Assignment 2: Evaluating the Design of an Open-Source Project

Software Design & Modeling

Due date: 2022-11-10 at 23:00

1 The assignment

Your assignment in a nutshell:

1. Pick a software **project** hosted on GitHub. The project should have:
   - at least 100 stars,
   - at least 100 forks,
   - at least 10 open issues, and
   - at least 100,000 lines of code.
2. Run some **flaw detection** tools on the project. On top of the analysis rules provided by the tools, you should **implement** and run at least **1 new rule** that you think can be useful for your analysis.
3. Using the tools' output to guide you, **inspect** the parts of the project whose design is more questionable.
4. Write down your findings in a **report**.

   Maximum length of the report: 10 pages (A4 with readable formatting) including any pictures and code snippets.

The assignment must be done in *pairs* of students;[1] student pairs must be different in each assignment; in other words, you should not work on this assignment with the same person you

[1] If there is an odd number of students, one group will be made of three students.

worked with for Assignment 1. Every group member should contribute equally to the work; the individual grading will also reflect this.

Once you have identified your partner for the assignment, write it in this spreadsheet **within 3 days** after this assignment is published. If you cannot find another student to work with within 3 days, the instructors will pair you up with some other available students.

This assignment contributes to **20%** of your overall grade in the course.

2 Tools and other resources

2.1 How to find projects on GitHub

See the instructions in Assignment 1.

2.2 Tools to detect design flaws

You should use at least one flaw detection tool, but you can also use more than one. If you select a single tool, you should explore different options and rules offered by the tool. Usually, a tool's analysis checks a fixed set of basic rules by default, but you can select additional rules (if you think they are more useful for the project you are analyzing) or disable some default rules (if you think they are irrelevant or ineffective for the project). If you use multiple tools, you should compare the results of running them on the same project. In case you decide to use multiple tools, you only have to implement the new rule that is required by this assignment for one of the tools.

We suggest two tools that can detect a variety of design and coding flaws and bad practices:

PMD: `https://pmd.github.io/`

SonarQube: `https://www.sonarqube.org/`

How to use PMD: Basic usage (from PMD's `bin` directory):

```
$ ./run.sh pmd -d /path/to/project/root/sources/                ## project path
               -f html                                          ## output format
               -rulesets java-basic,java-design,java-unusedcode ## analysis rules
               > /path/to/output/report.html                    ## output filename
```

For more options (and what each rule does) see the online documentation or run `./run.sh pmd -h`. PMD only requires source files to run.

Adding custom rules to PMD. You can extend PMD by writing your own rules. Documentation about how to do this is available in PMD's website (under *User Documentation* → *Extending PMD* → *Introduction to writing rukes*)[2].

[2] `https://pmd.sourceforge.io/pmd-6.50.0/pmd_userdocs_extending_writing_rules_intro.html`

How to use SonarQube: Download SonarQube community edition, as well as SonarQube Scanner[3] for your computer architecture. Basic usage:

1. From SonarQube's `bin` subdirectory for your computer architecture (such as `bin/macosx-universal-64`) launch the console:

   ```
   $ ./sonar.sh console
   ```

   and wait until the message "`SonarQube is up`" appears on screen.

2. In a different terminal (or after switching the console's process to the background):

   a) Add SonarQube Scanner's `bin` directory to the path

   ```
   $ export PATH="$PATH:$HOME/sonar-scanner/bin"
   ```

   b) Go the the project's root source directory

   c) Run the analysis by calling:

   ```
   sonar-scanner -DprojectKey=projectName
   ```

   where `projectName` is any identifier that will be used in the report to identify the project.[4]

3. Once the analysis terminates, open a browser to `http://localhost:9000` to inspect the results.

SonarQube requires source files as well as class files for Java projects.

Adding custom rules to SonarQube. You can extend SonarQube by writing your own rules. Documentation about how to do this is available in SonarQube's website.[5]

Other languages? PMD and SonarQube support multiple source languages. However, some analyses may not be available for a certain language. If you feel adventurous, or simply prefer to work with languages other than Java, you can apply PMD and SonarQube to projects written in a language other than Java, or even look for similar analysis tools that work for other languages. For example, there exists ReSharper for the .NET languages (https://www.jetbrains.com/resharper/) and Cppcheck for C++ (http://cppcheck.sourceforge.net/). If you choose to use other tools and languages, please check with the instructors that your choice of tools and language is reasonable before working on the assignment.

[3] At https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/.

[4] If you need to run the tool multiple times with custom options, you can also create a text file named `sonar-project.properties` with the various command-line options.

[5] https://docs.sonarqube.org/latest/extend/adding-coding-rules/

2.3 Custom rules

The assignment also requires you to write at least 1 custom rule, and then to check it on the project (together with the tool's other predefined rules).

The custom rule doesn't need to be particularly complex, and can reuse as much as possible the information that is already collected by the tools. For example, you can introduce variants of some of the rules that are built-in the tools. Another example is computing some source-code metrics and identifying a threshold for the metrics: when the metrics is above (or below) the threshold, the tool reports a rule violation.

3 What to write in the report

3.1 Report content

The structure of the report is free; it should include all the significant findings emerged during your work.

Things to include in the report's discussion:

- the project selection process, including mentioning projects you discarded;
- a short description of the selected project with some stats (size, number of contributors, whether it's a library or application, and so on);
- whether you analyzed all project or only some parts of it (e.g., did you analyze testing code as well?);
- a discussion of what predefined rules you selected for your project analysis and why;
- a quantitative summary of the tools' output, such as a table with how many problems have been found, of which kind, by what tool, and so on;
- whether you found any false positives in the tools' output, that is reports of rule violations that should not actually be considered violations;
- a discussion of how you chose the custom rule to implement, whether it was hard or easy to implement, and what checking this rule tells you about the project's design quality;
- a qualitative discussion of the kinds of problems the tools are more or less likely to report;
- your assessment of the project's design quality based on the tools' analysis combined with your own judgement;
- a brief (possibly speculative) discussion about whether your findings are likely to be applicable to other projects or, conversely, they are probably unique to the project you selected – and why you think this to be the case.

3.2 Tips and tricks

Do not trust the tool's output blindly; even when the tool's results are sound, try to understand the origin of the problems to come up with higher-level and more interesting findings.

4 How and what to turn in

Turn in your **report** as a single PDF file using *iCorsi* under *Assignment 2*.