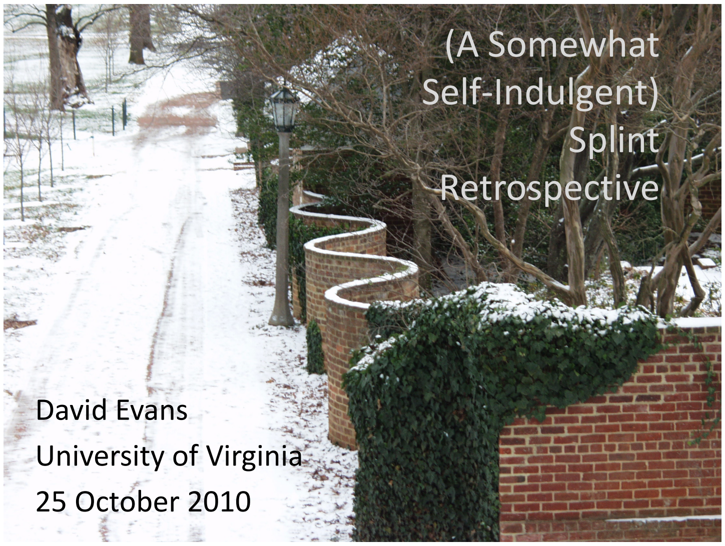


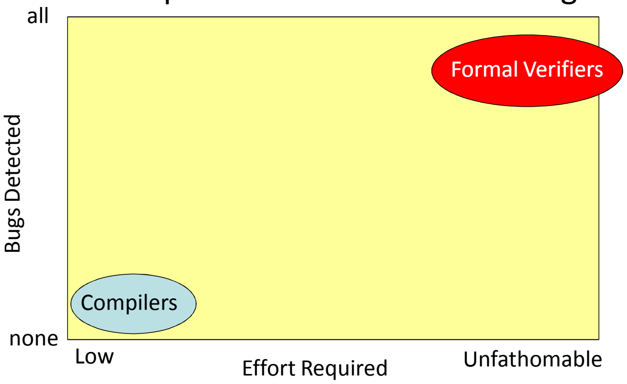
(A Somewhat Self-Indulgent) Splint Retrospective

David Evans
University of Virginia
25 October 2010

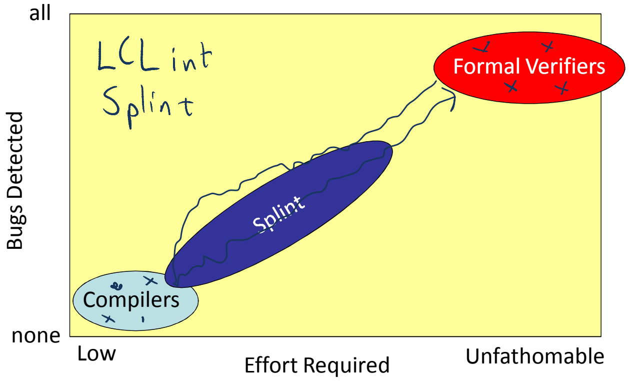
Splint Pre-History

- Pre-history
  - 1973: Steve Ziles – algebraic specification of se
  - 1975: John Guttag’s PhD thesis: algebraic specifications for abstract datatypes
  - 1983: Jeanette Wing’s PhD thesis: two-tiered specifications – separate program interface from underlying semantics
- 1993: John Guttag/Jeanette Wing seminar
  - Larch family specification, theorem prover
  - interface specification languages (including LCL)

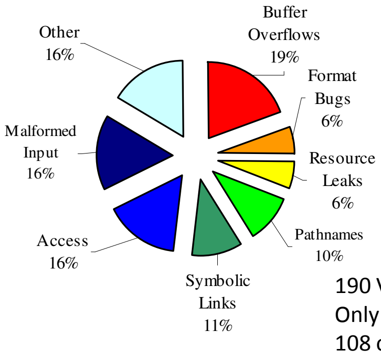
Formal verifiers are too expensive and time consuming…



Splint offers a low-effort Alternative



Security Flaws



Reported flaws in Common Vulnerabilities and Exposures Database, Jan-Sep 2001.
[Evans & Larochelle, IEEE Software, Jan 2002.]

190 Vulnerabilities
Only 4 having to do with crypto
108 of them could have been detected with simple static analyses!

(~~Almost~~) Everyone except for John Hates Specifications

- Hard to understand
- Lots of strange notations
- Don’t match what the code does
- Can’t even run them

(Almost) Everyone Likes Types

- Easy to Understand
- Easy to Use
- Quickly Detect Many Programming Errors
- Useful Documentation
- **…even though they are lots of work!**
  - 1/4 of text of typical C program is for types

Types	Attributes
Type of reference never changes	State changes along program paths
Language defines checking rules	System *or programmer* defines checking rules
One type per reference	Many attributes per reference

volatile
static
const char * s;

Approach

- Programmers add "annotations" (formal specifications)
  - Simple and precise
  - Describe programmers intent:
    - Types, memory management, data hiding, aliasing, modification, null-ity, buffer sizes, security, etc.
- Splint detects **inconsistencies between annotations and code**
  - Simple (fast!) dataflow analyses
  - **Intraprocedural**: except for annotations
  - **Unsound** and **incomplete**

Sample Annotation: **only**

```
extern /*@only@*/ char *gptr;
extern /*@only*/ /*@out null void *malloc (int);
```

- Reference (return value) *owns* storage
- No other persistent (non-local) references to it
- Implies obligation to transfer ownership
- Transfer ownership by:
  - Assigning it to an external **only** reference
  - Return it as an **only** result
  - Pass it as an **only** parameter: e.g.,

    extern void free (**only** void *);

Example

extern only null void *malloc (int); *in library*

```
1 int dummy (void) {
2   int *ip= (int *) malloc (sizeof (int));
3   *ip = 3;
4   return *ip;
5 }
```

Splint output:

```
dummy.c:3:4: Dereference of possibly null pointer ip: *ip
         dummy.c:2:13: Storage ip may become null
dummy.c:4:14: Fresh storage ip not released before return
dummy.c:2:43: Fresh storage ip allocated
```

Example: Buffer Overflows

- Most commonly exploited security vulnerability
  - 1988 Internet Worm
  - Still the most common attack
    - Code Red exploited buffer overflow in IIS
    - >50% of CERT advisories, 23% of CVE entries in 2001
- Attributes describe sizes of allocated buffers
- Heuristics for analyzing loops
- Found several known and unknown buffer overflow vulnerabilities in wu-ftpd

Adding Data Abstraction to C

```
typedef /*@abstract@*/ /*@immutable@*/ char *mstring;
```

- Warnings if code depends on the representation of an abstract type
- Biggest payoff in maintainability for minimal effort

Defining Properties to Check

- Many properties can be described in terms of state attributes
  - A file is *open* or *closed*
    - fopen: returns an *open* file
    - fclose: *open* → *closed*
    - fgets, etc. require open files
  - Reading/writing – must reset between certain operations

Defining Openness

```
attribute openness
  context reference FILE *
  oneof closed, open
  annotations
     open ==> open  closed ==> closed
  transfers
    open as closed ==> error
    closed as open ==> error
  merge open + closed ==> error
  losereference
    open ==> error "file not closed"
  defaults
    reference ==> open
end
```

if

Object cannot be open on one path, closed on another

Cannot abandon FILE in open state

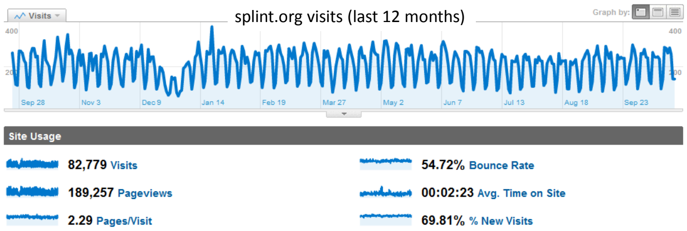
Specifying I/O Functions

```
/*@open@*/ FILE *fopen
    (const char *filename,
     const char *mode);

int fclose (/*@open@*/ FILE *stream)
  /*@ensures closed stream@*/ ;

char *fgets (char *s, int n,
             /*@open@*/ FILE *stream);
```

Checking

- Simple dataflow analysis
- Intraprocedural – except uses annotations to alter state around procedure calls
- Integrates with other Splint analyses (e.g., nullness, aliases, ownership, etc.)

Splint Success/Failure



82,779 Visits

189,257 Pageviews

2.29 Pages/Visit

54.72% Bounce Rate

00:02:23 Avg. Time on Site

69.81% % New Visits

Academic impact: over 1000 citations (4 papers with > 200 each)

Practice impact:

still used, mostly in embedded software development (C)

incorporated in popular Linux distributions, commercial products

"Splint-inspired" tools are widely used: **PREfix/PREfast, Fortify, FindBugs**

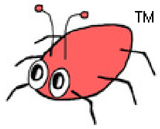
Failures:

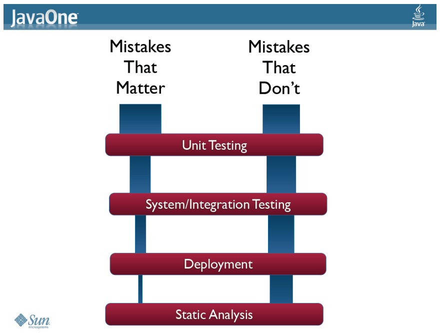
no slippery slope to more advanced uses of formal methods

did not build a self-maintaining open source community

FindBugs

http://findbugs.sourceforge.net




JavaOne

Mistakes That Matter

Mistakes That Don't

Unit Testing

System/Integration Testing

Deployment

Static Analysis

Slide from Bill Pugh's talk (2009)

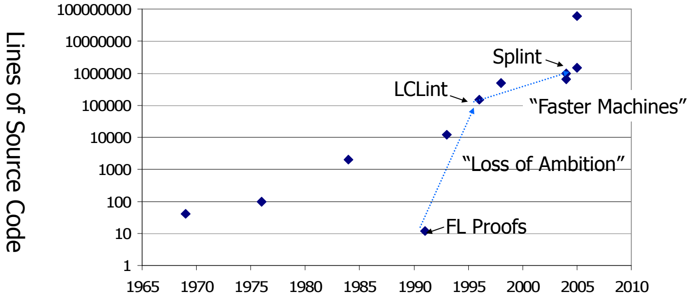
Static/Dynamic Analysis: Past, Present and Future

Verification Grand Challenge Workshop
SRI Menlo Park
22 February 2005

Original slides: with updates in orange boxes

David Evans
University of Virginia
Computer Science

The Past: Trends



The Present

- Microsoft PREfix/fast, SLAM→SDV
- ASTRÉE (Cousot) – Airbus A380

coverity

$35M in 2008

PolySpace TECHNOLOGIES

Acquired by MathWorks, 2007

FORTIFY SOFTWARE

$41M in 2008

ASA Security Analyzer

cigital

Secure Software

Reflective

OUNCE LABS

Acquired by IBM, 2009

The Present

- Static Analysis: good at checking generic requirements (types, buffer overflows, …)
- Dynamic Analysis: good at checking assertions inserted by programmer
- Bad at knowing what properties to check
  - Automatic inference techniques
  - Grand Challenge Repository
- No good techniques for combining static and dynamic analyses

A few since 2005!
Concolic Testing [Sen et al., 2007], SAGE (MSR)

The Future: Predictions for 2015

1. Software vendor will lose a major lawsuit because of a program bug

   Has this happened?

2. Someone will come up with a cool name like "VerXifiedProgramming" and sell a lot of books on program verification

   Still waiting…but 5 years left!

3. No more buffer overflows in major commercial software
   - Brian Snow at 20th Oakland conference (1999) predicted we will still be talking about buffer overflows in 2019

   SANS list 2010: Buffer overflows are still #3
   but…not in OWASP top ten

Predictions for 2015

4. Standard compilers prevent most concurrency problems

   Still a long way off…but lots of work going on

5. Programmers will still make dumb mistakes and resist change
6. “Good” CS degree programs will:
   - Incorporate verification into their first course
   - Include a course on identifying and checking program properties

Making Predictions

Never make predictions, especially about the future.
– Casey Stengel

The best way to predict the future is to invent it.
– Alan Kay, 1971

Our plan and our hope was that the next generation of kids would come along and do something better than Smalltalk around 1984 or so… But a variety of different things conspired together, and that next generation actually didn’t show up.
– Alan Kay, 2005