Part VI: Static Analyzers

Introduction to Static Analysis

What is a static analysis? Usually, static analysis means code review. In companies, especially in one project, colleagues will review your code manually.

- In one word, static analysis is mostly informal manual-human reviews of code.
- Also called "code reviews" or "compile-time analysis".

During our experience in `JSoup` project, we fork our project into our own repository and pull request into the master. One or two of our collaborators will peer view the PR code, give it some advice or directly merge it into the master.

There are best practices for the process of static analyzers.

1. Review small portions of code at a time
2. Record all feedback
3. Review code independently before gathering to discuss
4. Use checklists

Importance of Static Analysis

After talking about the basic concepts of static analysis, we come to the question why we need static analysis. Combined with its basic concepts, we concluded reasons of necessity of static analysis.

- Static analysis can help you find potential bugs early. Using static analysis, your code reviewers can help you find these bugs manually.
- Static analysis can help you stick to the same coding style or coding standard. Since more than one people are involved in one function or one line of code, the clarity of code will increase. In my opinion, to raise the possibility of merging code, you are not the only one who sees the code so that you need to make more comments and try to make the code neat and clear.
- Static analysis can help team collaboration. Your team need to involve in the same project and be responsible for every line of code. To achieve that, your team members must discuss more often and share your ideas on code.

Tools for Static Analysis

- Static Analyzers
    - First, describe the goals, purposes, and use of static analysis tools (i.e., static analyzers).
    - Use two different static analyzers on your project (on the whole or on the same subset of the code for each tool). Options:
        - Checkstyle
        - FindBugs or SpotBugs (not both)
        - Infer
        - PMD
    - Show the aggregated numbers for the results. First separate out the results for each tool.

- For each tool, dive into the warnings that they identify and describe in detail a few. Describe how they are or aren't actual problems in the code.
- Contrast the information that was provided by each tool. Did they provide information that overlaps in nature or are fundamentally different in their purposes? If they are similar, do they identify distinct warnings (i.e., does one identify warnings that the other does not, and vice versa)? Do they identify some of the same warnings, and if so, are the information provided by each tool of equal value? How so or how not?