"Software Engineering 2"
– *Metrics 1*

In this task several metrics should be calculated and evaluated.

1 Evaluating one Project

In the first task a single project should be evaluated. You may either use a given project or a (relatively large) project of your own. If you want to use the given project download the file "`Sample Projects.zip`" from the Moodle course, unpack it. It contains some sample projects of mine you may measure. (Alternatively you may find the projects in gitlab, see below).

1.1 Preparation

Unfortunately, the metrics-plugin to be use is no longer fully compatible with the current eclipse-version, while the metrics-view is still working, the layered-view seems to have issues – as far as I can see, it is working up to eclipse 4.11 ("2019-03") (you can download this version at https://www.eclipse.org/downloads/packages/release/2019-03/r).

Either which project you choose, do the following actions in Eclipse:

- Install the appropriate eclipse-version.
- Install the "Eclipse-metrics plugin continued" (http://metrics2.sourceforge.net/update/).
- Import the projects you want to measure – provided are the following projects
  - TreePanel: implements a Swing component for displaying tree-like structures; this may be interesting for *checking several metrics*; for measurement use the project `TreePanel – Core v1.4.2` (also in Gitlab: https://gitlab.rz.hft-stuttgart.de/deininger/se2-metrics-1-tree-panel)
  - BookApp which implements a fully functional REST-Application including database; this may be interesting for *checking the layers*; for measurement use `05.bookapp.restful.client` and `05.bookapp.restful.server`. (also in Gitlab: https://gitlab.rz.hft-stuttgart.de/deininger/se2-metrics-1-bookapp-server, https://gitlab.rz.hft-stuttgart.de/deininger/se2-metrics-1-bookapp-client, https://gitlab.rz.hft-stuttgart.de/deininger/se2-metrics-1-bookapp-database)
- Open the "Metrics View" and the "Layered Package Table View" in Eclipse
- Select the project you want to measure and "enable metrics" in the project properties.
- If the metrics are not calculated, perform a rebuild of your project.

1.2 Evaluation

Now evaluate the results:

- Check which metrics are beyond threshold (marked as red) and open the crucial methods (can be done in the metrics view).
- Inspect and rate the methods personally – are they really potential troublemakers?
- Decide if these methods really need to be done that way, or what could be done instead.
- Try to find out the crucial 10% classes in the project.
- Especially inspect the Martin-metrics
  - Efferent and afferent coupling: how can the results be interpreted?
  - Map the Distance for the packages: which is in the zone of pain / usefulness?
- Inspect the layers in the layers view: do you have separate layers, do they have a common concern?
- Export the metrics report as xml-file (through the metrics view menu)

1.3 Programmatic Export

For a (possible) nightly build metrics should be collected for each build (and probably evaluated over time). To do so create the following ant-script in the project you want to report:

```
<project name="Metric Export" default="main" basedir=".">

     <basename property="project.name" file="."          />
     <property name="build.delay"        value="5"      />

     <target name="init.stamps">
          <tstamp/>
     </target>

     <target name="export" depends="init.stamps">
          <echo>Exporting metrics for '${project.name}'</echo>
          <eclipse.refreshLocal resource="${project.name}" depth="infinite"/>
          <metrics.enable projectName="${project.name}"/>
          <eclipse.incrementalBuild project="${project.name}" kind="full"/>
          <echo>Wait for ${build.delay} seconds
               for the build to be finished</echo>
          <sleep seconds="${build.delay}"/>
          <metrics.export
               projectName="${project.name}"
               file="${project.name} - metrics - ${DSTAMP}-${TSTAMP}.xml"/>
          </target>

     <target name="main" depends="export">
     </target>

</project>
```

Attention: To successfully execute the script, **run it in the same JRE as the workspace** (otherwise you cannot access eclipse and metrics-tasks). To do so call “Run as..” and edit the launch configuration at the tab “JRE”).

Remark: The incremental build is a background process, as I could not find a way to determine, when the process is finished, I simply wait for 5 seconds. Then the export starts. Otherwise the export may fail, as the project is still busy and cannot be accessed – maybe you find a better solution for this.

After the script has been executed, a xml-file with the collected metrics should be available in your project. Open the xml-file and inspect the results.

2 Manual Metrics Calculation

2.1 Code Metrics

Given are the following two implementations of the greatest common denominator (gcd) – an iterative version (GCD1) and a recursive version (GCD2). Remark: For solving the exercise, it is not necessary to understand the algorithms fully.

As part of a larger project, all classes have a common copyright note, which would be the minimum content of a class.

```
/*******************************************************
 * Copyright (C) 2019-2020 Acme Corp. <info@acme.com>
 *
 * This file is part of the Exam-Project.
 *
 * The Exam-Project can not be copied and/or distributed
 * without the express permission of Acme Corp.
 *******************************************************/

public class GCD1 {

     public int gcd(int a, int b) {
          if(a < b) {
               int t = a;
               a = b;
               b = t;
          }
          while(a % b != 0) {
               int r = a % b;
               a = b;
               b = r;
          }
          return b;
     }

}
```

This is an alternative implementation:

```
/*****************************************************
 * Copyright (C) 2019-2020 Acme Corp. <info@acme.com>
 *
 * This file is part of the Exam-Project.
 *
 * The Exam-Project can not be copied and/or distributed
 * without the express permission of Acme Corp.
 *****************************************************/

public class GCD2 {

     public int gcd(int a, int b) {
          if (a < b)
               return gcd(b, a);
          else if (b == 0)
               return a;
          else
               return gcd(b, a % b);
     }

}
```

For doing a comparison of the complexity of these two implementations, the following metrics should be calculated:

LOC (Lines of Code): defined as all source lines including empty and comment lines.

LOC(GCD1): 26	LOC(GCD2): 21

CC (Cyclomatic Complexity): defined as the number of independent paths of the control-graph.

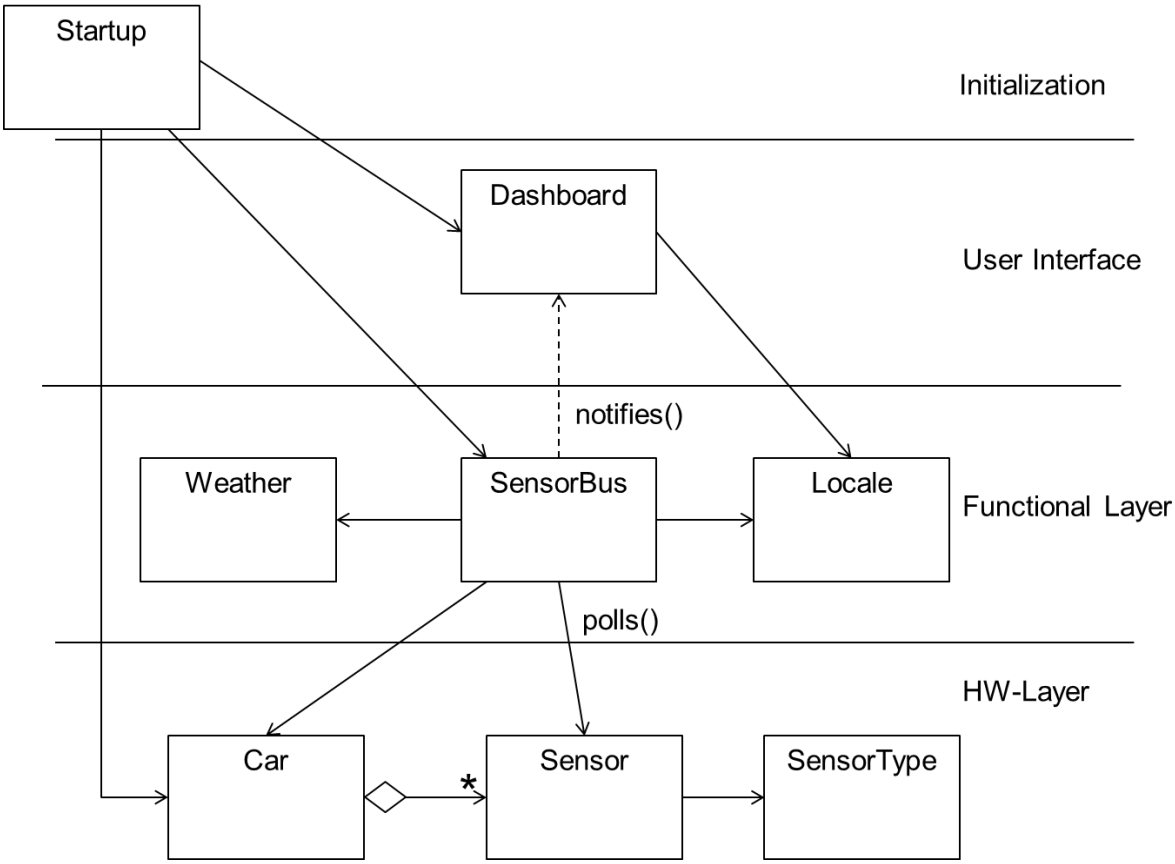
CC(GCD1): 3	CC(GCD2): 3

CR (Complexity Rating): Rating of a quality auditor with 0 [low], 1[medium], 2 [high], 3 [very high] – do this rating according to your personal experience (therefore you cannot do much wrong)

CR(GCD1): 0	CR(GCD2): 1

2.2 Martin-Metrics

Given is the UML-Diagram of the car-simulation application. Every layer represents its own package.



Calculate the Martin-Metrics for the four packages.

	CA_p	CE_p	A_p	I_p	D_p
Initialization	0	1	0	1	0
User Interface	1	1	1	1/3	0.3
Functional Layer	2	1	0	1/3	0.3
Hardware Layer	2	0	0	0	1

Apart from the initialization this is a correct example of a layered architecture. Therefore calculate the afferent and efferent coupling again, this time without the initialization package.

	CA_p	CE_p
User Interface	1	1
Functional Layer	1	1
Hardware Layer	1	0

Which number(s) give a hint, that these three packages may form a correctly layered architecture?