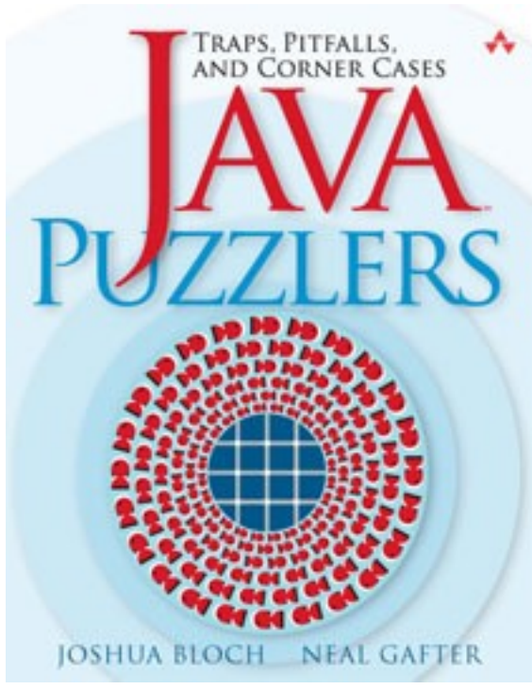
# Painful Java Puzzlers and Bug Patterns

DevIgnition 2012

Bill Pugh

# Painful puzzlers

- Java Programming Puzzlers:
  - Short program with curious behavior
  - What does it print? (multiple choice)
  - The mystery revealed
  - How to fix the problem
  - The moral
- More generally
  - mistakes arising from tricky bits of the Java language and APIs
  - Things to watch out for



# No “New” Java puzzlers

- Java Puzzlers has been a joint effort led by Joshua Bloch, with help from Neal Gafter, myself, Bob Lee and others
- We’ve scraped the bottle of the barrel
- Waiting for Java 8...
- In honor of that, we have 7 bug patterns/areas, in no particular order
- ... starting with 3 classic Java puzzlers

# 1. “Histogram Mystery”

```
public class Histogram {
    private static final String[] words =
      { "I", "recommend", "polygene", "lubricants" };
    public static void main(String[] args) {
        int[] histogram = new int[5];
        for (String word1 : words) {
            for (String word2 : words) {
                String pair = word1 + word2;
                int bucket = Math.abs(pair.hashCode())
                   % histogram.length;
                histogram[bucket]++;
            }
        }
        int pairCount = 0;
        for (int freq : histogram)
            pairCount += freq;
        System.out.println('C' + pairCount);
    }
}
```



4

# What Does It Print?

```
public class Histogram {
    private static final String[] words =
      { "I", "recommend", "polygene", "lubricants" };
    public static void main(String[] args) {
        int[] histogram = new int[5];
        for (String word1 : words) {
            for (String word2 : words) {
                String pair = word1 + word2;
                int bucket = Math.abs(pair.hashCode())
                   % histogram.length;
                histogram[bucket]++;
            }
        }
        int pairCount = 0;
        for (int freq : histogram)
            pairCount += freq;
        System.out.println('C' + pairCount);
    }
}
```

(a) `83`

(b) `C16`

(c) `S`

(d) None of the above



5

# What Does It Print?

(a) `83`

(b) `C16`

(c) `S`

(d) None of the above – throws `ArrayOutOfBoundsException`

**`Math.abs(int)`** can return a negative number,

and so can the `%` operator



6

# Another Look

```
public class Histogram {
    private static final String[] words = // Carefully chosen!
      { "I", "recommend", "polygene", "lubricants" };
    // "polygenelubricants".hashCode() == Integer.MIN_VALUE
    public static void main(String[] args) {
        int[] histogram = new int[5];
        for (String word1 : words) {
            for (String word2 : words) {
                String pair = word1 + word2;
                int bucket = Math.abs(pair.hashCode())
                   % histogram.length;
                histogram[bucket]++;
            }
        }
        int pairCount = 0;
        for (int freq : histogram)
            pairCount += freq;
        System.out.println('C' + pairCount);
    }
}
```



7

# How Do You Fix It?

```
public class Histogram {
    private static final String[] words =
      { "I", "recommend", "polygene", "lubricants" };
    public static void main(String[] args) {
        int[] histogram = new int[5];
        for (String word1 : words)
            for (String word2 : words) {
                String pair = word1 + word2;
                int bucket = Math.abs(pair.hashCode()
                   % histogram.length); // Math.abs follows %
                histogram[bucket]++;
            }
        int pairCount = 0;
        for (int freq : histogram)
            pairCount += freq;
        System.out.println('C' + pairCount);
    }
}
```



8

# Moral

- **`Math.abs`** doesn't guarantee nonnegative result
  - **`Integer.MIN_VALUE == -Integer.MIN_VALUE`**
- The `%` operator is remainder, not mod; can be negative
- To translate a signed hash value to a bucket
  - **`Math.abs(hashVal % buckets.length)`**
    - Or **`(hashVal >>> 1) % buckets.length`**
    - Or **`(hashVal & 0x7fffffff) % buckets.length`**
    - Or use power-of-two length array
      **`(hashVal & (buckets.length - 1))`**



9

# Related problems

- bytes are signed
  - and sign extended
- shifting an int by 32 bits, and then converting it to a long
- ints silently converted to float
- methods that return -1 (EOF) or 0-255
  - or -1 or an unsigned char

# 2. “The Joy of Sets”

```
public class ShortSet {
    public static void main(String args[]) {
        Set<Short> s = new HashSet<Short>();
        for (short i = 0; i < 100; i++) {
            s.add(i);
            s.remove(i - 1);
        }
        System.out.println(s.size());
    }
}
```

11

# What Does It Print?

```
public class ShortSet {
    public static void main(String args[]) {
        Set<Short> s = new HashSet<Short>();
        for (short i = 0; i < 100; i++) {
            s.add(i);
            s.remove(i - 1);
        }
        System.out.println(s.size());
    }
}
```

(a) 1

(b) 100

(c) Throws exception

(d) None of the above



12

# What Does It Print?

(a) `1`

(b) `100`

(c) Throws exception

(d) None of the above

**The set contains `Short` values, but we're removing `Integer` values**



13

# Another Look

```
public class ShortSet {

    public static void main(String args[]) {

        Set<Short> s = new HashSet<Short>();

        for (short i = 0; i < 100; i++) {

            s.add(i);

            s.remove(i - 1); // int-valued expression

        }

        System.out.println(s.size());

    }

}
```



14

# Another ‘nother Look

```
public class ShortSet {
    public static void main(String args[]) {
        Set<Short> s = new HashSet<Short>();
        for (short i = 0; i < 100; i++) {
            s.add(i);
            s.remove(i - 1); // int-valued expression
        }
        System.out.println(s.size());
    }
}
public interface Set<E>extends Collection<E> {
    public abstract boolean add(E e);
    public abstract boolean remove(Object o);
    ...
}
```



15

# How Do You Fix It?

```
public class ShortSet {

    public static void main(String args[]) {

        Set<Short> s = new HashSet<Short>();

        for(short i = 0; i < 100; i++) {

            s.add(i);

            s.remove((short) (i - 1));

        }

        System.out.println(s.size());

    }

}
```



16

# Moral

- **`Collection<E>.remove`** takes **`Object`**, not **`E`**
  - Also **`Collection.contains`**, **`Map.get`**
- Integral arithmetic always results in **`int`** or **`long`**
- Avoid mixing types
- Avoid **`short`**; prefer **`int`** and **`long`**
  - Arrays of **`short`** are the only compelling use case



17

# Mismatched types

- Lots of places where you can pass in an object of the wrong type, and nothing happens
- comparing incompatible objects with equals

# Map interface

```
public interface Map<K,V> {
  V put(K key, V value);
  V get(Object key);
  boolean containsKey(Object key);
  boolean containsValue(Object value);
  V remove(Object key);
  ...
}
```

# Map interface is mostly untyped

- It is type *safe* to pass any object to these methods
  - type parameter ignored
  - If it is an incompatible type, the call will do nothing
- I'm told it had to be this way for backwards compatibility
  - I'm getting to hate backwards compatibility

# Comparing objects of different types

- Code that compares an instance of Foo with a String for equality
  - almost always wrong
  - might be OK if Foo.equals checks for a String being passed as an argument
  - Foo shouldn't do this: break symmetry, and confusing as hell

# 3. “Mind the Gap”

```
import java.io.*;
public class Gap {
    private static final int GAP_SIZE = 10 * 1024;
    public static void main(String args[]) throws IOException {
        File tmp = File.createTempFile("gap", ".txt");
        FileOutputStream out = new FileOutputStream(tmp);
        out.write(1);
        out.write(new byte[GAP_SIZE]);
        out.write(2);
        out.close();
        InputStream in =
            new BufferedInputStream(new FileInputStream(tmp));
        int first = in.read();
        in.skip(GAP_SIZE);
        int last = in.read();
        System.out.println(first + last);
    }
}
```



22

# What does it print?

(a) 1

(b) 3

(c) Throws exception

(d) It varies

```
import java.io.*;
public class Gap {
    private static final int GAP_SIZE = 10 * 1024;
    public static void main(String args[]) throws IOException {
        File tmp = File.createTempFile("gap", ".txt");
        FileOutputStream out = new FileOutputStream(tmp);
        out.write(1);
        out.write(new byte[GAP_SIZE]);
        out.write(2);
        out.close();
        InputStream in =
            new BufferedInputStream(new FileInputStream(tmp));
        int first = in.read();
        in.skip(GAP_SIZE);
        int last = in.read();
        System.out.println(first + last);
    }
}
```



23

# What Does It Print?

(a) 1 (in practice)

(b) 3

(c) Throws exception

(d) It varies from run to run (according to spec)

**`skip` returns a value; ignore it at your peril.**
**Also it is difficult to use correctly.**



24

# Another look

```
import java.io.*;
public class Gap {
    private static final int GAP_SIZE = 10 * 1024;
    public static void main(String args[]) throws IOException {
        File tmp = File.createTempFile("gap", ".txt");
        FileOutputStream out = new FileOutputStream(tmp);
        out.write(1);
        out.write(new byte[GAP_SIZE]);
        out.write(2);
        out.close();
        InputStream in =
            new BufferedInputStream(new FileInputStream(tmp));
        int first = in.read();
        in.skip(GAP_SIZE); // return value ignored
        int last = in.read();
        System.out.println(first + last);
    }
}
```



25

# How Do You Fix It?

```
static void skipFully(InputStream in, long nBytes)
        throws IOException {
    long remaining = nBytes;
    while (remaining != 0) {
        long skipped = in.skip(remaining);
        if (skipped == 0)
            throw new EOFException();
        remaining -= skipped;
    }
}
```



26

# Moral

- The **`skip`** method is hard to use and error prone
- Use your **`skipFully`** instead of skip
  - There is an RFE to add it to **`InputStream`**
- More generally, if an API is broken, wrap it
- For API designers
  - Don’t violate the principle of least astonishment
  - Make it easy to do simple things



27

# Developers don't read the documentation

- If a developers adds a call to a method without reading the JavaDoc, are they likely to invoke it correctly?
  - does it a call look like it does one thing, but actually does another?
  - Does it return a value that only matters in exceptional circumstance?
  - Is it hard to call the method correctly?
    - InputStream.skip(...)
    - ConcurrentMap.putIfAbsent(x, y)

# More examples of bad method calls

// com.sun.rowset.CachedRowSetImpl

```
if (type == Types.DECIMAL || type == Types.NUMERIC)
        ((java.math.BigDecimal)x).setScale(scale);
```

// com.sun.xml.internal.txw2.output.XMLWriter

```
try { ... }
catch (IOException e) {
  new SAXException("Server side Exception:" + e);
  }
```



29

# Bad Method Invocations

- Methods whose return value should never be ignored
  - Strings are immutable, so functions like **`trim()`** and **`toLowerCase()`** return new String
- Methods that rarely return an exceptional value
  - `File.mkdir()`
- Dumb/useless methods
  - Invoking **`toString`** or equals on an array
- Lots of specific rules about particular API methods
  - Hard to memorize, easy to get wrong

# 4. Supported after all?

```
com.sun.corba.se.impl.io.IIOPInputStream:

 protected final Class resolveClass(ObjectStreamClass v)
   throws IOException, ClassNotFoundException {
       throw new IOException(
            "Method resolveClass not supported");
   }
```

- Class extends **`java.io.ObjectInputStream`**
- Surprisingly, calling **`resolveClass`** works same as in OIS, doesn't throw exception



31

# Does it override the superclass method?

```
java.io.ObjectInputStream:

 protected Class<?>
    resolveClass(ObjectStreamClass desc)
    throws IOException, ClassNotFoundException { … }

com.sun.corba.se.impl.io.IIOPInputStream:

 protected final Class resolveClass(ObjectStreamClass v)
   throws IOException, ClassNotFoundException { … }
```



32

# Look at those elided imports

```
com.sun.corba.se.impl.io.IIOPInputStream:

import com.sun.corba.se.impl.io.ObjectStreamClass;

protected final Class resolveClass(ObjectStreamClass v)
    throws IOException, ClassNotFoundException { … }
```

- Parameter types are different: same simple name, different packages
- Doesn't override method in superclass



33

# identity confusion problems

- Easy to mistakenly refer to or name the wrong thing
- define a method that should but doesn't override a method in a superclass
- self assignment of field (see JBoss)
- invoke the wrong version of an overloaded method

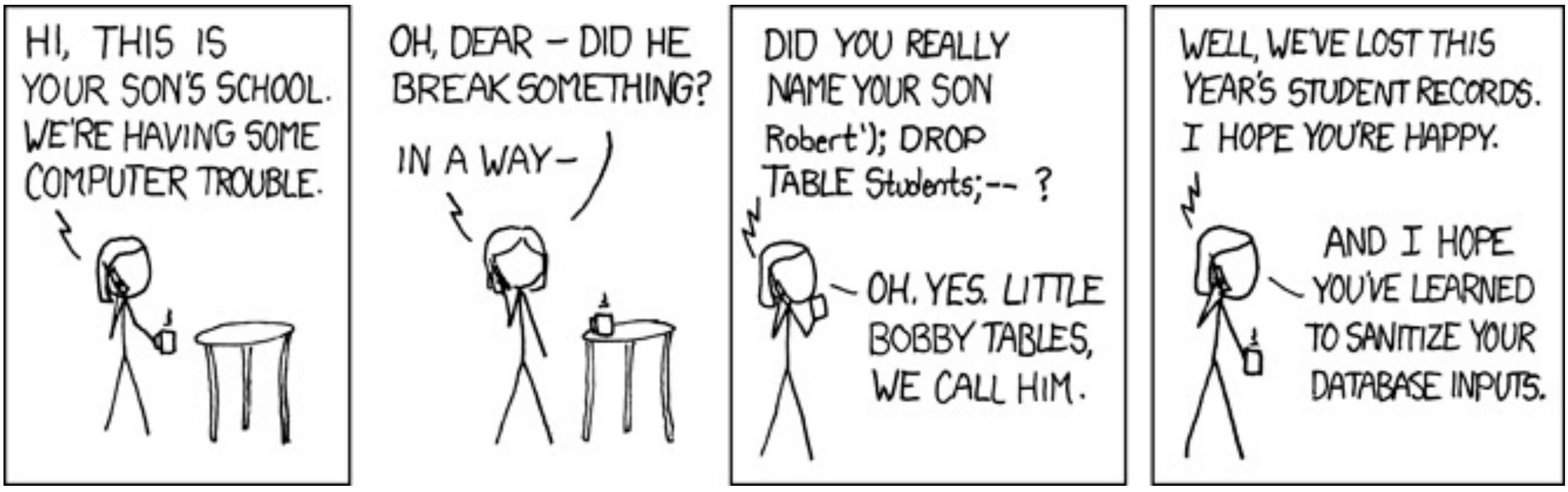
# Interlude

- Why are puzzlers particularly painful/dangerous?
  - Because they look *correct*
- They slide right through code review
- When trying to debug them, you keep circling them, checking everything else
- Particularly nasty if they fail silently

# 5. Security bugs

- SQL Injection
- XSS - Cross site scripting
- HTTP response splitting
- CSRF - Cross site request forgery
- Path traversal

# SQL Injection



# SQL injection

- building SQL statements with string concatenation of untrusted/unchecked user input
- In Java, use PreparedStatements
  - only case where SQL strings shouldn't be constant is when the table/column names need to be parameterized
  - be scared of such code
- In 2011, SQL injection was responsible for the compromises of many high-profile organizations, including Sony Pictures, PBS, MySQL.com, security company HBGary Federal, and many others.

# XSS - cross site scripting

- Reflected XSS - When a web server echos untrusted user data as part of a response to a request
- Stored XSS - when a web server stored untrusted data into a store, and then includes that data as part of responses to requests

# XSS - why is this a problem

- untrusted data could include Javascript
- which is executed in the context of the owner's web page, having access to cookies, session, etc
  - can take actions on your behalf, or hijack your session

# HTTP response splitting

- Putting untrusted user input into an HTTP header
- Can include new lines, take over entire response

# CSRF - cross site request forgery

- Post a web page that says “check out my book on Amazon”
- tweet/promote the web page

- If anyone visiting your web page, it generates a request to the Amazon server to “Buy now”
- If you are already logged into Amazon, it will send the cookie that authenticates you to Amazon
-

# Preventing XSRF

- All web pages that perform an action should be POST requests, not GET requests
  - restrict POST requests to only those that have an appropriate referer
  - possible to spoof referer, requires broken plugins, etc
- Can also include a secure hidden hash value in the form
  - only present on and send from authorized pages

# Path traversal

- Using String concatenation or

  ```
  new File(f, request.getParameter(name))
  ```

  to form file names, using untrusted user input

- untrusted user input can include ../../..

# 2011 CWE/SANS Top 25 Most Dangerous Software Errors

- http://cwe.mitre.org/top25/index.html
- SQL injection
- OS Command Injection
- Buffer Copy without Checking Size of Input
- Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- Missing Authentication for Critical Function
- ...

# 6. Concurrency bugs

- 10 years ago, a lot of us saw multicore coming, and we know that most coders didn't know how to write correct concurrent code
  - mostly, they still don't
  - but it hasn't been the disaster many suspected
    - but I've heard reports that it is a disaster on 168 core machines
  - Maybe mistakes that only bite less than one in a million times aren't the biggest problem software has

# concurrency bugs

- Dataraces
  - multiple threads simultaneously accessing something that isn't thread-safe
- Atomicity failure
  - A thread performs a sequence of operations on a thread-safe object
- Deadlock -- not much of a problem in practice
  - caused due to inconsistent lock ordering

# Datarace bug

- Did some performance tests of JBoss
- Ran running 30 clients against a JBoss server with 24 cores
- After load test was complete, load average on server stayed at 16

# bug, continued

- Multiple threads were putting some debugging information into a shared unsychronized HashMap
  - just debugging information, not criticial
  - except that if two threads try to resize a HashMap at the same time, they can introduce a cycle into the linked list of entries in a bucket
  - once that happens, threads that go into that bucket never come out

# Atomicity failure

```
ConcurrentHashMap map;
void foo(Object key, Object value) {
        if (map.get(key) == null)
            anyMap.put(key, value);
}
```

# Using putIfAbsent

- ConcurrentHashMap supports putIfAbsent, to do this atomically
  - but can be tricky to use correctly
- putIfAbsent returns null if it succeeded, returns the current value if it fails

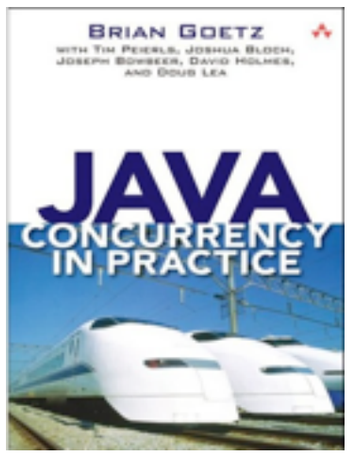
# Using putIfAbsent correctly

```
V cachedComputation(K key) {
   V value = map.get(key);
   if (value != null) return value;
   value = computeValue(key);
   Value v2
      = map.putIfAbsent(key, value);
   if (v2 != null)
       value = v2;

   return value;
   }
```

# Preventing concurrency bugs

- When possible, use higher level concurrency abstractions
  - no spaghetti concurrency control
  - try to avoid mixing concurrency logic and business logic
- Document your concurrent designs
- Read Java Concurrency in Practice
- Understand and use java.util.concurrent
  - Fork/Join framework in Java 7 is really nice
  - and parallel array constructs coming in Java 8 will be even nicer



# 7. Untested code

- If a method isn't ever executed, high chance that it doesn't work
  - A system test might be OK if not ideal
- If you don't have any situation that causes the method to be executed, why did you write it?

# 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?"

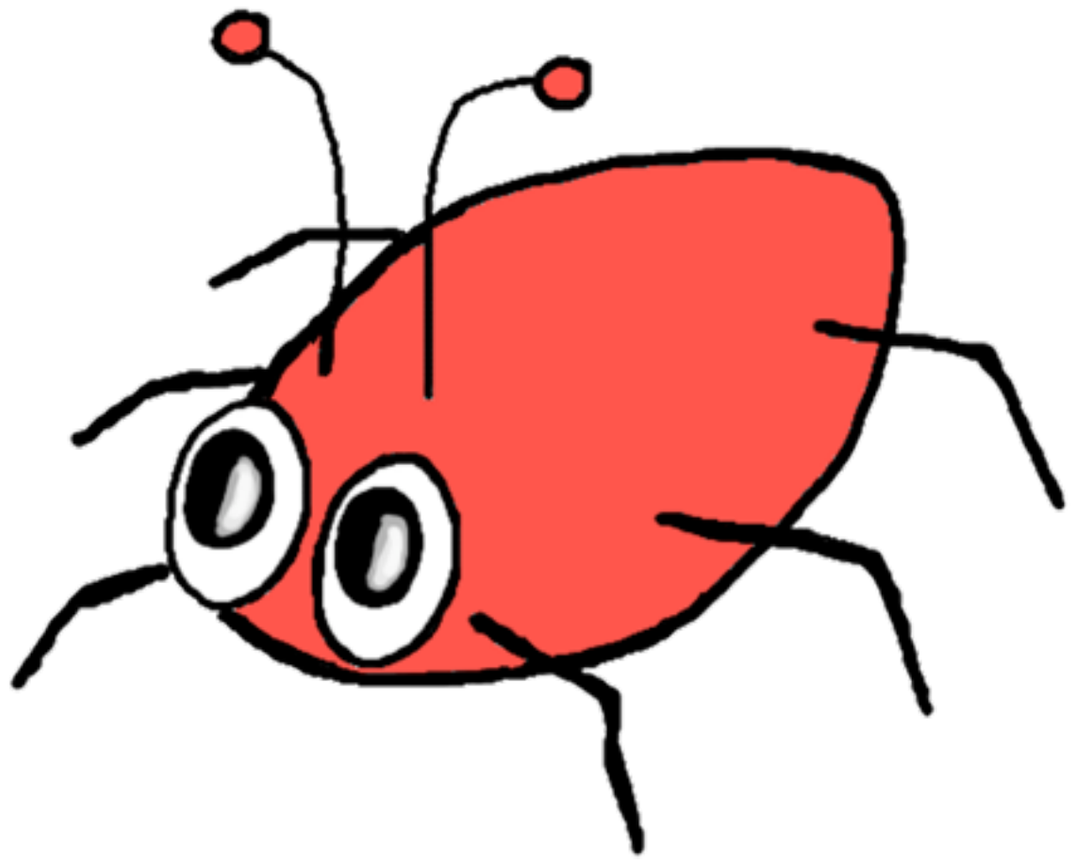
# Using FindBugs to find mistakes

- 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
- 1.7+ million downloads from 160+ countries
- 2.0.1 released
  - 2.0.2 out within a week



# 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

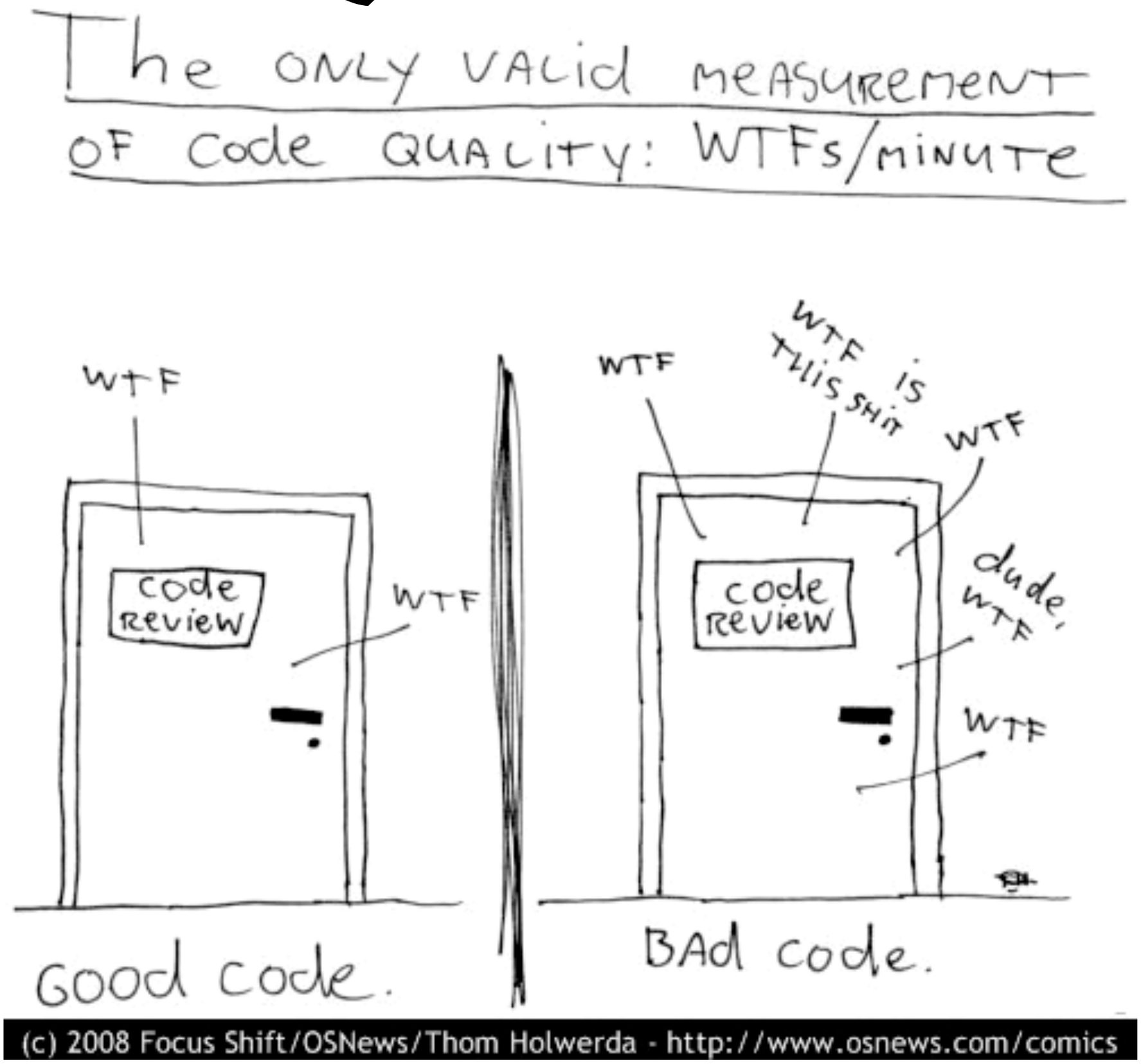
# FindBugs community review

- 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.

# Questions?


The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF is THIS SHIT
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics