spotbugs Documentation

Release 3.1.3

spotbugs community

May 07, 2018

Contents

CHAPTER 1

Indices and tables

- search

CHAPTER 2

Contents

2.1 Introduction

SpotBugs is a program to find bugs in Java programs. It looks for instances of "bug patterns" — code instances that are likely to be errors.

This document describes version 3.1.3 of SpotBugs. We are very interested in getting your feedback on SpotBugs. Please visit the SpotBugs web page for the latest information on SpotBugs, contact information, and support resources such as information about the SpotBugs GitHub organization.

2.2 Requirements

To use SpotBugs, you need a runtime environment compatible with Java version 1.8 or later. SpotBugs is platform independent, and is known to run on GNU/Linux, Windows, and MacOS X platforms.

You should have at least 512 MB of memory to use SpotBugs. To analyze very large projects, more memory may be needed.

2.3 Installing

This chapter explains how to install SpotBugs.

2.3.1 Extracting the Distribution

The easiest way to install SpotBugs is to download a binary distribution. Binary distributions are available in gzipped tar format and zip format. Once you have downloaded a binary distribution, extract it into a directory of your choice.

Extracting a gzipped tar format distribution:

```
$ gunzip -c spotbugs-3.1.3.tgz | tar xvf -
```

Extracting a zip format distribution:

```
C:\Software> unzip spotbugs-3.1.3.zip
```

Usually, extracting a binary distribution will create a directory ending in spotbugs-3.1.3. For example, if you extracted the binary distribution from the C:\Software directory, then the SpotBugs software will be extracted into the directory C:\Software\spotbugs-3.1.3. This directory is the SpotBugs home directory. We'll refer to it as `$SPOTBUGS_HOME` (or `%SPOTBUGS_HOME%` for Windows) throughout this manual.

2.4 Running SpotBugs

SpotBugs has two user interfaces: a graphical user interface (GUI) and a command line user interface. This chapter describes how to run each of these user interfaces.

2.4.1 Quick Start

If you are running SpotBugs on a Windows system, double-click on the file `%SPOTBUGS_HOME%\lib\spotbugs.jar` to start the SpotBugs GUI.

On a Unix, Linux, or macOS system, run the `$SPOTBUGS_HOME/bin/spotbugs` script, or run the command `java -jar $SPOTBUGS_HOME/lib/spotbugs.jar` to run the SpotBugs GUI.

Refer to *Using the SpotBugs GUI* for information on how to use the GUI.

2.4.2 Executing SpotBugs

This section describes how to invoke the SpotBugs program. There are two ways to invoke SpotBugs: directly, or using a wrapper script.

Direct invocation of SpotBugs

The preferred method of running SpotBugs is to directly execute `$SPOTBUGS_HOME/lib/spotbugs.jar` using the -jar command line switch of the JVM (java) executable. (Versions of SpotBugs prior to 1.3.5 required a wrapper script to invoke SpotBugs.)

The general syntax of invoking SpotBugs directly is the following:

```
java [JVM arguments] -jar $SPOTBUGS_HOME/lib/spotbugs.jar options...
```

Choosing the User Interface

The first command line option chooses the SpotBugs user interface to execute. Possible values are:

-gui: runs the graphical user interface (GUI)

-textui: runs the command line user interface

-version: displays the SpotBugs version number

-help: displays help information for the SpotBugs command line user interface

-gui1: executes the original (obsolete) SpotBugs graphical user interface

Java Virtual Machine (JVM) arguments

Several Java Virtual Machine arguments are useful when invoking SpotBugs.

-XmxNNm: Set the maximum Java heap size to NN megabytes. SpotBugs generally requires a large amount of memory. For a very large project, using 1500 megabytes is not unusual.

-Dname=value: Set a Java system property. For example, you might use the argument `-Duser.language=ja` to display GUI messages in Japanese.

Invocation of SpotBugs using a wrapper script

Another way to run SpotBugs is to use a wrapper script.

On Unix-like systems, use the following command to invoke the wrapper script:

```
$ $SPOTBUGS_HOME/bin/spotbugs options...
```

On Windows systems, the command to invoke the wrapper script is

```
C:\My Directory>%SPOTBUGS_HOME%\bin\spotbugs.bat options...
```

On both Unix-like and Windows systems, you can simply add the `$SPOTBUGS_HOME/bin` directory to your `PATH` environment variable and then invoke SpotBugs using the `spotbugs` command.

Wrapper script command line options

The SpotBugs wrapper scripts support the following command-line options. Note that these command line options are not handled by the SpotBugs program per se; rather, they are handled by the wrapper script.

-jvmArgs *args*: Specifies arguments to pass to the JVM. For example, you might want to set a JVM property:

```
$ spotbugs -textui -jvmArgs "-Duser.language=ja" myApp.jar
```

-javahome *directory*: Specifies the directory containing the JRE (Java Runtime Environment) to use to execute FindBugs.

-maxHeap *size*: Specifies the maximum Java heap size in megabytes. The default is 256. More memory may be required to analyze very large programs or libraries.

-debug: Prints a trace of detectors run and classes analyzed to standard output. Useful for troubleshooting unexpected analysis failures.

-property *name=value*: This option sets a system property. SpotBugs uses system properties to configure analysis options. See *Analysis Properties*. You can use this option multiple times in order to set multiple properties. Note: In most versions of Windows, the name=value string must be in quotes.

2.4.3 Command-line Options

This section describes the command line options supported by SpotBugs. These command line options may be used when invoking SpotBugs directly, or when using a wrapper script.

Common command-line options

These options may be used with both the GUI and command-line interfaces.

-effort:min: This option disables analyses that increase precision but also increase memory consumption. You may want to try this option if you find that SpotBugs runs out of memory, or takes an unusually long time to complete its analysis. See *Effort*.

-effort:max: Enable analyses which increase precision and find more bugs, but which may require more memory and take more time to complete. See *Effort*.

-project *project*: Specify a project to be analyzed. The project file you specify should be one that was created using the GUI interface. It will typically end in the extension .fb or .fbp.

GUI Options

These options are only accepted by the Graphical User Interface.

-look:plastic|gtk|native: Set Swing look and feel.

Text UI Options

These options are only accepted by the Text User Interface.

-sortByClass: Sort reported bug instances by class name.

-include *filterFile.xml*: Only report bug instances that match the filter specified by filterFile.xml. See *Filter file*.

-exclude *filterFile.xml*: Report all bug instances except those matching the filter specified by filterFile.xml. See *Filter file*.

-onlyAnalyze *com.foobar.MyClass,com.foobar.mypkg.:** Restrict analysis to find bugs to given comma-separated list of classes and packages. Unlike filtering, this option avoids running analysis on classes and packages that are not explicitly matched: for large projects, this may greatly reduce the amount of time needed to run the analysis. (However, some detectors may produce inaccurate results if they aren't run on the entire application.) Classes should be specified using their full classnames (including package), and packages should be specified in the same way they would in a Java import statement to import all classes in the package (i.e., add .* to the full name of the package). Replace .* with .- to also analyze all subpackages.

-low: Report all bugs.

-medium: Report medium and high priority bugs. This is the default setting.

-high: Report only high priority bugs.

-relaxed: Relaxed reporting mode. For many detectors, this option suppresses the heuristics used to avoid reporting false positives.

-xml: Produce the bug reports as XML. The XML data produced may be viewed in the GUI at a later time. You may also specify this option as `-xml:withMessages`; when this variant of the option is used, the XML output will contain human-readable messages describing the warnings contained in the file. XML files generated this way are easy to transform into reports.

-html: Generate HTML output. By default, SpotBugs will use the default.xsl XSLT stylesheet to generate the HTML: you can find this file in spotbugs.jar, or in the SpotBugs source or binary distributions. Variants of this option include `-html:plain.xsl`, `-html:fancy.xsl` and `-html:fancy-hist.xsl`. The `plain.xsl` stylesheet does not use Javascript or DOM, and may work better with older web browsers, or for printing. The `fancy.xsl` stylesheet uses DOM and Javascript for navigation and CSS for visual presentation. The

`fancy-hist.xsl` an evolution of `fancy.xsl` stylesheet. It makes an extensive use of DOM and Javascript for dynamically filtering the lists of bugs.

If you want to specify your own XSLT stylesheet to perform the transformation to HTML, specify the option as `-html:myStylesheet.xsl`, where `myStylesheet.xsl` is the filename of the stylesheet you want to use.

-emacs: Produce the bug reports in Emacs format.

-xdocs: Produce the bug reports in xdoc XML format for use with Apache Maven.

-output *filename*: Produce the output in the specified file.

-outputFile *filename*: This argument is deprecated. Use `-output` instead.

-nested[:true|false]: This option enables or disables scanning of nested jar and zip files found in the list of files and directories to be analyzed. By default, scanning of nested jar/zip files is enabled. To disable it, add `-nested:false` to the command line arguments.

-auxclasspath *classpath*: Set the auxiliary classpath for analysis. This classpath should include all jar files and directories containing classes that are part of the program being analyzed but you do not want to have analyzed for bugs.

-auxclasspathFromInput: Read the auxiliary classpath for analysis from standard input, each line adds new entry to the auxiliary classpath for analysis.

-auxclasspathFromFile *filepath*: Read the auxiliary classpath for analysis from file, each line adds new entry to the auxiliary classpath for analysis.

-analyzeFromFile *filepath*: Read the files to analyze from file, each line adds new entry to the classpath for analysis.

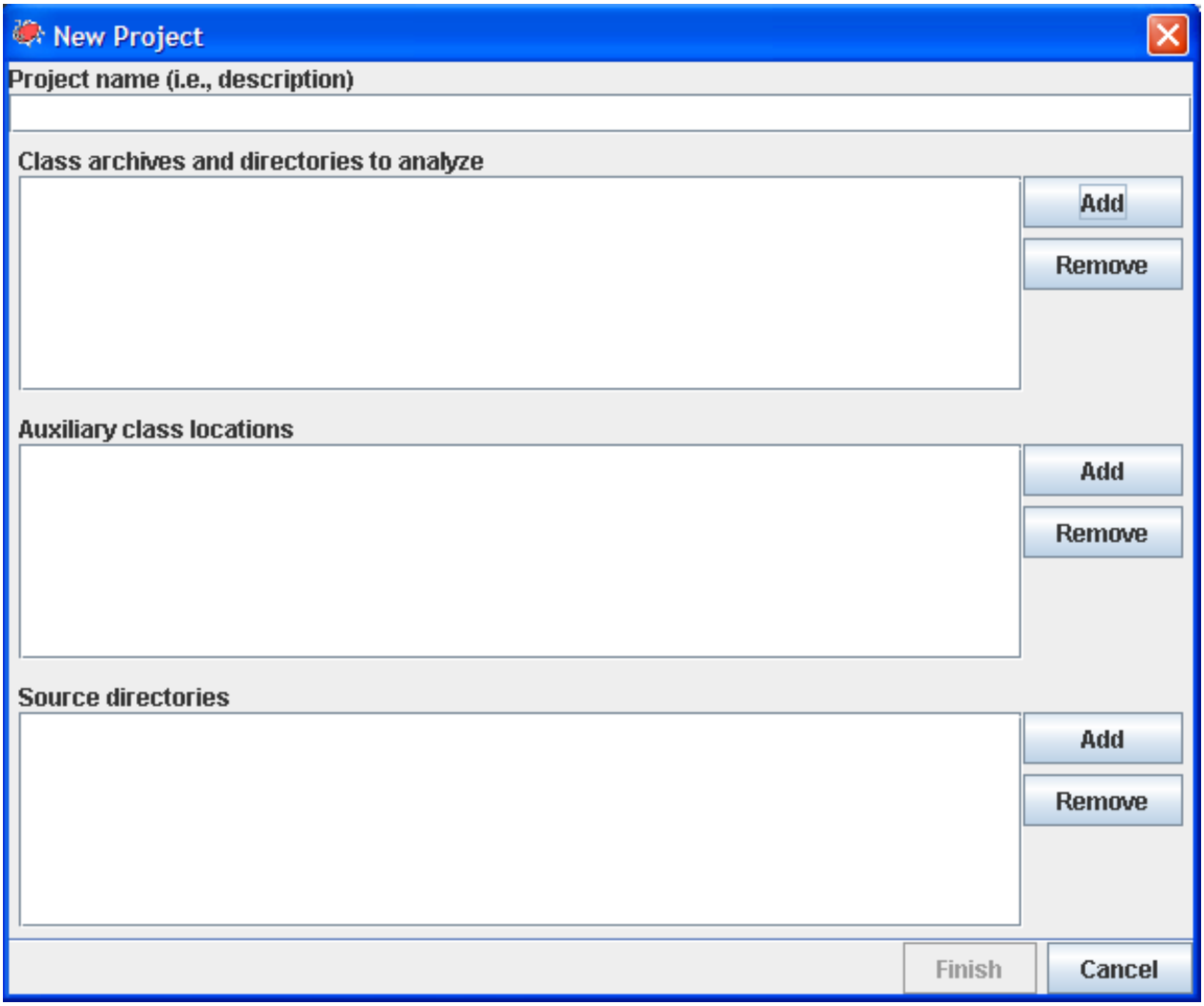
-userPrefs *edu.umd.cs.findbugs.core.prefs*: Set the path of the user preferences file to use, which might override some of the options above. Specifying userPrefs as first argument would mean some later options will override them, as last argument would mean they will override some previous options). This rationale behind this option is to reuse SpotBugs Eclipse project settings for command line execution.

2.5 Using the SpotBugs GUI

This chapter describes how to use the SpotBugs graphical user interface (GUI).

2.5.1 Creating a Project

After you have started SpotBugs using the `spotbugs` command, choose the `File → New Project` menu item. You will see a dialog which looks like this:



Use the "Add" button next to "Classpath to analyze" to select a Java archive file (zip, jar, ear, or war file) or directory containing java classes to analyze for bugs. You may add multiple archives/directories.

You can also add the source directories which contain the source code for the Java archives you are analyzing. This will enable SpotBugs to highlight the source code which contains a possible error. The source directories you add should be the roots of the Java package hierarchy. For example, if your application is contained in the `org.foobar.myapp` package, you should add the parent directory of the org directory to the source directory list for the project.

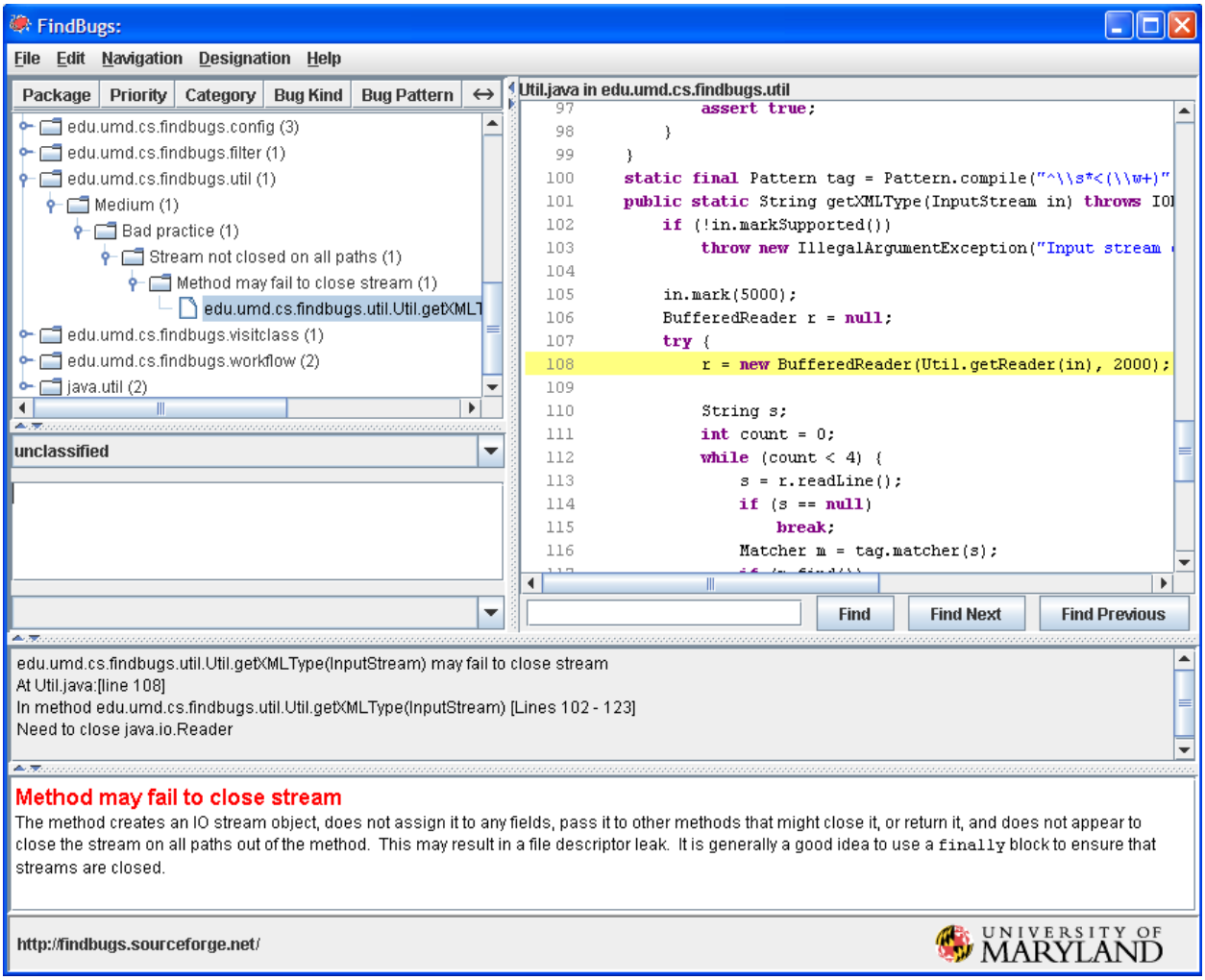
Another optional step is to add additional Jar files or directories as "Auxiliary classpath locations" entries. You should do this if the archives and directories you are analyzing have references to other classes which are not included in the analyzed archives/directories and are not in the standard runtime classpath. Some of the bug pattern detectors in FindBugs make use of class hierarchy information, so you will get more accurate results if the entire class hierarchy is available which FindBugs performs its analysis.

2.5.2 Running the Analysis

Once you have added all of the archives, directories, and source directories, click the "Analyze" button to analyze the classes contained in the Jar files. Note that for a very large program on an older computer, this may take quite a while (tens of minutes). A recent computer with ample memory will typically be able to analyze a large program in only a few minutes.

2.5.3 Browsing Results

When the analysis completes, you will see a screen like the following:



The upper left-hand pane of the window shows the bug tree; this is a hierarchical representation of all of the potential bugs detected in the analyzed Jar files.

When you select a particular bug instance in the top pane, you will see a description of the bug in the "Details" tab of the bottom pane. In addition, the source code pane on the upper-right will show the program source code where the potential bug occurs, if source is available. In the above example, the bug is a stream object that is not closed. The source code window highlights the line where the stream object is created.

You may add a textual annotations to bug instances. To do so, type them into the text box just below the hierarchical view. You can type any information which you would like to record. When you load and save bug results files, the annotations are preserved.

2.5.4 Saving and Opening

You may use the `File → Save as...` menu option to save your work. To save your work, including the jar file lists you specified and all bug results, choose "FindBugs analysis results (.xml)" from the drop-down list in the "Save as..." dialog. There are also options for saving just the jar file lists ("FindBugs project file (.fbp)") or just the results ("FindBugs analysis file (.fba)"). A saved file may be loaded with the `File → Open...` menu option.

2.6 Using the SpotBugs Eclipse plugin

The SpotBugs Eclipse plugin allows SpotBugs to be used within the Eclipse IDE. The SpotBugs Eclipse plugin was generously contributed by Peter Friese. Phil Crosby and Andrey Loskutov contributed major improvements to the plugin.

2.6.1 Requirements

To use the SpotBugs Plugin for Eclipse, you need Eclipse Neon (4.6) or later.

2.6.2 Installation

We provide update sites that allow you to automatically install SpotBugs into Eclipse and also query and install updates. There are three different update sites:

https://spotbugs.github.io/eclipse/ Only provides official releases of SpotBugs Eclipse plugin.

https://spotbugs.github.io/eclipse-candidate/ Provides official releases and release candidates of SpotBugs Eclipse plugin.

https://spotbugs.github.io/eclipse-latest/ Provides latest SpotBugs Eclipse plugin built from master branch.

Or just use Eclipse marketplace to install SpotBugs Eclipse plugin.

2.6.3 Using the Plugin

To get started, right click on a Java project in Package Explorer, and select the option labeled "Spot Bugs". SpotBugs will run, and problem markers (displayed in source windows, and also in the Eclipse Problems view) will point to locations in your code which have been identified as potential instances of bug patterns.

You can also run SpotBugs on existing java archives (jar, ear, zip, war etc). Simply create an empty Java project and attach archives to the project classpath. Having that, you can now right click the archive node in Package Explorer and select the option labeled "Spot Bugs". If you additionally configure the source code locations for the binaries, SpotBugs will also link the generated warnings to the right source files.

You may customize how SpotBugs runs by opening the Properties dialog for a Java project, and choosing the "SpotBugs" property page. Options you may choose include:

- Enable or disable the "Run SpotBugs Automatically" checkbox. When enabled, SpotBugs will run every time you modify a Java class within the project.
- Choose minimum warning priority and enabled bug categories. These options will choose which warnings are shown. For example, if you select the "Medium" warning priority, only Medium and High priority warnings will be shown. Similarly, if you uncheck the "Style" checkbox, no warnings in the Style category will be displayed.
- Select detectors. The table allows you to select which detectors you want to enable for your project.

2.6.4 Extending the Eclipse Plugin (since 2.0.0)

Eclipse plugin supports contribution of custom SpotBugs detectors (see also AddingDetectors.txt for more information). There are two ways to contribute custom plugins to the Eclipse:

- Existing standard SpotBugs detector packages can be configured via `Window → Preferences → Java → FindBugs → Misc. Settings → Custom Detectors`. Simply specify there locations of any additional plugin libraries. The benefit of this solution is that already existing detector packages can be used "as is", and that you can quickly verify the quality of third party detectors. The drawback is that you have to apply this settings in each new Eclipse workspace, and this settings can't be shared between team members.
- It is possible to contribute custom detectors via standard Eclipse extensions mechanism.

  Please check the documentation of the `eclipsePlugin/schema/detectorPlugins.exsd` extension point how to update the plugin.xml. Existing FindBugs detector plugins can be easily "extended" to be full featured SpotBugs AND Eclipse detector plugins. Usually you only need to add `META-INF/MANIFEST.MF` and `plugin.xml` to the jar and update your build scripts to not to override the `MANIFEST.MF` during the build.

  The benefit of this solution is that for given (shared) Eclipse installation each team member has exactly same detectors set, and there is no need to configure anything anymore. The (really small) precondition is that you have to convert your existing detectors package to the valid Eclipse plugin. You can do this even for third-party detector packages. Another major differentiator is the ability to extend the default SpotBugs classpath at runtime with required third party libraries (see AddingDetectors.txt for more information).

2.6.5 Troubleshooting

This section lists common problems with the plugin and (if known) how to resolve them.

- If you see OutOfMemory error dialogs after starting SpotBugs analysis in Eclipse, please increase JVM available memory: change `eclipse.ini` and add the lines below to the end of the file:

```
-vmargs
-Xmx1000m
```

  Important: the configuration arguments starting with the line `-vmargs` must be last lines in the `eclipse.ini` file, and only one argument per line is allowed!

- If you do not see any SpotBugs problem markers (in your source windows or in the Problems View), you may need to change your `Problems View` filter settings. See FAQ for more information.

2.7 Using the SpotBugs Ant task

This chapter describes how to integrate SpotBugs into a build script for Ant, which is a popular Java build and deployment tool. Using the SpotBugs Ant task, your build script can automatically run SpotBugs on your Java code.

The Ant task was generously contributed by Mike Fagan.

2.7.1 Installing the Ant task

To install the Ant task, simply copy `$SPOTBUGS_HOME/lib/spotbugs-ant.jar` into the lib subdirectory of your Ant installation.

Note: It is strongly recommended that you use the Ant task with the version of SpotBugs it was included with. We do not guarantee that the Ant task Jar file will work with any version of SpotBugs other than the one it was included with.

2.7.2 Modifying build.xml

To incorporate SpotBugs into build.xml (the build script for Ant), you first need to add a task definition. This should appear as follows:

```
<taskdef
    resource="edu/umd/cs/findbugs/anttask/tasks.properties"
    classpath="path/to/spotbugs-ant.jar" />
```

The task definition specifies that when a spotbugs element is seen in `build.xml`, it should use the indicated class to execute the task.

After you have added the task definition, you can define a target which uses the spotbugs task. Here is an example which could be added to the `build.xml` for the Apache BCEL library.

```
<property name="spotbugs.home" value="/export/home/daveho/work/spotbugs" />

<target name="spotbugs" depends="jar">
  <spotbugs home="${spotbugs.home}"
            output="xml"
            outputFile="bcel-sb.xml" >
    <auxClasspath path="${basedir}/lib/Regex.jar" />
    <sourcePath path="${basedir}/src/java" />
    <class location="${basedir}/bin/bcel.jar" />
  </spotbugs>
</target>
```

The spotbugs element must have the home attribute set to the directory in which SpotBugs is installed; in other words, `$SPOTBUGS_HOME`. See *Installing*.

This target will execute SpotBugs on `bcel.jar`, which is the Jar file produced by BCEL's build script. (By making it depend on the "jar" target, we ensure that the library is fully compiled before running SpotBugs on it.) The output of SpotBugs will be saved in XML format to a file called `bcel-sb.xml`. An auxiliary Jar file, Regex.jar, is added to the aux classpath, because it is referenced by the main BCEL library. A source path is specified so that the saved bug data will have accurate references to the BCEL source code.

2.7.3 Executing the task

Here is an example of invoking Ant from the command line, using the spotbugs target defined above.

```
[daveho@noir]$ ant spotbugs
Buildfile: build.xml

init:

compile:

examples:

jar:

spotbugs:
 [spotbugs] Running SpotBugs...
 [spotbugs] Bugs were found
 [spotbugs] Output saved to bcel-sb.xml
```

(continues on next page)

(continued from previous page)

```
BUILD SUCCESSFUL
Total time: 35 seconds
```

In this case, because we saved the bug results in an XML file, we can use the SpotBugs GUI to view the results; see *Running SpotBugs*.

2.7.4 Parameters

This section describes the parameters that may be specified when using the FindBugs task.

class A optional nested element specifying which classes to analyze. The class element must specify a location attribute which names the archive file (jar, zip, etc.), directory, or class file to be analyzed. Multiple class elements may be specified as children of a single spotbugs element.

In addition to or instead of specifying a class element, the SpotBugs task can contain one or more fileset element(s) that specify files to be analyzed. For example, you might use a fileset to specify that all of the jar files in a directory should be analyzed.

auxClasspath An optional nested element which specifies a classpath (Jar files or directories) containing classes used by the analyzed library or application, but which you don't want to analyze. It is specified the same way as Ant's classpath element for the Java task.

sourcePath An optional nested element which specifies a source directory path containing source files used to compile the Java code being analyzed. By specifying a source path, any generated XML bug output will have complete source information, which allows later viewing in the GUI.

home A required attribute. It must be set to the name of the directory where SpotBugs is installed.

quietErrors An optional boolean attribute. If true, reports of serious analysis errors and missing classes will be suppressed in the SpotBugs output. Default is false.

reportLevel An optional attribute. It specifies the confidence/priority threshold for reporting issues. If set to `low`, confidence is not used to filter bugs. If set to `medium` (the default), low confidence issues are supressed. If set to `high`, only high confidence bugs are reported.

output Optional attribute. It specifies the output format. If set to `xml` (the default), output is in XML format. If set to "xml:withMessages", output is in XML format augmented with human-readable messages. (You should use this format if you plan to generate a report using an XSL stylesheet.) If set to "html", output is in HTML formatted (default stylesheet is default.xsl). If set to `text`, output is in ad-hoc text format. If set to `emacs`, output is in Emacs error message format. If set to `xdocs`, output is xdoc XML for use with Apache Maven.

stylesheet Optional attribute. It specifies the stylesheet to use to generate html output when the output is set to html. Stylesheets included in the FindBugs distribution include default.xsl, fancy.xsl, fancy-hist.xsl, plain.xsl, and summary.xsl. The default value, if no stylesheet attribute is provided, is default.xsl.

sort Optional attribute. If the output attribute is set to `text`, then the sort attribute specifies whether or not reported bugs are sorted by class. Default is `true`.

outputFile Optional attribute. If specified, names the output file in which the FindBugs output will be saved. By default, the output is displayed directly by Ant.

debug Optional boolean attribute. If set to `true`, SpotBugs prints diagnostic information about which classes are being analyzed, and which bug pattern detectors are being run. Default is `false`.

effort Set the analysis effort level. The value specified should be one of `min`, `default`, or `max`. See *Command-line Options <running.html#command-line-options>*: for more information about setting the analysis level.

conserveSpace Synonym for `effort="min"`.

workHard Synonym for `effort="max"`.

visitors Optional attribute. It specifies a comma-separated list of bug detectors which should be run. The bug detectors are specified by their class names, without any package qualification. By default, all detectors which are not disabled by default are run.

omitVisitors Optional attribute. It specifies a comma-separated list of bug detectors. It is like the visitors attribute, except it specifies detectors which will not be run.

chooseVisitors Optional attribute. It specifies a comma-separated list of bug detectors prefixed with "+" or "-" to selectively enable/disable them.

excludeFilter Optional attribute. It specifies the filename of a filter specifying bugs to exclude from being reported. See *Filter file*.

includeFilter Optional attribute. It specifies the filename of a filter specifying which bugs are reported. See *Filter file*.

projectFile Optional attribute. It specifies the name of a project file. Project files are created by the FindBugs GUI, and specify classes, aux classpath entries, and source directories. By naming a project, you don't need to specify any class elements, nor do you need to specify `auxClasspath` or `sourcePath` attributes. See *Running SpotBugs* for how to create a project.

jvmargs Optional attribute. It specifies any arguments that should be passed to the Java virtual machine used to run SpotBugs. You may need to use this attribute to specify flags to increase the amount of memory the JVM may use if you are analyzing a very large program.

systemProperty Optional nested element. If specified, defines a system property. The name attribute specifies the name of the system property, and the value attribute specifies the value of the system property.

timeout Optional attribute. It specifies the amount of time, in milliseconds, that the Java process executing SpotBugs may run before it is assumed to be hung and is terminated. The default is 600,000 milliseconds, which is ten minutes. Note that for very large programs, SpotBugs may require more than ten minutes to complete its analysis.

failOnError Optional boolean attribute. Whether to abort the build process if there is an error running SpotBugs. Defaults to `false`.

errorProperty Optional attribute which specifies the name of a property that will be set to `true` if an error occurs while running SpotBugs.

warningsProperty Optional attribute which specifies the name of a property that will be set to `true` if any warnings are reported by SpotBugs on the analyzed program.

userPreferencesFile Optional attribute. Set the path of the user preferences file to use, which might override some of the options above. Specifying `userPreferencesFile` as first argument would mean some later options will override them, as last argument would mean they will override some previous options). This rationale behind this option is to reuse SpotBugs Eclipse project settings for command line execution.

nested Optional attribute which enables or disables scanning of nested jar and zip files found in the list of files and directories to be analyzed. By default, scanning of nested jar/zip files is enabled.

setExitCode Optional boolean attribute. Whether the exit code will be returned to the main ant job. Defaults to `true`.

2.8 Using the SpotBugs Maven Plugin

This chapter describes how to integrate SpotBugs into a Maven project.

2.8.1 Add spotbugs-maven-plugin to your pom.xml

Add `<plugin>` into your `pom.xml` like below:

```
<plugin>
  <groupId>com.github.spotbugs</groupId>
  <artifactId>spotbugs-maven-plugin</artifactId>
  <version>3.1.3</version>
  <dependencies>
    <!-- overwrite dependency on spotbugs if you want to specify the version of␣
↪spotbugs -->
    <dependency>
      <groupId>com.github.spotbugs</groupId>
      <artifactId>spotbugs</artifactId>
      <version>3.1.3</version>
    </dependency>
  </dependencies>
</plugin>
```

2.8.2 Goals of spotbugs-maven-plugin

spotbugs goal

`spotbugs` goal analyses target project by SpotBugs. For detail, please refer spotbugs goal description in maven site.

check goal

`check` goal runs analysis like `spotbugs` goal, and make the build failed if it found any bugs. For detail, please refer check goal description in maven site.

gui goal

`gui` goal launches SpotBugs GUI to check analysis result. For detail, please refer gui goal description in maven site.

help goal

`help` goal displays usage of this Maven plugin.

2.9 Using the SpotBugs Gradle Plugin

This chapter describes how to integrate SpotBugs into a build script for Gradle.

2.9.1 Use SpotBugs Gradle Plugin

Please follow instruction found on official Gradle Plugin page.

Note that SpotBugs Gradle Plugin does not support Gradle v3, you need to use v4 or later.

2.9.2 Tasks introduced by this Gradle Plugin

This Gradle Plugin introduces two tasks: *spotbugsMain* and *spotbugsTest*.

spotbugsMain task runs SpotBugs for your production Java source files. This task depends on *classes* task. *spotbugsTest* task runs SpotBugs for your test Java source files. This task depends on *testClasses* task.

SpotBugs Gradle Plugin adds task dependency from *check* to these tasks, so you can simply run `./gradlew check` to run SpotBugs.

2.9.3 Configure Gradle Plugin

Current version of SpotBugs Gradle Plugin uses the same way with FindBugs Gradle Plugin to configure. Please check the document for FindBugsExtension.

For instance, to specify the version of SpotBugs, you can configure like below:

```
spotbugs {
  toolVersion = '3.1.3'
}
```

2.9.4 Introduce SpotBugs Plugin

To introduce SpotBugs Plugin, please declare dependency in `dependencies` like below:

```
dependencies {
  spotbugsPlugins 'com.h3xstream.findsecbugs:findsecbugs-plugin:1.7.1'
}
```

2.9.5 Generate SpotBugs Tasks with Android Gradle Plugin

SpotBugs Gradle Plugin generates task for each sourceSet. But Android Gradle Plugin does not generate sourceSet by default (Java plugin does).

So define sourceSets explicitly, then SpotBugs Gradle plugin generates tasks for each of them.

```
sourceSets {
  // we define `main` sourceSet here, so SpotBugs Gradle Plugin generates␣
↪`spotbugsMain` task
  main {
    java.srcDirs = ['src/main/java']
  }
}

tasks.withType(com.github.spotbugs.SpotBugsTask) {
  // configure automatically generated tasks
}
```

2.10 Filter file

Filter files may be used to include or exclude bug reports for particular classes and methods. This chapter explains how to use filter files.

2.10.1 Introduction to Filter Files

Conceptually, a filter matches bug instances against a set of criteria. By defining a filter, you can select bug instances for special treatment; for example, to exclude or include them in a report.

A filter file is an XML document with a top-level `FindBugsFilter` element which has some number of Match elements as children. Each Match element represents a predicate which is applied to generated bug instances. Usually, a filter will be used to exclude bug instances. For example:

```
$ spotbugs -textui -exclude myExcludeFilter.xml myApp.jar
```

However, a filter could also be used to select bug instances to specifically report:

```
$ spotbugs -textui -include myIncludeFilter.xml myApp.jar
```

`Match` elements contain children, which are conjuncts of the predicate. In other words, each of the children must be `true` for the predicate to be `true`.

2.10.2 Types of Match clauses

<Bug>

This element specifies a particular bug `pattern or patterns to match. The ``pattern` attribute is a comma-separated list of bug pattern types. You can find the bug pattern types for particular warnings by looking at the output produced by the **-xml** output option (the type attribute of BugInstance elements), or from the *Bug descriptions*.

For more coarse-grained matching, use `code` attribute. It takes a comma-separated list of bug abbreviations. For most-coarse grained matching use `category` attriute, that takes a comma separated list of bug category names: `CORRECTNESS`, `MT_CORRECTNESS`, `BAD_PRACTICICE`, `PERFORMANCE`, `STYLE`.

If more than one of the attributes mentioned above are specified on the same <Bug> element, all bug patterns that match either one of specified pattern names, or abreviations, or categories will be matched.

As a backwards compatibility measure, <BugPattern> and <BugCode> elements may be used instead of <Bug> element. Each of these uses a name attribute for specifying accepted values list. Support for these elements may be removed in a future release.

<Confidence>

This element matches warnings with a particular bug confidence. The `value` attribute should be an integer value: 1 to match high-confidence warnings, 2 to match normal-confidence warnings, or 3 to match low-confidence warnings. `<Confidence>` replaced `<Priority>` in 2.0.0 release.

<Priority>

Same as `<Confidence>`, exists for backward compatibility.

<Rank>

This element matches warnings with a particular bug rank. The `value` attribute should be an integer value between 1 and 20, where 1 to 4 are scariest, 5 to 9 scary, 10 to 14 troubling, and 15 to 20 of concern bugs.

<Package>

This element matches warnings associated with classes within the package specified using `name` attribute. Nested packages are not included (along the lines of Java import statement). However matching multiple packages can be achieved easily using regex name match.

<Class>

This element matches warnings associated with a particular class. The `name` attribute is used to specify the exact or regex match pattern for the class name. The `role` attribute is the class role.

As a backward compatibility measure, instead of element of this type, you can use `class` attribute on a `Match` element to specify exact an class name or `classregex` attribute to specify a regular expression to match the class name against.

If the `Match` element contains neither a `Class` element, nor a `class` / `classregex` attribute, the predicate will apply to all classes. Such predicate is likely to match more bug instances than you want, unless it is refined further down with appropriate method or field predicates.

<Source>

This element matches warnings associated with a particular source file. The `name` attribute is used to specify the exact or regex match pattern for the source file name.

<Method>

This element specifies a method. The `name` attribute is used to specify the exact or regex match pattern for the method name. The `params` attribute is a comma-separated list of the types of the method's parameters. The `returns` attribute is the method's return type. The `role` attribute is the method role. In `params` and `returns`, class names must be fully qualified. (E.g., `"java.lang.String"` instead of just `"String"`.) If one of the latter attributes is specified the other is required for creating a method signature. Note that you can provide either `name` attribute or `params` and `returns` attributes or all three of them. This way you can provide various kinds of name and signature based matches.

<Field>

This element specifies a field. The `name` attribute is used to specify the exact or regex match pattern for the field name. You can also filter fields according to their signature - use `type` attribute to specify fully qualified type of the field. You can specify either or both of these attributes in order to perform name / signature based matches. The `role` attribute is the field role.

<Local>

This element specifies a local variable. The `name` attribute is used to specify the exact or regex match pattern for the local variable name. Local variables are variables defined within a method.

<Type>

This element matches warnings associated with a particular type. The `descriptor` attribute is used to specify the exact or regex match pattern for type descriptor. If the descriptor starts with the ~ character the rest of attribute content

is interpreted as a Java regular expression. The `role` attribute is the class role, and the `typeParameters` is the type parameters. Both of `role` and `typeParameters` are optional attributes.

<Or>

This element combines `Match` clauses as disjuncts. I.e., you can put two `Method` elements in an `Or` clause in order to match either method.

<And>

This element combines `Match` clauses which both must evaluate to `true`. I.e., you can put `Bug` and `Confidence` elements in an `And` clause in order to match specific bugs with given confidence only.

<Not>

This element inverts the included child `Match`. I.e., you can put a `Bug` element in a `Not` clause in order to match any bug excluding the given one.

2.10.3 Java element name matching

If the `name` attribute of `Class`, `Source`, `Method` or `Field` starts with the ~ character the rest of attribute content is interpreted as a Java regular expression that is matched against the names of the Java element in question.

Note that the pattern is matched against whole element name and therefore `.*` clauses need to be used at pattern beginning and/or end to perform substring matching.

See java.util.regex.Pattern documentation for pattern syntax.

2.10.4 Caveats

`Match` clauses can only match information that is actually contained in the bug instances. Every bug instance has a class, so in general, excluding bugs by class will work.

Some bug instances have two (or more) classes. For example, the DE (dropped exception) bugs report both the class containing the method where the dropped exception happens, and the class which represents the type of the dropped exception. Only the *first* (primary) class is matched against `Match` clauses. So, for example, if you want to suppress IC (initialization circularity) reports for classes "com.foobar.A" and "com.foobar.B", you would use two `Match` clauses:

```
<Match>
   <Class name="com.foobar.A" />
   <Bug code="IC" />
</Match>
<Match>
   <Class name="com.foobar.B" />
   <Bug code="IC" />
</Match>
```

By explicitly matching both classes, you ensure that the IC bug instance will be matched regardless of which class involved in the circularity happens to be listed first in the bug instance. (Of course, this approach might accidentally supress circularities involving "com.foobar.A" or "com.foobar.B" and a third class.)

Many kinds of bugs report what method they occur in. For those bug instances, you can put Method clauses in the Match element and they should work as expected.

2.10.5 Examples

Match all bug reports for a class

```
<Match>
  <Class name="com.foobar.MyClass" />
</Match>
```

Match certain tests from a class by specifying their abbreviations

```
<Match>
  <Class name="com.foobar.MyClass"/ >
  <Bug code="DE,UrF,SIC" />
</Match>
```

Match certain tests from all classes by specifying their abbreviations

```
<Match>
  <Bug code="DE,UrF,SIC" />
</Match>
```

Match certain tests from all classes by specifying their category

```
<Match>
  <Bug category="PERFORMANCE" />
</Match>
```

Match bug types from specified methods of a class by their abbreviations

```
<Match>
  <Class name="com.foobar.MyClass" />
  <Or>
    <Method name="frob" params="int,java.lang.String" returns="void" />
    <Method name="blat" params="" returns="boolean" />
  </Or>
  <Bug code="DC" />
</Match>
```

Match a particular bug pattern in a particular method

```
<!-- A method with an open stream false positive. -->
<Match>
  <Class name="com.foobar.MyClass" />
  <Method name="writeDataToFile" />
  <Bug pattern="OS_OPEN_STREAM" />
</Match>
```

Match a particular bug pattern with a given priority in a particular method

```
<!-- A method with a dead local store false positive (medium priority). -->
<Match>
  <Class name="com.foobar.MyClass" />
  <Method name="someMethod" />
  <Bug pattern="DLS_DEAD_LOCAL_STORE" />
  <Priority value="2" />
</Match>
```

Match minor bugs introduced by AspectJ compiler (you are probably not interested in these unless you are an AspectJ developer)

```
<Match>
  <Class name="~.*\$AjcClosure\d+" />
  <Bug pattern="DLS_DEAD_LOCAL_STORE" />
  <Method name="run" />
</Match>
<Match>
  <Bug pattern="UUF_UNUSED_FIELD" />
  <Field name="~ajc\$.*" />
</Match>
```

Match bugs in specific parts of the code base

```
<!-- match unused fields warnings in Messages classes in all packages -->
<Match>
  <Class name="~.*\.Messages" />
  <Bug code="UUF" />
</Match>
<!-- match mutable statics warnings in all internal packages -->
<Match>
  <Package name="~.*\.internal" />
  <Bug code="MS" />
</Match>
<!-- match anonymoous inner classes warnings in ui package hierarchy -->
<Match>
  <Package name="~com\.foobar\.fooproject\.ui.*" />
  <Bug pattern="SIC_INNER_SHOULD_BE_STATIC_ANON" />
</Match>
```

Match bugs on fields or methods with specific signatures

```
<!-- match System.exit(...) usage warnings in void main(String[]) methods in all␣
↪classes -->
<Match>
  <Method returns="void" name="main" params="java.lang.String[]" />
  <Bug pattern="DM_EXIT" />
</Match>
<!-- match UuF warnings on fields of type com.foobar.DebugInfo on all classes -->
<Match>
  <Field type="com.foobar.DebugInfo" />
```

(continues on next page)

(continued from previous page)

```
  <Bug code="UuF" />
</Match>
```

Match bugs using the Not filter operator

```
<!-- ignore all bugs in test classes, except for those bugs specifically relating to␣
↪JUnit tests -->
<!-- i.e. filter bug if ( classIsJUnitTest && ! bugIsRelatedToJUnit ) -->
<Match>
  <!-- the Match filter is equivalent to a logical 'And' -->

  <Class name="~.*\.*Test" />
  <!-- test classes are suffixed by 'Test' -->

  <Not>
      <Bug code="IJU" /> <!-- 'IJU' is the code for bugs related to JUnit test code --
↪>
  </Not>
</Match>
```

Full exclusion filter file to match all classes generated from Groovy source files

```
<?xml version="1.0" encoding="UTF-8"?>
<FindBugsFilter>
<Match>
  <Source name="~.*\.groovy" />
</Match>
</FindBugsFilter>
```

2.10.6 Complete Example

```
  <?xml version="1.0" encoding="UTF-8"?>
  <FindBugsFilter
             xmlns="https://github.com/spotbugs/filter/3.0.0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://github.com/spotbugs/filter/3.0.0 https://
↪raw.githubusercontent.com/spotbugs/spotbugs/3.1.0/spotbugs/etc/findbugsfilter.xsd">
  <Match>
    <Class name="com.foobar.ClassNotToBeAnalyzed" />
  </Match>

  <Match>
    <Class name="com.foobar.ClassWithSomeBugsMatched" />
    <Bug code="DE,UrF,SIC" />
  </Match>

  <!-- Match all XYZ violations. -->
  <Match>
    <Bug code="XYZ" />
  </Match>
```

(continues on next page)

(continued from previous page)

```
  <!-- Match all doublecheck violations in these methods of "AnotherClass". -->
  <Match>
    <Class name="com.foobar.AnotherClass" />
    <Or>
      <Method name="nonOverloadedMethod" />
      <Method name="frob" params="int,java.lang.String" returns="void" />
      <Method name="blat" params="" returns="boolean" />
    </Or>
    <Bug code="DC" />
  </Match>

  <!-- A method with a dead local store false positive (medium priority). -->
  <Match>
    <Class name="com.foobar.MyClass" />
    <Method name="someMethod" />
    <Bug pattern="DLS_DEAD_LOCAL_STORE" />
    <Priority value="2" />
  </Match>

  <!-- All bugs in test classes, except for JUnit-specific bugs -->
  <Match>
  <Class name="~.*\.*Test" />
  <Not>
    <Bug code="IJU" />
  </Not>
  </Match>
</FindBugsFilter>
```

2.11 Analysis Properties

SpotBugs allows several aspects of the analyses it performs to be customized. System properties are used to configure these options. This chapter describes the configurable analysis options.

The analysis options have two main purposes. First, they allow you to inform SpotBugs about the meaning of methods in your application, so that it can produce more accurate results, or produce fewer false warnings. Second, they allow you to configure the precision of the analysis performed. Reducing analysis precision can save memory and analysis time, at the expense of missing some real bugs, or producing more false warnings.

The analysis options are set using the `-property` command line option. For example:

```
$ spotbugs -textui -property "cfg.noprune=true" myApp.jar
```

The list of configurable analysis properties is shown in following table:

Property Name	Value	Meaning
findbugs.assertionmethods	Comma-separated list of fully qualified method names: e.g., "com.foo.MyClass.checkAssertion"	This property specifies the names of methods that are used to check program assertions. Specifying these methods allows the null pointer dereference bug detector to avoid reporting false warnings for values which are checked by assertion methods.
findbugs.de.comment	true or false	If true, the DroppedException detector scans source code for empty catch blocks for a comment, and if one is found, does not report a warning.
findbugs.maskedfields.locals	true or false	If true, emit low priority warnings for local variables which obscure fields. Default is false.
findbugs.nullderef.assumensp	true or false	not used (intention: If true, the null dereference detector assumes that any reference value returned from a method or passed to a method in a parameter might be null. Default is false. Note that enabling this property will very likely cause a large number of false warnings to be produced.)
findbugs.refcomp.reportAll	true or false	If true, all suspicious reference comparisons using the == and != operators are reported.,If false, only one such warning is issued per method.,Default is false.
findbugs.sf.comment	true or false	If true, the SwitchFallthrough detector will only report warnings for cases where the source code does not have a comment containing the words "fall" or "nobreak". (An accurate source path must be used for this feature to work correctly.) This helps find cases where the switch fallthrough is likely to be unintentional.

2.12 Effort

Effort value adjusts internal flags of SpotBugs, to reduce computation cost by lowering the prediction.

The default effort configuration is same with `more`.

Flags in FindBugs.java	Description	Effort Level			
		min	less	more	max
Accurate Exceptions	Determine (1) what exceptions can be thrown on exception edges, (2) which,catch blocks are reachable, and (3) which exception edges carry only, "implicit" runtime exceptions.				
Model Instanceof	Model the effect of instanceof checks in type analysis				
Track Guaranteed Value Derefs in Null Pointer Analysis	In the null pointer analysis, track null values that are guaranteed to be, dereferenced on some (non-implicit-exception) path.				
Track Value Numbers in Null Pointer Analysis	In the null pointer analysis, track value numbers that are known to be, null. This allows us to not lose track of null values that are not, currently in the stack frame but might be in a heap location where the, value is recoverable by redundant load elimination or forward, substitution.				
Interprocedural Analysis	Enable interprocedural analysis for application classes.				
Interprocedural Analysis of Referenced Classes	Enable interprocedural analysis for referenced classes (non-application classes).				
Conserve Space	Try to conserve space at the expense of precision. e.g. Prune unconditional exception thrower edges for control flow graph analysis, to reduce memory footprint.				
Skip Huge Methods	Skip method analysis if length of its bytecode is too long (6,000).				

2.13 Implement SpotBugs plugin

2.13.1 Create Maven project

Use spotbugs-archetype to create Maven project. Then Maven archetype plugin will ask you to decide plugin's groupId, artifactId, package and initial version.

```
$ mvn archetype:generate \
      -DarchetypeArtifactId=spotbugs-archetype \
      -DarchetypeGroupId=com.github.spotbugs \
      -DarchetypeVersion=0.2.0
```

2.13.2 Write java code to represent bug to find

In generated project, you can find a file named as BadCase.java. Update this file to represent the target bug to find.

If you have multiple patterns to represent, add more classes into `src/test/java` directory.

2.13.3 Write test case to ensure your detector can find bug

In generated project, you can find another file named as MyDetectorTest.java. The `spotbugs.performAnalysis(Path)` in this test runs SpotBugs with your plugin, and return all found bugs (here 1st argument of this method is a path of class file compiled from `BadCase.java`).

You can use BugInstanceMatcher to verify that your plugin can find bug as expected.

Currently this test should fail, because we've not updated detector itself yet.

2.13.4 Write java code to avoid false-positive

To avoid false-positive, it is good to ensure that in which case detector should NOT find bug.

Update GoodCase.java in your project, and represent such cases. After that, add a test method into `MyDetectorTest.java` which verify that no bug found from this `GoodCase` class.

If you have multiple patterns to represent, add more classes into `src/test/java` directory.

2.13.5 Update detector to pass all unit tests

Now you have tests to ensure that your detector can work as expected.

Note: TBU

Which super class you should choose

AnnotationDetector Base detector which analyzes annotations on classes, fields, methods, and method parameters.

BytecodeScanningDetector Base detector which analyzes java bytecode in class files.

OpcodeStackDetector Sub class of `BytecodeScanningDetector`, which can scan the bytecode of a method and use an operand stack.

2.13.6 Update findbugs.xml

SpotBugs reads `findbugs.xml` in each plugin to find detectors and bugs. So when you add new detector, you need to add new `<Detector>` element like below:

```
<Detector class="com.github.plugin.MyDetector" reports="MY_BUG" speed="fast" />
```

It is also necessary to add `<BugPattern>`, to describe type and category of your bug pattern.

```
<BugPattern type="MY_BUG" category="CORRECTNESS" />
```

You can find `findbugs.xml` in `src/main/resources` directory of generated Maven project.

2.13.7 Update messages.xml

SpotBugs reads `messages.xml` in each plugin to construct human readable message to report detected bug. It also supports reading localized messages from `messages_ja.xml`, `messages_fr.xml` and so on.

You can find `messages.xml` in `src/main/resources` directory of generated Maven project.

Update message of Detector

In <Detector> element, you can add detector's description message. Note that it should be plain text, HTML is not supported.

```
<Detector class="com.github.plugin.MyDetector">
  <Details>
    Original detector to detect MY_BUG bug pattern.
  </Details>
</Detector>
```

Update message of Bug Pattern

In <BugPattern> element, you can add bug pattern's description message. There are three kinds of messages:

ShortDescription Short description for bug pattern. Useful to tell its intent and character for users. It should be plain text, HTML is not supported.

LongDescription Longer description for bug pattern. You can use placeholder like {0} (0-indexed), then added data into BugInstance will be inserted at there. So this LongDescription is useful to tell detailed information about detected bug.

It should be plain text, HTML is not supported.

Details Detailed description for bug pattern. It should be HTML format, so this is useful to tell detailed specs/examples with table, list and code snippets.

```
<BugPattern type="MY_BUG">
  <ShortDescription>Explain bug pattern shortly.</ShortDescription>
  <LongDescription>
    Explain existing problem in code, and how developer should improve their␣
↪implementation.
  </LongDescription>
  <Details>
    <![CDATA[
      <p>Explain existing problem in code, and how developer should improve their␣
↪implementation.</p>
    ]]>
  </Details>
</BugPattern>
```

2.14 SpotBugs FAQ

This document contains answers to frequently asked questions about SpotBugs. If you just want general information about SpotBugs, have a look at the manual.

2.14.1 Q1: I'm getting java.lang.UnsupportedClassVersionError when I try to run SpotBugs

SpotBugs requires JRE8 or later to run. If you use an earlier version, you will see an exception error message similar to the following:

> Exception in thread "main" java.lang.UnsupportedClassVersionError: edu/umd/cs/findbugs/gui/FindBugsFrame (Unsupported major.minor version 52.0)

The solution is to upgrade to JRE8 or later.

2.14.2 Q2: SpotBugs is running out of memory, or is taking a long time to finish

In general, SpotBugs requires lots of memory and a relatively fast CPU. For large applications, 1024M or more of heap space may be required.

By default, SpotBugs allocates 768M of heap space. You can increase this using the `-maxHeap n` option, where n is the number of megabytes of heap space to allocate.

2.14.3 Q3: What is the "auxiliary classpath"? Why should I specify it?

Many important facts about a Java class require information about the classes that it references. For example:

- What other classes and interfaces the class inherits from
- What exceptions can be thrown by methods in external classes and interfaces

The "auxiliary classpath" is a list of Jar files, directories, and class files containing classes that are used by the code you want SpotBugs to analyze, but should not themselves be analyzed by SpotBugs.

If SpotBugs doesn't have complete information about referenced classes, it will not be able to produce results that are as accurate as possible. For example, having a complete repository of referenced classes allows SpotBugs to prune control flow information so it can concentrate on paths through methods that are most likely to be feasible at runtime. Also, some bug detectors (such as the suspicious reference comparison detector) rely on being able to perform type inference, which requires complete type hierarchy information.

For these reasons, we strongly recommend that you completely specify the auxiliary classpath when you run SpotBugs. You can do this by using the `-auxclasspath` command line option, or the "Classpath entries" list in the GUI project editor dialog.

If SpotBugs cannot find a class referenced by your application, it will print out a message when the analysis completes, specifying the classes that were missing. You should modify the auxiliary classpath to specify how to find the missing classes, and then run SpotBugs again.

2.14.4 Q4: The Eclipse plugin doesn't load

The symptom of this problem is that Eclipse fails to load the SpotBugs UI plugin with the message:

> Plug-in "edu.umd.cs.findbugs.plugin.eclipse" was disabled due to missing or disabled prerequisite plug-in "org.eclipse.ui.ide"

The reason for this problem is that the Eclipse plugin distributed with SpotBugs does not work with older 3.x versions of Eclipse. Please use Eclipse Neon (version 4.6) or newer.

2.14.5 Q5: I'm getting a lot of false "OS" and "ODR" warnings

By default, SpotBugs assumes that any method invocation can throw an unchecked runtime exception. As a result, it may assume that an unchecked exception thrown out of the method could bypass a call to a `close()` method for a stream or database resource.

You can use the `-workHard` command line argument or the `findbugs.workHard` boolean analysis property to make SpotBugs work harder to prune unlikely exception edges. This generally reduces the number of false warnings, at the expense of slowing down the analysis.

2.14.6 Q6: The Eclipse plugin loads, but doesn't work correctly

- Make sure the Java code you trying to analyze is built properly and has no classpath or compile errors.
- Make sure the project and workspace SpotBugs settings are valid - in doubt, revert them to defaults.
- Make sure the Error log view does not show errors.

2.14.7 Q7: Where is the Maven plugin for SpotBugs?

The Maven Plugin for SpotBugs may be found here. Please note that the Maven plugin is not maintained by the SpotBugs developers, so we can't answer questions about it.

2.15 SpotBugs Links

This page contains links to related projects, including tools that are similar to SpotBugs.

2.15.1 SpotBugs Plugins

fb-contrib A FindBugs/SpotBugs plugin for doing static code analysis on java byte code.

Find Security Bugs A FindBugs/SpotBugs plugin for security audits of Java web applications.

findbugs-slf4j A FindBugs/SpotBugs plugin to verify usage of SLF4J.

2.15.2 Similar/Related Tools

FindBugs-IDEA The FindBugs plugin for IntelliJ IDEA.

sonar-findbugs A SonarQube plugin which provides rules based on SpotBugs and its major plugins.

Checkstyle A style checker for Java.

PMD An extensible cross-language static code analyzer.

huntbugs New Java bytecode static analyzer tool based on Procyon Compiler Tools aimed to supersede the FindBugs.

Google Error Prone A static analysis tool for Java that catches common programming mistakes at compile-time.

Checker Framework A pluggable type-checking for Java.

2.16 Bug descriptions

This document lists the standard bug patterns reported by SpotBugs.

2.16.1 Bad practice (BAD_PRACTICE)

Violations of recommended and essential coding practice. Examples include hash code and equals problems, cloneable idiom, dropped exceptions, Serializable problems, and misuse of finalize. We strive to make this analysis accurate, although some groups may not care about some of the bad practices.

CNT: Rough value of known constant found (CNT_ROUGH_CONSTANT_VALUE)

NP: Method with Boolean return type returns explicit null (NP_BOOLEAN_RETURN_NULL)

SW: Certain swing methods needs to be invoked in Swing thread (SW_SWING_METHODS_INVOKED_IN_SWING_THREAD)

FI: Finalizer only nulls fields (FI_FINALIZER_ONLY_NULLS_FIELDS)

FI: Finalizer nulls fields (FI_FINALIZER_NULLS_FIELDS)

UI: Usage of GetResource may be unsafe if class is extended (UI_INHERITANCE_UNSAFE_GETRESOURCE)

AM: Creates an empty zip file entry (AM_CREATES_EMPTY_ZIP_FILE_ENTRY)

AM: Creates an empty jar file entry (AM_CREATES_EMPTY_JAR_FILE_ENTRY)

IMSE: Dubious catching of IllegalMonitorStateException (IMSE_DONT_CATCH_IMSE)

CN: Class defines clone() but doesn't implement Cloneable (CN_IMPLEMENTS_CLONE_BUT_NOT_CLONEABLE)

CN: Class implements Cloneable but does not define or use clone method (CN_IDIOM)

CN: clone method does not call super.clone() (CN_IDIOM_NO_SUPER_CALL)

DE: Method might drop exception (DE_MIGHT_DROP)

DE: Method might ignore exception (DE_MIGHT_IGNORE)

Dm: Method invokes System.exit(...) (DM_EXIT)

Nm: Use of identifier that is a keyword in later versions of Java (NM_FUTURE_KEYWORD_USED_AS_IDENTIFIER)

Nm: Use of identifier that is a keyword in later versions of Java (NM_FUTURE_KEYWORD_USED_AS_MEMBER_IDENTIFIER)

JCIP: Fields of immutable classes should be final (JCIP_FIELD_ISNT_FINAL_IN_IMMUTABLE_CLASS)

Dm: Method invokes dangerous method runFinalizersOnExit (DM_RUN_FINALIZERS_ON_EXIT)

NP: equals() method does not check for null argument (NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT)

FI: Empty finalizer should be deleted (FI_EMPTY)

FI: Finalizer nullifies superclass finalizer (FI_NULLIFY_SUPER)

FI: Finalizer does nothing but call superclass finalizer (FI_USELESS)

FI: Finalizer does not call superclass finalizer (FI_MISSING_SUPER_CALL)

FI: Explicit invocation of finalizer (FI_EXPLICIT_INVOCATION)

Eq: Equals checks for incompatible operand (EQ_CHECK_FOR_OPERAND_NOT_COMPATIBLE_WITH_THIS)

Eq: equals method fails for subtypes (EQ_GETCLASS_AND_CLASS_CONSTANT)

Eq: Covariant equals() method defined (EQ_SELF_NO_OBJECT)

NP: Method with Optional return type returns explicit null (NP_OPTIONAL_RETURN_NULL)

NP: Non-null field is not initialized (NP_NONNULL_FIELD_NOT_INITIALIZED_IN_CONSTRUCTOR)

VR: Class makes reference to unresolvable class or method (VR_UNRESOLVABLE_REFERENCE)

IL: An apparent infinite loop (IL_INFINITE_LOOP)

IO: Doomed attempt to append to an object output stream (IO_APPENDING_TO_OBJECT_OUTPUT_STREAM)

IL: An apparent infinite recursive loop (IL_INFINITE_RECURSIVE_LOOP)

IL: A collection is added to itself (IL_CONTAINER_ADDED_TO_ITSELF)

RpC: Repeated conditional tests (RpC_REPEATED_CONDITIONAL_TEST)

FL: Method performs math using floating point precision (FL_MATH_USING_FLOAT_PRECISION)

CAA: Possibly incompatible element is stored in covariant array (CAA_COVARIANT_ARRAY_ELEMENT_STORE)

Dm: Useless/vacuous call to EasyMock method (DMI_VACUOUS_CALL_TO_EASYMOCK_METHOD)

Dm: Futile attempt to change max pool size of ScheduledThreadPoolExecutor (DMI_FUTILE_ATTEMPT_TO_CHANGE_MAXPOOL_SIZE_OF_SCHEDULED_THREAD_POOL_EXECUTOR)

DMI: BigDecimal constructed from double that isn't represented precisely (DMI_BIGDECIMAL_CONSTRUCTED_FROM_DOUBLE)

Dm: Creation of ScheduledThreadPoolExecutor with zero core threads (DMI_SCHEDULED_THREAD_POOL_EXECUTOR_WITH_ZERO_CORE_THREADS)

Dm: Can't use reflection to check for presence of annotation without runtime retention (DMI_ANNOTATION_IS_NOT_VISIBLE_TO_REFLECTION)

NP: Method does not check for null argument (NP_ARGUMENT_MIGHT_BE_NULL)

RV: Bad attempt to compute absolute value of signed random integer (RV_ABSOLUTE_VALUE_OF_RANDOM_INT)

RV: Bad attempt to compute absolute value of signed 32-bit hashcode (RV_ABSOLUTE_VALUE_OF_HASHCODE)

RV: Random value from 0 to 1 is coerced to the integer 0 (RV_01_TO_INT)

Dm: Incorrect combination of Math.max and Math.min (DM_INVALID_MIN_MAX)

Eq: equals method compares class names rather than class objects (EQ_COMPARING_CLASS_NAMES)

Eq: equals method always returns true (EQ_ALWAYS_TRUE)

Eq: equals method always returns false (EQ_ALWAYS_FALSE)

Eq: Equals method overrides equals in superclass and may not be symmetric (EQ_OVERRIDING_EQUALS_NOT_SYMMETRIC)

Eq: Covariant equals() method defined for enum (EQ_DONT_DEFINE_EQUALS_FOR_ENUM)

SKIPPED: Class too big for analysis (SKIPPED_CLASS_TOO_BIG)

TEST: Unknown bug pattern (UNKNOWN)

TEST: Testing (TESTING)

TEST: Testing 1 (TESTING1)

TEST: Testing 2 (TESTING2)

TEST: Testing 3 (TESTING3)

OBL: Method may fail to clean up stream or resource (OBL_UNSATISFIED_OBLIGATION)

OBL: Method may fail to clean up stream or resource on checked exception (OBL_UNSATISFIED_OBLIGATION_EXCEPTION_EDGE)

LG: Potential lost logger changes due to weak reference in OpenJDK (LG_LOST_LOGGER_DUE_TO_WEAK_REFERENCE)

2.16.4 Internationalization (I18N)

code flaws having to do with internationalization and locale

Dm: Consider using Locale parameterized version of invoked method (DM_CONVERT_CASE)

Dm: Reliance on default encoding (DM_DEFAULT_ENCODING)

2.16.5 Malicious code vulnerability (MALICIOUS_CODE)

code that is vulnerable to attacks from untrusted code

DP: Method invoked that should be only be invoked inside a doPrivileged block (DP_DO_INSIDE_DO_PRIVILEGED)

DP: Classloaders should only be created inside doPrivileged block (DP_CREATE_CLASSLOADER_INSIDE_DO_PRIVILEGED)

FI: Finalizer should be protected, not public (FI_PUBLIC_SHOULD_BE_PROTECTED)

MS: Public static method may expose internal representation by returning array (MS_EXPOSE_REP)

EI: May expose internal representation by returning reference to mutable object (EI_EXPOSE_REP)

EI2: May expose internal representation by incorporating reference to mutable object (EI_EXPOSE_REP2)

MS: May expose internal static state by storing a mutable object into a static field (EI_EXPOSE_STATIC_REP2)

MS: Field should be moved out of an interface and made package protected (MS_OOI_PKGPROTECT)

MS: Field should be both final and package protected (MS_FINAL_PKGPROTECT)

MS: Field isn't final but should be (MS_SHOULD_BE_FINAL)

MS: Field isn't final but should be refactored to be so (MS_SHOULD_BE_REFACTORED_TO_BE_FINAL)

MS: Field should be package protected (MS_PKGPROTECT)

MS: Field is a mutable Hashtable (MS_MUTABLE_HASHTABLE)

MS: Field is a mutable array (MS_MUTABLE_ARRAY)

MS: Field is a mutable collection (MS_MUTABLE_COLLECTION)

MS: Field is a mutable collection which should be package protected (MS_MUTABLE_COLLECTION_PKGPROTECT)

MS: Field isn't final and can't be protected from malicious code (MS_CANNOT_BE_FINAL)

2.16.6 Multithreaded correctness (MT_CORRECTNESS)

code flaws having to do with threads, locks, and volatiles

AT: Sequence of calls to concurrent abstraction may not be atomic (AT_OPERATION_SEQUENCE_ON_CONCURRENT_ABSTRACTION)

STCAL: Static Calendar field (STCAL_STATIC_CALENDAR_INSTANCE)

STCAL: Static DateFormat (STCAL_STATIC_SIMPLE_DATE_FORMAT_INSTANCE)

STCAL: Call to static Calendar (STCAL_INVOKE_ON_STATIC_CALENDAR_INSTANCE)

STCAL: Call to static DateFormat (STCAL_INVOKE_ON_STATIC_DATE_FORMAT_INSTANCE)

NP: Synchronize and null check on the same field. (NP_SYNC_AND_NULL_CHECK_FIELD)

VO: A volatile reference to an array doesn't treat the array elements as volatile (VO_VOLATILE_REFERENCE_TO_ARRAY)

VO: An increment to a volatile field isn't atomic (VO_VOLATILE_INCREMENT)

Dm: Monitor wait() called on Condition (DM_MONITOR_WAIT_ON_CONDITION)

Dm: A thread was created using the default empty run method (DM_USELESS_THREAD)

DC: Possible double check of field (DC_DOUBLECHECK)

DC: Possible exposure of partially initialized object (DC_PARTIALLY_CONSTRUCTED)

DL: Synchronization on interned String (DL_SYNCHRONIZATION_ON_SHARED_CONSTANT)

DL: Synchronization on Boolean (DL_SYNCHRONIZATION_ON_BOOLEAN)

DL: Synchronization on boxed primitive (DL_SYNCHRONIZATION_ON_BOXED_PRIMITIVE)

DL: Synchronization on boxed primitive values (DL_SYNCHRONIZATION_ON_UNSHARED_BOXED_PRIMITIVE)

WL: Synchronization on getClass rather than class literal (WL_USING_GETCLASS_RATHER_THAN_CLASS_LITERA

ESync: Empty synchronized block (ESync_EMPTY_SYNC)

MSF: Mutable servlet field (MSF_MUTABLE_SERVLET_FIELD)

IS: Inconsistent synchronization (IS2_INCONSISTENT_SYNC)

NN: Naked notify (NN_NAKED_NOTIFY)

Ru: Invokes run on a thread (did you mean to start it instead?) (RU_INVOKE_RUN)

SP: Method spins on field (SP_SPIN_ON_FIELD)

TLW: Wait with two locks held (TLW_TWO_LOCK_WAIT)

UW: Unconditional wait (UW_UNCOND_WAIT)

UG: Unsynchronized get method, synchronized set method (UG_SYNC_SET_UNSYNC_GET)

IS: Field not guarded against concurrent access (IS_FIELD_NOT_GUARDED)

ML: Synchronization on field in futile attempt to guard that field (ML_SYNC_ON_FIELD_TO_GUARD_CHANGING_THAT_FIELD)

NOISE: Bogus warning about a null pointer dereference (NOISE_NULL_DEREFERENCE)

NOISE: Bogus warning about a method call (NOISE_METHOD_CALL)

NOISE: Bogus warning about a field reference (NOISE_FIELD_REFERENCE)

NOISE: Bogus warning about an operation (NOISE_OPERATION)

2.16.8 Performance (PERFORMANCE)

code that is not necessarily incorrect but may be inefficient

HSC: Huge string constants is duplicated across multiple class files (HSC_HUGE_SHARED_STRING_CONSTANT)

Dm: The equals and hashCode methods of URL are blocking (DMI_BLOCKING_METHODS_ON_URL)

Dm: Maps and sets of URLs can be performance hogs (DMI_COLLECTION_OF_URLS)

Dm: Method invokes inefficient new String(String) constructor (DM_STRING_CTOR)

Dm: Method invokes inefficient new String() constructor (DM_STRING_VOID_CTOR)

Dm: Method invokes toString() method on a String (DM_STRING_TOSTRING)

Dm: Explicit garbage collection; extremely dubious except in benchmarking code (DM_GC)

Dm: Method invokes inefficient Boolean constructor; use Boolean.valueOf(...) instead (DM_BOOLEAN_CTOR)

Bx: Method invokes inefficient Number constructor; use static valueOf instead (DM_NUMBER_CTOR)

Bx: Method invokes inefficient floating-point Number constructor; use static valueOf instead (DM_FP_NUMBER_CTOR)

Bx: Method allocates a boxed primitive just to call toString (DM_BOXED_PRIMITIVE_TOSTRING)

Bx: Boxing/unboxing to parse a primitive (DM_BOXED_PRIMITIVE_FOR_PARSING)

Bx: Boxing a primitive to compare (DM_BOXED_PRIMITIVE_FOR_COMPARE)

Bx: Primitive value is unboxed and coerced for ternary operator (BX_UNBOXED_AND_COERCED_FOR_TERNARY_OPERATOR)

Bx: Boxed value is unboxed and then immediately reboxed (BX_UNBOXING_IMMEDIATELY_REBOXED)

Bx: Primitive value is boxed and then immediately unboxed (BX_BOXING_IMMEDIATELY_UNBOXED)

Bx: Primitive value is boxed then unboxed to perform primitive coercion (BX_BOXING_IMMEDIATELY_UNBOXED_TO_PERFORM_COERCION)

Dm: Method allocates an object, only to get the class object (DM_NEW_FOR_GETCLASS)

Dm: Use the nextInt method of Random rather than nextDouble to generate a random integer (DM_NEXTINT_VIA_NEXTDOUBLE)

SS: Unread field: should this field be static? (SS_SHOULD_BE_STATIC)

UuF: Unused field (UUF_UNUSED_FIELD)

UrF: Unread field (URF_UNREAD_FIELD)

SIC: Should be a static inner class (SIC_INNER_SHOULD_BE_STATIC)

SIC: Could be refactored into a static inner class (SIC_INNER_SHOULD_BE_STATIC_NEEDS_THIS)

SIC: Could be refactored into a named static inner class (SIC_INNER_SHOULD_BE_STATIC_ANON)

XSS: Servlet reflected cross site scripting vulnerability in error page (XSS_REQUEST_PARAMETER_TO_SEND_ERROR)

XSS: Servlet reflected cross site scripting vulnerability (XSS_REQUEST_PARAMETER_TO_SERVLET_WRITER)

XSS: JSP reflected cross site scripting vulnerability (XSS_REQUEST_PARAMETER_TO_JSP_WRITER)

HRS: HTTP Response splitting vulnerability (HRS_REQUEST_PARAMETER_TO_HTTP_HEADER)

HRS: HTTP cookie formed from untrusted input (HRS_REQUEST_PARAMETER_TO_COOKIE)

PT: Absolute path traversal in servlet (PT_ABSOLUTE_PATH_TRAVERSAL)

PT: Relative path traversal in servlet (PT_RELATIVE_PATH_TRAVERSAL)

Dm: Hardcoded constant database password (DMI_CONSTANT_DB_PASSWORD)

Dm: Empty database password (DMI_EMPTY_DB_PASSWORD)

SQL: Nonconstant string passed to execute or addBatch method on an SQL statement (SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE)

SQL: A prepared statement is generated from a nonconstant String (SQL_PREPARED_STATEMENT_GENERATED_FROM_NONCONSTANT_STRING)

2.16.10 Dodgy code (STYLE)

code that is confusing, anomalous, or written in a way that leads itself to errors. Examples include dead local stores, switch fall through, unconfirmed casts, and redundant null check of value known to be null. More false positives accepted. In previous versions of SpotBugs, this category was known as Style.

CAA: Covariant array assignment to a field (CAA_COVARIANT_ARRAY_FIELD)

CAA: Covariant array is returned from the method (CAA_COVARIANT_ARRAY_RETURN)

CAA: Covariant array assignment to a local variable (CAA_COVARIANT_ARRAY_LOCAL)

Dm: Call to unsupported method (DMI_UNSUPPORTED_METHOD)

Dm: Thread passed where Runnable expected (DMI_THREAD_PASSED_WHERE_RUNNABLE_EXPECTED)

NP: Dereference of the result of readLine() without nullcheck (NP_DEREFERENCE_OF_READLINE_VALUE)

NP: Immediate dereference of the result of readLine() (NP_IMMEDIATE_DEREFERENCE_OF_READLINE)

RV: Remainder of 32-bit signed random integer (RV_REM_OF_RANDOM_INT)

RV: Remainder of hashCode could be negative (RV_REM_OF_HASHCODE)

Eq: Unusual equals method (EQ_UNUSUAL)

Eq: Class doesn't override equals in superclass (EQ_DOESNT_OVERRIDE_EQUALS)

NS: Questionable use of non-short-circuit logic (NS_NON_SHORT_CIRCUIT)

NS: Potentially dangerous use of non-short-circuit logic (NS_DANGEROUS_NON_SHORT_CIRCUIT)

IC: Initialization circularity (IC_INIT_CIRCULARITY)

IA: Potentially ambiguous invocation of either an inherited or outer method (IA_AMBIGUOUS_INVOCATION_OF_INHERITED_OR_OUTER_METHOD)

Se: Private readResolve method not inherited by subclasses (SE_PRIVATE_READ_RESOLVE_NOT_INHERITED)

Se: Transient field of class that isn't Serializable. (SE_TRANSIENT_FIELD_OF_NONSERIALIZABLE_CLASS)

SF: Switch statement found where one case falls through to the next case (SF_SWITCH_FALLTHROUGH)

SF: Switch statement found where default case is missing (SF_SWITCH_NO_DEFAULT)

UuF: Unused public or protected field (UUF_UNUSED_PUBLIC_OR_PROTECTED_FIELD)

UrF: Unread public/protected field (URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD)

QF: Complicated, subtle or wrong increment in for-loop (QF_QUESTIONABLE_FOR_LOOP)

NP: Read of unwritten public or protected field (NP_UNWRITTEN_PUBLIC_OR_PROTECTED_FIELD)

UwF: Field not initialized in constructor but dereferenced without null check (UWF_FIELD_NOT_INITIALIZED_IN_CONSTRUCTOR)

UwF: Unwritten public or protected field (UWF_UNWRITTEN_PUBLIC_OR_PROTECTED_FIELD)

UC: Useless control flow to next line (UCF_USELESS_CONTROL_FLOW_NEXT_LINE)

UC: Condition has no effect (UC_USELESS_CONDITION)

```
<!-- for Maven -->
<dependency>
  <groupId>com.github.spotbugs</groupId>
  <artifactId>spotbugs</artifactId>
  <version>3.1.3</version>
</dependency>
```

```
// for Gradle
compile 'com.github.spotbugs:spotbugs:3.1.3'
```

2.17.2 com.google.code.findbugs:jsr305

JSR305 is already Dormant status, so SpotBugs does not release `jsr305` jar file. Please continue using findbugs' one.

2.17.3 com.google.code.findbugs:findbugs-annotations

Please depend on `spotbugs-annotations` instead.

```
<!-- for Maven -->
<dependency>
  <groupId>com.github.spotbugs</groupId>
  <artifactId>spotbugs-annotations</artifactId>
  <version>3.1.3</version>
  <optional>true</optional>
</dependency>
```

```
// for Gradle
compileOnly 'com.github.spotbugs:spotbugs-annotations:3.1.3'
```

2.17.4 com.google.code.findbugs:annotations

Please depend on both of `spotbugs-annotations` and `net.jcip:jcip-annotations:1.0` instead.

```
<!-- for Maven -->
<dependency>
  <groupId>net.jcip</groupId>
  <artifactId>jcip-annotations</artifactId>
  <version>1.0</version>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>com.github.spotbugs</groupId>
  <artifactId>spotbugs-annotations</artifactId>
  <version>3.1.3</version>
  <optional>true</optional>
</dependency>
```

```
// for Gradle
compileOnly 'net.jcip:jcip-annotations:1.0'
compileOnly 'com.github.spotbugs:spotbugs-annotations:3.1.3'
```

2.17.5 FindBugs Ant task

Please replace `findbugs-ant.jar` with `spotbugs-ant.jar`.

```
<taskdef
  resource="edu/umd/cs/findbugs/anttask/tasks.properties"
  classpath="path/to/spotbugs-ant.jar" />
<property name="spotbugs.home" value="/path/to/spotbugs/home" />

<target name="spotbugs" depends="jar">
  <spotbugs home="${spotbugs.home}"
            output="xml"
            outputFile="bcel-fb.xml" >
    <auxClasspath path="${basedir}/lib/Regex.jar" />
    <sourcePath path="${basedir}/src/java" />
    <class location="${basedir}/bin/bcel.jar" />
  </spotbugs>
</target>
```

2.17.6 FindBugs Maven plugin

Please use *com.github.spotbugs:spotbugs-maven-plugin* instead of *org.codehaus.mojo:findbugs-maven-plugin*.

```
<plugin>
  <groupId>com.github.spotbugs</groupId>
  <artifactId>spotbugs-maven-plugin</artifactId>
  <version>3.1.3</version>
  <dependencies>
    <!-- overwrite dependency on spotbugs if you want to specify the version of␣
↪spotbugs -->
    <dependency>
      <groupId>com.github.spotbugs</groupId>
      <artifactId>spotbugs</artifactId>
      <version>3.1.3</version>
    </dependency>
  </dependencies>
</plugin>
```

2.17.7 FindBugs Gradle plugin

Please use spotbugs plugin found on https://plugins.gradle.org/plugin/com.github.spotbugs

```
plugins {
  id  'com.github.spotbugs' version '1.6.1'
}
spotbugs {
  toolVersion = '3.1.3'
}

// To generate an HTML report instead of XML
tasks.withType(com.github.spotbugs.SpotBugsTask) {
  reports {
    xml.enabled = false
    html.enabled = true
```

(continues on next page)

(continued from previous page)

```
    }
}
```

2.17.8 FindBugs Eclipse plugin

Please use following update site instead.

- https://spotbugs.github.io/eclipse/ (to use stable version)
- https://spotbugs.github.io/eclipse-candidate/ (to use candidate version)
- https://spotbugs.github.io/eclipse-latest/ (to use latest build)