Improving Your Software Using Static Analysis to Find Bugs

Brian Cole Daniel Hakim David Hovemeyer *
Reuven Lazarus William Pugh Kristin Stephens

Dept. of Computer Science
University of Maryland
College Park MD 20742 USA

Abstract

FindBugs looks for bugs in Java programs. It is based on the concept of bug patterns. A bug pattern is a code idiom that is often an error. Bug patterns arise for a variety of reasons, such as difficult language features, misunderstood API semantics, misunderstood invariants when code is modified during maintenance, and simple mistakes such as typos.

FindBugs uses static analysis to inspect Java bytecode for occurrences of bug patterns. We have found that FindBugs finds real errors in most Java software. Because its analysis is sometimes imprecise, FindBugs can report false warnings, which are warnings that do not indicate true errors. In practice, the rate of false warnings reported by FindBugs is generally lower than 50%, often much lower.

Categories and Subject Descriptors D.2.5 [*Testing and Debugging*]: Diagnostics, Symbolic Execution; D.2.2 [*Design Tools and Techniques*]: Programmer workbench

General Terms Human Factors, Languages, Verification

Keywords FindBugs, Java, static analysis

Introduction

FindBugs is a static analysis tool for finding bugs in Java programs. It is based on the concept of *bug patterns*: code idioms that are likely to be errors. FindBugs was developed using a *bug-driven* methodology. Starting from examples of bugs found in real code—production applications, libraries, and student programming projects—we developed the simplest static analysis techniques we could think of that would find at least some instances of similar bugs without generating too many false positives. Often, analysis based on simple pattern matching of bytecode instructions is sufficient to effectively find instances of a bug pattern.

Using this approach, we have found a surprising number of serious bugs in production software. On a typical production Java application or library, FindBugs finds one likely code defect per 800–2400 lines of non-comment source code.

By *likely code defect* we mean it is likely that most programmers would consider it important to correct the defect. Findbugs also finds bugs of lower priority and bugs in categories that some programmers might not care about.

1. Bugs Detected by FindBugs

The spirit of our bug-driven approach means that we attempt to automate recognition of any bug pattern we observe, no matter how unlikely or obscure the bug seems on the surface. Some examples of bug patterns detected by FindBugs include

- Infinite recursive loops: Methods that are guaranteed to invoke themselves again in a recursive loop terminating in a StackOverflowError.
- Statements or branches that if executed are guaranteed to result in a null pointer exception.
- Checked casts that are guaranteed to throw a ClassCastException
- Ignoring a return value that should never be ignored (e.g., `String.replace('.','/')`)
- Public static fields that can be modified by untrusted code

The kinds of bugs detected by FindBugs are often the result of simple mistakes. For example, many of the null pointer bugs detected by FindBugs are the result of using the wrong boolean operator, as shown in Figure 1.

The analysis to detect infinite recursive loops was inspired by code in a student programming project (shown in Figure 2). Although this is obviously a case of a novice programmer being confused about how constructors work, we have found dozens of examples of similar bugs in production software, such as the bug shown in Figure 3. This demonstrates that even experienced developers sometimes make dumb mistakes, and tools to catch such mistakes are valuable.

Table 1 shows the number of occurrences of several FindBugs warning types for a number of widely-used Java applications and libraries. Generally, more than 50% of such warnings correspond to a significant coding defect.

One company recently evaluated FindBugs to determine which bug categories and patterns were important for their codebase. Within those, they judged about 20% of the warnings to be high-

```
Control c= getControl();
if (c == null && c.isDisposed())
  return;
```

Figure 1. A null pointer bug in Eclipse 3.2.0

* Dept. of Physical Science, York College of Pennsylvania

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

Application	Classes	KNCSS	IL	NPE	EC	RV	MS
JDK 1.5.0_03	13,124	809	3	87	9	15	60
JDK 1.6.0-b51	16,361	979	5	99	14	19	105
weblogic90	26,063	1,969	9	315	26	1239	n/a
Eclipse-3.0	27,983	1,667	3	139	17	22	n/a

(FindBugs 1.0)

Column	Description
KNCSS	Thousands of non-commenting source statements
IL	Infinite recursive loop
NPE	Null pointer exception
EC	Incompatible equals comparison
RV	Ignored return value
MS	Mutable public static

Table 1. Warnings reported in several widely-used Java applications and libraries

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

Figure 2. An infinite recursive loop in a student programming project

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

Figure 3. An infinite recursive loop found in an early build of Sun's JDK 1.6

priority bugs, about 60% to be low-priority bugs, and about 20% not to be bugs.

2. Availability and Adoption

More information about FindBugs, including papers, presentation and software, is available from the FindBugs web site at `http://findbugs.sourceforge.net`. We have made FindBugs generally available under the open-source LGPL license. Since its initial public release, FindBugs has been downloaded more than 250,000 times. FindBugs users frequently contribute enhancements such as bug fixes, detectors for new bug patterns, and internationalization of messages.

Several FindBugs interfaces are available. The most basic is a batch-mode interface that performs static analysis on an application or library and saves the resulting static analysis warnings to a database file. Integration with the Apache Ant and Maven build tools are supported. A Swing GUI is available for visually browsing a warning database. FindBugs is available as a plugin for several Java development environments, including Eclipse, Netbeans, and BlueJ.

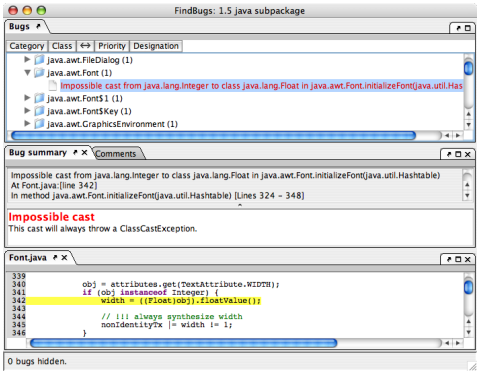

Figure 4. Swing GUI for FindBugs

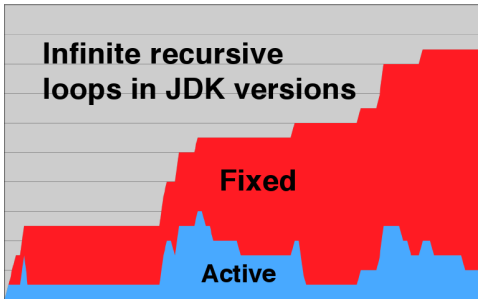

Figure 5. Number of infinite recursive loop bugs in the core JDK libraries over time

3. Bug History

Recently, we have added a history feature to FindBugs. This feature allows FindBugs to match "equivalent" static analysis warnings resulting from the analysis of two different versions of the same software artifact. This allows the user to perform many useful tasks.

One important task is to select the subset of warnings that is new in the most recent version of the software. Because some of the warnings emitted by FindBugs are false positives, it is important to allow developers to examine new warnings without having to reexamine all of the existing false positives.

Another useful capability added by the history feature is tracking when bugs are introduced and fixed. Figure 5 shows a plot of the number of active and fixed infinite recursive loop bugs in the core JDK libraries over time.

4. Conclusions

Using an approach that starts from examples of real bugs and uses simple static analysis to find similar bugs, we have developed a tool that finds thousands of serious bugs in production software, demonstrating that static analysis is a useful line of defense against coding errors.

Acknowledgments

Many individuals have contributed to FindBugs. An incomplete list is available at `http://findbugs.sourceforge.net/manual/acknowledgments.html`.

FindBugs is sponsored by Fortify Software, and also supported by Google, Sun Microsystems, and National Science Foundation grants ASC9720199 and CCR-0098162.