

SD Best Practices,
2006

Squashing Bugs with Static Analysis



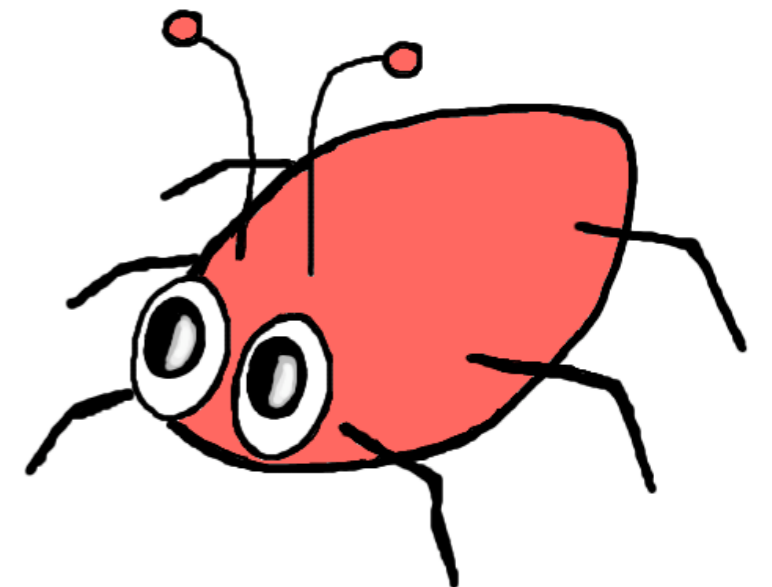
William Pugh

Univ. of Maryland

<http://www.cs.umd.edu/~pugh/>

<http://findbugs.sourceforge.net/>

FindBugs™



FindBugs

- Open source static analysis tool for finding defects in Java programs
- Analyzes classfiles
- Generates XML or text output
 - can run in Netbeans/Swing/Eclipse/Ant/SCA
- Total downloads from SourceForge: 274,291+

What is FindBugs?

- Static analysis tool to find defects in Java code
 - not a style checker
- Can find hundreds of defects in each of large apps such as Bea WebLogic, IBM Websphere, Sun's JDK
 - real defects, stuff that should be fixed
 - hundreds is conservative, probably *thousands*
- Doesn't focus on security
 - lower tolerance for false positives

Common Wisdom about Bugs

- Programmers are smart
- Smart people don't make dumb mistakes
- We have good techniques (e.g., unit testing, pair programming, code inspections) for finding bugs early
- So, bugs remaining in production code must be subtle, and require sophisticated techniques to find

Would You Write Code Like This?

```
if (in == null)
    try {
        in.close();
        ...
    }
```

- Oops
- This code is from Eclipse (versions 3.0 - 3.2)
- You may be surprised what is lurking in your code

Why Do Bugs Occur?

- Nobody is perfect
- Common types of errors:
 - Misunderstood language features, API methods
 - Typos (using wrong boolean operator, forgetting parentheses or brackets, etc.)
 - Misunderstood class or method invariants
- Everyone makes syntax errors, but the compiler catches them
 - What about bugs one step removed from a syntax error?

Bug Patterns

Infinite recursive loop

- Student came to office hours, was having trouble with his constructor:

```
/** Construct a WebSpider */  
  
public WebSpider() {  
    WebSpider w = new WebSpider();  
}
```

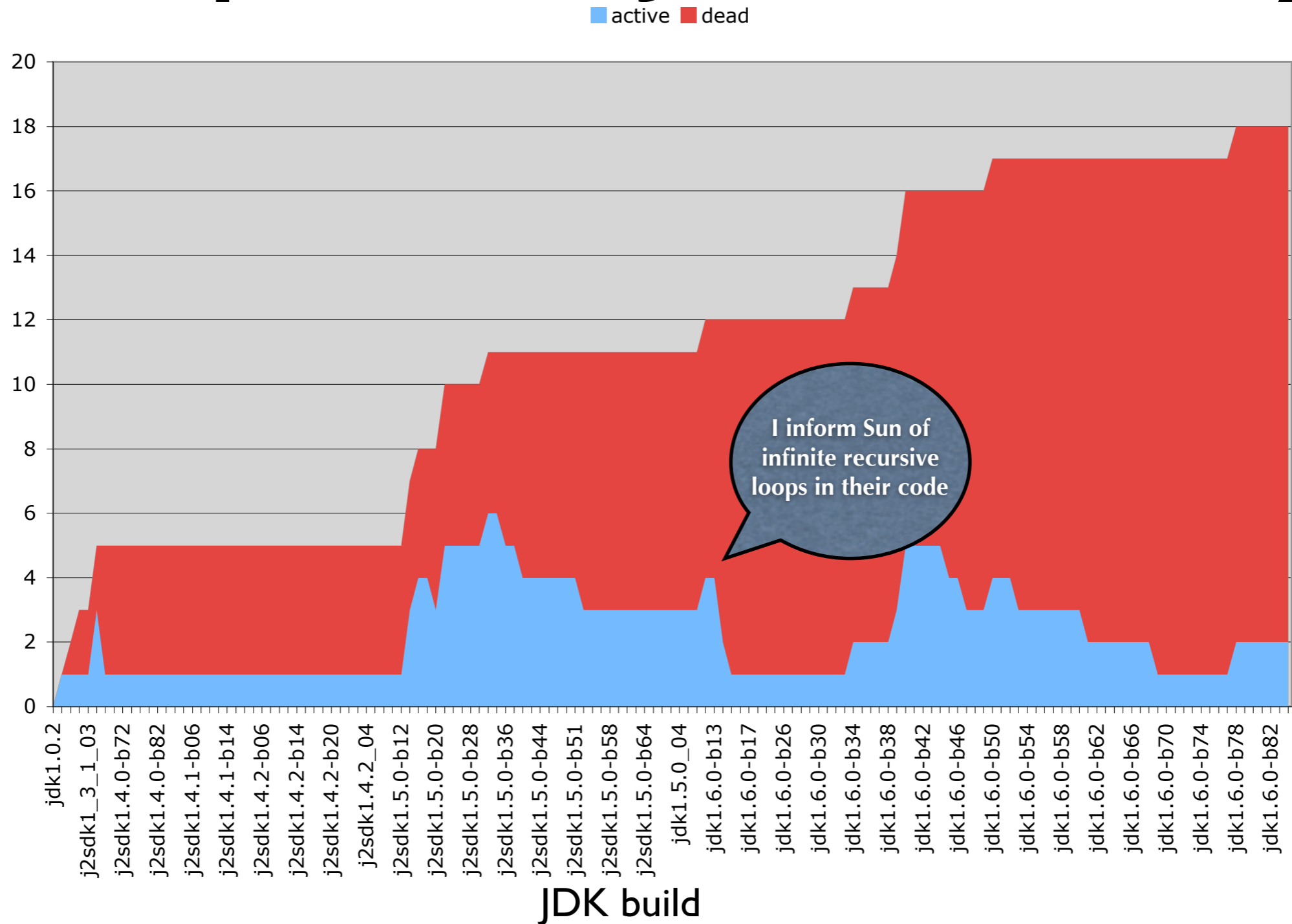
- A second student had the same bug
- Wrote a detector, found 3 other students with same bug

Double check against JDK

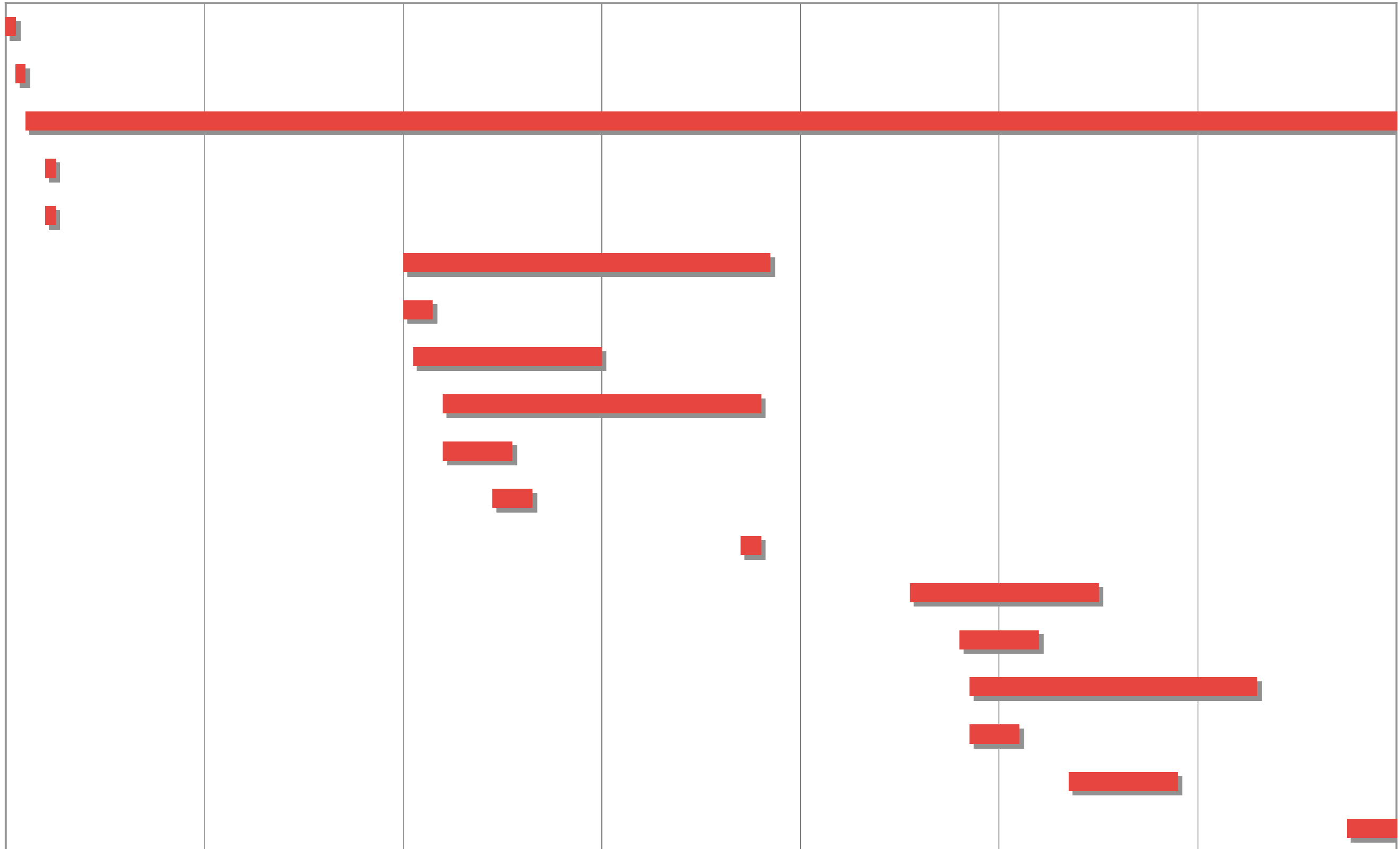
- Found 4 infinite recursive loops
- Including one written by Joshua Bloch

```
public String foundType() {  
    return this.foundType();  
}
```
- Smart people make dumb mistakes
- Embrace and fix your dumb mistakes

Infinite Recursive Loops: Sun JDK history



Duration of infinite recursive loop bugs in JDK



HashCode/Equals

- Equal objects must have equal hash codes
- Programmers sometimes override equals() but not hashCode()
 - Or, override hashCode() but not equals()
- Objects violating the contract won't work in hash tables, maps, sets
- Examples (53 bugs in 1.6.0-b29)
 - javax.management.Attribute
 - java.awt.geom.Area

Fixing hashCode

- What if you want to define equals, but don't think your objects will ever get put into a HashTable?
- Suggestion:

```
public int hashCode() {  
    assert false : "hashCode method not designed";  
    return 42;  
}
```

Null Pointer Dereference

- Dereferencing a null value results in `NullPointerException`
- Warn if there is a statement or branch that if executed, guarantees a NPE
- Example:

```
// Eclipse 3.0.0M8  
Control c = getControl();  
if (c == null && c.isDisposed())  
    return;
```

More Null Pointer Dereferences

```
// Eclipse 3.0.0M8
```

```
String sig = type.getSignature();
```

```
if (sig != null || sig.length() == 1) {  
    return sig;  
}
```

```
// JDK 1.5 build 42
```

```
if (name != null || name.length > 0) {
```

More Null Pointer Dereferences

```
javax.security.auth.kerberos.KerberosTicket, 1.5b42
// flags is a parameter
// this.flags is a field
if (flags != null) {
    if (flags.length >= NUM_FLAGS)
        this.flags = ...
    else
        this.flags = ...
} else
    this.flags = ...
if (flags[RENEWABLE_TICKET_FLAG]) {
```


Redundant Null Comparison

- Comparing a reference to null when it is definitely null or definitely non-null
- Not harmful per se, but often indicates an inconsistency that might be a bug
- Example (JBoss 4.0.0DR3):

```
protected Node findNode(Fqn fqn, ...) {  
    int treeNodeSize = fqn.size();  
    ...  
    if (fqn == null) return null;  
}
```

How should we fix this bug?

```
if (name != null || name.length > 0)
```

- Should we just change it to

```
if (name != null && name.length > 0)
```

- Will that fix it?
 - We have no idea. Obviously, we've never tested the situation when `name` is null.
- Try to write a test case first, then apply the obvious fix

Bad Binary operations

```
if ((f.getStyle () & Font.BOLD) == 1) {  
    sbuf.append ("<b>");  
    isBold = true;  
}
```

```
if ((f.getStyle () & Font.ITALIC) == 1) {  
    sbuf.append ("<i>");  
    isItalic = true;  
}
```

Doomed Equals

```
public static final ASDDVersion  
    getASDDVersion(BigDecimal version) {  
  
    if(SUN_APPSERVER_7_0.toString()  
        .equals(version))  
        return SUN_APPSERVER_7_0;
```

Unintended regular expression

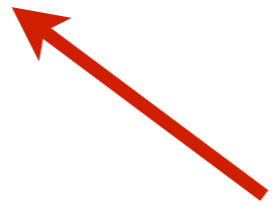
```
String[] valueSegments  
    = value.split("."); // NOI18N
```

Field Self Assignment

```
public TagHelpItem(String name, String file,  
                   String startText, int startOffset,  
                   String endText, int endOffset,  
                   String textBefore, String textAfter){  
    this.name = name;  
    this.file = file;  
    this.startText = startText;  
    this.startTextOffset = startTextOffset;  
    this.endText = endText;  
    this.endTextOffset = endTextOffset;  
    this.textBefore = textBefore;  
    this.textAfter = textAfter;  
    this.identical = null;  
}
```

Bad Naming

```
package org.eclipse.jface.dialogs;
public abstract class Dialog extends Window {
    protected Button getOKButton() {
        return getButton(IDialogConstants.OK_ID);
    };
}
public class InputDialog extends Dialog {
    protected Button getOkButton() {
        return okButton;
    };
}
```



Wrong capitalization

Confusing/bad naming

- **Methods with identical names and signatures**
 - but different capitalization of names
 - could mean you don't override method in superclass
 - confusing in general
- **Method name same as class name**
 - gets confused with constructor

Bad naming in BCEL (shipped in jdk1.6.0-b29)

```
/** @return a hash code value  
 *for the object.  
 */  
public int hashCode() {  
    return basic_type.hashCode()  
        ^ dimensions; }  
}
```

Ignored return values

- Lots of methods for which return value always should be checked
 - E.g., operations on immutable objects

```
// Eclipse 3.0.0M8
```

```
String name= workingCopy.getName();
```

```
name.replace('/', '.');
```

Ignored Exception Creation

```
/**
 * javax.management.ObjectInstance
 * reference impl., version 1.2.1
 **/
public ObjectInstance(ObjectName objectName,
                      String className) {
    if (objectName.isPattern()) {
        new RuntimeException(
            new IllegalArgumentException(
                "Invalid name->" + objectName.toString()));
    }
    this.name = objectName;
    this.className = className;
}
```

Inconsistent Synchronization

- Common idiom for thread safe classes is to synchronize on the receiver object (“this”)
- We look for field accesses
 - Find classes where lock on “this” is sometimes, but not always, held
 - Unsynchronized accesses, if reachable from multiple threads, constitute a race condition

Inconsistent Synchronization Example

- GNU Classpath 0.08, java.util.Vector

```
public int lastIndexOf(Object elem)
{
    return lastIndexOf(elem, elementCount - 1);
}
```

```
public synchronized int lastIndexOf(
    Object e, int index)
{
    ...
}
```

Unconditional Wait

- Before waiting on a monitor, the condition should be almost always be checked
 - Waiting unconditionally almost always a bug
 - If condition checked without lock held, could miss the notification

- Example (JBoss 4.0.0DR3):

```
if (!enabled) {  
    try {  
        log.debug(...);  
        synchronized (lock) {  
            lock.wait();  
        }  
    }  
}
```

condition can become true after it is checked

but before the wait occurs

Bug Categories

- Correctness
- Bad Practice
 - equals without hashCode, bad serialization, comparing Strings with ==, equals should handle null argument
- Dodgy
 - Dead store to local variable, load of known null value, overbroad catch
- Performance
- Multithreaded correctness
- Malicious code vulnerability

Demo

- Live code review
- Available as Java Webstart from
- <http://findbugs.sourceforge.net/demo.html>

Warning Density

Warning density

- Density of high and medium priority correctness warnings

Warnings/KNCSS	Software
0.1	SleepyCat DB
0.3	Eclipse 3.2
0.6	JDK 1.5.0_03
0.6	JDK 1.6.0 b51
0.9	IBM WebSphere 6.0.3

Some new-ish features

some have been around for a while but aren't well known (or well documented)

Annotations for Software Defect Detection

- Allow you to provide lightweight specifications through Java 5.0 annotations
- Examples
 - `@NonNull`
 - `@CheckForNull`
 - `@CheckReturnValue`

JSR-305

- JSR and expert group as formed to develop standard annotations that can be used by multiple tools
- IntelliJ also has annotations for nullness, but they aren't the same
- JSR will develop standard annotations in the javax namespace, with agreements as to their semantics
- Unofficial output: annotated versions of standard libraries

Computing bug history

- Keeps track of when bug are introduced, when they are resolved
- Historical bug data records all bugs reported for any build
- Can see when bugs were introduced and removed
- For example, can report all bugs introduced in the past 3 months

User bug designations annotations

- Our framework and new GUI allow users to designate specific bugs as “Must Fix” or “Not a Bug”
- can also provide free text annotation
- When matching previous analysis results with new analysis results, bugs are matched and annotations carried forward

New GUI

- Provides user designation and annotation support
- Highlights multiple source lines
- Use dragging to reorganize JTree

Command line tools

Command line tools

- We've got a lot of command line tools
 - some ant tasks, need to add more
 - but all the command lines tools can be invoked from within ant
- We need to build a bigger, better tool chain
 - we're open source, we welcome contributions
 - Maven (contributed), Cruise control (?), ...

XML analysis results

- We use XML as the standard output from our analysis engine
- XML analysis results can be filtered, processed, displayed in the GUI, annotated, converted to text or HTML
- XML can be plain, or with messages
 - with text/messages provides all the text to allow you to convert the XML into meaningful HTML without further FindBugs involvement

findbugs

- `findbugs -textui -xml rt.jar >rt.xml`
- run findbugs
 - using the test user interface, rather than the GUI
 - generate XML output, rather than one bug/line
 - also have emacs output mode
- analyze all the classes in `rt.jar`, write the output to `rt.xml`

filterBugs

- `filterBugs -priority H -category C rt.xml hc.xml`
- Read the bugs in `rt.xml`, filter out just the high priority correctness bugs, and write them to `hc.xml`

convertXmlToText

- `convertXmlToText hc.xml`
- convert to simple one bug/line format
- `convertXmlToText -html:fancy.xsl`
- convert to html using fancy.xsl style sheet

listBugDatabaseInfo & setBugDatabaseInfo

- Set information about the analysis
 - -name name this analysis/version
 - -time Give the timestamp for this analysis
 - -addSource add a source directory
 - -findSource find and add all relevant source directories

unionBugs

- combine results from analyzing disjoint classes into a single analysis file
- don't use this if the analysis files contain overlapping results

computeBugHistory

- `computeBugHistory -output db.xml old.xml new.xml`
- combine the analysis results in `old.xml` and `new.xml`
- write a historical analysis to `db.xml`
- `old.xml` can be a historical analysis

matching old bugs with new bugs

- We do a number of clever things (or things we think are clever) to match warnings from an old analysis with warnings in a new analysis
- Line numbers don't matter
- We err on overmatching
- if you modify a method, fixing one null pointer bug, and introducing another in the same method, we may think the bug hasn't changed

mineBugHistory

- `mineBugHistory -formatDates -noTabs db.xml`
- produce a tabular listing of the number of bugs introduced and eliminated in each build/version in a historical analysis

Historical bug databases

- Each historical bug database records a sequence of versions/builds/analysis results
- Each analysis result has a name, a date and a sequence number (starting at 0)

Combing back to filterBugs

- filterBugs has lots of options

```
filterBugs -first 1 db.xml | convertXmlToText
```

- filter out just the warnings that first appeared in sequence # 1 (the second analysis results), and convert the results to text

Importing bugs into your own bug databases

- if you want to bring our results into a database
 - generate xml with messages
 - use instance-hash
 - designed to be unique per bug, and match bugs across versions
 - Not as clever as the approach we use when matching XML, but still clever

FindBugs

Best Practices

What to look at

- First review high and medium priority correctness
- Low priority warnings can be of questionable value
- FindBugs doesn't report these by default
- more there for us to work to improve our accuracy, and figure out how to raise the priority of the important ones and drop the unimportant ones
- Other categories worth examining in a code review, but insisting that they all be reviewed immediately will make people unhappy

Compile with debugging information

- We produce more accurate results and more meaningful messages if the classfiles contain both line numbers and local variable names
- use `javac -g`
- If you are computing historical information, be consistent with whether you generate debugging information

FindBugs plugins

- Carefully consider and review open source FindBugs plugins
- Others have written plugins, some of which generate a lot more false positives or give bad advice
- You can write your own plugins
 - particularly great if you have bugs that are specific to your project

Incremental analysis and/or marking

- For sustainable use, you need to have some way to deal with false positives
 - mark in database
 - Only review new warnings
- Both of these require matching warnings from one analysis with results from a previous analysis

Developers like incremental analysis

- Developers don't like to be asked to scrub a million line code base and review 1000 warnings
- But they don't mind (as much) if you ask them to review a new warning introduced by a change they just made
- false positive rate still matters

Questions?