Analysis of Software Artifacts
Assignment #10

Mini-Project: Tool or Analysis Practicum

Team Nash Apr 7, 2009
HunJae Lee, SeongYong Lim, SeJoon Oh, Duri Kim

Contents

Overview

This document contains NASH team's practical experience with an analysis tool, FindBugs. It consists of four main parts. Firstly, we talk about overall introduction of this project. Then we explain our evaluations about this tool and its advantages and disadvantages. Finally, we introduce a plan to adapt this tool into our Studio project.

Introduction of the selected tool, FindBugs

FindBugs is a static analysis tool to look for bugs in Java code. It is free software developed by The University of Maryland. Our practice is conducted with FindBugs 1.3.8 version.

Selection Rationale

At the first time we select an analysis tool among given tool list; we want to use the tool that supports various complicated functions like Purify. However this kind of commercial tool's trial version does not provide us restricted use.

So, we established a couple of criteria to select tool. First, it should be a freeware to support its full use. Second, it should be active tool that authors or tool developers still elaborate to fix its bug even recently because we don't want to use buggy tool. At last, we want to make this tool to help our Studio in terms of quality code. So, it should support Java language and easy installation and usage.

Objective

The main purpose of this project is to see how well the tool can find bugs and to access how meaningful the bugs are. Through these activities, we can acquire its efficiency as an analysis tool compared to our former experience of other tools. We can get its adaptability of FindBugs when using in our Studio as well.

In addition, it is good opportunity to evaluate the open source software used in our Studio project. The results of this practicum cannot be crucial factor to evaluate open source software, but it could be auxiliary data to select COTs in our Studio project.

Feature

Our team chose some of main functions provided by FindBugs. When choosing target functions to evaluate, we consider the usefulness of functions and how convenient usage this tool gives.

We decided to access the function to find all kinds of the bugs defined in the tool. However, we did not evaluate the extension of bug finding filters and historical analysis.

Statistics of individual efforts

HunJae Lee	9 hours
SeongYong Lim	9 hours
SeJoon Oh	9 hours
Duri Kim	11 hours

Evaluation

This part contains our team's procedure to conduct for this practicum and its results. This part comprises the introduction of our evaluation criteria, how we conduct for this evaluation, the results, and our analysis about this tool and supporting examples.

Evaluation Criteria

Quantitative evaluation criteria

- Ratio of true positives and false positives
- Efficiency of the tool
- True positives per KLOC
- Code quality of the open source software
- Total bugs found per KLOC

Qualitative evaluation criteria

- Usability – easy to install and run
- Readability –precise description of bugs
- Accuracy – real bugs or not

Evaluation Approach

Target functionality of our evaluation

FindBugs features evaluated

- The FindBugs provides several options about pattern filters. But, we were not sure which pattern or pattern type is true positives or not. So, we decided to find all kinds of the bugs defined in the tool and evaluate the result in our perspective

FindBugs features not evaluated

- The FindBugs also provides its extensibility through importing and exporting filters which are described by users. But, because of lack of time, we forgive to try it even though this extensibility seems very useful.
- The organization which is developing the FindBugs is gathering the historical data from its users and shows some statistical results. But, we didn't analyze with that results and didn't provide our result.

Customization

There are options to select which kinds of bugs or issues you are going to detect, and you can also add filters to check certain kinds of bugs or issues by defining them and attaching them into the set of filters in the tool. For the evaluation of the tool, we have decided to detect all the kinds of bugs to see how well the tool finds bugs or issues and what kinds of them it is able to find; to do so, we checked the options of all the kinds of bugs. For the future usage, we would be able to use the checking options to find specific bugs or issues and the extensible filters to add a new kind of bugs or issues.

Environment

The FindBugs supports three types of its deployment types. To check whether there is any difference between them, we applied two types, eclipse plugin and standalone GUI of the FindBugs.

Resource

- Actually, we applied out previous results of the assignments. But, it is easy to imagine that the quality of our codes is not good and the FindBugs can produce lots of bugs.
- So, we need to focus on evaluating the FindBugs. First of all, we use sample program with seeded defects to find how well it finds the bugs which are seeded by us. For that purpose, we used Hnefatafl source with 5 seeded defects in the previous assignment.
- Also, our Studio project is supposed to use several the result of Open Source project. So, through applying this tool to some Open Sources, we want to check the qualities. Those are Shindig, STRUTS, iBATIS, and Tomcat.

Evaluation Result

Evaluation of Seeded Defects

We conducted a test for Hnefatafl framework, especially defect seeded version. The FindBugs cannot find any seeded bugs among 5 of them.

Evaluation of Open Source software

Source	**Size**		**TRUE Positive (ratio)**	**FALSE Positive (ratio)**	**T.P./ KLOC**	**Bugs/ KLOC**
	Files	**LOC**				
Shindig	922	28,682	68%	32%	1.185	1.848
STRUTS	1,044	62,991	52%	48%	1.270	16.939
iBATIS	205	14,228	64%	36%	3.584	5.623
Tomcat	1,115	170,785	1063			6.224

After executing the tool, we checked all bugs and bug types whether each bugs should be generally considered or not to determine true positives. But, the number of bugs in Tomcat is too big to investigate.

Analysis of evaluation results

Evaluation Criteria	**Analysis result**
Efficiency	0 among 5 → Cannot find semantic bugs T.P. about 60% vs. F.P. about 40% → 60% of bugs are to be considered
Usability	2 clicks to run One update site for Eclipse plug in Exe file to install the standalone
Readability	Bug description with rationale, possible cases, and examples
Accuracy	Not bugs, but recommendations

- As the result, the FindBugs cannot any semantic bugs which are not detected by mechanical pattern matching method.

- In our perspective, we should consider about 60 percent of the bugs found by the FindBugs as true positives.
- Users can easily install via the eclipse update site or execute with only 2clicks.
- Each bug is described with its rationale, possible cases and examples.
- But, the FindBugs just provides not bugs, but possible bug patterns.

Example bugs from FindBugs

Examples of Relevant True Positives

1. At the catch function with an exception, there is a condition that no exception is caught when exception is not thrown.

```
try {
        String prop_pool_ping_query = null;
        if (props == null) {
                throw new RuntimeException("SimpleDataSource: The properties map passed
                        to the initializer was null.");
} catch (Exception e) {
        log.error("SimpleDataSource: Error while loading properties. Cause: " + e.toString(), e);
        throw new RuntimeException("SimpleDataSource: Error while loading properties.
                Cause: " + e, e);
}
```

2. A result set, rs only start with index 1. But the counter of the loop, i, starts from 0 and it means that it tries to access a result set field with index 0.

```
while (rs.next()) {
        for (int i = 0; i < cols; i++) {
                String value = rs.getString(i);
                print(value + "\t");
        }
        println("");
}
```

Examples of Irrelevant True Positives

The main functionality of FindBugs is to detect bugs or issues in Java program sources, in general. The bugs or issues include not only possible bugs, but also best practices in general Java programming. Thus we were not able to find any reason to decide the issues or recommendations from FindBugs are not relevant to our studio project. It is because we need to detect all kind of bugs or issues in our Java codes to improve the quality of the codes; that is, all the true positives are relevant to our studio project.

Examples of False Positives

1. The FindBugs warns a Cloneable class which does not define or use clone method. But, the class has a protected clone() method.

```
public class Template implements Cloneable {
                ...
        protected Object clone() throws CloneNotSupportedException {
                return super.clone();
        }
}
```

2. The FindBugs warns the class, TestCase, defines a function, setup(), that doesn't call super.setUp(). But, the extended class, PropertyTest, calls super.setup().

```
public class PropertyTest extends StrutsTestCase {
        private XWorkConverter converter;
        public void setUp() throws Exception {
                super.setUp();
                "PropertyTest" uses StrutsTestCase's super setup()
                …
        }
}
```

Pros and Cons of FindBugs

As a result of the evaluation, we were able to identify various pros and cons of the FindBugs. Follows are the summary of the pros and cons. However, since we did not evaluate the full functionalities of the FindBugs, these pros and cons are limited to the boundary of our evaluation. For example, data mining features of the FindBugs was not tested because we did not have historical data for that. Thus most our evaluation was focused on the bug detecting functions.

Pros

- FindBugs well detected the bugs (or possibly bugs) and it provides clear rationale for each bugs.
- FindBugs looks good to improve code quality because it detected not only bugs, but also bad programming practices.
- The extensible filter plug-in framework of the FindBugs deserves a mention. While it has a lot of filters corresponding to bug types, it provides a way for developers to extend its capability by adding any needed filters.

Cons

- FindBugs has detected only syntactic issues, but not semantic errors
  For example, when we conducted a test against our bug seeded code, it did not detect any seeded bug. The reason was that all the seeded bugs were semantic bugs, not technical bugs.
- While FindBugs have detected many possible bugs, some of the bugs were not real bugs. Some issued bugs were related to programming practice or coding standard. In other word, if a development decide to use another set of coding standard, the result of the tool would not be sufficiently reasonable.
- Even after getting result from the tool, we had to investigate if the issued bugs were real or not. It was because we could not get substantial evidence to believe that all the issues would be real. The nature of the development of filters added such kind of suspect. Each filter corresponding to each bug could be developed by individual developers. This meant that all filters could not have a level of standard quality. Therefore the issued bugs could be only issues; we had to judge the reality of issues.

With Studio Project

After evaluation, we decided to use FindBugs in our studio project for following purposes.

- A part of evaluation of Open Source Software – we are about to use dozen of open source software. However we did not have an effective mean to verify its quality – code quality. This tool enabled us to examine its code quality in a very comprehensive way. Even though this tool could not perfectly prove the quality of the codes, we think it provide a reasonable clue to determine how the quality is.
- A tool for code review – if we start implementation, we are about to conduct code reviews for our codes. FindBugs will be really helpful tool for us to be able to perform code reviews effectively. Of course, this tool will not be able to totally replace our code review activities. Even after running this tool, we will have to do code reviews to determine if the issued bugs are real bugs. However, if we use this tool as a part of activities of our code review, it would certainly help us to raise the quality of our code review; for example, it could find possible bugs which we could miss in our manual reviews.