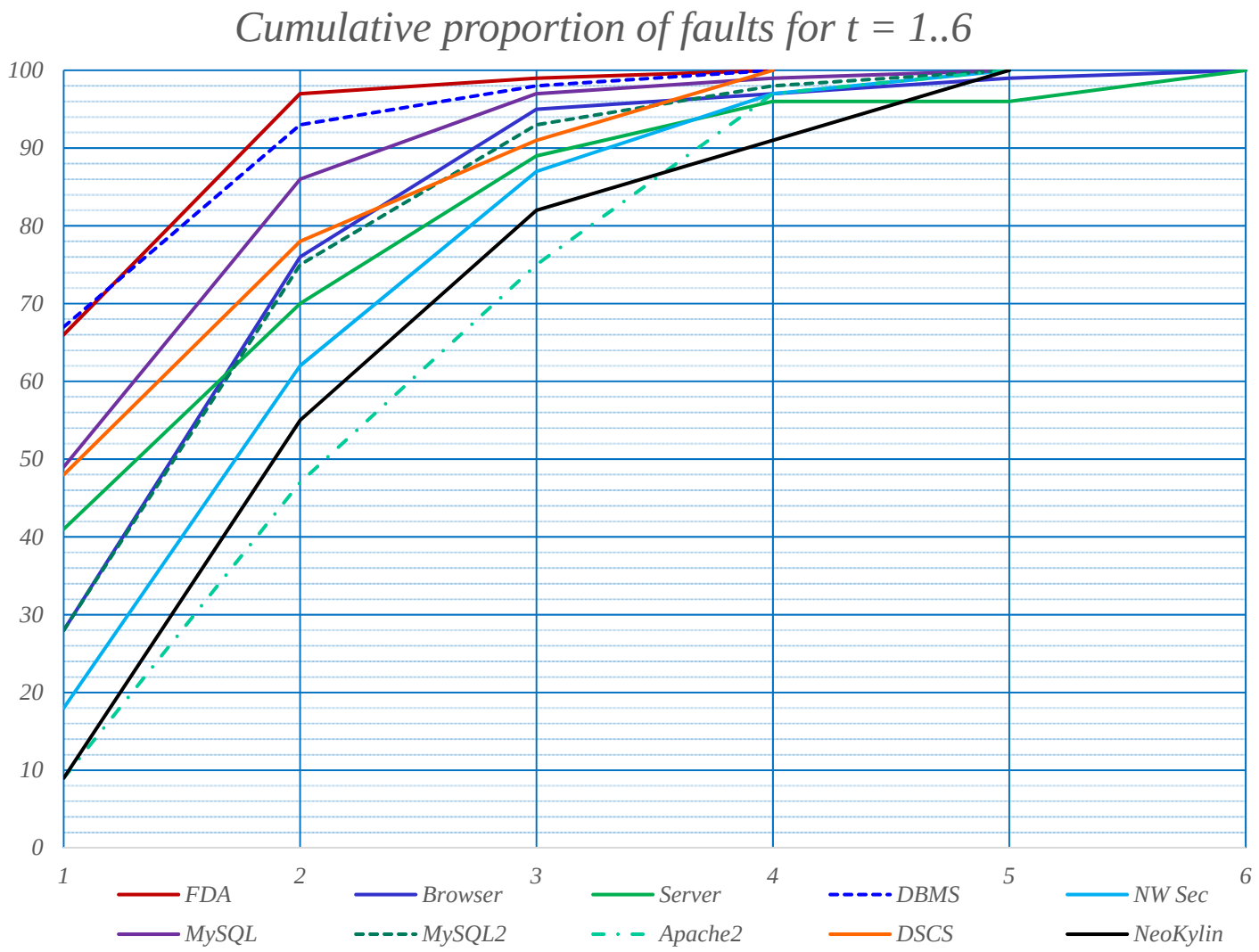
National Institute of Standards and Technology

TCP/IP



National Institute of Standards and Technology

Wait, there's more



- Number of factors involved in failures is small
- No failure involving more than 6 variables has been seen

How does this knowledge help?

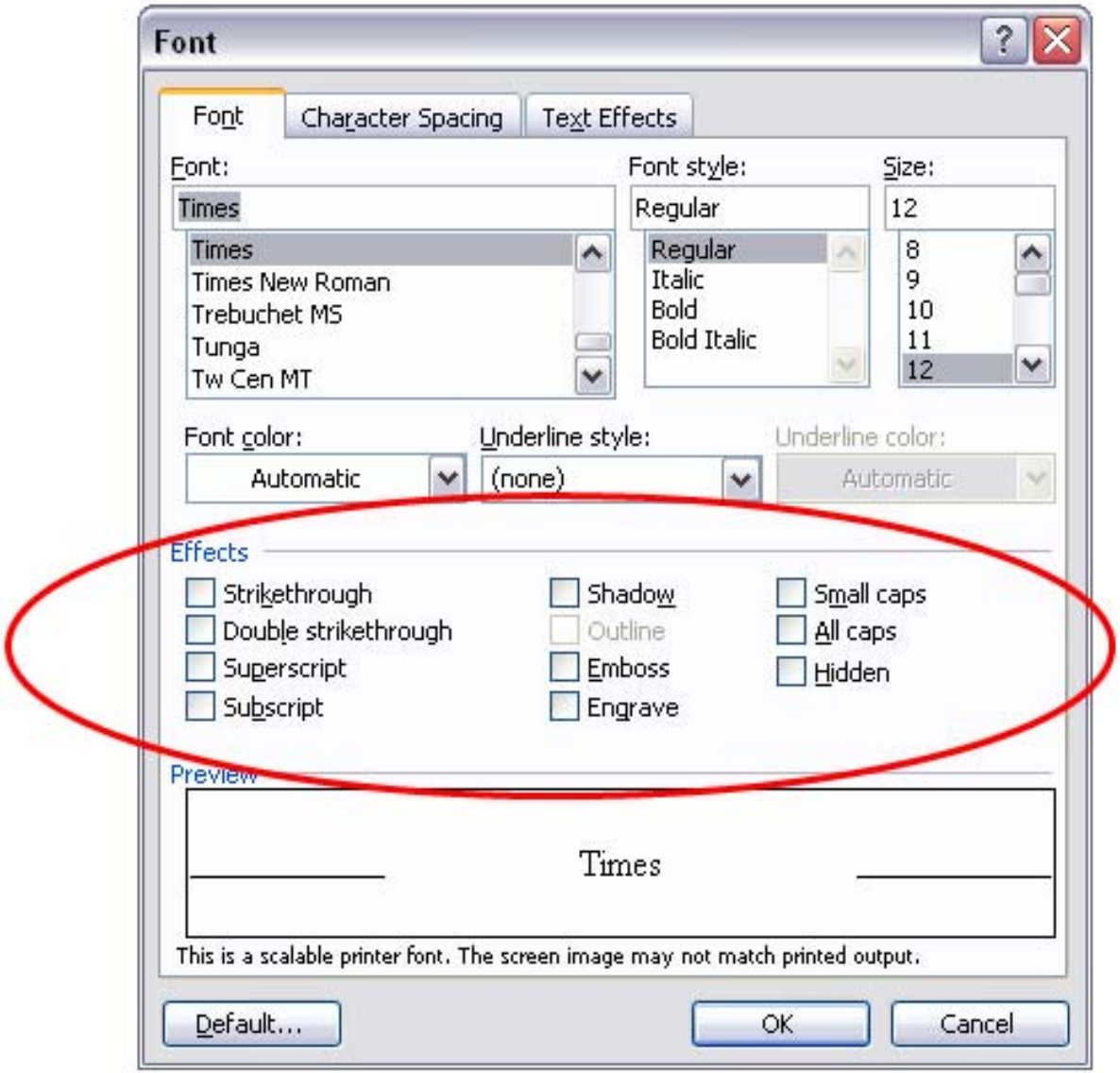
Interaction rule: When all faults are triggered by the interaction of *t* or fewer variables, then testing all *t*-way combinations is *pseudo-exhaustive* and can provide strong assurance.

It is nearly always impossible to exhaustively test all possible input combinations

The interaction rule says we don't have to

(within reason; we still have value propagation issues, equivalence partitioning, timing issues, more complex interactions, . . .)

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

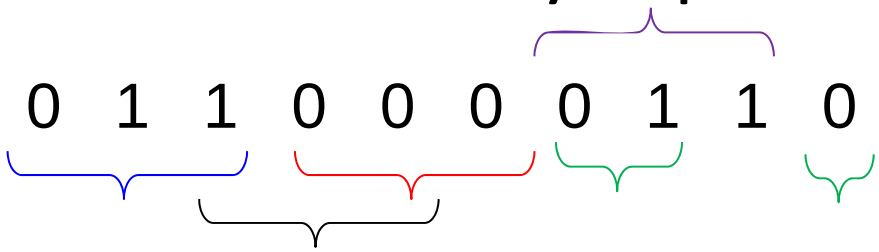


How Many Tests Would It Take?

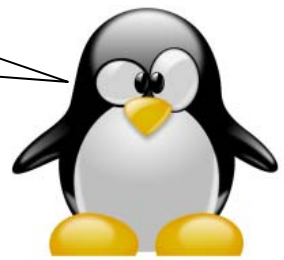
- There are 10 effects, each can be on or off
- All combinations is 2^{10} = 1,024 tests
- What if our budget is too limited for these tests?
- Instead, let's look at all 3-way interactions …

Now How Many Would It Take?

- There are $\binom{10}{3}$ = 120 3-way interactions.
- Each triple has 2^3 = 8 settings: 000, 001, 010, 011, ...
- 120 x 8 = 960 combinations
- Each test exercises many triples:



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



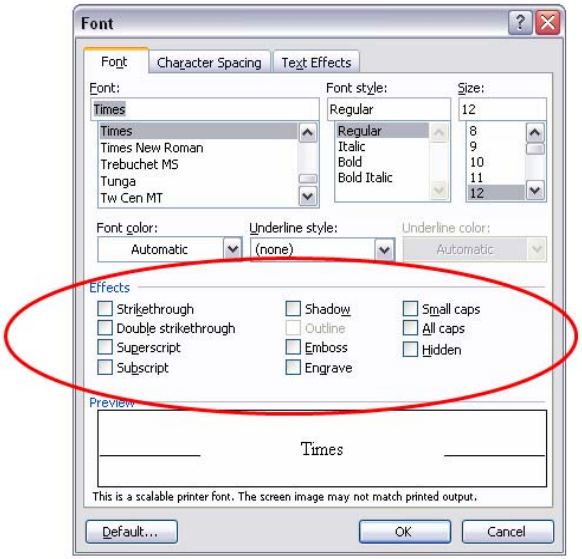
A covering array of 13 tests

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

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

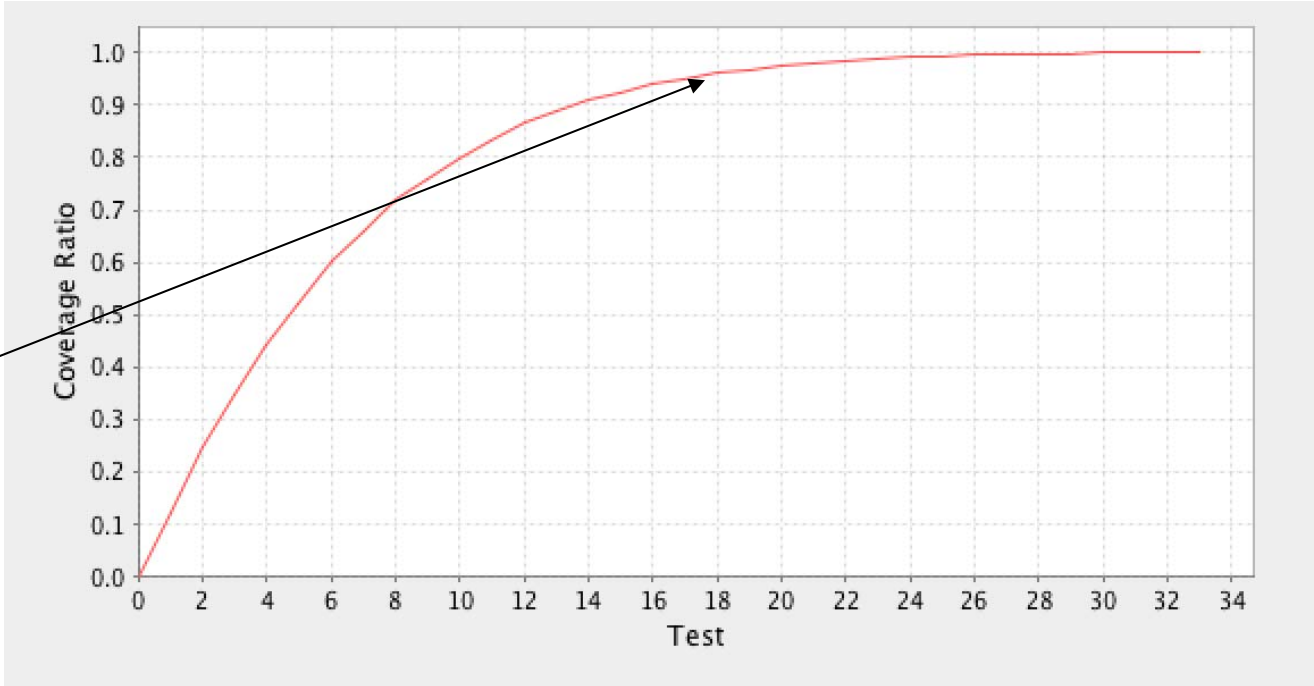
National Institute of Standards and Technology

How many tests are needed?

- Number of tests: proportional to $v^t \log n$ for v values, n variables, t-way interactions
- Good news: tests increase logarithmically with the number of parameters
  => even very large test problems are OK (e.g., 200 parameters)
- Bad news: increase exponentially with interaction strength t
  => select small number of representative values (but we always have to do this for any kind of testing)

However:

- coverage increases rapidly
- for 30 boolean variables
- 33 tests to cover all 3-way combinations
- but only 18 tests to cover about 95% of 3-way combinations



Testing inputs – combinations of variable values

Suppose we have a system with on-off switches.

Software must produce the right response for any combination of switch settings

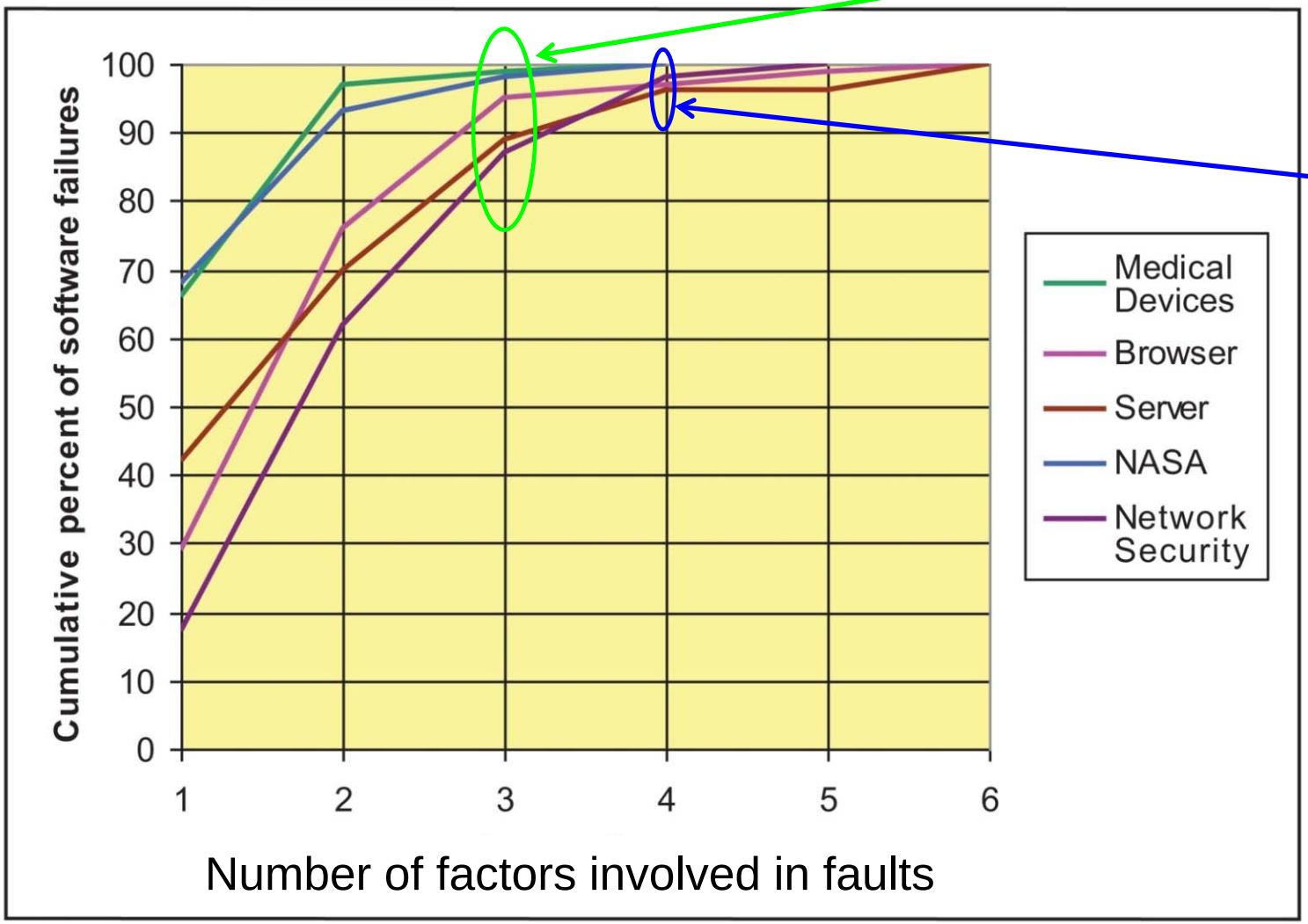
How do we test this?

34 switches = 2^{34} = 1.7 x 10^{10} possible inputs = 17 billion tests

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

- 34 switches = 17 billion tests
- For 3-way interactions, need only **33** tests
- For 4-way interactions, need only **85** tests

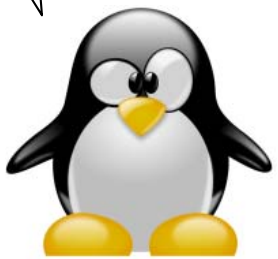
NIST
National Institute of Standards and Technology



33 tests for this (average) range of fault detection

85 tests for this (average) range of fault detection

That's way better than 17 billion!



Available Tools

- **Covering array generator** – basic tool for test input or configurations;
- **Input modeling tool** – design inputs to covering array generator using classification tree editor; useful for partitioning input variable values
- **Fault location tool** – identify combinations and sections of code likely to cause problem
- **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

Case study example: Subway control system

Real-world experiment by grad students, Univ. of Texas at Dallas

Original testing by company: 2 months

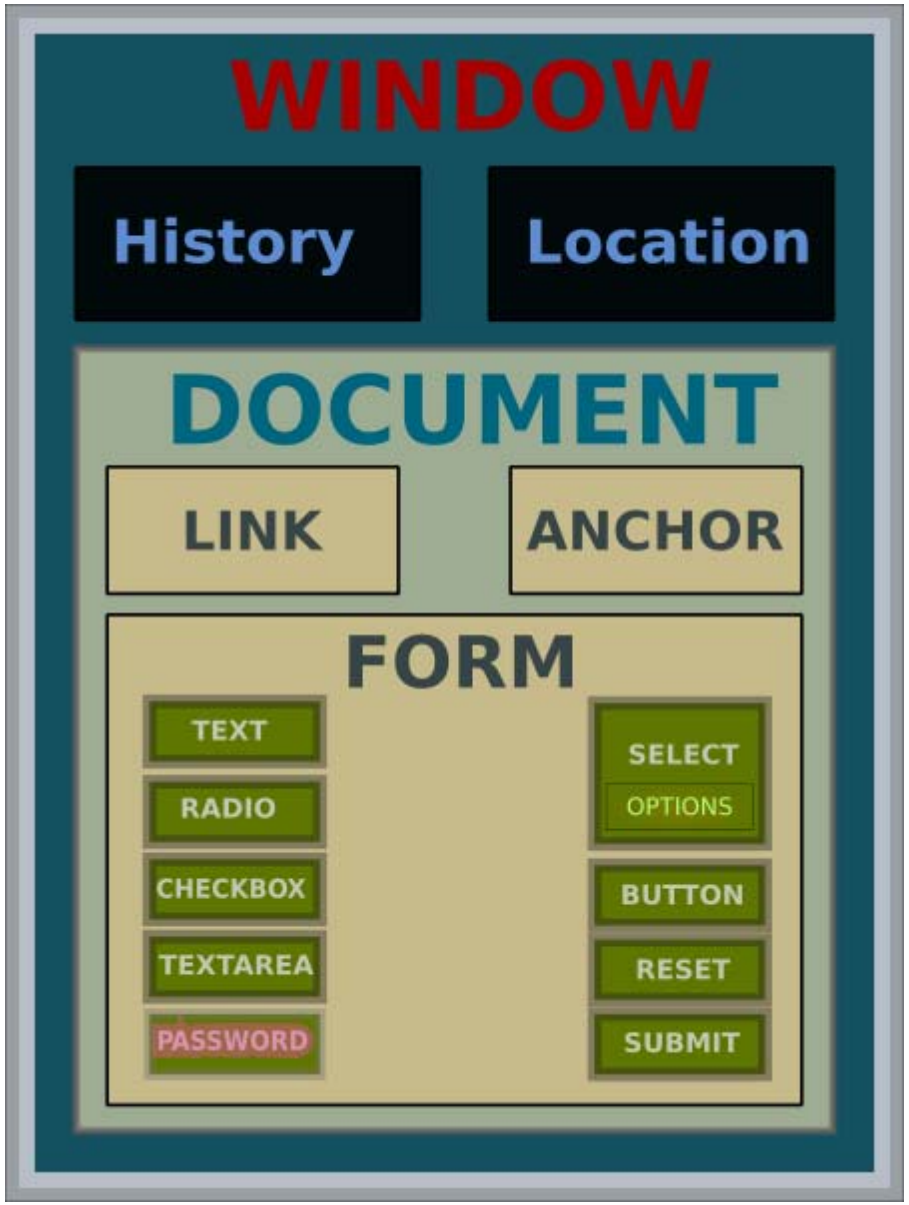
Combinatorial testing by U. Texas students: 2 weeks

Result: approximately 3X as many bugs found, in 1/4 the time
=> 12X improvement

Results

		Number of test cases	Number of bugs found	Did CT find all original bugs?
Package 1	Original	98	2	-
	CT	49	6	Yes
Package 2	Original	102	1	-
	CT	77	5	Yes
Package 3	Original	116	2	-
	CT	80	7	Miss 1
Package 4	Original	122	2	-
	CT	90	4	Yes

Research question – validate interaction rule?



- 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

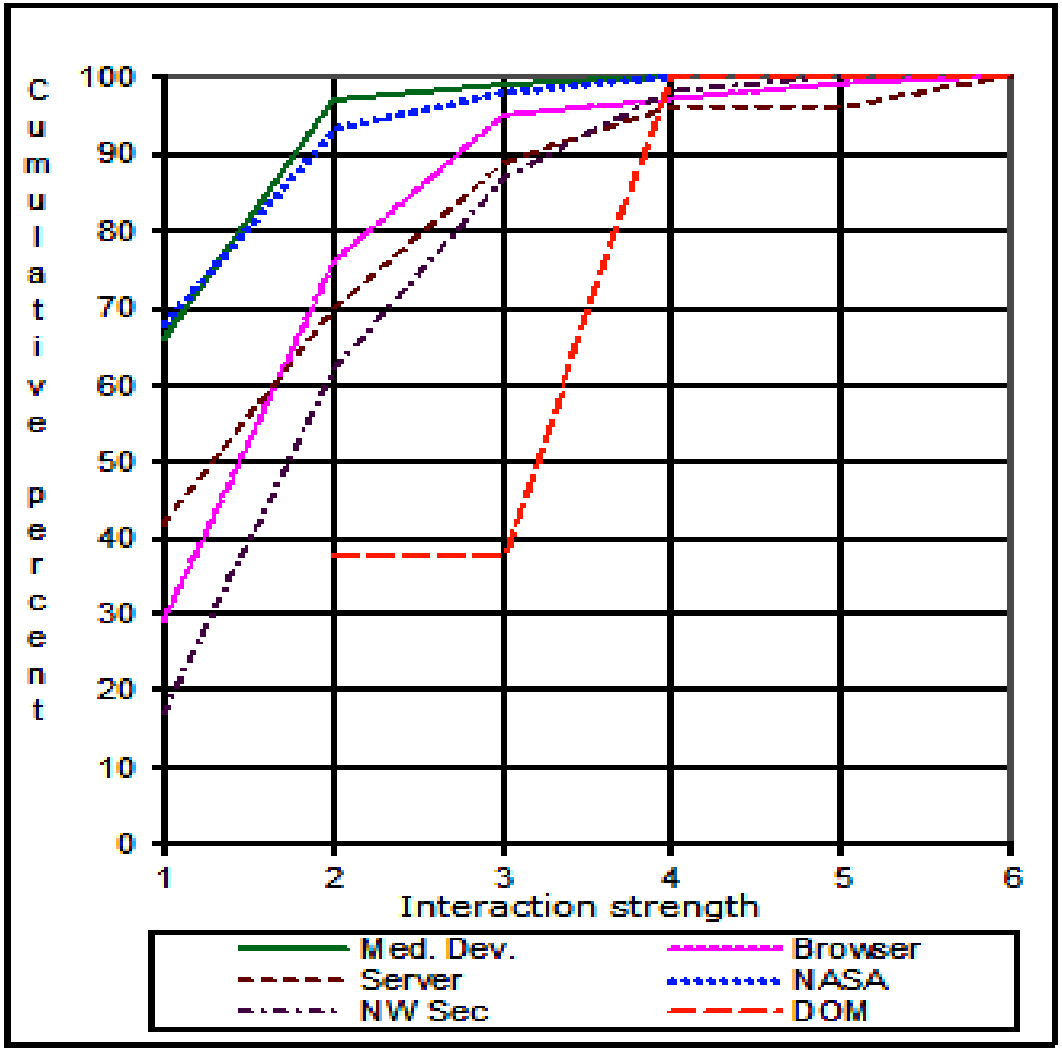
NIST National Institute of Standards and Technology

Document Object Model Events

Combinatorial test set:

t	Tests	% of Orig.	Test Results	
			Pass	Fail
2	702	1.92%	202	27
3	1342	3.67%	786	27
4	1818	4.96%	437	72
5	2742	7.49%	908	72
6	4227	11.54 %	1803	72

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



NIST
National Institute of Standards and Technology

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 range finder
b	connect telecom
c	connect satellite link
d	connect GPS
e	connect video
f	connect UAV

Sequence Covering Array

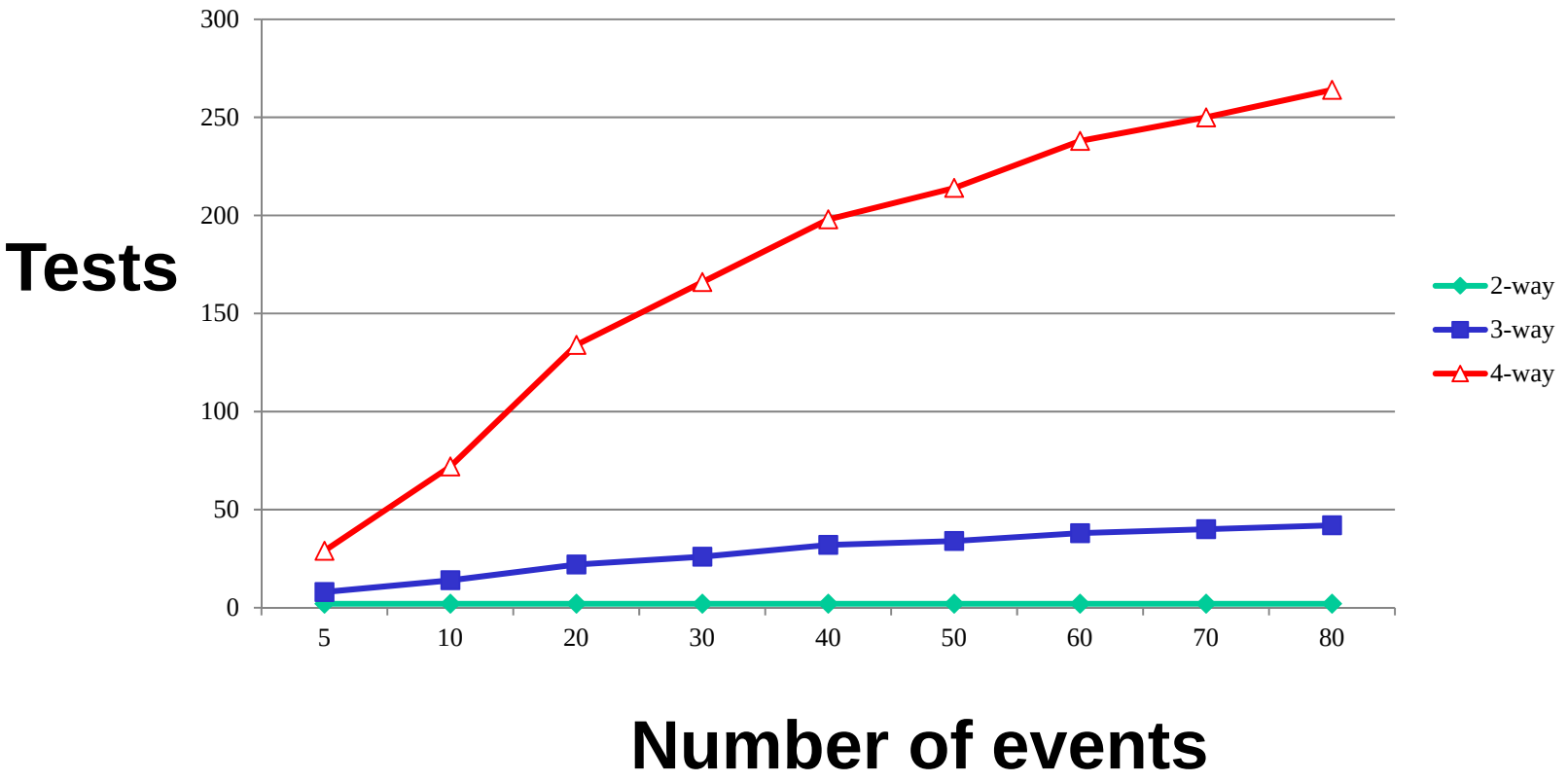
- With 6 events, all sequences = 6! = 720 tests
- Only 10 tests needed for all 3-way sequences, results even better for larger numbers of events
- Example: .*c.*f.*b.* covered. Any such 3-way seq covered.

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

NIST National Institute of Standards and Technology

Sequence Covering Array Properties

- 2-way sequences require only 2 tests (write in any order, reverse)
- For > 2-way, number of tests grows with log *n*, for *n* events
- Simple greedy algorithm produces compact test set
- Application not previously described in CS or math literature



Combinatorial Coverage

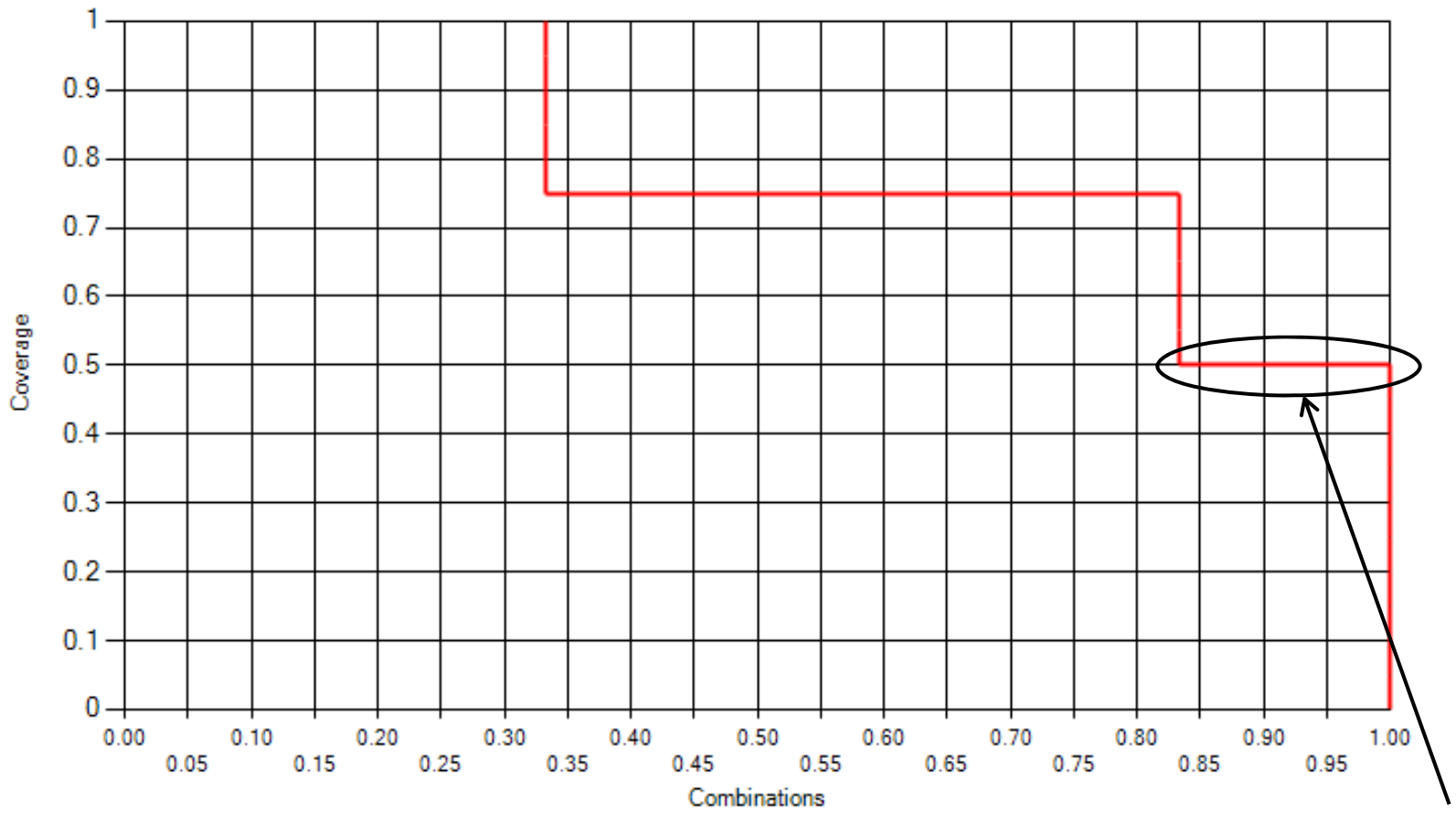
Tests	Variables			
	a	**b**	**c**	**d**
1	0	0	0	0
2	0	1	1	0
3	1	0	0	1
4	0	1	1	1

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



Coverage for file fig1.csv
Total 2-way = 0.792
Cov >= 0.00 = 6/6 = 1.00
Cov >= 0.05 = 6/6 = 1.00
Cov >= 0.10 = 6/6 = 1.00
Cov >= 0.15 = 6/6 = 1.00
Cov >= 0.20 = 6/6 = 1.00
Cov >= 0.25 = 6/6 = 1.00
Cov >= 0.30 = 6/6 = 1.00
Cov >= 0.35 = 6/6 = 1.00
Cov >= 0.40 = 6/6 = 1.00
Cov >= 0.45 = 6/6 = 1.00
Cov >= 0.50 = 6/6 = 1.00
Cov >= 0.55 = 5/6 = 0.83
Cov >= 0.60 = 5/6 = 0.83
Cov >= 0.65 = 5/6 = 0.83
Cov >= 0.70 = 5/6 = 0.83
Cov >= 0.75 = 5/6 = 0.83
Cov >= 0.80 = 2/6 = 0.33
Cov >= 0.85 = 2/6 = 0.33
Cov >= 0.90 = 2/6 = 0.33
Cov >= 0.95 = 2/6 = 0.33
Cov >= 1.00 = 2/6 = 0.33
2way
3way

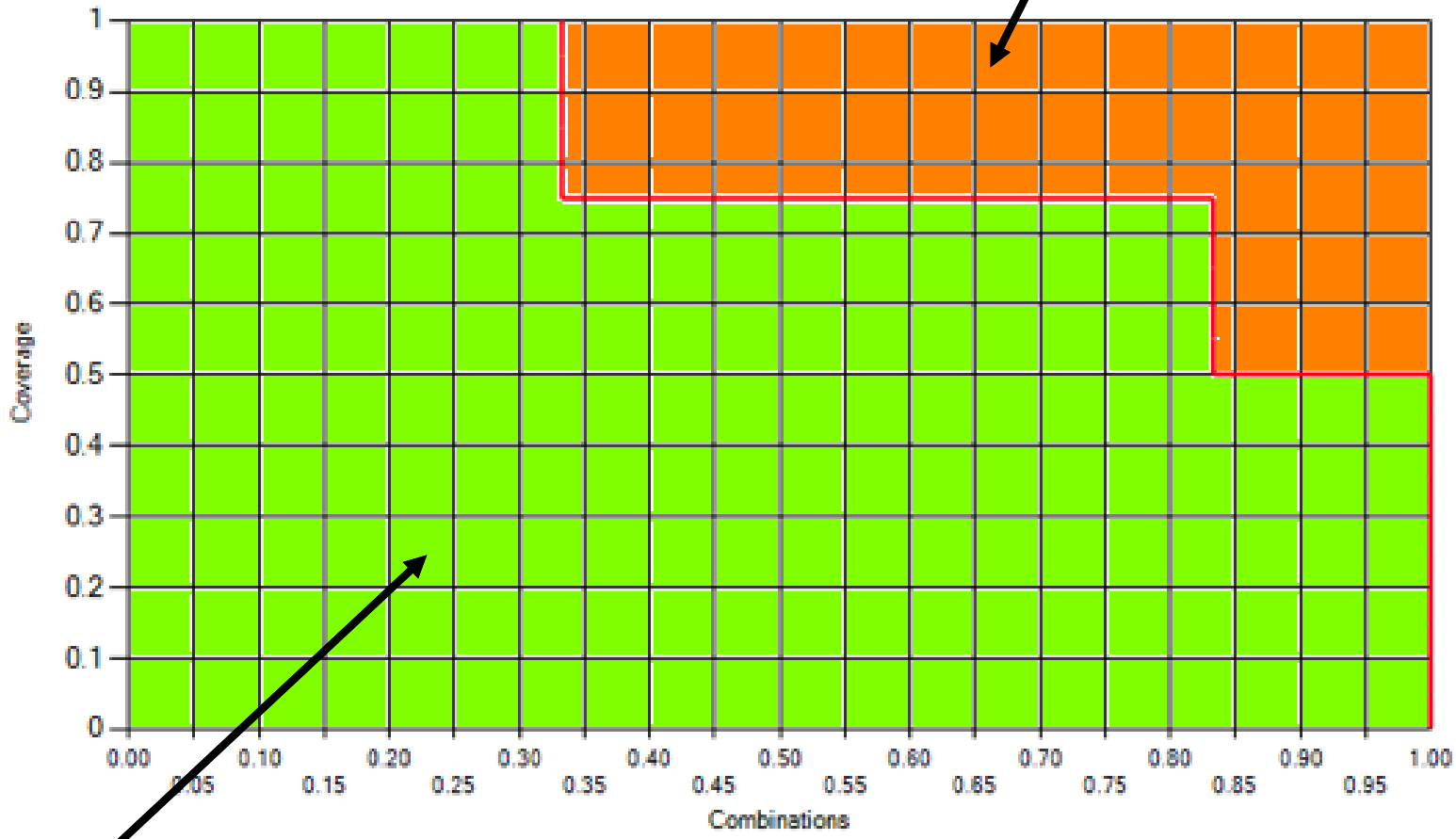
Bottom line:
All combinations covered to at least 50%

NIST
National Institute of Standards and Technology

What else does this chart show?

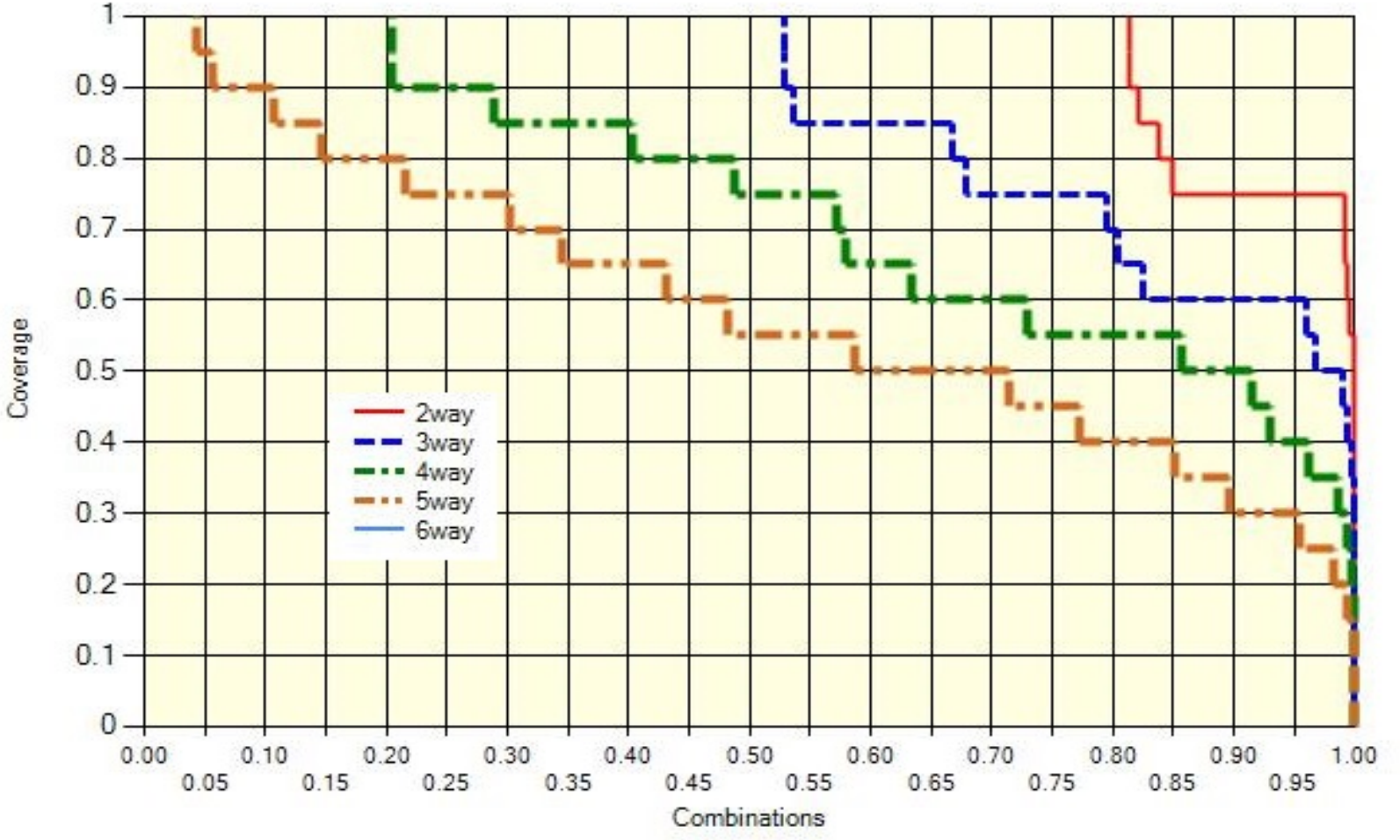
Untested combinations

(look for problems here)



Tested combinations

Spacecraft software example
82 variables, 7,489 tests, conventional test design (not covering arrays)



NIST
National Institute of Standards and Technology

Application to testing and assurance

- Measurable values with direct relevance to assurance
- Theorem relating (static) combinatorial coverage with (dynamic) code coverage
- To answer the question:

  How thorough is this test set?

  We can provide a defensible answer

Examples:

- Fuzz testing (random values) – good for finding bugs and security vulnerabilities, but how do you know you've done enough?
- Contract monitoring – How do you justify testing has been sufficient? Identify duplication of effort?

Oracle-free testing

Some current approaches:

- Fuzz testing – send random values until system fails, then analyze memory dump, execution traces
- Metamorphic testing – e.g. *cos*(*x*) = *cos*(*x*+360), so compare outputs for both, with a difference indicating an error.
- Partial test oracle – e.g., insert element x in data structure *S*, check $x \in S$

New method – using two-layer covering arrays

- requires only definition of equivalence classes
- we envision as part of the tool chain in development

Can this really work on practical code?

Experiment: TCAS code (standard set used to evaluate test methods)

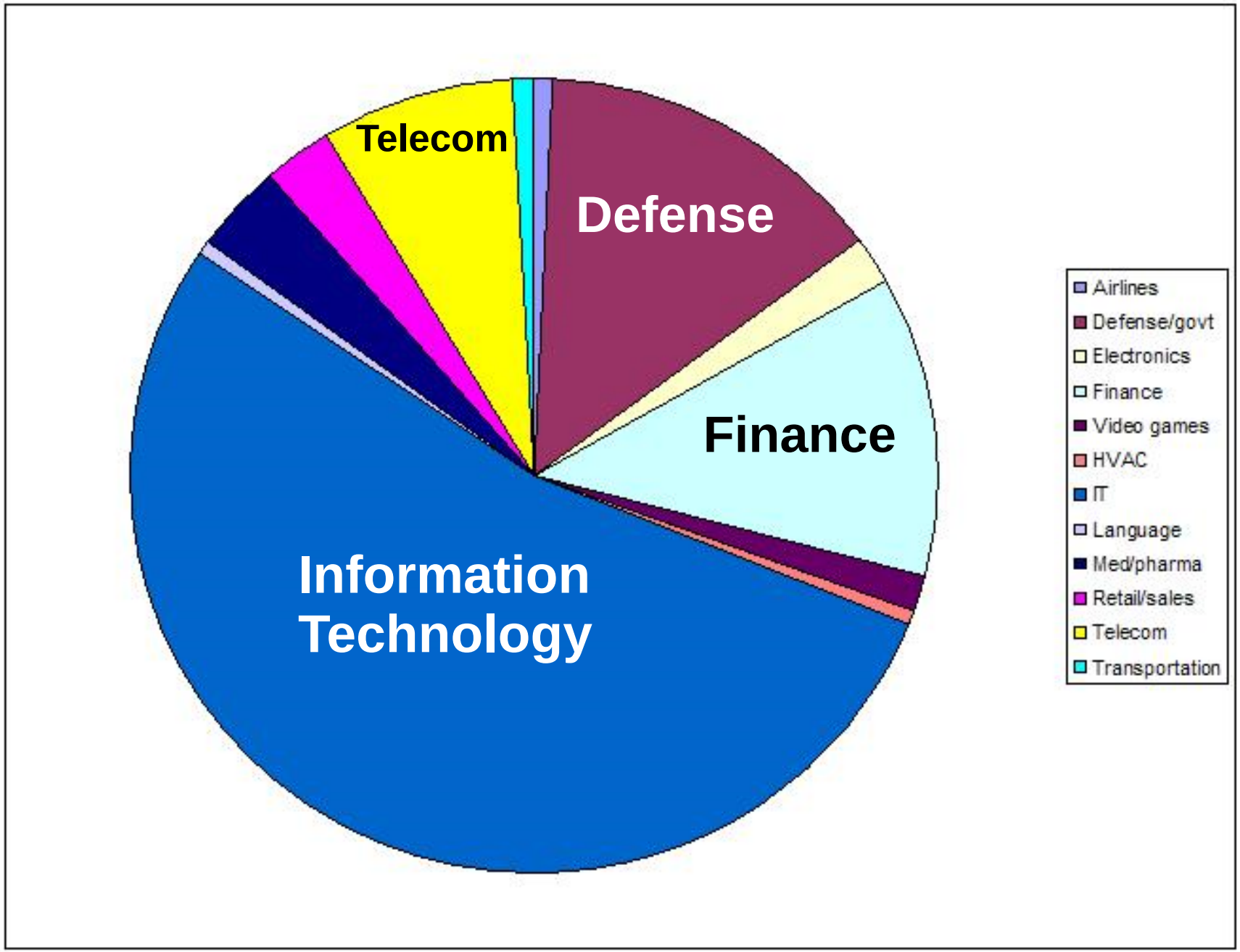
- Small C module, 12 variables
- Seeded faults in 41 variants

- Results:

Primary x secondary	#tests	total	faults detected
3-way x 3-way	285x8	2,280	6
4-way x 3-way	970x8	7,760	22

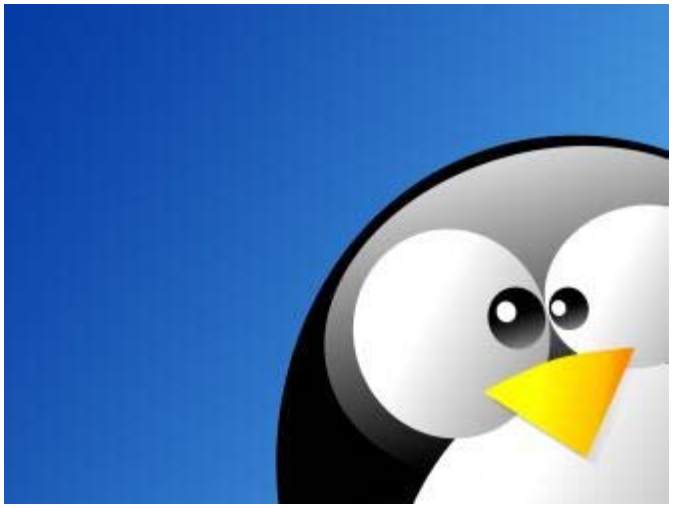
- More than half of faults detected
- Large number of tests -> but fully automated, no human intervention
- We envision this type of checking as part of the build process; can be used in parallel with static analysis, type checking

ACTS Users > 2,000 organizations



Bottom line: Significant cost savings and better testing shown in an extensive variety of application domains

Please contact us
if you're interested!



Rick Kuhn
kuhn@nist.gov

Raghu Kacker
raghu.kacker@nist.gov

http://csrc.nist.gov/acts

NIST
National Institute of Standards and Technology