8. Collect and analyses at least three different software metrics related to the project.

1. LOC:

<https://en.wikipedia.org/wiki/Source_lines_of_code>

Source lines of code (SLOC), also known as lines of code (LOC), is a [software metric](https://en.wikipedia.org/wiki/Software_metric) used to measure the size of a [computer program](https://en.wikipedia.org/wiki/Computer_program) by counting the number of lines in the text of the program's [source code](https://en.wikipedia.org/wiki/Source_code). SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate [programming productivity](https://en.wikipedia.org/wiki/Programming_productivity) or [maintainability](https://en.wikipedia.org/wiki/Maintainability) once the software is produced.

LOC measures are somewhat controversial, particularly in the way that they are sometimes misused. That is, programs with larger LOC values take more time to develop. Thus, LOC can be very effective in estimating effort. However, functionality is less well correlated with LOC: skilled developers may be able to develop the same functionality with far less code, so one program with fewer LOC may exhibit more functionality than another similar program. LOC is a poor productivity measure of individuals, since a developer can develop only a few lines and yet be far more productive in terms of functionality than a developer who ends up creating more lines (and generally spending more effort). Good developers may merge multiple code modules into a single module, improving the system yet appearing to have negative productivity because they remove code. Also, especially skilled developers tend to be assigned the most difficult tasks, and thus may sometimes appear less "productive" than other developers on a task by this measure. Furthermore, inexperienced developers often resort to code duplication, which is highly discouraged as it is more bug-prone and costly to maintain, but it results in higher LOC.

2. NOC:

<http://support.objecteering.com/objecteering6.1/help/us/metrics/metrics_in_detail/number_of_children.htm>

**Number of Children (NOC)**

**Overview**

Inheritance, otherwise called generalization, is one of the fundamental concepts of object models, and must be used advisedly. Non-abusive use is a sign of quality and a good understanding of the concept. A class from which several classes inherit is a sensitive class, to which the user must pay great attention. It should, therefore, be limited, notably for reasons of simplicity.

**Computation**

For a class, this is the number of child classes.

For a package, this is the number of child packages.

**Nominal range**

Between 1 and 4.

**Analysis**

The upper and lower limits of 1 and 3 correspond to a desirable average. This will not stop certain classes being the kind of utility classes which provide services to significantly more classes than 3.

In our project, most classes NOC is 0, only has one class has 8 NOC, because that class has 8 complexity, that make our project in this part has good quality and a good understanding of the concept.

3. WMC (Cyclomatic Complicity)

<https://pdepend.org/documentation/software-metrics/weighted-method-count.html>

<http://www.arisa.se/compendium/node97.html>

**WMC - Weighted Method Count A (the base metrics)**

The Weighted Method Count or Weighted Method per Class metric was originally defined in A Metrics Suite for Object Oriented Design by Chidamber & Kemerer.

The WMC metric is defined as the sum of complexities of all methods declared in a class. This metric is a good indicator how much effort will be necessary to maintain and develop a class. There are three slightly different definitions of the WMC, where each definition uses another metric as a measure of the methods' complexity. Possible complexity values are:

* McCabe's Cyclomatic Complexity
* Lines of Code
* 1 (Number of Methods or Unweighted WMC)

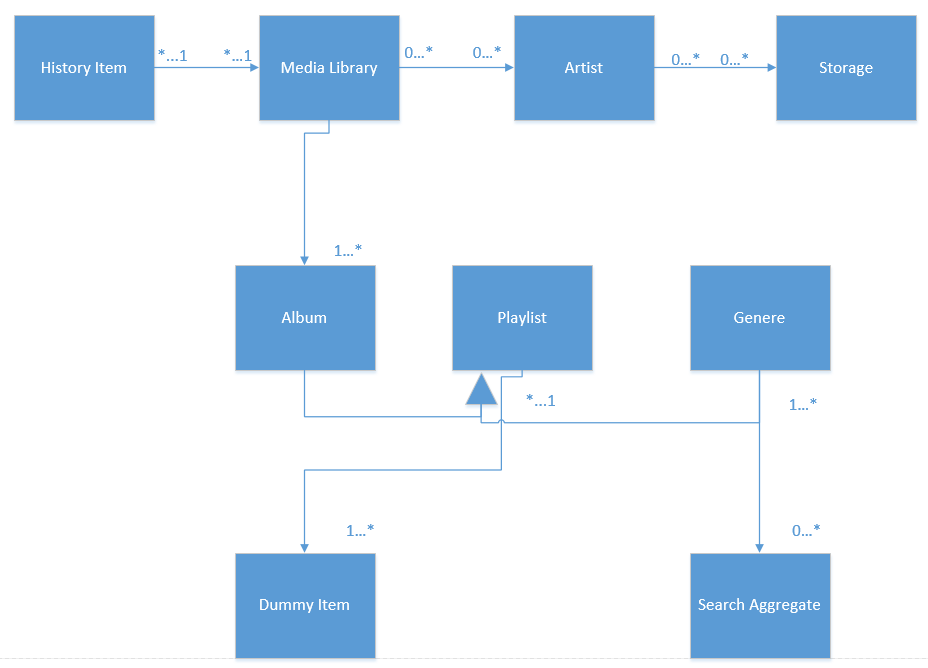
PHP\_Depend uses the sum of Cyclomatic Complexity Numbers of all methods and constructors declared in a class to calculate the WMC metric. A lower WMC usually indicates to a class with better abstraction and polymorphism. While a class with a high complexity value is a good indicator that it this class is very application specific and does more than one job, and therefore harder to test, reuse and maintain.

**Thresholds**

An appropriate threshold for the WMC lower limit is 1, because a class should at least consist of one method. An upper limit for the WMC of a class is harder to define but it seems that an upper limit of 50 is a good reference point for most projects that start to use the Weighted Method Count metric.

In our project most class has good WMC is in the good range, only has one class got high WMC, because that class has high parameters (NOP).

**CBO**

****