Using FindBugs On Production Software

Nathaniel Ayewah, William Pugh
University of Maryland
ayewah,pugh@cs.umd.edu

J. David Morgenthaler, John Penix, YuQian Zhou
Google, Inc.
jdm,jpenix,zhou@google.com

Abstract

This poster will present our experiences using FindBugs in production software development environments, including both open source efforts and Google's internal code base. We summarize the defects found, describe the issue of real but trivial defects, and discuss the integration of FindBugs into Google's Mondrian code review system.

Categories and Subject Descriptors F.3.2 [*Semantics of Programming Languages*]: Program analysis; D.2.4 [*Software/Program Verification*]: Reliability

General Terms Experimentation, Reliability, Security

Keywords FindBugs, static analysis, bugs, software defects, bug patterns, false positives, Java, software quality, Google

1. Introduction

Static analysis tools for software defect detection are becoming widely used in practice. However, there is little public information regarding the experimental evaluation of the accuracy and value of the warnings these tools report. In this poster, we discuss the warnings found by FindBugs, a static analysis tool that finds defects in Java programs. We discuss the kinds of warnings generated and the classification of warnings into false positives, trivial bugs and serious bugs. We also provide some insight into why static analysis tools often detect true but trivial bugs, and some information about defect warnings across the development lifetime of software release. We report data on the defect warnings in Sun's Java 6 JRE, in Sun's Glassfish J2EE server, Eclipse 3.3, and in portions of Google's Java codebase. Finally, we report on some experiences from incorporating static analysis into the software development process at Google. This poster reprises and updates our PASTE 2007 paper [1] with additional experiences during the spring and summer of 2007, the integration of FindBugs into Google's Mondrian code review system, and preliminary work on an observational study of the use of static analysis in production.

Static analysis for software defect detection has become a popular topic, and there are a number of commercial, open source and research tools that perform this analysis. Unfortunately, there is little public information about the experimental evaluation of these tools with regards to the accuracy and seriousness of the warnings they report. Commercial tools are very expensive and generally come with license agreements that forbid the publication of any experimental or evaluative data.

In this poster, we report on results of running FindBugs against several large software projects, including Sun's JDK, Eclipse and and portions of Google's Java code base. Due to space and time constraints, only a limited selection of data is included in this extended abstract, but more complete data is presented in our poster and available from the FindBugs web site.

2. FindBugs

FindBugs is an open source static analysis tool that analyzes Java classfiles looking for programming defects. The analysis engine reports nearly 300 different bug patterns. FindBugs has a plugin architecture, in which detectors can be defined, each of which may report several different bug patterns. Many simple detectors use a visitor pattern over the classfiles and/or the method bytecodes, often using a state machine and/or information about the types, constant values, special flags (e.g., is this value the result of calling hashCode) and about values stored on the stack or in local variables. But detectors can also traverse the control flow graph, using the results of data flow analysis such as type information, constant values and nullness. The data flow algorithms all generally use information from conditional tests, so that information from `instanceof` tests and null tests are incorporated into the analysis results.

3. True But Low Impact Defects

One unexpected finding from looking at a number of real defect warnings is that there are a number of cases in which FindBugs has correctly diagnosed what seems to be an obvious defect, yet it is also clear that the defect will not result in measurable misbehavior of the program. This issue is dis-

OOPSLA'07, October 21–25, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-865-7/07/0010.

```
// sun.jdbc.odbc.JdbcOdbcObject, lines 85-91
if ((b[offset] < 32) || (b[offset] > 128)) {
  asciiLine += ".";
}
else {
  asciiLine += new String (b, offset, 1);
}
```

Figure 1. Masked Error

cussed more fully in our PASTE 2007 paper [1], but we give one example here.

Figure 1 shows a masked error. The variable `b` is a byte array. Any value loaded from a byte array is treated as a signed byte and sign extended to an integer in the range -128 to 127. Thus, the test `b[offset] > 128` will always be false. However, the cases where the value would be greater than 128 if treated as an unsigned byte are caught by the test `b[offset] < 32`, so the defect cannot actually cause misbehavior. This example also shows a fairly common phenomenon where warnings are closely associated with other questionable code. Here, the code is constructing single character strings and incrementally appending them to a String (which has quadratic complexity) rather than simply appending a character to a StringBuffer.

3.1 When should such defects be fixed?

Should a defect that doesn't cause the program to significantly misbehave be fixed? The main arguments against fixing such defects is that they require engineering resources that could be better applied elsewhere, and that there is a chance that the attempt to fix the defect will introduce another, more serious bug that *does* significantly impact the behavior of the application. The primary argument for fixing such defects is that it makes the code easier to understand and maintain, and less likely to break in the face of future modifications or uses.

When sophisticated analysis finds an interprocedural error path involving aliasing and multiple conditions, understanding the defect and how and where to remedy it can take significantly more engineering time, and it can be more difficult to have confidence that the remedy resolves the issue without introducing new problems. However, most of the warnings generated by FindBugs are fairly obvious once you know what to look for, can be understood by looking at a few lines of code, and have straight forward and obvious fixes. Thus, with FindBugs warnings it is often possible to just understand the defect and fix it without expending the effort required to do a full analysis of the possible impact of the fault (or the fix) on application behavior. However, even simple defects suggest holes in test coverage and additional unit tests should be created to supplement defect fixes.

4. Common bug patterns

FindBugs reports hundreds of bug patterns, and there is a long tail of bug patterns (i.e., many bug patterns that effectively report problems but do so rarely; perhaps once per million lines of code). However, there are some basic bug patterns that are relatively common, including null pointer dereferences, invoking toString or equals on an array, ignoring an important return value, comparing incomparable types using equals and infinite recursive loops.

5. Experiences at Google

At Google, FindBugs is automatically run over any modified code, generating XML. The XML file is then imported into a database, which also contains reports from other static analysis tools. The instance-hash mechanism described in [2] is used to allow a defect to be matched with previous reportings of that warning. Previously, the database was reviewed by two engineers, who performed bug triage, evaluating which warnings should be reported to developers. A cost/benefits analysis showed that this would be the most effective way to evaluate which bug patterns were consistently appropriate to report to developers and to gain more experience with capturing and reporting FindBugs results. From this period, the result include:

- 1,307 issues identified and reviewed
- 938 reported as bugs to developers
- 651 fixed by developers

After gaining experience and confidence with FindBugs, we wanted to move the use of static analysis closer to developers, since the most effective time to have a developer review a static analysis warning is as soon as possible after they wrote or touched the code in which the warning is generated. Thus, we have just finished incorporating FindBugs into Google's Mondrian code review system. Whenever an engineer submits a code change for review by a coworker, the person doing the review sees the issues identified by static analysis in both the original and modified version of the code.

More details and experimental details available at our poster.

References

[1] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou. Evaluating static analysis defect warnings on production software. In *PASTE '07: Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 1–8, New York, NY, USA, 2007. ACM Press.

[2] J. Spacco, D. Hovemeyer, and W. Pugh. Tracking defect warnings across versions. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 133–136, New York, NY, USA, 2006. ACM Press.