Universität Zürich[UZH]

RERG

Software Quality Exercise 3

Software Metrics, Code Smells

1 Information

1.1 Dates

- Release: 26.03.2012 12.15pm
- Deadline: 02.04.2012 12.15pm
- Discussion: 23.04.2012

1.2 Formalities

Please submit your solution as a *pdf* and submit it via email to `charrada@ifi.uzh.ch`. The subject of the email must begin with *[FS 12 SWQ]*. Exercises should be solved and handed in in groups of three. Every member of a group must be able to answer questions about the group's solution. The document must include the names of the group members.

1.3 Tools

In this assignment we will use two static analysis tools: Eclipse-cs and JArchitect.

1.4 Eclipse-cs

Eclipse-cs is an Eclipse plug-in integrating the CheckStyle code analyzer. Eclipse-cs allows the user to define a set of coding rules that the tool then checks for in the project. All violations to these rules are displayed via the source code annotator of eclipse. CheckStyle is highly configurable, as the user can choose and edit the rules to be checked for in the code. More information about eclipse-cs (installation and documentation) can be found on the tool website: `http://eclipse-cs.sourceforge.net/index.html`.

1.5 JArchitect

JArchitect is an adaptation of the *ndepend*[1] tool for Java code. JArchitect is a commercial tool that runs on windows and linux. Although it can be used freely for academic purposes, some features are disabled or limited in the academic version . The tool can be found on this website: `http://www.javadepend.com/`

2 Documenting Code Smells

In the wiki[2], document two code smells. Explain what the smell is about, why can it be bad and how to remove it. Include a simple example if possible (Examples might be taken from the projects used in the next part of this assignment). To make sure that no code smell is documented twice, before choosing a smell, have a look at the wiki to see which smells have already been chosen by other groups. As soon as you decide which smell to document, add the name of the smell to the wiki and put the student number of one member of your group next to it. Include a copy of the documentation into the solution pdf.

3 Analyzing ImageJ

In this part you should use Eclipse-cs to identify code smells in the ImageJ project.

a) Configure Eclipse-cs to check the following rules:
   - A class should not have more than 2000 lines of code
   - A method should not have more than 7 parameters
   - There should be no duplicate code which is longer than 15 lines of code
   - The cyclomatic complexity of a method should not be more than 50.

b) Explain what consequences the breaking of each of these rules has.

c) Run Eclipse-cs to identify the code smells related the defined rules. Report a summary of the results you obtained.

d) Choose three examples for each type of smell, look closely into it and decide for each case whether it is a bad smell or not. If yes then propose a refactoring. Document your observations and propositions.

e) Create a new branch of ImageJ in the repository and implement the refactoring you proposed for two bad smells (choose two different types of smells). Commit your code to the repository.

[1] http://www.ndepend.com/

[2] `https://daiquiri.ifi.uzh.ch/trac/swq12/wiki/CodeSmells`

4 Code Smells / Design Disharmonies

4.1 Selecting a project

There are five projects available for this exercise:

- JHotDraw: `http://www.jhotdraw.org/`
- JFreeChart: `http://www.jfree.org/jfreechart/`
- JEdit: `http://www.jedit.org/`
- ArgoUml: `http://argouml.tigris.org/`
- FreeMind: `http://freemind.sourceforge.net`

Each group has to select one project and use it to answer the questions for this exercise. Five tickets, one for each project, are available in the trac environment. One member of each group should accept the ticket for the project that the group will work on. The source code should be checked out from the repository location mentioned in the ticket.

4.2 Exploring the project

a) Run JArchitect on the project you have chosen and explore the different features of JArchitect
b) Choose the *Metrics view* of JArchitect. What advantages does the graphical representation have over a value-based representation? Include a screenshot of the Metric view of your project (at the *Method level*) and describe the main observations you made.
c) Switch to the *package level* of the Metric view, then right click on the project and chose *View internal dependency cycles on matrix*. Now you are at the *Dependency matrix view of the project*. Take a screenshot of the matrix and include it to your solution file. What problems can be detected using the dependency matrix? Are there concrete issues that you see with your project?

4.3 Refactoring the project

d) Use JArchitect (and Eclipse-cs if needed) to detect two different design disharmonies in the code. Similar to the example presented in lecture, specify a detection strategy to detect the disharmonies / smells using the metrics provided by JArchitect, explain the strategy and why you think it is reasonable for detecting the particular code smell you chose, and describe the instance(s) you found for the two different disharmonies.
e) Propose a set of refactorings for each disharmony and apply it to the code. Explain why the refactoring you chose is appropriate to eliminate the concrete disharmony. Commit the changes you've made to the repository.