Finding Bugs in Eclipse (*Demonstration*)

William Pugh

University of Maryland

pugh@cs.umd.edu

Abstract

This will be a live demonstration of FindBugs, a static analysis bug finding tool, on the current development version of Eclipse 3.4. FindBugs reports issues such as null pointer dereferences, comparing incompatible types with equals, invalid method calls, infinite recursive loops, bad integer operations, and more. FindBugs reports more than 400 such issues in Eclipse 3.3.

During this demonstration, we'll give a quick overview of the FindBugs GUI and walk through 10-20 bug warnings, categorize each warning as to whether or not fixing the issue is important, and enter comments about the bug. We'll be able to browse warnings by date of introduction, so we can see if the issues introduced in the past month are more or less serious than the issues that have been in the code base since Eclipse 3.3, 3.2 or earlier. Vocal audience participation is encouraged, and participants with laptops can follow along and enter their own categorization and comments either during the demonstration or afterwards. Audience members with commit privileges to the Eclipse project will get free FindBugs T-shirts.

We'll also briefly demonstrate how to set up FindBugs as part of a production development environment.

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, Eclipse

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. FindBugs [2, 1] is an open source static analysis tool for finding programming errors in Java programs. While many people have read articles or seen presentations on FindBugs, fewer people have actually sat down with FindBugs to see how it works when applied to real software. In this demonstration, we will demonstrate the use of FindBugs on the Eclipse 3.3 release. We will demonstrate the FindBugs graphical user interface (Figure 1), the FindBugs plugin for Eclipse, and the warnings generated by Eclipse's own defect detection analysis. We'll briefly discuss how to set up FindBugs as part of your software development process and how to perform collaborative distributed issue auditing.

FindBugs reports more than thousands of issues for Eclipse, and more than 400 issues categories as "correctness" issues by FindBugs, which are the issues where the tool is most confident that the code does not correctly reflect the developers intentions. Our experience [1] has been that more than half of the correctness issues are issues that developers will want to address by changing the source code. Table 1 gives a summary count of some of the issues found by FindBugs in Eclipse.

Null pointer dereference	139
Reverse null point dereference	70
Doomed equals	31
Non short circuit evaluation	25
Dubious method invocation	15
Self assignment	12
Bad integer operation	18
Ignored return value	7
Doomed checked cast	4
Infinite recursive loop	3

Table 1. Selected correctness issues found in Eclipse

```
// org.eclipse.update.internal.core.ConfiguredSite
// lines 941
if (in == null)
  try {
    in.close();
  } catch (IOException e1) { }
```

Figure 2. Null pointer issue since Eclipse 2.0

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

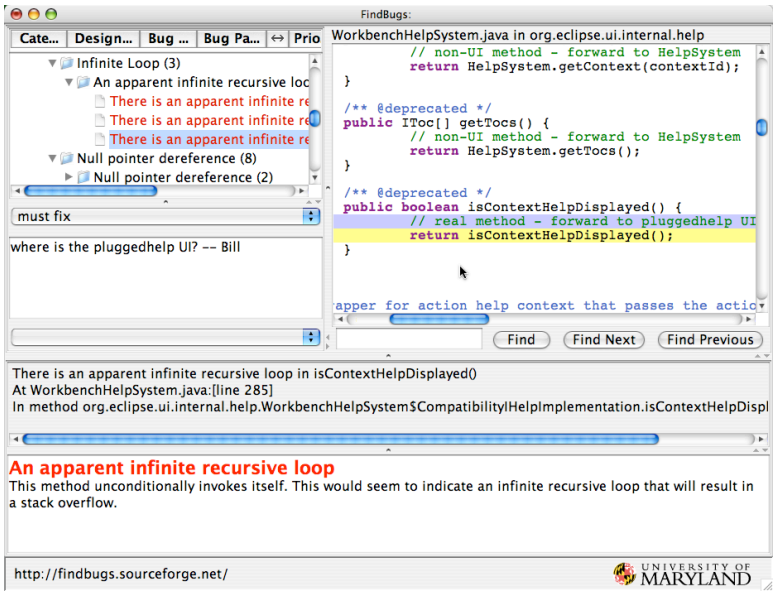

Figure 1. FindBugs GUI

```
// org.eclipse.pde.internal.core.PluginModelManager
// line 839
public void resetState(PDEState state) {
  // the following 2 lines were added after 3.3M7
  if (fState != null && fState.equals(state))
    return;
  // clear all models and add new ones
  int type = IModelProviderEvent.TARGET_CHANGED;
  IModel[] removed = fState.getTargetModels();
```

Figure 3. Null pointer issue introduced after 3.3M7

```
// org.eclipse.core.internal.localstore
//    .FileSystemResourceManager
// line 274
int third = (input.read() & 0xFF);
if (third == -1)
  return IFile.ENCODING_UNKNOWN;
```

Figure 4. Bad integer operation

```
// org.eclipse.swt.graphics.TextStyle
// lines 145-146
  if (underline) hash ^= hash;
  if (strikeout) hash ^= hash;
```

Figure 5. Nonsensical self-operation

During this demonstration, we will review a sample of the issues found by FindBugs in Eclipse. Audience participation will be encouraged in a discussion of which issues warrant being addressed by changes to the source code. Not every issue warants changing the source code, and we'll discuss some of the reasons for that phenominin, and how to deal with it when incorporating static analysis into your software development process.

We'll also discuss tracking issues across multiple versions. This allows us, for example, to determine which issues have been present since Eclipse 2.0 (e.g., the null pointer problem in Figure 2) and which were introduced after 3.3M7 (e.g., the null pointer problem in Figure 3). While FindBugs finds lots of null pointer issues, it also finds many other kinds of issues. Figure 4 shows a doomed comparison to -1, while Figure 5 shows a hash function that will generate a bad hash for an underline or strikeout text style.

Acknowledgments

Thanks to all the contributors to the FindBugs project. FindBugs is sponsored by Fortify Software and by SureLogic.

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] D. Hovemeyer and W. Pugh. Finding Bugs is Easy. In *Onward!, 19th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Vancouver, BC, October 2004.