Assignment 1: Documenting Design Patterns

Software Design & Modeling

Due date: 2022-10-28 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 50,000 lines of code.
2. Run a **design pattern detection** tool on the project.

   Using the tool's output to guide you, **inspect** the main applications of design patterns in the project.
3. Write down your findings in a **report**.

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

The assignment must be done in *pairs* of students;[1] student pairs must be different in each assignment. 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 7 days** after this assignment is published. If you cannot find another student to work with within 7 days, the instructors will pair you up with some other available students.

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

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

2 Tools and other resources

2.1 How to find projects on GitHub

Query GitHub using its API for projects with the required features. To list projects written in Java with at least 100 forks, 100 stars, 10 open issues use the URL:

```
https://api.github.com/search/repositories?q=language:Java&forks_count:
      >=100&stargazers_count:>=100&open_issues_count:>=10&page=1
```

This query lists the first page of results; replace `page=1` with `page=2`, `page=3`, and so on to browse all results.

You can add a constraint on *file size* by adding constraints of the form `&size:>100` (which searches for file at least 100 bytes large). There is no direct way to query for lines of code, but you can get an approximate count the **lines of code** of a given repository, without cloning the project, by summing up the count for each project contributor. You can use the following JavaScript snippet:[2]

```
// replace 'jquery/jquery' with the repository you're interested in
fetch('https://api.github.com/repos/jquery/jquery/stats/contributors')
    .then(response => response.json())
    .then(contributors => contributors
        .map(contributor => contributor.weeks
            .reduce((lineCount, week) => lineCount + week.a - week.d, 0)))
    .then(lineCounts => lineCounts.reduce((lineTotal, lineCount) => lineTotal + lineCount))
    .then(lines => window.alert(lines));
```

To run it, you can use the Chrome or Chromium browser: go to *More tools* $\longrightarrow$ *Developer tools*, open the *Console*, and paste the snippet. The answer will appear in a pop-up window.

Notice that this way of counting lines of code is imperfect as it only targets the default branch and may double count code that is merged.[3] Thus, remember to do a final sanity check, using tools such as `cloc`,[4] once you select and clone the project.

2.2 How to detect design patterns

We suggest to use `pattern4j` as pattern detection tool:

```
https://users.encs.concordia.ca/~nikolaos/pattern_detection.html
```

The tool works on Java projects, which means that you have to select a Java project on GitHub to use `pattern4j`.

[2] For example, to analyze GitHub project `https://github.com/sosy-lab/java-smt` replace `jquery/jquery` with `sosy-lab/java-smt`.

[3] See this Stack Overflow thread `https://bit.ly/2DgOA86`.

[4] `https://github.com/AlDanial/cloc`

How to use the tool: 1. compile the project (the tool works on `class` files), 2. launch the tool (`java -jar pattern4.jar`), 3. select the project root (where the `class` files are), 4. select which patterns to detect, 5. inspect the results.

While the tool itself runs with the latest JRE, it may occasionally fail analyzing bytecode generated by recent JDKs. If you run into problems (for example, the tool runs but fails to produce any meaningful output), try compiling your project with an older version of Java, and then run the pattern detection tool with the same JRE version as the JDK used to compile the project. If all else fails, you may want to try another project.

Other languages? If you feel adventurous, or simply prefer to work with languages other than Java, you can look for pattern detection tools for languages other than Java and use them to do the assignment targeting another language (which must however be object-oriented). If you choose to do so, please check with the instructors that your choice of tools and language is reasonable before working on the assignment.

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;
- if you initially selected a project (or projects) that turned out to have no design pattern;
- a short description of the selected project with some stats (size, number of contributors, whether it's a library or application, and so on);
- an estimate of the accuracy[5] (true vs. false positives, and true vs. false negatives) of the pattern detection tool;
- a quantitative summary of the findings, such as a table with the patterns that have been found, how many classes of the project they cover, how many applications of each patterns you found, and so on;
- a qualitative discussion of some concrete examples (with snippets of code or other project artifacts) that you think are interesting;
- whether you found patterns that are variations of the classic patterns we have seen in class (or in textbooks), and conversely how many verbatim applications of patterns you found;

[5] To be estimated by manually inspecting the tool's output. For a definition of false positive and false negative see `https://en.wikipedia.org/wiki/False_positives_and_false_negatives`.

- 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

- Looking for projects that are very active (frequent commits in recent months) may make your job easier and more interesting.
- Do not just look at the code but also at the documentation (README, JavaDoc, anything else) and other artifacts (such as test cases); they all may be referring to patterns.
- Do not trust the tool's output blindly; even when the tool's results are sound, try to add your own judgment on top of them 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 1*.