Assured Autonomy
- problems and possible solutions -

Data Science and Artificial Intelligence Regulatory Applications Workshop

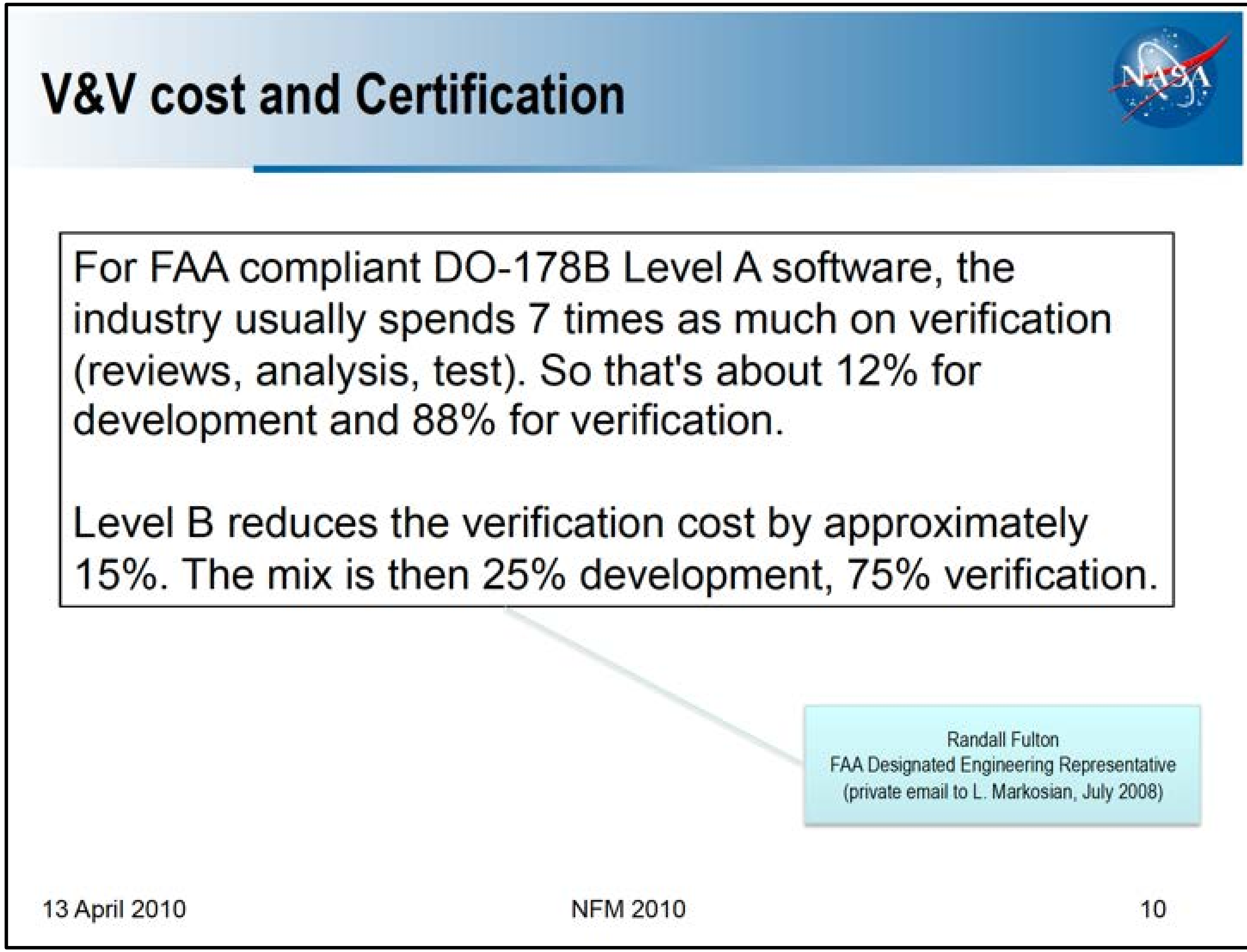
Rick Kuhn
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
kuhn@nist.gov

Software assurance is *very* expensive

Consumer level software cost: about 50% code development, 50% verification

For aviation life-critical,
12% code development,
<u>88% verification</u>
(Software is about 30% of cost for new civilian aircraft, higher for military)

Autonomy makes the problem even harder!



Why can't we use same processes as other safety-critical software ?

- Nearly all high assurance conventional software testing is based on structural coverage – ensuring that statements, decisions, paths are covered in testing
- Life-critical aviation software requires MCDC testing, white-box criterion that cannot be used for neural nets and other black-box methods

Coverage of input space can be measured

- Gold standard of assurance and verification of life-critical software can't be used for much of new life-critical autonomy software

- We can measure "neuron coverage", but indirect measure and not clear how closely related to accuracy and ability to correctly process all of the input space

- Measure the input space directly

- Then see if the AI system handles all of it correctly

Rare input combinations cause failures

- Multiple conditions involved in accidents
  - "The camera failed to recognize the white truck against a bright sky" (2 factors)

- "The sensors failed to pick up street signs, lane markings, and even pedestrians due to the angle of the car shifting in rain and the direction of the sun" (at least 3 factors)

- We need to understand what combinations of conditions are included in testing

Combinatorial value coverage - review

a	b	c	d
0	0	0	0
0	1	1	0
1	0	0	1
0	1	1	1

Vars	Combination values	Coverage
a b	00, 01, 10	.75
a c	00, 01, 10	.75
a d	00, 01, 11	.75
b c	00, 11	.50
b d	00, 01, 10, 11	1.0
c d	00, 01, 10, 11	1.0

19 combinations included in test set

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

Kuhn, D. R., Mendoza, I. D., Kacker, R. N., & Lei, Y. (2013). Combinatorial coverage measurement concepts and applications. *2013 IEEE Sixth Intl Conference on Software Testing, Verification and Validation Workshops*

Vars	Combination values	Coverage
a b	00, 01, 10	.75
a c	00, 01, 10	.75
a d	00, 01, 11	.75
b c	00, 11	.50
b d	00, 01, 10, 11	1.0
c d	00, 01, 10, 11	1.0

Total possible 2-way combinations = $2^2\binom{4}{2} = 24$

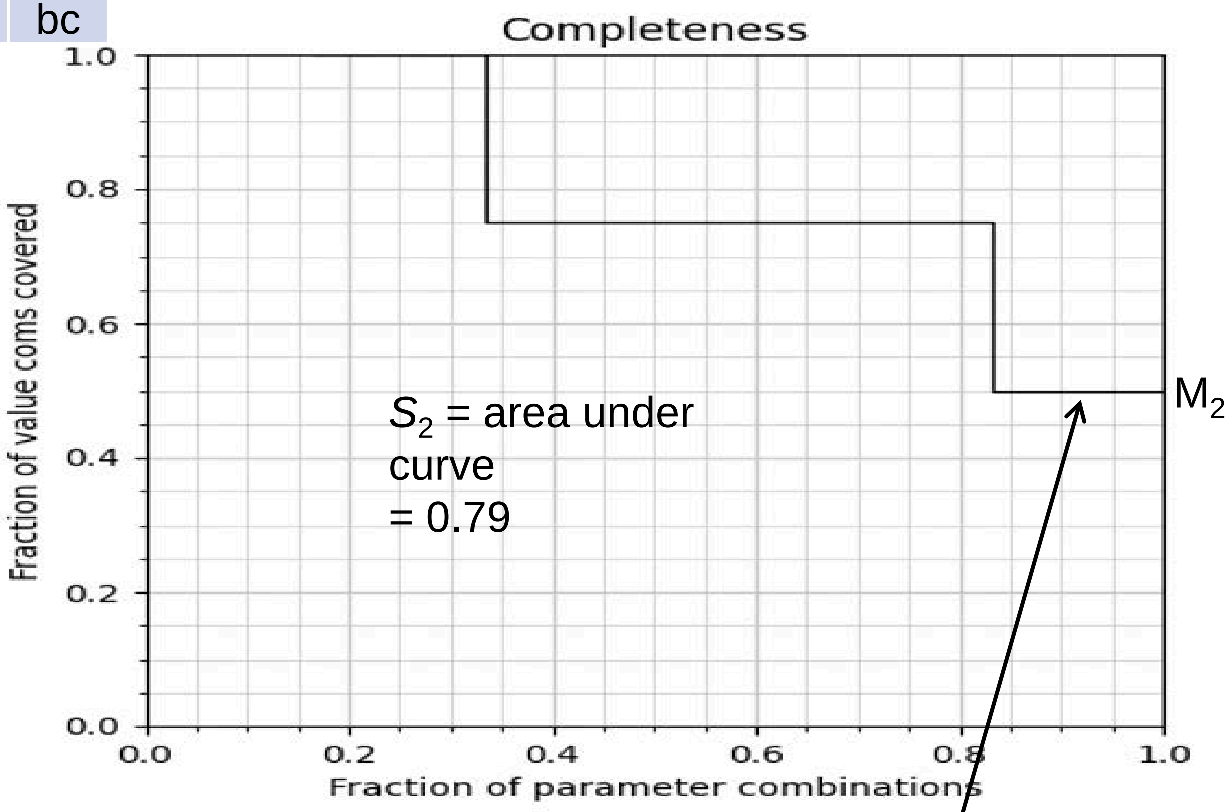
S_2 = fraction of 2-way combinations covered = 19/24 = 0.79

Rearranging the table:

1.00	00	00				
.75	01	01	00	00	00	
.50	10	10	01	01	01	00
.25	11	11	10	10	11	11
	bd	cd	ab	ac	ad	bc

Graphing Coverage Measurement

1.00	00	00				
.75	01	01	00	00	00	
.50	10	10	01	01	01	00
.25	11	11	10	10	11	11
	bd	cd	ab	ac	ad	bc



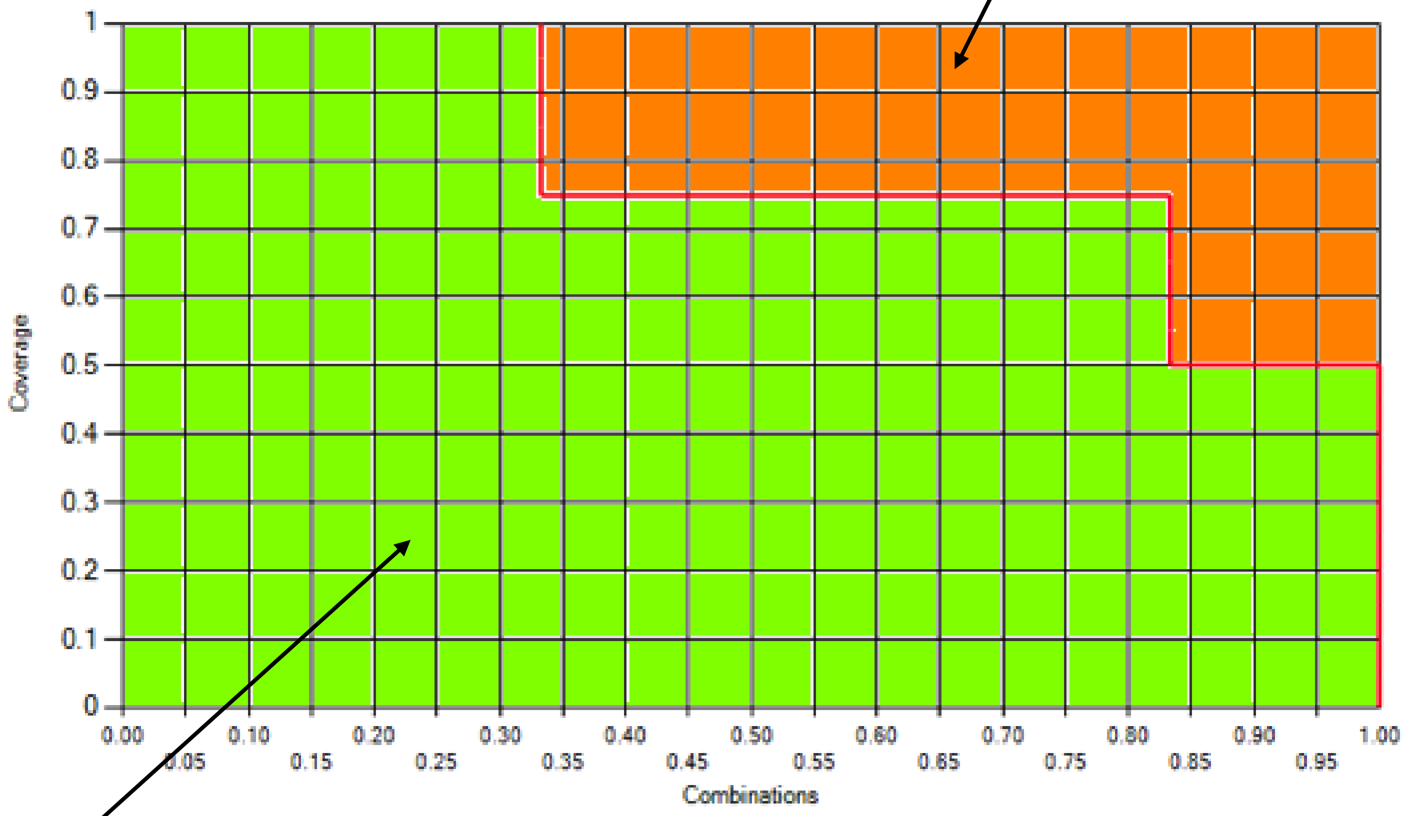
100% coverage of .33 of combinations
75% coverage of .50 of combinations
50% coverage of .16 of combinations

Bottom line:
All combinations covered to at least .50

What else does this chart show?

$1 - S_t$ = **Untested combinations**

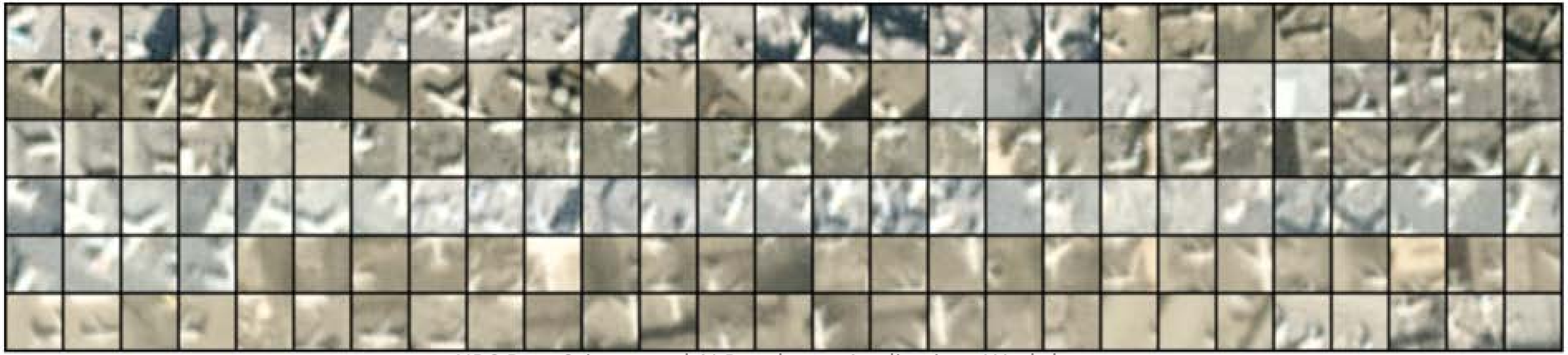
(look for problems here)



S_t = Tested combinations => code works for these

Transfer learning example – image analysis

- Planes in satellite imagery – Kaggle ML data set – determine if image <u>contains</u> or <u>does not contain</u> an airplane
- Two data sets – Southern California (SoCal, 21,151 images) or Northern California (NorCal, 10,849 images)
- 12 features, each discretized into 3 equal range bins



Transfer learning problem

- Train model on one set, apply to the other set
- Problem –
  - Model trained on larger, SoCal data applied to smaller, NorCal data → performance drop
  - Model trained on smaller, NorCal data applied to larger, SoCal data → NO performance drop
- This seems backwards!
- Isn’t it better to have more data?
- Can we explain this and predict it next time?

Density of combinations in one but not the other data set, 2-way

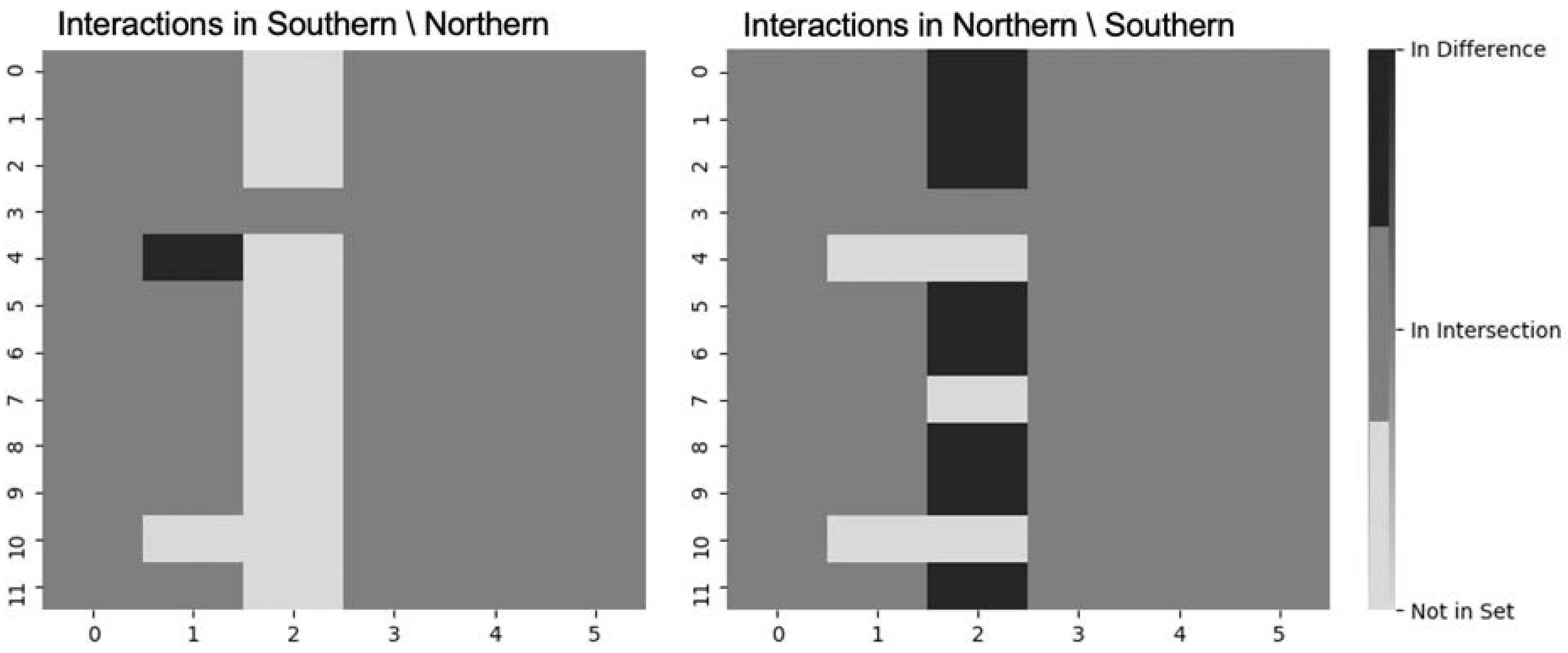


Image from Combinatorial Testing Metrics for Machine Learning, Lanus, Freeman, Kuhn, Kacker, IWCT 2021

For C = SoCal, N = NorCal,
|C\N| / |C| = 0.02
|N\C| / |N| = 0.12

The NorCal data set has fewer "never seen" combinations, even with half as many observations.

Critical for assurance

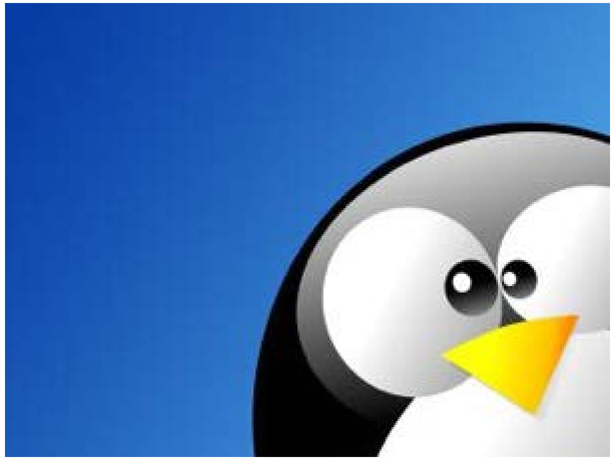
Summary – Transfer learning

- Current approaches to estimating success for transfer learning are largely ad-hoc and not highly effective
- Combinatorial methods show promise for improvements – <u>measurable quantities directly related to determining if one data set is representative of the field of application</u>
- Empirical studies planned
- Broader application for autonomous system assurance

Assured autonomy – key points & current state

- For capability and cost reasons, <u>autonomous components are becoming routine</u> in software engineering
- <u>Essential methods for high assurance in conventional systems do not apply</u> to many autonomous components
  - Structural coverage – not for neural nets, and others
  - Formal proofs – for some parts but limited
- Measures of test adequacy must consider coverage of input combinations and sequences
- Desirable assurance properties can be shown using these measures

Please contact us if you're interested!



Rick Kuhn, Raghu Kacker, M.S. Raunak
{kuhn, raghu.kacker, raunak}@nist.gov

http://csrc.nist.gov/acts