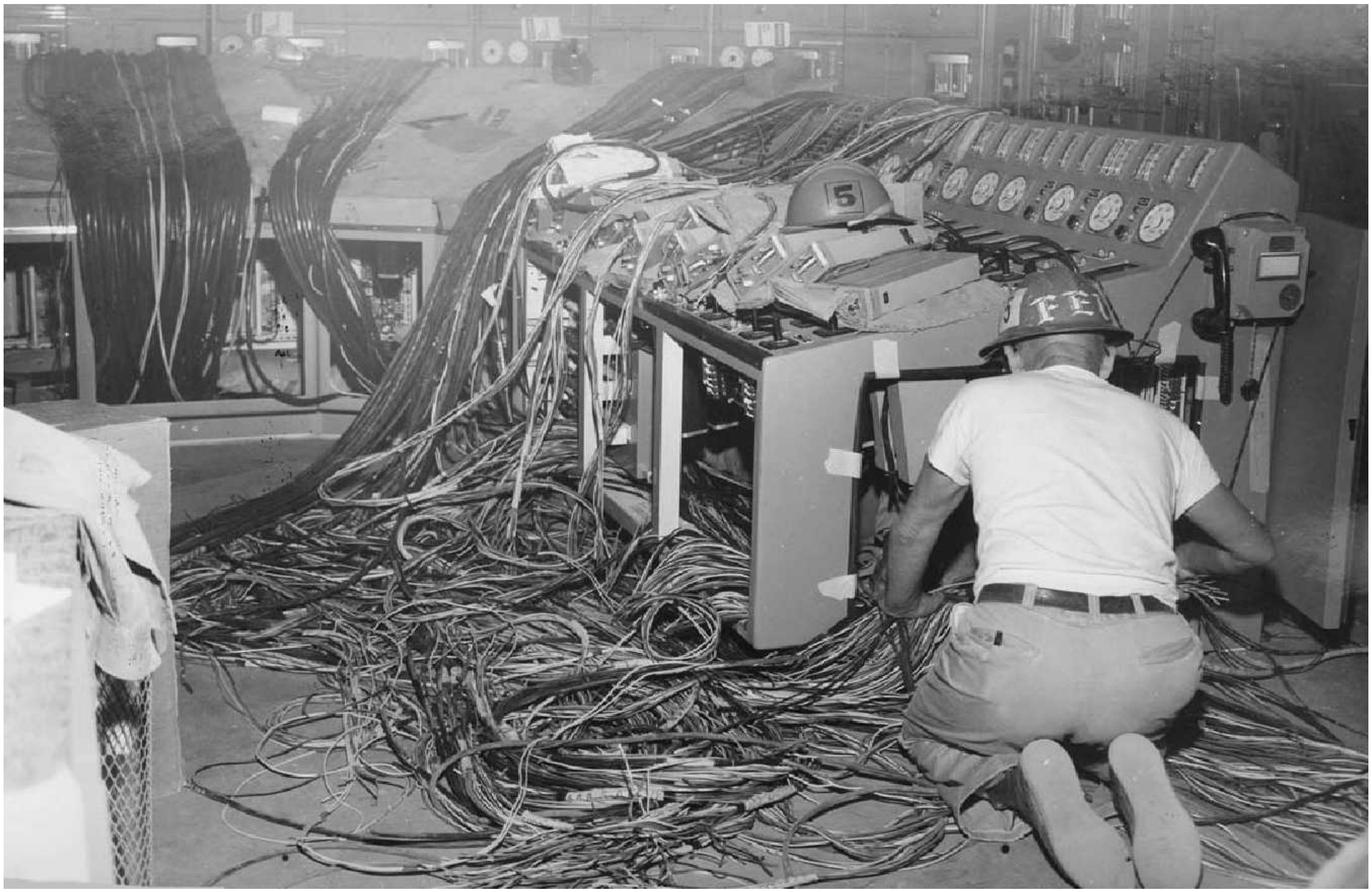
View on Software Conformance Testing

Paul E. Black

paul.black@nist.gov

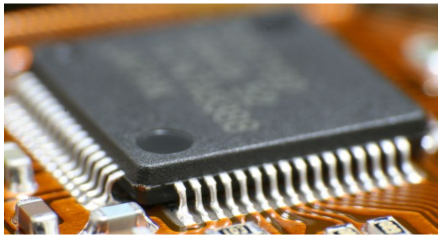
http://samate.nist.gov/

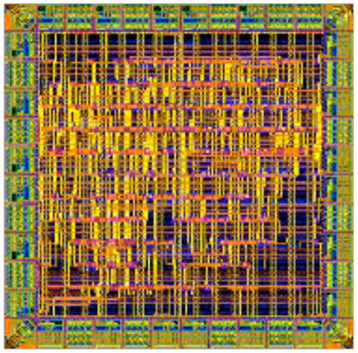
Automation doesn't make the complexity go away …



Automation just hides it in micron-sized spots.

Bad Character





- **Application: IC design parser**
  - **Input: computer chip design, 2D, WYSIWYG**
  - **Output: network list, plain text**
- **Failure: one strange character**

```
(V-WIRE_32 (A_[0..31] B_[0..31]) (Y_[0..31])
     ((G_0 (Y_0) T-WIRE (A_0 B_0)) (G_1 (Y_1) T-WIRE (A_1 B_1))
      (G_8 (Y_8) T-GIRE (A_8 B_8)) (G_9 (Y_9) T-WIRE (A_9 B_9))
      (G_10 (Y_10) T-WIRE (A_10 B_10)) (G_11 (Y_11) T-WIRE (A_11 B_11))
```

  - **Could not reproduce on my machine; *could* on engineer's**
  - **Different places or characters on other runs: memory overwrite?**
- **No hint of code overwrite (common in C)**
- **A table of *where* the failure occurred in output files showed that all had same low-order bits in hexadecimal**
- **Conclusion: flaky bit in output hardware!**

Assurance from three sources

$$A = f(p, s, e)$$

where *A* is functional assurance, *p* is process quality, *s* is assessed quality of software, and *e* is environment resilience.

p is process quality

$$A = f(p, s, e)$$

- **High assurance software must be developed with care, for instance:**
  - **Validated requirements**
  - **Good, simple system architecture**
  - **Safety designed- and built in**
  - **Trained programmers**
  - **Helpful programming language**

s is assessed quality of software

$$A = f(p, s, e)$$

- **There are two general kinds of software assessment:**
  - **Static analysis**
    - **e.g. code reviews and scanner tools**
    - **examines code**
  - **Testing (dynamic analysis)**
    - **e.g. simulations, fault injection, and test beds**
    - **runs code**

e is environment resilience

$$A = f(p, s, e)$$

- **The execution platform can add assurance that the system will function as intended.**
- **Some techniques are:**
  - **Physical enforcement mechanisms**
  - **Execute in a “sandbox” or virtual machine**
  - **Monitor execution and react to violations**
  - **Replicate processes and vote on output**

Static Analysis and Testing Complement Each Other

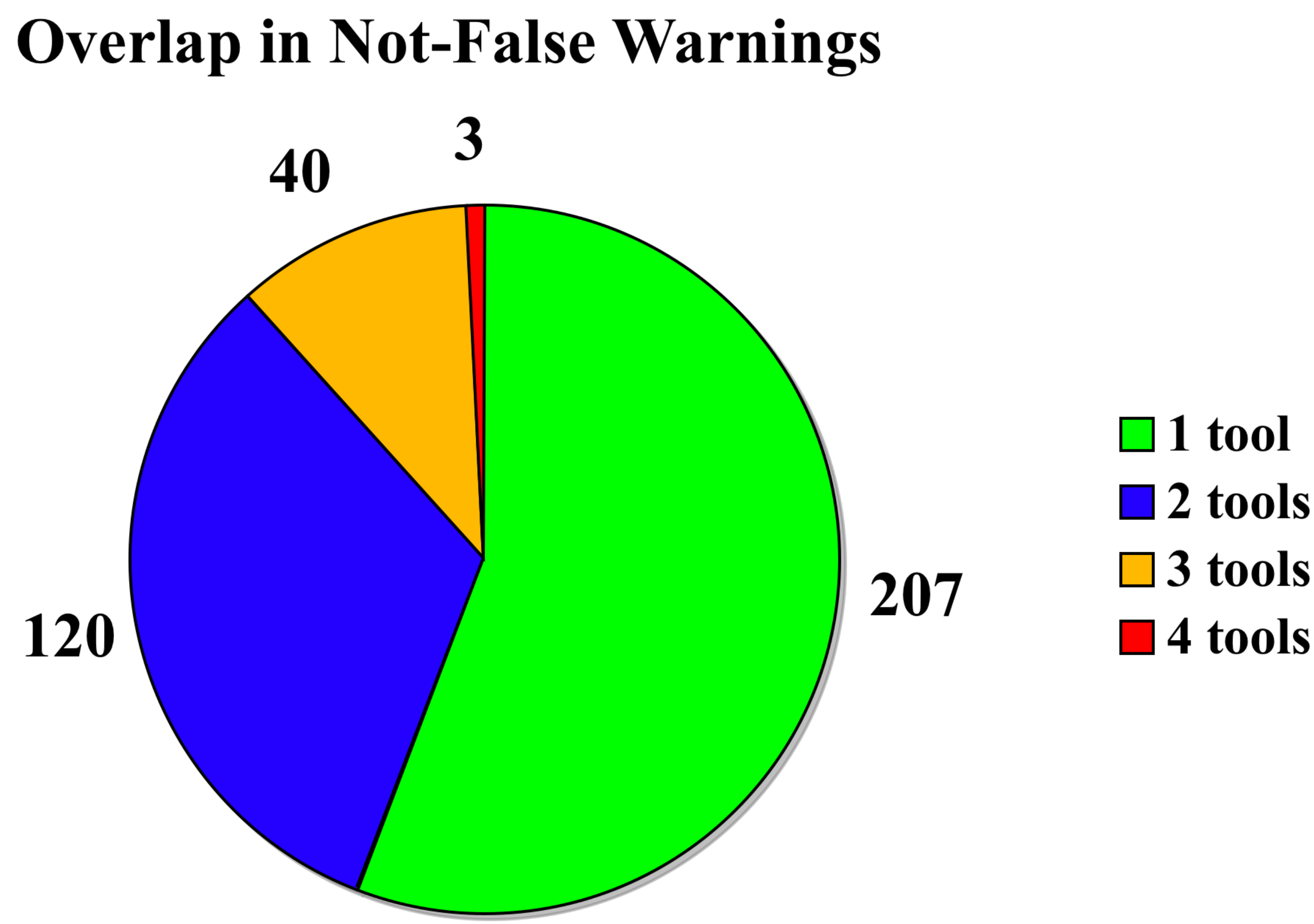
Static Analysis

- Handles unfinished code
- Higher level artifacts
- Can find backdoors, e.g., full access for user name "JoshuaCaleb"
- Potentially complete

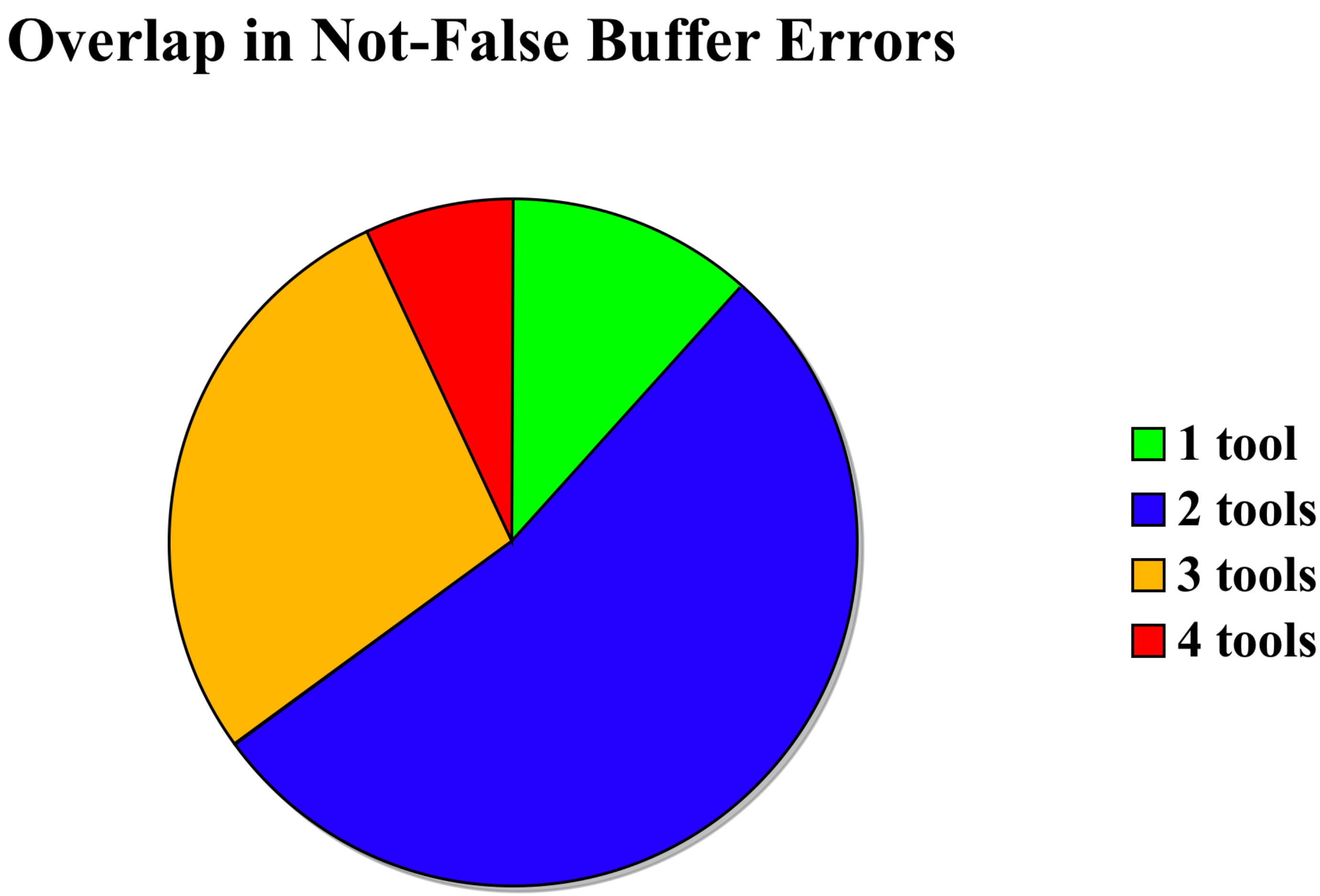
Testing

- Code not needed, e.g., embedded systems
- Has few(er) assumptions
- Covers end-to-end or system tests
- Assess as-installed

Static analysis tools don’t report the same warnings

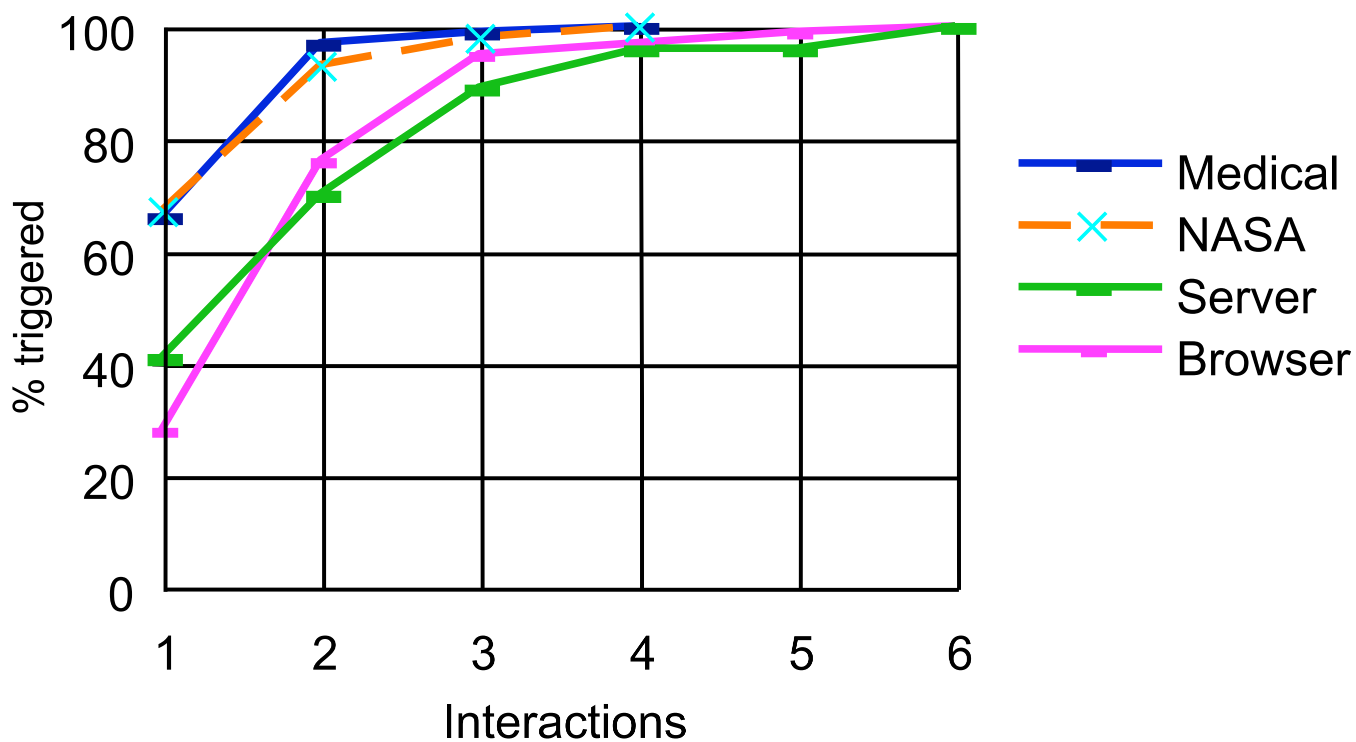


Tools overlap more on some types

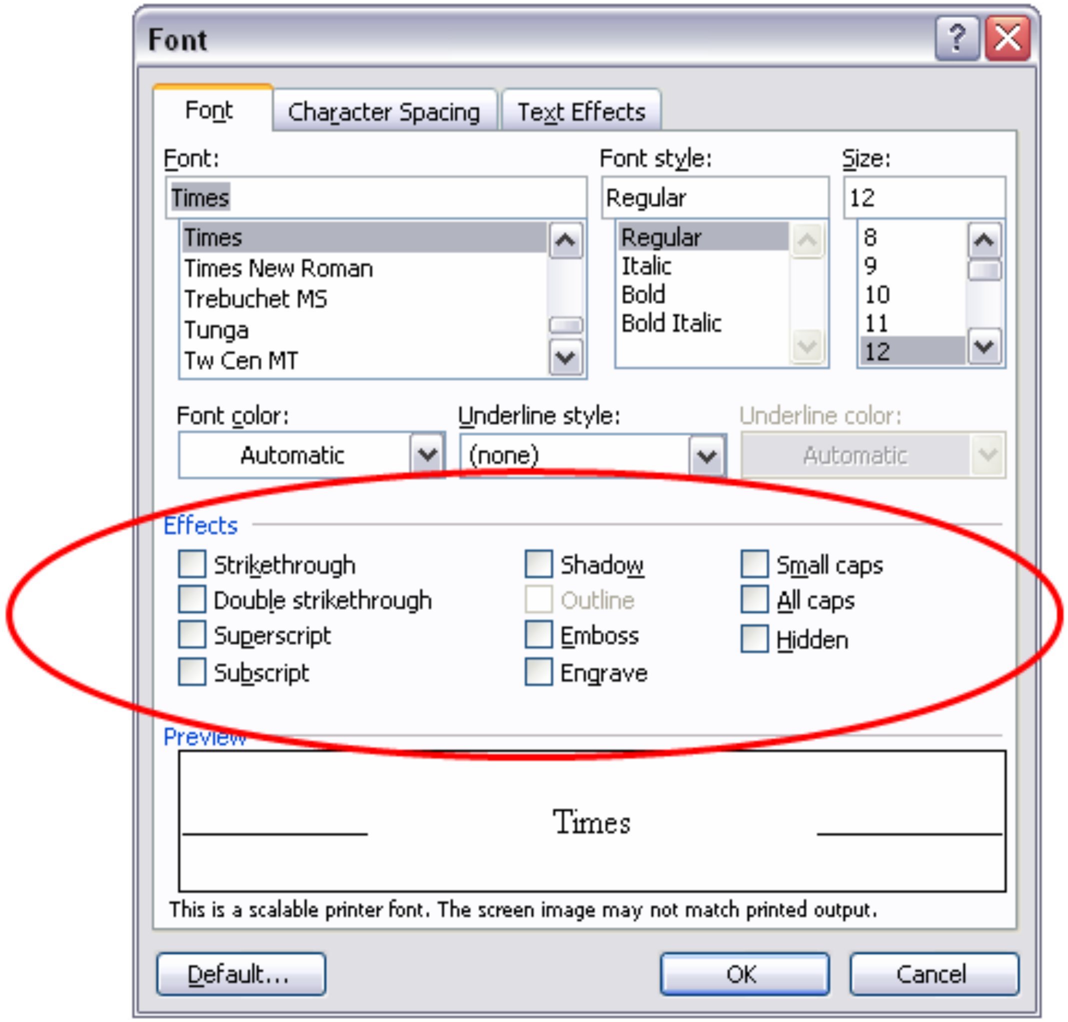


Combinatorial testing for software

- **NIST studied software failures in many fields**
- **Pairwise testing would not find all errors. But a maximum of 6-way testing triggered all faults.**



A simple example



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.**
- **But each test exercises many triples:**

0 0 0 1 1 1 0 1 0 1

We oughta be able to pack a lot in one test, so what's the smallest number we need?

All triples take only 13 tests!

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

Take aways

- **Assurance comes from 3 places:**
  - **Process quality**
  - **Software assessment**
  - **Environment resilience**
- **Testing and static analysis complement each other**
- **Combinatorial testing spreads test points throughout behavior space**