Statistical Debugging

Ben Liblit, University of Wisconsin–Madison
What’s This All About?

- Statistical Debugging & Cooperative Bug Isolation
  - Observe deployed software in the hands of real end users
  - Build statistical models of success & failure
  - Guide programmers to the root causes of bugs
  - Make software suck less

- Lecture plan
  1. Motivation for post-deployment debugging
  2. Instrumentation and feedback
  3. Statistical modeling and (some) program analysis
  4. Crazy hacks, cool tricks, & practical considerations
Credit Where Credit is Due

- Alex Aiken
- David Andrzejewski
- Piramanayagam Arumuga Nainar
- Ting Chen
- Greg Cooksey
- Evan Driscoll
- Jason Fletchall
- Michael Jordan
- Anne Mulhern
- Garrett Kolpin
- Akash Lal
- Junghee Lim
- Mayur Naik
- Jake Rosin
- Umair Saeed
- Alice Zheng
- Xiaojin Zhu

... and an anonymous cast of thousands!
- Or maybe just hundreds?
- I don’t really know
Motivations: Software Quality in the Real World
“There are no significant bugs in our released software that any significant number of users want fixed.”

Bill Gates, quoted in FOCUS Magazine
A Caricature of Software Development

Requirements

Architecture & Design

Implementation

Testing & Verification

Maintenance

Release!
Software Releases in the Real World

[Disclaimer: this may also be a caricature.]
Software Releases in the Real World

1. Coders & testers in tight feedback loop
   - Detailed monitoring, high repeatability
   - Testing approximates reality

2. Testers & management declare “Ship it!”
   - Perfection is not an option
   - Developers don’t decide when to ship
Software Releases in the Real World

3. Everyone goes on vacation
   - Congratulate yourselves on a job well done!
   - What could possibly go wrong?

4. Upon return, hide from tech support
   - Much can go wrong, and you know it
   - Users define reality, and it’s not pretty
     - Where “not pretty” means “badly approximated by testing”
Testing as Approximation of Reality

- Microsoft’s Watson error reporting system
  - Crash reports from 500,000 separate programs
  - $\chi\%$ of software errors cause 50% of user crashes
  - Care to guess what $\chi$ is?

- 1% of software errors cause 50% of user crashes

- Small mismatch $\Rightarrow$ big problems (sometimes)

- Big mismatch $\Rightarrow$ small problems? (sometimes!)
  - Perfection is usually not an economically viable option
Always One More Bug

- Imperfect world with imperfect software
  - Ship with known bugs
  - Users find new bugs

- Bug fixing is a matter of triage + guesswork
  - Limited resources: time, money, people
  - Little or no systematic feedback from field

- Our goal: reality-directed debugging
  - Fix bugs that afflict many users
The Good News: Users Can Help

- Important bugs happen often, to many users
  - User communities are big and growing fast
  - User runs ➞ testing runs
  - Users are networked

- We *can* do better, with help from users!
  - Users *know* define what bugs matter most

- Common gripe: “Software companies treat their users like beta testers”
  - OK, let’s make them *better* beta testers
Measure Reality and Respond

- Software quality as an *empirical* science
  - Observed trends rather than absolute proofs
  - Biologists do pretty well, even without source code

- Observational science requires … observation!
  - 7,600 Ad-Aware 2007 downloads during today’s lecture
  - 500,000,000 Halo 2 games in 20 months
  - Plenty to observe, provided we can get at the data
Bug and Crash Reporting Systems

- **Snapshot of Mozilla’s Bugzilla bug database**
  - Entire history of Mozilla; all products and versions
  - 60,866 open bug reports
  - 109,756 additional reports marked as duplicates

- **Snapshot of Mozilla’s Talkback crash reporter**
  - Firefox 2.0.0.4 for the last ten days
  - 101,812 unique users
  - 183,066 crash reports
  - 6,736,697 hours of user-driven “testing”
Real Engineers Measure Things; Are Software Engineers Real Engineers?
Real Engineering Constraints

- Millions of lines of code
- Loose semantics of buggy programs
- Limited performance overhead
- Limited disk, network bandwidth
- Incomplete & inconsistent information
- Mix of controlled, uncontrolled code
- Threads
- Privacy and security
High-Level Approach

1. Guess “potentially interesting” behaviors
   - Compile-time instrumentation

2. Collect sparse, fair subset of complete info
   - Generic sampling transformation
   - Feedback profile + outcome label

3. Find behavioral changes in good/bad runs
   - Statistical debugging
Instrumentation Framework
“The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong, it usually turns out to be impossible to get at or repair.”

Douglas Adams, *Mostly Harmless*
Bug Isolation Architecture

Program Source → Predicates → Sampler → Compiler → Shipping Application

Top bugs with likely causes → Statistical Debugging

Counts & 🙃/😄
Our Model of Behavior

Each behavior is expressed as a predicate $P$ on program state at a particular program point.

Count how often “$P$ observed true” and “$P$ observed” using sparse but fair random samples of complete behavior.
Predicate Injection: Guessing What’s Interesting

Program Source ➔ Predicates ➔ Sampler ➔ Compiler ➔ Shipping Application

Top bugs with likely causes ➔ Statistical Debugging ➔ Counts & 😊/🚫

People with 🙄/😊 icons
Branch Predicates Are Interesting

if (p)
    ...
else
    ...

Branch Predicate Counts

if (p)
    // p was true (nonzero)
else
    // p was false (zero)

- Syntax yields instrumentation site
- Site yields predicates on program behavior
- Exactly one predicate true per visit to site
Returned Values Are Interesting

n = fprintf(...);

- Did you know that `fprintf()` returns a value?
- Do you know what the return value means?
- Do you remember to check it?
Returned Value Predicate Counts

\[ n = \text{fprintf}(\ldots); \]

// return value < 0 ?
// return value == 0 ?
// return value > 0 ?

- Syntax yields instrumentation site
- Site yields predicates on program behavior
- Exactly one predicate true per visit to site
Pair Relationships Are Interesting

```c
int i, j, k;
...
i = ...;
```
Pair Relationship Predicate Counts

```c
int i, j, k;
...
i = ...;

// compare new value of i with...
// other vars: j, k, ...
// old value of i
// “important” constants
```
Many Other Behaviors of Interest

- Assert statements
  - Perhaps automatically introduced, e.g. by CCured

- Unusual floating point values
  - Did you know there are nine kinds?

- Coverage of modules, functions, basic blocks, …

- Reference counts: negative, zero, positive, invalid
  - I use the GNOME desktop, but it terrifies me!

- Kinds of pointer: stack, heap, null, …

- Temporal relationships: x before/after y

- More ideas? Toss them all into the mix!
Summarization and Reporting

- Observation stream ➔ observation count
  - How often is each predicate observed true?
  - Removes time dimension, for good or ill

- Bump exactly one counter per observation
  - Infer additional predicates (e.g. ≤, ≠, ≥) offline

- Feedback report is:
  1. Vector of predicate counters
  2. Success/failure outcome label

- Still quite a lot to measure
  - What about performance?
Fair Sampling Transformation

Program Source

Predicates

Sampler

Compiler

Shipping Application

Top bugs with likely causes

Statistical Debugging

Counts & 😊/ ipt
Sampling the Bernoulli Way

- Decide to examine or ignore each site...
  - Randomly
  - Independently
  - Dynamically

- Cannot use clock interrupt: no context
- Cannot be periodic: unfair temporal aliasing
- Cannot toss coin at each site: too slow
Amortized Coin Tossing

- Randomized global countdown
  - Small countdown ➔ upcoming sample

- Selected from geometric distribution
  - Inter-arrival time for biased coin toss
  - How many tails before next head?
  - Mean sampling rate is tunable parameter
Geometric Distribution

\[ \text{next} = \left\lfloor \frac{\log(\text{rand}(0,1))}{\log(1 - \frac{1}{D})} \right\rfloor + 1 \]

- \( D \) = mean of distribution
  - = expected sample density
Weighing Acyclic Regions

- Break CFG into acyclic regions
- Each region has:
  - Finite number of paths
  - Finite max number of instrumentation sites
- Compute max weight in bottom-up pass
Weighing Acyclic Regions

- Clone acyclic regions
  - “Fast” variant
  - “Slow” variant
- Choose at run time
- Retain decrements on fast path for now
  - Stay tuned…
Optimizations I

- Identify and ignore “weightless” functions
- Identify and ignore “weightless” cycles
- Cache global countdown in local variable
  - Global $\rightarrow$ local at function entry & after each call
  - Local $\rightarrow$ global at function exit & before each call
Optimizations II

- Avoid cloning
  - Instrumentation-free prefix or suffix
  - Weightless or singleton regions
- Static branch prediction at region heads
- Partition sites among several binaries
- Many additional possibilities…
Path Balancing Optimization

- Decrement on fast path are a bummer
  - Goal: batch them up
  - But some paths are shorter than others
- Idea: add extra “ghost” instrumentation sites
  - Pad out shorter paths
  - All paths now equal
Path Balancing Optimization

- Fast path is faster
  - One bulk counter decrement on entry
  - Instrumentation sites have no code at all

- Slow path is slower
  - More decrements

- Consume more randomness
Variations on Next-Sample Countdown

- **Fixed reset value**
  - Biased, but useful for benchmarking

- **Skip sampling transformation entirely**
  - Observe every site every time
  - Used for controlled, in-house experiments
  - Can simulate arbitrary sampling rates offline

- **Non-uniform sampling**
  - Decrement countdown more than once
  - Multiple countdowns at different rates
What Does This Give Us?

- Absolutely certain of what we do see
  - Subset of dynamic behavior
  - Success/failure label for entire run

- Uncertain of what we don’t see

- Given enough runs, samples $\approx$ reality
  - Common events seen most often
  - Rare events seen at proportionate rate
Statistical Debugging Basics
“What is luck?
Luck is probability taken personally. It is the excitement of bad math.”

Penn Jillette
Playing the Numbers Game

- Program Source
- Predicates
- Sampler
- Compiler
- Shipping Application
- Statistical Debugging
- Counts
- Top bugs with likely causes
Find Causes of Bugs

- Gather information about *many* predicates
  - 298,482 predicates in bc
  - 857,384 predicates in Rhythmbox
- Vast majority not related to any particular bug 😞
- How do we find the useful *bug predictors*?
  - Data is incomplete, noisy, irreproducible, …
Sharing the Cost of Assertions

- What to sample: `assert()` statements
- Look for assertions which sometimes fail on bad runs, but always succeed on good runs
- Overhead in assertion-dense CCured code
  - Unconditional: 55% average, 181% max
  - $1/_{100}$ sampling: 17% average, 46% max
  - $1/_{1000}$ sampling: 10% average, 26% max
Isolating a Deterministic Bug

- Hunt for crashing bug in `crypt-1.2`
- Sample function return values
  - Triple of counters per call site: $< 0, == 0, > 0$
- Use process of elimination
  - Look for predicates true on some bad runs, but never true on any good run
Winnowing Down the Culprits

- 1710 counters
  - 3 x 570 call sites
- 1569 zero on all runs
  - 141 remain
- 139 nonzero on at least one successful run
- Not much left!
  - `file_exists() > 0`
  - `xreadline() == 0`