Assignment 1
Finding and documenting design patterns

Software Design & Modeling

Submit by: Friday, 25 October 2024 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.[1]
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: 7 pages (A4 with readable formatting) including pictures and code snippets.

The assignment must be done in *individually*.
This assignment contributes to **28%** of your overall grade in the course.

[1] Comments count as lines of code, whereas empty lines do not.

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. As alternative, use this browser extension `https://chrome.google.com/webstore/detail/github-gloc/kaodcnpebhdbpaeeemkiobcokcnegdki?hl=en` to get an approximate count of lines of code.

Notice that both ways of counting lines of code are imperfect (for example, the JavaScript snippet[3] only considers the main branch and may double count code that is merged). 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 -XX:MaxRAMFraction=1 -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 should run 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, or it crashes with an "`Exception in thread JWT-EventQueue-0 java.lang.IllegalArgumentException`") try the following:

- Run the detection tool using an *old* version of Java (Java 8 is recommended);
- Make sure that the analyzed project is pure Java (no other JVM languages such as Scala or Kotlin are part of the codebase);
- Try compiling your project with an older version of Java: ideally, the same version as the JRE version used to run the detection tool; if this is not feasible, use the oldest version of the Java compiler that can 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; notice that choosing a suitable project, with a nontrivial usage of design patterns, is an integral part of the assignment's work
- 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 (true vs. false positives, and true vs. false negatives)[5] of the pattern detection tool;

[5] You can estimate the fraction of false positive by manually inspecting the tool's output (or a smaller sample of the output, if the tool reports many instances of patterns). False negatives are more elusive to detect manually, but you can try to comment whether you would have expected to see instances of some design pattern, given the nature of the project you are analyzing, that the tool did not find. For a definition of false positive and false negative see https://en.wikipedia.org/wiki/False_positives_and_false_negatives.

- 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 a few concrete examples (with snippets of code or other project artifacts) that you think are interesting; for example, 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;
- 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 given its nature – 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.*

4.1 Plagiarism policy

Students are allowed to generally discuss assignments and solutions among them, but each student must work on and write down their assignment independent of others. In particular, both sharing solutions of assignments among students and reusing solutions of assignments done by students of previous years or by anyone else are not allowed and constitute cheating.

In a similar vein, you are allowed, in fact encouraged, to loop up any examples and material that is available online you find useful; however, you are required to write down the solution completely on your own, and, if there is publicly available material (such as a website, blog, paper, or book chapter) that you based your work on, credit it in the report (explaining what is similar and how your work differs).

ChatGPT & Co.

The plagiarism policy also applies to AI tools such as ChatGPT or CoPilot:

1. You are allowed to get the help of such tools; however, you remain entirely responsible for the solution that you submit.

2. If you use any such tools, you must add a section to the report that summarizes which tool(s) you used and for what tasks, what kinds of prompts you used with the tools, how you checked the correctness and completeness of their suggestions, and what modifications (if any) you introduced on top of the tool's output.

Failure to abide by these rules, including failing to disclose using AI tools, will be considered plagiarism.