How Can We Provide Assured Autonomy?

Fordham University, New York City   January 9-10, 2024

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

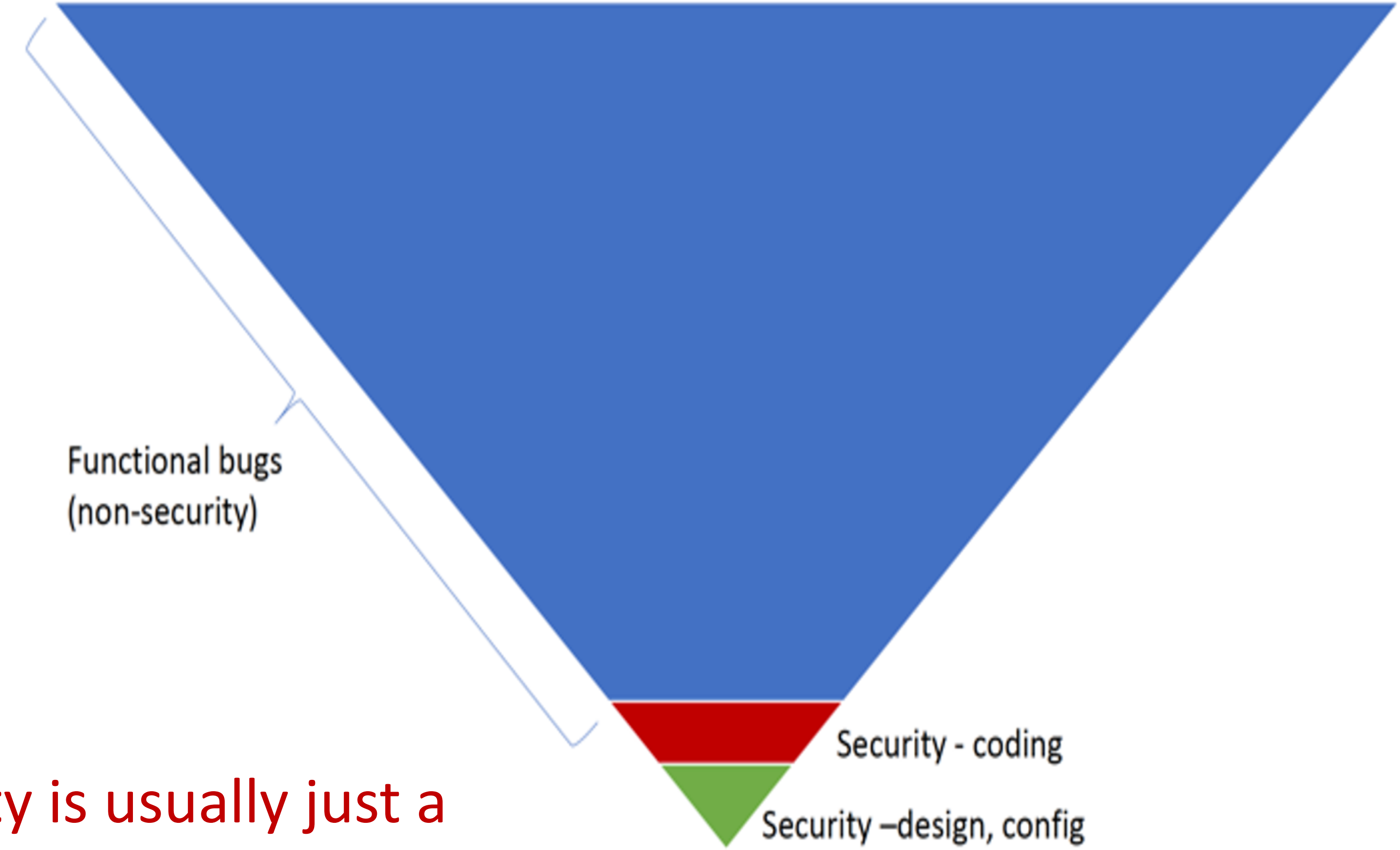
What is the problem, and why is it hard?

Autonomous vehicles must be *safe* and *secure*

Current testing and assurance methods are not suitable

Security flaws are generally a small subset of s/w flaws

Remember: a vulnerability is usually just a bug that can be used to use to defeat security properties



Outline

- Why current critical system testing processes are not suitable
- Assurance based on input space coverage,
- Transfer learning

Some problems in assured autonomy,
and potential solutions

What is NIST and why are we doing this?

- US Government agency, which supports US industry through developing better measurement and test methods
- 3,000 scientists, engineers, and staff including 4 Nobel laureates
- Broad involvement with industry and academia

What are interaction faults?

- 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

pressure < 10 & volume > 300

(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?

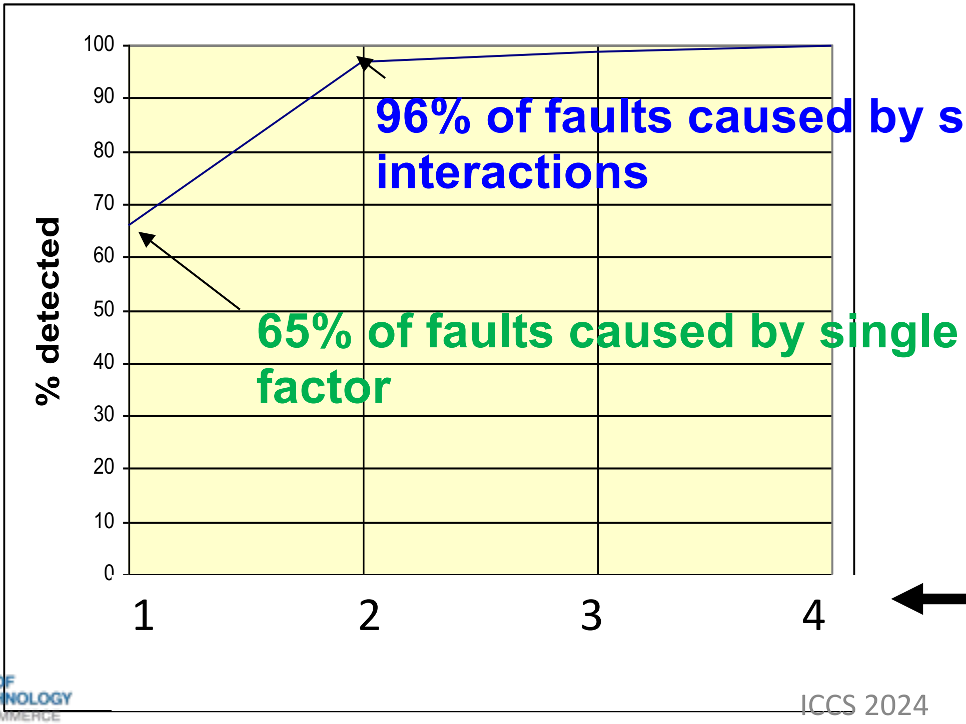
- Interactions   e.g., failure occurs if

  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



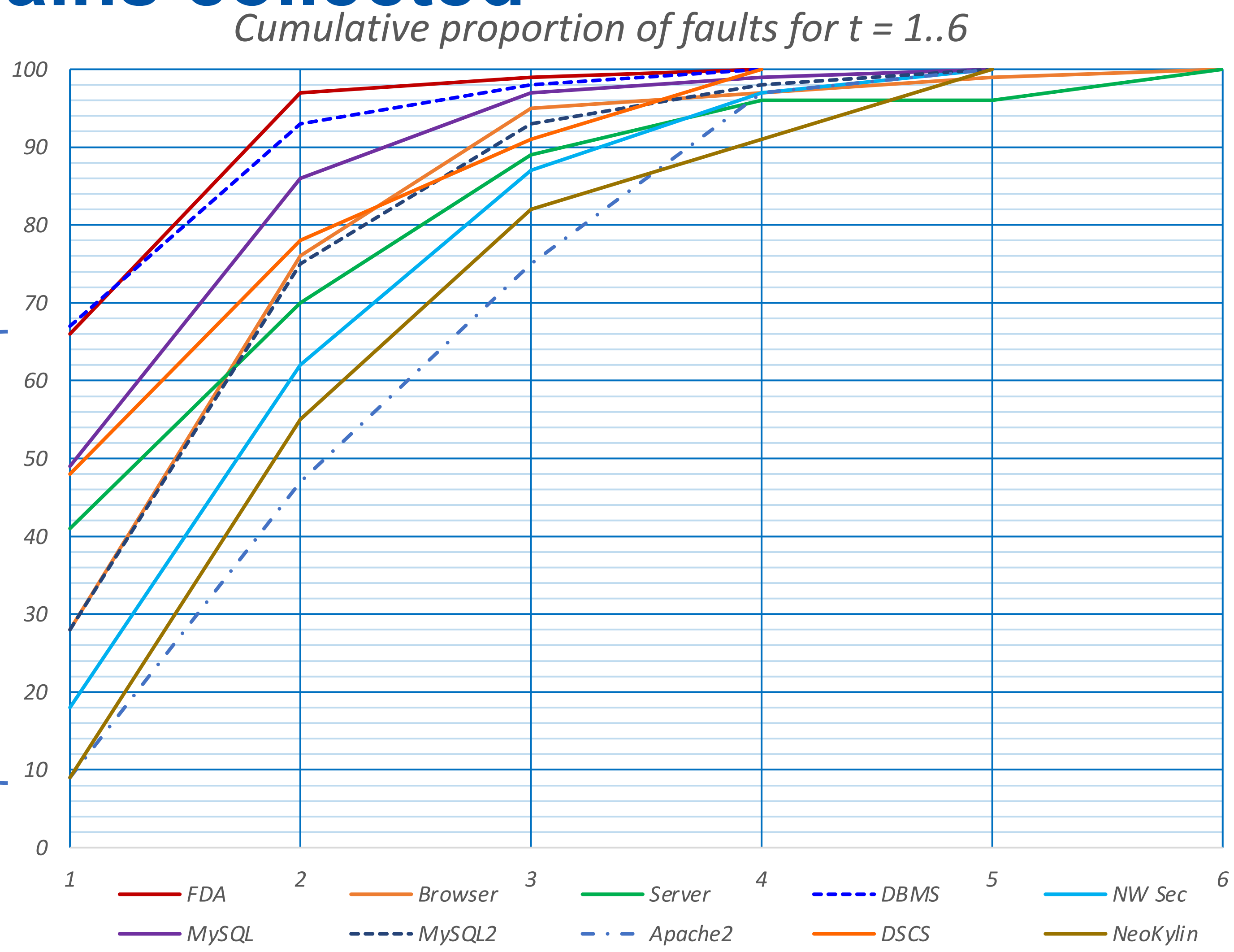
Number of factors involved

Various domains collected

Wide variation in percent of failures caused by single factor

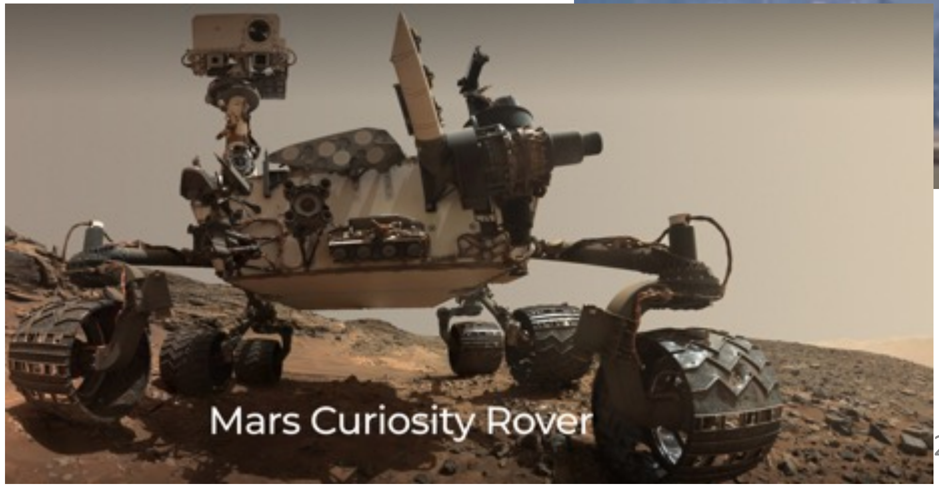
Variability decreases as number of factors increases

More testing or users
=> harder to find errors, fewer single factor failures



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

How is all this related to autonomous systems?



(Slide from Darryl Ahner, US Air Force Institute of Technology)

Defense Science Board Study

STAT T&E COE: *Scientia Prudentia et Valor*

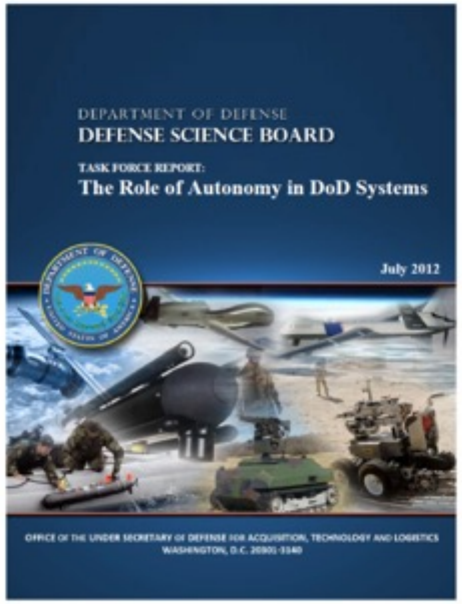
DSB 2012 The Role of Autonomy in DoD Systems Study recommends:

"USD(AT&L) to create developmental and operational T&E techniques that focus on the unique challenges of autonomy (to include developing operational training techniques that explicitly build trust in autonomous systems)."

Recommendation:

USD(AT&L) establish developmental and operational T&E techniques that focus on the unique challenges of autonomy

- Coping with the difficulty of enumerating all conditions and non-deterministic responses
- Basis for system decisions often not apparent to user
- Measuring trust that the autonomous system will interact with its human supervisor as intended
- Leverage the benefits of robust simulation



AFIT

National Institute of Standards and Technology

Software assurance is already *very* expensive

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

For aviation life-critical,
12% code development,
88% testing and verification

Autonomy makes the problem even harder!

V&V cost and Certification

For FAA compliant DO-178B Level A software, the industry usually spends 7 times as much on verification (reviews, analysis, test). So that's about 12% for development and 88% for verification.

Level B reduces the verification cost by approximately 15%. The mix is then 25% development, 75% verification.

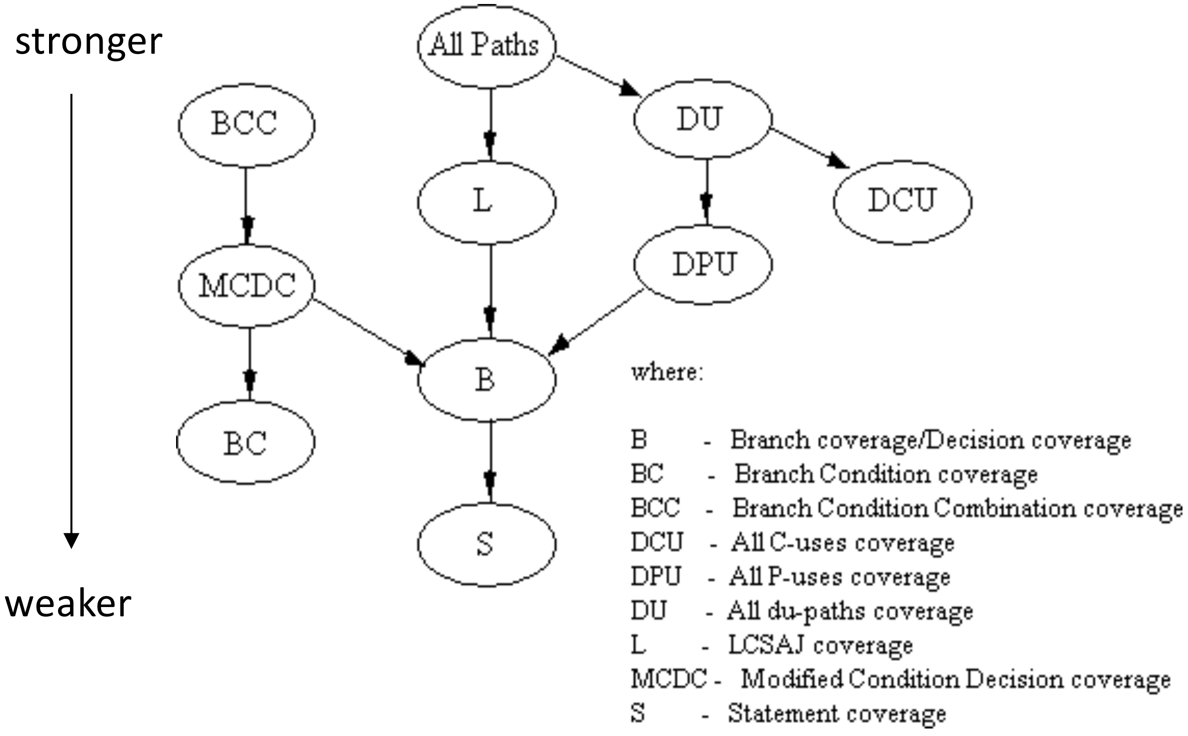
Randall Fulton
FAA Designated Engineering Representative
(private email to L. Markosian, July 2008)

13 April 2010 NFM 2010

Why can't we use same processes as other high assurance software ?

- Conventional critical software testing is based on structural coverage – ensuring that conditions, decisions, paths are covered in testing
- Life-critical aviation software requires MCDC testing, white-box criterion that doesn't fit neural nets and other black-box methods where input is what matters

Code coverage works well - for conventional software



Subsumption relationships of structural coverage criteria

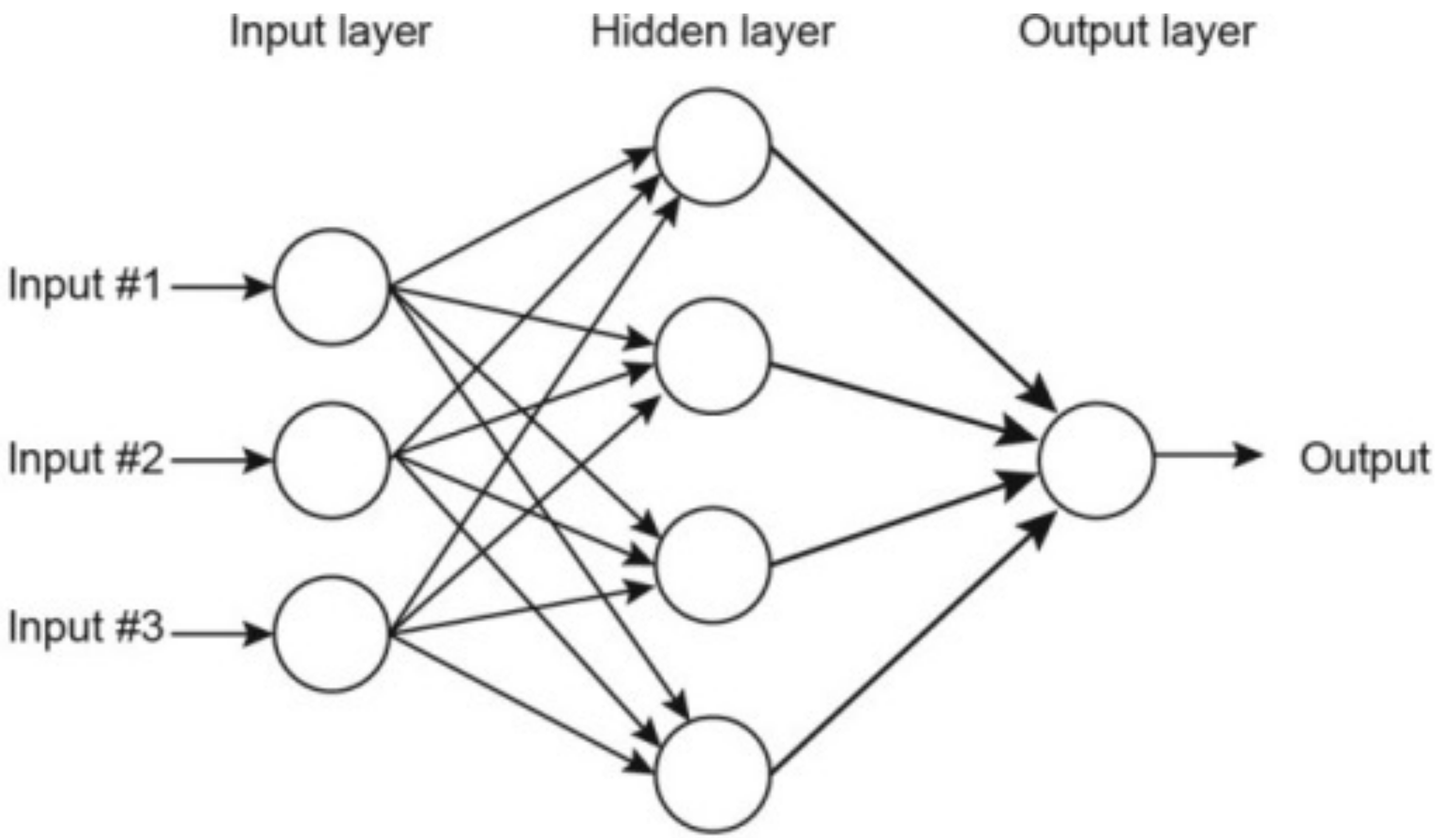
- Test coverage has traditionally been defined using graph-based structural coverage criteria:
  - statement (weak)
  - branch (better)
  - etc.
- Based on paths through the code

- We may have perfect structural coverage of code, but what does that tell us about response to rare inputs?

- What if the code is always the same, and only the inputs matter?

Can we use code coverage for machine learning?

- Much of AI/ML depends on various neural nets
- Algorithm and code stays the same
- Connections and weights vary
- Behavior changes depending on inputs used in training



NIST
National Institute of Standards and Technology

Input space coverage is needed

- Gold standard of assurance and verification of life-critical software is not suitable 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

Outline

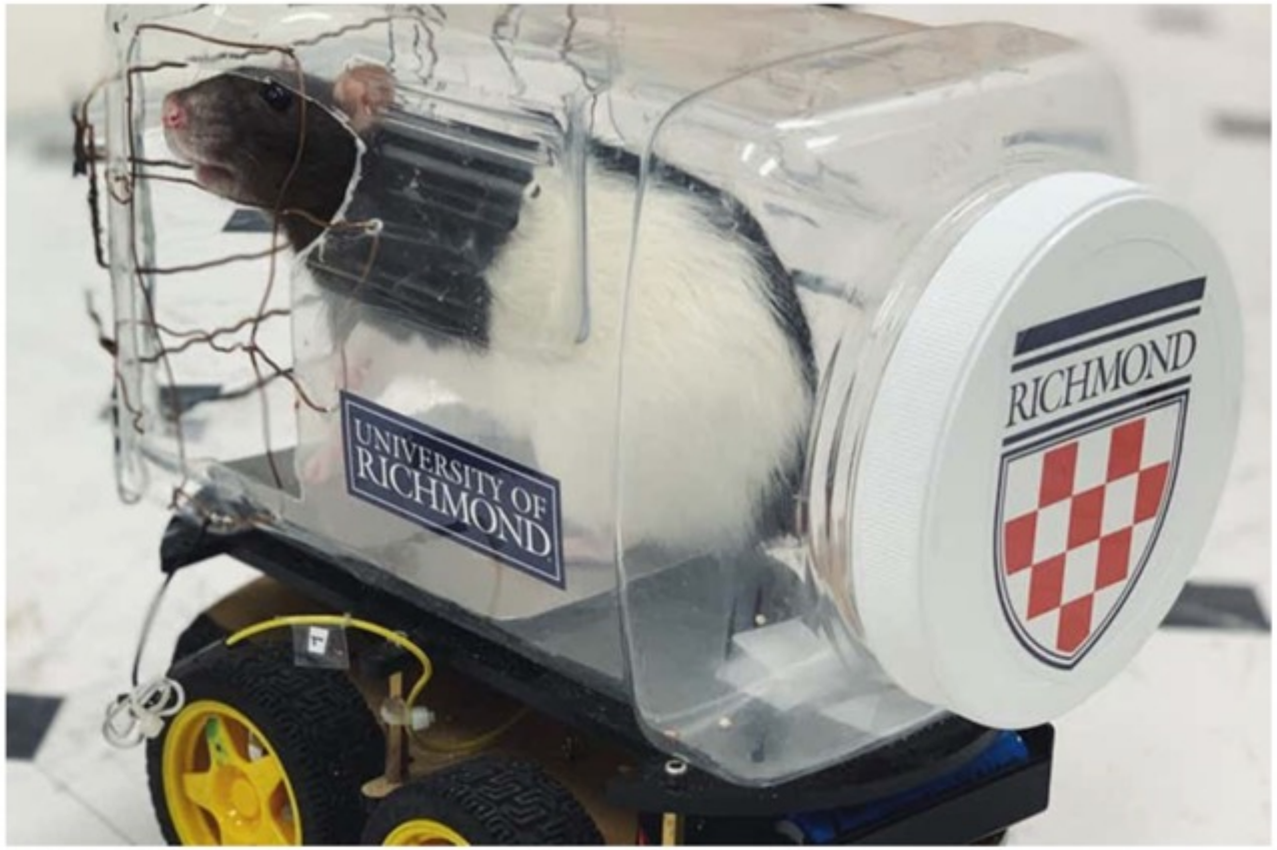
- Why current critical system testing processes are not suitable
- Assurance based on input space coverage
- Transfer learning

NewScientist

Scientists have trained rats to drive tiny cars to collect food

LIFE 22 October 2019

By Alice Klein



It doesn’t take much intelligence to drive a car. Even rats can do it!

But can they do it under all kinds of conditions ?

The problem is harder outside of a constrained environment

Things get tricky as the scene becomes complex

- Multiple conditions involved in accidents
  - "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" (3 factors)
  - "… the *white side* of the tractor trailer against a *brightly lit sky*, so the brake was not applied. The *high ride height* of the trailer combined with its *positioning across the road* …" (4 factors)

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

NIST
National Institute of Standards and Technology

How can we measure interaction fault detection capability?

rows of input

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

19 combinations included in test set

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

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

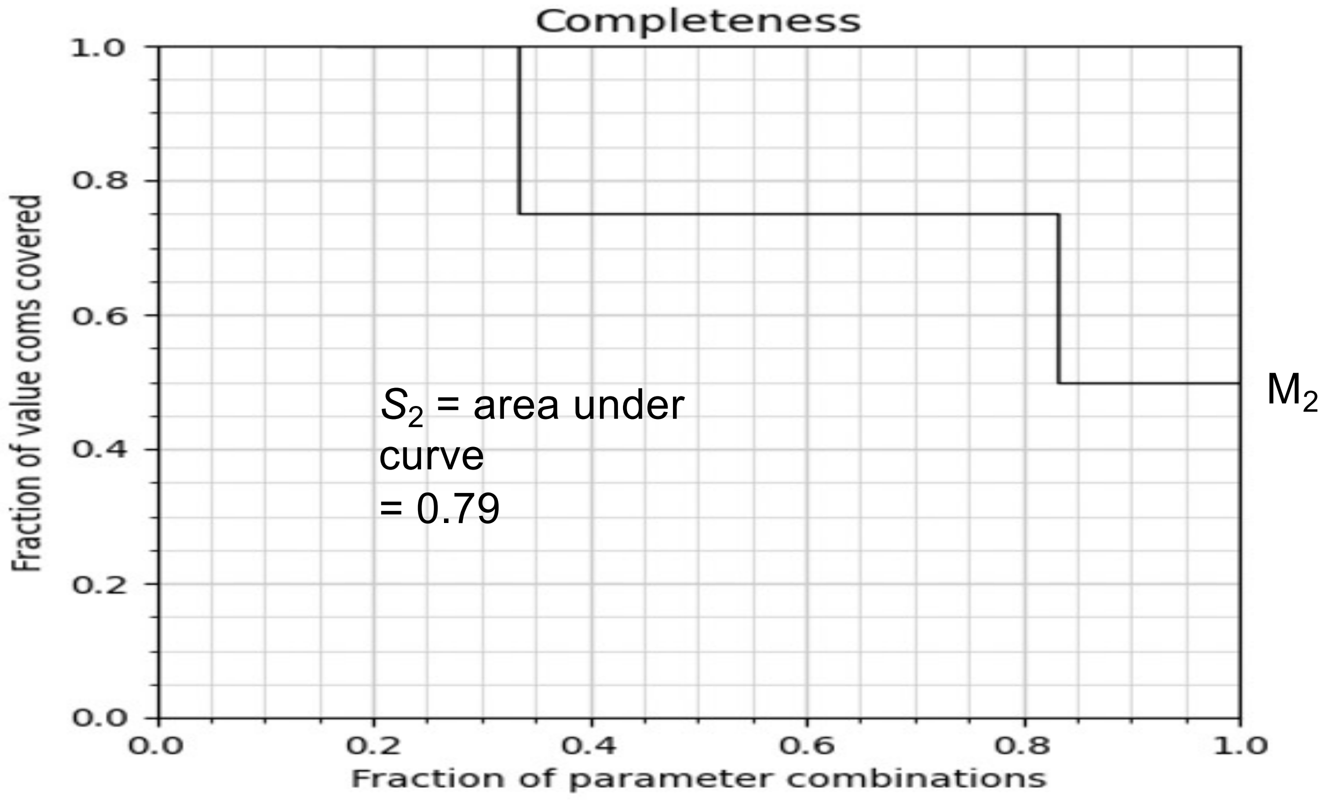
Rearranging the table:

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

NIST National Institute of Standards and Technology

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

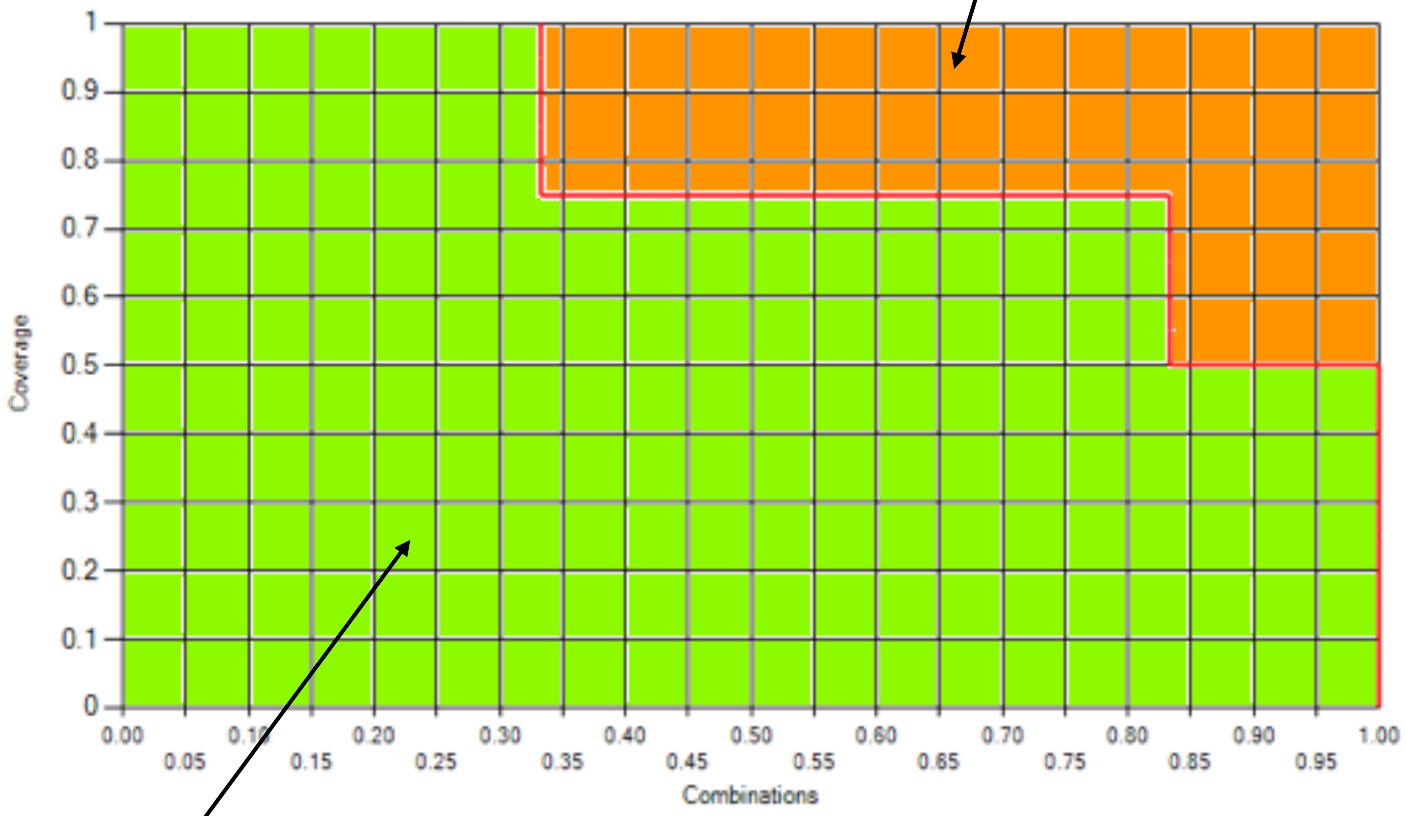
Bottom line:
All combinations covered to at least .50

National Institute of Standards and Technology

What else does this chart show?

$1 - S_t$ = **Missing combinations**

(look for problems here)

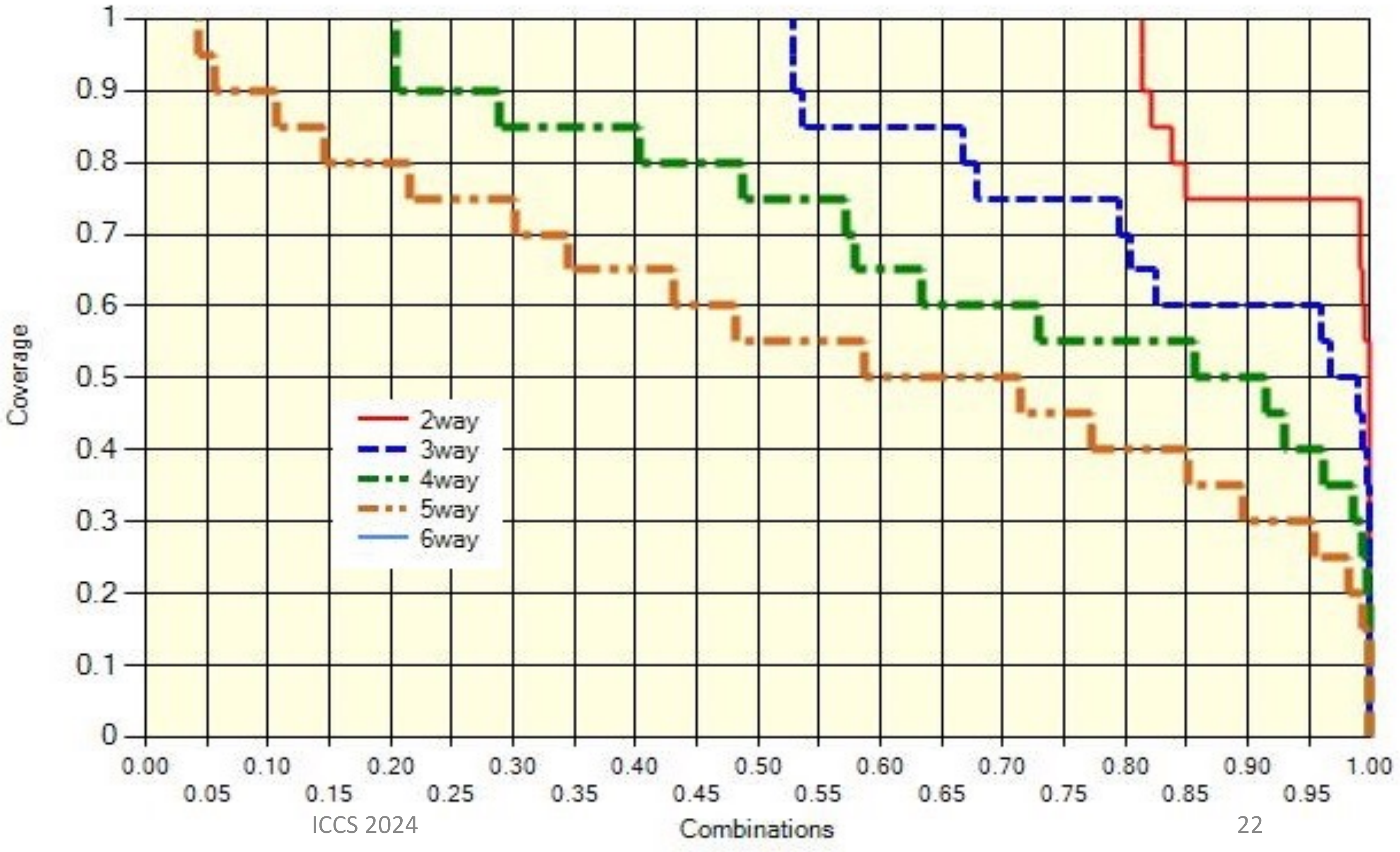


S_t = combinations in training and testing => model works for these

National Institute of Standards and Technology

How much combinatorial coverage is achieved with conventional tests?

<u>Spacecraft software example</u>
- 82 variables,
- 7,489 tests,
- conventional test design



Outline

- Why current critical system testing processes are not suitable
- Assurance based on input space coverage
- Transfer learning – example application

Transfer learning – what is the problem?

- Differences inevitably exist between training data sets, test data sets, and later real-world data
- Further differences exist between data from two or more different environments

- How do we predict performance of a model trained on one data set when applied to another?
  - New environment
  - Changed environment
  - Additional possible values, etc.

Lanus, E., Freeman, L. J., Kuhn, D. R., & Kacker, R. N. (2021, April). Combinatorial Testing Metrics for Machine Learning. In *2021 IEEE Intl Conference on Software Testing, Verification and Validation Workshops (ICSTW)*

Transfer learning – conventional practice

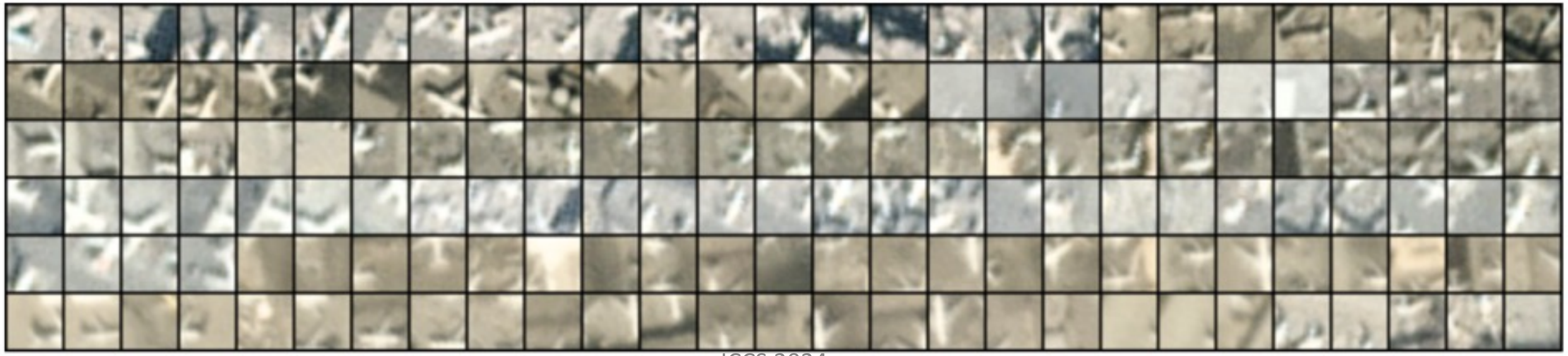
Randomized selection – but how much random data will be sufficient, especially with smaller data sets?

Ensure at least one of each object type – but this may not be representative of object attribute distributions

Interactions are critical to consider in most ML problems, especially for safety, but conventional practice does little to ensure data sets are adequately representative of interactions

Example – image analysis

- Planes in satellite imagery – Kaggle ML data set – determine if image contains or does not contain 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



NIST National Institute of Standards and Technology

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 measure, explain and predict it next time?

Density of combinations in one versus the other data set, 2-way

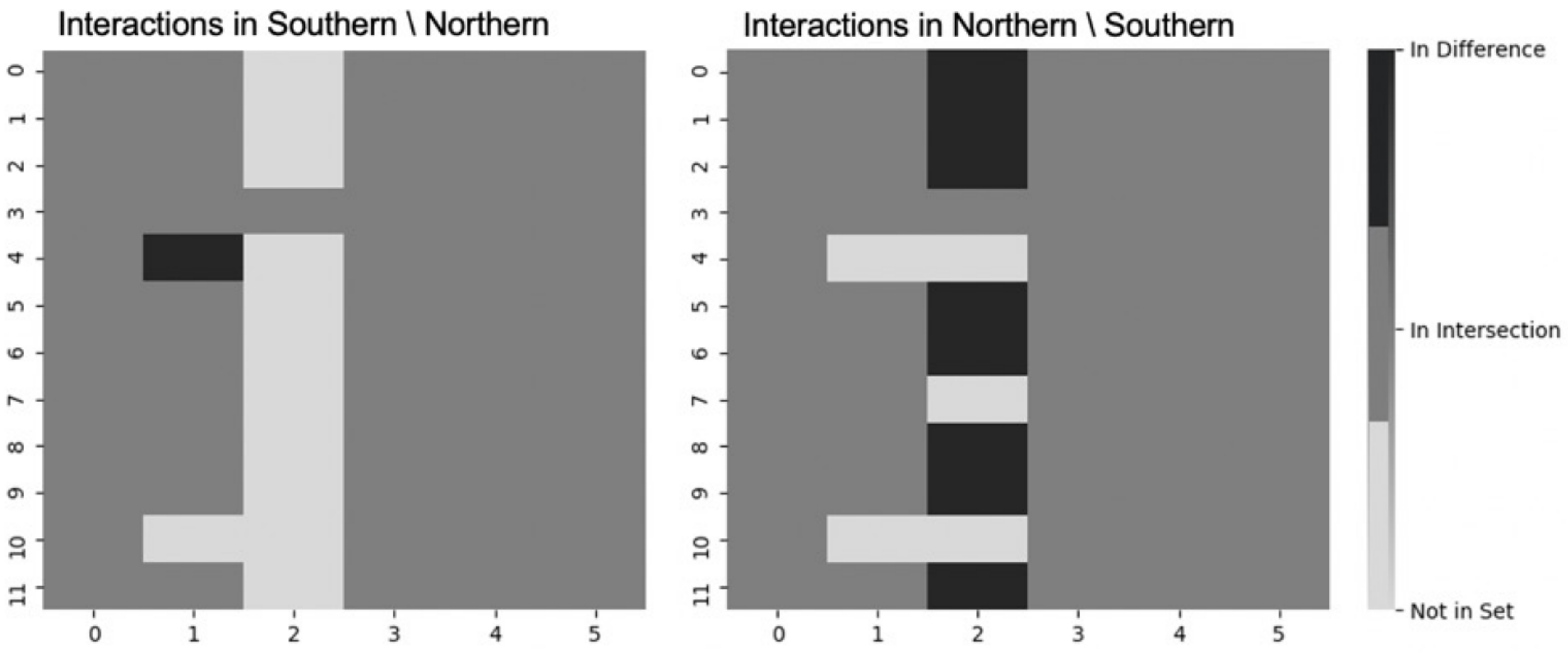


Image from Combinatorial Testing Metrics for Machine Learning, Lanus, Freeman, Kuhn, Kacker, IWCT 2021

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

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 – measurable quantities directly related to determining if one data set is representative of the field of application
- Much additional work is needed to evaluate this idea, and to understand the link between combinatorial difference values and prediction accuracy
- Empirical studies planned

Assured autonomy – more questions than answers

- Interactions of learning components with programmed components – especially replacing humans

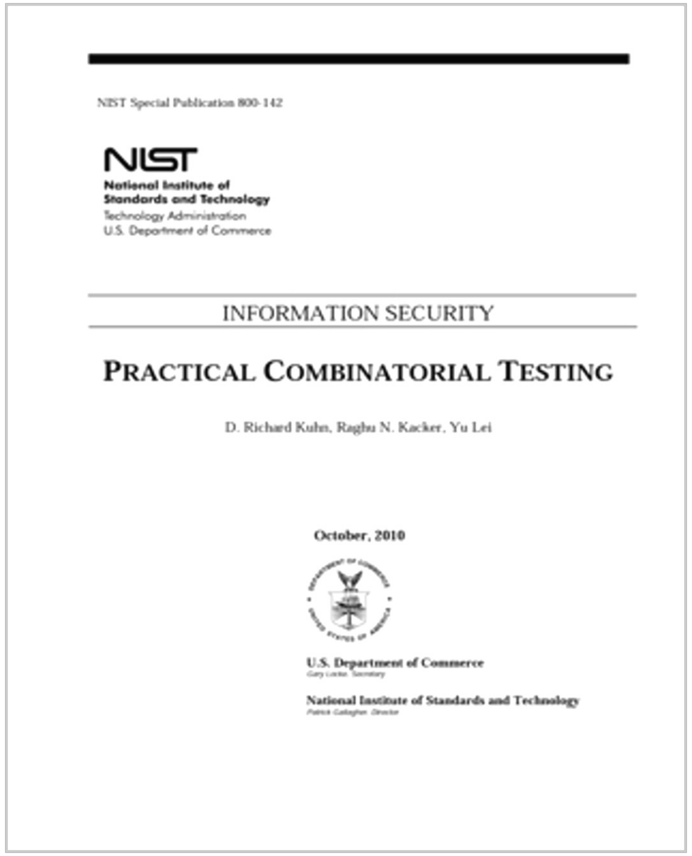
- Changes the nature of system failures
- More like failures involving human factors issues? ☺ Turing test for bugs! Distinguish between human-triggered and AI-triggered system failures?

Assured autonomy – key points & current state

- For capability and cost reasons, autonomous components are becoming routine in software engineering
- Many, or most, methods used in high assurance conventional systems are not sufficient for many autonomous components
  - Structural coverage – not for neural nets, and others
  - Formal proofs – for some parts but limited
- How to deal with learning, dynamic changes in system?
- Understanding and measuring interaction coverage is necessary

Learning and Applying Combinatorial Methods

- Self-contained tutorial on using combinatorial testing for real-world software
- Advanced topics such as the use of formal models and test oracle generation
- Costs and practical considerations
- Designed for testers or undergraduate students of computer science or engineering


NIST Special Publication 800-142

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

INFORMATION SECURITY

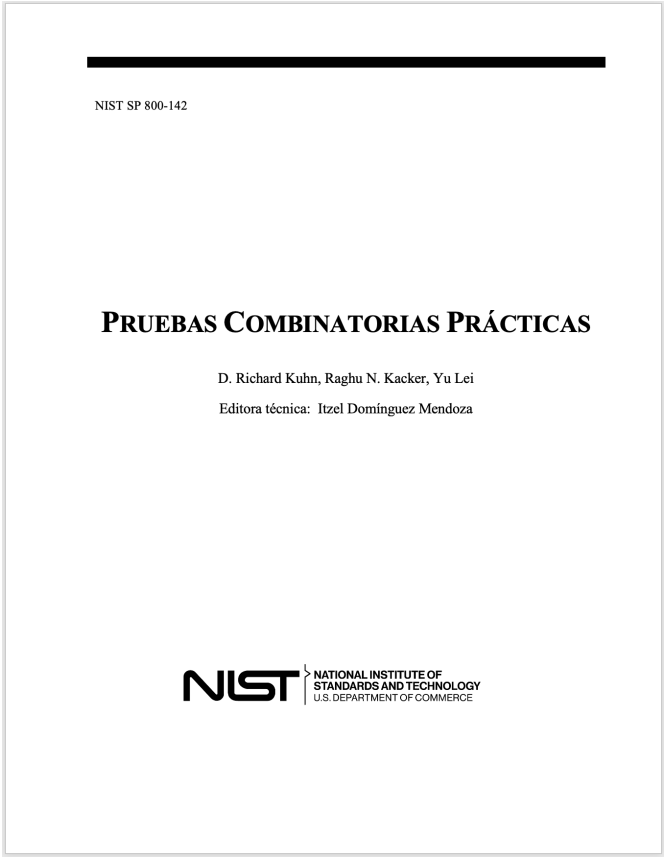
PRACTICAL COMBINATORIAL TESTING

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

October, 2010

U.S. Department of Commerce

National Institute of Standards and Technology


NIST SP 800-142

PRUEBAS COMBINATORIAS PRÁCTICAS

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

Editora técnica: Itzel Domínguez Mendoza

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

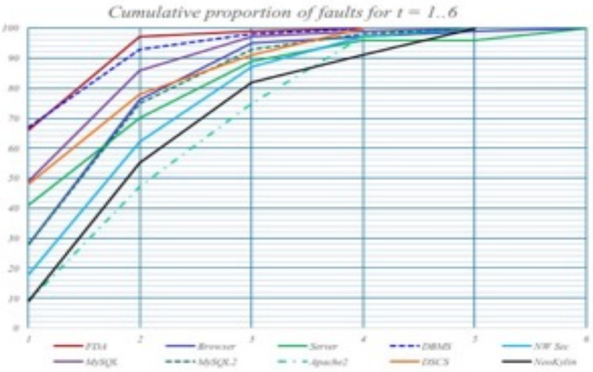
http://csrc.nist.gov/acts

Automated Combinatorial Testing for Software ACTS

Overview

Combinatorial methods can reduce costs for software testing, and *have significant applications in software engineering:*

- **Combinatorial or *t*-way testing** is a proven method for more effective testing at lower cost. The key insight underlying its effectiveness resulted from a series of studies by NIST from 1999 to 2004. NIST research showed that most software bugs and failures are caused by one or two parameters, with progressively fewer by three or more, which means that combinatorial testing can provide more efficient fault detection than conventional methods. Multiple studies have shown fault detection equal to exhaustive testing with a 20X to 700X reduction in test set size. New algorithms compressing combinations into a small number of tests have made this method practical for industrial use, providing better testing at lower cost. See articles on high assurance software testing or security and reliability.



- **Autonomous systems assurance:** Input space coverage measurements are needed in life-critical assurance and verification of autonomous systems, because current methods for assurance of safety critical systems rely on measures of structural coverage, which do not apply to many autonomous systems. Combinatorial methods, including a theorem relating measures of input space coverage, offer a better approach for autonomous system verification.
- **Metrology* for software engineering.** Sound engineering requires adequate measurement and analysis. Structural coverage enables formally defined criteria for test completeness, but even full coverage may miss faults related to rare inputs. Combinatorial methods open new possibilities for metrology in software engineering, providing a more scientific approach to assurance and verification.

  *Metrology is the science of measurement (NIST is the US national metrology institute)

***NEW:* Combinatorial Coverage Difference Measurement for assurance of autonomous systems and**

PROJECT LINKS

Overview

FAQs

ADDITIONAL PAGES

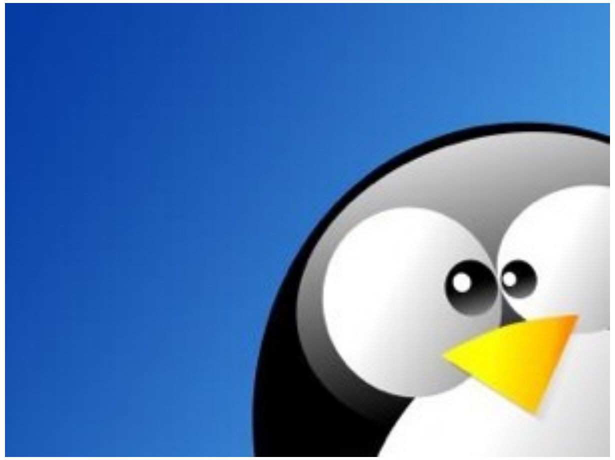
- **Quick start**
- **Downloadable Tools**
- **Combinatorial Methods in Testing**
  - Why do Combinatorial Testing?
  - Event Sequence Testing
  - Oracle-free Testing and Test Automation
  - Case Studies
- **Input Space Measurement**
  - Why Measure Input Space?
  - Case studies
- **Assured autonomy**
  - Explainable AI, Verification, and Validation
  - Rule-based Expert Systems and Formal Methods
  - Case studies
- **Cybersecurity Testing**
  - Combinatorial approach
  - Magic mirror vulnerability testing tool
  - Case studies
- **Software Testing Methodology**
  - NIST Testing Process
  - DOs and DON'Ts of testing
- **ACTS Library**
  - Fundamental background papers
  - Papers on combinatorial test methods
  - Covering Array Library

National Institute of Standards and Technology

Freely Available Tools

- **Covering array generator** – basic tool for test input or configurations;
- **Combinatorial coverage measurement** – detailed analysis of combination coverage; automated generation of supplemental tests; helpful for integrating c/t with existing test methods
- **Sequence covering array generator** – new concept; applies combinatorial methods to event sequence testing
- **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

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