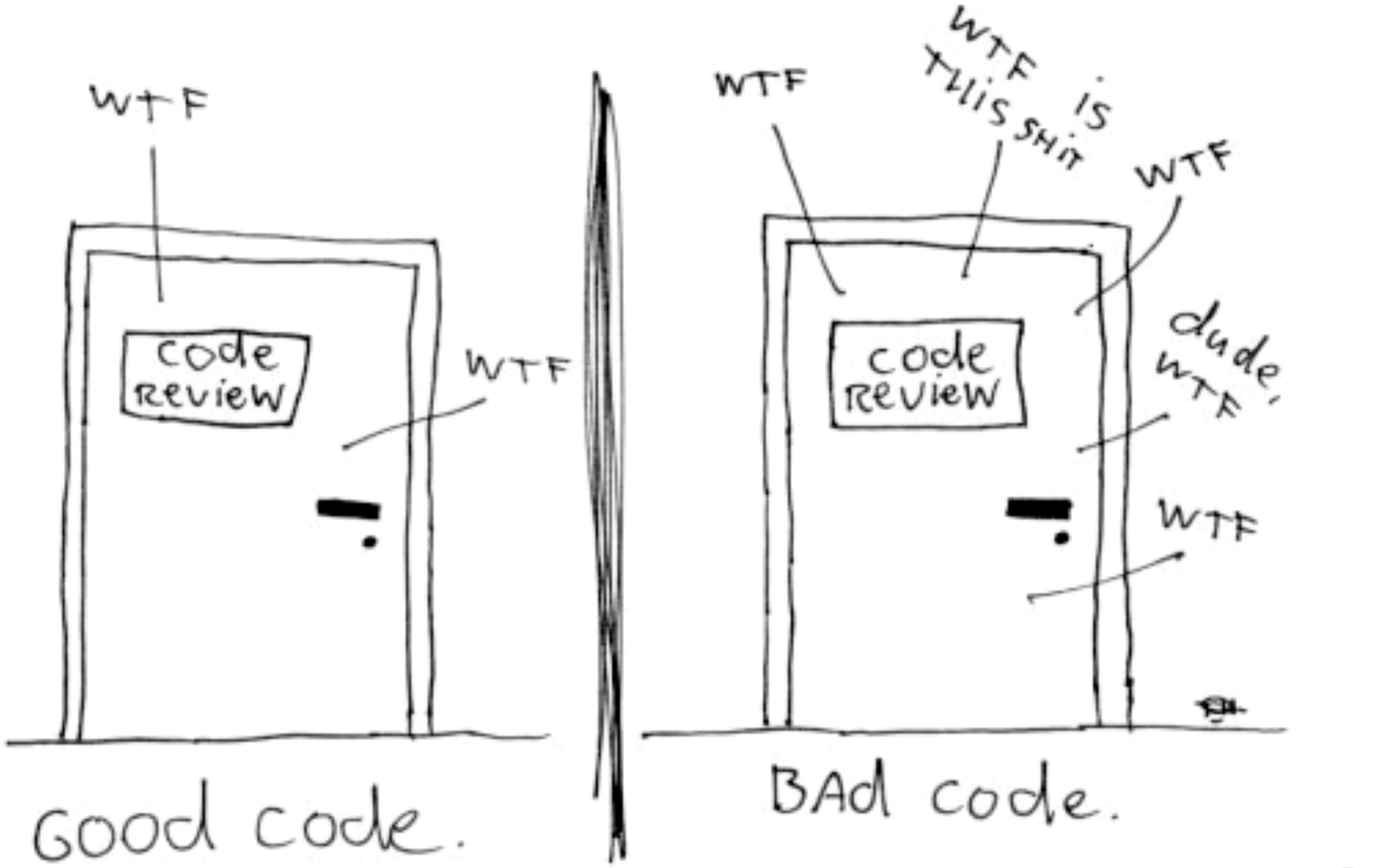
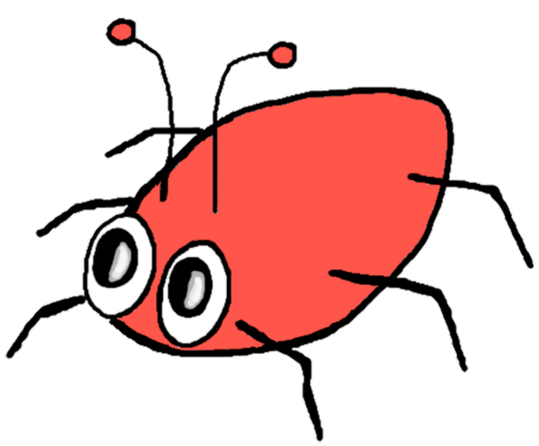
Effective use of FindBugs in large software development efforts

The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE



CodeMash 2012

William Pugh



Code has bugs

- no perfect correctness or security
- you shouldn't try to fix everything that is wrong with your code
- engineering effort is limited and zero sum
- how can you get the best return on the investment of engineering time using FindBugs

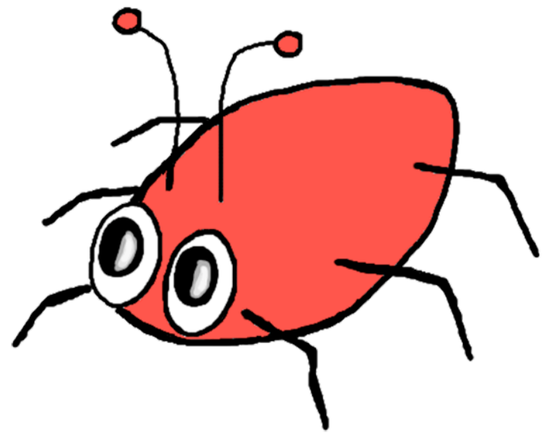
Defective Java Code

Learning from mistakes

- I'm the lead on FindBugs
  - static analysis tool for defect detection
    - more than a million downloads
- Spent a lot of time at Google
  - Found thousands of errors
    - not style issues, honest to god coding mistakes
    - but mistakes found weren't causing problems in production

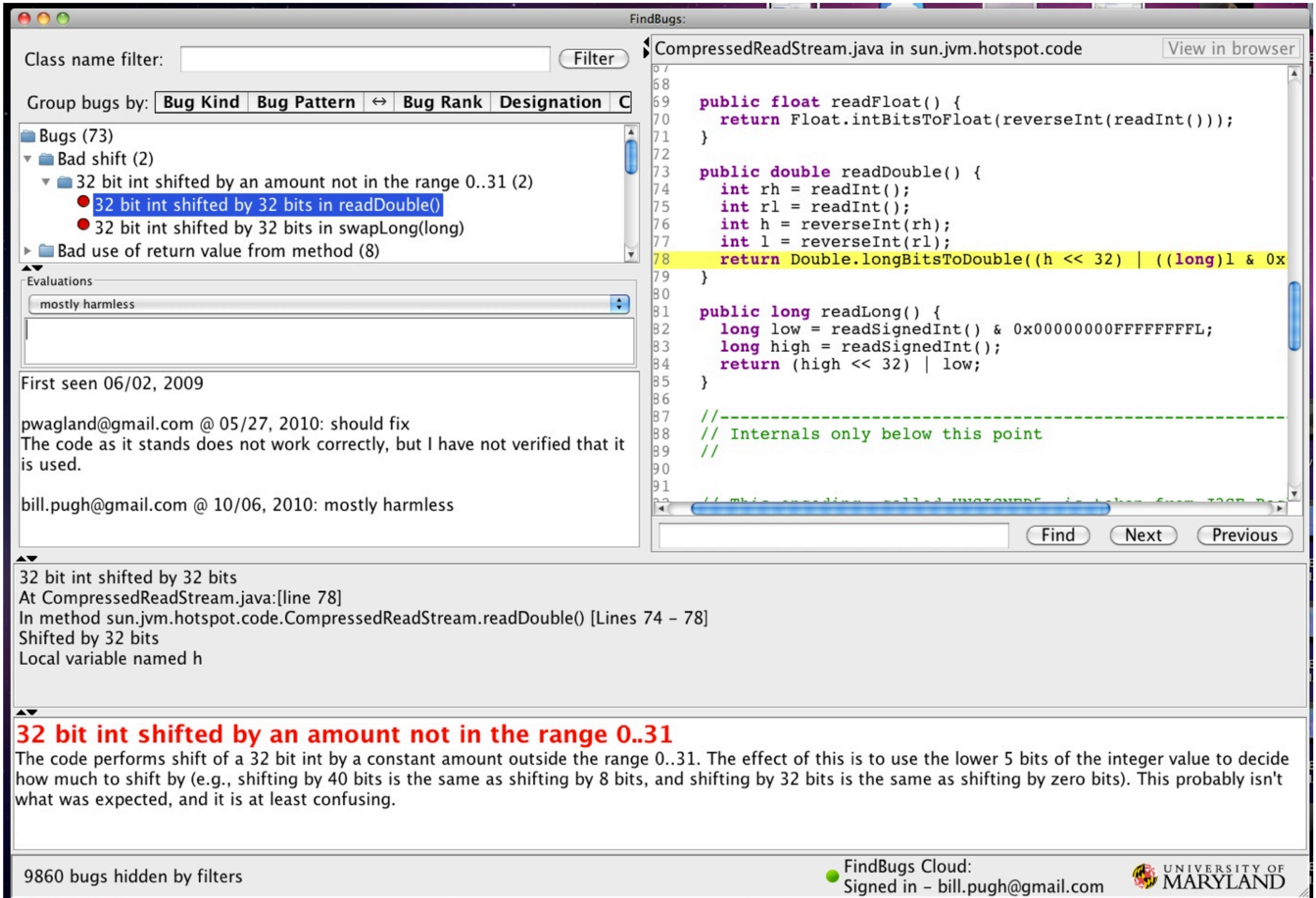
FindBugs fixit @ Google
May 2009

- 4,000 issues to review
  - Bug patterns most relevant to Google
- 8,000 reviews
  - 81+% must/should fix
  - many issues independently reviewed by multiple engineers



> 1,800 bugs filed
  > more than 600 fixed
> More than 1,500 issues removed in several days

FindBugs demo



FindBugs web start

- Go to http://findbugs.sourceforge.net/findbugs2.html
- Click on one of the links for communal reviews of FindBugs issues

Learned wisdom

- Static analysis typically finds mistakes (often just inconsistencies)
  - but some mistakes don't matter
  - need to find the intersection of stupid and important
- The bug that *matter* depend on context
- Static analysis, *at best*, might catch 5-10% of your software quality problems
  - 80+% for certain specific defects
  - but overall, not a magic bullet
- Used effectively, static analysis is cheaper than other techniques for catching the same bugs

What is wrong with this code?

Eclipse 3.7

org.eclipse.update.internal.ui.views.FeaturesStateAction

```
public void run() {
  try {
    if ((adapters == null) && (adapters.length == 0))
      return;
    IStatus status
      = OperationsManager
        .getValidator()
        .validatePlatformConfigValid();
    if (status != null)
      throw new CoreException(status);
    ...
```

What is wrong?

- Definitely no test cases for when `adapters` is null
- Probably no test cases for when `adapters` is empty
- Need to replace
  `(adapters == null) && (adapters.length == 0)`
  with
  `(adapters == null) || (adapters.length == 0)`

Effective use of a static analysis tool

- Tune it to report only the kinds of issues you care about
- Run it automatically, alerting you when new serious issues are found
- Deal with issues where you don't want to change the code
- Figure out how to deal to legacy bugs: clear mistakes that have existed in the code for a long time

What bugs matter to you?

- If you have a public static final field pointing to an array
  - anyone can change the contains of the array
- A big concern if you are concerned about untrusted code running in the same VM
  - a minor concern otherwise
- Are you concerned about internationalization, character encodings, etc?
  - lots of issues here, only matters in some applications

Compiler warnings

- compiler warnings are a similar issue
- At Google, they've spent some time thinking about the compiler warnings they care about
- Try to fix the ones they care about, globally disable the ones they don't care about

Running it automatically

- Most changes don't introduce serious new issues detected by FindBugs (probably less than 2%)
- You don't want developers to have to think about running it, or be blocked while it is running
  - their time and focus is too valuable; too little return
- But, some of the mistakes caught will cause developers to go on a frustrating hours long debugging hunt

How?

- Need better IDE integration
  - we've got some work to do here
- Need a way to know which issues are new and scary
- Run at unit test time, or at continuous build time
  - ... need to write a shim for launching it from a unit test...

Dealing with issues where you don't want to change the code

- FindBugs is very accurate, certainly compared to many other tools
  - For rank 1-12 issues, Google engineers said they were "should fix" 81% of the time
- But sometimes, the warning doesn't inspire you to want to change the code
- We have 55 such issues in the FindBugs code base
  - only 10 of them at rank 1-18

Dealing with “not a bug”

- Put an annotation in the source code
  - Careful: annotations can suppress future issues that shouldn’t be suppressed
  - In many circumstances, resistance to changing source code to suppress issues
- Store issues and evaluations in a central database
  - used by every major commercial static analysis tool

legacy bugs

- Understand whether the code is being executed now, and whether the buggy behavior is occurring now
  - code coverage from production?
- If the code isn't being executed, consider just deleting the code, or adding logging if it ever does get executed
- If you want to fix it, figure out the right behavior and write a test case to document it
  - *then* fix it

Maybe you shouldn't fix all old issues

- If a mistake was written into your code two years ago, and it hasn't caused any problems, maybe you shouldn't fix it.
- Probably no test cases, code may not be used or understood
- Changing the code to silence the warning without really understanding the code or having any test cases is dangerous
    - it just removes the WTF from the code.

Bug fix regressions

- Whenever you try to fix a bug, there is a chance that you will won't do so correctly
- might make things worse, or only partially fix the problem
- Estimates of incomplete/bad bug fixes range from 5-30%

Important concepts in FindBugs

- Ways to run FindBugs
- Bug attributes:
  - confidence, rank, category, kind, pattern
- Ways to filter and rank bugs
- Baseline bugs
- Bug clouds
- plugins

Running FindBugs

- Works on JVM classfiles
  - Some detectors produce poor results for some non-Java languages, such as Scala
- Runs on command line, ant, maven, Eclipse, Netbeans, IntelliJ, Jenkins, Fortify, Coverity,

Bug attributes

- Each bug is an instance of a pattern
  - patterns are groups by category (e.g., internationalization) and kind (e.g., null pointer dereference)
- Each instance has a confidence (low, medium high)
  - priority in previous versions of FindBugs, but this confused people because priorities weren't comparable between different bug patterns

BugRank

- Each instance has a rank 1-20, with 1 being scariest
  - Scariest: rank 1-4
  - Scary: rank 5-9
  - troubling: rank 10-14
  - of concern: rank 15-20
- Scariest are issues most likely to cause significant and stealthy changes in behavior
  - roughly corresponds to the OMG level

Customizing bug rank

- Bug ranks can be and should be customized for production deployments
- can create a plugin that contains a bugrank.txt file, and add plugin to your deployement or project

Filtering Bugs

- You can filter bugs using either options to a command-line or ant task, or via a filter file
- Filter files can involve more complicated logic, including things such as "filter warnings of type X if they involving invoking method Y"
- Filters can be put into a plugin

Baseline bugs

- Easy way to show just new bugs
- Filter a bug report, excluding issues that are already present in another bug report
- Allows you to say: show me just the issues that weren't in the previous release

Comparing bugs across versions

- FindBugs using techniques that use the bug pattern, class, method, and other components of the issue to identify when two different analysis reports contain the same issue
- it is confused by refactorings such as class and method renaming

Bug clouds

- Previously, we had provided a way for you to store evaluations of issues in the XML used to store the analysis results
  - but it was very hard to share results among a team
- We now provide bug clouds, where we store information about the first time an issue was seen, and any evaluations of the issue

Which bug cloud?

- We provide a free bug cloud, hosted on Google app engine, suitable for use on open source or other non-confidential projects
  - people have to sign in using open-id before anything is stored there.
- You can set up your own bug cloud on your own servers
  - At the moment, requires making some changes to the distro and rebuilding, should soon be possible to configure as separate plugin

Plugins

- FindBugs has had plugins for a long time, but we've really added lots of features
- A plugin might just consist of some xml files specifying various properties
- Plugins are loaded from the findbugs installation directory and from a .findbugs directory in the user's home directory
- in both, looks in subdirections plugin and optionalPlugin

Enabling plugins

- Plugins loaded from a plugin directory are enabled by default
  - those loaded from optionalPlugin are not
- You can set which plugins are enabled for a particular project

Some privacy and confidentiality issues

FindBugs update check

- FindBugs does an update check to see if there is a new version of FindBugs
  - doesn't report anything about the code being analyzed
  - but does report things like OS, Java version, locale, invocation mechanism (Ant, Maven, command line, GUI)
- You can install a plugin that completely blocks this check, or write your own plugin that reroutes the

FindBugs communal cloud

- We are hosting a free server to record information about bugs
  - when the bug was first seen, and any evaluations of the issue by developers.
    - e.g., "On Jan 11th, Sam marked this as a "Should Fix" issue and said "...."
- Appropriate for open source and other non-confidential source code

FindBugs communal cloud privacy

- Source code is never uploaded
- You have to select the “FindBugs Communal Cloud”, and log in with an open-id account, before anything is uploaded into the cloud
- You can remove the FindBugs communal cloud from your configuration if you are concerned

Defect density

- For Eclipse 3.0 (fairly typical)
  - Scariest: 30 per million
  - Scary: 160 per million LOC
  - Troubling: 480 per million LOC
  - Of concern: 6000 per million LOC

Understand your risk/bug environment

- What are the expensive risks?
- Is it OK to just pop up an error message for one web request or GUI event?
  - how do you ensure you don't show the fail whale to everyone?
- Could a failure destroy equipment, leak or loose sensitive/valuable data, kill people?

mistakes charactertistics

- Will you know quickly if it manifests itself?
- What techniques are good for finding it?
  - Is unit testing effective?
- Might a change in circumstances cause it to start manifesting itself?
- What is the cost of it manifesting itself?
- If is does manifest itself, will it come on slowly or in a tidal wave

Bugs in Google's code

- Google's code base contains thousands of "serious" errors
  - code that could *never* function in the way the developer intended
    - If noticed during code review, would definitely have been fixed
    - Most of the issues found by looking at Google's entire codebase have been there for months or years
- despite efforts, unable to find any causing noticeable problems in production

As issues/bugs age

- go up:
  - cost of understanding potential issues, deciding if they are bugs
  - cost and risk of changing code to remedy bugs
- goes down:
  - chance that bug will manifest itself as misbehavior

More efficient to look at issues early

- be prepared for disappointment when you look at old issues
- may not find many serious issues
- don't be too eager to "fix" all the old issues

Where bugs live

- code that is never tested
  - If code isn't unit or system tested, it probably doesn't work
- `throw new UnsupportedOperationException()` is vastly underrated
- if your current functionality doesn't need an equals method, and you don't want to write unit tests for it, make it throw `UnsupportedOperationException`
- Particularly an issue when you implement an interface with 12 methods, and your current use case only needs 2

Bug manifestation

- You can have an error in your code that *never* manifests itself
  - Perhaps the code is never executed
  - Or the code is never executed in the state required for the bug to manifest itself
- But if it does manifest itself, how does it manifest itself?

Bug manifestations

- Always throws a Runtime exception
- Sometimes/rarely throws a runtime exception
  - either probabilistically, or based on data
- Silently computes the wrong value
- Sometimes computes the wrong value

Improving software quality

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

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?"

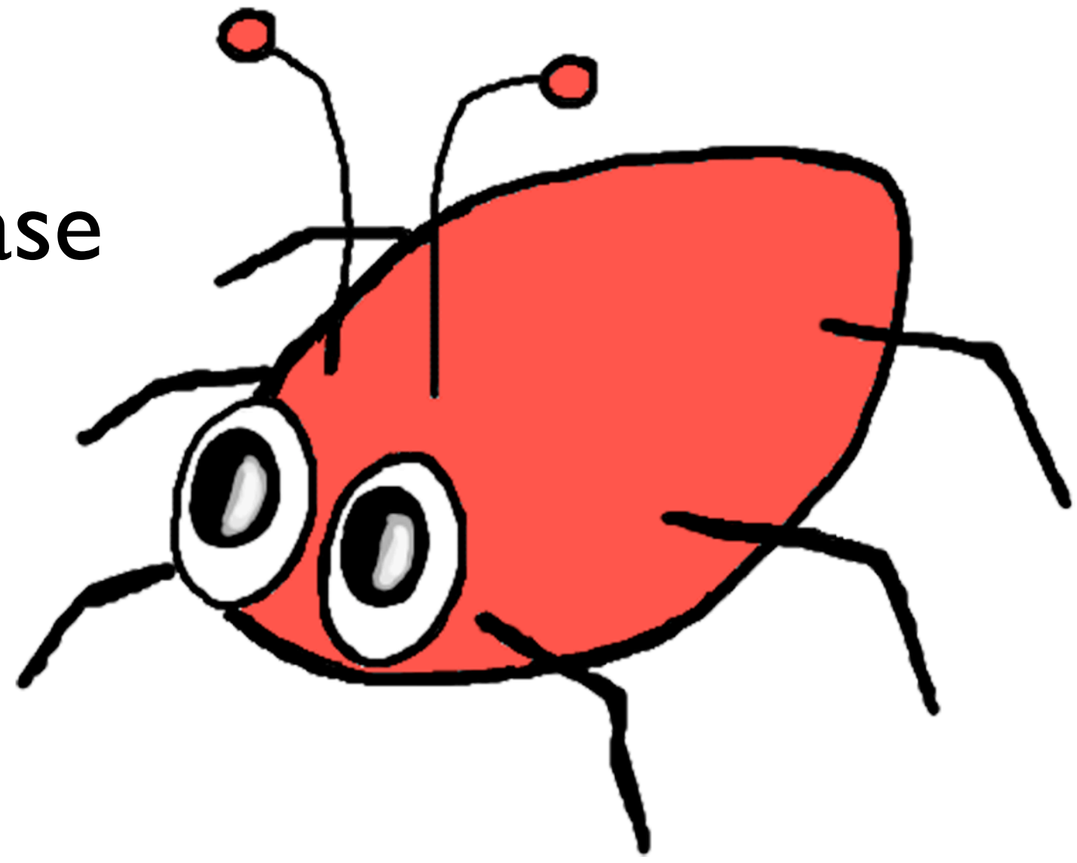
Using FindBugs to find mistakes

- FindBugs is accurate at finding coding mistakes
  - 75+% evaluated as a mistake that should be fixed
- But many mistakes have low costs
  - memory/type safety lowers cost of mistakes
  - If applied to existing production code, many expensive mistakes have already been removed
    - perhaps painfully
- Need to lower cost of using FindBugs to sell to some projects/teams

FindBugs integration at Google

- FindBugs has been in use for years at Google
- In the past week, finally turned on as a presubmit check at Google
- When you want to commit a change, you need a code review
  - now, FindBugs will comment on your code and you need to respond to newly introduced issues and discuss them with the person doing your code review

- First research paper published in 2004
- FindBugs 1.0 released in 2006
- 1,150,000+ downlads from 160+ countries
- Released 1.3.9 in last year
- Working towards 2.0.0 release



FindBugs 2.0

- FindBugs analysis engine continues to improve, but only incrementally
- Focus on efficiently incorporating static analysis into the large scale software development
  - Review of issues done by a community
  - Once issue is marked as “not a bug”, never forget
  - Integration into bug tracking and source code version control systems

Bug ranking

- FindBugs reported a priority for an issue, but it was only meaningful when comparing instances of the same bug pattern
  - a medium priority X bug might be more important than a high priority Y bug
- Now each issue receives a bug rank (a score, 1-20)
  - Can be customized according to your priorities
  - Grouped into Scariest, Scary, Troubling, and Of Concern

FindBugs community review

- Whenever / where ever you run FindBugs, after completing or loading an analysis
  - it talks to the cloud
    - sees how we've been seeing this issue
    - sees if anyone has marked the issue as "should fix" or "not a bug"
- As soon you classify an issue or enter text about the issue, that is sent to the cloud
- Talk

More cloud integration

- Integration with bug tracking systems
  - One click to bring up pre-populated web page in bug tracker describing issue
  - If bug already filed against issue, click shows you existing issue in bug tracker
- Integration with web based source viewers, such as FishEye
  - Allow viewing of file history, change lists, etc.