Bugs in the Wild: Examining the Effectiveness of Static Analyzers at Finding Real-World Bugs

David A. Tomassi
University of California, Davis
United States
datomassi@ucdavis.edu

ABSTRACT

Static analysis is a powerful technique to find software bugs. In past years, a few static analysis tools have become available for developers to find certain kinds of bugs in their programs. However, there is no evidence on how effective the tools are in finding bugs in real-world software. In this paper, we present a preliminary study on the popular static analyzers ErrorProne and SpotBugs. Specifically, we consider 320 real Java bugs from the BugSwarm dataset, and determine which of these bugs can potentially be found by the analyzers, and how many are indeed detected. We find that 30.3% and 40.3% of the bugs are candidates for detection by ErrorProne and SpotBugs, respectively. Our evaluation shows that the analyzers are relatively easy to incorporate into the tool chain of diverse projects that use the Maven build system. However, the analyzers are not as effective detecting the bugs under study, with only one bug successfully detected by SpotBugs.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**;

KEYWORDS

bug finding tools, static analysis, BugSwarm

ACM Reference Format:
David A. Tomassi. 2018. Bugs in the Wild: Examining the Effectiveness of Static Analyzers at Finding Real-World Bugs. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3236024.3275439

1 RESEARCH PROBLEM

Static analysis tools (e.g., [1, 3, 4]) for bug finding are useful in software development. These tools analyze the program to find common patterns that lead to troublesome code. However, there is a question of their effectiveness for finding real bugs in a diverse world of software. Such information could be valuable for developers when determining which tool(s) to employ, and for the tool developers themselves when identifying opportunities to further improve their tools.

In this paper, we evaluate the static analyzers ErrorProne [1] and SpotBugs [3]. ErrorProne and SpotBugs are static analysis tools available as plugins for the Java compiler. ErrorProne applies patterns to find bugs ranging from infinite recursion to incompatible argument types in Java programs. Similarly, SpotBugs is a pattern-based tool that finds bugs that include null pointer dereferences, casting errors, and infinite loops. We study 320 real-world Java bugs from 48 distinct Java projects found in the BugSwarm dataset [7]. Our goal is to answer the following research questions: (1) how many of the bugs could potentially be found by the analyzers?, and (2) how many of these bugs are successfully detected?

The first challenge in this study is to manually examine the 320 bugs (and their respective fixes) to determine whether these bugs are good candidates for ErrorProne and SpotBugs. The second challenge is to run the analyzers on each of the program versions (320×2), to produce the corresponding bug reports. Finally, we analyze the bug reports to determine whether the analyzers successfully detect a given bug.

Our manual inspection reveals that 30.3% and 40.3% of the bugs under study fall into the categories of bugs reported to be found by ErrorProne and SpotBugs, respectively. The tools are relatively easy to incorporate into the tool chain of the projects, and successfully analyze on average 73% of the relevant buggy programs. However, the tools were unable to detect any bugs, except for one bug that is detected by SpotBugs.

2 RELATED WORK

Previous studies have compared static-analysis tools based on various metrics. Ayewah et al. [6] conducted a user-centric study of the static analyzer FindBugs in which the tool is run against the Sun's JDK implementation to determine what bugs are reported.

Rutar et al. [10] performed a comparison of Bandera, ESC/Java 2, FindBugs, JLint, and PMD on five mid-sized programs to determine the strengths and weaknesses of each tool. Their analysis included a taxonomy of detected bugs, time to analysis completion, and number of warnings and bugs found. Similarly, Wagner et al. [11] compared FindBugs, PMD, and QJ Pro on five projects, four of which are unnamed industrial projects. The authors also categorized the types of bugs found by each tool and compared their bug detection rates. Unlike the above studies, we do not compare tools based on metrics such as run-time or number of bugs reported. Instead, we examine tools independently to determine their effectiveness at finding real bugs within their scope.

Most recently, and concurrent to our work, Habib and Pradel [8] conducted an evaluation of ErrorProne, Infer, and SpotBugs on the Defects4J dataset [9]. The authors ran the tools on 594 bugs from 15 Java projects, and reported that 95.5% of the bugs could not be found by any of the tools. Our study is conducted on a different

Table 1: Characteristics of Top Three Projects

Project Repo	# Bugs	LOC	# Tests
raphw/byte-buddy	105	170,885	4,533
checkstyle/checkstyle	50	135,950	1,662
tananaev/traccar	33	59,803	237

Table 2: Manual Bug Categorization

Description	# Bugs	Description	# Bugs
Logic Error	94	Dependency Error	14
Test Error	50	Identifier Error	9
Assertion Error	38	Visibility Error	7
NullPointerException	32	Casting Error	2
Configuration Error	28	Resource Leak	1

set of bugs from 48 Java projects. Furthermore, we first examine the bugs to identify candidates for the tools under study, and then report how many of those bugs are detected.

3 TECHNICAL APPROACH

ErrorProne and SpotBugs are not designed to find *all* possible bugs. This section answers the following questions: (1) how many of a sample of real bugs could potentially be found by the analyzers?, and (2) how many of these bugs are indeed detected?

Bug Selection. We consider 320 real Java bugs from 48 distinct projects that use the Maven build system. We randomly sampled these bugs from 1,768 Java bugs found in the BugSwarm dataset. For each bug, BugSwarm provides a docker container that includes buggy and fixed versions of the source code, regression tests, and scripts to build and run the tests. Each bug has at least one failing test. Table 1 lists the three projects with the most bug samples.

Manual Bug Inspection. In conjunction with the BugSwarm developers, we carefully examined the 320 bugs under study. Table 2 shows the identified bug categories. We referred to [2] and [5] to determine which of these categories the tools can detect. We found that bugs classified as Logic Error, Casting Error, and Resource Leak are good candidates for ErrorProne and SpotBugs. Additionally, NullPointerException bugs are also handled by SpotBugs. Categories such as Test, Configuration, and Dependency Errors were found to fall outside the scope of the tools.

Bug Report Inspection. We ran ErrorProne and SpotBugs on *all* bugs in our sample, but focused on the bugs that fit within the selected categories to determine *tool effectiveness*. The tools are available as a compiler plugin, so we modified the Maven pom files. After incorporating the tools into each program's build process, the rest was significantly easier as BugSwarm provides docker images for each of the bugs with scripts to build the code. We measured effectiveness based on how many of the bugs within the criteria were detected. We ran the analyzers twice for each bug (the buggy and fixed versions). The tools produce a report for each run. Reports include the description and line number(s) of each bug detected. We examine these reports to find whether there was any reference to the diff between the buggy and fixed version of the code. In particular, we compare the reports for both versions; if a bug report were related to the actual bug, we would expect it to not be present in the report produced for the fixed version of the code.

4 PRELIMINARY RESULTS

How many bugs could potentially be detected by the tools? After our manual bug classification, and studying the bug finding capabilities of each tool, we found that 97 (30.3%) and 129 (40.3%) out of 320 bugs could potentially be detected by ErrorProne and SpotBugs, respectively.

Table 3: Results

Description	ErrorProne	SpotBugs
Bug Candidates	97	129
Bug Reports	35	60
Bug Candidates Found	0	1
Total Compilation Issues	47	88
Total Tool Crashes	12	0

How many bugs were detected by the tools? We ran the analyzers on all 320 pairs of buggy/fixed programs. We found that 59 for ErrorProne and 88 for SpotBugs did not run because of either a compilation issue or a tool crash. We then focused on examining the reports produced for the 97 and 129 pairs of buggy and fixed programs within the selected bug categories. For this subset, 27 and 33 programs resulted in either a compilation error or a crash. Table 3 shows the results. ErrorProne reported potential bugs in 35 out of 97 buggy programs. However, the reports did not include the bugs under study. SpotBugs reported potential bugs in 60 out of 129 buggy programs, but only one of the bugs matched the actual bug (a null pointer dereference). The reports for the buggy and fixed versions were identical in all but 4 and 1 instances for ErrorProne and SpotBugs, respectively. The reports differed because the report for the fixed program included a new bug introduced by the fix.

An example of a reported potential bug by ErrorProne and the line in question are in Example 4.1 and Example 4.2, respectively. The actual bug and its fix, which correspond to the incorrect use of an identifier, is given in Example 4.3.

Example 4.1. .../Transactional MethodVisibilityCheck.java:[54,51] [CollectionIncompatibleType] Argument 'Modifier.PUBLIC' should not be passed to this method; its type Modifier is not compatible with its collection's type argument ModifierTree.

Example 4.2.

```
boolean isPublic=m.modifiers().contains(Modifier.PUBLIC);
```

Example 4.3.

```
-public class UppercaseSuffixes__CheckTest {
+public class UppercaseSuffixesCheckTest {
```

5 CONCLUSIONS AND FUTURE WORK

Although we conducted this preliminary study on a large number of bugs, our results cannot be generalized. There are many avenues that can extend this study. The first is that there are many other bug-finding tools (e.g., Infer) that could be included in this study. Second, is that we could look more in-depth in why the tools did not find the bugs that are in their criteria. Lastly, BugSwarm is continuously growing, thus we expect to have access to an order of magnitude more bugs in the near future that we could use to continue this study.

ACKNOWLEDGMENTS

Special thanks to Naji Dmeiri and Yichen Wang for their contribution to the classification of BugSwarm bugs. This work was supported in part by NSF grants CNS-1629976 and CCF-1464439, and a Microsoft Azure Award.

REFERENCES

[1] 2018. Error Prone. https://github.com/google/error-prone. (2018).

[2] 2018. Error Prone Bug Patterns. https://errorprone.info/bugpatterns. (2018).

[3] 2018. Find Bugs. http://findbugs.sourceforge.net/. (2018).

[4] 2018. Infer. http://fbinfer.com/. (2018).

[5] 2018. SpotBugs Bug Descriptions. https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html. (2018).

[6] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh. 2008. Using Static Analysis to Find Bugs. *IEEE Software* 25, 5 (Sept 2008), 22–29. https://doi.org/10.1109/MS.2008.130

[7] Naji Dmeiri, David A. Tomassi, Yichen Wang, Antara Bhomick, Yen-Chuan Liu, Premkumar T. Devanbu, Bogdan Vasilescu, and Cindy Rubio-González. 2018. BugSwarm: A Large-Scale Database of Reproducible Real-World Failures and Fixes. http://www.bugswarm.org/. (2018).

[8] Andrew Habib and Michael Pradel. 2018. How many of all bugs do we find? a study of static bug detectors. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018.* 317–328. https://doi.org/10.1145/3238147.3238213

[9] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014)*. ACM, New York, NY, USA, 437–440. https://doi.org/10.1145/2610384.2628055

[10] Nick Rutar, Christian B. Almazan, and Jeffrey S. Foster. 2004. A Comparison of Bug Finding Tools for Java. In *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE '04)*. IEEE Computer Society, Washington, DC, USA, 245–256. https://doi.org/10.1109/ISSRE.2004.1

[11] Stefan Wagner, Jan Jürjens, Claudia Koller, and Peter Trischberger. 2017. Comparing Bug Finding Tools with Reviews and Tests. *CoRR* abs/1711.05019 (2017). arXiv:1711.05019 http://arxiv.org/abs/1711.05019