**Lab sheet 2 (Week 3: 14th October 2020):**

**Analysing code-based metrics**

In this lab, we’ll look at some software metrics from Java systems.

The Excel file we will use has been posted on BBL (see Exercise 1). You can use R or SPSS or any data manipulation tool if you so choose. **We will be covering what some of these metrics mean in the lecture, so don’t worry if you don’t follow some of them in the tasks at the moment.**

**QUESTION:** What is the purpose of this lab sheet? Where does it fit into the general scheme of the module and why is it useful for you?

**ANSWER:** In the CS3003 coursework, you will be asked to carry out analysis of data from a system or systems – and metrics may be at the heart of them. So the purpose of this lab sheet is for you to get used to metrics from real systems and analyse them.

**EXERCISE 1**

Download the Excel file called “lab\_2\_proper.xlsx” from BBL labs folder and save it somewhere. This is real data from real open-source systems. This file contains metric data from four systems (see Column A for the project name). Find out what these systems actually do (i.e., what is their purpose).

Taken from the Apache website: “Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications”.

Taken from Wikipedia (so it must be true!!): “Apache Camel is an [open source](https://en.wikipedia.org/wiki/Open-source_software) [framework](https://en.wikipedia.org/wiki/Framework-oriented_design) for message-oriented [middleware](https://en.wikipedia.org/wiki/Middleware) with a rule-based routing and mediation engine that provides a [Java object](https://en.wikipedia.org/wiki/Plain_Old_Java_Object)-based implementation of the [Enterprise Integration Patterns](https://en.wikipedia.org/wiki/Enterprise_Integration_Patterns) using an [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface) (or declarative Java [domain-specific language](https://en.wikipedia.org/wiki/Domain-specific_language)) to configure routing and mediation rules”.

Again, from Wikipedia: “Apache Ivy is a transitive package manager. It is a sub-project of the Apache Ant project, with which Ivy works to resolve project dependencies. An external XML file defines project dependencies and lists the resources necessary to build a project. Ivy then resolves and downloads resources from an artifact repository: either a private repository or one publicly available on the Internet”.

jEdit is free text editor. Enough said.

**EXERCISE 2**

You will notice that that there are many columns in this dataset that you won’t recognise. Please don’t worry about whether you understand what they mean for now. However, there are two metrics in this file which are worth exploring.

Have a look at the “MAXCC” and “AVGCC” columns. These are two metrics that represent the “maximum complexity” of all the methods in the class and the “average complexity” of all the methods in the class. For now, don’t worry too much about how they are calculated – we will cover these in the lecture. But just be aware that they try to capture how complex a method is in some sense. You might like to think how \*you\* would capture complexity of a class. Give that some thought.

From the data set, calculate the average, median, maximum and minimum values of MAXCC and AVGCC just to get a flavour of the distribution of the values for each system. State which system in your opinion is most complex and also the least complex?

For ant: MAXCC: average = 3.85; median = 2; maximum = 53 and minimum = 0

For camel: MAXCC: average = 1.19; median = 1; maximum = 2 and minimum = 1

For ivy: MAXCC: average = 2.26; median = 2; maximum = 3 and minimum = 2

For jEdit: MAXCC: average = 7.91; median = 5; maximum = 167 and minimum = 3

For ant: AVGCC: average = 1.23; median = 1; maximum = 8.45 and minimum = 0

For camel: AVGCC: average = 0.92; median = 0.86; maximum = 9.5 and minimum = 0

For ivy: AVGCC: average = 1.18; median = 1; maximum = 8 and minimum = 0

For jEdit: AVGCC: average = 1.65; median = 1; maximum = 28.67 and minimum = 0

Hey presto, surely jEdit is the most complex system from the data? It has high MAXCC values and very high AVGCC values compared with the other systems. The least complex is surely camel (it has low values compared with the other three systems).

**EXERCISE 3**

Order the set of classes on descending order of AVGCC and then take the top 5 classes from that list. Find the code that relates to each of those classes. You can do this through a simple search. What do you notice about these classes? Are there any particular characteristics that stand out? Hint: maybe the logic is quite complex. If that is the case, what type of logic is making these classes complex?

If you do that – you’ll see that it’s the jEdit system that shows all top five classes in terms of AVGCC, and the top four are **ALL THE SAME CLASS**, but for different versions of the systems and it is called: **org.gjt.sp.jedit.gui.KeyEventWorkaround**. If you look at this class, you’ll see that it has a lot of CASE statements and lots of conditionals as well. So that is why it has a high complexity value. The description of the class in its header is:

This class contains various hacks to get keyboard event handling to behave in

\* a consistent manner across Java implementations, many of which are

\* hopelessly broken in this regard.

The fifth class is called **org.gjt.sp.jedit.pluginmgr.PluginManager$ActionHandler** and if you inspect the code, you’ll see that again, it has a large amount of conditional and heavily nested code. That is why it is so complex given by the metric values you produced in Exercise 2.

**EXERCISE 4**

Order the set of classes on ascending order of AVGCC and again take the top 5 classes from that list. Find the code that relates to each of those classes. What do you notice about these classes? Are there any particular characteristics that stand out? Hint: maybe the logic is quite simple here.

If you do that, then the five classes are:

org.apache.tools.ant.Constants,

org.apache.tools.tar.TarConstants,

org.apache.tools.ant.taskdefs.LogStreamHandler,

org.apache.tools.ant.taskdefs.Execute$1 and

org.apache.tools.ant.ProjectHelper$1

If you look at the first of these classes, there is NO logic in this class at all. It is a class which defines constant values and that is all.

**EXERCISE 5**

Produce a scatterplot of AVGCC versus MAXCC. On this figure you should be able to see FOUR extreme outlying values for MAXCC on the far right side of the figure. What are these four classes? Why do you think this type of class is an outlier? Have a look at the code again.

Here is the scatterplot (x-axis shows MAXCC, y-axis shows AVGCC):

The four outliers are all Parser-based classes and all are in jEdit. The bsh.ParserTokenManager has hundreds of case (logic) statements, so it is no wonder that it has high complexity.

**EXERCISE 6**

Look briefly at the WMC metric in Column D. This is the number of methods in a class. Draw a scatterplot of WMC versus AVGCC. What sort of correlation value do you get?

Here is the scatterplot; WMC is on the y-axis. The correlation value is only 0.18. Now, this is interesting, since it shows that number of methods in a class DOES NOT NECESSARILY mean that the class is complex!