Worksheet 7: Modularity

Updated: 9th May, 2020
Revised: 25th April 2022

This worksheet explores modularity, but also involves code review, a concept from the "agile project management" lecture. Code reviews are often used to find defects (bugs/- faults) in the code, but that's not exactly what we're going to use them for here.

In this worksheet, you will conduct a code review to find *modularity issues* in a piece of code: high coupling, low cohesion, and redundancy. You will then determine how best to refactor them to improve modularity and hence *maintainability*.

1. Discussion

Before we get to the review, here are some quick modularity conundrums.

(a) Do global *constants* create a modularity problem? Why, or why not?

(b) If a method/function takes 10 parameters, how/why could this create a high level of coupling?

(c) Why do we want to avoid code duplication? Is it possible to have code with no duplication at all.

2. Checklist

As part of the code review, first construct a checklist based on all the modularity issues raised in the lecture.

You'll be looking for particular modularity issues in a given piece of source code. To do this, it will help to first simply *list* all the different issues that you're looking for. They are all described in the lecture slides, and relate to high coupling, low cohesion and redundancy. However, your checklist should be more specific than that.

Note: See the "agile project management" lecture for details on the generic idea of a checklist. Briefly, a checklist consists of a series of yes-or-no questions about the code (or whatever is being reviewed). To *write* these questions, you need to consider what kinds of things the reviewer should be looking for. Write the questions such that answering "no" would mean there is something wrong.

Not every reviewer would use a checklist in the real world, but they can help the effectiveness of a review, especially if you're not an experienced reviewer.

Note: See the "modularity" lecture for details on the issues that might be expressed as checklist questions.

Briefly, poor modularity leads to poor maintainability; i.e., a reduced ability to do *maintainance* work, such as adding features or fixing bugs. Note that a piece of code that runs correctly can still have poor modularity, and hence be difficult to modify.

3. Review

Obtain either `Statistics.java` or `Statistics.py` from Blackboard, as per your preference. These files contain equivalent code written Java and Python.

Using your checklist, conduct a review of either the Java or Python file, paying attention to the comments. Record and describe each modularity issue you encounter, and the checklist question it relates to (if any).

If you encounter any modularity issues not relate to the checklist questions, update the checklist to include them also.

Note: Use you checklist to guide you. However, you are not restricted to finding issues listed on the checklist. There may be others as well.

4. Refactoring

For each issue you identified above, do the following:

(a) *Describe* (in words) how to fix the issue, in order to improve modularity.

(b) *Implement* your changes.

(c) Check to make sure the program still works!

End of Worksheet