Defective Java Code: Mistakes That Matter

William Pugh
Univ. of Maryland

FindBugs

Defective Java Code: Mistakes That Matter

William Pugh
Univ. of Maryland

FindBugs

- Use to get excited by being able to automatically find bugs in code
  - Too easy, not rewarding enough
- Now, focused on helping people find and fix mistakes that matter

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

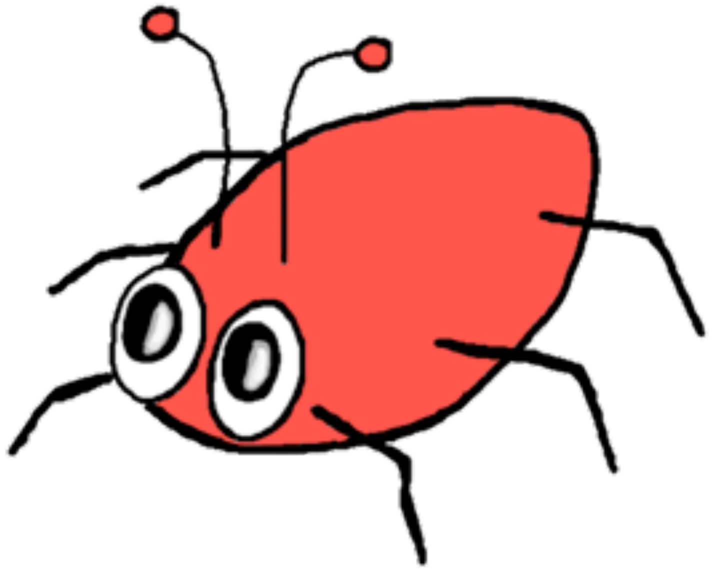
Defective Java Code

Learning from mistakes

- I’m the lead on FindBugs
    - static analysis tool for defect detection
- 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

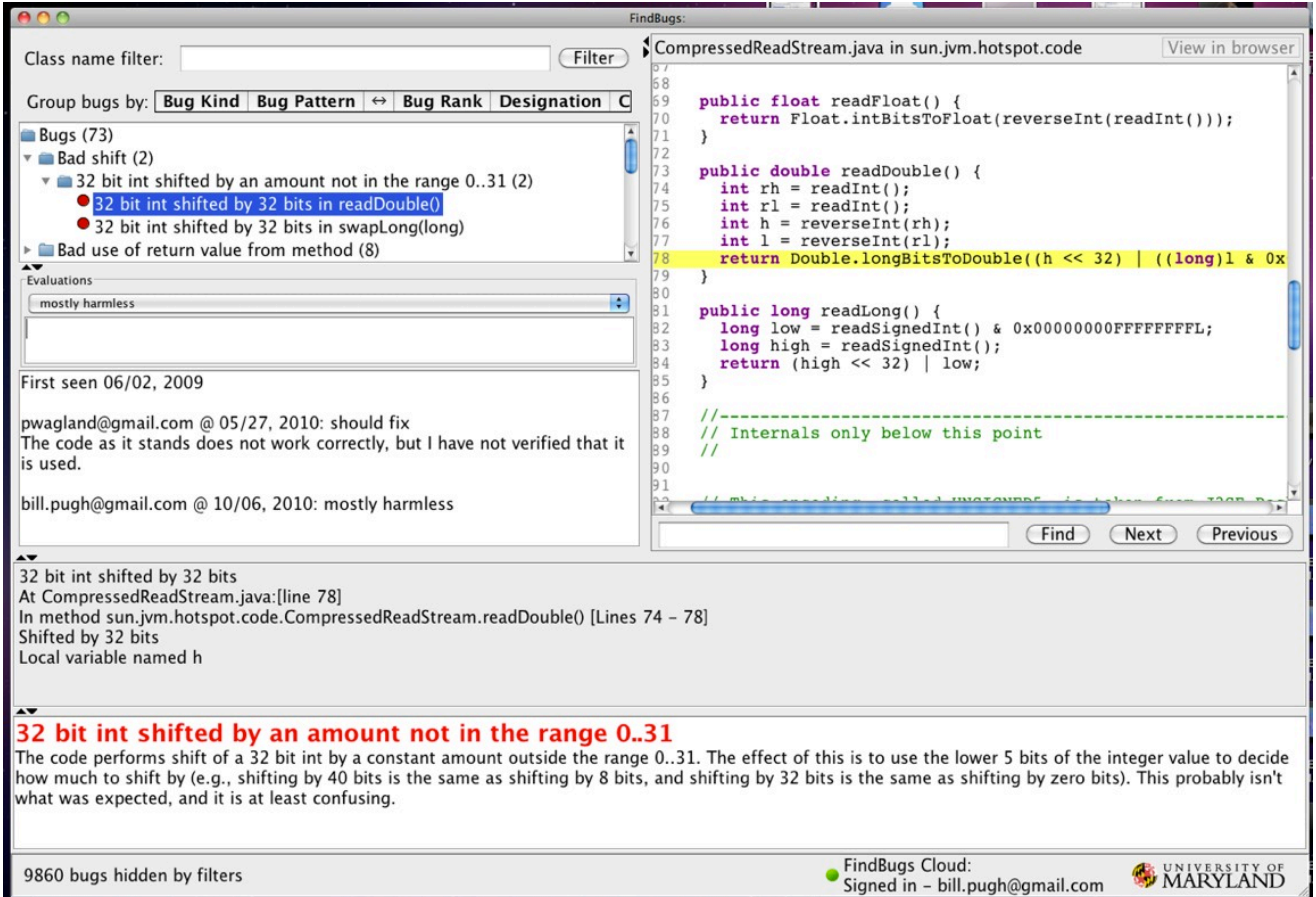
FindBugs fixit @ Google
May 2009

- 4,000 issues to review
  - Bug patterns most relevant to Google
- 8,000 reviews
  - 75+% 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



Bugs (73)
- Bad shift (2)
  - 32 bit int shifted by an amount not in the range 0..31 (2)
    - ● 32 bit int shifted by 32 bits in readDouble()
    - ● 32 bit int shifted by 32 bits in swapLong(long)
- Bad use of return value from method (8)

First seen 06/02, 2009

pwagland@gmail.com @ 05/27, 2010: should fix
The code as it stands does not work correctly, but I have not verified that it is used.

bill.pugh@gmail.com @ 10/06, 2010: mostly harmless

CompressedReadStream.java in sun.jvm.hotspot.code

```
68
69    public float readFloat() {
70      return Float.intBitsToFloat(reverseInt(readInt()));
71    }
72
73    public double readDouble() {
74      int rh = readInt();
75      int rl = readInt();
76      int h = reverseInt(rh);
77      int l = reverseInt(rl);
78      return Double.longBitsToDouble((h << 32) | ((long)l & 0x
79    }
80
81    public long readLong() {
82      long low = readSignedInt() & 0x00000000FFFFFFFFL;
83      long high = readSignedInt();
84      return (high << 32) | low;
85    }
86
87    //------------------------------------------------------------
88    // Internals only below this point
89    //
90
91
```

32 bit int shifted by 32 bits
At CompressedReadStream.java:[line 78]
In method sun.jvm.hotspot.code.CompressedReadStream.readDouble() [Lines 74 – 78]
Shifted by 32 bits
Local variable named h

32 bit int shifted by an amount not in the range 0..31
The code performs shift of a 32 bit int by a constant amount outside the range 0..31. The effect of this is to use the lower 5 bits of the integer value to decide how much to shift by (e.g., shifting by 40 bits is the same as shifting by 8 bits, and shifting by 32 bits is the same as shifting by zero bits). This probably isn't what was expected, and it is at least confusing.

9860 bugs hidden by filters

● FindBugs Cloud:
Signed in – bill.pugh@gmail.com

UNIVERSITY OF MARYLAND

Learned wisdom

- Static analysis typically finds mistakes
  - 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

Audience interaction time

- Which code is better?

```
a)    if (x.equals("name")) { ... }

b)    if ("name".equals(x)) { ... }
```

Discussion

- `"name".equals(x)` handles `x` being null by computing false
- `x.equals("name")` throws NPE if `x` is null
- Do I anticipate that `x` might be null?
- If I don't anticipate that `x` might be null, and it is, what would I prefer?
  - a silent behavior I didn't anticipate
  - a runtime exception

When you write code, it has errors

- Untested code likely isn't correct
  - unit tests / regression tests / system tests
- Your code probably doesn't correctly handle situations you didn't anticipate
- But perfect can only be approached asymptotically
- If you can't prevent an error, can you detect it and log it?
  - if you detect it, is it OK to fail safe?

Runtime exceptions can be your friend

- Pretty common to wrap operations in a try/catch block
  - web transactions, processing a GUI event, etc.
- Most systems will degrade gracefully when they hit runtime exceptions
  - the action that threw the exception fails, but the system keeps going
- If something unanticipated happens, I want to know it

Testing equality to a string constant, *revisited*

- What if I know x might be null? Which do I prefer?

```
a)    x != null && x.equals("foo")

b)    "foo".equals(x)
```

(a) clearly documents that `x` might be null,

(b) might just have been chosen because developer read it in a style guide, although developer doesn't anticipate `x` will ever be null

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

Java Bug Bestiary

Null bug

- From Eclipse, 3.5RC3:
  org.eclipse.update.internal.ui.views.FeatureStateAction

```
if (adapters == null && adapters.length == 0)
    return;
```

- Clearly a mistake
  - First seen in Eclipse 3.2
  - but in practice, adapters is probably never null
- Is there any impact from this?
  - we would probably notice a null pointer exception
  - we don’t immediately return if length is 0

Cost when a mistake causes a fault/failure

- How quickly/reliability would you notice?
- What is the impact of the misbehavior caused by the mistake?
- How easily could you diagnose the problem and the fix?
- What is the cost to deliver a fix?

Null pointer bugs @ Google

- Google's code contains more than a thousand null pointer bugs
- statements or branches that if executed guarantee a null pointer exception
- From looking at exceptions logged in production, can tell you that few if any of the NPE that occur in production are caused by those kinds of mistakes
- typically, caused because message doesn't have an expected component

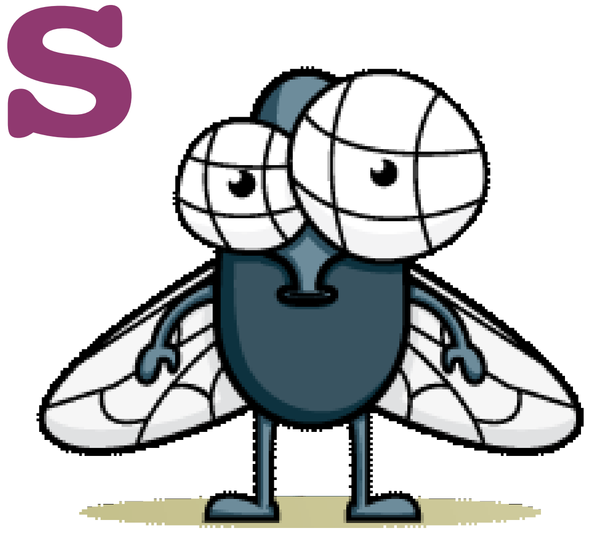
Mistakes in web services

- Some mistakes would manifest themselves by throwing a runtime exception
  - Should be logged and noticed
- If it isn't happening now, a change might cause it to start happening in the future
  - But if it does, the exception will likely pinpoint the mistake
  - And pushing a fix into production is cheaper than pushing a fix to desktop or mobile applications

Expensive mistakes (your results may vary)

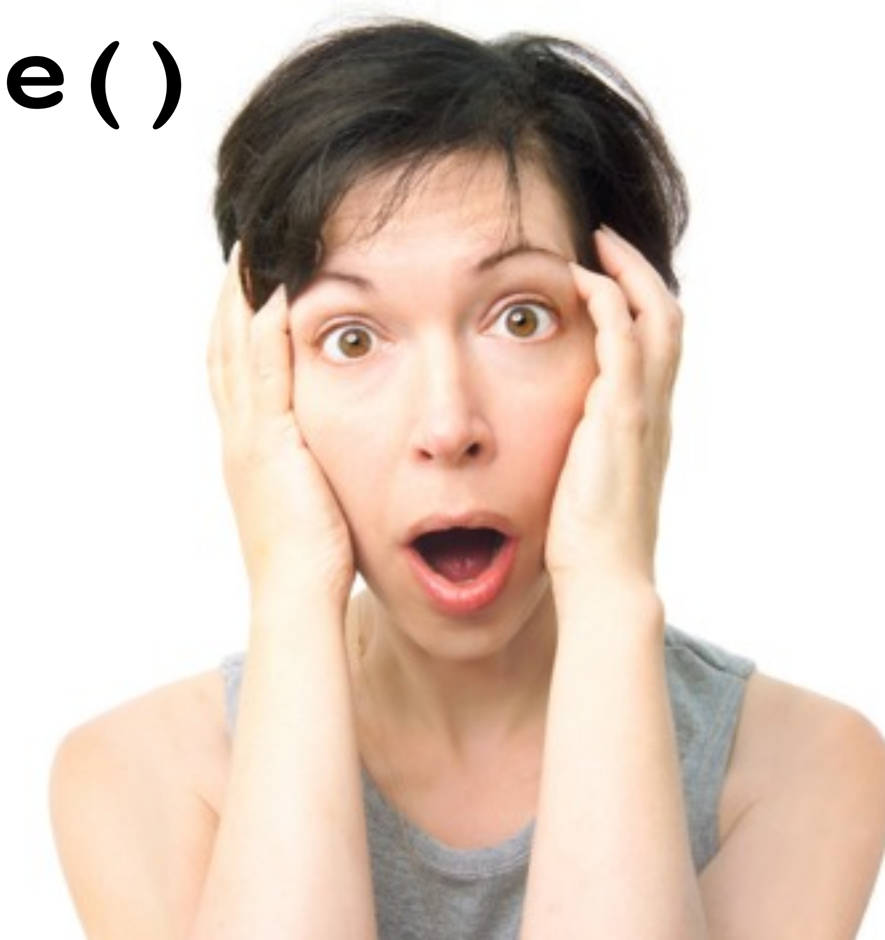
- Mistakes that might cost millions of dollars on the first day they manifest
- Mistakes that silently cause the wrong answer to be computed
  - might be going wrong now, millions of times a day
  - or might be OK now, but when it does go wrong, it won't be noticed until somewhere downstream of mistake
- Mistakes that are expensive or impossible to fix

Using reference equality rather than `.equals`



from Google's code (no one is perfect)

```
class MutableDouble {

  private double value_;

  public boolean equals(final Object o) {
    return o instanceof MutableDouble &&
      ((MutableDouble)o).doubleValue()
        == doubleValue();
  }

  public Double doubleValue() {
    return value_;
  }
```



Using == to compare objects rather than .equals

- For boxed primitives, == and != are computed using pointer equality, but <, <=, >, >= are computed by comparing unboxed primitive values
- Sometimes, equal boxed values are represented using the same object
  - but only sometimes
- This can bite you on other classes (e.g., **`String`**)
  - but boxed primitives is where people get bit

Heisenbugs vs. deterministic bugs

- A Heisenbug is a mistake that only sometimes manifests itself (e.g., a data race)
- Testing not likely to show error
  - if a test fails, rerunning the test may succeed
- Can be very nasty to track down, impossible to debug
- But how dangerous is a bug that only bites once out of 4 billion times?

Ignoring the return value of putIfAbsent

org.jgroups.protocols.pbcast.NAKACK

```
ConcurrentMap<Long,XmitTimeStat>
      xmit_time_stat = ...;

.....
XmitTimeStat stat = xmit_time_stats.get(key);
if(stat == null) {
 stat = new XmitTimeStat();
 xmit_time_stats.putIfAbsent(key, stat);
}
stat.xmit_reqs_received.addAndGet(rcvd);
stat.xmit_rsps_sent.addAndGet(sent);
```

misusing putIfAbsent

- ConcurrentMap provides putIfAbsent
  - atomically add key → value mapping
    - but only if the key isn't already in the map
  - if non-null value is returned, put failed and value returned is the value already associated with the key
- Mistake:
  - ignore return value of putIfAbsent, and
  - reuse value passed as second argument, and
  - matters if two callers get two different values

Fixed in revision 1.179

org.jgroups.protocols.pbcast.NAKACK

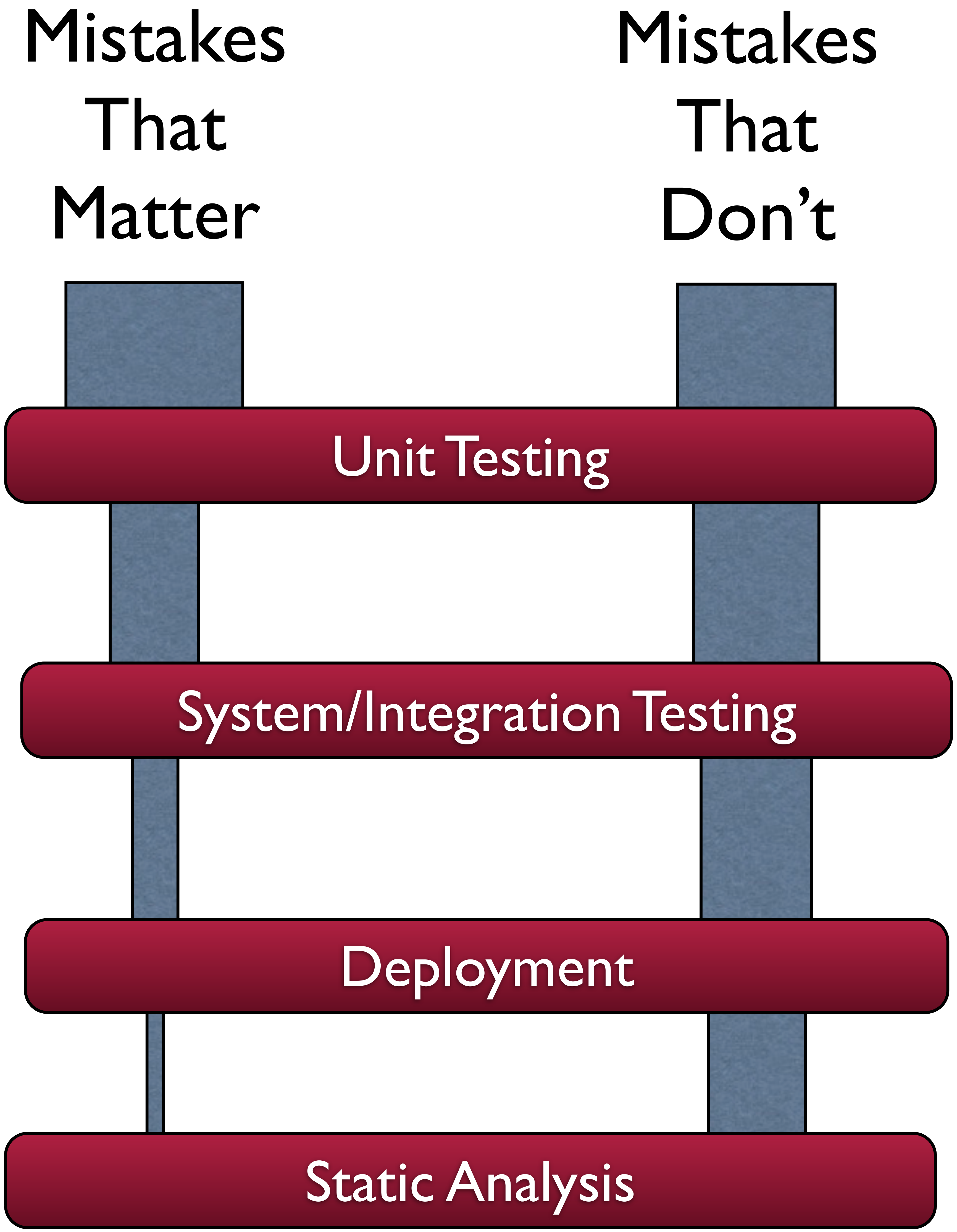
```
XmitTimeStat stat=xmit_time_stats.get(key);

if(stat == null) {

 stat=new XmitTimeStat();

 XmitTimeStat stat2
    = xmit_time_stats.putIfAbsent(key, stat);
   if (stat2 != null)
     stat = stat2;

}

stat.xmit_reqs_received.addAndGet(rcvd);

stat.xmit_rsps_sent.addAndGet(sent)
```

Some lessons

- Concurrency is tricky
- **`putIfAbsent`** is tricky to use correctly
    - engineers at Google got it wrong more than 10% of the time
- Unless you need to *ensure* a single value, just use **`get`** followed by **`put`** if not found
- If you need to ensure a single unique value shared by all threads, use **`putIfAbsent`** and check return value

Survivor effect

- as code comes off of developers fingertips, it contains bugs
- some of these bugs will cause the software to perform incorrectly, some will not
- various measures will remove some of the bugs
- unit test, code review, system test
- These measures tend to be more effective at removing bugs that cause misbehavior than bugs that don't
- Thus, bugs that have been in the system for months or years are genetically fit at surviving



Mistakes That Matter

Mistakes That Don’t

Unit Testing

System/Integration Testing

Deployment

Static Analysis

Static analysis earlier is better

- Find mistakes detected by static analysis before that are detected using more expensive techniques
- Get them to developers while the code is still fresh in developers heads, before anyone else is depending on it or using it
  - Fixing a mistake in code last touched 6 months or 6 years ago isn't fun
- Of course, this only applies if your mistakes are generally caught by other steps in your quality

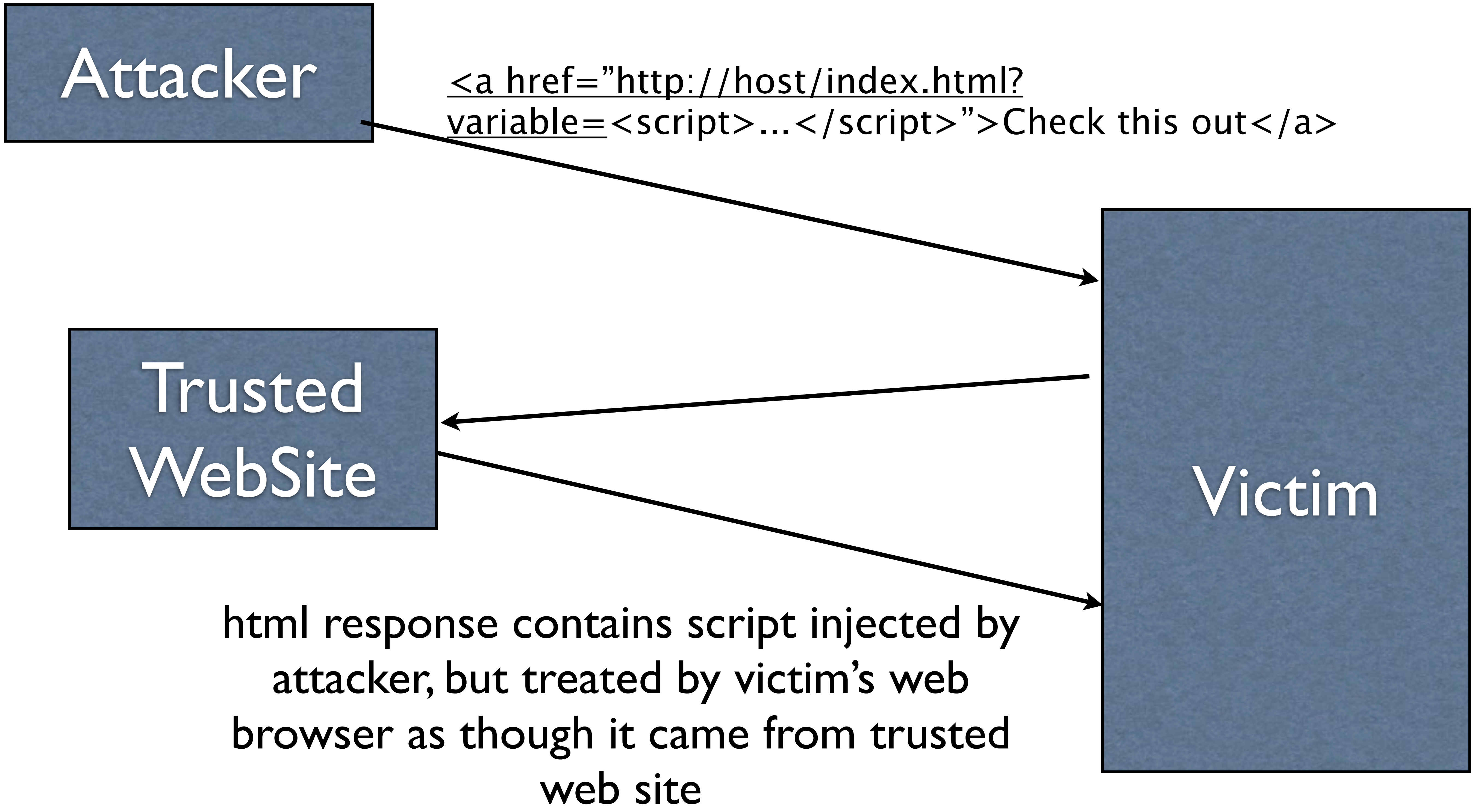
Cross-site scripting

```
public void doGet(HttpServletRequest req,
     HttpServletResponse res) {
  ...

  String target = req.getParameter("url");

  InputStream in = this.getClass()
    .getResourceAsStream("META-INF/resources/"
        + target;

  if (in == null) {
    res.getWriter().println(
      "<p>Unable to locate resource: "
             + target);
    return;
    }
```

Cross-site scripting

- Putting untrusted/unchecked data directly into generated html
  - can contain Javascript, which gets executed in your context
  - untrusted input can be injected into your database, or through a URL query parameter
    - via a link sent from attacker to victim

Cross site scripting



Attacker

<a href=”http://host/index.html?variable=<script>...</script>”>Check this out</a>

Trusted WebSite

Victim

html response contains script injected by attacker, but treated by victim’s web browser as though it came from trusted web site

Security vulnerabilities

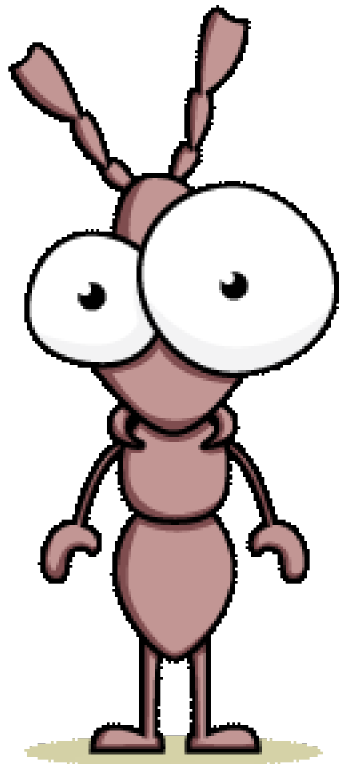
- Not exposed by normal/expected use cases
- Need some combination of:
  - architectural risk analysis
  - careful design
  - static analysis
  - dynamic testing and analysis
- FindBugs only does simple, shallow analysis for network security vulnerabilities

Incomparable equality

org.eclipse.jdt.internal.debug.eval.ast.engine.AstInstructionCompiler

```
SimpleType simpleType = (SimpleType) type;
if ("java.lang.String".equals(simpleType.getName()))
   return Instruction.T_String;
```

- SimpleType.getName() returns a org.eclipse.jdt.core.dom.Name
- In Eclipse since 2.0 (June 2002)
- Finally fixed June 29, 2010
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=318333



Many variations, assisted by weak typing in APIs

- Using .equals to compare incompatible types
- Using .equals to compare arrays
  - only checks if the same array
- Checking to see if a **`Set<Long>`** contains an **`Integer`**
  - never found, even if the same integral value is contained in the map
- Calling **`get(String)`** on a **`Map<Integer,String>`**

Silent, nasty bugs

- Very hard to find these bugs by inspection
  - types not always visible/explicit
- In some cases, could be introduced by refactoring
  - Change the key type of a **`Map`** from **`Integer`** to **`Long`**
  - Fix all the places where you get type errors
  - Leave behind bugs
- Google had an issue with a refactoring that changed a method to return **`byte[]`** rather than **`String`**
  - introduced silent errors

Bug introduced between Eclipse 3.5RC1 and RC2

org.eclipse.pde.internal.build.BrandingIron

```
File rootFolder
 = getCanonicalFile(new File(initialRoot));

if (!rootFolder.equals(target)) {
     rootFolder.delete();
     ...
     }
```

Listen to your bug stories

- In Joshua Bloch's 2009 JavaOne, he said that his #1 takeaway message was don't lock on **`ConcurrentMaps`**
  - My reaction was "Really?"
  - Clearly wrong and a bug, but surely that so obviously wrong it would be exceptionally rare
  - But I wrote a detector for FindBugs

JBoss 5.1.0-GA

- 22 synchonizations on **`ConcurrentHashMap`**
- 9 synchronizations on **`CopyOnWriteArrayList`**
  - In Java 5, **`COWAL`** implementation using **`synchronized(this)`**
  - in Java 6+ **`COWAL`** implementation synchronizes on internal **`Lock`** object
- 3 synchronizations on **`AtomicBoolean`**

Google code

- Just checked overnight
- more than 150 synchronizations on some class in **`java.util.concurrent...`**
    - none on **`CopyOnWriteArrayList`**
- Might not be a problem
    - Sometimes used to allow for wait/notify
    - Sometimes just a handy object to lock
    - Only a problem if expected to block other concurrent actions on object

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?”

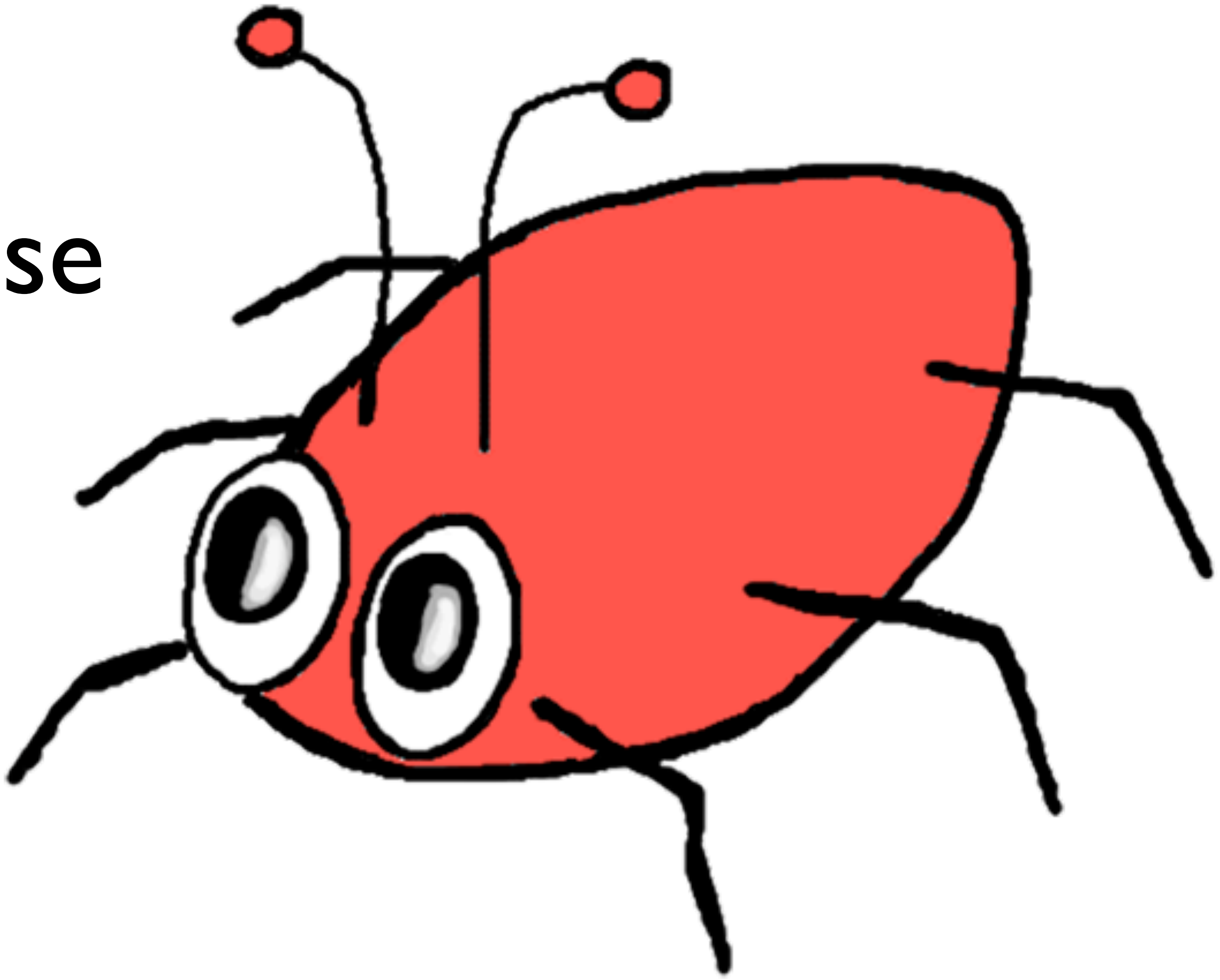
Using FindBugs to find mistakes

- 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.

- Open source system from UMD for managing student programming projects
  - automated web-based testing, with controlled opportunities for testing to help students learn good software skills and TDD
  - Code review system to allow and assign code reviews by instructions and students
  - http://marmoset.cs.umd.edu/
  - http://sourceforge.net/projects/marmoset/