

Using FindBugs



Outline

- **Motivation**
- The FindBugs Tool
- How it works
- Conclusions

David Hovemeyer <daveho@cs.umd.edu>
<http://findbugs.sourceforge.net>

1

2

Motivation

- Software is hard to get right
 - Complex library APIs
 - Difficult language features: e.g., threads
- Nobody is perfect 100% of the time
- Result: bugs
 - Wasted development time, frustrated users

3

Using Tools to Find Bugs

- We can write programs, ‘bug checkers’, to analyze code for potential errors
- Running the bug checker produces a list of potential bugs in the code
- Goal: find bugs early
 - Before debugging and testing
 - Before program is distributed to users

4

Observation

- Many bugs share common characteristics
- *Bug Pattern*: a code idiom that is frequently an error
- Is it possible to detect instances of bug patterns automatically?
 - Yes!

5

Our Approach

- We have developed a tool, FindBugs, to find instances of bug patterns
- We try to find the simplest approach to detecting potential bugs
- This approach can be effective:
 - About 45 bug patterns recognized
 - Hundreds of bugs found in real applications

6

Limitations

- *Static Analysis* is the process of analyzing a program's code to find out how the program will behave at runtime
- Nontrivial properties of programs are undecidable
 - E.g., the halting problem
- We can never determine all possible program behaviors

7

Consequences of Imprecision

- Given that we can't predict all possible program behaviors:
 - We try to infer *likely* program behavior
- False positives:
 - Tool reports a bug that can't really happen
- False negatives:
 - Tool fails to report a bug than can happen

8

Limitations of Tools

- Bug finding tools are not a panacea
- They generally can't help ensure that your code does what you intend
- However, they are a useful first line of defense
- Helpful when learning new areas of the language

9

Outline

- Motivation
- **The FindBugs Tool**
- How it works
- Conclusions

10

FindBugs

- A tool to find instances of bug patterns in Java programs
 - Misuses of API functions
 - Language semantics: e.g., null pointer exceptions
 - Thread problems
- Written in Java
- Developed by Bill Pugh

11

Installing FindBugs

- Download from FindBugs website:
 - <http://findbugs.sourceforge.net>
- See instructions in manual:
 - <http://findbugs.sourceforge.net/manual/index.html>
- Main user interfaces: command line, Swing GUI
- Eclipse plugin: recently added, still needs work
 - We are actively working to make it better

12

Running FindBugs

- FindBugs analyzes Java class files
 - Individual class files
 - Jar, zip archives
 - Directories containing class files
- Produces results in two formats:
 - Minimal text output
 - XML output (can view in GUI: recommended)

13

Demo

14

What Can FindBugs Find?

- We have used FindBugs to find hundreds of bugs in real applications
- We have been surprised at how obvious many of the bugs are
- A few examples...

15

Null Dereference

- Dereferencing a null pointer is almost always a mistake
- Eclipse 2.1.0,
org.eclipse.jdt.internal.ui.javaeditor.ClassFileEditor
if (**entry == null**) {
 IClasspathContainer container=
 JavaCore.getClasspathContainer(**entry.getPath()**,
 root.getJavaProject());
 ...
}

16

Null Dereference (2)

- Eclipse 2.1.0,
org.eclipse.help.ui.internal.search.HelpSearchPage
if (!searchQueryData.isBookFiltering()
 && (**lastWS != null** || **lastWS.length()** > 0)) {
 ...
}

17

Unused Return Value

- String objects are immutable
 - Methods that modify Strings return a new object
- Example: Eclipse 2.1.0:
if (i < label.length())
 label= label.substring(0, i) + label.substring(i+1);
else
 label.substring(0, i);

18

Suspicious Reference Comparison

- Objects should generally be compared using the `equals(Object)` method, not `==` and `!=` operators
 - Those operators test *reference equality*, not *object equality*
- GNU Classpath 0.06,
`gnu.java.net.protocol.jar.Handler.parseURL()`

```
String file = url.getFile();  
if (file != null && file != ""){ //has context url
```

19

Thread Bug Patterns

- FindBugs looks for several bug patterns related to threads
 - Problems starting threads
 - Fields locked inconsistently
 - Wait/notify problems
- This may be helpful when you do Project 4

20

Outline

- Motivation
- The FindBugs Tool
- **How it works**
- Conclusions

21

How It Works

- Two approaches to analyzing Java programs:
 - Source code
 - Bytecode (class files)
- Analyzing source code has some advantages, but is more complicated
- We analyze bytecode
 - Using the Apache Byte Code Engineering Library (BCEL)

22

Java Bytecode

- Java source files are translated into *bytecode*
 - Essentially, a machine language for the Java Virtual Machine
- Instead of registers, bytecode instructions use an operand stack
 - Each value on the stack is an object reference or numeric value
- Bytecode is very easy to analyze

23

Java Bytecode Example

- The command “`javap -c classname`” prints the bytecode for methods in a class
- Demo...

24

Analyzing Bytecode

- How can we analyze bytecode to figure out what it does?
- FindBugs uses several approaches:
 - Simple: Scanning
 - More complex: Scanning with control flow
 - Most complex: Dataflow analysis
- Very similar to techniques used in compilers

25

Bytecode Scanning

- Really simple approach: just scan through bytecode instructions, driving a state machine
- Example: unconditional wait

```
// Wrong:
while (!someCondition) {
    synchronized (lock) {
        lock.wait();
    }
}

// Right:
synchronized (lock) {
    while (!someCondition) {
        lock.wait();
    }
}
```

26

Recognizing Unconditional Wait

- Observation: a unconditional wait is when a lock is acquired, immediately followed by a call to `Object.wait()`
 - With no intervening branches
- Acquiring a lock: `monitorenter`
- Calling wait: `invokevirtual Object.wait()`
- Scan for these instructions!
- Example...

27

Control Flow

- Scanning is good for bug patterns that don't involve control flow
- Often, control flow is important
 - Conditional control flow gives us information: e.g.
`if (foo == null) {`
`...foo is null here...`
- Control Flow Graph (CFG) gives us this information

28

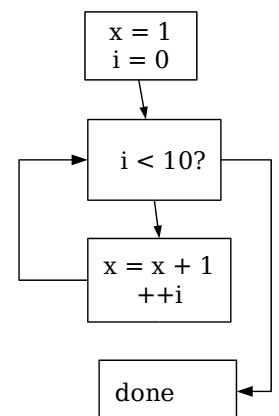
Control Flow Graphs (CFGs)

- A Control Flow Graph is a data structure comprised of *basic blocks* and *control edges*
- Basic block: linear sequence of instructions with no control flow
- Control edge: indicates control transfer from one block to another

29

CFG Example

```
x = 1;
for (i = 0; i < 10; ++i) {
    x = x + 1;
}
```



30

Using CFGs

- How does the CFG help us?
- Scanning approach can now take control flow into account
 - E.g., for an “if” statement, continue scanning on both branches
 - Note: this can lead to exponential cost
- *Dataflow analysis*

31

Dataflow Analysis

- A technique used to conservatively approximate facts about a program
 - Used extensively in compilers
- E.g.: “where might a null pointer be dereferenced”?
- Models the values of variables (locations on the operand stack), taking control flow into account

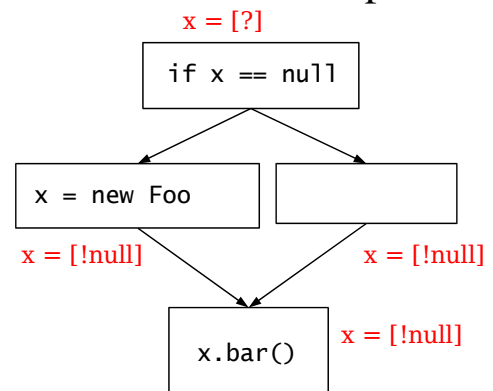
32

Dataflow (continued)

- A dataflow value is an abstract representation of a runtime value
 - E.g.: “value is null”, “value is not null”, “value could be either”
- Transfer functions take dataflow values and model the effects of a basic block
- A merge function combines data flow values for when control paths merge

33

Dataflow Example



34

Dataflow (continued)

- Dataflow values computed iteratively
 - To handle loops in the CFG
- How well does it work?
 - Pretty well
 - However, we lose information at control merges
 - Modeling state of heap variables: difficult

35

Outline

- Motivation
- The FindBugs Tool
- How it works
- **Conclusions**

36

Conclusions

- Tools can be useful for finding bugs
 - Help steer you toward correct use of language features and APIs
- FindBugs is a work in progress:
 - We will continue to add new bug patterns
 - Eclipse integration: coming soon
 - Send us your ideas!

Related Work

- There are several similar tools available:
 - PMD: <http://pmd.sourceforge.net/>
 - Jlint: <http://artho.com/jlint/>
 - CheckStyle: <http://checkstyle.sourceforge.net/>
- They each have different strengths; all of them are worth checking out