**Martin Packaging Metrics**

**Abstractness(A)**

Abstractness in this context, is the number of abstract classes in the package to the number of all classes.

This metric help developers to measure the degree of abstraction of the package and it has a range of [0,1]. A=0 indicates a completely concrete package, A=1 indicates a completely abstract package.

Analyzing “**A**” metrics with excel Box Plots:

As we can see on the Abstractness Box Plot(blue) in excel, we have a low average of abstraction (0,25), and we only have one bigger package abstraction (1.0), as we can see on the top point of the Abstractness Box Plot, that is “net.sourceforge.ganttproject.gui.options.model” package, with 3 abstract classes/interfaces.

Also, we have many packages without any abstract class/interface, such as “net.sourceforge.ganttproject.chart.item”, to avoid the complexity, and turn the code more efficient, “CharItem” class should be an abstract class.

We can conclude that this project uses a low level of abstraction, the developers should increase to a near number 1 number as possible (as the example given upper), by hiding some implementation details that are not useful right now and simplify code structure.

**Afferent Couplings(Ca)**

The Afferent Couplings packaging metric represents the number of classes of other packages that depend to this package.

Analyzing “**Ca**” metrics with excel Box Plots:

The higher value is on “net.sourceforge.ganttproject.task”(2559), how we can see on the top point of “Afferent Couplings Box Plot”(yellow).

This package has many interfaces/abstract classes, so it’s normal to have a substantial quantity of “Ca”, but “2559”? “Yes, that’s the problem”. So, if we want to replace something on a class, we have breaking changes repeatedly to lots of code, in this case, to all the classes from other packages (1-2559 except abstract classes/interfaces) that use the replaced class from other package.

**Efferent Couplings(Ce)**

The Efferent Couplings packaging metric represents the number of classes of this package that depend to other package classes.

Commonly, excessive values are inconvenient because they can cause problems with the maintenance and code development.

Analyzing “**Ce**” metrics with excel Box Plots:

The classes in “net.sourceforge.ganttproject” have a very high “Ce” value(how we can see on the top point of “Efferent Coupling Box Plot” (yellow)). This is because the classes are not well structured, this package has more than 50 classes/interfaces, instead, those should be divided and reorganized into different packages (e.g. “Resource..” classes inside “resource” package, already created, and/or delete unused classes like “ResourceTreeImageGenerator”).

There are many other classes in the same situation, and the average (202) is very high, that causes instability of a package. As “Ca”, if we want to change one class, this will produce a numerous external classes modification to the package.

**Distance from the Main Sequence(D)**

This metric is used to measure the balance between stability and abstractness (The closer D is to 0, better (means that “A” and ~“I” are near)).

Analyzing “**D**” metrics with Bar Chart excel:

There are some packages, with stability and no abstraction, and the opposite, those sides represent the higher values of “D” (maximum unbalance) (e.g. “net.sourceforge.ganttproject.filter” package has less abstraction, than stability and “net.sourceforge.ganttproject.gui.options” package has less stability than abstaction).

We conclude, the packages of the project are with low stability, and low abstraction, as we can observe on the excel Bar Chart(D).

**Instability(I)**

This metric is used to measure the relative susceptibility of class to changes. According to the definition instability is the ration of outgoing dependencies to all package dependencies and it accepts value from 0 to 1. Better values are near to 1, that means this package can be easily changed.

So, having many classes of other packages depending on this package, and not depending on almost any package class, give stability to the package, so high “Ce” numbers and low “Ca” values is the ideal pattern.

Analyzing “**I**” metrics with excel Box Plots:

As we can see on “Instability Box Plot” (gray), the average of stability is 0,50, so that means the project have many package classes depending on others, not s sustainable structure.

An example of that is “biz.ganttproject.core.option” package(I=0,0), a not structured package. This happens by the number of classes/interfaces inside, so because of that, many classes from other packages are depending on this package, and that induces code instability.

**CODE SMELLS RELATION to 1 phase identification:**

As we are talking about packaging metrics, sometimes there isn’t a direct relation between class Code Smells, but we can still analyze some potential correlation between them.

* Data Class – Unnecessary data classes in a package can make “Ca” higher and “Ce” even higher, and consequently Instability(I) problems because this “data classes”, that we don’t need (Ce), are depending on another ones, and other ones in this one (Ca).
* Duplicated Code – Having less abstraction(A) increases the Duplicated Code in package classes and the same inversely.
* Message Chains – The “message chain” classes of a package are dependent on navigation along all the class structure, so it can increase “Ce” and “I”.