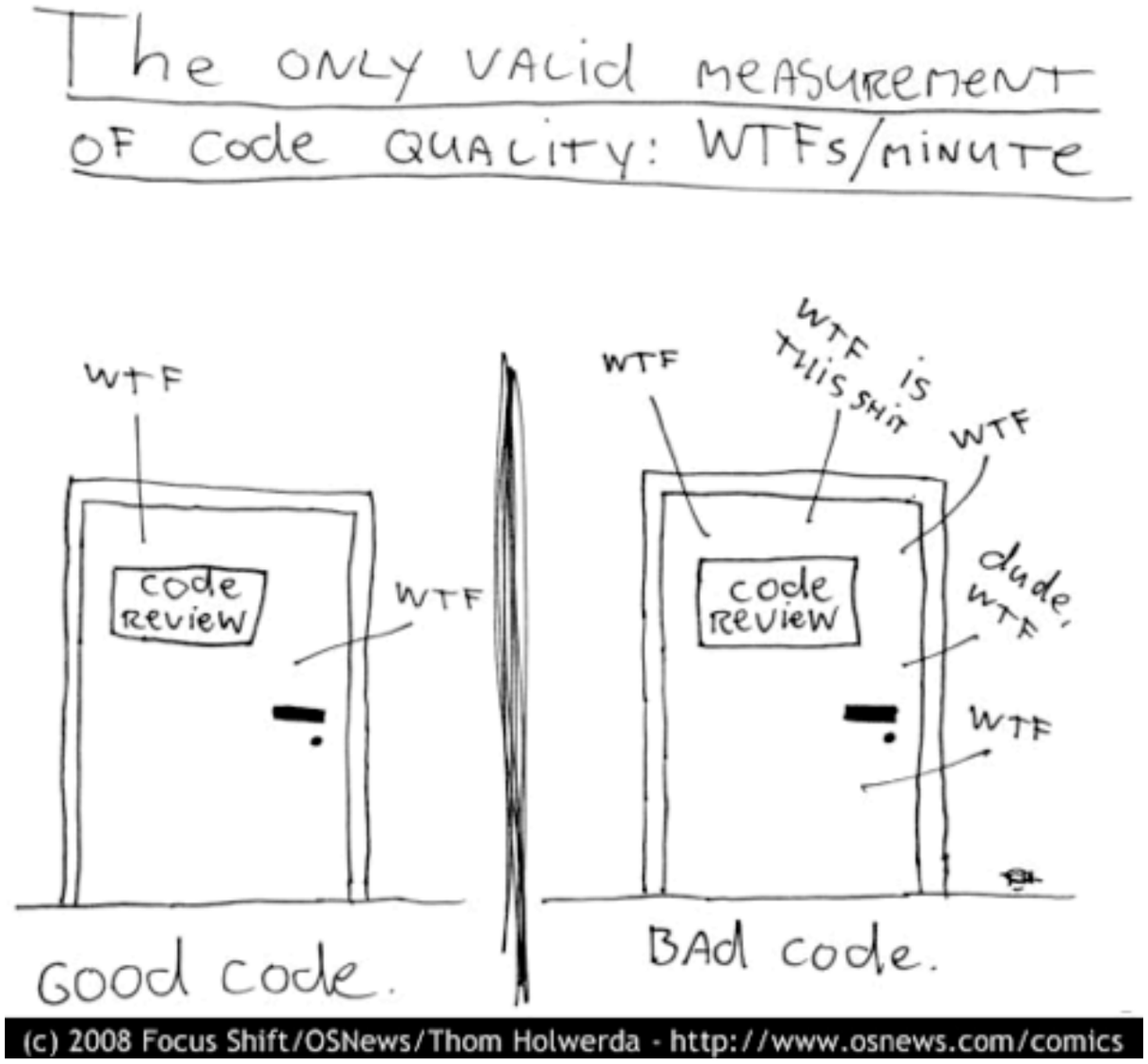
Academic researchers should listen to their customers



William Pugh
Univ. of Maryland

Learning From Experience

FindBugs

- Static analysis tool
  - Tries to find coding mistakes in Java code
- Driven by real mistakes
  - use the simplest possible analysis that effectively finds the mistake
- More than one million downloads

FindBugs @ Google

- Google has been attempting to systematically use FindBugs since Summer 2006
  - Still not doing so
- Bugbot effort started Summer 2006
  - Google killed all static analysis effort in October 2008 (two months after start of my sabbatical)
- Successful FindBugs fixit May 2009
  - still born; technology not pushed into deployment

Google's decision was rational

- I came to support Google's decision
  - Google was trying to optimize software development effectiveness
- I wasn't paying enough attention to the needs of my customers
  - are you solving their important problems?
  - Or do you have a hammer and are you just looking for nails?

Software is imperfect

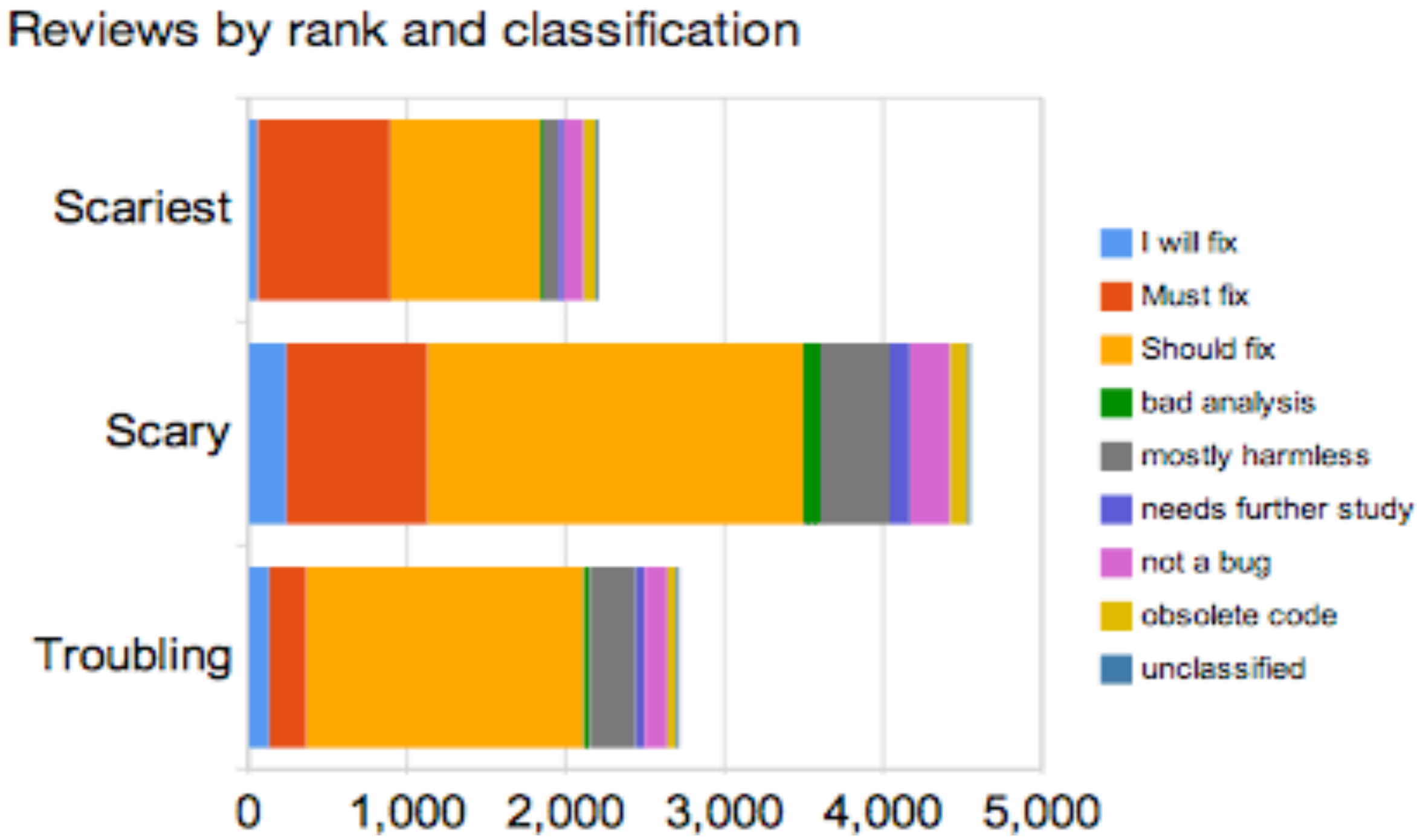
- Software will always be imperfect
  - some imperfections are more important than others
- Developers are always busy
  - developer effort is zero sum
- For each defect, there are multiple ways to detect and remove the mistake
  - Is your way the most cost effective way?

The Google FindBugs fixit

Google fixit

- Held May 13-14, 2009
- Reported 3,800 issues on Google's Java code
- 700 engineers viewed some issues FindBugs
  - 280 classified issues
- 9,364 classifications
  - 81% must fix or should fix

Classifications



Interpretation

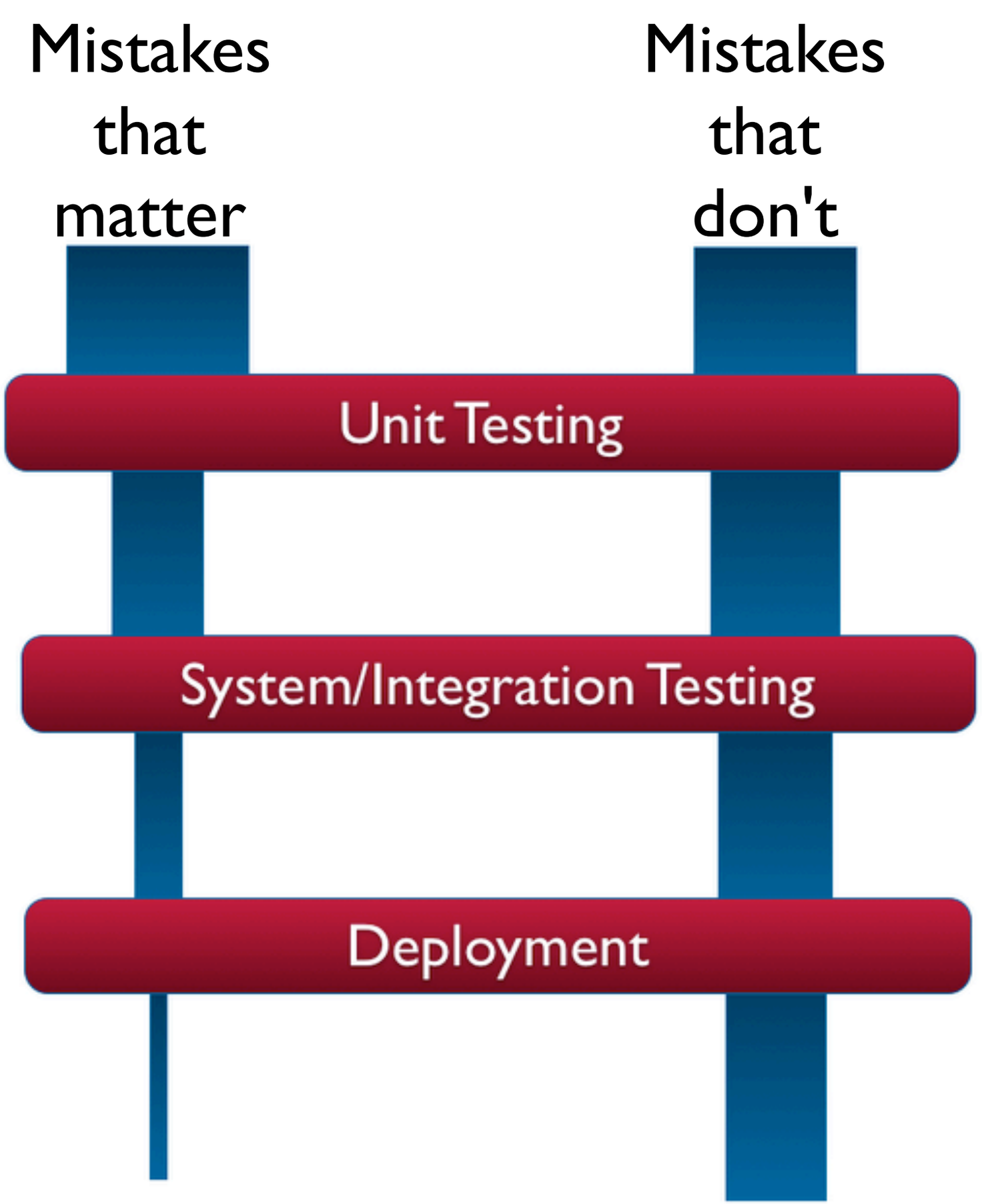
- Very good at finding issues engineers wanted to see
  - if noticed at code review time, would have been fixed
    - Managers were surprised at how high the "should fix" rates were
      - static analysis had a bad reputation at Google.
- Very efficient to review issues
  - roughly 30-90 seconds per issue

Mistakes that matter

- We weren't successful at identifying bugs that were reported as having significant impact in production
  - FindBugs sometimes works better as a detector of unused and untested code than as a detector of defects with impact in production
- Why?

Mistakes, faults and failures

- Some developers consider a mistake to matter only if it causes software to misbehave
- Some mistakes cause misbehavior, some don't
- Some code is unreached/unreachable
- Some code isn't used yet
  - and we don't know if we will wind up using it
- Some projects are abandoned/unused
- Some mistakes simply reflect an inconsistency in the design
- Some mistakes cause incorrect execution, but don't result in incorrect application behavior


Mistakes that matter
Mistakes that don't
Unit Testing
System/Integration Testing
Deployment

Scariest bugs

- Bugs that silently result in the wrong answer being computed
  - Doesn't throw an exception
- Often hard to see at code review time
  - Might be going wrong right now in production
  - or when it does go wrong, fault becomes obvious only significantly downstream of the mistake

Google fixit results

But some mistakes do matter

- Buganizer entry XXXXX
  - refactoring introduced bug
    - depending on IDE to show where cleanup was needed
      - one place wasn't caught by IDE
- Code was changed May 13th (first day of fixit), outside of normal release cycle
  - compiled into XXX to test new functionality
  - XXX picked up by all gcronned mapreduces, caused one to generate bad data (May 14th)
  - caught fairly quickly, XXX and CL rolled back
- Caught both by static analysis and by internal deployment

A scary bug, caught in preproduction

XXXAcountingServiceImpl bookRevenue(...)

```
// calculate DR amount by aggregating CR amounts
BigDecimal drAmount = new BigDecimal(0);
for (JournalEntry je: journalEntries)
   drAmount.add(je.getCrAmount());
// persist to db
getXXXTrxnService().saveJournalEntry(gc,
    createJournalEntry(sobId, ...
      drAmount,  // aggregated amount
      true, // Debit
     "USD",
     "Revenue"));
```

More investigation

- Code added in CL that added 4 new files, 1094 lines of code
  - Code added 5/28/2009
  - No code review comments or unit tests
  - not in production yet
- Buganizer issue XXXX
  - Fixed within 30 minutes of being reported

More bugs...

Recent bug in Trix

- Buganizer issue XXXXXX

```
if (!resultCell.getValue().equals(
        resultCell.getPreviousValue())) {
  return resultCell;
```

- getValue() returns a String
- getPreviousValue() returns a Value

- Broke Trix functionality for strict validation
- Introduced June 15th, deployed to corp, not yet pushed to production, not caught by tests
- Fixed within 30 minutes of being reported

Another bug

- Constructor with parameters with same name as fields
  - assignments of the form this.foobar = foobar;
    - except that one of the parameter names was misspelled
    - assignment read field and assigned value of field to field, ignoring parameter
- Reported to developers, first reaction was “oh shit, how many customers are we going to have to contact?”

Not a problem

- Class in question had two constructors
- The faulty constructor was only called from test code
  - tests only passed the value 0 for the parameter, the fact that the parameter was ignored wasn't observable
- The constructor called in production was fine

Overhead of reviewing new issues

- Scariest (rank 1-4):
  - 4 per day total, 350 over 4 months
  - one per month in gmail, 5 over 4 months

- All issues considered for fixit (rank 1-12):
  - 40 per day total, 4,000 over 4 months
  - one per day for gmail, 90 issues over 4 months

Code monkeys and bananas

- Most projects have a fairly small set of issues, and most days don't see any new rank 1-12 issues
  - Can't tell developers to *run* FindBugs
  - If most of the time they pull the lever, no banana appears, they soon stop pulling the lever

- Once set up, needs to run transparently, in a way that the developer doesn't notice when no issues are found

Unexpected difficulty

- Hard to determine which code is deployed and executed in production
- Hard to determine which bug tracker component to file against
  - hard to determine which engineer to assign to bug

Improving software quality

- Many different things can catch mistakes and/or improve software quality
  - Each technique more efficient at finding some mistakes than others
  - Each subject to diminishing returns
  - No magic bullet
  - Find the right combination for you and for the mistakes that matter to you

Test, test, test...

- Many times FindBugs will identify bugs
  - that leave you thinking "Did anyone test this code?"
    - And you find other mistakes in the same vicinity
      - FindBugs might be more useful as an untested code detector than as a bug detector
- Overall, testing is far more valuable than static analysis
  - I'm agnostic on unit tests vs. system tests
  - But *no one* writes code so good you don't need to check that it does the right thing
    - I've learned this from personal painful experience

Dead code

- Many projects contain lots of dead code
  - abandoned packages and classes
  - classes that implement 12 methods; only 3 are used
- Code coverage is a very useful tool
  - but pushing to very high code coverage may not be worthwhile
  - you'd have to cover lots of code that never gets executed in production

Dead code at Google

- The vast majority of code at Google is in one source code repository
  - most of the java code is in two source trees
    - java/com/google/...
    - javatests/com/google/...
- Contains projects that have been killed, and projects that were abandoned before they were ever launched

Code coverage from production

- If you can sample code coverage from production, great
  - look for code executed in production but not covered in unit or system test

Cool idea

- If you can't get code coverage from production
- Just get list of loaded classes
  - just your code, ignoring classes loaded from core classes or libraries
  - Very light weight instrumentation
- Log the data
  - could then ask queries such as "Which web services loaded the **`FooBar`** class this month?"

Leveraging class initialization logging

- You've got class initialization logging
- But want to know if a particular method or statement is reached
- Define a nested class with a static method with an empty body

```
static class Foo {
   static void loadClass() {};
   }
```

Listening to your customers

Listening to your customers

- Bugbot system, developed at Google Summer 2006-October 2008
  - Very few users
    - Usage not tracked
  - Painfully slow and difficult to use
- I was focused on making the results more accurate, and finding new important errors

Pain points

- While we were finding defective code
  - we weren't finding important defects
- Other techniques (testing, etc) were finding important mistakes
  - Google had a flat headcount in Fall 2008, other software quality efforts needed engineers

FindBugs fixit infrastructure

- Allowed mass evaluation of FindBugs
  - managers thrilled with high “should fix” rate
  - but lack of important bugs found requires a lower cost deployment
- 1 minute per day per developer is *very* expensive
  - cost to run analysis when nothing is found

Status at Google

- Working to make static analysis results available during code review
  - code review is required for all commits
- Will happen for all commits, no action required by developer
  - Also include test results, code coverage, etc

Doing it at code review time

- Harder than it seems
  - original code review system didn't require compilable snapshot
- Asynchronous
  - code review doesn't wait on analysis results

Your customer's important bugs

Which bugs matter to a developer?

- Love to get better information about important bugs, and what might have been done to find them cheaply
- Work from real bugs, real developers

Customer bugs

- Are there general issues that need better solutions
  - e.g., data races
- Or are the important bugs specific to a particular group or API
  - e.g., rules about using internal GMail APIs
  - How could we allow the GMail team to write and deploy detectors for those bugs
- Is there a long tail that means a small number of patterns can't catch a significant number of issues

Moral

- Get a customer
- Understand their needs and pain points
- Pick a problem of theirs you are going to solve

Questions?