Analysis of Software Artifacts

An Evaluation of FindBugs

Collection of unsound bug detectors for Java

Group 10 – Sandcastle
4/10/2009

Table of Contents

Overview of FindBugs

FindBugs is a tool that finds potential problems in Java code by using static analysis. Static analysis is a way to inspect code without executing the program. Basically, the tool inspects Java byte code which is saved in the form of complied class files, to detect occurrences of bug patterns.

Bug patterns

Bug patterns are checklist items for possible problems in the Java source. These patterns are defined in the tool and users can select patterns that they want to use to analyze the source. The patterns are categorized by the list below:

- Malicious code vulnerability – code that can be maliciously altered by other code.
- Dodgy – code that can lead to errors.
- Bad practice – code that violates the recommended coding practices.
- Correctness – code that might give different results than the developer intended.
- Internationalization – code that can inhibit the use of international characters.
- Performance – code that could be written differently to improve performance.
- Security – code that can cause possible security problems.
- Multithreaded correctness – code that could cause problems in multi-threaded environment.
- Experimental – code that could miss clean up of steams, database objects, or other objects that require cleanup operation.

Each category has multiple patterns, and the user can select patterns that will be used during the analysis. For example, there is a pattern that looks for infinite loops in the correctness category. The user can select this pattern to check for infinite loops.

Setup of the Experiment

Source Code to Analyze

We used Findbugs to analyze the checkers game that was developed in assignment 8 and 9. The game includes codes for the game framework, the game plug-in, the UI plug-in, and unit tests. Before analyzing with FindBugs, the application has gone through unit tests integration tests, and system tests. All the tests were successful. In addition, the application was verified with plural analysis.

Tool Setup

The tool provided three different user interfaces: command line, standalone GUI, and Eclipse plug-in. For this assignment, we used the Eclipse plug-in version. We configured FindBugs as follows:

- Detector configuration – We included all the patterns that FindBugs has provided except all the hidden patterns.

- Reporter configuration – We set to the tool to report all priorities (High, Medium, and Low) and all bug categories.

We included all the patterns because we wanted to see all potential defects that the tool could detect.

How to run

- Install FindBugs as an Eclipse plug-in
- Configure FindBugs as described in the "Tool setup" section.
- Import the checker game to Eclipse.
- Run FindBugs by selecting the FindBugs menu item for the project.
- The results are shown in the "Problems" panel.

Experiment Results

Size of the Analyzed Source Code

- 2,115 LOC (excluding comments)
- 56 Java classes

FindBugs Results

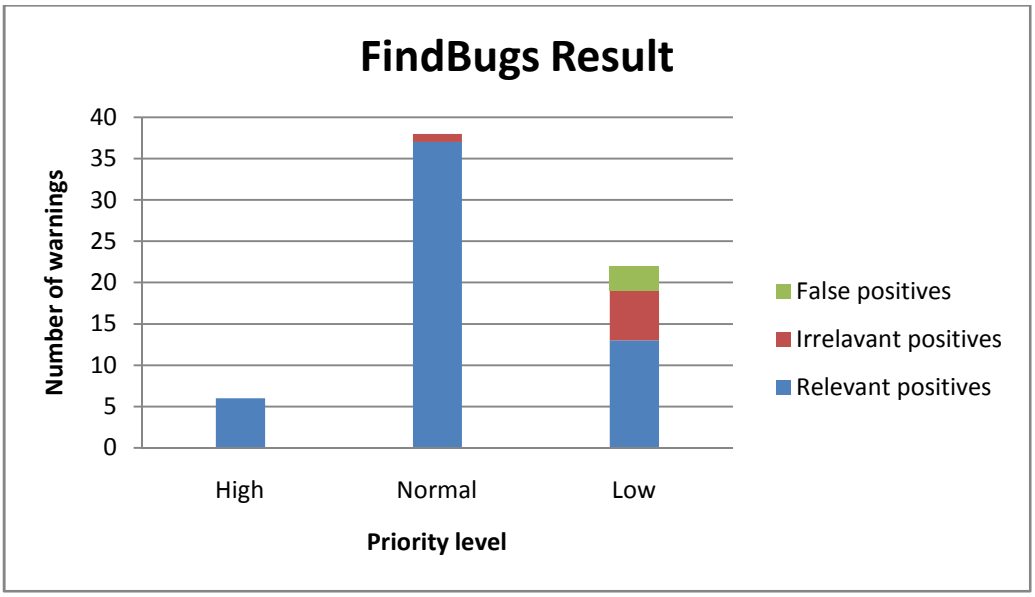
The total time that FindBugs used to analyze the code and provide the warnings was 9.3 seconds. Table 1 and graph 1 show the number of warnings found by Findbugs, and they are categorized by the priority reported by FindBugs.

The team analyzed each warning reported by FindBugs and categorized them into three types:

- Relevant positive – a bug that the developers must fix or should fix.
- Irrelevant positive –a bug but it is irrelevant to the program and does not need to be fixed.
- False positive – Not a bug.

Priority	Relevant positives	Irrelevant positives	False positives	Total
High	6	0	0	6
Normal	37	1	0	38
Low	13	6	3	22
Total	56	7	3	66

Table 1 – The number of warnings for each priority



Graph 1 – FindBugs result

Relevant True Positives

Relevant true positives were cases where the tool reported valid defects in the code and the code should be changed to eliminate the errors.

Associated Code

```
public class CheckersGameState implements State {
    ...
    public CheckersGameState clone() {
        ...
    }
}
```

Error Description

The following description was attached to the error:

```
aialg.checkers.CheckersGameState defines clone() but doesn't implement Cloneable
CheckersGameState.java     Group10 Game Framework/src/aialg/checkers      line 1
FindBugs Problem (High Priority)
```

The CheckersGameState class implements the clone method and allows cloning. However, it does not implement the Cloneable interface preventing it from being used in places that expect a Cloneable type. As indicated by the error, CheckersGameState should be changed to implement the Cloneable interface.

Associated Code

```
public class CheckersGameState implements State {
    public static int BOARD_SIZE = 8;
}
```

Error Description

The following description was attached to the error:

```
aialg.checkers.CheckersGameState.BOARD_SIZE isn't final but should be
CheckersGameState.java     Group10 Game Framework/src/aialg/checkers      line 16
FindBugs Problem (High Priority)
```

The error from FindBugs indicates that BOARD_SIZE should be final since it is a static variable that is being assigned a constant value. This is a valid defect since not declaring BOARD_SIZE final allows the variable to be changed which can cause problems in other code where BOARD_SIZE used and assumed to be unchanging.

Irrelevant True Positives

Irrelevant true positives were cases where the tool reported errors which were not defects in the code. The code may be changed to eliminate the error, or it may safely be left unchanged. The tool reported

Associated Code

```
public class CheckersGame extends Game {
    ...
    public AIPlayer createAIPlayer(String name, Color color, int level)
            throws NotSupportedException {
        ...
    }
    ...
}

public class CheckersAIPlayer extends AIPlayer {
    ...
    public Move suggestMove(@Full("gameStarted") Game game)
            throws GameOverException {
        ...
        AiBoardConverter converter = new AiBoardConverter();
        ...
    }
    ...
}

public class AiBoardConverter {
    ...
    public CheckersGameState convertBoard(@Pure CheckersGame game) {
        ...
    }
}
```

Error Description

The following error was attached to the three highlighted lines:

```
Class checkers.CheckersGame has a circular dependency with other classes
CheckersGame.java   Group10 Game Framework/src/checkers     line 1
FindBugs Problem (Normal Priority)
```

The FindBugs error indicates that there is a circular dependency between CheckersGame, CheckersAIPlayer, and AiBoardConverter. In most circumstances, circular dependencies should be avoided since it introduces unnecessary dependencies in the code. However, eliminating circular dependencies requires significant refactoring of the code as it would in this case, and sometimes is not possible. We decided that since this does not introduce any runtime defect, it would be preferable to leave the code with the circular dependency.

Associated Code

```
public abstract class Game {
    ...
    public void restoreState(String filename) {
        ...
            try {
                ...
            } catch (Exception e) {
                ...
            }
        }
        ...
    }
    ...
}
```

Error Details

```
Exception is caught when Exception is not thrown
Game.java   Group10 Game Framework/src/com/boredgames/group10      line 515
FindBugs Problem (Low Priority)
```

According to the error, there should not be a catch block for Exception since no method called within the try block explicitly throws the Exception type. This may indicate in a defect in the code since programmers often inadvertently catch runtime exceptions with this type of catch block. While we could change the code to catch only exceptions that are explicitly thrown by the called methods, we felt that it would not be incorrect to leave the code as is.

False Positives

False positives were cases where changing the code to eliminate errors reported by the tool would be incorrect. The tool reported three false positives which were different instances of the same error. Both of the false positives reported by the tool were in the unit tests for our application. These false positives may be valid errors in production code, but were invalid in the context of test code.

Associated Code

The three errors occurred on the highlighted lines in the following code.

```
public class RobustnessTest {
    public Game createInitializationBoardWithPlayers() throws Exception {

        Game testCheckers = null;
        CheckersGameFactory testFactory = new CheckersGameFactory();
        testCheckers = testFactory.createGame();

        assertTrue(testCheckers != null);

        Player testWhite = testCheckers.createPlayer("testWhite", Color.white);
        Player testBlack = testCheckers.createPlayer("testBlack", Color.black);

        assertTrue(testWhite != null);
        assertTrue(testBlack != null);

        testCheckers.addPlayer(testWhite);
        testCheckers.addPlayer(testBlack);

        testCheckers.initialize();
```

```
        return testCheckers;
    }
}

public class CheckersGameFactory extends GameFactory {
      public Game createGame() {
            return new CheckersGame();
    }
}
```

Error Description

The following description was attached to the first of the three errors (with similar descriptions for the other two errors:

```
Redundant nullcheck of testCheckers, which is known to be non-null
RobustnessTest.java Group10 Game Framework/src/tests line 47
FindBugs Problem (Low Priority)
```

Findbugs was smart enough to know that the createGame method always returns a reference to a valid object. As indicated by the error, knowing that createGame always returns a valid object, there is no need to test if the return value is null.

However, this code is in a unit test which should not assume anything about the code being tested. Thus, the test code cannot assume that the return value is never null and should test that the return value is not null as done above. Eliminating the FindBugs error by changing the code so that it does not test whether the return value is null would be incorrect.

Lessons Learned

In general, using FindBugs was a very good learning experience. The tool itself was very easy to use and helpful. While using FindBugs we realized that static analysis is a very useful method for finding defects in code. Trying to find these defects manually or by code inspection would have taken a lot longer than the automated checking that FindBugs provided. We will continue to use FindBugs in the future for our studio project.

Benefits

There were a number of things that made using FindBugs easy and beneficial:

- FindBugs provided good suggestions when a bug was examined in the Eclipse IDE. It provided detailed descriptions of why a certain class of bug was harmful.
- We encountered a very low number of false positives. So the tool is generally reliable and finds valid bugs.
- It provided categories of bugs, which we found to be very useful. We found the option of turning off scanning for certain types of bugs to be a useful feature which would increase the convenience of using FindBugs in different situations.

- From our experience we think that FindBugs is a good way to learn good coding practices for Java, especially for the novice software engineer. It helps users find common pieces of bad code and avoid them in the future.

Usability

From a usability perspective, the tool was very easy to use. We used the Eclipse plug-in for this evaluation, although command line and GUI versions were also available. There is even a Java WebStart version of the tool available online.

Performance

We found the performance of FindBugs to be very good. It was able to analyze 2115 lines of code (not counting comments) in around 9 seconds for the first time. Subsequent scans were considerably shorter.

Drawbacks

Here are some areas for improvement that we noticed while using FindBugs:

- The Eclipse plug-in version of FindBugs can make better use of the quick fix feature by providing automatic fixes for common bugs that it detects. This would increase the convenience of using it in Eclipse.

- One feature that we found lacking was the ability to suppress a specific instance of a bug found by FindBugs. Such a feature would be helpful in cases where it found false positives or irrelevant positives.

- FindBugs has its own defined coding style (naming conventions of methods etc.). There should be a way to customize the coding style so that it can analyze the code against a custom coding style specified by the user. It will not always be the case that the user will be coding in the style defined in FindBugs.

- It was hard to share FindBugs configuration across teams. We had to manually make sure our configurations matched. Even then, while running on the same codebase with the same configuration, we noticed differences in the number of high, normal, and low priority bugs that FindBugs was reporting between team members.

Scope of applicability

The fact that FindBugs supports only Java at this time limits its uses to Java based applications. However, since FindBugs supports detection of various categories of bugs, it can be used in different settings. For instance, it can be used to detect performance bugs in embedded applications. It can be used to detect concurrency bugs in an application that has complex multithreading. The ability to select which bugs to scan for makes FindBugs a versatile tool which can be used on various types of codebases.

Works Cited

FindBugs. April 2009 <http://findbugs.sourceforge.net/>