Finding Bugs is Easy

William Pugh
Dept. of Computer Science
University of Maryland
http://findbugs.sourceforge.net/
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, Sun's Appserver
- real defects, stuff that should be fixed
- hundreds is conservative, probably *thousands*
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?

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

• Oops

• This code is from Eclipse 3.0.0 M8

• 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?
Infinite recursive loop

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

```java
/** 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

```java
public String foundType() {
    return this.foundType();
}
```
- Smart people make dumb mistakes
- Embrace and fix your dumb mistakes
Infinite Recursive Loops: Sun JDK history

I inform Sun of infinite recursive loops in their code.
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:

```java
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):

```java
protected Node findNode(Fqn fqn, ...) {
    int treeNodeSize = fqn.size();
    ...
    if (fqn == null) return null;
```
Other bug patterns

- Classes that implement Serializable but aren't Serializable
- Inconsistent synchronizations
- Bad use of wait/notify
- Streams not closed
Behavior Annotations

- Allow you to provide lightweight specifications through Java 5.0 annotations
- Examples
  - @NonNull
  - @CheckForNull
  - @CheckReturnValue
Why you should pay attention

• Our false positive rates are low
  • many warnings are completely accurate
  • target false positive rate is less than 50%

• We report simple, shallow bugs
  • easy to evaluate and fix

• If we can find this many bugs in your tested, deployed code, what could we do on the developer's desktop?
Glassfish b10 warnings (M/H correctness)

- 29 ignored return values
- 59 classes that define equals but not hashCode
- 9 calls to equals that will always return false
- 18 statements and 98 branches that if ever executed are guaranteed to throw a NPE
- 10 methods that, if ever called, will call themselves again in a infinite recursive loop
- 1 impossible casts
FindBugs workflow

- Running FindBugs
- FindBugs output
- Warning attributes
- Filtering warnings
- False positive suppression
- Warning history
Running FindBugs

- FindBugs analyzes bytecode
- Runs in a number of environments
  - command line tool, ant task
  - stand-alone Swing GUI
  - Eclipse plugin (NetBeans under dev.)
- Open source, more stuff added all the time
FindBugs Output

- Analysis output can be generated/saved as plain text or XML
- various tools to convert XML to various report formats, such as HTML
Warning attributes

- **Priority**: high, medium or low

- **Category**: e.g., correctness, performance, multithreaded correctness, vulnerability to malicious code

- **Kind**: e.g., ignored return value, dereference of null pointer

- **Pattern**: e.g. equals method doesn't handle null argument
Warning Filtering

- Within IDE's and reporting tools, can specify which characteristics you are (or are not) interested in seeing
  - e.g., normally don't display low priority warnings
- Not all bug patterns are universally appropriate
  - perhaps you don't care about vulnerability to untrusted code
False positive suppression

• We will generate (some) false positives
  • a warning that is just plain wrong
  • or a warning that is correct but developer doesn't believe warrants a code change

• Can be suppressed via:
  • Java 5.0 annotation
  • specific filter file
  • annotation and history...
Annotating a warning

- In Swing tool, can annotate a warning as a false positive
- stored in warning database, not source
- custom annotations as well (e.g., security review)
- Ability to perform annotations in Eclipse, other tools under development
Warning History

- When program is modified
  - analysis is rerun
  - warnings from previous analysis and current analysis are matched up
  - annotations are carried forward
- Database stores history of warnings
  - when they were created, etc.
Fuzzy matching

- We perform fuzzy matching of old and new warnings
- want to match up a warning even if you modified the method it occurs in
- Work from precise to fuzzy, pairing up warnings
- can control how fuzzy to get, but default is pretty fuzzy
Using history

- Can filter or display only warnings introduced since last release (or in the last week)
- Can compute statistics over time about warning occurrences in software
History stuff is ready for beta

- The stuff to record history, propagate annotations forward, etc is ready for alpha/beta testing
- Works, but need feedback