CO4206 / CO7206 / CO7506 Assignment 3
Re-Engineering AssertJ

(This assignment carries 50% of the final mark)

Deadline: 23:59 GMT, Sunday 6th of January 2019

1 Overview

AssertJ is currently one of the most popular Java projects on Github (that are Maven-buildable). It is a relatively large system, comprising of the order of tens of thousands of lines of code, and over 500 classes (you'll calculate the precise metrics throughout the course of this assignment).

1.1 Setup

As with the other assignments, we will use Github Classroom for this assignment. Please clone your personal version of this repository here:

https://classroom.github.com/a/_4zauxdP

We have created a base repository for you, which contains the following familiar directory structure:

- **analysisCode**: This is where you keep the code that you use to analyse the system. As a starting point, we have included the Java, R, and Bash scripts that have been developed in the labs and in Assignment 1. You are free to (and expected to) edit and improve them to suit your needs, and are free to do so in any way that you wish (as long as this does not involve any additional 3rd party code libraries or tools).

- **assertj-core** This contains a snapshot of the assertj-core repository. This contains the full subject system that you are expected to use as the basis for your analysis. It does *not* contain any git data. If you wish to analyse the commit-history of this repository (as we did in our lab session), you will have to clone a separate copy of this on your local machine, and check-out the specific repository we used in our repository (9d45e93). For this you would have to write:
git clone https://github.com/joel-costigliola/assertj-core.git
change directory into the assertj-core directory
git checkout 9d45e93

- **dataFiles** This is where you can store any data files that are produced (e.g. CSV files with metrics, data traces, images, etc.).

- **notesAndReports** This is where you are expected to store your final report, along with any other notes that you use in its production.

2 Instructions

For this assignment, the goal is to put all of the skills that we have learnt throughout the course into practice.

Put yourself into the position you'll hopefully find yourself in a few months from now. You are a graduate employee, put in charge of re-engineering a large system (in this case AssertJ).

Everything in your repository will count as your submission. The main component will be the write-up, which will provide the necessary guidelines.

The detailed submission instructions (along with guidelines on how much detail is expected in the report), are provided below. Each exercise is associated with a section title you are expected to use in the report, along with a guideline for how much detail is required, and to what extent it contributes to the final mark.

Five sections are to be completed:

1. Investigate the system without using any tools, to gain a broad under- standing of its function, its key components, and architectural features. In doing so, adopt the relevant re-engineering patterns, such as ”Read the Code in One Hour”, or ”Skim Documentation”.
   In your write-up, list every pattern that you used, how you applied it in the context of this assignment, and what it enabled you to learn about the system. This latter point is especially important. Try to highlight gen- uinely insightful observations. Include any notes taken whilst applying the patterns in the notesAndReports directory.
   Submission: ”Initial Exploration”, 1.5 - 3 pages of the report.
   Assessment of this section contributes to 15% of the final mark.

2. Carry out some static and dynamic analysis to calculate metrics that can give you an insight into potential weaknesses (or strengths) within the system. For this you should:
   Calculate the LOC and the Weighted Method COunt for each class, and calculate the LOC (i.e. the number of nodes in the CFG) and the Cyclomatic Complexity for each method.
   For particularly large methods (if there are any), you may wish to employ slice-based metrics to gain an understanding, for example, of how cohesive the methods are.
   Analyse the repository log to identify classes that have been particularly change-prone throughout the development of the project.
   Use dynamic analysis to highlight any particularly important classes that contribute to the system when it executes.
   In your write-up, detail what you did and how you went about it. If you wrote your own code to carry out some form of analysis (on top of what is already in the code

base), provide some details. Although you are not expected to include complete tables of results (these would be too large), you are expected to focus on specific sub-groups - e.g. the top 10 results for a given metric. Do not include visualisations or charts yet. The purpose of this part is to focus on method you used, and to highlight individual classes or methods that appear to be notable.
Commit any scripts or code you used to the repository, and ensure that this is described in the report. **If you commit code to the repository and do not mention it, then it will probably not be counted towards your assignment.**
Submission: "Metrics and Analysis", 4 - 7 pages, including at most 2.5 pages of tables.
Assessment of this section contributes to 25% of the final mark.

3. Use visualisation to give you the big picture:
   Use charts created in Excel or R to visualise metrics, or relationships between metrics, to point you towards key problematic areas within the system.

   - Visualise the class structure.
   - Use visualisation to home-in on any code duplication in the system.

   Given that there are over 500 classes in the system, you'll need to use some ingenuity to ensure that the results are readable when it comes to class diagrams or duplication plots. For example, you could think of leaving out classes that are never executed or have a metric value below a given threshold. In the class diagram you could also use metrics to visually accentuate the most relevant / important classes (and to correspondingly reduce the prominence of less important classes).
   In your write-up, provide any visualisations you produce that you believe to be insightful. For every visualisation, be sure to describe what it is showing, and why you believe it to be useful (do not merely make this an assertion, as is the case throughout this report, you need to provide an argument to justify such statements). As with the previous section (and throughout the report) be sure to commit and describe any code.
   Submission: "The Big Picture", 4 - 7 pages, including at most 3.5 pages of visualisations.
   Assessment of this section contributes to 25% of the final mark.

4. Use the evidence you have collected from your analyses to set out which areas of the system could be candidates for re-engineering.
   Review all of the evidence you have collected, choose a particular prob- lem that you wish to focus on. Explain why this is a good candidate, using the evidence that you have collected.
   Submission: "Reengineering", 1 page.
   Assessment of this section contributes to 10% of the final mark.

5. Carry out the re-engineering task.
   Carry out (or attempt to carry out) the reengineering task. Write test cases to try to ensure that your refactoring has not broken anything and be incremental with regular commits. Once you have finished, gather evidence to demonstrate that you have succeeded in improving the quality of the system.
   Note - depending on the scale of the reengineering task, it is possible that you will only partially succeed with the stated reengineering aim. This is fine. It is better to aim for a partial completion of a larger refactoring task, that to carry out a small, trivial task. Assessment of this step will be based on the appropriateness of the selected task (based on the data you have collected – is this really the most urgent task?), and your selection and application of reengineering strategies.
   Once you are done, make sure that you commit the final version of the reengineered system.
   Submission: “Reengineering”, 3-7 pages, including at most 3.5 pages of visualisations.
   Assessment of this section contributes to 25% of the final mark.

3 Submission Instructions

Deadline:

Please ensure that your report is committed to the *notesAndReports* folder, and is formatted as a PDF.

Please ensure that your tools are executable via scripts or test-cases, and that these are documented in the report.

Marking Criteria for Sections 1, 2, 3, 4 and 5:

Marks will be judged according to the percentage of task completion and correctness of technical discussions, for each section. For example, if the tasks are 80% complete and technical discussions are 60% correct, the mark for this section will be (80%+60%)/2= 70% of this section’s mark.

Plagiarism:

This work is subject to departmental plagiarism protocols.

- For students starting from October 2018, use the link to the Blackboard course site Academic Integrity at Informatics (https://bit.ly/2PaCym2).
- For other students, use the link to the PDF: https://campus.cs.le.ac.uk/ForStudents/plagiarism/DoAIF.pdf