Individual Project

Investigating and Critiquing Joda Time

Version 1.3.1 (29/3/2022)

Text added to the previous version is shown in red.

1. Overview

Joda-Time is a long-standing Java framework to support work with times and dates. Before Java 8, it was the de-facto library (and was largely absorbed into what is now the Java JDK date and time library.

It is a genuine piece of legacy code - it's 20 years old and still going strong. It is still built into many current software applications that have not migrated over to the built-in Java time library.

Your job is to investigate and to critique its design, and to identify potential reengineering opportunities.

This assessment is to be carried out on an individual basis. You must not discuss your solutions or findings with group members.

2. Setup

For students who have already forked their projects, you can either opt to start afresh (starting from the next step), or there are some instructions on the next page that will help you to retain the work you have already done.

Log in to GitLab, and create a new project, called "Individual Project".

Make sure that the project visibility level is set to "private".

Once you have created the project, initialise it by cloning the Individual Project base directory from here. You can achieve this by:

```
git clone https://stugitlab.dcs.shef.ac.uk/courses/com3523/2021-22/individual-project.git
cd individual-project/
git remote remove origin
git remote add origin https://stugitlab.dcs.shef.ac.uk/UNI_USERNAME/REPO_NAME.git
git push -u origin master
```

It is vital that we are able to access your repository, so please **make sure that you give the following people (and only the following people) "maintainer" access** as soon as you initiate the project**:**

- Neil Walkinshaw (@ac1nw)
- Islam Elgendy (@ac1ie)
- Richard Somers (@acs21rs)
- Andy Clark (@acp19agc)

You can find the "add members" section in the "Members" portion of your newly created project in GitLab.

If you have already "forked" your project according to the previous instructions, you can skip the first two steps in the above commands. At the command line, navigate to the directory representing the forked project, and execute the commands starting from "git remote remove origin".
Once you have done this, please remove your forked project from GitLab.

The base repo contains three directories: A clone of Joda-Time, a clone of the Reengineering Toolkit, and an almost empty directory called "submission".

Joda-Time and Reengineering Toolkit have been stripped of their specific git metadata, to avoid any confusion.

You must only use tools that are available in the Reengineering toolkit, or refinements / enhancements of these tools, which are to be committed as part of your submission. You must not resort to third-party tools - e.g. plugins to IntelliJ.

Finally, the submission directory is where you will include your written submission, and any additional data or figures that you'd like to include.

3. Instructions

For this assignment, the goal is to put all of the skills that we have covered so far into practice. The main assessed part of the work will be a write-up, but figures and committed code will be required as well. The report needs to adopt the following structure:

1. **Initial analysis**

   ~~Analyse all of the non-testing .java files in the joda-time package. Analyse the distribution of LOC. Analyse the number of times that they have featured in version repository commits.~~

   ~~Since we have stripped out the Git meta-data so that it would not conflict with the Git data for the assignment as a whole, for the repository analysis you will need to clone the original repository directly from GitHub into a separate directory. For this you should run:~~

   ~~git clone https://github.com/JodaOrg/joda-time.git~~
   ~~git checkout d5efcf6~~

   ~~You should capture the results in some graphical format such as a bar-plot, or some other appropriate visualisation.~~

   Write up briefly what these results indicate about the system. Are there any features that stand out? Are there potential areas of concern? Are these substantiated by a brief informal analysis of the source code?

2. **Static analysis**

   Reverse engineer a class diagram that provides some insight into one or more of the classes highlighted in the superficial analysis. Again, exclude JUnit test classes here. Given the number of classes in the module, this may require you to focus on a particular package or sub-package.

   Change the visualisation of the class diagram, so that either the LOC, or the number of times

a file has been changed, is reflected in the class diagram.

Present the resulting class diagram and briefly discuss any particular highlights.

3. Dynamic analysis

Identify an aspect of functionality that you believe should exercise code that features in the results that you have collected so far. Identify or manually construct a JUnit test case that exercises this function.

Apply software reconnaissance to accurately identify the set of code units that are specifically relevant to this feature.

Discuss your findings. Are the (possibly) problematic classes identified previously specific to the functionality that you have investigated? Or are they classes that appear to be more general (e.g. utility classes)?

For COM6523 students only (*not* undergraduate COM3523 students):

Write your dynamic analysis to record the amount of clock-time that is spent executing each method. When a method calls another method, then the amount of time spent on the callee should *also* be counted as clock time for the method that calls it.

For the same traces as are used for the software reconnaissance (recorded with your enhanced analysis), calculate (e.g. in Excel) the total amount of time spent executing each method.

Can you derive any additional insights from this analysis? For example, is there any correlation between the amount of time spent executing a class and the size of its methods?

4. Code clone detection

Calculate the Jaccard score for every pair of non-testing classes. Compare a few of the pairs with the top scores (provided that these are of a reasonable size and complexity).

Visualise the code clones for one pair of classes as a dot-plot, and describe any clones therein.

5. Metrics

Create a CSV file where each row contains a class (excluding test classes). For each class, you should compute:

- The Fan-in for the class.
- The Fan-out for the class.
- The number of methods in that class.
- The number of data-members of that class.

For the first two (Fan-in and Fan-out), you will need to compute the call graph. For the second two you will need Reflection. Use the approaches covered in the Static Analysis lab to compute this.

6. **Critique**

Referring back to the evidence that you have collected in the previous topics, along with any additional metrics you have computed where appropriate, identify your three top design

problems with the system.

This must draw upon the material covered in lectures - offering some useful justification on the basis of the various forms of information collected, as well as knowledge about good design principles.

Approximate word-length for COM3523: 1000-1500 words.
Approximate word-length for COM6523: 1500-2000 words.

Assessment

The assessment of your submission will be conducted according to the schema below. Pay attention to the different weightings for your respective degree programmes.

Section	0-39%	40%-54%	55%-70%	70%-100%	Weighting for COM3523	Weighting for COM6523
1 (Initial analysis)	No successful attempt at either an analysis of file sizes, or analysis of the version repository.	There is either a successful analysis of file size distributions, or of file commits. Some analysis of the results is there, though it may be lacking in places.	Both analyses have been successful. There is an analysis that covers the results and delivers insights. No substantive successful effort to customise the reengineering tools or scripts.	Both analyses are described in detail. The analysis scripts or tools have been enhanced to deliver additional insights.	10%	10%
2 (Static analysis)	No successful attempt to reverse-engineer a class diagram.	A basic class diagram is reverse-engineered (without amending to accommodate additional data such as LOC). The accompanying analysis is limited.	You have produced a class diagram with the appropriate amendments, and a limited accompanying analysis.	You have produced a class diagram with the appropriate amendments, and a comprehensive and well illustrated accompanying analysis.	10%	5%
3 (Dynamic Analysis)	You have not managed to collect traces or analyse them.	You have collected traces (though perhaps not appropriately chosen). You have a crude form of the analysis working.	You have chosen sensible trace-sets, and the analysis works well. Your discussion of the results or the software tools you have developed may be limited.	You have produced sensible trace sets, with a comprehensive and insightful discussion of the results. Your software tooling is of a high quality, and your discussion of the results is insightful and complete.	15%	20%

Section	0-39%	40%-54%	55%-70%	70%-100%	Weighting for COM3523	Weighting for COM6523
4 (Code clone detection)	You have chosen an inappropriate set of classes to mine for clones. You might not have managed to produce a dot plot.	You have produced a dot plot, and have justified your choice of classes to compare. You have a limited analysis of the results.	Your discussion of the pair of classes you are comparing is comprehensive and insightful.	Your analysis extends to discuss potential properties of the design that might underpin the code clones. Alternatively, it involves some innovative tooling that enables some helpful insights.	10%	10%
5 (Metrics)	You fail to produce any of the metrics requested above.	You are able to produce two of the four requested metrics, and provide an accompanying description.	You are able to produce all of the metrics, alongside a description.	You produce all of the metrics, along with an additional metric that highlights a point in your description.	10%	10%
6 (Critique)	You fail to produce a well justified and reasoned critique of the system design.	You produce a basic critique of the design, drawing on some of the evidence collected previously.	You produce a comprehensive and well justified critique of the system design that draws on some of the evidence collected previously.	You produce a comprehensive and well justified critique of the system design that draws on *all* of the evidence collected previously.	30%	30%
Overall	Some parts of the submission are missing.	Every part of the submission is present, but the accompanying text tends to be difficult to follow.	Every part of the submission is present, and easy to follow.	In more than one of the sections, you have shown some innovation in how you have presented the results.	15%	15%

Submission

Your submission material should be placed inside the directory called “submission”, in the root directory of your GitLab repository [the other two directories in the root directory should be “joda-time” and “reengineering-toolkit”].

Your submission will be your GitLab repository (specifically, the final commit you make to it before the deadline).

This should include:

(1) A written report, submitted as a PDF (this is strict - please do not submit other formats, such as Word documents). This should include the main figures, tables, and charts, with the written material for each of the 6 parts detailed above.

Please ensure that the front page of your report contains your university user name (not your registration number, but ID that you use to log into lab computers, e.g. mine is ‘ac1nw’).

(2) Separate data files for the individual six activities (e.g. csv files etc.). If there are lots of them, they can be put into a separate sub-directory or zipped into a zip file. *You must only use zip*

compression as a format, no other formats will be processed.

Data files for each of the six activities should be committed and pushed separately. Do not commit and push everything in a single go. The point is that you make regular commits as you work on this.

(3) Any code that was used to enhance the analyses. These code changes should be committed to the GitLab repository with commit messages that highlight what they are doing. Code changes should also be explicitly referenced in the relevant parts of the PDF.

There will not be a submission-point on Blackboard, and submissions via email will not be accepted.

The deadline for the submission is 15:00 GMT, on the 29th of April.

Late submissions will be subject to standard late submission penalties, as stipulated by the university regulations. A summary of these can be found in the UG and PG student handbooks.

Support

To ensure fairness, we will only respond to queries on the Blackboard forum dedicated to this assignment. As usual, please read other queries before you post your own, to ensure that your query has not already been answered. To assist your colleagues when they are checking the forum, please give your query a descriptive title.

You must under no circumstances post your own solutions (or parts thereof) to the forum - so please be mindful when you are posing questions.

Unfair Means

It is important to bear in mind the departmental rules on unfair means. Activities such as plagiarism or collusion will be treated as a serious academic offence. This could lead to the award of a grade of zero for this assignment. Since this accounts for 80% of the module mark, this would automatically result in a failure of the module, with severe implications for the possibility of obtaining a degree.

We will be closely scrutinising submissions to detect such practices, because it is important that this assessment is a genuine reflection of your own understanding of the module.

To avoid any potential accidental wrongdoings, it is especially important that you do not discuss your solutions with other students. If you have questions about this assignment, please use the discussion forum.