Tool Practicum - FindBugs

Gunsik Choi, Wonjae Lee, Hyunho Kim Tool Practicum Due: Tue Mar 25, 2008

1. Overview

1.1. Tool Name

FindBugs Ver.1.3.2

1.2. Members

Our team consists of 3 members: Gunsik Choi, Hyunho Kim, Wonjae Lee. The configuration of our team corresponds to that of U-Challenger studio team.

Name	Contact
Gunsik Choi	gchoi@andrew.cmu.edu
Hyunho Kim	hyunhok@andrew.cmu.edu
Wonjae Lee	wonjael@andrew.cmu.edu

2. Project Description

It is hard to achieve quality in software because of many reasons. One important reason is that unexpected bugs can occur. Complex library APIs and various language features provide many causes of bugs in software. However, bugs can waste development time and make users frustrated.

We can use tools to find potential bugs in code. Static analysis, one of analysis techniques, is tool-supported direct static evaluation of formal software artifacts. Using static analysis, we can predict potential errors early without running codes.

In this project, we will test “FindBugs” which is a Java static analysis tool, to find bugs in the codes of the two different projects, and figure out what kinds of bugs this tool finds.

2.1. Objectives of Project

The primary objectives of this project are to gain an in-depth practical experience with an analysis tool or technique and reflect on the experience.

All of our members work in a same studio project for developing collaborative filtering system. Because the most portion of the system will be developed using Java, we expect that we will use this Java static analysis tool to achieve higher quality based on experience of this project.

3. Tool Description

3.1. Overview of FindBugs

FindBugs is a Java static analysis tool which can find code instances of bug patterns in Java code. Bugs can be categorized into certain patterns according to common characteristics, and FindBugs automatically detects code instances of bug patterns as the result of matching the Java byte-code with those patterns. This can generate a list of a variety of patterns of potential bugs. Using this tool, we can analyze Java code without running the code.

3.2. How FindBugs Analyze Code

FindBugs analyzes Java class files using BCEL (Apache Byte Code Engineering Library). BCEL provides analysis framework for CFG (Control Flow Graph) construction, generic dataflow analysis, and various specific dataflow analyses.

And, FindBugs analyzes byte-code in three ways: byte-code scanning, scanning with control flow, and dataflow analysis. Firstly, as following byte-codes, it drives the state machine and checks whether the state machine is in the correct state. Secondly, it can scan control flow thorough control flow graphs. Thirdly, through dataflow analysis, it can check dataflow values iteratively.

3.3. Feature

3.3.1. Bug Pattern Detectors

FindBugs detects instances of bug patterns in Java program based on BCEL, an open source bytecode analysis and instrumentation library. The bug patterns broadly falls into following four categories:

- Single-threaded correctness issue
- Thread/synchronization correctness issue
- Performance issue
- Security and vulnerability to malicious untrusted code

The implementation strategies of FindBugs for finding vulnerabilities in Java bytecode can be divided into four categories:

- Class structure and inheritance hierarchy: This strategy looks at the structure of the analyzed classes without looking at the code.
- Linear code scan: This strategy makes a linear scan through the bytecode and drives a state machine. Instead of using complete control flow, heuristics such as identifying the targets of branch instruction are used.
- Control sensitive: This strategy makes uses of a control flow graph, and the patterns are compared to it.
- Dataflow: This strategy uses the control and data flow graphs generated from analyzing a program. For example, the null pointer dereference detectors use this strategy.

3.3.2. Bug Patterns

Based on FindBugs version 1.3.3-dev-2008031, total 292 of bug patterns are registered. The detail bug description is attached in Appendix 1.

Bug category	Bug pattterns
Bad practice	78
Correctness	109
Internationalization	1
Malicious code vulnerability	12
Multithreaded correctness	12
Performance	22
Security	8
Dodgy	50
Summary	292

3.3.3. Custom detectors

FindBugs can be extended and customized to meet a team's unique requirements. It supports to make and add application-specific bug detectors.

3.4. Usage

FindBugs can be run as a standalone application or an Eclipse plug-in. FindBugs as a standalone application provide two user interfaces: a graphical user interface and a command line user interface. FindBugs can also be integrated into a build script for Ant.

We will focus on running FindBugs as an Eclipse plug-in because Eclipse IDE provides various benefits. Eclipse IDE enables developers to execute the FindBugs directly, and analyze the codes without separately running it. Then the basic functions of Eclipse can be used to fix the found bugs.

3.4.1. Installation

3.4.1.1. Standalone Application

Step 1) Download binary distributions that are available in gzipped tar format and zip format.

Step 2) Extract from the archive file to a destination folder.

3.4.1.2. Eclipse Plug-in

Step 1) Start Eclipse and choose Help → Software Updates → Find and Install…

Step 2) Select the Search for new features to install option, and click the Next button.

Step 3) Select New Remote Site

Step 4) As a Name, enter FindBugs update

As a URL, enter http://findbugs.cs.umd.edu/eclipse

And click OK button.

Step 5) Check the checkbox for “FindBugs update” in Sites to include in search and click next button.

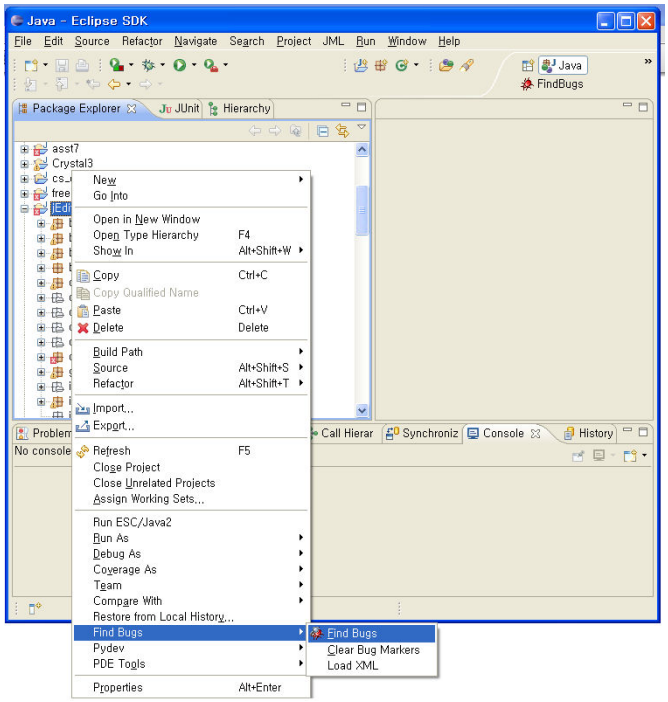
Step 6) Check the checkbox for “FindBugs Feature” in Select features to install and click the next button.

Step 7) Select the **I accept** option and click the next button.

Step 8) Select the location and click the Finish button.

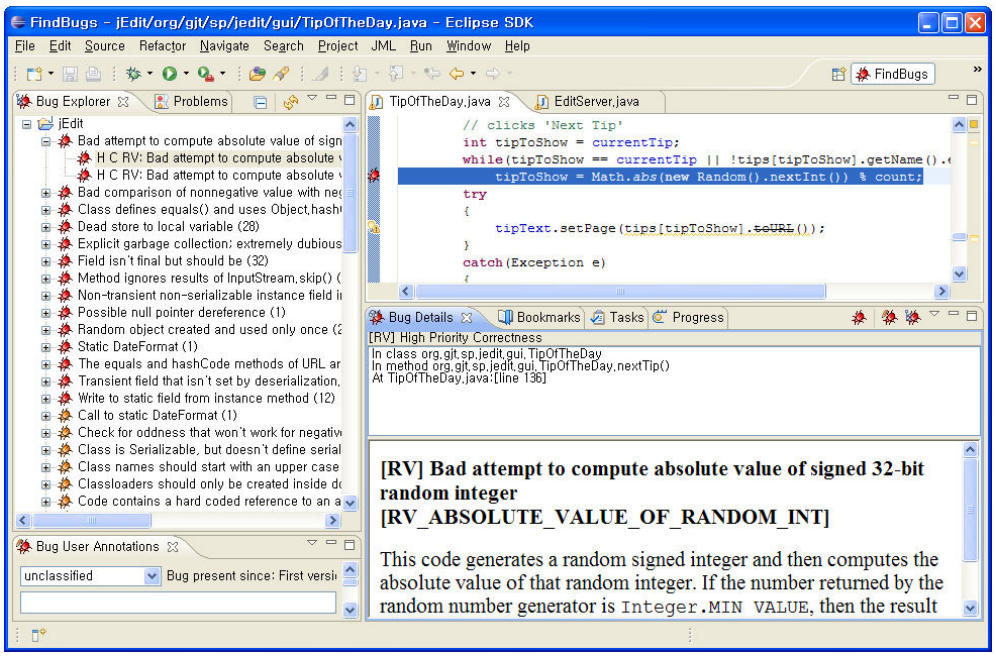
3.4.2. Using the Eclipse Plug-in

To start a bug scanning, click right button on a project and select “Find Bugs” → “Find Bugs” as shown on the next screen shot.



When a bug scanning is completed, found bugs can be explored by selecting the FindBugs perspective as shown on the next screen shot.

In the Bug Explorer, found bugs are classified by their types and importance. A user can assign his own classification to a found bug in the Bug User Annotations.



3.4.3. Customization

FindBugs can be customized by opening the Properties dialog for a Java project, and choosing the “FindBugs” property page as shown on the next screen shot.

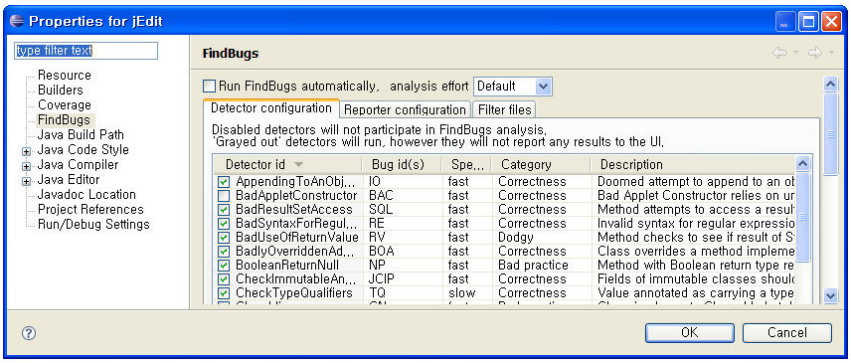
You can choose whether FindBugs runs automatically when a Java class is modified by enabling or disabling the “Run FindBugs Automatically” checkbox.

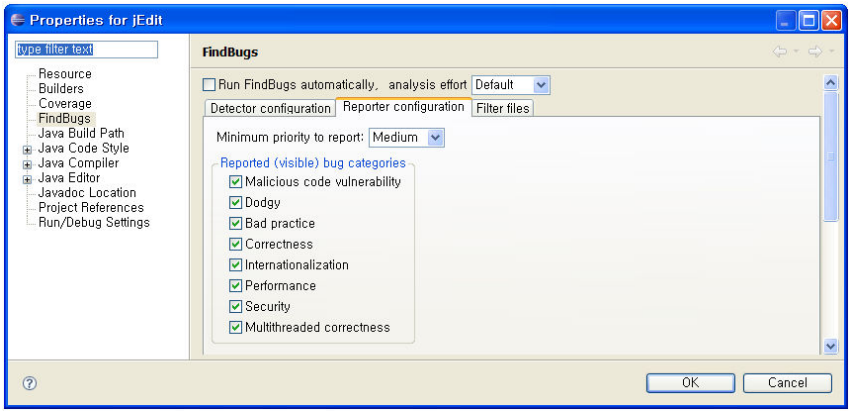
Minimum warning priority and enabled bug categories can be chosen by selecting “analysis effort” pull-down menu.

On the Detector configuration tab, detectors can be enabled or disabled for the project.

On the Reporter configuration tab, minimum priority to report and reported bug categories can be chosen.

On the Filter files tab, filters for filtering bugs can be added.





3.4.4. Documentation

The manual is provided at *http://findbugs.sourceforge.net/manual/index.html*. While most features are documented, explanation about the Eclipse plug-in is less detailed. And more screen shots would have been helpful.

4. Tool Evaluation

4.1. Evaluation Environment Setup

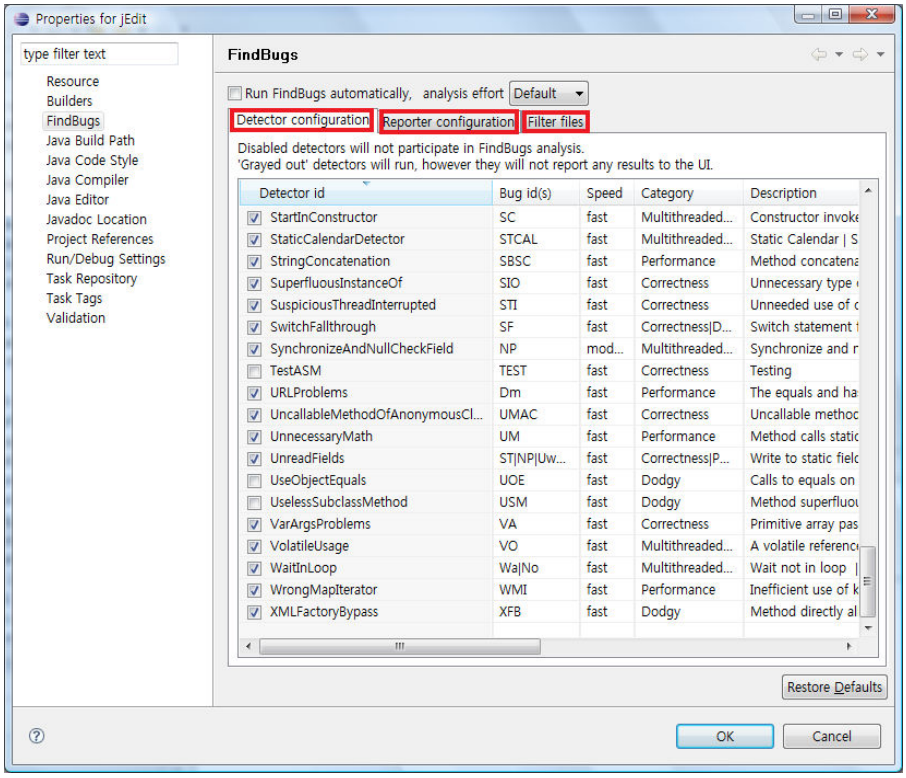
4.1.1. The Experimental Setup

We performed static analysis using FindBugs with JDK/JRE 5.0 version in Windows VISTA operating system. The major specification of hardware of the computer which performed FindBugs is the following:

- CPU: Intel® Core™ 2 Duo CPU T7500 @ 2.20GHz
- Memory: 2046MB

4.1.1. Tool Customization

FindBugs provides properties option to allow user customization. Through the option, a tool user can manipulate detector configuration, reporter configuration, and filter files.



<Figure. Properties for jEdit>

- In detector configuration, we can disable bug detectors that we don't want. The default option doesn't include detectors such as BadAppletConstructor, FindCircularDependencies,InefficientMemberAccess, InfiniteRecursiveLoop2, PublicSemaphores, TestASM, UseObjectEquals, and UselessSubclassMethod.
- In reporter configuration, we can choose minimum priority to report, which has 3 scales: high, medium, and low. And, we can choose bug categories which will be reported. Bug categories that FindBugs support are performance, correctness, internationalization, multithread correctness, bad practice, malicious code vulnerability, dodgy, and security. The default option includes all kinds of bug categories, and set minimum priority to report as medium.
- In filter files, we can include or exclude filter files, and exclude baseline bugs. The default option doesn't specify any list of files.

In this project, we used default configuration of detectors, filter files, and reported bug categories. However, we customized minimum priority to report as high because we don't enough time to figure out all kinds of bugs that have the medium priority.

4.2. Analysis Result: jEdit

4.2.1. Purpose

The purpose of the analysis is to evaluate the effectiveness of FindBugs when applied to mature open source software.

Number of found bugs, categories of found bugs, importance of found bugs, true/false positive rates, time to scan, time to verify, and details of reports were measured.

We could evaluate the effectiveness of FindBugs based on the measures.

4.2.2. Artifacts

jEdit is a mature text editor for programmers. It is open source software under the terms of the GPL 2.0. It provides many features for programming and supports plug-ins.

jEdit uses multi-threaded I/O to improve responsiveness. All buffer input/output operations are executed asynchronously.

jEdit consists of 702 classes with 88712 lines of source code.

4.2.3. Bug Analysis

The analysis time of the program jEdit was 2:40 sec.

- Bug Category

Bug categories that FindBugs detected for jEdit are bad practice, correctness, dodgy, performance, and vulnerability. FindBugs didn't find any bugs related to other categories such as malicious code vulnerability, multithreaded correctness, security, and internationalization.

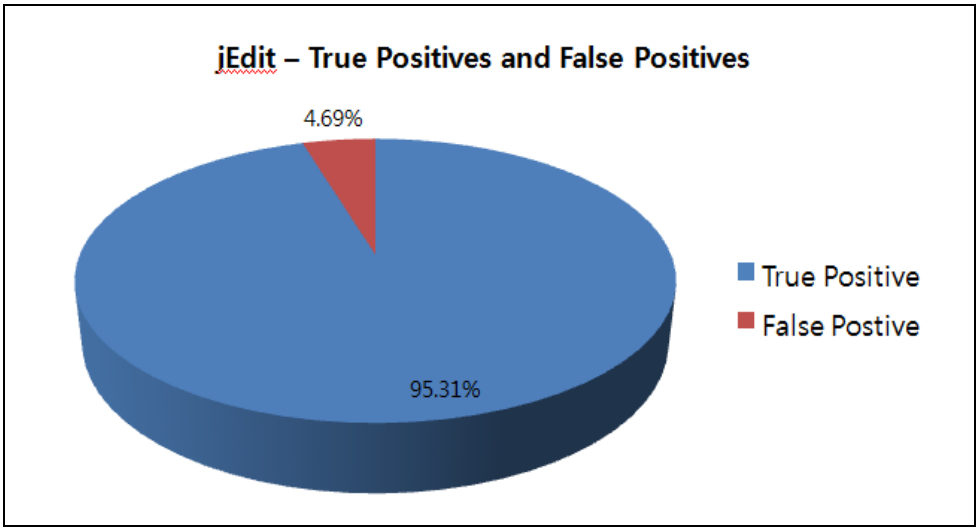
	Name of Category	# of Bugs	Percentage
Bug Category	Bad practice	11	17.2%
	Correctness	7	10.9%
	Dodgy	12	18.8%
	Performance	2	3.1%
	Vulnerability	32	50.0%
	Malicious code vulnerability	0	0.0%
	Multithreaded correctness	0	0.0%
	Security	0	0.0%
	Internationalization	0	0.0%
	Total	64	100.0%

■ # of true positives and false positives that FindBugs detected for jEdit

True positive is a bug that FindBugs reports correctly based on the bug description which FindBugs provides. And, false positive is a bug that FindBugs reports incorrectly based the bug description which FindBugs provides.

The number of bugs that FindBugs detected for jEdit is 64. The number of true positives is 61 and that of false positives is 3. True positives occupy the greater percentage, 95.31%.

Category	# of bugs detected in jEdit
True Positives	61
False Positives	3
# of Total	64



< Percentage of true positives that FindBugs detected for jEdit >

■ # of bug annotations that are assigned to bugs of jEdit

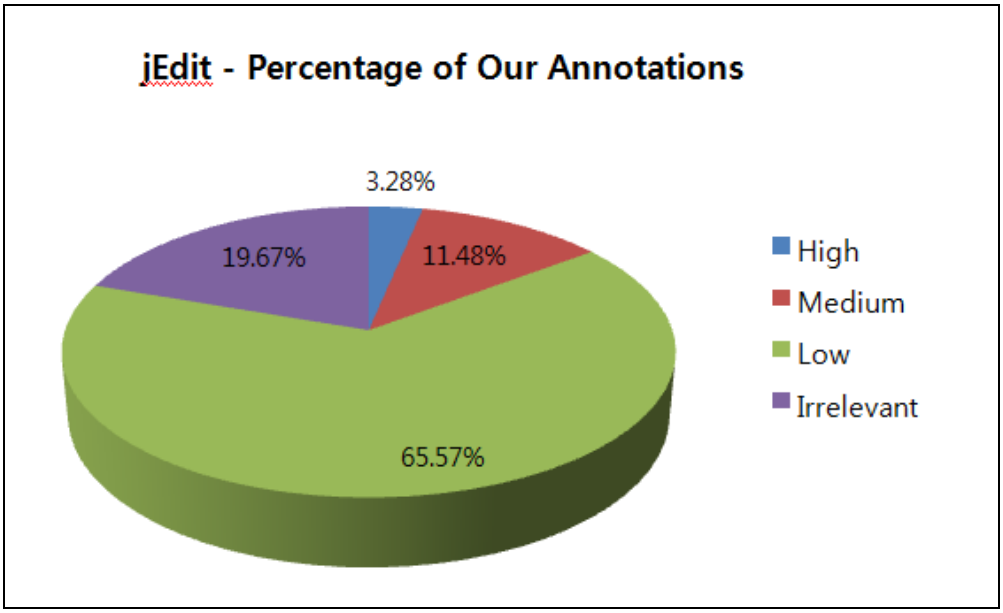
We assigned annotations, which consists of high, medium, low, and irrelevant, to each detect bug. 4 different types of annotation are used to indicate how much a bug affects to system. The meaning of each annotation can be corresponded to priorities given by FindBugs. However, we additionally added the irrelevant annotation to indicate that the found bug doesn't affect to system.

Because all false positives are irrelevant naturally, we show the bug annotations about the true positives in this section. The important thing that we figured out among true positives is that some of true positives are irrelevant to system. Those kinds of true positives are correct based on bug patterns that FindBugs finds. However, we found that they don't affect to authors' intention about system.

The number of bugs annotated as high is 2. The number of bugs annotated as medium is 7. The number of bugs annotated as low is 40. The number of bugs annotated as irrelevant is 12.

Low annotations occupy the greatest percentage, 65.57%. Also, irrelevant annotations occupy the secondly greatest percentage, 11.48%. Because we only considered bugs reported as the high priority from FindBugs, we can see that the result of our annotations is different from priorities that FindBugs gives to bugs. Much of bugs reported as the high priority don't have critical effects to the system based on our judgment.

Annotation	# of bugs
High	2
Medium	7
Low	40
Irrelevant	12
# of Total	61



<Percentage of bug annotations that are assigned to bugs of jEdit>

■ Time to verify

The total amount of time to verify all bugs is 19:02 minutes. And the average time to verify each bug is 1:02 minutes.

The total amount of time to verify irrelevant bugs is 7:29 minutes. And the average time to verify each irrelevant bug is 2:05 minutes.

The total amount of time to verify relevant bugs is 11:03 minutes. And the average time to verify each relevant bug is 0:43 minutes.

	All Bugs	Irrelevant Bugs	Relevant Bugs
Total Time (min:sec)	19:02	7:29	11:33
Average (min:sec)	1:02	2:05	0:43

The descriptions of bugs were specific and detailed. A priority, category, location, problematic portion, and explanation of why it is considered as a bug were described. For example, a high priority bug of dodgy category was found in org.gjt.sp.jedit.syntax.DisplayTokenHandler.java file. FindBugs reports that "`H D DLS: Dead store to wrapMargin in org.gjt.sp.jedit.syntax.DisplayTokenHandler.init(SyntaxStyle[], FontRenderContext, TabExpander, List, float)`" is happened. And explanation about the "Dead store to local variable" type bug is "`This instruction assigns a value to a local variable, but the value is not read by any subsequent instruction. Often, this indicates an error, because the value computed is never used. Note that Sun's javac compiler often generates dead stores for final local variables. Because FindBugs is a bytecode-based tool, there is no easy way to eliminate these false positives.`"

4.3. Analysis Result: FreeMind

4.3.1. Purpose

The purpose of this analysis is to evaluate the tool Findbugs. The evaluation will be performed by measuring the tool performance, analysis effectiveness, and usability. The tool performance measures the time to scan and the size of code. The tool effectiveness for analysis measures the number of found bugs, bug categories, importance of found bugs, true/false positive rates and the time to verify the found bugs. The usability measures the easiness of the tool qualitatively.

4.3.2. Artifacts

The FreeMind is an open source application for mind-mapping written in Java. Although the latest stable release verion is 0.8.1, the development community is much active. The total source size is 4.41MB, the number of classes are 427, and the source line is 83,147 (including braces).

4.3.3. Bug Analysis

The analysis time of the program FreeMind in FindBugs was 1:58 sec.

The analysis result shows that the FindBugs found 291 bugs, but our team decided to verify only the defects of "high" importance because of the available time and human resource constraints. The following analysis reports are based on the bugs of "high" importance in the FindBugs.

- Bug Category

The total number of "high" important bugs found in the FreeMind program was 13. The percentage of bug category shows the similar distribution to the result of jEdit program, in which the vulnerability bugs were almost half number and the bugs of dodgy, bad practice and other parts were another half number.

	Name of Category	# of Bugs	Percentage
Bug Category	Bad practice	2	15.4%
	Correctness	1	7.7%
	Dodgy	4	30.8%
	Performance	0	
	Vulnerability	6	46.1%
	Malicious code vulnerability	0	0.0%
	Multithreaded correctness	0	0.0%
	Security	0	0.0%
	Internationalization	0	0.0%

■ # of true positives and false positives that FindBugs detected for FreeMind

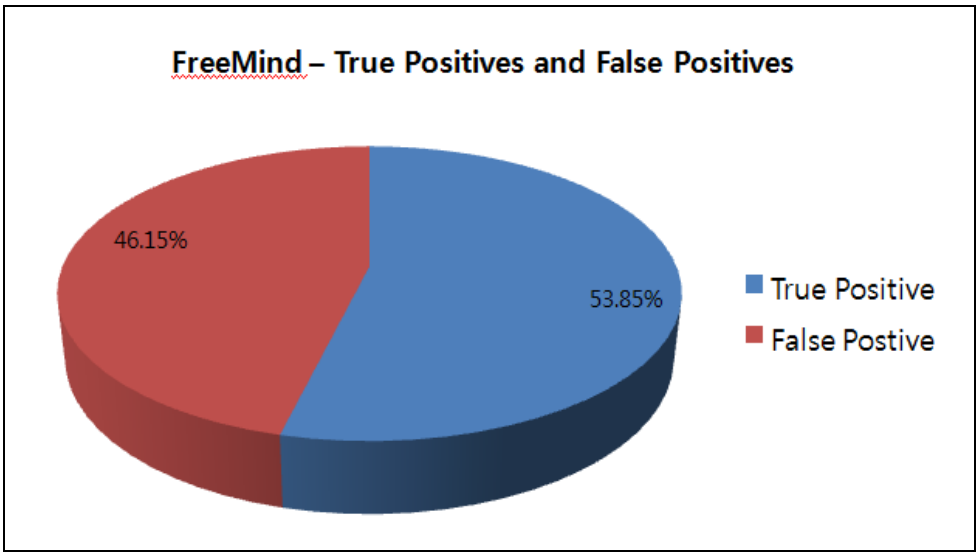
The below table shows that the false positives of the FindBugs is high compared to false positive, but the evaluation for the result needs more consideration for this figures. The false positives consist of 6 bugs as follows:

- 5 bugs: `[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]`
- 1 bug: `[DE] Method might ignore exception [DE_MIGHT_IGNORE]`

The first 5 bugs show that the static field should be defined with “final” keyword, but the source code uses the static variables and changes them. So we evaluated them as false positive. The other one is the bug which is wrongly detected by FindBugs. The source code shows the exception handling should be included in the outer block, but the FindBugs interpreted that the exception handling isn’t needed in the outer bock.

This case shows that the customization of the tool is required for reducing the false positive.

Category	# of bugs detected in jEdit
True Positives	7
False Positives	6
# of Total	13



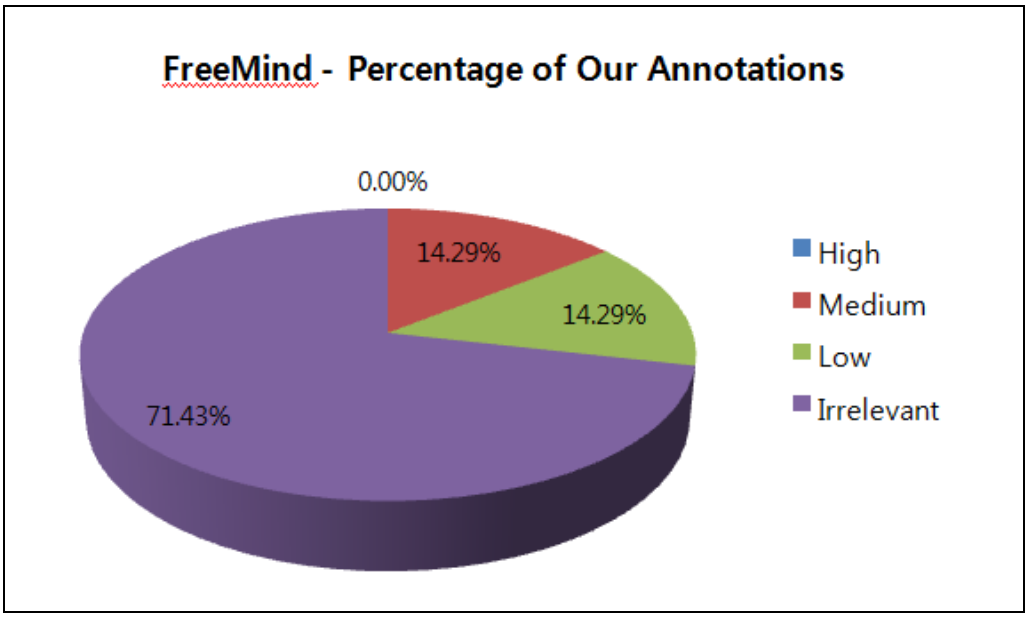
< Percentage of true positives that FindBugs detected for FreeMind >

■ # of bug annotations that are assigned to bugs of FreeMind

The following table shows the distribution of the importance of bugs in the true positives by our team. We evaluated each bugs and most bugs were irrelevant. The medium and low bugs we evaluated were as follows:

- [ES] Comparison of String objects using == or != [ES_COMPARING_STRINGS_WITH_EQ]
- [DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]

Annotation	# of bugs
High	0
Medium	1
Low	1
Irrelevant	5
# of Total	7



<Percentage of bug annotations that are assigned to bugs of FreeMind>

- Time to verify

The following table shows that the time to verify the relevant bugs was lower than the irrelevant bugs. The main reason was that the relevant bugs can be easily conceived using the detail description of the tool, but the irrelevant bugs need to verify the counter example and case for the description in the tool.

	All Bugs	Irrelevant Bugs	Relevant Bugs
Total Time (min:sec)	17:28	12:00	5:28
Average (min:sec)	1:20	1:05	2:44

4.4. Concrete Examples

4.4.1. Example of True positive That Is Reverent

One example of true positive that is relevant to the project is the following.

FindBugs reported "`H C NP: Possible null pointer dereference of point in org.gjt.sp.jedit.textarea. JEditTextArea.scrollTo(int, int, boolean)`" in jEdit code. The relevant code is the following.

```
if(point == null)
{
	Log.log(Log.ERROR,this,"BUG: screenLine=" + screenLine
		+ ",visibleLines=" + visibleLines
		+ ",physicalLine=" + line
		+ ",firstPhysicalLine=" + getFirstPhysicalLine()
		+ ",lastPhysicalLine=" + getLastPhysicalLine());
}
```

```
point.x += extraEndVirt;
```

The explanation by FindBugs is "[NP] Possible null pointer dereference [NP_NULL_ON_SOME_PATH]

There is a branch of statement that, if executed, guarantees that a null value will be dereferenced, which would generate a NullPointerException when the code is executed. Of course, the problem might be that the branch or statement is infeasible and that the null pointer exception can't ever be executed; deciding that is beyond the ability of FindBugs."

In this case, when the point variable is a null pointer, the problem is logged. However, it does not return after the logging, and this can cause a possible null pointer dereference at the last line. The code can be fixed by adding a return statement after logging.

Another example of true positive that is relevant to the project is the following.

FindBugs reported "H B BC: Random object created and used only once in new org.gjt.sp.jedit.EditServer(String)" in jEdit code. The relevant code is the following.

```
// Bind to any port on localhost; accept 2 simultaneous
// connection attempts before rejecting connections
socket = new ServerSocket(0, 2,
InetAddress.getByName("127.0.0.1"));
authKey = Math.abs(new Random().nextInt());
int port = socket.getLocalPort();
```

The explanation by FindBugs is "[BC] Random object created and used only once [DMI_RANDOM_USED_ONLY_ONCE]

This code creates a java.util.Random object, uses it to generate one random number, and then discards the Random object. This produces mediocre quality random numbers and is inefficient. If possible, rewrite the code so that the Random object is created once and saved, and each time a new random number is required invoke a method on the existing Random object to obtain it.

If it is important that the generated Random numbers not be guessable, you must not create a new Random for each random number; the values are too easily guessable. You should strongly consider using a java.security.SecureRandom instead (and avoid allocating a new SecureRandom for each random number needed)."

In this particular case, the issue is not just about inefficiency. Because the value of `authKey` variable can be easily guessable, it can be a security hole during authentication. A java.security.SecureRandom should have been used to generate a value for the `authKey`.

4.4.2. Example of True Positive That Is Irrelevant

One example of true positive that is not relevant to the project is the following.

FindBugs reported "`H D DMI: Hard coded reference to an absolute pathname in accessories.plugins.util.xslt.ExportDialog.main(String[])`" in freeMind code. The relevant code is the following.

```
public static void main(String[] args) {
        Properties sysprops   = System.getProperties();
        Enumeration propnames = sysprops.propertyNames();
        while (propnames.hasMoreElements()) {
            String propname = (String)propnames.nextElement();
            System.out.println(
            propname + "=" + System.getProperty(propname)
            );
        };

        ExportDialog wnd = new ExportDialog(new
File("/home/testtrans.xml"));
        wnd.setVisible(true);
    }
```

The explanation by FindBugs is "`[DMI] Code contains a hard coded reference to an absolute pathname [DMI_HARDCODED_ABSOLUTE_FILENAME]`

`This code constructs a File object using a hard coded to an absolute pathname (e.g., new File("/home/dannyc/workspace/j2ee/src/share/com/sun/enterprise/deployment");`".

While it is recommended that a hard coded reference should not be used, a hard coded reference is used for testing purposes in the above code. Thus, while it is a technically correct warning, it is not relevant to the freeMind project.

Another example of true positive that is not relevant to the project is the following.

FindBugs reported "`H D ST: Write to static field org.gjt.sp.jedit.gui.HistoryModel.modified from instance method org.gjt.sp.jedit.gui.HistoryModel.addItem(String)`" in jEdit code. The relevant code is the following.

```
public void addItem(String text)
{
      if(text == null || text.length() == 0)
            return;
            modified = true;
```

```
            int index = data.indexOf(text);
      if(index != -1)
            data.removeElementAt(index);
            data.insertElementAt(text,0);
            while(getSize() > max)
            data.removeElementAt(data.size() - 1);
} //}}}
```

The explanation by FindBugs is "[ST] Write to static field from instance method [ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD]

This instance method writes to a static field. This is tricky to get correct if multiple instances are being manipulated, and generally bad practice."

While writing to a static field by an instance method can be problematic, it is intentional in this case. The static modified field should be set to true when an item is added, and there is no problem even when there are multiple instances. Thus, while it is a technically correct warning, it is not relevant to the jEdit project.

4.4.3. Example of False positive

One example of false positive is the following.

FindBugs reported "H C INT: Bad comparison of nonnegative value with -1 in installer. CBZip2InputStream.getAndMoveToFrontDecode()" in jEdit code. The relevant code is the following.

```
char thech = 0;
try
{
    thech = (char)m_input.read();
}
catch( IOException e )
{
    compressedStreamEOF();
}
if( thech == -1 )
{
    compressedStreamEOF();
}
```

The explanation by FindBugs is "[INT] Bad comparison of nonnegative value with negative constant [INT_BAD_COMPARISON_WITH_NONNEGATIVE_VALUE]
This code compares a value that is guaranteed to be non-negative with a negative constant."

The read method in the above code can return -1 when the end of the stream is reached. Thus thech can be -1, and the warning is a false positive.

Another example of false positive is the following.

FindBugs reported "`H V MS: freemind.controller.MindMapNodesSelection.mindMapNodesFlavor isn't final but should be`" in freeMind code. The relevant code is the following.

```
public static DataFlavor mindMapNodesFlavor = null;
```

The explanation by FindBugs is "`[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]`
`A mutable static field could be changed by malicious code or by accident from another package. The field could be made final to avoid this vulnerability.`"

In this case, mindMapNodesFlavor cannot be final because its value is changed later. Thus, the warning is a false positive.

5. Lessons learned

We spent much time on verifying false positives or irrelevant bugs. If we could customize options suited to project characteristics, the productivity would be better.

5.1. Benefits

FindBugs found non-trivial bugs in two fairly mature open source programs. We expect FindBugs would find more valuable bugs during development. Finding a bug in early stages of the process can reduce considerable reworks.

FindBugs found insidious kinds of bugs. Detection of these bugs can considerably improve productivity.

FindBugs has a vast array of detectable bugs. It can be applied to various domains.

FindBugs is highly customizable. Each detector can be turned on or off individually. Minimum priority to report can be selected. Customization can improve productivity.

Detailed explanations about bugs were helpful in verification.

Integration with Eclipse provides various productivity advantages. When utilizing a feature that FindBugs runs automatically on modified Java codes, bug scanning time can be practically ignored.

Programmers can learn many good practices of Java by verifying found bugs. This would improve overall quality of code.

5.2. Drawbacks

Eclipse plug-in version of FindBugs lacks some necessary features. Exporting the list of found bugs is not supported in Eclipse plug-in. Documentation for Eclipse plug-in version needs improvement.

Reported priorities of bugs can be somewhat inconsistent because priorities are determined by each bug detector. While a user can use bug filters, this inconsistency can make customization difficult.

Appendix1. Bug patterns that FindBugs detected for jEdit

priority	category	description	true/ false positive	importance	time to verify	detail of reports
high	correctness	[RV] Bad attempt to compute absolute value of signed 32-bit random integer [RV_ABSOLUTE_VALUE_OF_RANDOM_INT]	TRUE	med	4:18	high
high	correctness	[RV] Bad attempt to compute absolute value of signed 32-bit random integer [RV_ABSOLUTE_VALUE_OF_RANDOM_INT]	TRUE	med	0:05	high
high	correctness	[INT] Bad comparison of nonnegative value with negative constant	FALSE	irrelevant	2:13	high
high	correctness	[INT] Bad comparison of nonnegative value with negative constant	FALSE	irrelevant	2:13	high
high	correctness	[INT] Bad comparison of nonnegative value with negative constant	FALSE	irrelevant	0:05	high
high	bad practice	[HE] Class defines equals() and uses Object.hashCode()	TRUE	med	5:15	high
high	dodgy	[DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]	TRUE	low	2:48	high
high	dodgy	[DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]	TRUE	low	0:53	high
high	dodgy	[DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]	TRUE	low	0:05	high
high	dodgy	[DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]	TRUE	low	0:05	high
high	dodgy	[DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]	TRUE	low	0:05	high
high	dodgy	[DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]	TRUE	low	0:05	high
high	dodgy	[DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]	TRUE	low	2:58	high
high	dodgy	[DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]	TRUE	low	0:05	high
high	performance	[Dm] Explicit garbage collection; extremely dubious except in benchmarking code [DM_GC]	TRUE	irrelevant	1:13	high
high	vulerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	TRUE	low	0:52	high
high	vulerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit	[MS] Field isn't final but should be [MS_S	TRUE	low	0:05	high

	y	HOULD_BE_FINAL]				
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	vulerabilit y	[MS] Field isn't final but should be [MS_S HOULD_BE_FINAL]	TRUE	low	0:05	high
high	bad pract ice	[RR] Method ignores results of InputStre am.skip() [SR_NOT_CHECKED]	TRUE	irrelevan t	5:17	high
high	bad pract	[Se] Non-transient non-serializable insta	TRUE	med	3:12	high

	ice	nce field in serializable class [SE_BAD_FIELD]				
high	bad practice	[Se] Non-transient non-serializable instance field in serializable class [SE_BAD_FIELD]	TRUE	med	0:05	high
high	bad practice	[Se] Non-transient non-serializable instance field in serializable class [SE_BAD_FIELD]	TRUE	med	0:05	high
high	correctness	[NP] Possible null pointer dereference [NP_NULL_ON_SOME_PATH]	TRUE	high	2:10	high
high	bad practice	[BC] Random object created and used only once [DMI_RANDOM_USED_ONLY_ONCE]	TRUE	irrelevant	1:40	high
high	bad practice	[BC] Random object created and used only once [DMI_RANDOM_USED_ONLY_ONCE]	TRUE	high	7:00	high
high	correctness	[STCAL] Static DateFormat [STCAL_STATIC_SIMPLE_DATE_FORMAT_INSTANCE]	TRUE	med	2:52	high
high	performance	[Dm] The equals and hashCode methods of URL are blocking [DMI_BLOCKING_METHODS_ON_URL]	TRUE	irrelevant	1:52	high
high	bad practice	[Se] Transient field that isn't set by deserialization. [SE_TRANSIENT_FIELD_NOT_RESTORED]	TRUE	irrelevant	6:09	high
high	bad practice	[Se] Transient field that isn't set by deserialization. [SE_TRANSIENT_FIELD_NOT_RESTORED]	TRUE	irrelevant	0:05	high
high	bad practice	[Se] Transient field that isn't set by deserialization. [SE_TRANSIENT_FIELD_NOT_RESTORED]	TRUE	irrelevant	0:05	high
high	bad practice	[Se] Transient field that isn't set by deserialization. [SE_TRANSIENT_FIELD_NOT_RESTORED]	TRUE	irrelevant	0:05	high
high	dodgy	[ST] Write to static field from instance method [ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD]	TRUE	irrelevant	4:46	high
high	dodgy	[ST] Write to static field from instance method [ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD]	TRUE	irrelevant	4:46	high
high	dodgy	[ST] Write to static field from instance method [ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD]	TRUE	irrelevant	0:30	high
high	dodgy	[ST] Write to static field from instance method [ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD]	TRUE	irrelevant	0:30	high

Appendix2. Bug patterns that FindBugs detected for FreeMind

priority	category	description	true/false positive	importance	time to verify	detail of reports
high	bad practice	[ES] Comparison of String objects using == or != [ES_COMPARING_STRINGS_WITH_EQ]	TRUE	med	3:12	high
high	bad practice	[DE] Method might ignore exception [DE_MIGHT_IGNORE]	FALSE	irrelevant	1:59	high
high	correctness	[LI] Incorrect lazy initialization and update of static field [LI_LAZY_INIT_UPDATE_STATIC]	TRUE	irrelevant	2:00	high
high	dodgy	[DMI] Code contains a hard coded reference to an absolute pathname [DMI_HARDCODED_ABSOLUTE_FILENAME]	TRUE	irrelevant	2:10	high
high	dodgy	[DLS] Dead store to local variable [DLS_DEAD_LOCAL_STORE]	TRUE	low	2:16	high
high	dodgy	[REC] Exception is caught when Exception is not thrown [REC_CATCH_EXCEPTION]	TRUE	irrelevant	1:56	high
high	dodgy	[ST] Write to static field from instance method [ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD]	TRUE	irrelevant	2:30	high
high	vulnerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	TRUE	irrelevant	1:00	high
high	vulnerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	FALSE	irrelevant	0:05	high
high	vulnerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	FALSE	irrelevant	0:05	high
high	vulnerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	FALSE	irrelevant	0:05	high
high	vulnerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	FALSE	irrelevant	0:05	high
high	vulnerability	[MS] Field isn't final but should be [MS_SHOULD_BE_FINAL]	FALSE	irrelevant	0:05	high

Appendix3. Complete Bug Patterns in FindBugs

Code	Description	Category
AM	Creates an empty jar file entry	Bad practice
AM	Creates an empty zip file entry	Bad practice
BC	Equals method should not assume anything about the type of its argument	Bad practice
BC	Random object created and used only once	Bad practice
BIT	Check for sign of bitwise operation	Bad practice
CN	Class implements Cloneable but does not define or use clone method	Bad practice
CN	clone method does not call super.clone()	Bad practice
CN	Class defines clone() but doesn't implement Cloneable	Bad practice
Co	Abstract class defines covariant compareTo() method	Bad practice
Co	Covariant compareTo() method defined	Bad practice
DE	Method might drop exception	Bad practice
DE	Method might ignore exception	Bad practice
DP	Classloaders should only be created inside doPrivileged block	Bad practice
DP	Method invoked that should be only be invoked inside a doPrivileged block	Bad practice
Dm	Method invokes System.exit(...)	Bad practice
Dm	Method invokes dangerous method runFinalizersOnExit	Bad practice
ES	Comparison of String parameter using == or !=	Bad practice
ES	Comparison of String objects using == or !=	Bad practice
Eq	Abstract class defines covariant equals() method	Bad practice
Eq	Class defines compareTo(...) and uses Object.equals()	Bad practice
Eq	Covariant equals() method defined	Bad practice
FI	Empty finalizer should be deleted	Bad practice
FI	Explicit invocation of finalizer	Bad practice
FI	Finalizer nulls fields	Bad practice
FI	Finalizer only nulls fields	Bad practice
FI	Finalizer does not call superclass finalizer	Bad practice
FI	Finalizer nullifies superclass finalizer	Bad practice
FI	Finalizer does nothing but call superclass finalizer	Bad practice
HE	Class defines equals() but not hashCode()	Bad practice
HE	Class defines equals() and uses Object.hashCode()	Bad practice
HE	Class defines hashCode() but not equals()	Bad practice
HE	Class defines hashCode() and uses Object.equals()	Bad practice
HE	Class inherits equals() and uses Object.hashCode()	Bad practice
IC	Superclass uses subclass during initialization	Bad practice
IMSE	Dubious catching of IllegalMonitorStateException	Bad practice
ISC	Needless instantiation of class that only supplies static methods	Bad practice
It	Iterator next() method can't throw NoSuchElement exception	Bad practice
J2EE	Store of non serializable object into HttpSession	Bad practice
NP	Method with Boolean return type returns explicit null	Bad practice
NP	Clone method may return null	Bad practice
NP	equals() method does not check for null argument	Bad practice
NP	toString method may return null	Bad practice

Nm	Class names should start with an upper case letter	Bad practice
Nm	Class is not derived from an Exception, even though it is named as such	Bad practice
Nm	Confusing method names	Bad practice
Nm	Field names should start with a lower case letter	Bad practice
Nm	Use of identifier that is a keyword in later versions of Java	Bad practice
Nm	Use of identifier that is a keyword in later versions of Java	Bad practice
Nm	Method names should start with a lower case letter	Bad practice
Nm	Class names shouldn't shadow simple name of implemented interface	Bad practice
Nm	Class names shouldn't shadow simple name of superclass	Bad practice
Nm	Very confusing method names	Bad practice
Nm	Method doesn't override method in superclass due to wrong package for parameter	Bad practice
ODR	Method may fail to close database resource	Bad practice
ODR	Method may fail to close database resource on exception	Bad practice
OS	Method may fail to close stream	Bad practice
OS	Method may fail to close stream on exception	Bad practice
RC	Suspicious reference comparison	Bad practice
RR	Method ignores results of InputStream.read()	Bad practice
RR	Method ignores results of InputStream.skip()	Bad practice
RV	Method ignores exceptional return value	Bad practice
SI	Static initializer creates instance before all static final fields assigned	Bad practice
SW	Certain swing methods needs to be invoked in Swing thread	Bad practice
Se	Non-transient non-serializable instance field in serializable class	Bad practice
Se	Non-serializable class has a serializable inner class	Bad practice
Se	Non-serializable value stored into instance field of a serializable class	Bad practice
Se	Comparator doesn't implement Serializable	Bad practice
Se	Serializable inner class	Bad practice
Se	Method must be private in order for serialization to work	Bad practice
Se	serialVersionUID isn't final	Bad practice
Se	serialVersionUID isn't long	Bad practice
Se	serialVersionUID isn't static	Bad practice
Se	Class is Serializable but its superclass doesn't define a void constructor	Bad practice
Se	Class is Externalizable but doesn't define a void constructor	Bad practice
Se	The readResolve method must be declared with a return type of Object.	Bad practice
Se	Transient field that isn't set by deserialization.	Bad practice
SnVI	Class is Serializable, but doesn't define serialVersionUID	Bad practice
UI	Usage of GetResource may be unsafe if class is extended	Bad practice
BC	Impossible cast	Correctness
BC	instanceof will always return false	Correctness
BIT	Incompatible bit masks	Correctness
BIT	Check to see if ((...) & 0) == 0	Correctness
BIT	Incompatible bit masks	Correctness
BIT	Bitwise OR of signed byte value	Correctness
BIT	Check for sign of bitwise operation	Correctness
BOA	Class overrides a method implemented in super class Adapter wrongly	Correctness

Bx	Primitive value is unboxed and coerced for ternary operator	Correctness
DLS	Useless assignment in return statement	Correctness
DLS	Dead store of class literal	Correctness
DLS	Overwritten increment	Correctness
DMI	Bad constant value for month	Correctness
DMI	hasNext method invokes next	Correctness
DMI	Invocation of toString on an array	Correctness
DMI	Invocation of toString on an array	Correctness
DMI	Double.longBitsToDouble invoked on an int	Correctness
Dm	Can't use reflection to check for presence of annotation with default retention	Correctness
EC	equals() used to compare array and nonarray	Correctness
EC	Invocation of equals() on an array, which is equivalent to ==	Correctness
EC	Call to equals() with null argument	Correctness
EC	Call to equals() comparing unrelated class and interface	Correctness
EC	Call to equals() comparing different interface types	Correctness
EC	Call to equals() comparing different types	Correctness
EC	Using pointer equality to compare different types	Correctness
Eq	Covariant equals() method defined for enum	Correctness
Eq	equals() method defined that doesn't override equals(Object)	Correctness
Eq	equals() method defined that doesn't override Object.equals(Object)	Correctness
Eq	equals method overrides equals in superclass and may not be symmetric	Correctness
Eq	Covariant equals() method defined, Object.equals(Object) inherited	Correctness
FE	Doomed test for equality to NaN	Correctness
GC	No relationship between generic parameter and method argument	Correctness
HE	Use of class without a hashCode() method in a hashed data structure	Correctness
ICAST	Integer shift by an amount not in the range 0..31	Correctness
ICAST	int value cast to double and then passed to Math.ceil	Correctness
ICAST	int value cast to float and then passed to Math.round	Correctness
IJU	JUnit assertion in run method will not be noticed by JUnit	Correctness
IJU	TestCase declares a bad suite method	Correctness
IJU	TestCase has no tests	Correctness
IJU	TestCase implements setUp but doesn't call super.setUp()	Correctness
IJU	TestCase implements a non-static suite method	Correctness
IJU	TestCase implements tearDown but doesn't call super.tearDown()	Correctness
IL	A container is added to itself	Correctness
IL	An apparent infinite loop	Correctness
IL	An apparent infinite recursive loop	Correctness
IM	Integer multiply of result of integer remainder	Correctness
INT	Bad comparison of nonnegative value with negative constant	Correctness
INT	Bad comparison of signed byte	Correctness
INT	Integer remainder modulo 1	Correctness
IO	Doomed attempt to append to an object output stream	Correctness
IP	A parameter is dead upon entry to a method but overwritten	Correctness
JCIP	Fields of immutable classes should be final	Correctness

MF	Class defines field that masks a superclass field	Correctness
MF	Method defines a variable that obscures a field	Correctness
NP	Null pointer dereference	Correctness
NP	Null pointer dereference in method on exception path	Correctness
NP	Method does not check for null argument	Correctness
NP	Null value is guaranteed to be dereferenced	Correctness
NP	Value is null and guaranteed to be dereferenced on exception path	Correctness
NP	Method call passes null to a nonnull parameter	Correctness
NP	Method may return null, but is declared @NonNull	Correctness
NP	A known null value is checked to see if it is an instance of a type	Correctness
NP	Possible null pointer dereference	Correctness
NP	Possible null pointer dereference in method on exception path	Correctness
NP	Method call passes null for unconditionally dereferenced parameter	Correctness
NP	Method call passes null for unconditionally dereferenced parameter	Correctness
NP	Non-virtual method call passes null for unconditionally dereferenced parameter	Correctness
NP	Store of null value into field annotated NonNull	Correctness
NP	Read of unwritten field	Correctness
Nm	Class defines equal(); should it be equals()?	Correctness
Nm	Class defines hashcode(); should it be hashCode()?	Correctness
Nm	Class defines tostring(); should it be toString()?	Correctness
Nm	Apparent method/constructor confusion	Correctness
Nm	Very confusing method names	Correctness
Nm	Method doesn't override method in superclass due to wrong package for parameter	Correctness
QBA	Method assigns boolean literal in boolean expression	Correctness
RCN	Nullcheck of value previously dereferenced	Correctness
RE	Invalid syntax for regular expression	Correctness
RE	File.separator used for regular expression	Correctness
RE	. used for regular expression	Correctness
RV	Random value from 0 to 1 is coerced to the integer 0	Correctness
RV	Bad attempt to compute absolute value of signed 32-bit hashcode	Correctness
RV	Bad attempt to compute absolute value of signed 32-bit random integer	Correctness
RV	Exception created and dropped rather than thrown	Correctness
RV	Method ignores return value	Correctness
SA	Double assignment of field	Correctness
SA	Self assignment of field	Correctness
SA	Self comparison of field with itself	Correctness
SA	Nonsensical self computation involving a field (e.g., x & x)	Correctness
SA	Double assignment of local variable	Correctness
SA	Self comparison of value with itself	Correctness
SA	Nonsensical self computation involving a variable (e.g., x & x)	Correctness
SF	Dead store due to switch statement fall through	Correctness
SIO	Unnecessary type check done using instanceof operator	Correctness
SQL	Method attempts to access a prepared statement parameter with index 0	Correctness
SQL	Method attempts to access a result set field with index 0	Correctness

STI	Unneeded use of currentThread() call, to call interrupted()	Correctness
STI	Static Thread.interrupted() method invoked on thread instance	Correctness
TQ	Value annotated as carrying a type qualifier used where a value that must not carry that qualifier is required	Correctness
TQ	Value that might not carry a type qualifier reaches a use requiring that type qualifier	Correctness
TQ	Unknown value reaches a use which forbids values carrying type qualifier annotation	Correctness
TQ	Value annotated as never carrying a type qualifier used where value carrying that qualifier is required	Correctness
UCF	Useless control flow to next line	Correctness
UMAC	Uncallable method defined in anonymous class	Correctness
UR	Uninitialized read of field in constructor	Correctness
UwF	Field only ever set to null	Correctness
UwF	Unwritten field	Correctness
VA	Number of format-string arguments does not correspond to number of placeholders	Correctness
VA	Primitive array passed to function expecting a variable number of object arguments	Correctness
Dm	Consider using Locale parameterized version of invoked method	Internationalization
EI	May expose internal representation by returning reference to mutable object	Malicious code vulnerability
EI2	May expose internal representation by incorporating reference to mutable object	Malicious code vulnerability
FI	Finalizer should be protected, not public	Malicious code vulnerability
MS	May expose internal static state by storing a mutable object into a static field	Malicious code vulnerability
MS	Field isn't final and can't be protected from malicious code	Malicious code vulnerability
MS	Public static method may expose internal representation by returning array	Malicious code vulnerability
MS	Field should be both final and package protected	Malicious code vulnerability
MS	Field is a mutable array	Malicious code vulnerability
MS	Field is a mutable Hashtable	Malicious code vulnerability
MS	Field should be moved out of an interface and made package protected	Malicious code vulnerability
MS	Field should be package protected	Malicious code vulnerability
MS	Field isn't final but should be	Malicious code vulnerability
DC	Possible double check of field	Multithreaded correctness
DL	Synchronization on Boolean could lead to deadlock	Multithreaded correctness
DL	Synchronization on boxed primative could lead to deadlock	Multithreaded correctness
DL	Synchronization on shared constant could lead to deadlock	Multithreaded correctness

DL	Synchronization boxed primative values	Multithreaded correctness
Dm	Monitor wait() called on Condition	Multithreaded correctness
Dm	A thread was created using the default empty run method	Multithreaded correctness
ESync	Empty synchronized block	Multithreaded correctness
IS	Inconsistent synchronization	Multithreaded correctness
IS	Field not guarded against concurrent access	Multithreaded correctness
JLM	Synchronization performed on java.util.concurrent Lock	Multithreaded correctness
LI	Incorrect lazy initialization of static field	Multithreaded correctness
LI	Incorrect lazy initialization and update of static field	Multithreaded correctness
ML	Synchronization on field in futile attempt to guard that field	Multithreaded correctness
ML	Method synchronizes on an updated field	Multithreaded correctness
MWN	Mismatched notify()	Multithreaded correctness
MWN	Mismatched wait()	Multithreaded correctness
NN	Naked notify	Multithreaded correctness
NP	Synchronize and null check on the same field.	Multithreaded correctness
No	Using notify() rather than notifyAll()	Multithreaded correctness
RS	Class's readObject() method is synchronized	Multithreaded correctness
Ru	Invokes run on a thread (did you mean to start it instead?)	Multithreaded correctness
SC	Constructor invokes Thread.start()	Multithreaded correctness
SP	Method spins on field	Multithreaded correctness
STCAL	Call to static Calendar	Multithreaded correctness
STCAL	Call to static DateFormat	Multithreaded correctness
STCAL	Static Calendar	Multithreaded correctness
STCAL	Static DateFormat	Multithreaded correctness
SWL	Method calls Thread.sleep() with a lock held	Multithreaded correctness
TLW	Wait with two locks held	Multithreaded correctness
UG	Unsynchronized get method, synchronized set method	Multithreaded correctness

UL	Method does not release lock on all paths	Multithreaded correctness
UL	Method does not release lock on all exception paths	Multithreaded correctness
UW	Unconditional wait	Multithreaded correctness
VO	A volatile reference to an array doesn't treat the array elements as volatile	Multithreaded correctness
WS	Class's writeObject() method is synchronized but nothing else is	Multithreaded correctness
Wa	Condition.await() not in loop	Multithreaded correctness
Wa	Wait not in loop	Multithreaded correctness
Bx	Primitive value is boxed and then immediately unboxed	Performance
Bx	Primitive value is boxed then unboxed to perform primitive coercion	Performance
Bx	Method allocates a boxed primitive just to call toString	Performance
Bx	Method invokes inefficient floating-point Number constructor; use static valueOf instead	Performance
Bx	Method invokes inefficient Number constructor; use static valueOf instead	Performance
Dm	The equals and hashCode methods of URL are blocking	Performance
Dm	Maps and sets of URLs can be performance hogs	Performance
Dm	Method invokes inefficient Boolean constructor; use Boolean.valueOf(...) instead	Performance
Dm	Explicit garbage collection; extremely dubious except in benchmarking code	Performance
Dm	Method allocates an object, only to get the class object	Performance
Dm	Use the nextInt method of Random rather than nextDouble to generate a random integer	Performance
Dm	Method invokes inefficient new String(String) constructor	Performance
Dm	Method invokes inefficient String.equals(""); use String.length() == 0 instead	Performance
Dm	Method invokes toString() method on a String	Performance
Dm	Method invokes inefficient new String() constructor	Performance
HSC	Huge string constants is duplicated across multiple class files	Performance
ITA	Method uses toArray() with zero-length array argument	Performance
SBSC	Method concatenates strings using + in a loop	Performance
SIC	Should be a static inner class	Performance
SIC	Could be refactored into a named static inner class	Performance
SIC	Could be refactored into a static inner class	Performance
SS: Unread field	should this field be static?	Performance
UM	Method calls static Math class method on a constant value	Performance
UPM	Private method is never called	Performance
UrF	Unread field	Performance
UuF	Unused field	Performance
WMI	Inefficient use of keySet iterator instead of entrySet iterator	Performance
Dm	Hardcoded constant database password	Security

Dm	Empty database password	Security
HRS	HTTP cookie formed from untrusted input	Security
HRS	HTTP Response splitting vulnerability	Security
SQL	Nonconstant string passed to execute method on an SQL statement	Security
SQL	A prepared statement is generated from a nonconstant String	Security
XSS	JSP reflected cross site scripting vulnerability	Security
XSS	Servlet reflected cross site scripting vulnerability	Security
BC	Questionable cast to abstract collection	Dodgy
BC	Questionable cast to concrete collection	Dodgy
BC	Unchecked/unconfirmed cast	Dodgy
BC	instanceof will always return true	Dodgy
CI	Class is final but declares protected field	Dodgy
DB	Method uses the same code for two branches	Dodgy
DB	Method uses the same code for two switch clauses	Dodgy
DLS	Dead store to local variable	Dodgy
DLS	Dead store of null to local variable	Dodgy
DMI	Code contains a hard coded reference to an absolute pathname	Dodgy
DMI	Non serializable object written to ObjectOutput	Dodgy
DMI	Invocation of substring(0), which returns the original value	Dodgy
Dm	Thread passed where Runnable expected	Dodgy
Eq	Class doesn't override equals in superclass	Dodgy
FE	Test for floating point equality	Dodgy
IA	Ambiguous invocation of either an inherited or outer method	Dodgy
IC	Initialization circularity	Dodgy
ICAST	int division result cast to double or float	Dodgy
ICAST	Result of integer multiplication cast to long	Dodgy
ICAST	Unsigned right shift cast to short/byte	Dodgy
IM	Computation of average could overflow	Dodgy
IM	Check for oddness that won't work for negative numbers	Dodgy
INT	Vacuous comparison of integer value	Dodgy
MTIA	Class extends Servlet class and uses instance variables	Dodgy
MTIA	Class extends Struts Action class and uses instance variables	Dodgy
NP	Immediate dereference of the result of readLine()	Dodgy
NP	Load of known null value	Dodgy
NP	Possible null pointer dereference due to return value of called method	Dodgy
NP	Possible null pointer dereference on path that might be infeasible	Dodgy
NS	Potentially dangerous use of non-short-circuit logic	Dodgy
NS	Questionable use of non-short-circuit logic	Dodgy
PZLA	Consider returning a zero length array rather than null	Dodgy
QF	Complicated, subtle or wrong increment in for-loop	Dodgy
RCN	Redundant comparison of non-null value to null	Dodgy
RCN	Redundant comparison of two null values	Dodgy
RCN	Redundant nullcheck of value known to be non-null	Dodgy
RCN	Redundant nullcheck of value known to be null	Dodgy
REC	Exception is caught when Exception is not thrown	Dodgy
RI	Class implements same interface as superclass	Dodgy

RV	Method checks to see if result of String.indexOf is positive	Dodgy
RV	Method discards result of readLine after checking if it is nonnull	Dodgy
RV	Remainder of hashCode could be negative	Dodgy
RV	Remainder of 32-bit signed random integer	Dodgy
SA	Self assignment of local variable	Dodgy
SF	Switch statement found where one case falls through to the next case	Dodgy
ST	Write to static field from instance method	Dodgy
Se	Transient field of class that isn't Serializable.	Dodgy
UCF	Useless control flow	Dodgy
UwF	Field not initialized in constructor	Dodgy
XFB	Method directly allocates a specific implementation of xml interfaces	Dodgy