STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIX**, Number 1, 2014

ON EVALUATING THE STRUCTURE OF SOFTWARE PACKAGES

ZSUZSANNA MARIAN

Abstract. In this paper we present a study on how a measure previously introduced in the literature can be used to evaluate the structure of software packages in a software system. Three open-source case studies are used and for each case study the value of this measure for four different divisions into packages is investigated. For these case studies we compute the value of other metrics from the literature as well, and a comparison on how they evaluate these divisions is given. We conclude about the relevance of the analyzed measure for evaluating the quality of a partitioning into packages.

1. Introduction

Modern software systems are increasingly complex, made of a great number of different components, which are usually organized into some kind of hierarchical structures. For example, in case of object-oriented systems, the components of the system are the classes, which are organized into packages. Finding how exactly should classes be divided between the packages is not at all a trivial problem, since many different criteria have to be considered, when designing these packages. Usually, the main criterion is related to the dependencies between the classes, but others can be used as well: how similar the names of the classes are, what other classes are used by the classes in the package, and so on.

How classes are organized into packages depends also on the architecture of the software system. There is a general rule, "low coupling, high cohesion", but this cannot be applied for every software system. While in case of frameworks this rule is true, one wants separate, loosely coupled packages, in case of a layered architecture there are relations between elements belonging to different

Received by the editors: April 2, 2014.

2010 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.8 [**Software Engineering**]: Metrics – Performance measures; D.2.10 [**Software Engineering**]: Design.

Key words and phrases. software engineering, software package, evaluation measure.

layers, and classes belonging to visualization and business layers should not be placed in the same package, just to reduce coupling between packages.

In [11] we have proposed a novel clustering-based [9] method, called *HASP*, for restructuring classes into packages in a software system. Our method, applicable in case of frameworks, is based on the value of several features, aggregated into a single score, which was used as a distance measure during the clustering process. In the same paper we have introduced a second measure, which can evaluate how well-structured a software system is, with respect to an apriori known good structure. In this paper we present a study performed in order to evaluate how well packages in a software system are structured using the evaluation measures introduced in [11]. In our study we will consider and evaluate four different package structures for three open-source software systems using the above-mentioned measures.

The rest of this paper is structured in the following way: Section 2 presents a background on package structure measures existing in the literature. Section 3 represents the experimental part of our paper, consisting of the description of the case studies and of the performed experiments. The results of the experiments, their analysis and a comparison to other metrics presented in the literature are given in Section 4. Finally, Section 5 concludes the paper and outlines some further research directions.

2. Background

In this section we will briefly present the evaluation measure previously introduced in [11] that will be used in our experiments to evaluate how well a software system is divided into software packages. A brief review of other metrics presented in the literature to measure how well the packages of a software system are structured will also be presented.

2.1. **Evaluation measure for software package structures.**

In [11] we have introduced a new evaluation measure (Formula 1), which measures how well one package is structured in a software system. In order to formally define this measure, we will consider that a software system S is a set of application classes, $S = \{s_1, s_2, ..., s_n\}$. The set $\mathcal{K} = \{K_1, K_2, ..., K_v\}$ is called a partition into packages of the software system S iff:

- $1 \leq v \leq n$;
- $K_i \subseteq S,\ K_i \neq \emptyset,\ \forall i,\ 1 \leq i \leq v$;
- $\bigcup\limits_{i=1}^{v} K_i = S,\ K_i \bigcap K_j = \emptyset,\ \forall i, j,\ 1 \leq i, j, \leq v,\ i \neq j$.

In order to characterize how well-structured a package K_i is, with respect to a partition $\mathcal{K}$, we have defined five features. In defining these features we have concentrated on characterizing packages from frameworks, and tried to go beyond the simple "low coupling, high cohesion" principle. The five features for a package P, as presented in [11], are the following:

- F_1 **- Package cohesion** - measures how cohesive the classes inside a package P are, by counting the dependencies between elements of the package P.
- F_2 **- Package reuse** - measures how many packages from the system depend on package P.
- F_3 **- Package coupling** - measures on how many packages from the system does the package P depend on.
- F_4 **- Name cohesion** - measures how similar the names of the classes in package P are.
- F_5 **- Dependency similarity** - measures how similar the classes on which classes from package P depend on are.

Using the above presented five features we have defined a measure to evaluate the division into packages of a whole software system. For a partition $\mathcal{K}$ of a software system, this measure, called *overallScore*, is computed in the following way [11]:

$$overallScore(\mathcal{K}) = \frac{\sum_{i=1}^{v} sc(K_i, \mathcal{K})}{v} \tag{1}$$

where sc is a score that denotes how well-structured a package K_i is, with respect to the partition $\mathcal{K}$, and is computed as [11]:

$$sc(K_i, \mathcal{K}) = \begin{cases} \frac{\sum_{i=1}^{2} w_i * F_i - w_3 * F_3}{|K_i|^2 - 1} + \sum_{i=4}^{5} w_i * F_i, & if\ |K_i| > 1 \\ 0, & otherwise \end{cases} \tag{2}$$

In Formula (2) F_i $(1 \leq i \leq 5)$ denotes the features presented above, while w_i $(1 \leq i \leq 5)$ represents some weights associated to these features.

2.2. Literature Review.

There are several methods presented in the literature for evaluating the quality of a software package. A set of 13 such metrics is presented by Sarkar et al. in [13]. They consider a large set of possible relations between classes and methods, and define metrics which measure, for example, module interaction, intermodule coupling, association-induced coupling, and so on. Unfortunately, their metrics have a disadvantage: they consider that each module from a software system has some declared APIs (which can be of two types, Service API and Extension API), but there are many cases, when such APIs are not defined, so their metrics cannot be computed.

This disadvantage is overcome in [6] and [10], where a set of non-API based metrics are proposed for evaluating the remodularization of a software system. Ducasse et al. firstly define the modularity principles on which their metrics are based (for example, information hiding, encapsulation and changeability) then introduce seven metrics that measure these principles. The metrics are based on two different types of dependencies between the packages (constructed from dependencies between the classes from the packages): extension and usage.

A different set of package metrics is introduced by Abreu et al. in [7], not just for characterizing the modularization of a software systems, but also for remodularizing it in a better structure, using a clustering-based approach. They consider 12 different kinds of relations/coupling types between classes, such as: direct inheritance, attribute type, return in operation, local attribute, and so on, but also claim that only minimizing coupling and increasing cohesion is not sufficient for achieving a good package structure.

Ponisio et al. in [12] propose one single cohesion metric for packages, called *Common-Use*. In order to compute it, they consider four types of dependencies/interactions: inheritance, state, class reference and message sends. The main idea behind the *Common-Use* metric is that if all clients of a package use all classes from the package together, then the package is cohesive, even if the classes from the package do not use each other directly.

3. Main results

In this section we will describe the experiments that we have performed in order to evaluate different package structures for software systems. More precisely, for three different software systems (described in Section 3.2) we will evaluate four different package structures using the *overallScore* measure presented in Section 2.1.

3.1. The CIP measure.

Besides *overallScore*, we have introduced in [11] another measure, called *CIP - Cohesion of Identified Packages*, which adapts the measure introduced in [8] for evaluating the results of aspect mining techniques. The value of *CIP* measures how close one partition (one division of classes into packages) is to another partition. More exactly, given a partition known to be correct, denoted by $\mathcal{K}^{good}$, and another partition, denoted by $\mathcal{K}$, $CIP(\mathcal{K}^{good}, \mathcal{K})$ measures the cohesion of the packages from $\mathcal{K}^{good}$ in $\mathcal{K}$. *CIP* is computed in the following way [11]:

$$CIP(\mathcal{K}^{good}, \mathcal{K}) = \frac{1}{q} \sum_{i=1}^{q} cip(K_i^{good}, \mathcal{K}). \tag{3}$$

where q is the number of packages in the partition $\mathcal{K}^{good}$, K_i^{good} is the i-th package in $\mathcal{K}^{good}$ and $cip(K_i^{good}, \mathcal{K})$ measures the cohesion of package K_i^{good} in partition $\mathcal{K}$ defined as:

$$cip(K_i^{good}, \mathcal{K}) = \frac{\displaystyle\sum_{k \in M_{K_i^{good}}} \frac{|K_i^{good} \cap k|}{|K_i^{good} \cup k|}}{|M_{K_i^{good}}|} \tag{4}$$

where $M_{K_i^{good}}$ is:

$$M_{K_i^{good}} = \{k | k \in \mathcal{K}, K_i^{good} \cap k \neq \emptyset\} \tag{5}$$

The value of the *CIP* measure is always between 0 and 1, the value of 1 being achieved when $\mathcal{K}$ coincides with the good structure $\mathcal{K}^{good}$. The higher the value of *CIP*, the better the structure of $\mathcal{K}$ with respect to $\mathcal{K}^{good}$.

3.2. **Case studies.** Since both the *overallScore* measure and the five features presented in Section 2.1 were defined in order to evaluate software systems which are frameworks, we have chosen three different open-source frameworks for this case study. All three systems are part of the *Apache Commons* [4] project.

The first case study is the *DbUtils* framework, version 1.5, available at [1], which is a small set of classes designed to make working with JDBC easier. It consists of 25 classes, divided into three packages: *handlers*, *wrappers* and the *default* package.

The second case study is the *Email* framework, version 1.3.2, available at [3], which is a framework for sending emails, built on top of the Java Mail API, but tries to simplify it. It consists of 19 classes, divided into three packages: *resolver*, *util* and the *default* package.

The third case study is the *EL* framework, version 1.0, available at [2], a JSP 2.0 Expression Language interpreter. It consists of 57 classes, divided into two packages: *parser* and the *default* package.

The reasons for choosing these software systems as case studies are the following:

- All three of them are frameworks, which is very important, since the *overallScore* measure was defined for frameworks.
- They are openly available.

- They have a relatively small number of classes, which allows manual verification and analysis.

3.3. **Experiments.** Through the performed experiments we aim at emphasizing that the *overallScore* measure is well-correlated with the *CIP* measure. Thus, instead of *CIP*, which requires the apriori knowledge of a good partition, *overallScore* can be used for evaluating a software package structure.

For all three open-source cases studies presented in Section 3.2 we have considered four different package structures, created in the following way:

- **Original** - is the original package structure for the system.
- **HASP** - the package structure indicated by our *HASP* approach introduced in [11].
- **WP1** - a "wrong partition" created by taking some classes from the packages in the original package structure and moving them into one or more newly created packages. In this way, part of the original structure is kept, but new, incorrect, packages are created.
- **WP2** - a second "wrong partition" created by dividing randomly all the classes into packages.

During the experimental evaluation we have computed the value of the *overallScore* measure for each partition. We have also computed the value of the *CIP* measure for each partition. When computing the *CIP* measure, one has to provide a correct partition, $\mathcal{K}^{good}$. Out of the four partitions, *WP1* and *WP2* cannot be considered as a correct partition, so we had to decide between the original partition and the one provided by our algorithm. After an analysis of both partitions, we have decided to consider the partition provided by the *HASP* approach as the correct one for the *DbUtils* and the *EL* system, while for the *Email* system we have computed the *CIP* measure twice, once considering as $\mathcal{K}^{good}$ the original partition and once the *HASP* partition. In the following these values will be denoted by CIP_O and CIP_H, respectively. Our reasons for these decisions are:

- For the *DbUtils* system we have explained in detail in [11] why we consider the *HASP* partition better than the original one.
- In case of the *Email* system, there are four classes which are placed in a different package in the *HASP* partition than in the original one. Out of these four classes we consider that two were moved correctly (they should rather be in that package), but the other two were better placed in their original package. This is why we consider both the original and the *HASP* partition as a possibly correct partition when computing the value of the *CIP* measure.

- In case of the *EL* system, there is quite a big difference between the original and the *HASP* partition. The original partition has two packages, while the *HASP* partition has seven. Out of these seven packages, one corresponds exactly with the *parser* package from the original partition, while the remaining six packages contain the classes which were originally in the *default* package. Out of these classes, divided in the six packages, our analysis concluded that six classes are not placed correctly. Even with these misplaced classes, we consider that this structure is better than one huge package containing 50 classes.

For computing the *overallScore* measure, values for the weights from Formula (2) are needed. In [11] we have described a grid-search process, through which we identified good values for the weights. Thus, for the experiments presented in this paper, we used the weights reported in [11], namely: $w_1 = 0.22$, $w_2 = 0.25$, $w_3 = 0.2$, $w_4 = 0.52$ and $w_5 = 0.72$.

4. Results and discussion

For all three open-source case studies presented in Section 3.2 we have computed both the *overallScore* and the *CIP* measure, for all four package structures described in Section 3.3. The results are presented in Tables 1, 2 and 3.

Analyzing the values in these tables we can observe that in case of the *overallScore* measure the two "wrong partitions", *WP1* and *WP2*, have lower *overallScores* than the two other partitions. Moreover, *WP2*, the partition where the classes were randomly assigned to the packages, has always the lowest *overallScore* value. Considering the *Original* and the *HASP* partitions, we can see that in case of the first two projects, the *Original* partition has higher *overallScore*, while for the *EL* project, *overallScore* is higher for the *HASP* partition.

Analyzing the values of the *CIP* measure we can see that it always has the value 1 for the *HASP* partition, which is caused by the fact that we consider the *HASP* partition the correct one, with the exception of the *Email* system where

Partition	**OverallScore**	**CIP**
Original	0.9009	0.4183
HASP	0.7349	1
WP1	0.5658	0.4146
WP2	0.2436	0.1372

Table 1. Values for the *overallScore* and *CIP* measures for the *DbUtils* system.

Partition	**OverallScore**	CIP_O	CIP_H
Original	1.146	1	0.6874
HASP	0.9696	0.6688	1
WP1	0.8335	0.54	0.4895
WP2	0.4026	0.1781	0.1912

Table 2. Values for the *overallScore* and *CIP* measures for the *Email* system.

Partition	**OverallScore**	**CIP**
Original	0.5241	0.2857
HASP	0.8261	1
WP1	0.3752	0.2672
WP2	0.1601	0.1299

Table 3. Values for the *overallScore* and *CIP* measures for the *EL* system.

we consider as the correct partition the *Original* one as well. For *DbUtils*, *EL* and CIP_H for *Email* (i.e., the partition provided by the *HASP* algorithm for the *Email* system), the order of partitions given by the *CIP* measure is the same: the *Original* partition has the second highest value, *WP1* the third one and finally, *WP2* has the lowest value. In case of CIP_O for *Email* (i.e., the original partition for the *Email* system), the *Original* partition has the highest value and the *HASP* partition has the second highest value.

In order to see how correlated these values are, we have computed two rank-based correlation measures between the *overallScore* and the *CIP* values: the Spearman correlation [14], [15] and Spearman's footrule [5], which is computed as the sum of the absolute values of the differences between the ranks of the elements. These values are presented in Table 4. Since for the *Email* system we computed two *CIP* values, we will have two lines with correlation values as well. From this table we can see that the values for the correlations are quite high, especially for the *EL* system and CIP_O for *Email*, both having perfect values. The lowest correlation values are achieved for the *DbUtils* system and CIP_H for *Email*, but even these values show a strong positive correlation between the two measures. In case of Spearman's footrule a lower value corresponds to a better association, so the values for Spearman's footrule from Table 4 show a strong positive correlation of *overallScore* and *CIP* as well.

Project	**Spearman correlation**	**Spearman's footrule**
DbUtils	0.8	2
Email - CIP_O	1	0
Email - CIP_H	0.8	2
EL	1	0

TABLE 4. Correlations between *overallScore* and *CIP* for the three case studies.

In order to compare our evaluation measure with other measures from the literature, we computed the value of the seven metrics introduced in [6]. These metrics are:

- IIPU - Index of Inter-Package Usage.
- IIPE - Index of Inter-Package Extending.
- IPCI - Index of Package Changing Impact.
- IIPUD - Index of Inter-Package Usage Diversion.
- IIPED - Index of Inter-Package Extending Diversion.
- PF - Package Focus.
- IPSC -Index of Package Services Cohesion.

For each metric, we considered as extending relation the implementation of interfaces as well. Each metric takes values in the $[0, 1]$ interval, according to [6], higher values correspond to better package structures. The values of these metrics for the four partitions of the three case studies are presented on Tables 5, 6 and 7.

Analyzing the values from Tables 5, 6 and 7 we can observe that generally the same tendency is followed as for the *overallScore* and *CIP* measures: out of the total of 21 cases, in 20 the smallest value was for the *WP2* partition (but in the remaining one case it has a value of 1, which means a perfect structure). Moreover, in 16 cases, the values for the *Original* and the *HASP* partitions are higher than the ones for *WP1* and *WP2*.

Partition	**IIPU**	**IIPE**	**IPCI**	**IIPUD**	**IIPED**	**PF**	**IPSC**
Original	0.4286	0.5333	0.8333	1	1	1	1
HASP	0.2857	1	0.8333	1	1	1	1
WP1	0.1429	0.6667	0.65	0.9	0.9	0.8333	0.9667
WP2	0.1429	0.1333	0.15	0.8	0.5333	0.7667	1

TABLE 5. Values for metrics from [6] for the *DbUtils* system.

Partition	**IIPU**	**IIPE**	**IPCI**	**IIPUD**	**IIPED**	**PF**	**IPSC**
Original	0.8889	1	0.8333	1	1	1	1
HASP	0.7778	1	0.8333	1	1	1	1
WP1	0.6667	0.5	0.7	1	0.9	0.9	1
WP2	0.2222	0.25	0.25	0.9167	0.75	0.6806	0.9167

Table 6. Values for metrics from [6] for the *Email* system.

Partition	**IIPU**	**IIPE**	**IPCI**	**IIPUD**	**IIPED**	**PF**	**IPSC**
Original	0.9362	1	0.5	1	1	1	1
HASP	0.3404	0.8571	0.7381	0.8881	0.9286	0.9524	0.9841
WP1	0.4894	0.3143	0.4167	0.9167	0.625	0.8125	0.9688
WP2	0.1064	0.1714	0.3	0.6347	0.5	0.4972	0.8901

Table 7. Values for metrics from [6] for the *EL* system.

Still, there are a lot of cases, when the values of a metric are the same for more partitions, in nine cases two partitions have the same value (usually *Original* and *HASP*, having a value of 1) and in three cases there are three equal values, which shows that the metrics can not always differentiate between two partitions. Moreover, we have computed the values of these seven metrics for a fifth partition, one where all classes are in the same package. We did not use this partition for the rest of the experiments, because it is a trivial one, but in case of these seven metrics we noticed that when all classes are in the same package, the value of these metrics is 1, suggesting a perfect "modularization".

To compare the value of these metrics with the value of the *overallScore* measure, we have decided to compute the value of Spearman's footrule for the *overallScore* measure and some of the metrics. In order to avoid having equal ranks we have only considered those metrics which had four distinct values for the partitions. Thus, for the *DbUtils* system we have only considered the IIPE metric, for *Email* only the IIPU metric and for the *EL* system we considered all seven of them. The values for Spearman's footrule for these projects are presented on Table 8.

A visual comparison of the footrule is presented on Figure 1, where dark gray bars represent the footrule values between the *overallScore* and *CIP* measures (presented in Table 4), while the light gray bars represent footrule values between *overallScore* and metrics from [6] (presented in Table 8). In case of systems where we had multiple footrule values (there were two values for the *Email* system in Table 4 and there were seven different values for *EL* in Table 8) we used the average of the values. From Figure 1 we can see that for the

Project	**Metric**	**Spearman's footrule**
DbUtils	IIPE	4
Email	IIPU	0
EL	IIPU	4
	IIPE	4
	IPCI	0
	IIPUD	4
	IIPED	2
	PF	2
	IPSC	2

Table 8. Spearman's footrule for the case studies between the value of *overallScore* and some metrics from [6]

DbUtils and *EL* systems the light gray bars are a lot higher than the dark gray ones, meaning that the footrule values between *overallScore* and *CIP* are a lot better. In case of the *Email* system this is reversed, the footrule value between *overallScore* and other metrics is lower.

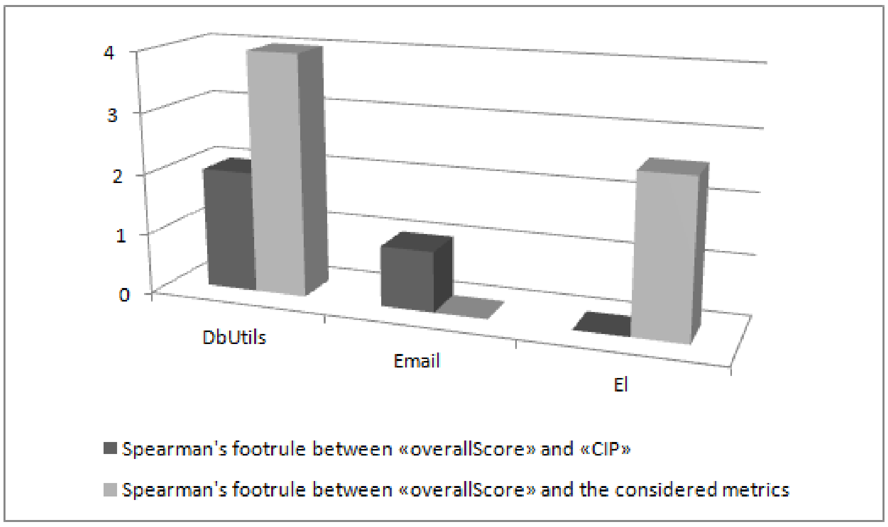

Figure 1. Comparison of footrule values.

Considering the comparison of the footrule values presented in Figure 1 and the fact that the considered metrics, presented in [6], often return equal values for different partitions, we can conclude that the *overallScore* measure is more suitable for evaluating a partition of a software system than the other metrics. Moreover, since the values of the *overallScore* and *CIP* measures are strongly positively correlated, as presented in Table 4, *overallScore* can be used instead of *CIP*. This is important, since *CIP* measures how close a partition is to an apriori known correct partition, but when one has to restructure into packages a software system, the good partition is not known. Since the values of the two measures are correlated, *overallScore* can be used, instead of *CIP*, to evaluate a partition.

5. Conclusions and further work

In this paper we have presented a study on the value of two measures designed to evaluate different modularizations of a framework software system. We have considered three open-source case studies and four different partitions for each of them and computed the value of two measures, *overallScore* and *CIP* for them. These two measures were used to automatically restructure a framework into packages. We have also computed the values of other metrics presented in the literature that measure modularization of a software system for these case studies.

Analyzing the values of these metrics and the correlations between them, we have observed that our measures, *overallScore* and *CIP*, are capable of differentiating between a good and a bad partitioning of a software system, which did not always happen in case of the other metrics taken from the literature.

As further work, we would like to perform experiments on other open-source case studies, and compare our measures to other metrics reported in the literature, besides the ones used in this paper. Since our *overallScore* measure was defined for software systems which are frameworks, we would also like to define such a score for systems with other architectures.

References

[1] Dbutils. http://commons.apache.org/proper/commons-dbutils/.

[2] El. http://commons.apache.org/proper/commons-el/.

[3] Email. http://commons.apache.org/proper/commons-email/.

[4] Apache commons project. http://commons.apache.org/.

[5] Persi Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society*, 39(2):262–268, 1977.

[6] Stéphane Ducasse, Nicolas Anquetil, Usman Bhatti, and Andre Cavalcante Hora. Software metrics for package remodularization. Technical report, Institut National de Recherche en Informatique et en Automatique, 2011.

[7] Fernando Brito e Abreu and Miguel Goulão. Coupling and cohesion metrics as modularization drivers: Are we being over-persuaded? In *Software Maintenance and Reengineering, 2001. Fifth European Conference on*, pages 47–57, 2011.

[8] Istvan Gergely Czibula Gabriela Czibula, Grigoreta Sofia Cojocar. Evaluation measures for partitioning based aspect mining techniques. *International Journal of Computers, Communications & Control*, VI(1):72–80, 2011.

[9] Jiawei Han. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[10] Houari Sahraoui Hani Abdeen, Stéphane Ducasse. Modularization metrics: Assessing package organization in legacy large object-oriented software. In *International WorkingConference on Reverse Engineering*, pages 394–398, 2011.

[11] Zsuzsanna Marian, Gabriela Czibula, and Istvan Gergely Czibula. Software packages refactoring using a hierarchical clustering-based approach. *Information Systems*, 2014. Under review.

[12] Laura Ponisio and Oscar Nierstrasz. Using contextual information to asses package cohesion. Technical report, 2006.

[13] Santonu Sarkar, Avinash C. Kak, and Girish Maskeri Rama. Metrics for measuring the quality of modularization of large-scale object-oriented software. *IEEE Transactions on Software Engineering*, 34(5):700–720, 2008.

[14] C. Spearman. The proof and measurement of association between two things. *Amer. J. Psychol.* ***15***, pages 72–101, 1904.

[15] Kelly H. Zou, Kemal Tuncali, and Stuart G. Silverman. Correlation and simple linear regression. *Radiology*, (227):617–622, 2003.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Kogălniceanu 1, 400084 Cluj-Napoca, Romania
E-mail address: marianzsu@cs.ubbcluj.ro