SOEN 7481 Software Verification and Testing

Fall 2018

1 Assignment Description.

In this assignment, you will need to implement a static analysis tool to detect bugs in source code. Static analysis tools are commonly used as part of the continuous integration process to ensure the quality of the code (more details in the slides and the papers in week 3). The tool will be implemented in Java. There are two tools that you may use to help you with analyzing the code:

- Eclipse JDT, an Abstract Syntax Tree (AST) parser that is part of the Eclipse IDE.
- JavaParser, an open source AST parser for Java.

2 Bug Patterns.

You will need to implement a total of 10 bug patterns. The descriptions of the bug patterns are listed below:

1. *Class defines equals() but not hashCode().* This class overrides equals(Object), but does not override hashCode(). Therefore, the class may violate the invariant that equal objects must have equal hashcodes. **From FindBugs**[1]
2. *Comparison of String objects using == or !=.* This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by two different String objects. Consider using the equals(Object) method instead. **From FindBugs**[2]
3. *Method may fail to close stream on exception.* The method creates an IO stream object, does not assign it to any fields, pass it to other methods, or return it, and does not appear to close it on all possible exception paths out of the method. This may result in a file descriptor leak. It is generally a good idea to use a finally block to ensure that streams are closed. **From FindBugs**[3]
4. *Condition has no effect.* This condition always produces the same result as the value of the involved variable was narrowed before. Namely, the condition or the boolean variable in if/while always returns either true or false. Probably something else was meant or condition can be removed. **From FindBugs**[4]

   For example:

   ```
   if (true) {
     ...
   }
   ```

   or

   ```
   boolean var = false;
   ...
   if (var) {
     ...
   }
   ```

[1] http://findbugs.sourceforge.net/bugDescriptions.html#HE_EQUALS_NO_HASHCODE
[2] http://findbugs.sourceforge.net/bugDescriptions.html#ES_COMPARING_STRINGS_WITH_EQ
[3] http://findbugs.sourceforge.net/bugDescriptions.html#OS_OPEN_STREAM_EXCEPTION_PATH
[4] http://findbugs.sourceforge.net/bugDescriptions.html#UC_USELESS_CONDITION

5. *Inadequate logging information in catch blocks.* Developers usually rely on logs for error diagnostics when exceptions occur. However, sometimes, duplicate logging statements in different *catch* blocks of the same *try* block may cause debugging difficulties since the logs fail to tell which exception occurred.

   For example:

   ```
   ...
   } catch (AlreadyClosedException closedException) {
       s_logger.warn("Connection to AMQP service is lost.");
   } catch (ConnectException connectException) {
       s_logger.warn("Connection to AMQP service is lost.");
   }
   ...
   ```

6. *Unneeded computation in loops.* There may be unneeded computation in loops, where you call a method inside a loop, but the return value is never used.

   For example:

   ```
   ...
   for (int i = 0; i < 1000; i++){
     Result r = computeResult(i); // unneeded computation
     doSomething(i);
   }
   ...
   ```

7. *Unused methods.* Detect both *private* and *public* methods that are not referenced/called anywhere in the code.

8. *Empty exception.* There is no debug message when an exception occurs, which may cause debugging difficulties.[5]

9. *Unfinished exception handling code.* There is a comment such as TODO or FIXME in the *catch* block of exceptions.[5]

10. *Over-catching an exception with system-termination.* Developers are over-catching an exception (i.e., catching very high-level exceptions, such as *Exception* or *RunTimeException*), and are calling *abort* or *System.exit()* in the *catch* block.[5]

Note that, you can use either FindBugs or SpotBugs, since these two are almost identical (SpotBugs is successor of FindBugs).

3 Testing your code.

You will need to design test cases for your code. For each bug pattern, you need to design some test cases, and more importantly, *you need to describe your design decision in the final report.* You may use any testing framework (e.g., JUnit or TestNG) for your test case. There is no right or wrong answer in terms of test case design, but you must provide a valid reason for your design.

4 Discussing your detection results.

You need to run your tool on two systems: Hadoop 3.0.0[6] and CloudStack 4.9[7].

[5] From https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf

[6] https://github.com/apache/hadoop/tree/branch-3.0.0

[7] https://github.com/apache/cloudstack/tree/4.9

For bug pattern 1–4, you need to compare your detection results with that of FindBugs (since FindBugs is also able to detect these bugs). Whenever there is any difference in the detection result, you need to include a discussion in your final report regarding why the difference exist.

5 Final report.

The final report must be in ACM conference format (i.e., double column). You can find an example here. You can also find the template file on the course website. Note that you have to follow the template; otherwise, you will lose marks in the assignment.

The final report should contain five or more pages. It needs to include (at least): abstract, introduction, detection approach, design of the test cases, detection result and discussion, and conclusion. You must include the link to your GitHub repository in the report.

The final format of the report must be in *.pdf* format.

6 Hosting code and report on GitHub.

Everything needs to be stored on GitHub, including all of your code, test case, and report. The report should preferable be written in Latex. However, if you struggle with Latex, you may also use other tools (e.g., Google Doc), as long as it can show the edit/version history.

7 Submitting your assignment.

Please submit your assignment by emailing the pdf file to:

SOEN7481Concordia@gmail.com

The subject of the email must be: **[SOEN7481][Assignment][name1, name2, name3]**

8 Evaluation

The final evaluation will be based on all factors in the deliverables. We will look at code quality, design of the test case, quality of the report, and detection result and discussion.