Weird things that surprise academics trying to commercialize a static checking tool.

Andy Chou, Ben Chelf, Seth Hallem
Charles Henri-Gros, Bryan Fulton, Ted Unangst
Chris Zak
Coverity

Dawson Engler
Stanford

A naïve view

- Initial market analysis:
  - **"We handle linux, bsd, we just need a pretty box!"**
  - **Not quite.**
- First rule of static analysis: no check, no bug.
  - **Two first order examples we never would have guessed.**
  - **Problem 1: if you can't find the code, can't check it.**
  - **Problem 2: if you can't compile code, you can't check it.**
- And then: how to make money on software tool?
  - **"Tools. Huh. Tools are hard." Any VC in early 2000.**

Myth: the C (or C++) language exists.

- Well, not really. The standard is not a compiler.
  - **What exists: gcc-2.1.9-ac7-prepatch-alpha, xcc-i-did-not-understand-pages4,33,208-242-of-standard.**
  - **Oh. And Microsoft. Conformance = competitive disadvantage. Do the math on how this deforms .c files**
  - **Basic LALR law: What can be parsed will be written.**
- Rule: static analysis must compile code to check.
  - **If you cannot (correctly) parse "language" cannot check.**
  - **Common (mis)usage model: "allegedly C" header file does something bizarre not-C thing. Included by all source. Customer watches your compiler emit voluminous parse errors. (This is not impressive.)**
  - **Of course: gets way worse with C++ (which we support)**

Some bad examples to find in headers

- Banal. But take more time than you can believe:

```
void x;
```

```
short x; int *y = &(int)x;
```

```
int foo(int a, int a);
```

```
unsigned x @ "TEXT";
```

```
// unless "-packed"!
__packed (…) struct foo { … }
```

```
unsigned x = 0xdead_beef;
```

```
End lines with "\r" rather than "\n"
```

- And, of course, asm:

```
#pragma asm
   mov eax, eab
#pragma end_asm
```

```
asm foo() {
      mov eax, eab;
}
```

```
// newline = end
__asm mov eax, eab

// "]" = end
__asm [
      mov eax, eab
]
```

Microsoft example: precompiled headers

- Spec:

> **The compiler treats all code occurring before the .h file as precompiled. It skips to just beyond the #include directive associated with the .h file, uses the code contained in the .pch file, and then compiles all code after filename**

- Implication

```
I can put whatever I want here.
It doesn't have to compile.
If your compiler gives an error it sucks.
#include <some-precompiled-header.h>
```

- It gets worse: on-the-fly header fabrication

Solution: pre-preprocessing rewrite rules.

- Supply regular expressions to rewrite bad constructs

```
#pragma asm
…
#pragma end_asm
```

↓

```
ppp_translate ("/#pragma asm/#if 0/");
ppp_translate("/#pragma end_asm/#endif/");
```

↓

```
#if 0
…
#endif
```

What this all means concretely.

- We use Edison Design Group (EDG) frontend
  - Pretty much everyone uses. Been around since 1989.
  - Aggressive support for gcc, microsoft, etc. (bug compat!)
- Still: coverity by far the largest source of EDG bugs:
  - 146 parsing test cases (i.e., we got burned)
  - 219 compiler line translation test cases (i.e., ibid).
  - 163 places where frontend hacked ("#ifdef COVERITY")
  - Still need custom rewriter for many supported compilers:

```
205 hpux_compilers.c
215 iar_compiler.c
240 ti_compiler.c
251 green_hills_compiler.c
377 intel_compilers.c
453 diab_compilers.c

453 sun_compilers.c
485 arm_compilers.c
617 gnu_compilers.c
748 microsoft_compilers.c
1587 metrowerks_compilers.c
...
```

Academics don't understand money.

- "We'll just charge per seat like everyone else"
  - Finish the story: "Company X buys three Purify seats, one for Asian, one for Europe and one for the US…"
- Try #2: "we'll charge per lines of code"
  - "That is a really stupid idea: (1) …, (2) … , … (n) …"
  - Actually works. I'm still in shock. Would recommend it.
- Good feature for seller:
  - No seat games. Revenue grows with code size. Run on another code base = new sale.
- Good feature for buyer: No seat-model problems
  - Buy once for project, then done. No per-seat or per-usage cost; no node lock problems; no problems adding, removing or renaming developers (or machines)
  - People actually seem to like this pitch.

Some experience.

- Surprise: Sales guys are great
  - Easy to evaluate. Modular.
- Company X buys tool, then downsizes. Good or bad?
  - For sales, very good: X fires 110 people. They get jobs elsewhere. Recommend coverity. 4 closed deals.
- Large companies "want" to be honest
  - Veritas: want monitoring so don't accidently violate!
- What can you sell?
  - User not same as tool builder. Naïve. Inattentive. Cruel.
  - Makes it difficult to deploy anything sophisticated.
  - Example: statistical inference.
  - Some ways, checkers lag much behind our research ones.

"No, your tool is broken: that's not a bug"

- "No, the loop will go through once!"

```
for(i=1; i < 0; i++) {
  …deadcode…
}
```

```
for(s=0; s < n; s++) {
  …
  switch(s) {
  case 0: assert(0);
return;
  …
  }
  …dead code…
```

- "No, && is 'or'!"

```
void *foo(void *p, void *q) {
  if(!p && !q)
    return 0;
```

- "No, ANSI lets you write 1 past end of the array!"
  ("We'll have to agree to disagree." !!!!)

```
unsigned p[4];   p[4] = 1;
```

Coverity's commercial history

Breakthrough technology out of Stanford	Company incorporated	Achieved profitability	Product growth and proliferation
2000-2002	2002	2003	2004-05
• Meta-level compilation checker ("Stanford Checker") detects 2000+ bugs in Linux.	• Deluge of requests from companies wanting access to the new technology. • First customer signs: Sanera systems	• 7 early adopter customers, including VMWare, SUN, Handspring. • Coverity achieves profitability.	• Version 2.0 product released. • Company quadruples • 70+ customers including Juniper, Synopsys, Oracle, Veritas, nVidia, palmOne. • Self funded

Stanford

A partial list of 70+ customers…

EDA: Synopsys, Mentor Graphics, cadence

Storage: VERITAS, EMC, panasas

Security: McAfee, PGP, Check Point

Networking: Cisco Systems, Juniper, Ericsson, AVICI

Government: NASA National Aeronautics and Space Administration, Jet Propulsion Laboratory California Institute of Technology

Embedded: QUALCOMM, symbol, KEITHLEY

Biz Applications: ORACLE, bmcsoftware

OS: WIND RIVER, palmOne

Open Source: MySQL, mozilla, Apache

Slide 8

DE6 whole bunch of options: razor blade model, where we give away checkers for free and charge for system. or inverse razor where we give away system and charge for checkers. or charge per seat, or charge per lines of code (prefix). get a lot of pushback on the last one. prefix worked ok, but not what we would consider a success.

i argued very strongly against per line model. completely wrong.
Dawson Engler, 8/23/2005

Slide 9

DE7 count how often something true versus not. sort in decreasing deviance. inspect until hit fp. developer inspect all, mark as FP. say tool sucks to everyone.
Dawson Engler, 8/23/2005

Slide 10

DE8 we know about these since happen in results meetings. a bit dangerous, but usually other developers will laugh at the confused one. scary to think of other times when things just marked as FP.
Dawson Engler, 8/23/2005

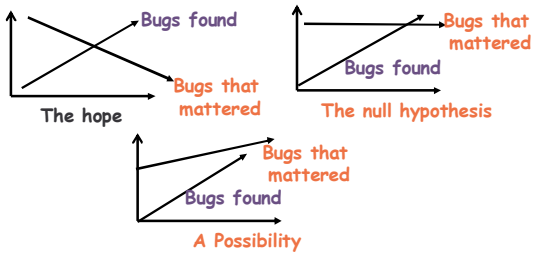
Summary

- Static analysis
  - Better at checking surface properties
  - Big wins: don't run code, all paths. Easy diagnosis.
  - Low incremental cost per line of code
  - Can get results in an afternoon: much easier to commercialize.
  - 10-100x more bugs.
- Model checking
  - Better at checking code implications.
  - Major win over testing: explore all actions a state can do before going to next
  - Makes low-probability events as probable as high.
  - Works very well when massive interleavings and bugs horrible.

Open Q: how to get the bugs that matter?

- Myth: all bugs matter and all will be fixed
  - *FALSE*
  - Find 10 bugs, all get fixed. Find 10,000…
- Reality
  - All sites have many open bugs (observed by us & PREfix)
  - Myth lives because state-of-art is so bad at bug finding
  - What users really want: The 5-10 that "really matter"
- General belief: bugs follow 90/10 distribution
  - Out of 1000, 100 account for most pain.
  - Fixing 900 waste of resources & may make things worse
- How to find worst? No one has a good answer to this.

Open Q: Do static tools really help?



Dangers: Opportunity cost. Deterministic bugs to non-deterministic.

Some cursory static analysis experiences

- Bugs are everywhere
  - Initially worried we'd resort to historical data…
  - 100 checks? You'll find bugs (if not, bug in analysis)
- Finding errors often easy, saying why is hard
  - Have to track and articulate all reasons.
- Ease-of-inspection *crucial*
  - Extreme: Don't report errors that are too hard.
- The advantage of checking human-level operations
  - Easy for people? Easy for analysis. Hard for analysis? Hard for people.
- Soundness not needed for good results.

Myth: more analysis is always better

- Does not always improve results, and can make worse
- The best error:
  - Easy to diagnose
  - True error
- More analysis used, the worse it is for both
  - More analysis = the harder error is to reason about, since user has to manually emulate each analysis step.
  - Number of steps increase, so does the chance that one went wrong. No analysis = no mistake.
- In practice:
  - Demote errors based on how much analysis required
  - Revert to weaker analysis to cherry pick easy bugs
  - Give up on errors that are too hard to diagnose.

Myth: Soundness is a virtue.

- Soundness: Find all bugs of type X.
  - Not a bad thing. More bugs good.
  - BUT: can only do if you check weak properties.
- What soundness really wants to be when it grows up:
  - Total correctness: Find all bugs.
  - Most direct approximation: find as many bugs as possible.
- Opportunity cost:
  - Diminishing returns: Initial analysis finds most bugs
  - Spend on what gets the next biggest set of bugs
  - Easy experiment: bug counts for sound vs unsound tools.
- End-to-end argument:
  - "It generally does not make much sense to reduce the residual error rate of one system component (property) much below that of the others.