**1. Initial analysis**

**Basic**

First, use initial-analysis.sh to call listFileSizes.sh & repMining.sh and generate csv at once. Then, analyse those csv files and save results to xlsx.

Here is the LOC of .java files in main and their bar chart analysis. (In submission/initial-analysis/javaFileSizes.xlsx)

Chart

Description automatically generated

Figure 1.1 LOC data and its distribution and Pareto chart.

For the commit results, first get the count of timestamp. (In submission/initial-analysis/joda-time.xlsx$PivotTable)

Graphical user interface, application

Description automatically generated

Figure 1.2 The number of commits of each file in the repo and its bar chart.

Matching the two columns we get (In submission/initial-analysis/joda-time.xlsx$Correlation):

Chart, scatter chart

Description automatically generated

Figure 1.3 The correlation dot chart between LOC and counts of commits.

**Extra**

With adding specific path to git log in repMining.sh we get targetMining.sh. It can get commits with .java file changes. The result is in submission/initial-analysis/joda-time-javaTarget.xlsx. With this we can know which classes are related during changes after some processes:

* ignore .java file with only upload.
* ignore commits related to annotation and Javadoc.
* Ignore non-functional huge commits, which include most .java files.

Write up brieﬂy what these results indicate about the system.

The number of times the Java file has been modified is basically not more than 20 times. In addition to the small file size, it may also indicate that the correlation between components is not large.

Are there any features that stand out?

There are many commits and thousands of files in the repo analysis. However, only 166 .java files are found in main directory.

The size of the .java file is proportional to the number of commits.

Are there potential areas of concern?

The difference in the number of files is a concern.

Are these substantiated by a brief informal analysis of the source code?

The repo experienced some major changes such as movements, split and v2.0.

**Static analysis**

The whole project includes three parts, basic time format, time format conversion, and complex time format, which is based on the first two parts.

Usually the core classes (with many other classes associated) are modified less frequently to ensure the stability of the project. Classes with many modifications usually have some problems or high difficulty.

Among them, the class org.joda.time.DateTimeZone is not only very critical, but also modified many times. It is quite certain that some problems lie in this place.

**Dynamic analysis**