

1 Static Analysis Warnings Collection

In this section, we describe the study setup utilized to collect the data from each SAT and the data collection process. We analyzed a single snapshot of each project, namely the release available in the dataset of each of the 112 available systems.

Checkstyle. The *JAR* file for the Checkstyle analysis was downloaded directly from Checkstyle’s website¹

to engage the analysis from the command line. The executable *JAR* file used in this case was *checkstyle-8.30-all.jar*. In addition to downloading the *JAR* executable, Checkstyle offers two different types of rule sets for the analysis¹. For each of the rule sets, the configuration file was downloaded directly from Checkstyle’s website². To start the analysis, the files *checkstyle-8.30-all.jar* and the configuration file in question were saved in the directory where all the cloned repositories from Java Qualitas Corpus resided. A bash script was made to execute the analysis for each project in one go to make the analysis more swift.

This can be seen in Listing 1.

```
1  #!/bin/bash
2  while read in; do java -jar checkstyle-8.30-all.jar
3  -c /RULESET -f xml "$in"/ > "$in"_CS_RULESET.xml ;
4  done < projectList.txt"
```

Listing 1: Checkstyle bash script tailored towards each rule set

where RULESET represents the rule set used for the analysis, “\$in” represents the project name which is imported from *projectList.txt*, and “\$in”_CS_RULESET.xml represents the export file name of the analysis results in XML format. The text file *projectList.txt* consists of all the project names to execute the analysis for all projects in one go.

Listing 2 show an example of how the projects were analyzed with Checkstyle according to the rule set Google Java Style³.

```
1  #!/bin/bash
2  while read in; do java -jar checkstyle-8.30-all.jar
3  -c /google_checks.xml -f xml "$in"/ >
4  "$in"_CS_Google_Checks.xml ;
5  done < projectList.txt"
```

Listing 2: Example of Checkstyle bash script for Google Java Style configuration.

¹<https://checkstyle.org/>

²<https://github.com/checkstyle/checkstyle/tree/master/src/main/resources>

³https://github.com/checkstyle/checkstyle/blob/master/src/main/resources/google_checks.xml

Findbugs. FindBugs 3.0.1 was installed by running *brew install findbugs* in the command line. Once installed, the GUI was engaged by writing *spotbugs*. From the GUI, the analysis was executed through File → New Project. The classpath for the analysis was identified to be the location of the project directory. Moreover, the source directories were identified as the project *JAR* executables. Once the classpath and source directories were identified, the analysis was engaged by clicking Analyze in the GUI. Once the analysis finished, the results were saved through File → Save as using the XML file format.

PMD. PMD 6.23.0 was downloaded from GitHub⁴ as a zip file. After unzipping, the analysis was engaged by identifying several parameters: project directory, export file format, rule set, and export file name. In addition to downloading the zip file, PMD offers 32 different types of rule sets for Java written projects⁵. To make the analysis swifter, a bash script was made to engage the analysis for each project in one go. This can be seen in Listing 3.

```
1  #!/bin/bash
2  while read in;
3  do $HOME/pmd-bin-6.23.0/bin/run.sh pmd -dir "$in"/
4  -f xml -R rulesets/java/RULESET
5  -reportfile "$in"_PMD_RULESET.xml;
6  done < projectList.txt"
```

Listing 3: PMD bash script tailored towards each rule set.

where “*\$HOME*” represents the full path where the binary resides, “*\$in*” represents the project name which is imported from *projectList.txt*, *RULESET* represents the rule set used for the analysis, and “*\$in*”_PMD_RULESET.xml represents the export file name of the analysis results in XML format. Like in Listing 1, *projectList.txt* consists of all the project names. An example of how the projects were analyzed with PMD according to the rule set Clone Implementation⁶ is shown in Listing 4.

```
1  #!/bin/bash
2  while read in;
3  do $HOME/pmd-bin-6.23.0/bin/run.sh pmd -dir "$in"/
4  -f xml -R rulesets/java/clone.xml
5  -reportfile "$in"_PMD_Clone.xml;
6  done < projectList.txt"
```

Listing 4: Example of PMD bash script for Clone Implementation configuration.

SonarQube. We first installed SonarQube LTS 6.7.7 on a private server with 128 GB RAM and 4 processors. Because of the limitations of the open-source version of SonarQube, we are allowed to use only one core, therefore more

⁴[https://github.com/pmd/pmd/releases/download/pmd\\$_\\$releases%2F6.23.0/pmd-bin-6.23.0.zip](https://github.com/pmd/pmd/releases/download/pmd$_$releases%2F6.23.0/pmd-bin-6.23.0.zip)

⁵<https://github.com/pmd/pmd/tree/master/pmd-java/src/main/resources/rulesets/java>

⁶<https://github.com/pmd/pmd/blob/master/pmd-java/src/main/resources/rulesets/java/clone.xml>

cores would have not been beneficial for our scope. We adopted the LTS version (Long-Time Support) since it is the most stable and best-supported version. We executed SonarQube on each project using SDK 8 (Oracle) and the *sonar-scanner* package version 4.2. Each project was analyzed using the sources and the binaries provided in the dataset. Moreover, we configured the analysis (in the file *sonar-project.properties*) reporting information regarding the project key, project name, project version, source directories, test directories, binary directories, and library directories. It is important to note that the analysis performed using the original binaries reduced the compilation errors and missing libraries. Moreover, it also helped to reduce issues related to false positives⁷. After all the projects had been analyzed, we extracted the data related to the projects using the “*SonarQube Exporter*” tool⁸. “*SonarQube Exporter*” makes it possible to extract SonarQube project measures and issues in CSV format.

2 Architectural Smells Collection

AS were detected on a Windows machine with 4 cores and 24 GB of RAM. The entire Qualitas Corpus dataset was analyzed using ARCAN in less than 24 hours. The tool is freely available and easy to install and use⁹. We should notice that for subsystem (component) we mean a set of packages and classes which identifies an independent unit of the system responsible for a certain functionality.

⁷<https://docs.sonarqube.org/latest/analysis/languages/java/>

⁸<https://github.com/pervcomp/SonarQube-Exporter/releases>

⁹<http://essere.disco.unimib.it/wiki/arcan>