Analysis of Software Artifacts
Mini Project Report

Apr. 24th 2007

Team Members:

Eunjeong Choi
Wooseok Choi
Minhyuk Oh
Taeho Kim
Jihyun Lee

1. Overview

1.1 *Tool Name*

FindBugs Ver1.2.0-rc5 (Standalone and Eclipse plug-in)

1.2 Team Member

Name	*Contact*	*Position*
Wooseok Choi	wchoi@andrew.cmu.edu	Team Leader
Taeho Kim	taehokim@andrew.cmu.edu	Tool Study
Eunjeong Choi	eunjeong@andrew.cmu.edu	Bug Analyzer
Jihyun Lee	jihyunl@andrew.cmu.edu	Bug Investigator
Minhyuk Oh	minhyuko@andrew.cmu.edu	Presenter

1.3 Purpose of the project

The purpose of the project is to evaluate FindBugs tool by comparing between the bugs that are found by unit test, automated test, and code review and the bugs that are found by FindBugs in order to see the effectiveness of FIndBugs tool, and evaluate the tool that how many false positives are found and see that the found bugs are valuable to fix in terms of schedule pressure and cost of fixing bugs.

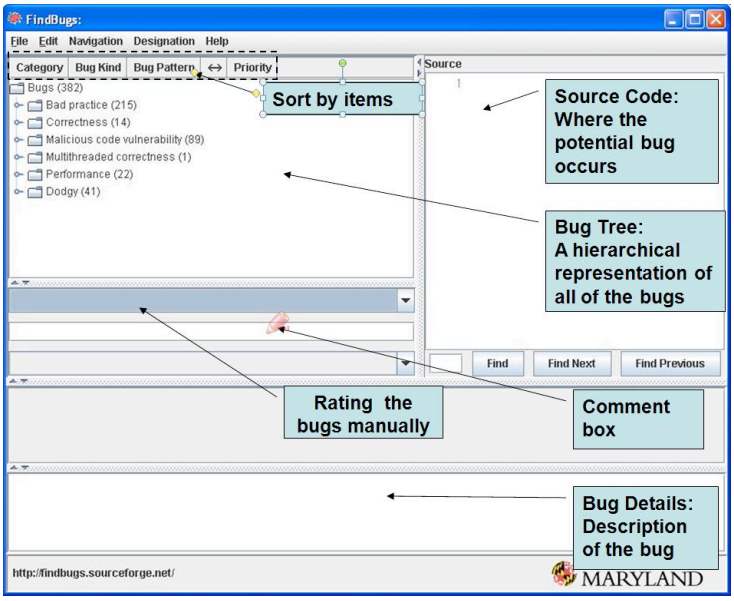
2. Tool Description

2.1 What is FindBugs?

A static analysis tool that examines java class or JAR files looking for potential problems by matching java byte-codes against a list of bug patterns. With static analysis tools, we can analyze software without actually running the program. Instead the form or structure of the class files is analyzed to determine the program's intent, often using the Visitor pattern.

2.2 Tool Usage

We included the screenshot of FindBugs's main screen with the explanation of what each field or button does. "Category, Bug Kind, Bug Pattern" are represents how the found bugs will be sorted, and right under the sort option, the field shows the list of bugs that are found from the application. In addition, FindBugs allow us to decide the importance of bugs manually such as "Mostly harmless, not a bug, Should fix, Must fix, etc", so that we can compare our decision on importance of bugs and FindBugs's decision. The bottom box shows the detail information on bugs. It gives us the suggestion of what to do and the description of why it happens. Source box is activated whenever a bug is selected from Bug Tree. If the bug is selected, the source code is shown and indicates where the bug occurs specifically.



<Figure 2.2.1. Usage of FindBugs>

FingBugs provides several options: filter files, data mining, annotations and so on. Rather than investigating all the options, we chose to test an annotation option, because it provides useful functionality based on Java 5 annotation feature. FindBugs has 13 types of annotations, and most of annotations are related to null checking. To use this feature, the annotations.jar file must be placed on the classpath while compiling a program.

In this tool evaluation, we would like to introduce four kinds of annotations according to <http://findbugs.sourceforge.net/manual/annotations.html>.

Annotation type	Brief description	Example

@CheckReturnValue	It is used to denote a method whose return value should always be checked after invoking the method.	@CheckReturnValue
		public Object dummyTest()
		{
		return null;
		}

@NonNul	The annotated element must not be null. Annotated methods must have non-null return values.	@NonNull String m_str = new String("test");
@Nullable	The annotated element could be null under some circumstances.	@Nullable Vector<String> m_vec = **null**;

@DefaultAnnotation	Indicates that all members of the class or package should be annotated with the default value of the supplied annotation classes.	@DefaultAnnotation(NonNull.**class**)
		public class ThreadTest **extends** Thread {
		...
		}

```
@DefaultAnnotation(NonNull.class)
public class ThreadTest extends Thread {
```

<Figure 2.2.2 An example of class with annotation>

Figure 2.2.2 shows that ThreadTest class has DefaultAnnotation with NonNull property, which means every filed and methods in this class will have a non-null annotation as a default property.

```
    @CheckReturnValue
    public Object dummyTest()
    {
        return null;
    }

    public void testAnnotations()
    {
        Object obj = dummyTest();
        if(obj != null)
            System.out.println(obj);
        dummyTest();
    }
```

<Figure 2.2.3 An example of methods with annotation>

However, as seen by figure 2.2.3, dummyTest() method returns null value, so FindBugs marks the line of 'return null;' with a red tiny bug picture. Also, dummyTest() method has an annotation of @CheckReturnValue, but testAnnotations method does not check the return value while calling dummyTest() method. So FindBugs marks the line of 'dummyTest();' in testAnnotations() method.

This feature provides useful functionality which supports designer's intention, but FindBugs still have very limited kinds of annotations, comparing to SAL annotation or SPEC#; most of annotations of FindBugs are related to null checking issues. So, to fully utilize this feature, we may need to wait for more later version of FindBugs.

2.3 Installation

2.3.1 Standalone FindBugs

Step 1) Access to (http://findbugs.sourceforge.net/downloads.html)
Step 2) Download one of the zip file
Step 3) Unzip the downloaded file in the designated folder

2.3.2 Eclipse Plug-in

Step 1) In Eclipse, click on **Help** -> **Software Update** -> Find and Install...
Step 2) Choose the **Search for new features to install** option, and click **Next**
Step 3) Click **New Remote Site**
Step 4) Enter the following:
Step 5) **Name:** FindBugs update site

- **URL:** one of the following (note: no final slash on the URL)

- http://findbugs.cs.umd.edu/eclipse for official releases
- http://findbugs.cs.umd.edu/eclipse-candidate for candidate releases and official releases
- http://findbugs.cs.umd.edu/eclipse-daily for all releases, including developmental ones

And click **OK**.

Step 6) "FindBugs update site" should appear under **Sites to include in search**. Click the checkbox next to it to select it, and click **Finish**.

Step 7) You should see **FindBugs Feature** under **Select features to install**. (You may have to click on one or two triangles to make it visible in the tree.) Select the checkbox next to it and click next

Step 8) Select the **I accept** option to accept the license and click **Next**

Step 9) Make sure the location is correct where you're installing it. The default (your workspace) should be fine. Click **Finish**.

Step 10) The plug-in is not digitally signed. Go ahead and install it anyway. Click **Yes** to make Eclipse restart itself.

2.4 How to use

2.4.1 Standalone

Step 1) Go to the installed directory

Step 2) Change the folder to "**bin"** directory

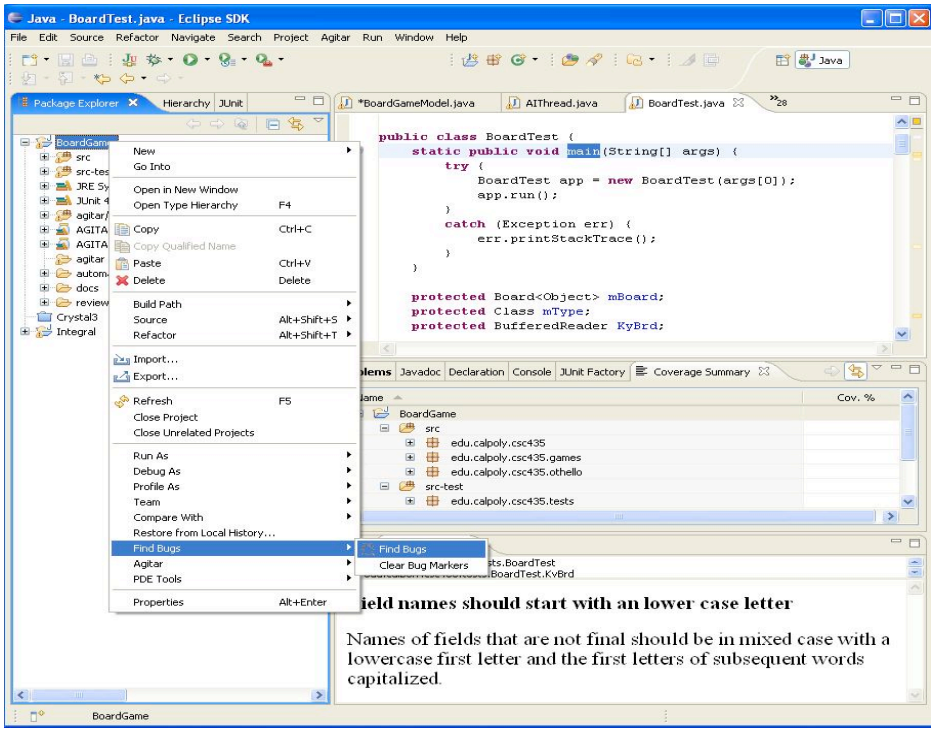
Step 3) Double click on "**findbugs.bat**"

2.4.2 Eclipse Plug -in

Step 1) Select the project

Step 2) Click left button on the selected project

Step 3) Select the "**Find Bugs**" -> "**Find Bugs**" as shown below



<Figure 2.4.1. Eclipse Plug-in>

2.4.3 The difference between standalone and plug-in

- Standalone: We can categorize bugs as a bug pattern, priority and so on. This feature can improve visibility of bug list.
- Eclipse plug-in: We can quickly reflect feedback from FindBugs to the source code, because we can see the bug list in Eclipse's problem window. The bugs in the problem window are directly linked to the source code.

2.5 Bug Categories

This table shows the bug categories that FindBugs can find. Each category contains a set of bug patterns.

Category	Description	Example
Bad practice	Violations of recommended and essential coding practice. Examples include hash code and equals problems, cloneable idioms, dropped exceptions, serializable problems, and misuses of finalize. (More false positives bugs)	Clone method may return null
Correctness	Probable bug - an apparent coding mistake resulting in code that was probably not what the developer intended	Call to equals() with null argument
Internationalization	Misuse of the platform's default encoding	Consider using Locale parameterized version of invoked method
Malicious code vulnerability	Variables, fields or methods that are used by unauthorized classes or packages	Finalizer should be protected, not public
Multithreaded correctness	Issues that related to the thread synchronization such an inconsistent synchronization and unconditional wait	Field not guarded against concurrent access
Performance	Code that is using inefficient memory, or degrade computation	Method invokes inefficient new String (String) constructor
Dodgy	Code that is confusing, anomalous, or written in a way that leads itself to errors. Examples include dead local stores, switch fall through, unconfirmed casts, and redundant null check of value known to be null. (More false positives bugs)	Redundant comparison of non-null value to null

3. Tool Evaluation

The tests performed with JDK/JRE 5.0 version in Windows XP Operating System.

3.1 Evaluation with Standalone Application

3.1.1 Purpose

The purpose of this analysis is to find out that what kinds of bugs FindBugs can find in the Othello application. We evaluate FindBugs tool that how many false

positive bugs it generates. In addition, we found the bugs for Othello application manually by using the unit, code review, manual, and automated testing. We hope to compare the bugs that we found against the bugs that FindBugs found for Othello application. In section 3.1, we only evaluate the bugs that FindBugs found, and the comparison between them will be discussed in section 3.2.

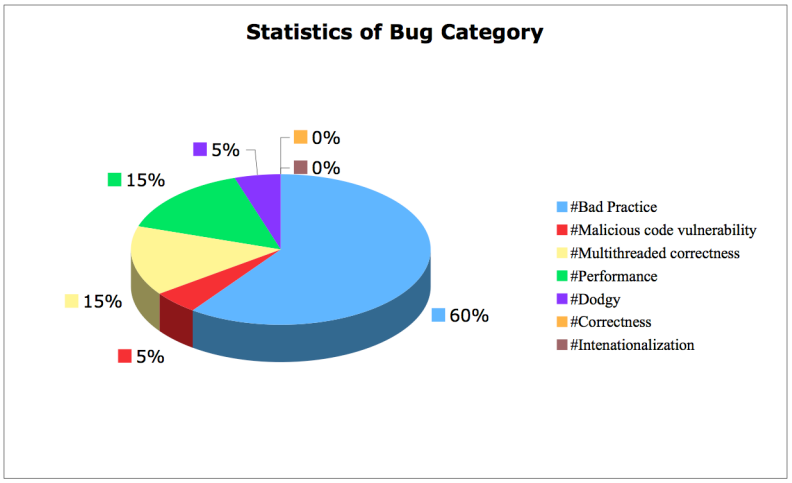
3.1.2 Artifacts

We ran FindBugs on Othello Application. The application was given in the Analysis of Software Artifact class as the second assignment. The application has 41 classes including the source files and test files, and the total size of the classes is approximately 3877 lines of code.

3.1.3 Bug Category

FindBugs find the 7 categories of bugs: Bad practice, Malicious code vulnerability, Multithreaded correctness, performance, and dodgy. While we evaluated FindBugs tool with Othello application, the tool found the bugs that are categorized below.

	Name of Category	#Bugs	Percentage
Bug Category	#Bad practice	12	60.0%
	#Malicious code vulnerability	1	5.0%
	#Multithreaded correctness	3	15.0%
	#Performance	3	15.0%
	#Dodgy	1	5.0%
	#Correctness	0	0.0%
	#Internationalization	0	0.0%
	#Total	20	100.0%



<Figure 3.1.1 Statistics of Bug Category>

3.1.4 Analysis of Bug Patterns

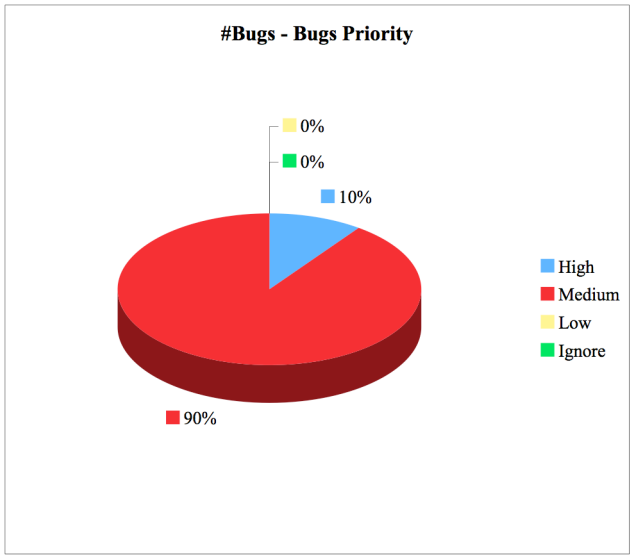
After we ran FindBugs tool on the Othello Application, we found 20 bugs. For found bugs, we spent time to find out why FindBugs considers this as the bugs. After we investigate the bugs, we rate the priority as our point of view in order to see that what bugs that FindBugs considers as high priority compared to what we consider as high priority. Lastly, we fixed the bugs based on the comment that was given from FindBugs, and measured the fixing time in order to see the fixing cost. The below table shows the summary of what we analyzed from the result that FindBugs analyzed.

The bug patterns and priority are gathered from FindBugs result, and we measured analyzed priority, investigation time and fixing time. The analyzed priority is decided based on our discussion. We followed the categories that FindBugs tool provided: Must fix, should fix, mostly harmless, not a bug and needs further study. Must fix is assigned if and only if the bug causes the system failure. Should fix is assigned if and only if the bug can potentially make the system perform incorrect behavior. Mostly harmless is assigned if and only if the bug affects the system, but not causing the system failure. Not a bug is assigned if and only if the bug does not affect the system at all. Last priority type is "needs further study", which requires more investigation time to analyze how much it can affect to the system. When comparing investigation time with fixing time, investigation time took longer time than we originally expected, because some of bug types are not familiar with us. Until being familiar with some of bugs that FindBugs found, we went through learning curve. As seen by figure 3.1.2, even if we investigated the same bug patterns, 'Class defines compareTo() and uses Object equals()', time to investigate them was different. First investigation of the same bug pattern took more time than next ones. However, fixing time took less than investigation time, because FindBugs gave good information about where bug occurred and what problem is. Also, the bugs are not so complex to fix.

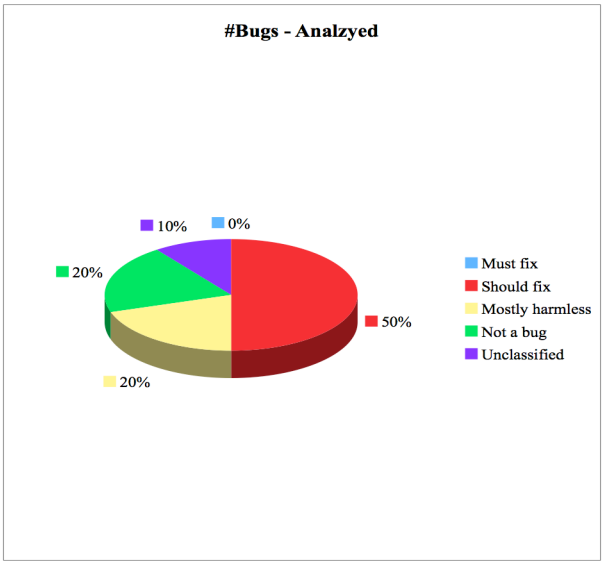
Bug Pattern	Priority	Analyzed Priority	Investig. Time (min)	Fix. Time (min)
Class defines compareTo(…) and uses Object equals()	Medium	should fix	20.00	5.00
Class defines compareTo(…) and uses Object equals()	Medium	should fix	4.20	2.00
Class names should start with an upper case letter	Medium	Not a bug	0.08	0.00
Class names should start with an upper case letter	Medium	Not a bug	0.02	0.00
Field names should start with an lower case letter	Medium	Not a bug	0.43	0.00
Non-transient non-serializable instance field in serialzable calss	Medium	should fix	5.15	14.00
Non-transient non-serializable instance field in serialzable calss	Medium	should fix	0.03	1.00
Non-transient non-serializable instance field in serialzable	Medium	should fix	3.00	6.00

calss				
Non-Serializable value stored into instance field of a serializable class	High	should fix	2.20	1.00
Class is Serializable, but doesn't define serialVersionUID	Medium	should fix	1.78	5.00
Usage of GetResource may be unsafe if class is extended	Medium	needs further study	5.50	0.00
Usage of GetResource may be unsafe if class is extended	Medium	needs further study	0.25	0.00
Field isn't final but should be	High	mostly harmless	0.75	0.62
Inconsistent synchronization	Medium	should fix	4.85	1.00
Inconsistent synchronization	Medium	should fix	0.13	3.00
Inconsistent synchronization	Medium	should fix	0.17	0.17
Method invokes inefficient Number constructor; use static valueOf instead	Medium	mostly harmless	0.57	0.85
Method invokes inefficient new String(String) constructor	Medium	mostly harmless	0.13	0.63
Method invokes inefficient new String(String) constructor	Medium	mostly harmless	0.03	0.87
Exception is caught when Exception is not thrown	Medium	Not a bug	1.80	0.30

<Figure 3.1.2 Bug Patterns that FindBugs found for Othello>



<Figure 3.1.3 Priority that FindBugs assigned>



<Figure 3.1.4 Priority that is assigned manually>

Above two figures show the percentage for each priority. The figure 3.1.3 shows the percent of priority that FindBugs found, and the figure 3.1.4 shows the percent of priority that we rated for each bug.

FindBugs considers most bugs as Medium priority, 90% of bugs, and 10% as high priority bugs. We assume that High is critical to the system that can cause the system failure, medium might potentially cause the system failure; but not currently, low priority is just minor error that does not causes critical impact to the system, and ignore is “not a bug”, which means that do not need to be considered for the system.

However, when we investigated the bugs, we could not find the bugs that are critical to the system. One of the examples that FingBugs found as high priority is “Field isn't final but should be”, but the field value is not changed in the system, which means that it could potentially cause the error, but since the field is not changed, it does not cause the problem, so we consider this as “should fix”. Likewise, our analysis on bugs’ priority is much different from FindBugs’s analysis.

In addition, FindBugs does not have any ignore bugs, which can be considered as false positive errors, but we found a couple of bugs that are not considered as bugs. For example, there is a class that starts with lower case letter for the class name. FindBugs catches these as bugs (medium priority), but these do not affect the system at all. These were the style of program as we concluded. Therefore, we consider this as not a bug. Likewise, we found 4 bugs that are not bugs. **20% of bugs are false positives.**

Based on this analysis, we will compare the result against bugs that we found by using manual, unit, code review, and automated testing in next section.

3.2 Comparison with Assignment 2

3.2.1 Purpose

The purpose of this evaluation is to compare the bugs that FindBugs found and bugs that we found by manual, code review, unit, and automated system testing. By comparing these approaches, we would like to achieve information how useful it is compared to other analysis methods, and its limitation.

3.2.2 Artifacts

Bug description of Othello from assignment 2

3.2.3 Comparison

Now, we are about to compare the result from FindBugs with bug list of assignment 2. In the assignment 2, we conducted four types of testing, and it can be briefly summarized as follows, according to our answer of the assignment 2. See appendix for Bug list information

- Manual: Manual testing finds bugs that are classified as failures and errors. These include incorrect outputs that clearly violated the requirements behavioral and specifications. GUI and usability issues can be also found with manual testing.
- Automated System: Automated System testing finds behavioral bugs, both failures and errors.

- Unit: Unit testing predominantly finds faults local to a method such as parameter ordering, as well as some violations of behavior described in the requirements. This is the primary method by which we can find higher-level design and architecture issues.
- Code Review: Some of these bugs are relatively low in importance, such as an object that was not checked for being NULL, or a variable declared public, not private. Other bugs found by code review may cause violation of behaviors requirements as well as failures, so are significantly important.

Based on the description of each testing, we assumed that several bugs will be overlapped with the bugs found by code review, because the manual, automatic and unit testing are supposed to check behavioral bugs based on method contracts. Since FindBugs is a static analysis tool, we expected FindBugs to find most of bugs similar with the bugs found by code review.

Unfortunately, we found only one similar bug (Figure 3.2.1) in bug list of assignment 2, comparing with the bugs of FindBugs, even if FindBugs found a lot of unsound bugs. The bug found by assignment 2 describes a race condition issue, but the bug is not the exactly matched with a FindBugs's bug either.

	Bug description
Bug found by FindBugs	The fields of this class appear to be accessed inconsistently with respect to synchronization.
Bug found by code review	The program has a race condition when open the saved game. After a user opens the saved game and does not one move; then, it messes up the board due to two threads running.

<Figure 3.2.1 Bug Description>

As a result, we only found one common bug in bug list of assignment2 and bugs from FindBug, even if FindBugs detected lots of unsound bugs. The first reason of that is because we did code review based on both the contract of methods and the code review checklist, but FindBugs performed the static analysis regardless of the contract of methods since it could not verify the contracts. Therefore, even though both approaches are static, the results are different. For example, a race condition bug that FindBug found is not based on the contract, so FindBugs only describes the bug generally. Even FindBugs indicates the bugs that related to race condition issue, the bug description is not exactly matched with the bug that we found by code review.

Another reason of that is because the Othello application does not contain any bugs related to correctness that can be analyzed by static analysis such as Null reference, infinite loop; in other words, if the Othello application contains those bugs, they could be discovered by both FindBugs and Code review. However, the Othello does not contain the correctness bugs, and the bugs that we found do not cause the system failure; the bugs found by code review are mostly related to the incorrect behavior of Othello. Therefore, we could not find the common bugs from two different testing.

Finally, FindBugs found many unsound bugs we couldn't discover in code review, because our code review usually depends on our domain knowledge, such as Java language and Objected Oriented Programming, in spite of using a code review checklist. Additionally, FindBugs has a bug database internally, based on the research of a FindBug development team, so that FindBugs can find predefined unsound bugs better than human.

These are the reasons that there is only one common bug that we could find by using FindBugs and Code review. In addition, according to this result, we could see that when and how we should use FindBugs. We will discuss this in the Lesson Learned section.

3.3 Evaluation with Other Applications

3.3.1 Purpose

The purposes of this test are as follows. The first of all, comparison between the size of code and running time on FindBugs was performed to analyze the performance of FindBugs tool. Secondly, the percent of correctness bugs on total bugs found by FindBugs is analyzed because correctness bugs are critical to program, but other bugs might not be critical.

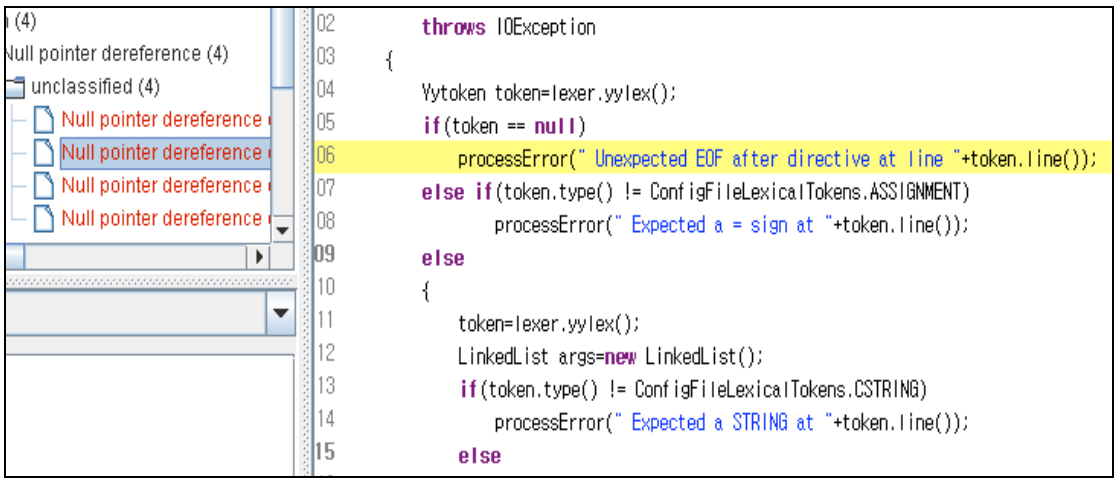
3.3.2 Artifacts

Othello, netMANj, JMSN, and JNM applications are tested on FindBugs tool and evaluated. Refer to 3.1 for Othello application. netMANj is a network management software in JAVA. JMSN is an open source application of the MSN messenger client written entirely in Java under the BSD license, including instant messaging and File Send/Receive. Java Network Monitor (JNM) is an kind of open source Java application for the purpose of defining monitors which can poll network services at defined intervals and execute actions defined by users when the service status changes. JNM includes two monitor implementations: TCP monitor and HTTP monitor.

3.3.3 Analysis

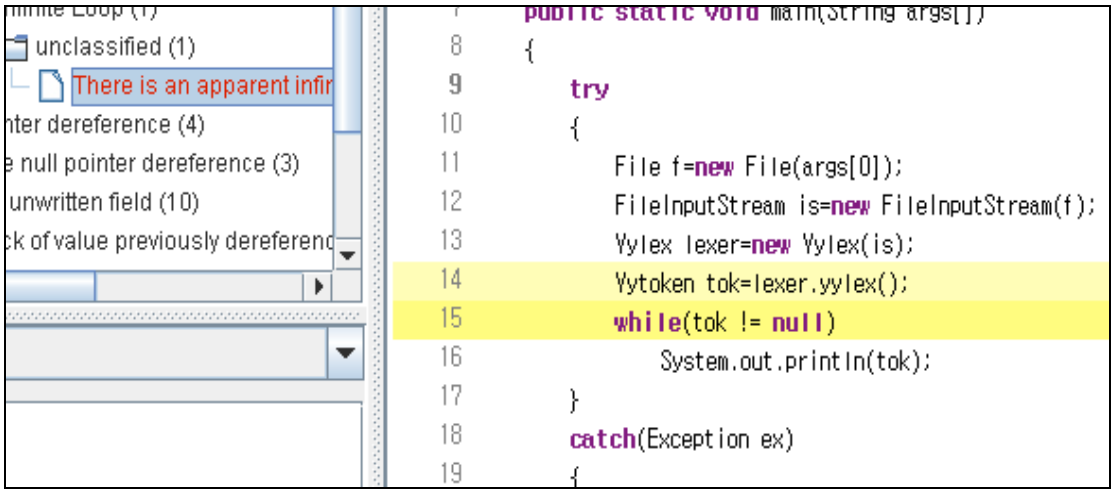
The 350 bugs from netMANj are found by FindBugs. From JNM, total 44 bugs are found. Among these bugs, most bugs are correspondent to the bad practices and performance category. We analyzed the version 0.9 of JMSN and checked defects that were presented in this tool. We found the total of 60 bugs, which corresponds to a defect of 0.18/ KLOC.

For other applications, FindBugs found several correctness bugs. Especially, unlike Othello, FindBugs found 28 correctness bugs in netMANj application. Followings are some examples of them.



<Figure 3.3.1>

Figure 3.3.1 shows that FindBugs found null pointer dereference bugs. In this case, 'processError()' method tries to call token.line() method, even if 'token' is equal to null value in the source code highlighted with yellow. That is, this bug must be critical defect in the source code, and FindBugs also regarded this bug as high priority.



<Figure 3.3.2>

Figure 3.3.2 shows that FindBugs found an apparent infinite loop bug. In this case, while 'tok' is not equal to null value, 'System.out.println(tok);' is executed. However, 'System.out.println(tok)' does not change the value of 'tok'. Therefore, once 'System.out.println(tok)' is executed, while loop will be executed infinitely. That is, this bug must be critical defect in the source code, and FindBugs also regarded this bug as high priority.

As a result, we can know that FindBugs can still find explicit bugs, which have critical impact on the application.

The bugs are evaluated by two ways. The statistics of bug category are shown in Figure 3.3.2.

Analysis of comparison between the size of code and running time is shown in the Figure 3.3.1. The analysis time of each program on FindBugs is proportional to the size of the program.

Program	#Download	#Lines	Running Time (sec)
Othello	N/A	3,877	7
JNM	7515	5396	8
JMSN	246,262	11188	17
netMANj	2417	24,933	33

<Figure 3.3.1 Test Programs Info>

The proportion of correctness bugs to the total bugs is evaluated in order to analyze how FindBugs can find critical bugs because the correctness bugs are critical to the system compared to other bug categories such as bad practice, dodgy, performance, malicious code vulnerability, and multithreaded correctness. However, as shown in Figure 3.3.2, the percent of the correctness bugs are the smallest compared to other bug categories, and other bugs do not cause the serious problems. Therefore, if the project is under time pressure, it is not a good idea to use FindBugs tool to find bugs because FindBugs finds more bugs that are not directly related to the system failure or other bugs that violate the requirements.

Program	Bad Practice (%)	Malicious code vulnerability (%)	Multithreaded correctness (%)	Performance (%)	Dodgy (%)	Correctness (%)	Total (%)
Othello	60.00%	5.00%	15.00%	15.00%	5.00%	0.00%	100.00%
netMANj	13.14%	8%	0.57%	59.72%	10.57%	8.00%	100.00%
JNM	56.8%	11.4%	2.3%	29.5%	0.0%	0.0%	100.00%
JMSN	71.7%	3.3%	1.7%	6.7%	13.3%	3.3%	100.00%
Average	**50.41%**	**6.93%**	**4.89%**	**27.73%**	**7.22%**	**2.83%**	100.00%

<Figure 3.3.2 Statistics of Bug Category>

4. Lesson Learned

4.1 Benefits

FindBugs provides the various versions of application: standalone, eclipse plug-in, or web applet. It also easy to use since it only has the couple of options to select, and it performs the operation in reasonable time even though the testing files are big. FindBugs can analyze the 276 types of bugs that not only include correctness, but also performance, code vulnerability, and others, so that it improves the robustness of the application even though the bugs are not critical to the application. In addition, it provides the traceability where bugs occur in the source code and detail information that describes what the problem is, so it helps the developers fix bugs easily and quickly.

4.2 Drawbacks

Since FindBugs analyzes the application statically, it could not find the bugs that could happen in runtime such as race condition, and also it could not analyze the contract, so that FindBugs finds the bugs mechanically. Even though we can add annotation to let FindBugs understand our design intension (contracts), FindBugs could not find all bugs based on the contract due to the limited annotations, which are mostly related to null dereference checking. In addition, as we discussed in section 3.2, FindBugs tool could not find all the bugs, so that it should be used with other testing tools and methods.

4.3 Scope of applicability

FindBugs has disadvantages, but it still helps to ensure the quality of applications. This tool is mainly used for analyzing the applications statically to check null dereference, infinite loop, type checking, and others that related to the correctness, performance, and vulnerability issue. Therefore, developers can use this tool for pre-inspection in early development stage, so that it could reduce the testing time in later test phase.

5. References

[1] http://findbugs.cs.umd.edu/
[2] http://findbugs.sourceforge.net/
[3] http://www-128.ibm.com/developerworks/java/library/j-findbug1/

Appendix – Bug list of Assignment 2

#	Bug Description	File (if known)	Lines	Found with
1	The program supposes to save the file with its own extension such as ".oth", but it does not provide the extension automatically when the game is saved.	BoardGameWin.java	223, 246 ~ 256, 308 ~ 316	Manual System Testing, Unit Testing, Code reviews, Automated System Testing
2	The program supposes to open a game with the saved file, but the program could not recognize the file, which is made through the saving procedure in the program. But, it read a file with .oth extension.			Manual System Testing
3	The program does not provide the information that who won the game at the end of the game.			Manual System Testing, Automated System Testing
4	When the Minimax depth is set to 0, the computer's move should be occurred randomly and only in valid spot, but it always goes to the coordinate (0,0) because the algorithm for minimax level could not understand level 0 due to the code, "level < mLevel". This should be level <= mLevel.	AIThread.java	100	Manual System Testing, Code reviews
5	The history of the game should be displayed properly, but it does not work on MS Windows. It works in Mac OSX. It needs pack() to regenerate the screen correctly.			Manual System Testing
6	(Require Clear Explanation) The program undoes just one movement. However, the user expects the undo as changes the status back to the user.			Manual System Testing
7	In the othello rule window, the side weight and near side weight give confusion because of the position. The position of side and near side seems to be switched in GUI.			Manual System Testing
8	The program has a race condition when open the saved game. After a user opens the saved game and does one move; then, it messes up the board due to two threads running.	BoardGameWin.java	359	Manual System Testing, Code reviews
9	The program provides an error messages to the .out file when an undo is performed right after loading a saved game file.			Automated System

				Testing
10	The program doesn't show the message of "Game Over" when the game is finished.			Automated System Testing, Manual Test
11	The program shows a wrong message, 'Bad end of game: XX', when the game is over.			Automated System Testing
12	The program doesn't reflect the new weight values to all the pieces in case the weights are changed to new values.			Automated System Testing, Manual Test
13	In checkWeightEqual method, the method invokes getWeight method with switched row and column. Ex) opts.getWeight(r, c) == opts2.getWeight(c, r) This should be opts2.getWeight(r,c)	CloneTest.java	62	Unit Testing, Code reviews
14	In getWeight method, the method receives row (r) and column (c) as arguments, but when it uses, row is used as column and column is used as row. Ex) public int getWeight(int r, int c) { return mWeights[c][r]; }	Options.java	76	Unit Testing, Code reviews
15	In the middle of the game, if I do passmove, and undo the passmove, and do passmove again. Then, the game is over, which it should not be.	OthelloBoard.java		Unit Testing
16	toString method recognizes as Passmove only if row is -1, but when a row is less than -1 such as -2, -3, it should be passmove as well. However, it does not.	OthelloMove.java	59	Unit Testing, Code reviews
17	The method, clone(), copies a reference rather than actual values for int[][]. It means that if the original object gets changed, the copied one also gets changed. This is not proper clone method.	Options.java	53	Unit Testing, Code reviews
18	In unDoLastMove method, the mValue is not calculated properly.	OthelloBoard.java	265 ~ 267	Unit, Code reviews, Manual System, Automated System

19	Othello's extension is supposed to be ".oth" since it only reads the file with the extension ".oth", but the system recognizes ".othello" as readable file name, and even it can't be opened	Othello.java	26	Code Review, Manual System Testing
20	After a user plays a game with a status opened, if a user plays a new game, the status does not get changed until the user does his new move. Then, the status view is updated. (The status should be updated as long as new game is clicked.)	BoardGameWin.java	273 ~ 300	Code reviews, Manual System Testing
21	Once a file is opened for use, it should be closed, but it does not.	BoardGameWin.java	308 ~ 316	Code reviews
22	If a user opens a saved game when the system just launched, the status view, game config, swap function, etc are disabled. Those should be enabled when the game starts.	BoardGameWin.java	332 ~ 370	Code reviews
23	mNumPass is located in the wrong place because it should be increased before the system checks isGameOver(). Otherwise, the system could not print out the message "Game over".	OthelloBoard.java	96 ~ 107	Code reviews
24	(writeObject method) When the system saves the file, does not record the history of the moves. This is necessary for Undo.	OthelloBoard.java	364 ~ 370	Code reviews
25	(readObject method) When a file is opened, the history is always created as new history rather than bringing the saved history.	OthelloBoard.java	380	Code reviews
26	The program receives "MakeMove" when the game is over. For example, when pass and pass occur, the program is over, but if the test executes "MakeMove" several times, then the sequence of output will be pass, pass, (0,0), pass, pass, (0,0), …			Automated System Testing
27	When the program is launched, and a user tries to open the saved game, it will causes error because at first there is no running thread, but the opening file method tries to catch a thread to stop.	BoardGameModel.java	120	Manual System Testing, Code reviews
28	mWhosTurn variables is not properly set. For example, in the code, mWhosTurn = new JLabel ("Black Wins"); However, in the code, mWhosTurn should be set to the screen by using "setText" rather than creating new JLabel.	StatusView.java	75	Code reviews