COMP5911M Advanced Software Engineering

Coursework 2

This assignment is based on the lecture material dealing with software metrics. There are two separate tasks, with a total of 40 marks available.

Task 1 has several questions associated with it. You should submit your answers to these questions using Gradescope. Task 2 involves programming, and your code should also be uploaded to Gradescope.

This assignment contributes 15% to your overall module grade.

Preparation

Download `task1.zip` from the 'Coursework 2' folder in Minerva. Unzip `task1.zip` and then delete it. This should leave you with two directories named `before` and `after`. These directories contain code for the Car Rental example used in the lectures and exercise, before and after the refactorings of Exercise 4 have been applied.

Tasks

Task 1 (22 marks)

1. Compute the following metrics for source code provided in the `before` directory:
   - Total SLOC (Source Lines of Code) in the package
   - Average number of methods in a class
   - Maximum method complexity for each of `Car`, `Rental` and `Customer`
   - Instability for each of `Car`, `Rental` and `Customer`
   - Abstractness of the package

   A total of 9 marks are available for these calculations.
2. Compute the same set of metrics for the code in the `after` directory. Again, these calculations are worth 9 marks.
3. Compare the values of the metrics before and after refactoring. Relate this to what has been achieved by the refactoring. Do the metrics tell us anything useful about how the software has changed? Are they misleading in any way? [4 marks]

PLEASE NOTE:

- Unless otherwise stated, metrics should be computed by hand, not using a tool. If you actually have to calculate anything, rather than simply counting, make sure that you show your working, as there are marks awarded for this as well as the final result.
- SLOC is defined as the number of non-blank, non-comment lines. You can compute this manually or using a tool such as David Wheeler's **sloccount** (`https://dwheeler.com/sloccount/`)
- Use McConnell's simplified approach to compute complexity.
- Do not include unit testing code in your calculations.
- Do not use more than two decimal places when quoting non-integer results.

Task 2 (18 marks)

Develop a software tool that can does **one** of the following calculations:

- SLOC for each individual class in a given package[1] [6 marks]

[1] We use package in the same sense as Java package here. You might also see this described as a 'module' or a 'namespace'.

- Average and maximum McConnell complexity for each class in a given package [12 marks]
- Abstractness and instability for a given package of classes, within a collection of packages making up a larger software application [18 marks]

You can develop your tool in any sensible modern programming language. If you are unsure whether your choice would be suitable, please speak to Nick.

Your tool can analyze code written in any sensible modern programming language. Note: this does *not* need to be the same as the language used to implement the tool! For example, you could write a tool in Python that analyzes code written in Java if you wanted . . .

Your submission should include a README file giving instructions on how to build your tool (if that is required) and how to run it. It should be possible to do all of this via the command line; your tool should not depend on the use of any particular development environment.

The mark awarded will depend on the level of challenge posed by the chosen metric (see above), the sophistication of the approach you have used, and the quality of your implementation.

Think carefully about an approach that would be effective in solving the problem. You might find that regular expressions are a useful tool. Most modern languages provide good support for using them.

More sophisticated approaches might make use of the **reflection** capabilities of your chosen implementation language. Java, for example, has a powerful reflection API allowing a Java program to analyze the characteristics of other Java code. C# and Python have similar capabilities.

Another possibility, if you are feeling ambitious enough, would be to use a dedicated parser for the language that your tool analyzes. If following this approach, note that you would definitely NOT be expected to implement such a parser from scratch; instead, we would expect to see you using a parser generator tool such as ANTLR to produce the parser code.

If you have questions about any of this, please ask them in Microsoft Teams.

Submitting Your Work

Question answers and code should be submitted to the 'Coursework 2' assignment on Gradescope. You can access this via the link in the 'Submit My Work' folder in Minerva.

Your code submission for Task 2 must be in the form of a single Zip archive (not any other archive format such as tarfile, gzipped tarfile or RAR file). This Zip file should include the README file mentioned earlier. It should not contain an executable or any other build artifacts.