Dependency Metrics

Ana Gadelha 59943

# Introduction

In this report I will analyze the different Dependency Metrics

# Metrics:

* Cyclic - Number of Cyclic Dependencies
* Dcy - Number of Dependencies
* Dcy\* - Number of Transitive Dependencies
* Dpt - Number of Dependents
* Dpt\* - Number of Transitive Dependents
* PDcy - Number of Package Dependencies
* PDpt – Number of Dependent Packages

# Number of Cyclic Dependencies

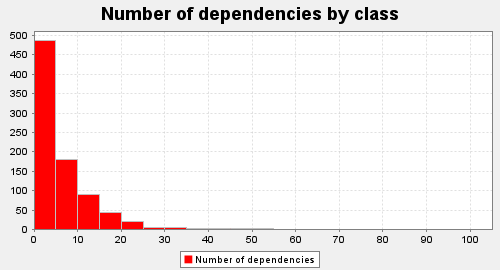
* This dependency calculates the number of classes or interfaces that depend directly or indirectly on another that in turn depends directly or indirectly on the first one. This type of cyclic dependency can result in code that is difficult to test and understand.
* In this histogram we can see that most values are either between 0 and 5 or around 500 what shows that there are a lot of classes that depend on one another, what can become a code smell.

Uma imagem com mesa

Descrição gerada automaticamente

# Number of Dependencies and Transitive Dependencies

* These dependencies calculate the number of classes or interfaces which each class directly depends on.
* Uma imagem com mesa

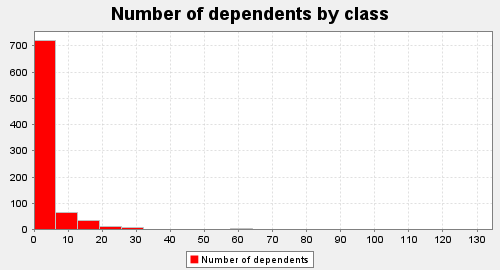
  Descrição gerada automaticamenteThe Transitive dependencies differ from the normal ones because these also count the number of classes or interfaces which each class indirectly depends on.

* In the first histogram we can see that the number of dependencies varies mostly between 0 and 10 and then we have a few more between 10 and 25 and the remaining from 30 until one with 100.
* In the second histogram we have the number of transitive dependencies that varies from 0 to 50, in a good number of classes and from 50 to 60, in some classes, and then between 873 and 886 in the remaining.

# Number of Dependents and Transitive Dependents

* This dependency calculates the number of classes or interfaces which directly depend on each class.
* Once more the difference between this and the transitive version is that the transitive dependents also count the number of classes or interfaces which indirectly depend on each class.

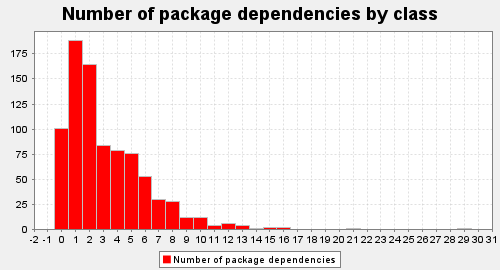
Uma imagem com mesa

Descrição gerada automaticamente

* In the first histogram the values vary mostly between 0 and 6 with some exceptions afterwards between 7 and 96 and with one class that has 128 dependents.
* In the second one we have values between 0 and 43 dependents per class and then there is a jump to values between 642 and 692, and once more there is one class that has a number of 729 dependents.
* By comparing the two we can see that there are drastic changes when we had the indirectly dependents to the count.
* Having too many dependents is a sign of coupled code, what can lead to code smells like the Shotgun Surgery, which basically is when we want to make a change to the code in a place and that change causes a cascade of alterations in other places.

# Number of Package Dependencies

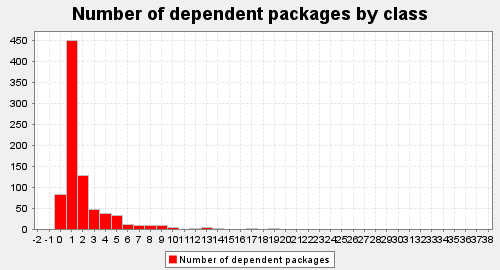
* This dependency calculates the number of packages on which class directly or indirectly depends on.



* In this histogram the values vary mostly between 0 and 10 per class, with some exceptions afterwards of values between 10 and 29.

# Number of Dependent Packages

* This dependency calculates the number of packages which directly or indirectly depend on each class.



* In this histogram we can see that the values vary mostly between 0 and 3 but there are classes with values between 3 and 36.
* In comparation with the package dependencies there are not that many changes in the numbers range, only on the quantity of each dependency in each class (the package dependencies are less).

# Dependency Metrics and Code Smells

* Too many dependencies between classes, interfaces or packages can be a sign of strong coupling, what we do not want. Low coupling is preferred so that the classes, interfaces, or packages can be as independent as possible.
* This level of coupling can cause code smells like
  + **Shotgun Surgery** - like mentioned before.
  + **Inappropriate Intimacy** - that is when two or more classes are interlinked with each other too much (for example, by having public fields instead of getters and setters, or public fields that should be private).
  + **Feature Envy** – When a method of one class uses too much the functionalities of another class.

## Relation Between the Metrics and the Found Code Smells