Improving Software Quality with Static Analysis

William Pugh
Professor, Univ. of Maryland
http://www.cs.umd.edu/~pugh
Static Analysis

- Analyzes your program without executing it
- Doesn’t depend on having good test cases
  - or even any test cases
- Generally, doesn’t know what your software is supposed to do
  - Looks for violations of reasonable programming
    - Shouldn’t throw NPE
    - Shouldn’t allow SQL injection
- Not a replacement for testing
  - Very good at finding problems on untested paths
  - But many defects can’t be found with static analysis
Common Wisdom about Bugs and Static Analysis

- 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 finding them must require sophisticated static analysis techniques
  - I tried lint and it sucked: lots of warnings, few real issues
Can You Find The Bug?

if (listeners == null)
    listeners.remove(listener);

- JDK1.6.0, b105, sun.awt.x11.XMSelection
- lines 243-244
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 Categories
Selected categories for today's discussion

- Correctness - the code seems to be clearly doing something the developer did not intend
- Bad practice - the code violates good practice
Bug Patterns

- Some big, broad and common patterns
  - Dereferencing a null pointer
  - An impossible checked cast
  - Methods whose return value should not be ignored
- Lots of small, specific bug patterns, that together find lots of bugs
  - Every Programming Puzzler
  - Every chapter in *Effective Java*
  - Many postings to http://thedailywtf.com/
The code in the example file needs to be re-written. The application may look like this:

```java
String s = readInput();
String trimmed = s.trim();
int length = trimmed.length();
System.out.println(length);
```

In this example:
- The `readInput()` method reads a line of input from the user.
- The `trim()` method removes leading and trailing whitespace from the string.
- The `length()` method returns the length of the trimmed string.
- The `System.out.println(length);` method prints the length of the trimmed string to the console.

The use of `trim()` is important because it ensures that the input is properly formatted before it is used in the application. This helps to prevent potential issues caused by leading or trailing whitespace.

**Analysis Techniques:**

- **Local pattern matching**
  - If you invoke `String.toLowerCase()`, don't ignore the return value.
- **Intraprocedural dataflow analysis**
  - Null pointer, type cast errors.
- **Interprocedural method summaries**
  - This method always dereferences its parameter.
- **Context sensitive interprocedural analysis**
  - Interprocedural flow of untrusted data:
    - SQL injection, cross site scripting.

**Whatever you need to find the bugs:**

The techniques described above can help to identify and fix bugs in the application. By using local pattern matching, intraprocedural dataflow analysis, interprocedural method summaries, and context sensitive interprocedural analysis, developers can detect issues in the code and improve the application's security and reliability.
Infinite recursive loop

... Students are good bug generators

- 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 JDK1.6.0-b13

- Found 5 infinite recursive loops
- Including one written by Joshua Bloch

```java
public String foundType() {
    return this.foundType();
}
```

- Smart people make dumb mistakes
  - 27 across all versions of JDK, 40+ in Google’s Java code
- Embrace and fix your dumb mistakes
Finding Null Pointer Bugs with FindBugs

- FindBugs looks for a statement or branch that, if executed, guarantees a null pointer exception.
- Either a null pointer exception could be thrown, or the program contains a statement/branch that can’t be executed.

- Could look for exceptions that only occur on a path:
  - e.g., if the condition on line 29 is true and the condition on line 38 is false, then a NPE will be thrown.
  - but would need to worry about whether that path is feasible.
Null Pointer Bugs Found by FindBugs

JDK1.6.0-b105

- 109 statements/branches that, if executed, guarantee NPE
  - We judge at least 54 of them to be serious bugs that could generate a NPE on valid input

- Most of the others were deemed to be unreachable branches or statements, or reachable only with erroneous input
  - Only one case where the analysis was wrong
Examples of null pointer bugs

simple ones

// com.sun.corba.se.impl.naming.cosnaming.NamingContextImpl
if (name != null || name.length > 0)

// com.sun.xml.internal.ws.wsdl.parser.RuntimeWSDLParser
if (part == null | part.equals(""))

// sun.awt.x11.ScrollPanePeer
if (g != null)
    paintScrollBars(g, colors);
g.dispose();
Redundant Check For Null

Also known as a reverse null dereference error

- Checking a value to see if it is null
  - When it can't possibly be null

// java.awt.image.LoopupOp, lines 236-247

```java
public final WritableRaster filter(
    Raster src, WritableRaster dst) {
    int dstLength = dst.getNumBands();
    // Create a new destination Raster, if needed
    if (dst == null)
        dst = createCompatibleDestRaster(src);
```
Redundant Check For Null

Is it a bug or a redundant check?

- Check the JavaDoc for the method
- Performs a lookup operation on a Raster.
  - If the destination Raster is null,
  - a new Raster will be created.
- Is this case, a bug
  - particularly look for those cases where we know it can't be null because there would have been a NPE if it were null
Bad Method Invocation

- Methods whose return value shouldn't be ignored
  - Strings are immutable, so functions like `trim()` and `toLowerCase()` return new String
- 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
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);
}
Type Analysis

- Impossible checked casts
- Useless calls
  - equals takes an `Object` as a parameter
    - but comparing a `String` to `StringBuffer` with `equals(....)` is pointless, and almost certainly not what was intended
  - `Map<K,V>.get` also takes an `Object` as a parameter
    - supplying an object with the wrong type as a parameter to `get` doesn't generate a compile time error
    - just a `get` that always returns null
Lots of Little Bug Patterns

• checking if \( d == \text{Double.NaN} \)

• Bit shifting an \text{int} by a value greater than 31 bits

• Every Puzzler this year
  • more than half for most years
When Bad Code Isn't A Bug

- Static analysis tools will sometimes find ugly, nasty code
  - that can't cause your application to misbehave
- Cleaning this up is a good thing
  - makes the code easier to understand and maintain
- But for ugly code already in production
  - sometimes you just don't want to touch it
- We've found more cases like this than we expected
When Bad Code Isn't A Bug

bad code that does what it was intended to do

// com.sun.jndi.dns.DnsName, lines 345-347
if (n instanceof CompositeName) {
    // force ClassCastException
    n = (DnsName) n;
}

// sun.jdbc.odbc.JdbcOdbcObject, lines 85-91
if ((b[offset] < 32) || (b[offset] > 128)) {
    asciiLine += ".";
}
When Bad Code Isn't A Bug

Code that shouldn't go wrong

// com.sun.corba.se.impl.dynamicany.DynAnyComplexImpl
String expectedMemberName = null;
try {
    expectedMemberName
        = expectedTypeCode.member_name(i);
} catch (BadKind badKind) { // impossible
} catch (Bounds bounds) { // impossible
}
if ( !(expectedMemberName.equals(memberName) ... ))
{

When Bad Code Isn't A Bug

When you are already doomed

// com.sun.org.apache.xml.internal.security.encryption.XMLCiper
// lines 2224-2228

if (null == element) {
    //complain
}   
String algorithm = element.getAttributeNS(...);
Overall Correctness Results From FindBugs


- JDK1.6.0-b105
  - 379 correctness warnings
    - we judge that at least 213 of these are serious issues that should be fixed
- Google's Java codebase
  - over a 6 month period, using various versions of FindBugs
  - 1,127 warnings
  - 807 filed as bugs
  - 518 fixed in code
Bad Practice

- A class that defines an equals method but inherits hashCode from Object
  - Violates contract that any two equal objects have the same hash code
- equals method doesn't handle null argument
- Serializable class without a serialVersionUID
- Exception caught and ignored
- Broken out from the correctness category
Fixing hashCode

• What if you want to define equals, but don't think your objects will ever get put into a HashMap?

• Suggestion:
  
  ```java
  public int hashCode() {
    assert false : "hashCode method not designed";
    return 42;
  }
  ```
Use of Unhashable Classes

• FindBugs previously reported all classes that defined equals but not hashCode as a correctness problem
  • but some developers didn’t care
• Now reported as bad practice
  • but separately report use of such a class in a HashMap/HashTable as a correctness warning
Integrating Static Analysis

● Want to make it part of your development process
  ● Just like running unit tests
● Have to tune the tool to report what you are interested in
  ● Different situations have different needs
● Need a workflow for issues
  ● Almost all tools will report some issues that, after reviewing, you decide not to fix
  ● Need to have a way to manage such issues
Defect/Issue Workflow

- How do issues get reviewed/audited?
- Can you do team auditing and assign issues?
- Once you've reviewed an issue, does the system remember your evaluation when it analyzes that code again?
  - even if it is now reported on a different line number?
- Can you identify new issues
  - since last build?
  - since last release to customer/production?
Learning from mistakes

- With FindBugs, we've always started from bugs
- When you get bit by a bug
  - writing a test case is good
  - considering whether it can be generalized into a bug pattern is better
- You'd be surprised at the number of times you make a mistake so stupid “no one else could possibly make the same mistake”
  - but they do
Wrap up

- Static analysis is effective at finding bad code
  - Is bad code found by static analysis an important problem?
- Getting static analysis into the software development process can’t be taken for granted