Team Simulacrum (Choudhari, Jaspan, Mouri, Nagata, Urie) 6/16/06

Analysis Tool Project

Tool Overview

The tool we chose to analyze was the Java static analysis tool FindBugs (http://findbugs.sourceforge.net/).

FindBugs is

- A framework for writing static analyses
- Developed at the University of Maryland
- Actively maintained
- Open source
- Written in Java

FindBugs does

- Static analysis of java code
- Class structure analysis
- Linear code scans
- Control sensitive analysis
- Dataflow analysis

FindBugs uses

- A java parser called BCEL
- Around 40 pre-written static analyses
- “Bug patterns” to find bugs

Purpose

The purpose of FindBugs is to show that a large amount of bugs can be found by writing small and simple analyses. FindBugs believes this is true because developers tend to make frequent mistakes and many times these mistakes are simple. For example, in a Java serializable class it is very common to forget the version id, or to have a reference to a non-serializable class. These types of bugs are sometimes hard to track down by running the application or by code reviews; however, they are simple to discover using simple static analyses.

BCEL

Byte Code Engineering Library (BCEL) is a library written by apache that parses java byte code. Java byte code contains all of the symbolic information needed to do static analyses, including methods, fields, inheritance, and byte code instructions.

BCEL has written a custom Java Virtual Machine (JVM) class loader that reads in java byte code and dumps out a file in its own custom format. It then provides an OO interface to that file which allows FindBugs to access everything that it needs to do its static analysis.

Bug Patterns

After noticing that developers make similar, simple mistakes FindBugs developed the concept of a bug pattern. A bug pattern is a code idiom that has a high probability of being an error. For example, not checking return values from functions that are known to return error values. Most of the time forgetting to check these return values is an error. Furthermore, since the problem would only been seen through an uncommon error path it isn't likely to be caught in black box testing.

FindBugs ships with over detector for over 200 bug patterns. They range from synchronization errors to bad exception handling to hard coded references. Each bug pattern has varying levels of soundness and accuracy but most of them are simple. The longest detector is around 1000 LOC but over ½ are less then 100 LOC.

An interesting aspect of FindBugs is that all of the bug patterns are heuristic based. This means the tool isn't sound or correct, but it is still useful (as our data shows). Find bugs isn't trying to find all the bugs in your application nor is it trying to only report valid bugs. It is trying to report the low hanging fruit. By using bug patterns it is trying to report the bugs that will be easy to verify and easy to fix.

Find Bugs as a Framework

The pre-written bug pattern detectors in FindBugs are written using FindBugs powerful framework. Even though FindBugs main use case is to use the pre-written detectors, it is easy to write your own analysis.

The find bugs framework provides the ability to do:

- **Class structure analyses:** An analysis can look at the structure of the class to find possible defects. Some examples of what an analysis can look at are inheritance, methods, fields, as well as the access modifiers on methods and fields.
- **Linear code scans:** An analysis can do a linear code scan through byte code to drive a state machine. These analyses can approximate a control flow graph but they don't make use of a complete control flow graph.
- **Context sensitive analyses:** An analysis can make use of an accurate control flow graph to do analysis of methods.
- **Dataflow analyses:** An analysis can make sure of both control and data flow information. These analyses are more complex then the other 3 but are more powerful. An example is FindBugs pre-written null point analysis (which isn't as good as the one we wrote).

Tool Usage

FindBugs can be run as a standalone application utilizing a Swing interface, a command-line application, or an Eclipse plug-in. There is even an Ant task provided so that FindBugs can be easily incorporated into an Ant-based project.

Standalone Application

The standalone, Swing version of FindBugs is very simple to get started with. The GUI presents the user with a blank window at first. To setup a project to analyze you go to `File->New Project`.

You are then presented with a window with places to browse for and add each of the required sets of files and directories. To analyze a program, FindBugs requires the

- Source files.
- Compiled class files.
- Libraries (jars or class files).

The main window is shown in Figure 1. After supplying the required files and directories, you click the "Find Bugs!" button at the bottom of the window.

Analysis Tool Project

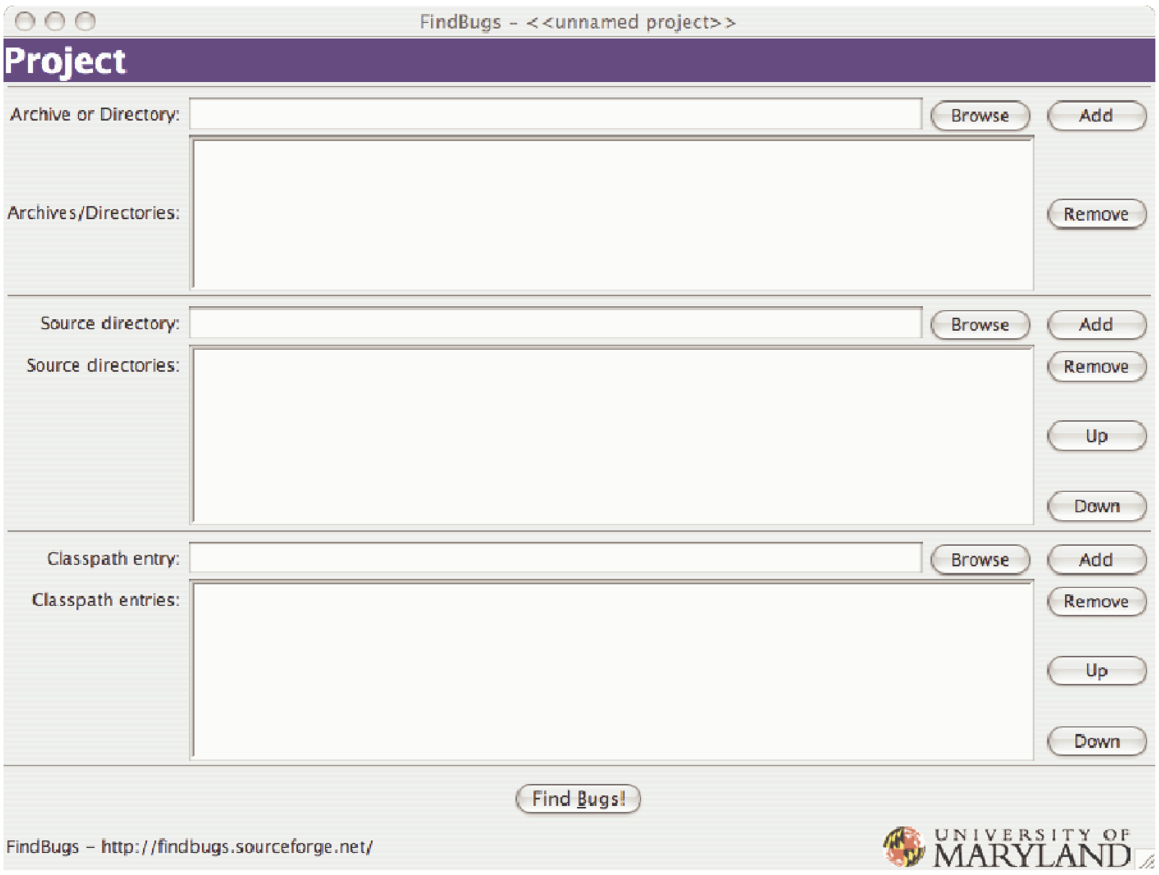


Figure 1 The main FindBugs window.

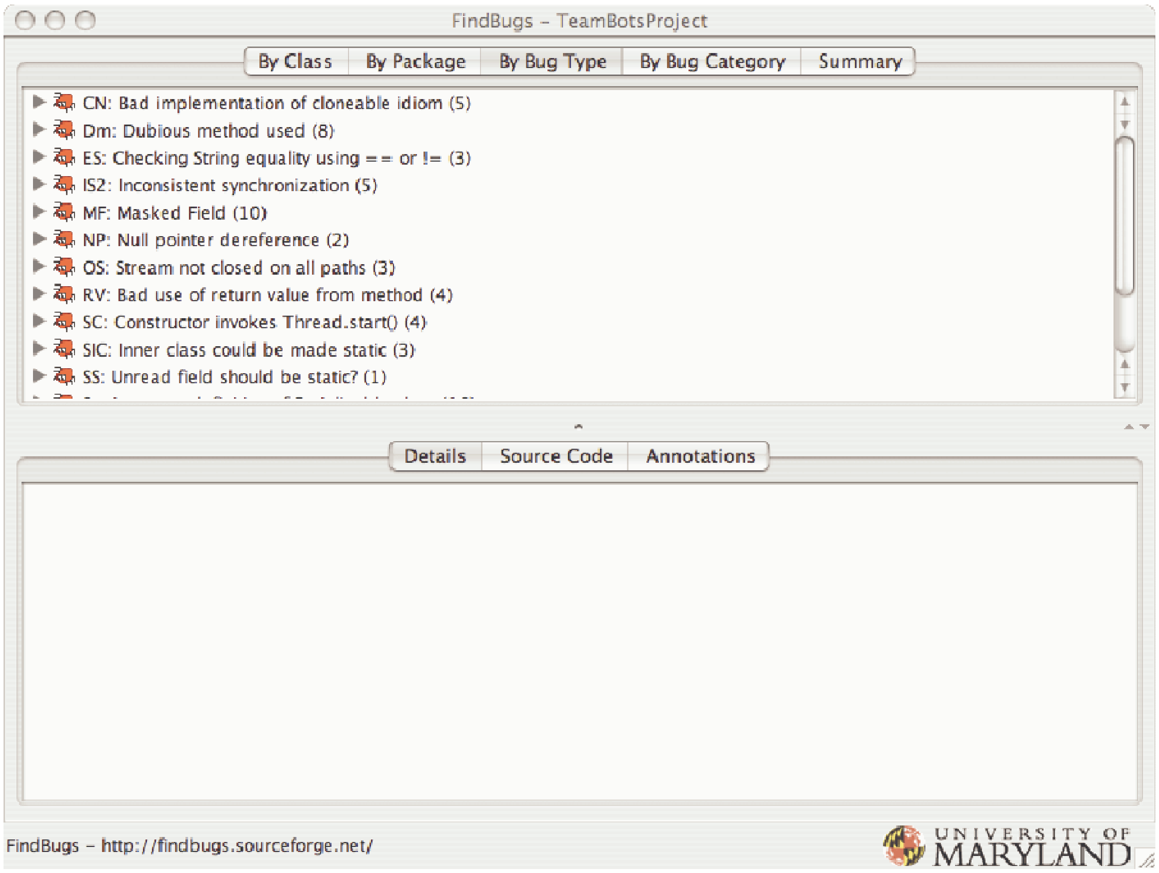
FindBugs provides a few customization features. It separates bugs into 6 categories:

1. Correctness
2. Multithreaded correctness
3. Performance
4. Style
5. Internationalization
6. Malicious code vulnerability

Each category can be turned off if you don't care about those kinds of bugs. Additionally, you can tell FindBugs how much effort to put into detecting bugs

- Low
- Medium
- High

The higher the effort, the longer the analysis takes. You supposedly find more bugs by making FindBugs put in more effort.



CN: Bad implementation of cloneable idiom (5)
Dm: Dubious method used (8)
ES: Checking String equality using == or != (3)
IS2: Inconsistent synchronization (5)
MF: Masked Field (10)
NP: Null pointer dereference (2)
OS: Stream not closed on all paths (3)
RV: Bad use of return value from method (4)
SC: Constructor invokes Thread.start() (4)
SIC: Inner class could be made static (3)
SS: Unread field should be static? (1)

Figure 2 FindBugs bug report window.

FindBugs also comes with many detectors. A detector is a module designed to find a specific type of bug. The application provides a nice table listing all of the detectors in `Settings->Configure Detectors`. The resulting window lists the detectors by name, how fast they are, and whether they are enabled or not. You can toggle any detector's active state from this window.

The analysis time will vary depending on which of the above options you chose. We never saw it take longer than about 5 minutes on a project consisting of 237 classes with all of the options maxed out.

After the program analyzes your code, you are presented with an error window if it had trouble finding any classes. If it did, it will still present you with the results of the tests that worked.

In the window that presents the bugs found, there are 5 tabs that give you the information in different ways:

1. By Class
2. By Package

3. By Bug Type
4. By Bug Category
5. Summary

You can also change what bugs you see by changing the warning filters in `View->Filter Warnings.` The program prioritizes warnings from Low to High and Experimental which seems to lie below the Low priority. The Experimental warnings are those associated with experimental detectors or features of detectors.

Clicking on a reported bug shows you a description of the bug is the lower pane by default. Here, the developers were smart and nice enough to not only describe the bug, but also suggestions on how to check if it is a bug (they admit many times that not everything reported is a genuine bug) and if it is, they offer suggestions on how to fix it.

The lower pane has 2 other tabs in addition to the bug description pane. The second one shows where in the source code the bug occurs including the surrounding lines. The third tab shows any annotations.

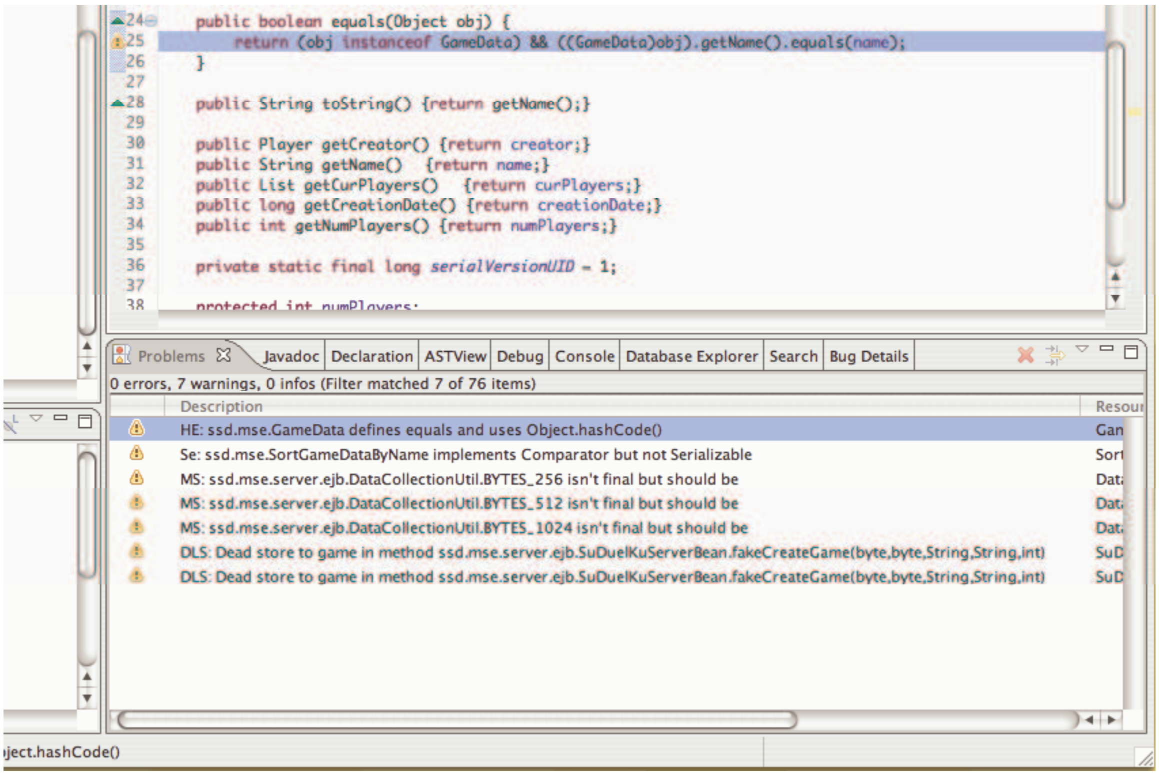
Eclipse Plug-in

The FindBugs Eclipse Plug-in is very easy to install and use. As a plug-in FindBugs works seamlessly with the code editor, which makes the task of verifying and fixing bugs very easy.

To install the plug-in, you can download it from http://prdownloads.sourceforge.net/findbugs/de.tobject.findbugs_0.0.20.zip?download and unzip it in your Eclipse plugin directory. Once installed, FindBugs will be accessible through the right-click menu of the Java project. To run FindBugs, simply right click on the project and select “Find Bugs”.

In the testing environment where we used the plug-in, Findbugs plugin ran within a minute. After completing its run, FindBugs displays the bugs found in the Eclipse Problems Tab, as shown in Figure 3

Analysis Tool Project



```
public boolean equals(Object obj) {
    return (obj instanceof GameData) && ((GameData)obj).getName().equals(name);
}

public String toString() {return getName();}

public Player getCreator() {return creator;}
public String getName()  {return name;}
public List getCurPlayers()  {return curPlayers;}
public long getCreationDate() {return creationDate;}
public int getNumPlayers() {return numPlayers;}

private static final long serialVersionUID = 1;

protected int numPlayers;
```

0 errors, 7 warnings, 0 infos (Filter matched 7 of 76 items)

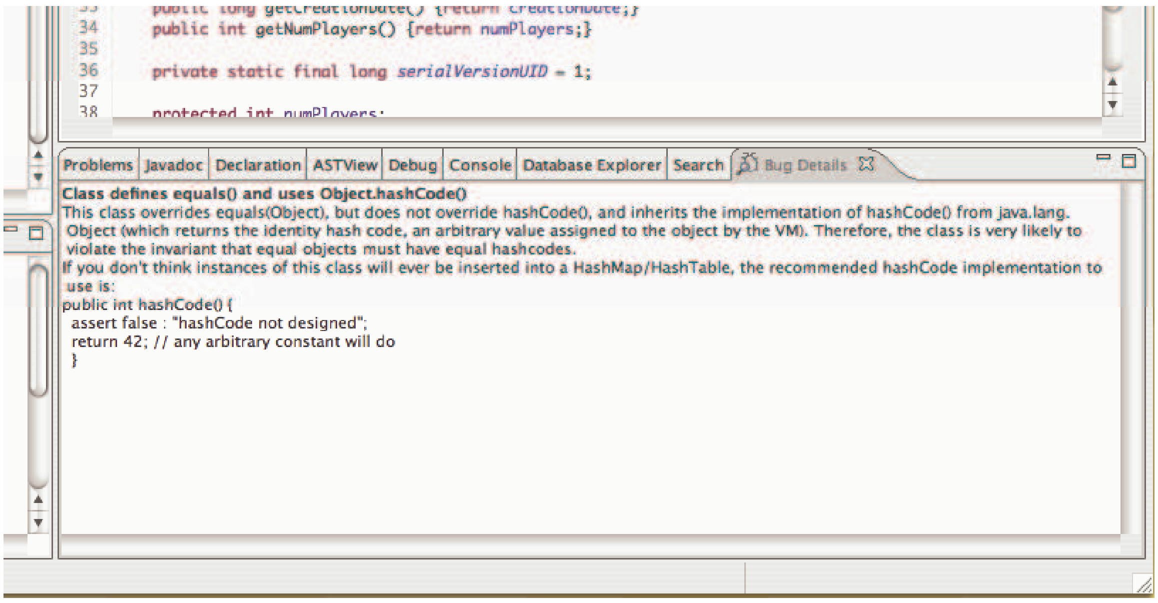
Description	Resour
HE: ssd.mse.GameData defines equals and uses Object.hashCode()	Gan
Se: ssd.mse.SortGameDataByName implements Comparator but not Serializable	Sort
MS: ssd.mse.server.ejb.DataCollectionUtil.BYTES_256 isn't final but should be	Data
MS: ssd.mse.server.ejb.DataCollectionUtil.BYTES_512 isn't final but should be	Data
MS: ssd.mse.server.ejb.DataCollectionUtil.BYTES_1024 isn't final but should be	Data
DLS: Dead store to game in method ssd.mse.server.ejb.SuDuelKuServerBean.fakeCreateGame(byte,byte,String,String,int)	SuD
DLS: Dead store to game in method ssd.mse.server.ejb.SuDuelKuServerBean.fakeCreateGame(byte,byte,String,String,int)	SuD

ject.hashCode()

Figure 3 - FindBugs reporting bugs in Eclipse Problems tab

The problems are associated with lines in the source code which simplifies the navigation between the different bugs: simply double click on the problem and Eclipse will show the file and line where the problem is.

FindBugs provide its own custom window called “Bug Details”. If you need more detail on the type of bug that was found, you can right click on the bug in the Problems tab, and choose “Show bug details”.



```
    public long getCreationDate() {return creationDate;}
    public int getNumPlayers() {return numPlayers;}

    private static final long serialVersionUID = 1;

    protected int numPlayers;
```

Class defines equals() and uses Object.hashCode()
This class overrides equals(Object), but does not override hashCode(), and inherits the implementation of hashCode() from java.lang.Object (which returns the identity hash code, an arbitrary value assigned to the object by the VM). Therefore, the class is very likely to violate the invariant that equal objects must have equal hashcodes.
If you don't think instances of this class will ever be inserted into a HashMap/HashTable, the recommended hashCode implementation to use is:

```
public int hashCode() {
  assert false : "hashCode not designed";
  return 42; // any arbitrary constant will do
  }
```

Figure 4 - Bug details in Eclipse

The same configurations (bug detector, effort, etc) that are available in the standalone version are available with the plugin.

Testing Environment

We tested FindBugs with two Java projects:

1. TeamBots
2. SuDuelKu

TeamBots

TeamBots is a open source API to control intelligent mobile agents. More information can be found here: http://www.cs.cmu.edu/~trb/TeamBots/

The general size of TeamBots:

Team Bots Info	
KLOC	20865
Number of Classes	231

Table 1 – Size of TeamBots

SuDuelKu

SuDuelKu is an EJB multiplayer SuDoKu game. More information can be found here:

http://www.ece.cmu.edu/~ece749/teams-06/team1/html/index.html

The general size of SuDuelKu:

SuDuelKu Info	
KLOC	7715
Number of Classes	183

Table 2 – Size of SuDuelKu

Setup

We ran FindBugs on both WindowsXP and MacOSX. We mostly ran it as the standalone application but we did play with the Eclipse plug-in.

We decided to filter out all of the warnings from the Style, Internationalization, and Malicious Code Vulnerability categories. We did this because we wanted to focus on the most significant bugs that FindBugs was capable of analyzing. The malicious code vulnerability bugs were originally considered but it turned out they were all warnings about disclosing a class's implementation, or making a variable package protected. While these would be important to look at eventually, we wanted to see if there were any bugs that might cause a malfunction of the program.

We set the effort to maximum and ran the analyses with all of the detectors. We filtered out all of the results whose priorities were not low or experimental. We also ignored the Unread Field and Unused Field errors because we felt those were not serious, program-breaking errors.

Results

Highlights

The bugs that FindBugs reported were

- Mostly "valid" (27% false positives)
- Easy to validate (Median 1 minute, Mean 1.57 minutes)
- Easy to fix (Median 1 minute, Mean 2 minutes)

However, the vast majority of our bugs were fault and not errors. They were problems that the user would never see; however, if the classes were used differently, or changed they could easily become errors.

In our experimentation, we missed the opportunity to classify the bugs found by their severity. Severity categorization could take in consideration either faults and errors, and evaluate their potential impact on the final product. FindBugs classify the bugs according to priorities (High or Medium) but we could not establish a connection between priority and severity. The severity categorization would have allowed us to evaluate FindBugs value, for a real environment situation when bugs are scheduled to be fixed according to a severity categorization.

Totals

Generally, FindBugs reported few false positives

Total reported bugs	65
Total false positives	18
Total bugs	47
% of false positives	27.69%

Table 3 – Total bugs reported by Find Bugs

Runtime

FindBugs runs relatively fast, although we are worried about how well it scales.

Project	**Time (secs)**
TeamBots	26
SuDuelKu	15

Table 4 – Runtime of Find Bugs

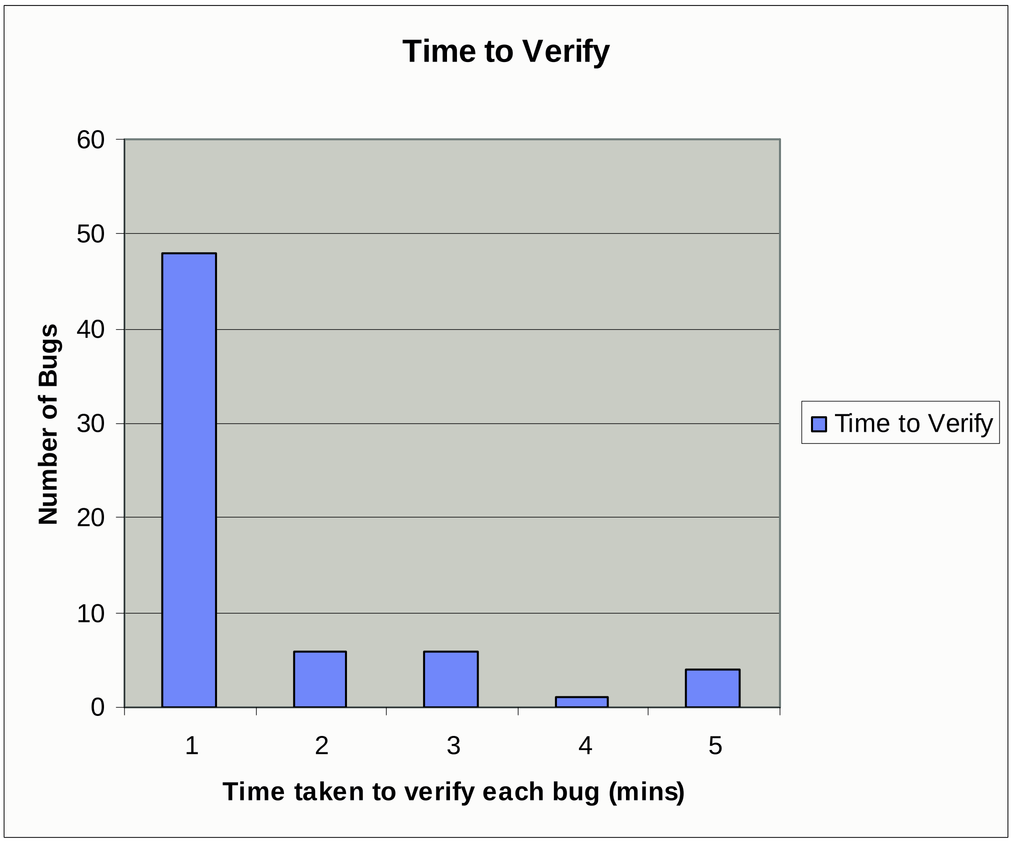
Verification

Find bugs reported bugs that were easy to verify.

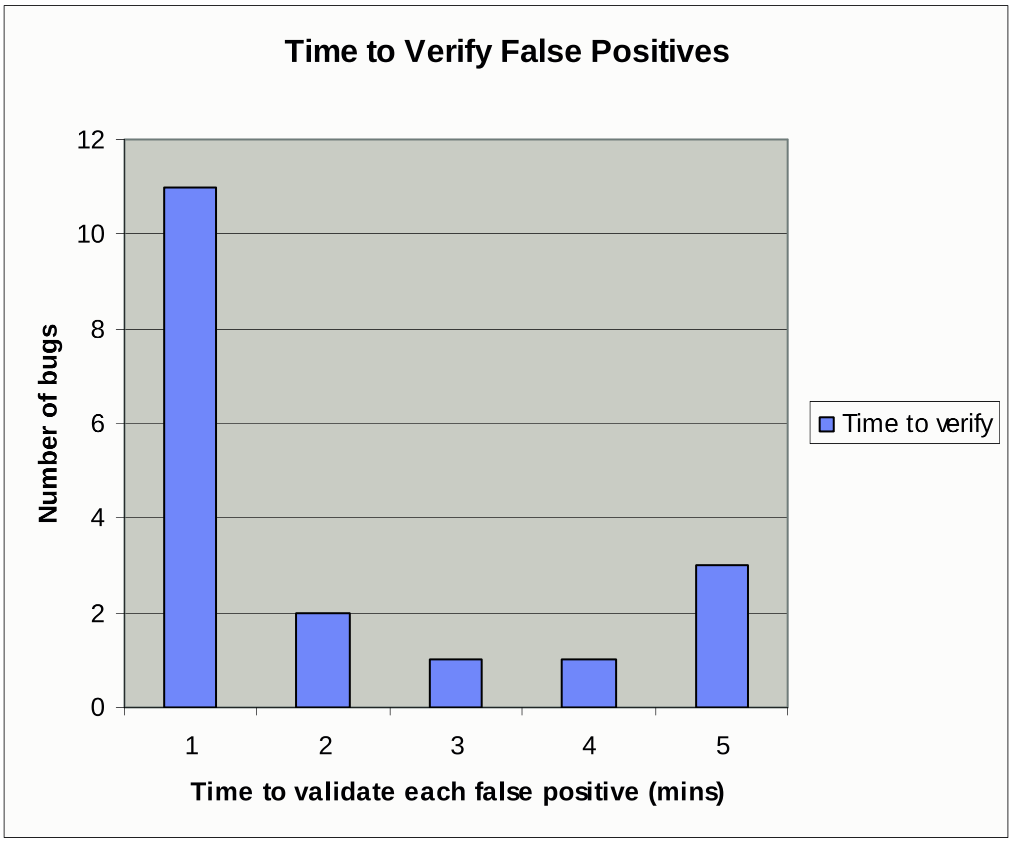
Verification	
All Bugs	
Total time to verify	102.00
Mean time to verify	1.57
Max time to verify	5.00
Min time to verify	1.00
Median time to verify	1.00
Standard deviation on time to verify	1.13
False Positives	
Total time to verify	37.00
Mean time to verify	2.06
Max time to verify	5.00
Mix time to verify	1.00

Median time to verify	1.00
Standard deviation on time to verify	1.59

Table 5 – Verification times of for bugs reported by FindBugs



Graph 1 – The majority of bugs reported by FindBugs take a short amount of time to verify.



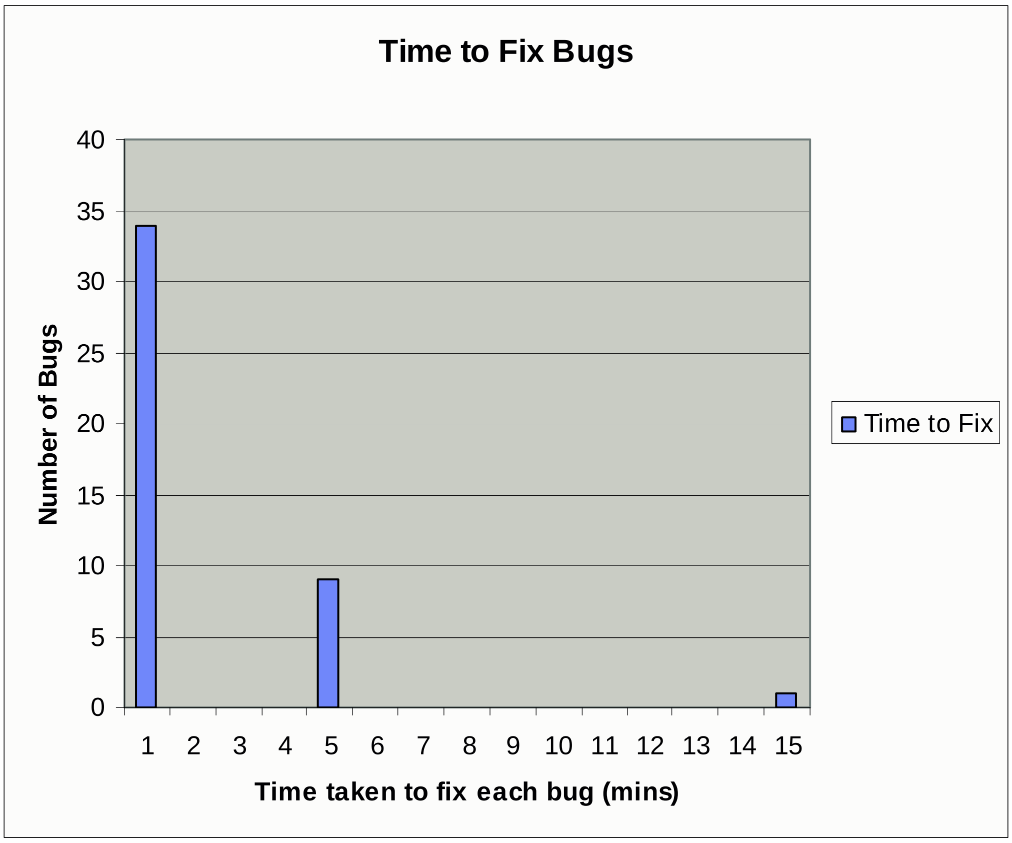
Graph 2 – Even the false positives take a short amount of time to verify.

Fixing

Find bugs reported bugs that were easy to fix.

Fix	
Total time to fix	94.00
Mean time to fix	2.00
Max time to fix	15.00
Min time to fix	1.00
Median time to fix	1.00
Standard deviation on time to fix	2.46

Table 6 – Fix times of for bugs reported by FindBugs



Graph 3 – Fixing the valid bugs found by FindBugs also takes a short amount of time.

All

Complete data is attached in .xls file

Lessons Learned

FindBugs is easy to set up and get running. Its customizability makes it easy to tailor it to the bugs you are looking for. It doesn't take too long to run and the report window makes it very simple to find the bugs, classes, or packages of interest. The summary pane is very helpful if only for improving one's own sense of self-satisfaction as the total bugs and percentages drop.

FindBugs, however, is noticeably beta-ware. Setting the warning filters and detector choices was a bit frustrating on Windows where, if you set them and then saved the project, those options would reset to their prior state. This didn't happen on the Mac. Also, the menus and menu items do not always make sense. For example, if you're looking at the bugs, change some of the detector or effort options and want to run it again you have to choose `View->Project Details` to get the initial window. This window is apparently different from the report window but you can't view both at the same time. Furthermore, priority filter settings would reset to the default settings every time you ran FindBugs, but any custom settings would appear to be saved, even when they were not. This can be very confusing to anyone wanting to rerun FindBugs on different sets of code with different filters. The source code viewer seemed to also be buggy in a small subset of cases, where the results pane would not bring up the appropriate block of code.

In addition, there seemed to be a max limit on the number of bugs FindBugs would find at once. For example, when the tool was run, a certain number of bugs were found. When these bugs were fixed, and the tool rerun to verify, one or two additional bugs in the same category would be detected. This was odd, because the new bugs were in completely separate classes than the previous bugs and were local to their classes, so there was no way these new bugs could have been introduced by fixing the old ones.

Furthermore, one very confusing thing about FindBugs is that the priorities that it gives to bugs seem to be random. There were many bugs that appeared multiple times in several different classes, but none would be rated the same priority. This seemed rather arbitrary and no explanation could be found.

Another interesting thing about FindBugs is that in some cases it gives fairly good recommendation for avoiding potential future bugs. For instance, calling the start method of the thread in the constructor of a class could potentially be a bug if the class is sub classed. FindBugs detects and recommends to change these errors. However, not all the recommendation given by FindBugs are relevant and they all essentially fall under false positive category.

Also, we came across a situation where FindBugs reported an error but it never really gave enough information about the bug for us to be able to fix it. It just reported the type of error and the class but it did not report any further information about what exactly was the error and where in the class that error was.

Tool Limitations

While we found that while the heuristics used in the bug detectors could be fairly accurate, it was obvious that some detectors were better than others. Additionally, according to the documentation, FindBugs lists the following shortcomings in several of its detectors

- **Pruning infeasible exception paths:** For some detectors, such as those looking for null pointer accesses, FindBugs is not able to eliminate branches of code that would make a null pointer impossible. For example, the following block of code returned a false positive in FindBugs as a possible null pointer error:

```
        if (pg != null)
        {
             pg.dispose();
        }
```

- **Accuracy in synchronization detector:** Even though FindBugs offers a synchronization detector to determine whether reads and writes to synchronized fields in separate threads were being properly locked and unlocked, FindBugs is not able to statically detect all situations in which a lock is held. Because of this, the probability of finding false positives in the synchronization detector can be somewhat high, extending the time a developer would need to validate such bugs.

Bottom Line

The bottom line is we would use FindBugs on future projects because FindBugs reports bugs that are

- Mostly valid
- Easy to verify
- Easy to fix.

This means that the team would spend little time using the tool while gaining considerable value. Furthermore, the tool helps find bugs that are difficult to find using other testing methods like white and black box testing. The only reasons we may not use the tool is if we couldn't find a way to suppress verified false positives.