Report of Project
Cassiopée N°92

Software testing tools: comparing existing solutions

Authors:

DONG Yifang

WANG Yuwei

ZHENG Qi

Advisor:

KUSHIK Natalia

Contents

I. Introduction

Nowadays, software testing is more and more necessary, because "humans make mistakes all the time". Some mistakes may be unimportant, but some of them are expensive or dangerous. It is very essential to ensure the quality of the product.

Therefore, for our project Cassiopée, we chose the subject "Software testing tools: comparing existing solutions" to study in the domain of software testing.

This project is devoted to the study of existing methods and tools for effective software testing. Our missions are comparing different testing techniques considering various criteria and giving some improving solutions when an existing tool is not enough perfect.

In this report, we will introduce the detailed work we have done for our research of the software testing, especially of the static analysis. And how we arranged our project with our advisor. We will also present the final results of our research.

The structure of this report is as follows. We start with a brief introduction. And then section 2 contains preliminaries. Section 3 contains state of the art. Section 4 contains objectives and tasks. Section 5 contains the results of research. At last, there is the conclusion and references.

II. Preliminaries

i. Software testing concept

Software testing is a process used to identify the correctness, completeness and quality of developed computer software [1]. It includes a set of activities conducted with the intent of finding errors in software so that it could be corrected before the product is released to the users. In simple words, software testing is an activity to check that the software system is defect free. Software testing is important because that software bugs can be expensive or even dangerous.

ii. Classification of software testing

There are many methods in software testing. Those methods are traditionally divided into "white box" and "black box" testing assumptions. In "white box" assumption we know everything about the system under test including the code; in the "black box" we just have access to the interfaces. "White box" assumption is usually done at the unit level. However, "black box" typically comprises most testing at higher levels like integration and system.

Software testing methods are also divided into static testing and dynamic testing. Under static testing, code is reviewed, walked through or inspected rather than executed, while under dynamic testing, programmed code with some given test cases is executed.

Classified by testing levels, there are generally considered unit testing, integration testing, and system testing. Unit testing refers to tests by developers that verify the functionality of a specific section of code, usually at the function level. Integration testing is to verify the interfaces between components against a software design. System testing tests a completely integrated system to verify that the system meets its requirements.

As for testing software performance, it contains load testing, volume testing and stress testing. Those three functions put software into different conditions and get its performance by performance parameters. In performance testing, we [1] often use QTP, WinRunner, Selenium, Watir etc.

iii. Static analysis & "white box testing" assumption

After we have come to know the overview of software testing, we decided to study the different static code analysis tools. We will focus on java language. In other words, we use "white box" assumption testing method for our project.

Static testing is a software testing method that involves examination of the program's code and its associated documentation but does not require the program be executed. Hence, the name "static". Main objective of static testing is to improve the quality of software products by finding errors in early stages of the development cycle. Specific types of static software testing include code analysis, inspection, code reviews and walkthroughs.

"White box" assumption is also known as clear box testing, glass box testing, transparent box testing and structural testing. It is used to test internal structures or workings of a program by seeing the source code.

III. State of the art

i. Static software testing tools: existing solutions

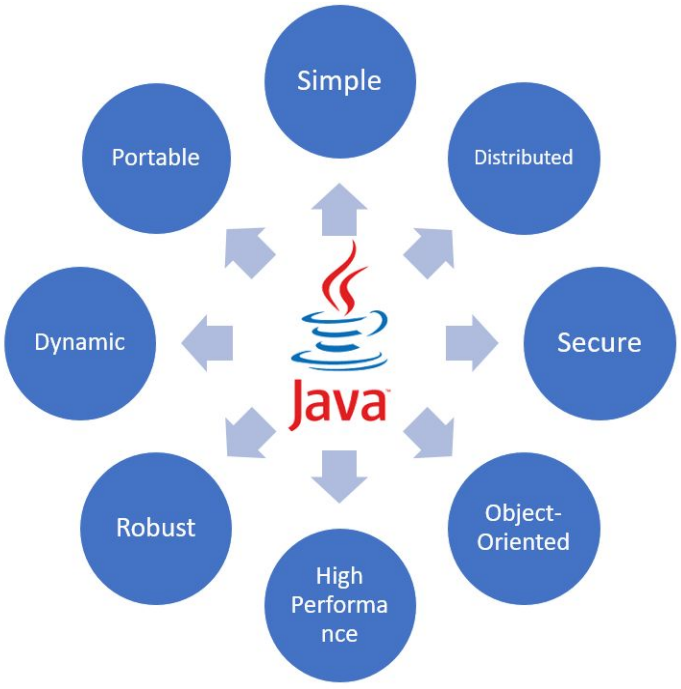
Static software testing is a testing technique to test code without executing it. There are two types of software testing [2]:

Code review: used to find errors or ambiguities in documents, like class design, test cases, specification, etc.

Static analysis: use tools to analyze code to find defects that may cause problem when running the program.

By static analysis, a lot of types of defect can be detected, such as unused code block, out of bounds, a variable with an undefined value and so on. To carry out static software testing, there exists many tools for each programming language to simplify the procedures. Some of them are open source, but some of them are not free.

Java is the most widely used programming language. As we can see in the following image, Java has lots of positive factors:



Advantages of JAVA

We can use Java in almost every domain. In banking, Java can be used to deal with transaction management; in scientific and research community, a framework written by Java called Hadoop is mainly used to deal with big data.

So it is necessary and important to test Java code. We found that plenty of tools can analysis statically Java code, such as FindBugs, PMD, Checkstyle, Lint4J, Classycle, JDepend, SISSy, Google Codepro, etc.

Among those tools, as PMD, FindBugs and Checkstyle are open source tools and have excellent performance when it is integrated into development process. Therefore, in our work, we decided to compare PMD, FindBugs and Checkstyle.

ii. Introduction of PMD, FindBugs and Checkstyle

PMD is a static Java source code analyzer. It uses a set of rules to define when a piece of source is erroneous. Typically, issues reported by PMD aren't true errors, but rather inefficient code [3].

FindBugs is an open source static code analyzer which detects possible bugs in Java programs. FindBugs operates on Java bytecode rather than source code [4].

Checkstyle is a static code analysis tool used in software development for checking if Java source code complies with coding rules [5].

iii. Existing works on tools' comparison

Because FindBugs, PMD and Checkstyle are widely used in Java code static testing, a huge number of comparison are carried out among those three tools. Most researchers [6, 7, 8] find that there is a good amount of overlap between FindBugs and PMD, despite that they use different principles, FindBugs works on byte code, whereas PMD works on source code. Although Checkstyle is a tool for analyzing coding style and conventions, it also has some overlap with FindBugs and PMD such as empty blocks and equals() vs "==".

What is interesting is that although these three have overlap, programmers usually use them at the same time because of unique service provided by each tool. They are not interchangeable. There is few research compare these three tools by classifying services and find their advantages and weakness. So we decided to compare them by studying concrete problems detected by these three tools and find the different performances of these three tools for each type of problems. We also want to find problems cannot or hardly detected then, search for some methods to resolve them.

IV. Objectives & Tasks

The first objective of this project is to study different static code analyzers developed for Java language, namely PMD, FindBugs, and Checkstyle.

The second objective is to list bugs that these three tools can detect and compare the result of their capacities.

The third objective is to propose solutions to find bugs hardly detected by these three tools.

In order to achieve these objectives, we divide project as five main tasks:

1. Doing background study of software testing.

2. Defining the criteria to compare, creating the "interesting" codes to be tested and configuring the environment for the tools.
3. Comparing these three tools based on their documentation and the results of analysis of the benchmark code.
4. Looking for limitations of static code analysis tools, and thinking of how to solve them.
5. Studying other techniques or proposing novel solutions for undetected bugs.

V. Results of research: Comparing static code analysis tools

i. Comparison of three tools: PMD, FindBugs and Checkstyle

The static code analysis tools FindBugs, PMD and Checkstyle are widely used in the Java development activities. Each of them has their own purpose, strength and weaknesses. At the same time, these three tools also have some aspects and rules in common.

We have read each document and classified the tested problems as different categories. And then we check the tool respectively by the tested codes for each category if this tool is able to detect or not. Here is the summary table:

N°	Problems	PMD	FindBugs	Checkstyle
1	Naming Conventions	√	√	√
2	Equals() Problems	√	√	√
3	Class Design	√	√	√
4	Strict Exceptions	√	√	√
5	String	√	√	√
6	Clone Implementation	√	√	√
7	Modifiers	√	√	√
8	Incorrect Usage of Junit	√	√	√
9	Comments	√	-	√
10	Empty Code	√	-	√
11	Braces	√	-	√
12	Import statements	√	-	√
13	Size Violations	√	-	√
14	Coupling	√	-	√
15	Code Complexity	√	-	√
16	Unused & Unnecessary Code	√	√	-
17	Finalizer	√	√	-
18	Performance optimization	√	√	-
19	J2EE	√	√	-
20	Java Logging	√	-	-
21	Incorrect Usage of SQL	-	√	-
22	Override	-	√	-
23	Multithreaded correctness	-	√	-
24	Range(out of bounds)	-	√	-
25	White Space	-	-	√
26	Annotations	-	-	√

Comparison of three tools

As we can see in the table, we concluded 26 categories of problems. Obviously, some of them can be detected by all those three tools, but some of them can be detected by only one or two of them.

ii. limitations of PMD, FindBugs and Checkstyle

Before we research the limitations of these three tools, we firstly divided the bugs into four levels according to their harmfulness: Critical, High, Medium and Low.

After comparing the results, we found that there are some problems which are hard to be detected and which can never be detected. We defined that those only one tool can detect are the problems hard to be detected. And those no tool can detect are the problems never to be detected.

In conclusion, for the problems hard to be detected, level "low" includes the categories "White space" and "Java Logging"; level "Medium" includes "Annotations", "JDBC" and "Multithreaded correctness"; level "High" includes "Override"; level "Critical" includes "Array index (constant) out of bounds". For the problems never to be detected, we proposed some usual problems. One for level "Low" is "One line has only one variable", the others for level "Critical" are "Overflow", "Division by 0", "Implicate out of bounds" and "Parameters don't match". The table below gives the concrete examples of each problems.

A	B	C	D	E	F
	Problems Not Detected	Example			

1	Implicitly out of bounds	int[] array = new int[10];
		for(int i = -1; i <= array.length; i++) {
		System.out.println(array[i]);
		}
		int i = 11;
		System.out.println(array[i]);			

2	Division by 0	int k=2/0;			

3	Parameters don't match	public int sum(int a, int b){
		return a+b;
		}
		sum(1);
			This problem can be detected by IDE & compiler		

4	Overflow	public static int overflow(){
		return Integer.MAX_VALUE+Integer.MAX_VALUE;
		}			

5	One line has only one variable	int i,j,k =0			

Example of problems never detected

Besides, the problems detected that is indicated in the document is not always correct. We found an example in the document of FindBugs as below:

CI: Class is final but declares protected field (CI_CONFUSED_INHERITANCE)

This class is declared to be final, but declares fields to be protected. Since the class is final, it cannot be derived from, and the use of protected is confusing. The access modifier for the field should be changed to private or public to represent the true use for the field.

We use following code to test. However, FindBugs does not give a report about this problem

```
public final class CI_CONFUSED_INHERITANCE {
        private int x;
        protected int y;
        CI_CONFUSED_INHERITANCE() {
                x = 1;
                y = 2;
        }
}
```

Example code of CI_CONFUSED_INHERITANCE

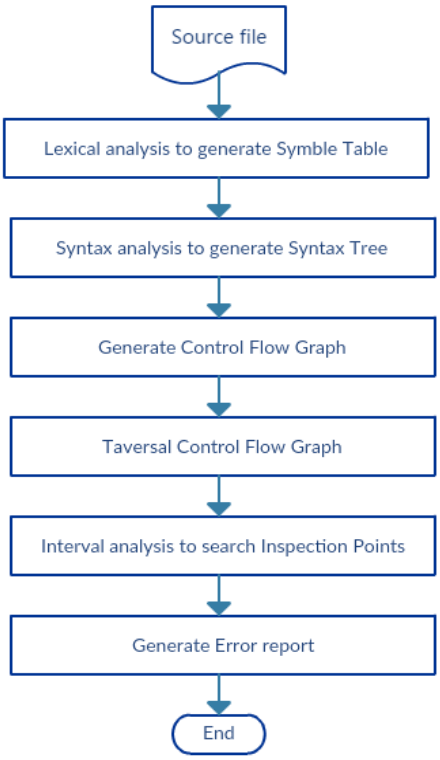
iii. How to deal with undetected bugs?

Among these limitations of these three tools, we chose one of the problems to study furtherly to see how to deal with it.

Array index out of bound is a common failure of buffer overflow failures. It is a serious problem which can cause a system crash. Among these three tools, only FindBugs can detect array index out of bound with constant index rather than various index.

Array index out of bound is usually divided into overflow and underflow. For example, declare an array whose size is *len*, then its subscript range is *R = [0, len-1]*. If the array element is referenced by j as an array index, an underflow occurs when j <0, and an overflow occurs when j> len-1.

The solution was discussed in the article [9]:



The steps to detect Array index out of bound

Lexical analysis: divide the source code into separate words and then form the variety of symbol table, type table, keyword table, constant table, table and decomposition table. The symbol table contains some information that is useful for error finding, such as location, line number, and error type, which are important for both error search and positioning.

Syntax analysis: This step is mainly to identify the input string as a word stream, here is to find out the variable declaration, and separate the various types of variables. Follow the standard Java grammar rules to further analyze the source files, such as: what is the variable definition, what is the assignment statement, what is the function and so on. Finally, generation of the syntax tree.

Generate control flow graphs: Based on the syntax tree, the control flow graphs are generated by traversing the syntax tree with the recursive algorithm according to the control structure of the program. Each node in the control flow graph represents a statement of the program that corresponds to the node on the grammar tree.

Find out of bound problem: The breadth-first traversal is performed from the entry of the program control flow graph. In the process of traversing, the integer interval is constructed according to the array subscript expression integer interval set. If the interval is not included in the array interval, there is an array of transboundary faults.

Conclusion

This report has given the comparison of three tools PMD, FindBugs and Checkstyle in static software testing. From our study, we got some good practice advice as follows:

1. When we use PMD, FindBugs and Checkstyle to do testing, be careful with problems such as "Override" and "Array index (constant) out of bounds", because they are dangerous and are hard to be detected by these tools.
2. "Overflow", "Division by 0", "Implicate out of bounds" and "Parameters don't match" are also the dangerous problems that cannot be detected by these tools, we need to find another technique to detect them.

The study has enhanced our understanding of static code analysis and carry out "white box" assumption testing. Especially the capabilities and limitations of PMD, FindBugs and Checkstyle.

Further research might compare more static software testing tools such as JArchitec, Lint4J, JDepend, Google Codepro, etc.

References

[1] Software Testing is a Process Used to Identify the Correctness https://zh.scribd.com/document/44522963/Software-Testing-is-a-Process-Used-to-Identify-the-Correctness

[2] static testing https://www.tutorialspoint.com/software_testing_dictionary/static_testing.htm

[3] PMD https://pmd.github.io/

[4] FindBugs http://findbugs.sourceforge.net/

[5] Checkstyle http://checkstyle.sourceforge.net/

[6] Static Code Analyzers - Checkstyle, PMD, FindBugs - Are those alternatives or not? - http://tirthalpatel.blogspot.fr/2014/01/static-code-analyzers-checkstyle-pmd-findbugs.html

[7] Checkstyle vs PMD vs Findbugs http://continuousdev.com/2015/08/checkstyle-vs-pmd-vs-findbugs/

[8] Comparison of Static Code Analysis Tools for Java - Findbugs vs PMD vs Checkstyle http://www.sw-engineering-candies.com/blog-1/comparison-of-findbugs-pmd-and-checkstyle

[9] ZHAO Peng-yu, LI Jian-ru, GONG Yun-zhan. Research on static test about array index out of range in java language. Computer Engineering and Applications,2008,44(27):87-90