

Cláudia Santos 57049

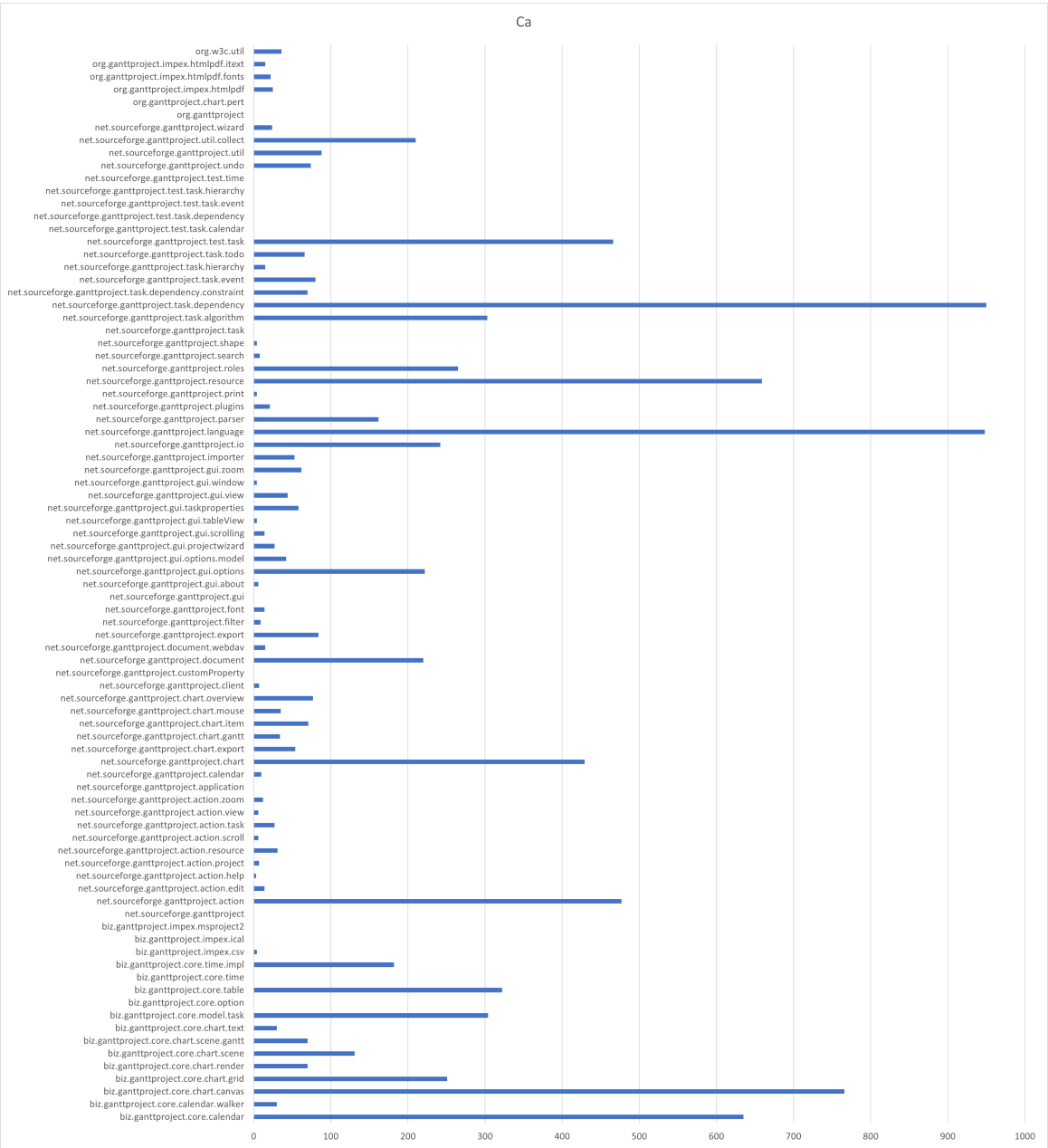
3 de dezembro de 2022

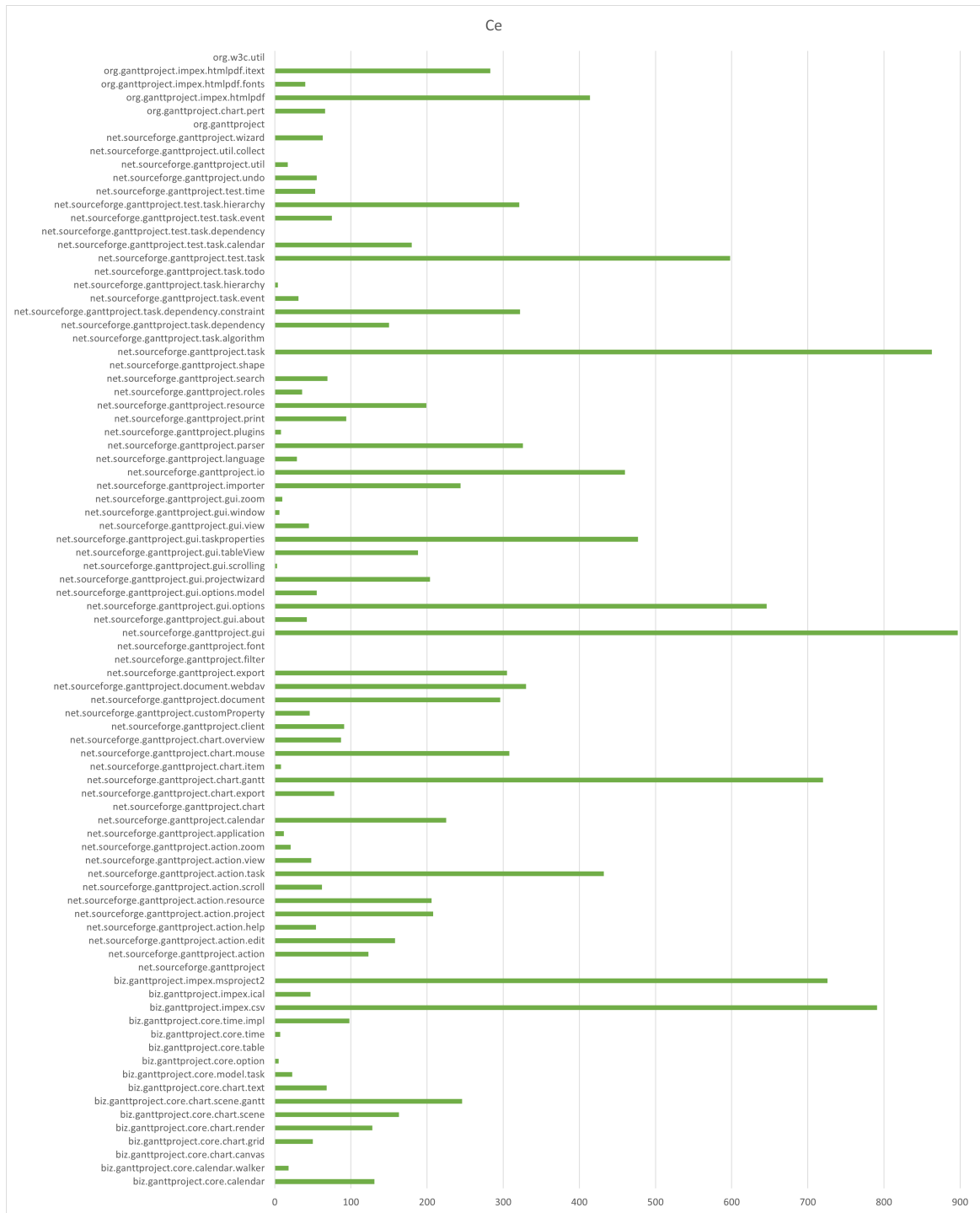
1 Martin Packaging Metrics

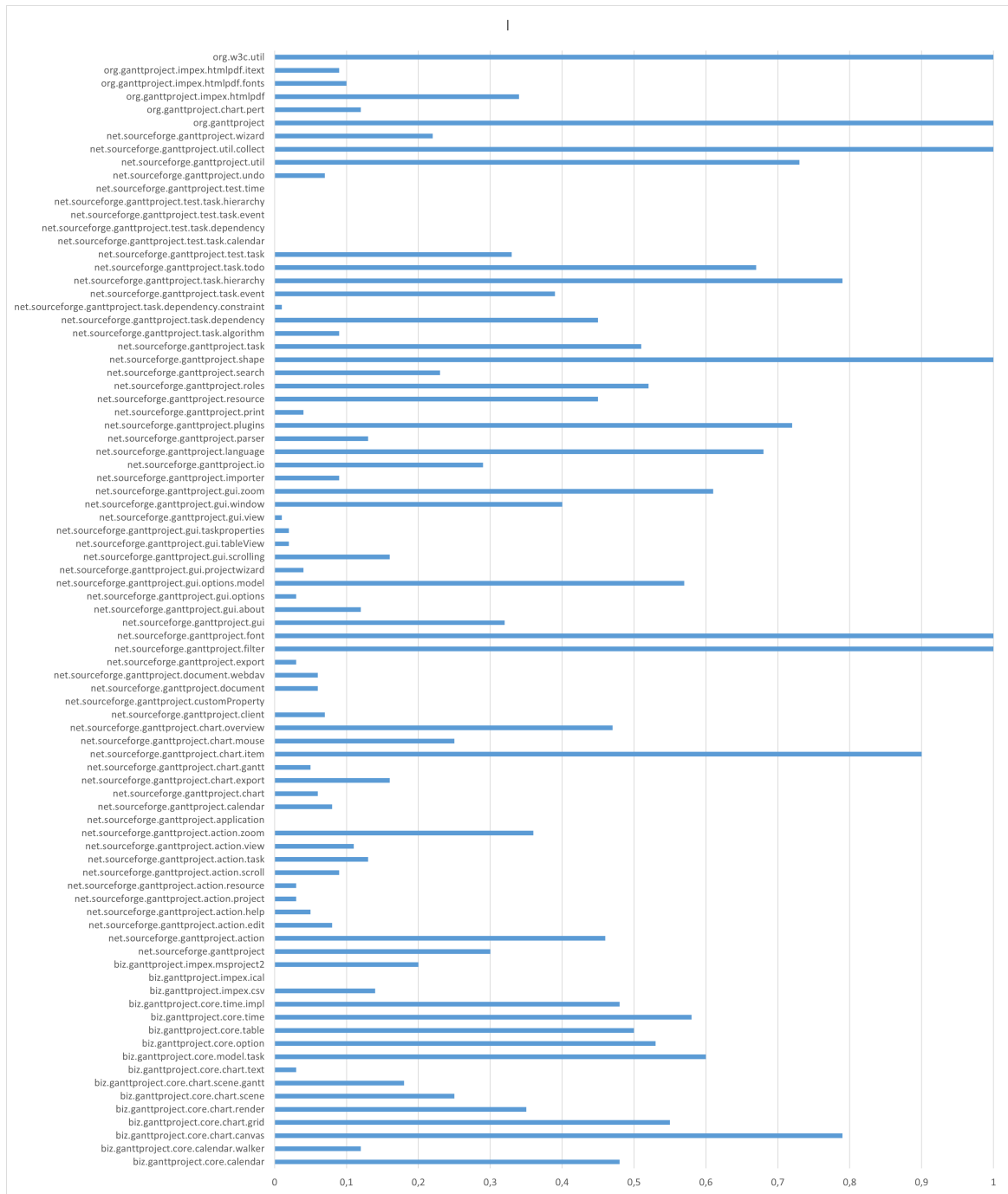
The Martin Packaging Metrics focus on the relationship between packages in the project.

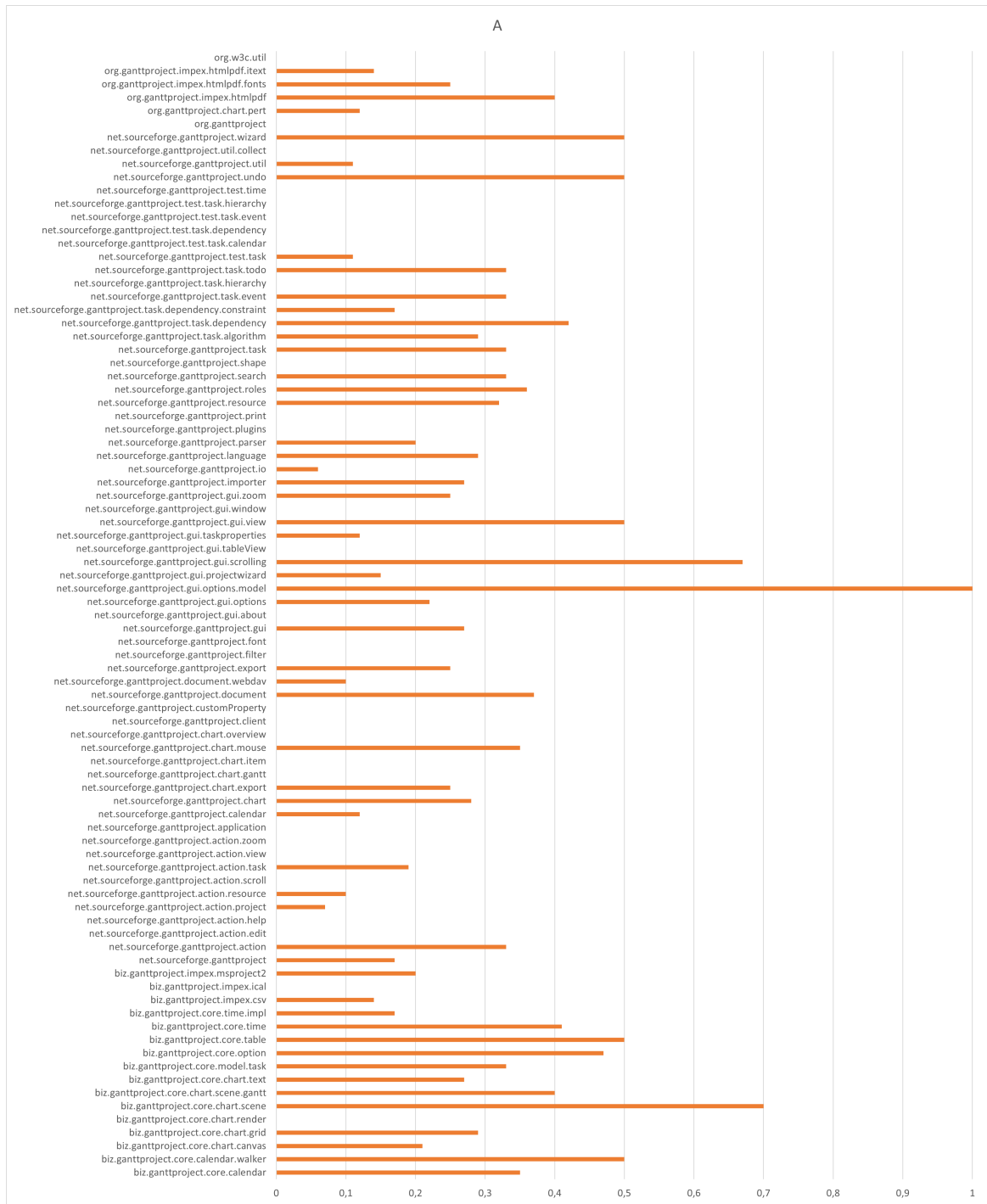
- **Afferent Coupling (Ca):** This metric measures the number of classes of other packages that depend on classes of a given package.
- **Efferent Coupling (Ce):** This metric measures the number of classes in a given package that depend on classes of other packages.
- **Instability (I):** This metric indicates how easy or difficult it is to modify the classes of a given package (if they are more stable or unstable).
- **Abstractness (A):** This metric compares the number of abstract classes and interfaces to the number of concrete classes of a given package.
- **Normalized Distance from Main Sequence (D):** This metric indicates the balance between abstractness and stability of the classes of a given package.

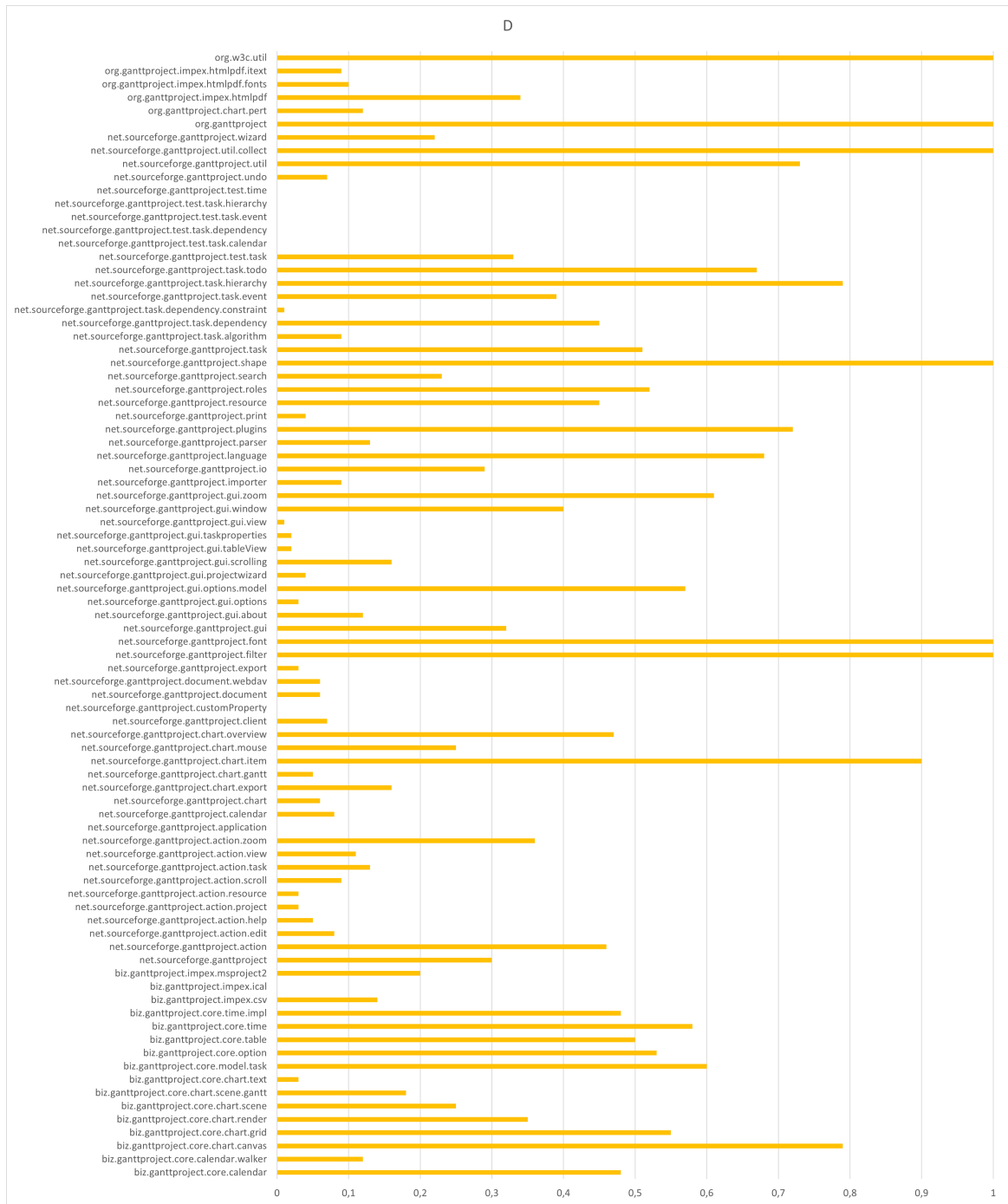
2 Data Collection











3 Code Smells and Code Analysis

We can infer that the classes in the package `net.sourceforge.ganttproject.gui` have some abnormal values.

Ce	Ca	I	A	D
897	1302	0,39	0,27	0,32

The extremely high **Ce** metric indicate that changes on packages of which this package depend on might lead to changes in itself, but, vice-versa, at the same time the high **Ca** metric indicates that any changes to the package itself might also lead to changes to packages that depend on it.

The package has a low abstractness metric and an undesirable instability level. Packages should not be of intermediate instability, they should either be very unstable (easy to change) or stable (sturdy because of greater responsibilities), and if they are very stable, they should also have a good level of abstractness, to compensate for their sturdiness and make them easy to extend.

This balance between stability and abstractness is measured by the **D** metric, which should be the closest possible to 0. In this package, it is 0,32, which, accounting for all the other values, is not great.

Multiple classes in this package may suffer from feature envy as well as inappropriate intimacy, which can all lead to shotgun surgery very quickly.

A great example of this is the `GanttTaskPropertiesBean` class. To make a change to a task property and to its view in the GUI, we witnessed a Shotgun Surgery situation, where we had to edit multiple classes (like the `Task` implementation, interface, the `TaskMutator` class, etc) to make a simple change.

Another example that had a code smell discussed in Phase 1 is the `TaskManagerImpl`, in the `net.sourceforge.ganttproject.task` package.

Ce	Ca	I	A	D
863	4369	0,16	0,33	0,51

The values have the same issues as explained in the first package, this time with an even more obvious **Ca** metric that indicates that classes in this package depend on many other classes from different packages.

In the case of `TaskManagerImpl`, as it was discussed in Phase 1, it suffers from the Long Method code smell, but also the Cognitive Complexity and Feature Envy code smells.