Finding Bugs is Easy [2]

Overview

- Motivation
- Problem
- Approach
- Experiments

Motivation

- Bugs are a serious problem
- Many techniques developed to automatically find bugs
  - Formal methods
  - Sophisticated program analysis
- Existing techniques are difficult to apply, and aren't always effective in finding real bugs

3

Problem

- How to detect bugs using simple and brad techniques, rather than focused and narrow techniques?

4

Pattern-based Bug Detection

- Bug patterns are code idioms that are often errors
- Start by looking at actual bugs in real code, extract the bug patterns, and then develop bug pattern detectors to find similar bugs

5

Bug Pattern Detectors

- 45 bug pattern detectors are implemented using BCEL
- Four categories of detectors:
  - Single-threaded correctness issue
  - Thread/synchronization correctness issue
  - Performance issue
  - Security and vulnerability to malicious untrusted code

6

Four categories of implementation strategies

- Class structure and inheritance hierarchy only
  - Some of the detectors simply look at the structure of analyzed classes without looking at the code
  - E.g., equals() and hashcode() should be defined together
- Linear code scan
  - No control flow analysis
  - E.g., bad covariant definition of equals:
    public boolean equals(Foo obj) {…}

7

Four categories of implementation strategies

- Control sensitive
  - Control flow analysis
  - E.g., WaitNotInLoop:
    - Object.wait() method waits on a monitor for another thread to call notify() or notifyAll()
    - Usually, wait() is waiting for a particular condition to become true
    - The most robust way is to put it in a loop, where the waited-for condition is checked each time the thread wakes up

8

Four categories of implementation strategies

- Data flow
  - Control and data flow analysis
  - E.g., null pointer dereference

```
if (foo == null)
{
   …
   foo.f …
}
```

9

Selected Bug Pattern Detectors

Inconsistent Synchronization

- Detect fields which are sometimes accessed with a self lock held and sometimes without are candidate instances
- Several heuristics are used to reduce the number of false positives
  - Public or volatile fields are ignored
  - Fields that are never read without a lock are ignored (?)

11

Inconsistent Synchronization

- Heuristics to reduce false positives (cont'd)
  - Accesses in object lifecycle methods (such as constructors and finalizers), or in nonpublic methods reachable only from these lifecycle methods, are ignored
  - If there is a high proportion of unlocked accesses (>=1/3), ignore it
    - 2(RU + 2WU) > (RL + 2WL)

12

Static Field Modifiable by Untrusted Code

- Untrusted code is allowed to modify static fields, thereby modifying the behavior of the library for all uses
  - A static non-final field has public or protected access
  - A static final field has public or protected access, and references a mutable structure such as an array or Hashtable
  - A method returns a reference to a static mutable structure such as an array or Hashtable

13

Where are bug patterns from?

- Many of the bug patterns are suggested by Java semantics
- A number of books describe potential Java coding pitfalls
- Several bug patterns are observed in student projects and later implemented
- Several bug patterns are suggested by FindBugs users

14

Evaluation

- Run FindBugs on six applications
  - GNU Classpath, version 0.08
  - rt.jar from Sun JDK 1.5.0, build 59
  - Eclipse, version 3.0
  - DrJava, version stable-20040326
  - JBoss, version 4.0.0RC1
  - jEdit, version 4.2pre15

15

	classpath-0.06					rt.jar 1.5.0 build 18				
	warnings	serious	harmless	dubious	false pos	warnings	serious	harmless	dubious	false pos
DC	0	—	—	—	—	6	83%	0%	0%	16%
IS2	18	72%	16%	0%	11%	52	30%	63%	0%	5%
NP	7	85%	0%	0%	14%	21	95%	0%	0%	4%
OS	9	22%	33%	22%	22%	5	0%	0%	0%	100%
RR	7	100%	0%	0%	0%	10	100%	0%	0%	0%
RV	11	45%	0%	0%	54%	2	100%	0%	0%	0%
UR	3	100%	0%	0%	0%	3	100%	0%	0%	0%
UW	2	0%	0%	0%	100%	6	33%	0%	0%	66%
Wa	2	0%	0%	0%	100%	6	16%	0%	0%	83%

	eclipse-2.1.0					drjava-stable-20030822				
	warnings	serious	harmless	dubious	false pos	warnings	serious	harmless	dubious	false pos
NP	43	93%	0%	6%	0%	0	—	—	—	—
OS	16	6%	6%	18%	68%	5	40%	0%	40%	20%
RR	22	4%	0%	0%	95%	0	—	—	—	—
RV	9	100%	0%	0%	0%	0	—	—	—	—
UR	0	—	—	—	—	1	100%	0%	0%	0%
UW	0	—	—	—	—	3	66%	0%	0%	33%

	jboss-3.2.2RC3					jedit-4.1				
	warnings	serious	harmless	dubious	false pos	warnings	serious	harmless	dubious	false pos
IS2	2	50%	0%	0%	50%	1	0%	100%	0%	0%
NP	10	100%	0%	0%	0%	0	—	—	—	—
OS	2	100%	0%	0%	0%	1	100%	0%	0%	0%
RR	0	—	—	—	—	1	100%	0%	0%	0%
RV	2	0%	0%	0%	100%	0	—	—	—	—
UR	2	50%	0%	0%	50%	2	50%	0%	50%	0%
UW	1	100%	0%	0%	0%	1	100%	0%	0%	0%
Wa	0	—	—	—	—	2	50%	0%	0%	50%

16

Interesting Observations

- No type of bug has been so "dumb" or "obvious" that we have failed to find examples of it in real code
- The potential for misuse of language features and APIs is enormous
- FindBugs can effectively raise the awareness of developers about subtle correctness issues

17

Reference

[1] David Hovemeyer, William Pugh, Finding Bugs is Easy, ACM SIGPLAN Notices '04

18