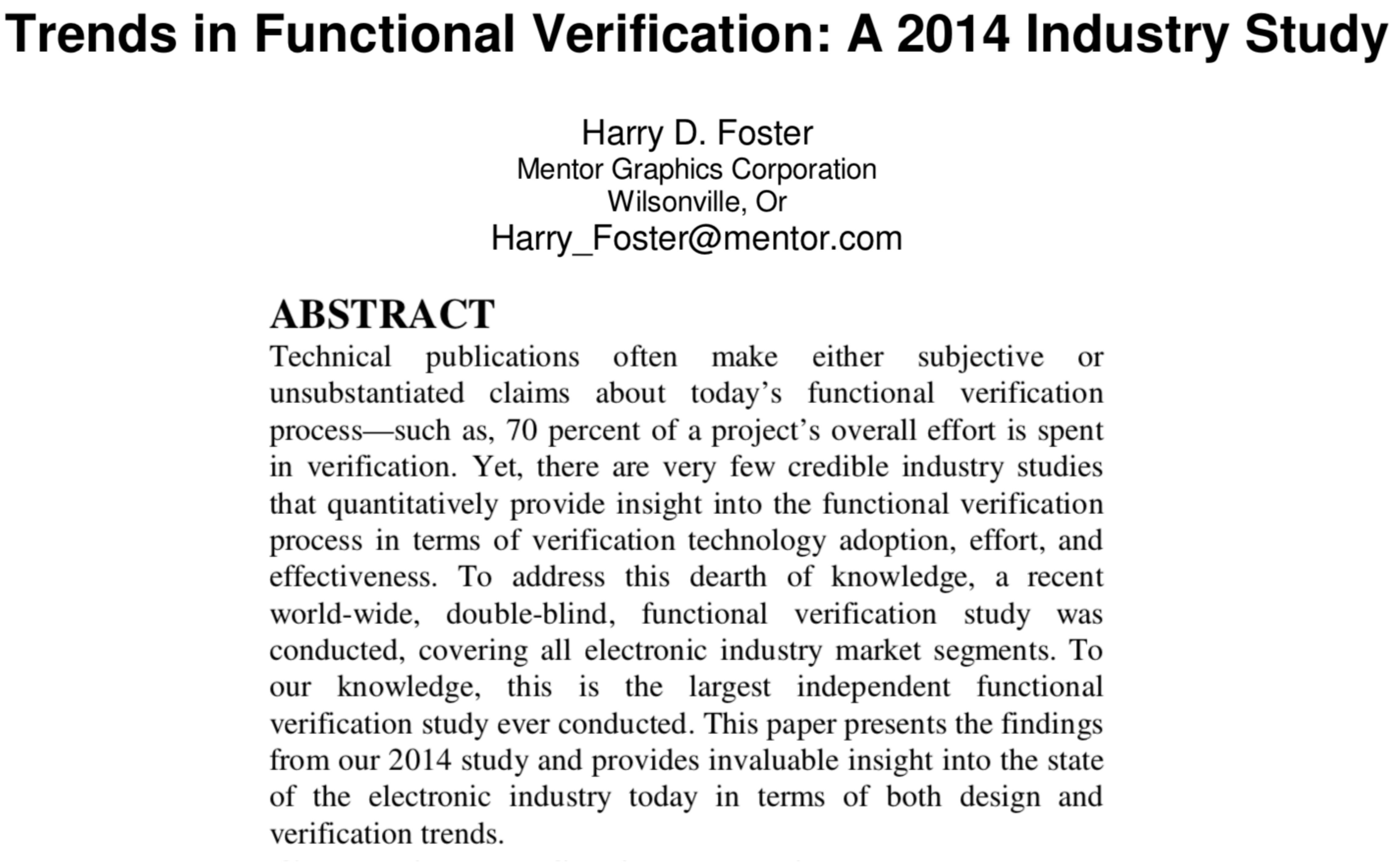
Property-Based Testing in Industry

Pete Manolios
Northeastern

Logic and Computation, 1/24/2019



Trends in Functional Verification: A 2014 Industry Study

Harry D. Foster
Mentor Graphics Corporation
Wilsonville, Or
Harry_Foster@mentor.com

ABSTRACT

Technical publications often make either subjective or unsubstantiated claims about today's functional verification process—such as, 70 percent of a project's overall effort is spent in verification. Yet, there are very few credible industry studies that quantitatively provide insight into the functional verification process in terms of verification technology adoption, effort, and effectiveness. To address this dearth of knowledge, a recent world-wide, double-blind, functional verification study was conducted, covering all electronic industry market segments. To our knowledge, this is the largest independent functional verification study ever conducted. This paper presents the findings from our 2014 study and provides invaluable insight into the state of the electronic industry today in terms of both design and verification trends.

Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

Property-Based Testing in Industry

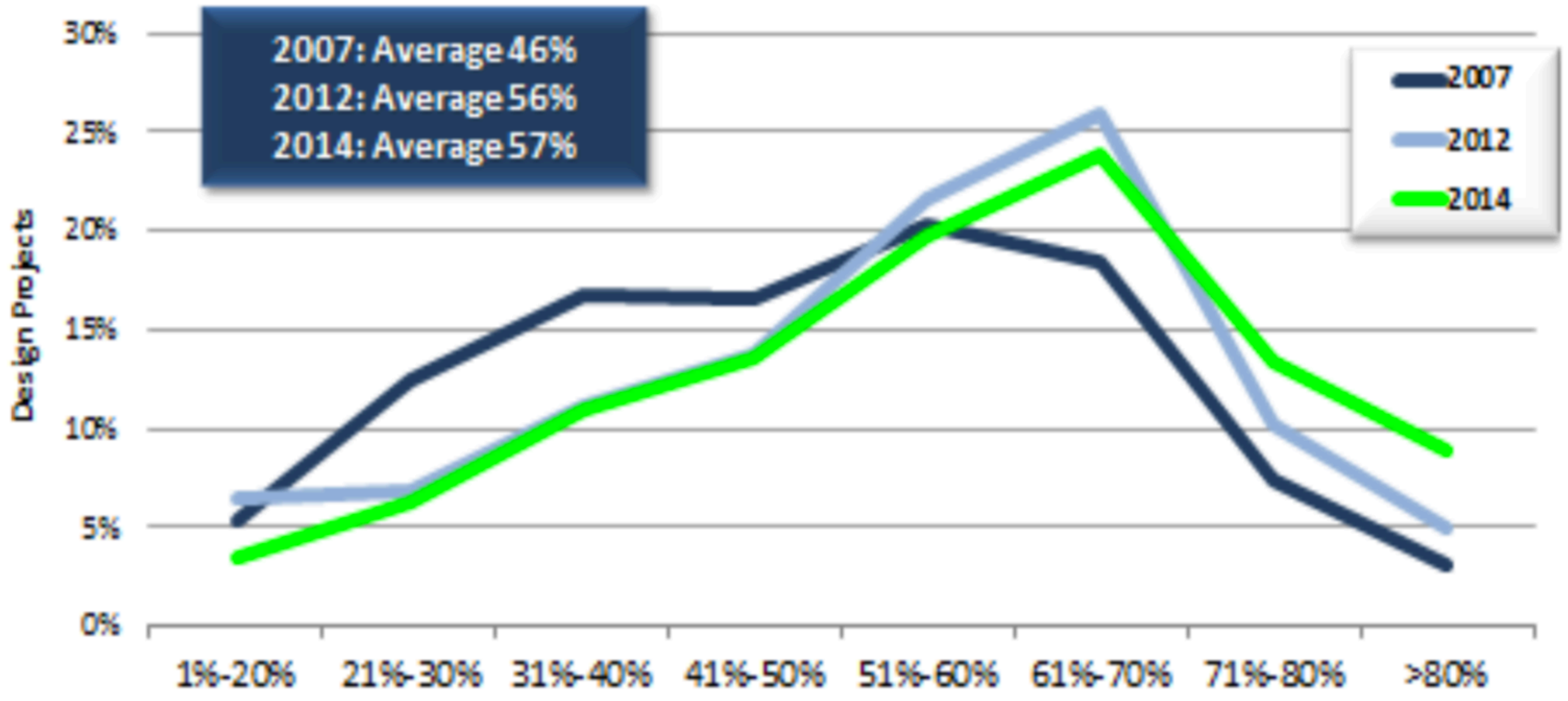


Figure 3. Percentage of Project Time Spent in Verification

Property-Based Testing in Industry

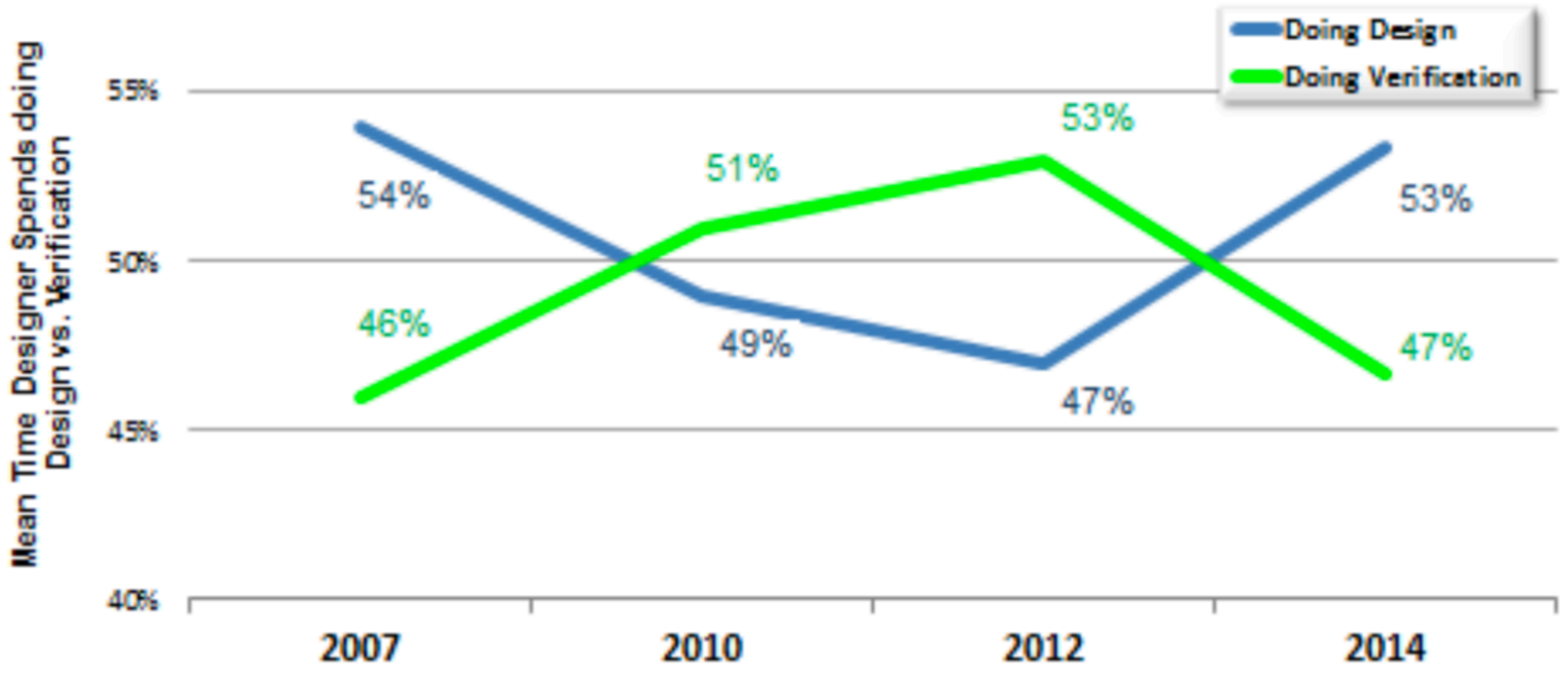


Figure 5. Where Design Engineers Spend Their Time

Property-Based Testing in Industry

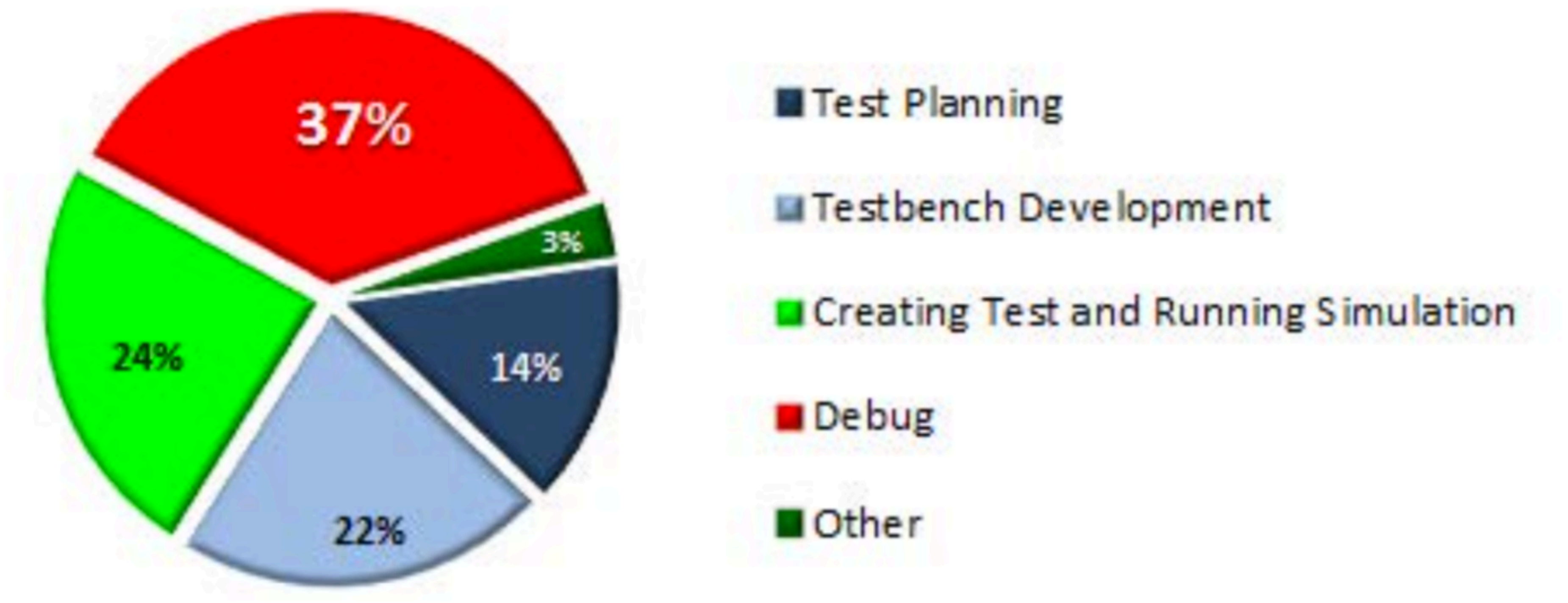


Figure 6. Where Verification Engineers Spend Their Time

Property-Based Testing in Industry

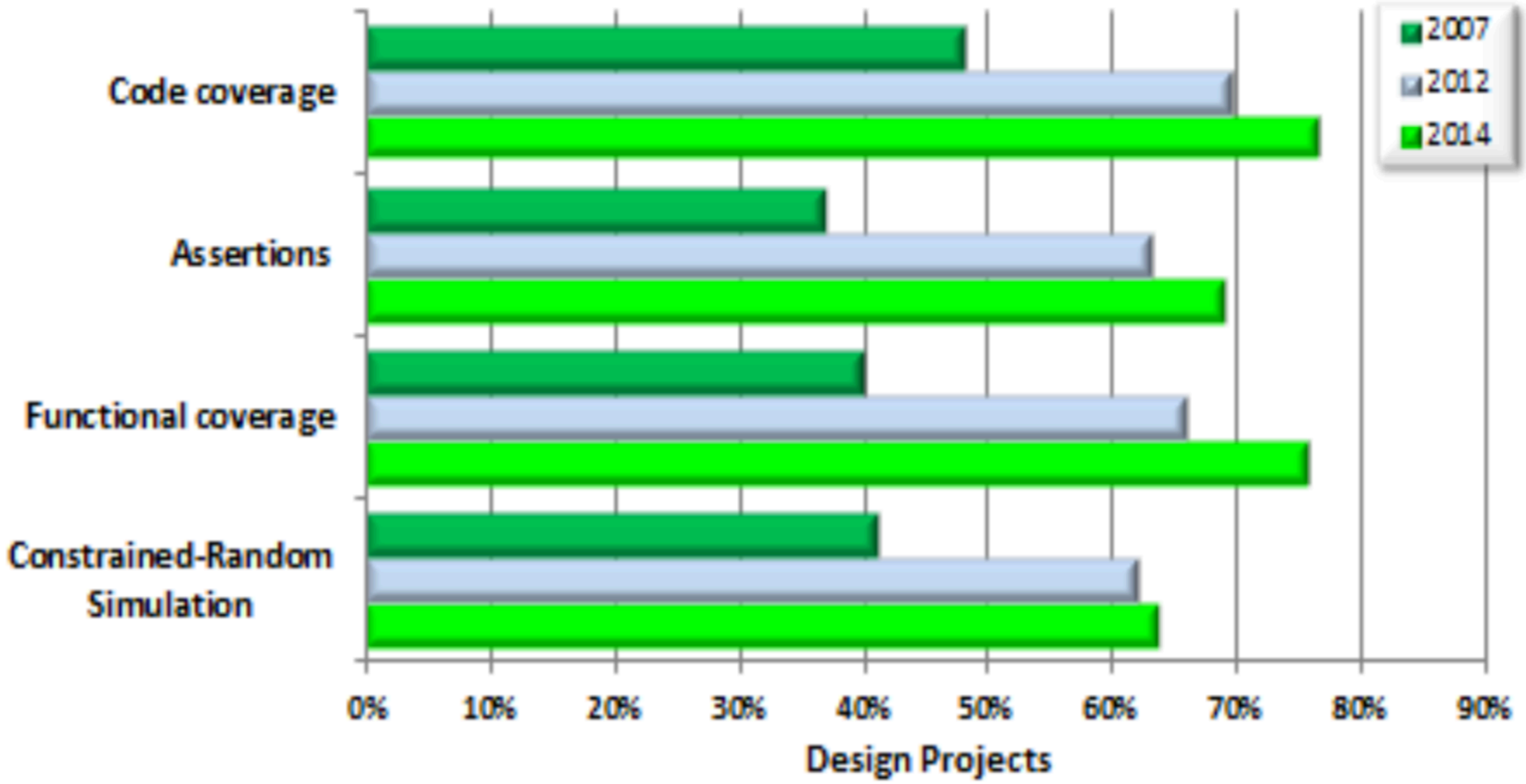


Figure 7. Dynamic Verification Technology Adoption Trends

Fuzzing

Software testing technique that uses unexpected inputs to test software.

This idea goes back to the 1950's when random punched cards were used to test programs.

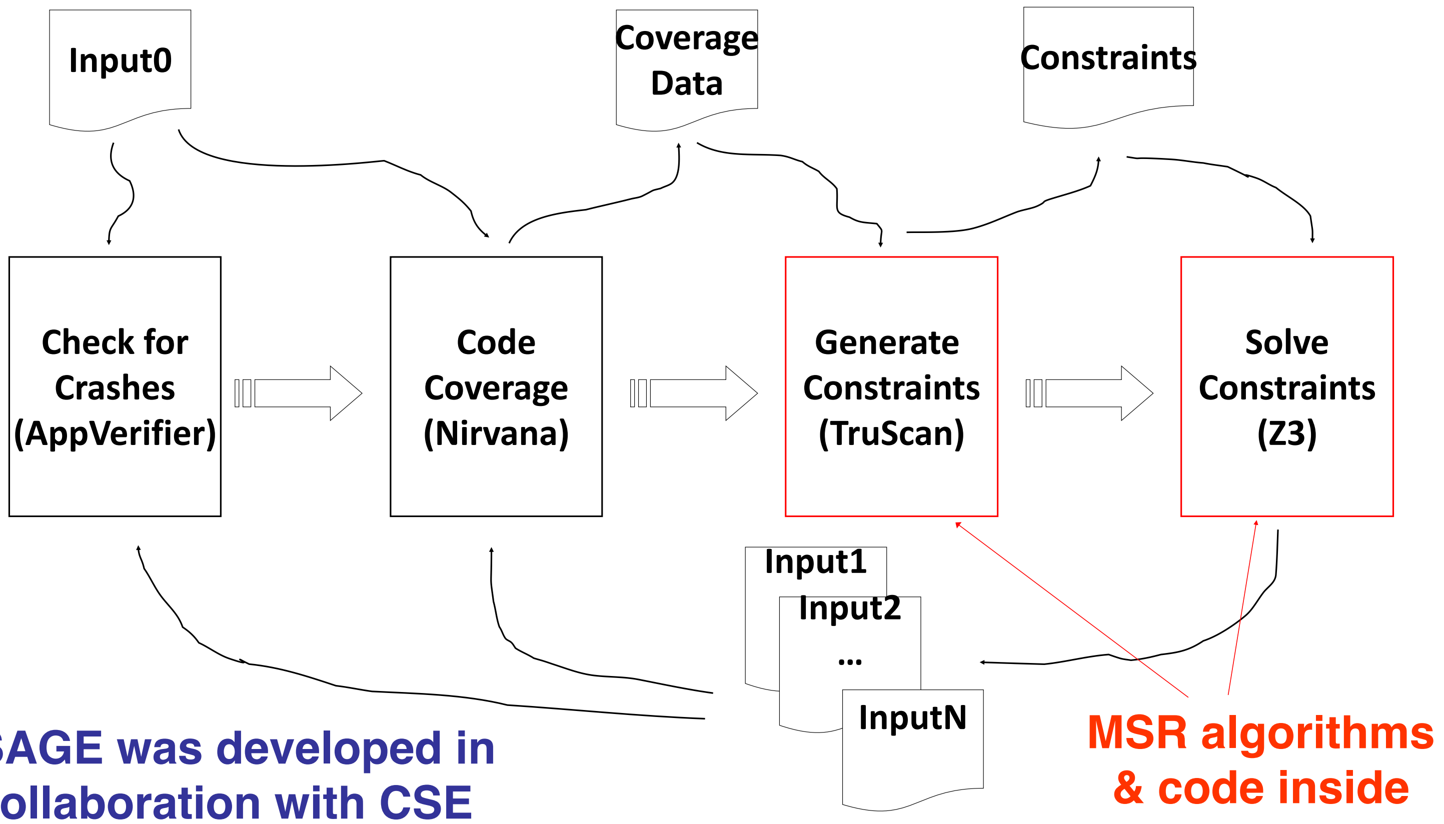
What about using contracts?
This would solve lots of problems.

The … work was inspired by being logged on to a modem during a storm with lots of line noise. And the line noise was generating junk characters that seemingly was causing programs to crash. The noise suggested the term "fuzz".

--Barton Miller, University of Wisconsin (1988)

SAGE: Whitebox Fuzzing for Security Testing

Basic idea:
1. Run the program with first inputs,
2. gather constraints on inputs at conditional statements,
3. use a constraint solver to generate new test inputs,
4. repeat - possibly forever!



SAGE was developed in collaboration with CSE

Impact: since 2007
- 200+ machine years (in largest fuzzing lab in the world)
- 1 Billion+ constraints (largest SMT solver usage ever!)
- 100s of apps, 100s of bugs (missed by everything else…)
- Ex: 1/3 of all Win7 WEX security bugs found by SAGE →
- Bug fixes shipped quietly (no MSRCs) to 1 Billion+ PCs
- Millions of dollars saved (for Microsoft and the world)
- SAGE is now used daily in Windows, Office, etc.

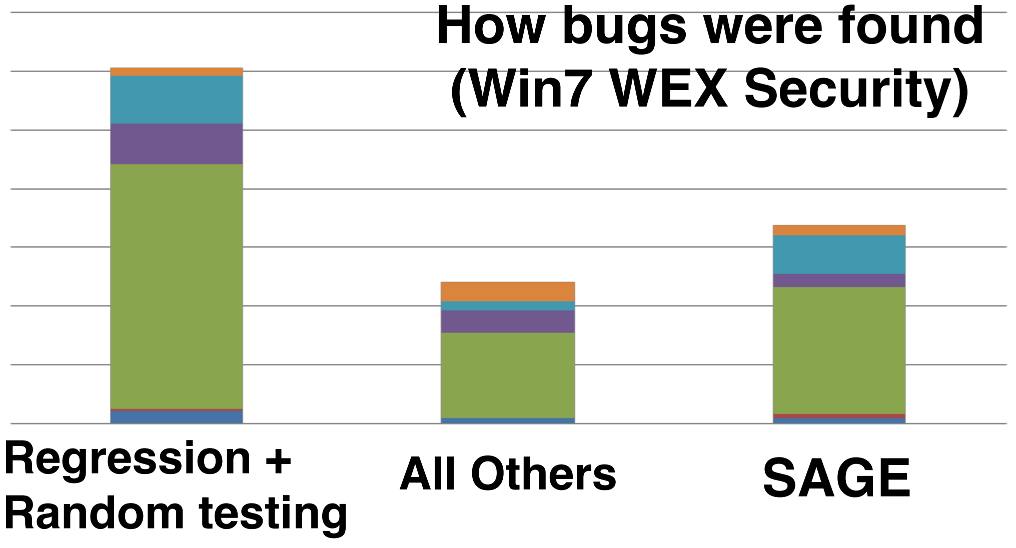
The SAGE team:
MSR: E. Bounimova, P. Godefroid, D. Molnar
CSE: M. Levin, Ch. Marsh, L. Fang, S. de Jong,…
+ thanks to all the SAGE users!
Windows: N. Bartmon, E. Douglas, D. Duran, I. Sheldon
Office: T. Gallagher, E. Jarvi, O. Timofte

SAGE is the first whitebox fuzzer

Research Challenges:
- How to recover from imprecision ? PLDI'05, PLDI'11
- How to scale to billions of x86 instructions? NDSS'08
- How to check many properties together? EMSOFT'08
- How to leverage grammar specifications? PLDI'08
- How to deal with path explosion ? POPL'07,TACAS'08
- How to reason precisely about pointers? ISSTA'09
- How to deal with floating-point instr.? ISSTA'10
- How to deal with input-dependent loops? ISSTA'11

+ research on constraint solvers

How bugs were found (Win7 WEX Security)



SAGE: Whitebox Fuzzing for Security Testing

- See http://queue.acm.org/detail.cfm?id=2094081

SAGE example

```
int foo(int x) { // x is an input
  int y = x + 3;
  if (y == 13) abort(); // error
  return 0;
}
```

- A software error is a violation of a property the program should satisfy
  - In ACL2s, that would be a conjecture (or property) that is not true
- One can phrase many properties in terms of reachability
  - If execution can reach a particular statement (e.g., abort) the property is violated

SAGE example2

```
void top (char input[4] {
  int cnt=0;
  if (input[0] == 'b') cnt++;
  if (input[1] == 'a') cnt++;
  if (input[2] == 'd') cnt++;
  if (input[3] == '!') cnt++;
  if (cnt >= 4) abort(); ?? error
}
```

- The program takes four bytes as input and contains an error when the value of the variable cnt is greater than or equal to four.

SAGE example in ACL2s

- Demo

Security

- Not all software errors are security vulnerabilities
- But software security vulnerabilities are just software errors
  - Linus Torvalds (11/2017): *Some security people have scoffed at me when I say that security problems are primarily "just bugs." Those security people are f*cking morons*

Next Time

- Propositional Logic
- Chapter 3