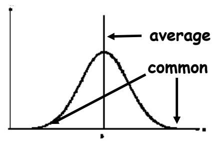


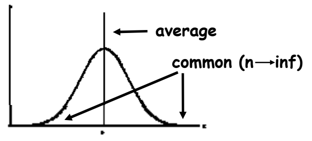
A few billion lines of code later: static checking in the real world

Andy Chou, Ben Chelf, Seth Hallem
Scott McPeak, Bryan Fulton, Charles-Henri Gros,
Ken Block, Anuj Goyal, Al Bessey
Chris Zak
& many others
Coverity

Dawson Engler
Associate Professor
Stanford

Many stories, two basic plots.

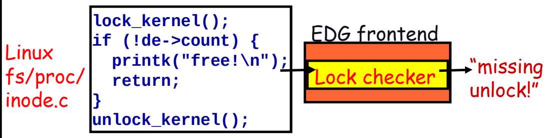
- Fun with Gaussian distributions:



- Social vs Technical: "What part of NO! do you not understand?"
  - **No: you cannot touch the build.**
  - **No: we will not change the source.**
  - **No: this illegal code is not illegal.**
  - **No: we will not understand your tool.**
  - **No: we do not understand static analysis.**

One slide of context

- Our tool [~2000]:
  - **The religion: Max bugs, min false errors or manual work.**
  - **Inter-procedural, context-sensitive, a bit path-sensitive**
  - **Aggressively unsound. Annotations viewed as evil.**



```
lock_kernel();
if (!de->count) {
  printk("free!\n");
  return;
}
unlock_kernel();
```

Worked reasonably well. Lots of bugs, papers, tenure.
Company successful enough that there is a marketing dept. (Next: proof)

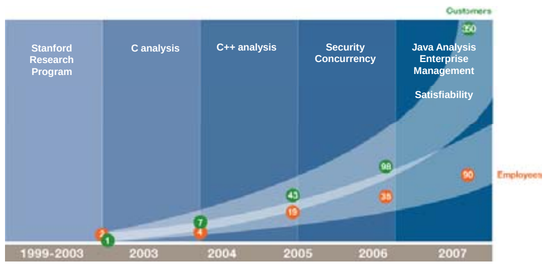
Caveats

- My (former) students run the company
  - **I am just a voyeur**
- Company is only tall for a midget
  - **Focus on things that will matter even more in larger settings**
- This is just "a" way to do things, not THE way
  - **Just how we did things; not as claim that they are best**
- General inferences from one data point = dubious.
  - **Our needs roughly a lowest common denominator(*)**
  - **(*): people building tool for single company need less: please speak up when your experience differs!**

Our Mission

To improve software quality by automatically identifying and resolving critical defects and security vulnerabilities in your source code

A short history of time [99-07]

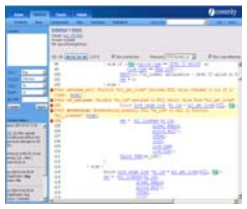


Over 1 Billion Lines of Code

Coverity Trial Process

Test your code quality
- Analyze your largest code base
- One day set up, two hours for results presentation
- Test drive the product at your facility

Benefit to your team
- Post trial report describing summary of findings
- Sample defects from your code base
- Fully functional defect resolution dashboard



Trial = a cornerstone verb of company.

- "Does your thing work worth a damn on my code?"
  - **Ship "sales engineer" and sales guy to company**
  - **Run over code; next day go over results**
  - **If bugs good, they (may) buy. If suck…**
- First order requirements:
  - **Must *regularly* go in cold, touch nothing, shove 1-10MLOC through tool, get good results.**
  - **Error reports must be clear since won't understand code**
  - **Low false positives, good bugs since can't cherry pick.**
- Some features:
  - **$0. Most people can sign for that. Cuts days of negotiation. Sales guy goes farther.**
  - **Straight-technology sale. Often buyer=user.**

Overview

- Context
- Now:
  - **A crucial myth.**
  - **Some laws of static analysis**
  - **And how much both matter.**
- Then: The rest of the talk.

A naïve view

- Initial market analysis:
  - **"We handle Linux, BSD, we just need a pretty box!"**
  - **Obviously naïve.**
  - **But not for the obvious reasons.**
- First law of checking: no check = no bug.
  - **Don't check a system, a path, a property, then find no bugs.**

  - **Two even more basic laws we'd never have guessed mattered.**

Law of static analysis: cannot check code you do not see.

How to find all code?

- Simple: intercept and rewrite build commands.

```
make -w  > & out
replay.pl out  # replace 'gcc' w/ 'prevent'
```

  - In theory: see all compilation calls and all options etc.
- Worked fine for a few customers.
  - Then: "make?"
  - Kept plowing ahead.
  - "Why do I have to re-install my OS from CD after I run your tool?"
  - Good question…

The right solution

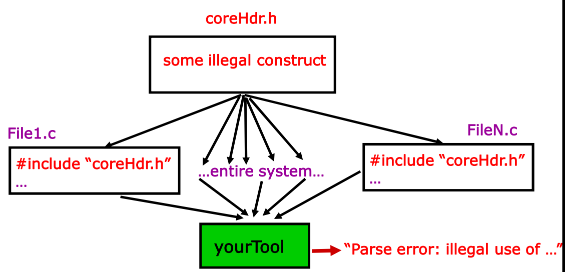
- Kick off build and intercept all system calls
  - Rewrite "<build command>" as "cov_build <build command>"
  - Know exact location of compiler, version, options, environ.
- In early 2000s more important than quality of analysis?
  - Go into company cold, touch nothing, kick off, see all code.
  - Big lever: 10x more code = 10x more bugs
  - Not bulletproof. Law:"Can't find code w/o command prompt"
- A typical story of $ transmuting the trite to first order
  - On windows: intercept = run compiler in debugger.
  - So?
  - Widely-used msoft compiler has a use-after-free bug
  - Works fine normally. Until run w/ debugger!
  - Solution?

(Another) Law of static analysis: cannot check code you cannot parse

Myth: the C language exists.

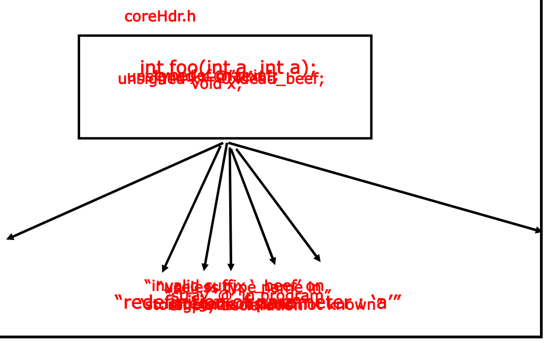
- Well, not really. The standard is not a compiler.
  - The language people code in?
  - Whatever strings their compiler accepts.
  - Fed illegal code, your frontend will reject it.
  - It's *your* problem. Their compiler "certified" it.
- Amplifiers:
  - Embedded = weird.
  - Msoft: standard conformance = competitive disadvantage.
  - C++ = language standard measured in kilos.
- Basic LALR law:
  - What can be parsed will be written. Promptly.
  - The inverse of "the strong Whorfian hypothesis" is a empirical fact, given enough monkeys..

A sad movie that will gross exactly $0.

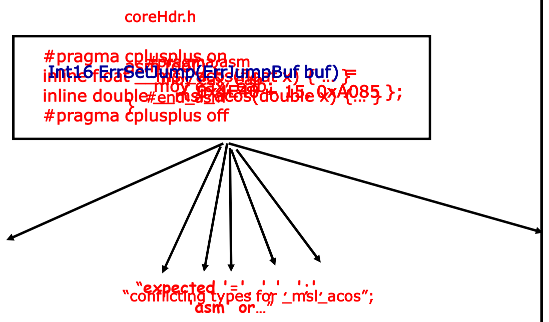


- "Deep analysis?! Your tool is so weak it can't even parse C!"

Some specific example stories.


coreHdr.h
int foo(int a, int a);
unsigned x = 0xdead_beef;
void x;
"invalid suffix '_beef' on …"
"stray '@' in program"
"redefinition of parameter 'a'"
"storage size of 'x' not known"

Some specific example stories.


coreHdr.h

```
#pragma cplusplus on
inline float __msl_acos(float x) { ... }
inline double __msl_acos(double x) { ... }
#pragma cplusplus off

Int16 ErrSetJump(ErrJumpBuf buf) =
{ 0x4E40 + 15, 0xA085 };

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

"conflicting types for '_msl_acos'";
"expected '=', ',', ';', 'asm' or…"

Great moments in unsound hacks

- Tool doesn't handle (illegal) construct?
  - Have reg-ex that runs before preprocessor to rip it out.
  - Amazingly gross.
  - Actually works.

```
#pragma asm
…
#pragma end_asm
```

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

Unsound = more bugs

```
#if 0
…
#endif
```

OK: so just how much does C not exist?

- 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:
  - 406 places where frontend hacked ("#ifdef COVERITY")
  - 266 "add compiler flag" calls
  - still need custom rewriter for many supported compilers:

912 arm.cpp	1656 metrowerks.cpp	457 sun.cpp
629 bcc.cpp	1294 microsoft.cpp	294 sun_java_.cpp
334 cosmic.cpp	285 picc.cpp	756 xlc.cpp
1848 cw.cpp	160 qnx.cpp	280 hpux.cpp
673 diab.cpp	1861 renesa.cpp	603 iccmsa.cpp
914 gnu.cpp	384 st.cpp	421 intel.cpp
		1425 keil.cpp

Completely unbelievable!

- Incredibly banal. But if not done, can't play.
  - Takes more effort than can imagine.
  - Full time team.
  - This is their mission. Never finished.
  - Certainly not in only 5 years.
- Two examples from trial reports from within *72 hours* of making this slide:

`__creregister volatile int x;` → `volatile int x;`

`#pragma packed 4 struct foo {…};` → `#pragma packed (4) struct foo{…};`

- *Never* would have guessed this is the first-order bound on how much bugs you find.

Annoying amplifier: Can we get source?

- *NO*!
  - Despite NDAs
  - Even for parse errors
  - Even for preprocessed
  - Might just be because too small to sue?
- Sales engineer has to type in from memory.
  - And this works as well as you'd expect.
  - Even worse for performance problems.
  - Oh, and you get about 3 tries to fix a problem.
- Bonus: add a TLA and things get worse.
  - NSA = Can we see source? NO!
  - FDA, FAA = frozen toolchain. Theirs. Yours. Banal, crucial: Where to get license for a 20+yr/old compiler?

The end result

- Heuristic: If you've heard of it, will wind up supporting it:

Windows, Solaris, GCC, Green Hills, Microsoft Visual Studio, HP-UX, hp, Sun, Wind River, ANSI, NetBSD, FreeBSD, ARM, intel, Texas Instruments

- Forced support for many things haven't heard of (or read obituary for).

A compiler development company / Photography company: "Specializing in Anime and SF/Fantasy Convention photography and other costuming photography. We can also do on-location photoshoots."

Overview

- Context
- The banal hand of reality:
  - Law: Cannot check code you can't find
  - Law: Cannot check code you can't parse
  - Myth: C exists
- Next:
  - Do bugs matter?
  - Do false positives matter?
  - Do false negatives matter?
- Academics meet reality. Reality wins.
  - You fix all bugs, right?
  - The evils of non-determinism

Do bugs matter?

- Shockingly common: clear, ugly crash error.
  - "So?"
  - "Isn't that bad? What happens?"
  - "Oh, will crash. We will get a call." Shrug.
- If developers don't feel pain, they often don't care.
  - If QA cannot reproduce, then no blame.
  - Our own engineers: If forced to run cprevent & fix bugs before checkin, what do you expect them to do?
- But bugs matter right?
  - Not if: Too many. Too hard. [More later]
- The next step down: "That's not a bug"
  - Recognition requires understanding.
  - Cubicles are plentiful. Understanding, not so much.

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

- "No, it's *loop*."

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

- "No, I meant to do that: they are next to each other"

```
int a[2], b;
memset(a, 0, 12);
```

- "No, that's ok: there is no malloc() between"

```
free(foo);
foo->bar = …;
```

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

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

(Often) People don't understand much.

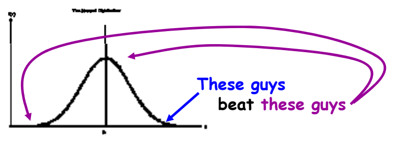
- Our initial naïve expectation: People who write code for money understand it. Instead:
  - "To build, I just press this button…"
  - "I'm just the security guy"
  - "That bug is in 3rd party code"
  - "Is it a leak? Author left years ago…"
- People don't even understand compilers.
  - "Static" analysis? What is the performance overhead?
  - Business card at customer site: "Static analyzer" (?!)
  - Anything that finds bugs = testing.
  - "Think of it as super compiler warnings"

Pop quiz

- If user doesn't understand error they:
  - A) Read manual
  - B) Mark as a false positive
- Social has *major* big impact on technical.
  - User not same as tool builder.
  - Uninformed. Inattentive. Cruel. Lazy.
  - HUGE problem. Prevents getting many things out in world.
- Give up on error classes that need too much sophistication.
  - statistical inference,
  - race conditions,
  - heap tracking
  - globals.
  - In some ways, checkers lag much behind our research ones.

How to handle cluelessness?

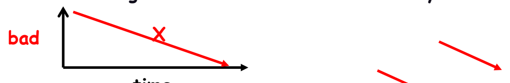
- Can't argue
  - Stupidity works with modular & emotional arithmetic.
- Instead: use normal distributions.
  - Try to get a large meeting. (Schedule before lunch?)



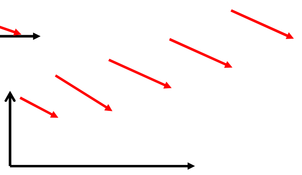
- More people in room = more likely someone in room that:
  - Cares; is very smart; can diagnose error; has been burned by similar error; loses bonus for errors; …
  - Is in another group!
  - If layoffs happen: will be fired(!)

What happens when can't fix all the bugs?

- Rough heuristic:
  - < 1000 bugs? Fix them all.
  - >= 1000? "Baseline"
- Tool improvement viewed as "bad"
  - You are manager. Forall metrics X of badness you want:



No manager gets a bonus for:



How to upgrade when more bugs != good?

- Upgrade cycles:
  - Never. Guaranteed "improvement."
  - Never before release (when could be most crucial)
  - Never before a meeting (at least is rational).
  - Upgrade. Roll back. (~ once per company.)
  - Renew, but don't upgrade. (Not cheap.)
  - Once a year (most large customers). "Rebaseline"
  - Upgrade only checkers where you fix all/most errors.
- People really will complain when your tool gets better.
  - V2.4: 2,400 initial errors. Fixed to get to 1,200
  - Upgrade to V3.5 = 2,600 errors.
  - *MAD* For both reasons.

Do false positives matter? Yes. No. Maybe.

- > 30% false rate = big problem.
  - Ignore tool. Miss true errors amidst false.
  - Low trust = Complex bugs called false positives. Vicious cycle.
  - Caveat: some users accept 70% (or more: security guys).
  - Current deployment threshold = ~20%.
  - Unfortunately: many cases "high FP" rate not analysis problem
- Not all false positives equal:
  - Initial N reports false? "Tool sucks" (N ~ 3?)

  - *Crucial*: no embarrassing FPs.
  - Stupid FP? Implies tool stupid. Not good for credibility.
  - Social: don't want to embarrass tool champions internally

  - Important: no failed merges.
  - Mark FP once? Fine. Reappears & mark again? email support.

Do false negatives matter?

- Of course not! Invisible! Oops:
  - Trial: intentionally put in bugs. "Why didn't you find it?"
  - Easiest sale: horribly burned by specific bug last week. You find it. If you don't?

  - Compare tool A to tool B. If tool A >> B but misses some bugs B finds, then they see A and B as sort-of equiv.
  - Variation: find "enough" of certain type, then don't need more.

  - Upgrade checker: set of defects shifts slightly
  - "Dude, where is my bug?"
  - They know in theory "not a verifier". Different when they actually see you lose known errors. Rule: 5% jitter.
- Currently: favor analysis hacks to remove FPs at cost of FNs

Jitter = bad → Non-determinism = bad.

- Major difference from academia
  - People **really** want the same result from run to run.
  - Even if they changed code base
  - Even if they upgraded tool.
  - Their model = compiler warnings.

  - Classic determism: same input + same function = same result.
  - Customer determinism: different input (modified code base) + different function (tool version) = same result.
- Determinism requirement really sucks.
  - Often tool changes or fixes have very unclear implications.
  - Often randomization = elegant solution to scalability. Can't do.

Caching: Biggest source of non-determinism

- Our hack for handling exponential paths in code
  - If checker reaches same statement in same state assume will produce same result and stop
- Works well for finding many bugs on large code bases.
  - Not so well at finding the *same* bugs
- True story:
  - Version 2.0: follows true path, then false.
  - Version 3.0: follows false path, then true.
  - 20% fluctuation in errors. People went *insane* Soln?
- Bad: don't analyze an interesting path b/c cache hit.
  - The occasional *very* stupid false negative
  - Hurts trust in tool.

  - Lost *huge* sale:
    - found lots of bugs just not this one:

```
if(…)
   x = 0;
for(…)
   switch(…)
   …
w = y / x;
```

DE41

Myth: more analysis is always better

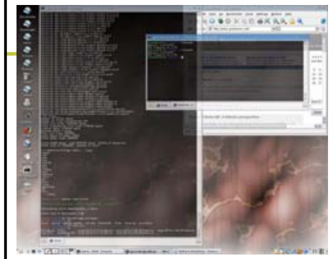
- Does not always improve results, and can make worse
- The best error:
  - **Easy to diagnose**
  - **True error**
- More analysis used, the worse it is for both
  - **More analysis = the harder error is to reason about, since user has to manually emulate each analysis step.**
  - **Number of steps 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 error classes that are too hard to diagnose.**

DE43

No bug is too stupid to check for.

- Someone, somewhere will do anything you can think of.
- Best recent example:
  - **From security patch for bug found by Coverity in X windows that lets almost any local user get root.**

    Got on foxnews (website, not O'Riley)
    So important marketing went to town:



```
if (getuid() != 0 &&
    geteuid == 0) {
       ErrorF("only root");
       exit(1);
 }
```

Since without the parentheses, the code is simply checking to see if the geteuid function in libc was loaded somewhere other than address 0 (which is pretty much guaranteed to be true), it was reporting it was safe to allow risky options for all users, and thus a security hole was born.
- Alan Coopersmith, Sun Developer

[CVE-2006-

First exploit was published 5 hours after the hole was publicly reported

One of the best stupid checks: Deadcode

- Programmer generally intends to do useful work
  - **Flag code where all paths to it are impossible or it makes no sense. Often serious logic bug.**
- From trial at chemotherapy device company
  - **During results meeting: literally ran out to fix**
  - **Note: heavily redacted**

```
enum Tube { TUBE0, TUBE1 };
void PickAndMix(int i) {
  enum Tube tfirst, tlast;

  if (TUBE0 == i) {
    tfirst=TUBE0;
    tlast=TUBE1;
  } else if (TUBE0 == i)  {
    tfirst=TUBE1;
    tlast=TUBE0;
  }
  MixDrugs(tfirst,tlast);
}
```

Overview

- Context
- The banal, vicious laws of reality and its cruel myths
- What actually matters?

- Academics meet reality, good and hard.
  - **The evils of non-determinism**

- Business factoids an academic finds amusing.

Slide 37

DE41 a form of locality: screw up pretty promptly. often going to interprocedural doesn't catch huge more bugs, other than getting wrappers for different functions.

if you don't have ranking, can make results horrible by washing out all the good bugs.
Dawson Engler, 4/18/2006

Slide 38

DE43 in some ways the more stupid the bug the more serious it is: its hard to be extravagantly wacked out in a harmless way.
Dawson Engler, 9/20/2006

Technical can help social

- Tool has simple message: "No touch, low false positives, good bugs"
  - Can explain it to mom? Then can explain to almost all sales guys & customers
  - Complicated? Population that understands much smaller.
  - This effect is not trivial.
- Relationship therapy through tool "objectivity"
  - UK company B outsources to India company A
  - B complains about A's code quality. They fight.
  - Decide to use Coverity as arbitor. Happy. (I still can't believe this.)
- Wide tool use = seismic change in the last ~4 years.
  - People get it. "Static" no longer = "huh?" or "lint" (i.e., suck)
  - Networking effects.
  - Result: Much much much easier to sell tools now.

Some commercial experiences

- Surprise: Sales guys are great
  - Easy to evaluate. Modular.
- Careful what you wish for: bad competitor tools
  - Time to sale ~ max(time for all competitors to do trial).
- Business as an inversion operator:
  - Early on: did 15+ trials across huge company.
  - Much time, effort, manpower.
  - In the end lost a seven figure perpetual license deal.
  - Good or bad?

  - Company X bought license.
  - Next week fired 110 people.
  - Good or bad?

Some useful numbers

- Already seen:
  - 1000: number of bugs after which they baseline.
  - 1.0: probability error labeled as FP if they don't understand
  - -m: slope of bug trend line for manager to get bonus.
  - 20: age of compiler if toolchain change requires recertification.
- Code numbers:
  - 12hr, 24hr: common upper bounds for analysis time
  - 10M: "large" code base
- Bugs:
  - 10-30(?): number of bugs reported where all get fixed.
  - 3: number of attempts you can make to fix a bug in your tool
  - 10: reduction in fix time if you assign blame for bugs
- People:
  - 40: upper bound on opportunities / year sales guy can manage
  - $0: price of initial trial.
  - 6% of revenue from X% of sales?

Laws of static bug finding

- Vacuous tautologies that imply trouble
  - Can't find code, can't check.
  - Can't compile code, can't check.
- A nice, balancing empirical tautology
  - If can find code
  - AND checked system is big
  - AND can compile (enough) of it
  - THEN: will *always* find serious errors.
- A nice special case:
  - Check rule never checked? Always find bugs. Otherwise immediate kneejerk: what wrong with checker???

DE44

How to get into a top (US) PhD program?

- Stanford's current algorithm up to ~2006:
  - Top 1 or 2 at Tsinghua.
  - That's it. (Japan, Korea even worse. India ~ same)
- Why so stupid?
  - Top schools want to know if you can do research
  - Grades, GREs and (often) letters have problems.
  - Careful with answers to the "easy" application questions!
  - Your most reliable method: good paper at top conference.
- Thus: Post 2006:
  - Top 1 or 2 at Tsinghua + (MSR interns w/ papers OR Stanford Master's)
  - Still, we're probably missing someone.
  - Help!

Static vs dynamic bug finding

- Static: precondition = compile (some) code.
  - All paths + don't need to run + easy diagnosis.
  - Low incremental cost per line of code
  - Can get results in an afternoon.
  - 10-100x more bugs.
- Dynamic: precondition = compile all code + run
  - What does code do? How to build? How to run?
  - Pros: on executed paths:
    - Runs code, so can check implications.
    - End-to-end check: all ways to cause crash.
    - Reasonable coverage: surprised when crash.
- Result:
  - Static better at checking properties visible in source, dynamic better at properties implied by source.

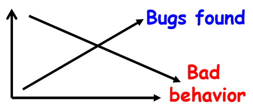
Slide 47

DE44 optimal number of linux bugs to fix. some of best trials led to passes because too effective.
Dawson Engler, 4/18/2006

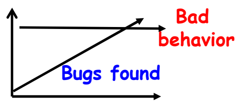
Assertion: Soundness is often a distraction

- 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 time on what gets the next biggest set of bugs
  - Easy experiment: bug counts for sound vs unsound tools.
- Soundness violates 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."

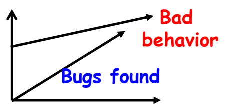
Open Q: Do static tools really help?



The optimistic hope



The null hypothesis



An Ugly Possibility

Danger: Opportunity cost.
Danger: Deterministic canary bugs to non-deterministic.

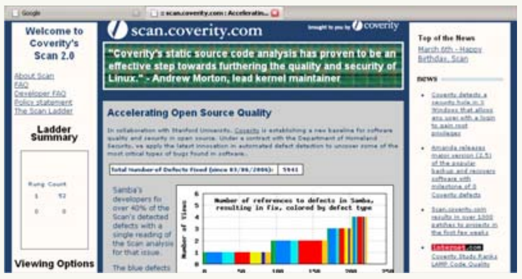
DE30

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
  - 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 (10? or 1?) 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.
  - Possibilities: promote bugs on executed paths or in code people care about, …

Scan's One Year Anniversary

Website Relaunch on March 6th, 2007



Welcome to Coverity's Scan 2.0

About Scan
FAQ
Developer FAQ
Policy statement
The Scan Ladder

Ladder Summary

Viewing Options

scan.coverity.com

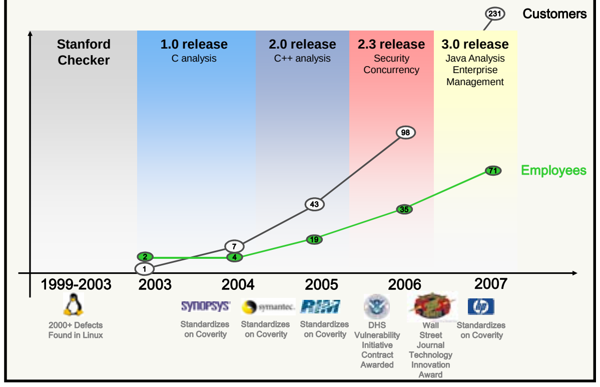
"Coverity's static source code analysis has proven to be an effective step towards furthering the quality and security of Linux." - Andrew Morton, lead kernel maintainer

Accelerating Open Source Quality

Top of the News
March 6th - Happy Birthday, Scan

Number of references to defects in Samba, resulting in fix, colored by defect type
Number of Views

History of Research&Growth of Coverity [2007:outdated]



Slide 49

DE37 Soundness is what you do when you don't have any better ideas.

Once you come up with a new check, there are a million incrementalists that will make it sound if necessary.

Dawson Engler, 2/22/2005

Slide 51

DE30 optimal number of linux bugs to fix. some of best trials led to passes because too effective.
Dawson Engler, 4/18/2006