Code Reusability Estimation based on Static Analysis Metrics

Ioanna Apostolina, Konstantinos Samaras-Tsakiris

Electrical and Computer Engineering Deptartment

Aristotle University Of Thessaloniki

Thessaloniki, Greece

Email: {igaposto, kisamara}@ece.auth.gr

*Abstract*—The idea of reusing software components has been present in software engineering for several decades. This idea becomes more and more popular nowadays that open source software repositories are a common fact. So far, several tools have been developed to assess the software quality of these repositories which depends on fixed metric thresholds for defining the ground truth. In this work we present an organized approach that relates quality of software components with static analysis metrics and it estimates the reusability of software components in popular GitHub repositories. Our methodology includes two models: a one-class classifier, used to exclude all low quality code find inside classes of our 137 popular repositories and a neural network that computes a reusability score for each class of each repository. Evaluations shows!!!!!

Keywords—source code quality, reusability, static analysis, user-perceived quality, neural network, one-class classification

# Introduction

The numerous open source projects exist in online repositories such as GitHub have been used as a tool to reduce the development cost and time of the software. Nowadays, majority of software systems are being developed to a certain degree from an assembly of already existing reusable components. In order to assess these open source software components effectively, we should measure the *reusability* of these components which is closely related to the *quality* of the component.

Quality is a complex concept because it means different things to different people, it is highly context-dependent [2]. There was a common need for standardization of *quality* so international standard ISO/IEC 25010:2011 created. This international standard defines that the quality of a software component can be measured with eight quality characteristics to ensure *Functional Suitability*, *Usability*, *Performance* *Efficiency*, *Portability, Operability*, *Security*, *Compatibility*, and *Maintainability* [3]. However from a user's perspective quality is referred to the reusability to a software component and it is related with only four of the above characteristics: *Functional Suitability*, *Usability*, *Maintainability and* *Portability* [4].

Researchers during their attempt to measure software quality in an appropriate manner has led to the development of various metrics that we will discuss later. However most of them have a problem with determining the thresholds of these metrics. Defining a threshold for these metrics is not an easy task and it is usually performed by an expert [5]. This method is not flexible enough to describe in an effective way major software components. Nevertheless, a method that uses user-perceived quality as a measure of the reusability of a software component was introduced a year ago and had good evaluation results as we will describe in the next chapter [6].

In this paper, we **appraise the code quality in terms of reusability of a software component through the use of static analysis metrics. We do the hypothesis that popularity from a user's point of view is a measure of reusability of a software component. In other words, we use the popularity of these projects as the ground truth. In the next section, we will provide a proof for this hypothesis and examine the relationship between stars and forks for a sufficient dataset of repositories residing in GitHub. Based on this hypothesis , we design and create a system based on the four software quality characteristics:** *Functional Suitability*, *Maintainability* , *Usability* *and* *Portability* with the simultaneous use of a set of static analysis metrics. Estimation of the reusability of a software component becomes possible. Our system consists of two main models: a one-class classifier trained with a support vector machine (SVM) that shows if a class component overcomes a quality threshold, and an artificial neural network (ANN) that estimates the reusability of software components given the static analysis metrics. It is crucial to refer that the analysis and the models are built with the use of static analysis metrics referred to the classes of repositories. Thus the system is based upon finding the reusability of each class of a repository and in this way come to a conclusion about the reusability of the overall repository.

The rest of this paper is organized as follows. Section II reviews the current state-of-the-art upon the problem. Section III analyses the construction of the target set based on the rating of each repository. Section III describes the overall preprocessing of data ant the ANN which helps us with the reusability estimation. Section V evaluate our methodology and Section VI concludes the paper and provides aspects for further research.

# Research Overview

Previous years various software metrics have been proposed from research for measuring the quality of software components [7, 8] but this problem remains not an easy one. This task requires an exact specification of software metrics and it is usually accomplished by an expert. Nevertheless, the help of an expert is not always available. For this reason, various methods have been proposed.

Estimating quality characteristics has been a popular research topic for many years. Many methods have been developed about software metrics thresholds definition. Nevertheless, most of them do not be effective when it comes to real world scenarios. A common practice is the adaptable quality estimation with the design of models [9]. One other practice is by analyzing the results of numerous software metrics that are computed. Ferreira et al. have proposed a method that uses the value of the computed metrics with probability distribution of them and indicate the bounds of metrics [10]. Another common approach is the design of quality evaluation systems which target to a single quality characteristic. Kumar proposes an SVM -based classifier and in this ways builds a reusability estimation system for software components[11]. One other method works efficient is this of Papamichail et al. which given static analysis metrics and using the popularity of software repositories create a system wich combines two models: a one- class classifier and an artificial neural network that estimates the quality of the code [6].

Although most of these proposals do not work properly in real-world scenarios as we referred earlier. Firstly, they are limited within certain quality thresholds; something that shows a limitation to the extent of being objective for all classes [9]. Automated system from the other side need the knowledge of an expert for training the model so we cannot use them every time that we need to evaluate the quality of a software component. Furthermore, these systems do not offer a single output measurement the user-perceived quality.

In this work, we build a system that estimates the code reusability and provides a single metric based on a set of static analysis metrics referred to classes, related to user-perceived quality and specifically referred to classes of each repository . Given that popularity of components has a strong relationship with reuse [12], we consider that popularity means high quality of components. In other words, a repository residing in GitHub which is rated with many stars and forks possibly is a high quality project and reusable one. Based on this idea, we build a system that provides a reusability score for each repository and information about the values of metrics of classes.

# SCORING CLASSES OF REPOSITORIES

In this section, we discuss the target of our dataset and we build a metric based on the number of stars and forks.

At first, we have to define the reason that lead us to the creation of a target of our dataset that is based both on the number of forks and stars for each repository To support thic claim, we computed the correlation between stars and forks of a repository. As illustrated in figure 2, there is no strong correlation between stars and forks. Actually the value of the correlation between the two metrics is 0.474.

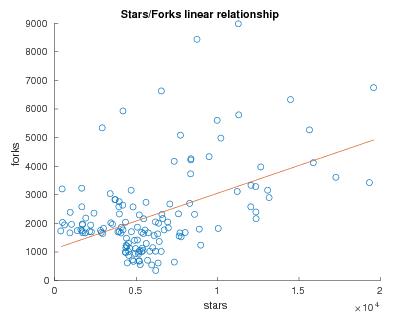


Fig. 1. Stars - Forks Diagram

The dataset consists of 60 static analysis metrics computed for the most 137 GitHub repositories .Nevertheless, the number of stars and forks is not enough and a score is required per class of each repository and reflects its reusability individually. We should note that the number of scores and stars cannot be equally distributed among the classes of a repository as the classes have not the same significance and impact for the reuse of a repository. The score for a class is given by two terms: one that includes star rating of a repository and accounts also the significance of the class as an independent entity based on stars and the second term that includes forks rating of the repository and the significance of the class based on forks. Finally the score for a class of a repository is given by the following equation:

where represents the target score for class i which is included in repository j, is the Coupling Between Object classes Inverse metric for class i, represents the number of stars for repository j, represents the number of forks, shows the number of classes into the j repository and is the Total Number Of Local Public Methods metric for class i.

Equation (1) that we wrote above consists of 2 main terms. The first one consists of two factors. The first one assigns a base score to all classes in the same repository based on the star rating of it. The second shows the significance of the class inside the repository by adding value to the score proportional to the percentage of classes of the repository j that use the class i. The logarithmic function is used as a smoothing factor of the diversity between the numerous classes into different software repositories. This is a very important function because of the fact that other repositories could have only 200 classes and other could have more than 3000. In other words it's a way to have representative results.

The second term is 0.6\*.The term assigns a base score to all the classes of the same repository based on the fork rating of a software project. The second term shows how significant is the class inside the repository by adding value proportional to the percentage of the total number of local public methods being defined inside class i of repository j. Lastly. the logarithmic function is used again as a smoothing factor of the diversity between the classes that exist inside different repositories. Thus, we should refer that the coefficient 0.4 and 0.6 that are multiplied with terms of our equation are the non - negative weights using weighted arithmetic mean method. Weights are defined based on the observation of histograms of stars and forks. Forks shown a better normal distribution than stars .

Finally, the histogram of our target shown (Figure 2) that it is a Gaussian Distribution and values taken are in the interval [0, 10].

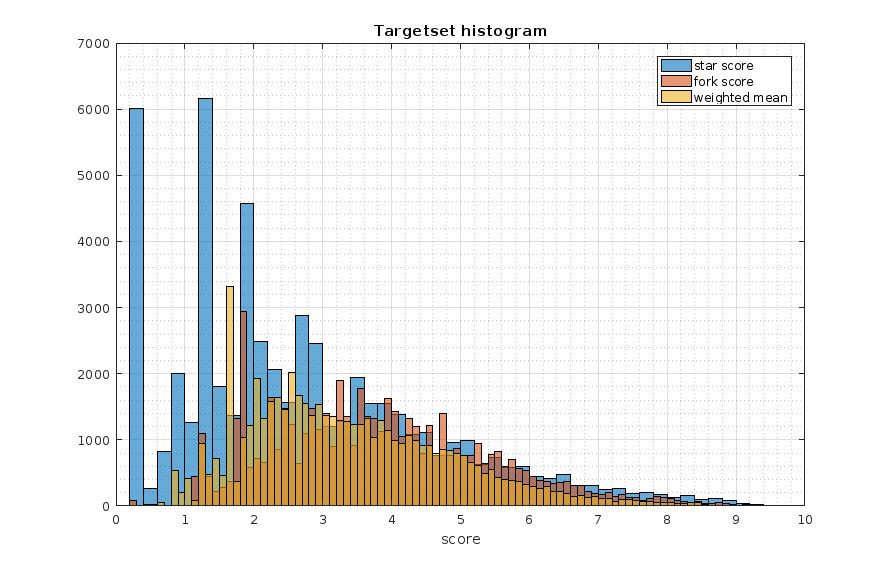


Fig. 2. Histogram of the Target Set

# System Design

This section presented the analysis of the provided dataset and gives a thorough description of our reusability estimation system.

## System Overview

In this effort to estimate quality we use a set of static analysis metrics referred to every class of our dataset so as to train two models: a one class-classifier and a artificial neural network. Before the training of the model what we do is preprocessing the rating metrics of repositories (stars and forks) in a way that will ensure a better target set and we remove low-quality and correlated metrics for our set of static analysis metrics. Figure 3 depicts the steps being taken for the final estimation of classes reusability.

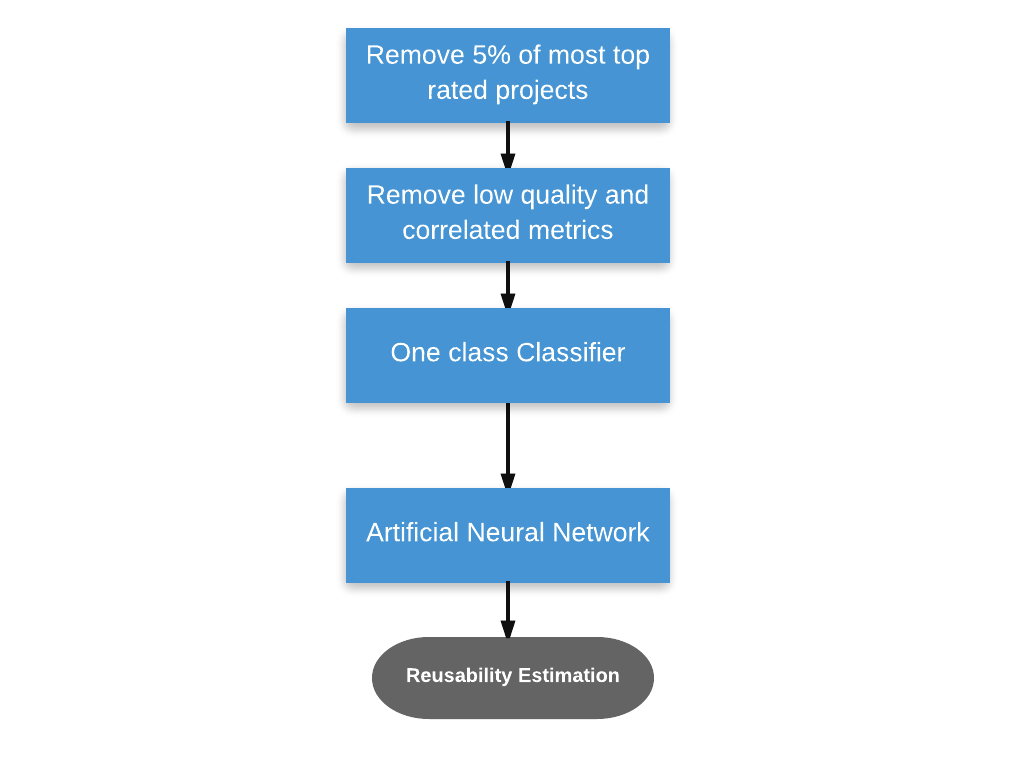


Fig.3. Overview of the Reusability Estimation System

## Metrics initial preprocessing

Before the training of our models it seems necessary to process both the rating data of the available repositories (137 in total) and the set of static analysis metrics computed for each class of a repository. At first, we remove five percent (5 %) of the most rated repositories. Projects being removed do not follow similar tendency as the greatest part of repositories. This initial step contributes to a better determinable target set. Next step is remove low quality classes out of our dataset via handpicked filters. First action taken is removing the classes that have a Number of Statements (NOS) less than five. Furthermore, we exclude from our dataset the classes that Coupling between Object classes (CBO) metric is zero which means that these classes are not used directly from other classes and consequently not reusable. One last criterion is removing the classes that do not have any defined public methods inside. The final step taken is the removal of correlated metrics that are depicted in figure 4.

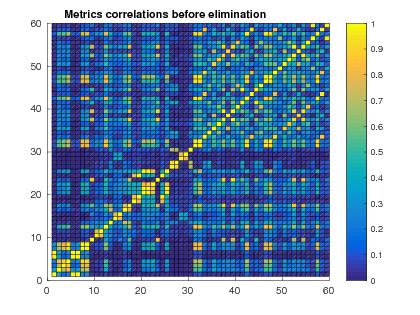


Fig.4. Correlation of static analysis metrics before elimination

We consider that the information about classes that these metrics provide us with; is given already by other static analysis metrics. Result shown in figure 5.

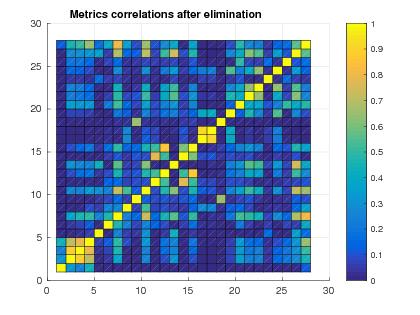
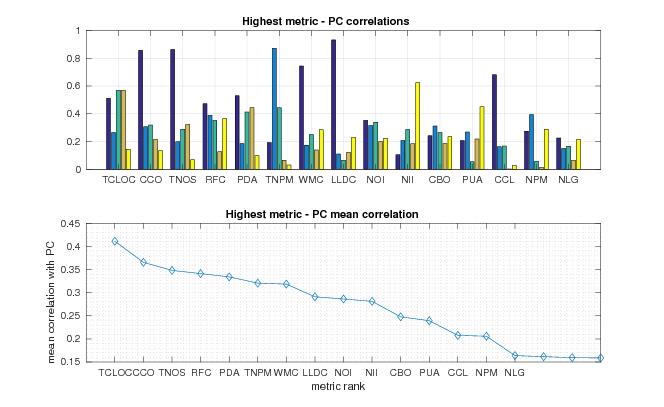


Fig. 5. Correlation of static analysis metrics after elimination

## Data Preprocessing

We use an one-class classifier to remove the code with non-acceptable quality. At first, in the effort to train this model we use principal component analysis (PCA) to select the metrics with the maximum effect on the decision taken. The results have shown (Figure 6) that the metrics with the most significant impact are Total Comment Lines of Code, Total Number of Public Methods, Clone Complexity, Number of Incoming Invocations, Total Number of Statements, Public Documented API, Response set for Class, Weighted Methods per Class, Logical Lines of Duplicated Code, Number of Outgoing Invocations and Number of Incoming Invocations. As these metrics have the greatest impact to our dataset what we do is train our one-class classifier with them.

 Fig. 6. Static Analysis Metrics - Top 5 Component correlation diagram

The Gaussian radial basis kernel function is used to train the one-class classifier that eliminates the five percent of outliers of our dataset. It is important to refer that the selection parameter nu used for the support vector machine is equal to 0.2 and the scale of the kernel is equal to 0.49.

## Artificial Neural Network Model

Our artificial neural (?)

Note that seventy percent of the data samples were used as training set, fifteen percent as tetsing set and fifteen percent for validation.

## …

…

# Evaluation

## Evaluation Methodology

Write what are the metrics that you have selected, what are the axes on which you assess your system, what are the experiments performed, etc.

## Evaluation Results

Present the results of the experiments, possibly one subsection per experiment. Create tables, figures and any other supporting material. For an example of a table see Table 1.

1. Table Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copy |  |  |

Table 1 presents …

# Conclusions

Write here the findings of your analysis.

##### References

1. IEEE Manuscript Templates for Conference Proceedings, online: http:// www.ieee.org/conferences\_events/conferences/publishing/templates.
2. Kitchenham, Barbara, and Shari Lawrence Pfleeger. "Software quality: the elusive target [special issues section]." *IEEE software* 13.1 (1996): 12-21.
3. "CISQ Code Quality Standarts." [Retrieved March 2017]. [Online]. Available: http://it-cisq.org/standards/
4. Diamantopoulos, Themistoklis, Klearchos Thomopoulos, and Andreas Symeonidis. "QualBoa: reusability-aware recommendations of source code components." *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016.
5. Zhong, Shi, Taghi M. Khoshgoftaar, and Naeem Seliya. "Unsupervised Learning for Expert-Based Software Quality Estimation." *HASE*. 2004.
6. M. Papamichail, T. Diamantopoulos and A. Symeonidis, "User-Perceived Source Code Quality Estimation Based on Static Analysis Metrics," *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Vienna, 2016, pp. 100-107.
7. Padhy, Neelamadhab, Suresh Satapathy, and R. P. Singh. "Utility of an Object Oriented Reusability Metrics and Estimation Complexity." *Indian Journal of Science and Technology* 8.1 (2017).
8. Ferreira, Kecia AM, et al. "Identifying thresholds for object-oriented software metrics." *Journal of Systems and Software* 85.2 (2012): 244-257.
9. Cai T, Lyu MR, Wong KF, Wong M. ComPARE: A generic quality assessment environment for component-based software systems. InProceedings of the 2001 *International Symposium on Information Systems and Engineering 2001*.
10. Ferreira, Kecia AM, et al. "Identifying thresholds for object-oriented software metrics." *Journal of Systems and Software* 85.2 (2012): 244-257.
11. Kumar, Ajay. "Measuring Software reusability using SVM based classifier approach." *International Journal of Information Technology and Knowledge Management* 5.1 (2012): 205-209.
12. Borges, Hudson, Andre Hora, and Marco Tulio Valente. "Predicting the popularity of github repositories." *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2016.