Communications and Network, 2016, 8, 179-192
Published Online August 2016 in SciRes. http://www.scirp.org/journal/cn
http://dx.doi.org/10.4236/cn.2016.83018

Software Reusability Classification and Predication Using Self-Organizing Map (SOM)

Amjad Hudaib, Ammar Huneiti, Islam Othman

Department of Computer Information Systems, King Abdullah II for Information Technology, The University of Jordan, Amman, Jordan
Email: Ahudaib@ju.edu.jo, A.huneiti@ju.edu.jo, islamalqaryouti@yahoo.com

Received 26 July 2016; accepted 14 August 2016; published 17 August 2016

Abstract

Due to rapid development in software industry, it was necessary to reduce time and efforts in the software development process. Software Reusability is an important measure that can be applied to improve software development and software quality. Reusability reduces time, effort, errors, and hence the overall cost of the development process. Reusability prediction models are established in the early stage of the system development cycle to support an early reusability assessment. In Object-Oriented systems, Reusability of software components (classes) can be obtained by investigating its metrics values. Analyzing software metric values can help to avoid developing components from scratch. In this paper, we use Chidamber and Kemerer (CK) metrics suite in order to identify the reuse level of object-oriented classes. Self-Organizing Map (SOM) was used to cluster datasets of CK metrics values that were extracted from three different java-based systems. The goal was to find the relationship between CK metrics values and the reusability level of the class. The reusability level of the class was classified into three main categorizes (High Reusable, Medium Reusable and Low Reusable). The clustering was based on metrics threshold values that were used to achieve the experiments. The proposed methodology succeeds in classifying classes to their reusability level (High Reusable, Medium Reusable and Low Reusable). The experiments show how SOM can be applied on software CK metrics with different sizes of SOM grids to provide different levels of metrics details. The results show that Depth of Inheritance Tree (DIT) and Number of Children (NOC) metrics dominated the clustering process, so these two metrics were discarded from the experiments to achieve a successful clustering. The most efficient SOM topology [2 × 2] grid size is used to predict the reusability of classes.

Keywords

Component Based System Development (CBSD), Software Reusability, Software Metrics,

How to cite this paper: Hudaib, A., Huneiti, A. and Othman, I. (2016) Software Reusability Classification and Predication Using Self-Organizing Map (SOM). *Communications and Network*, **8**, 179-192. http://dx.doi.org/10.4236/cn.2016.83018

Classification, Self-Organizing Map (SOM)

1. Introduction

Reuse-Based Software Engineering "is a software engineering strategy where the development process is geared to reusing existing software" [1]. There are many advantages that are referred to reusability, such as cost and time reduction, increasing the quality of software [2], reducing the cost of implementation [3].

Software metrics is "a measure of some property of software artifacts or its specifications" [4]. Software metrics can be used as indicator of software quality, help the project managers to assess and control the development lifecycle and help the developers to evaluate the quality of software [5]. Using metrics becomes manifest in reusability measurement in Object-Oriented (OO) paradigms [6].

Reusability is an important aspect in Component Based System Development (CBSD) [7]. Software components could be programs, part of programs, classes, modules, even architectures and test cases [8]. Component can be viewed as an independent "black box" with a high level of abstraction, and has a public interface to integrate with other components. The ability to define the reusable components becomes critical task to avoid the failure of reuse, especially in OO components [9]. Analyzing software metric values will help developers to identify the most suitable reusable components, to avoid developing components from scratch [5].

This paper proposes Object Oriented software reusability classification and prediction model. The prediction model uses software metrics to predict the ability of reusing components. By adopting the prediction model, we can define which components can be reused in an early stage of software development. This model can be used continuously to assess the ability of reusing existing components using the Self-Organizing Map (SOM) technique.

In this paper, we applied SOM as a clustering technique in order to help in categorizing the reusable software components using Chidamber and Kemerer (CK) metrics suite. The classification process is based on software metrics values to discover the component reusability level (High Reusable, Medium Reusable and Low Reusable).

The rest of this paper is organized as follows. Section 2 covers the software reusability concepts and related work. Section 3 provides a brief description of Self-Organizing Map (SOM) as a data classification technique. In Section 4, the methodology followed in this paper is outlined. Section 5 describes the experimental environment and the experiments that have been applied. The analyzed results are also presented in this section.

2. Software Reusability

Weinreich and Sametinger define the goal of CBSD; is to produce fine-grain components that could be reusable, rather than develop a software system and reuse it as whole. But the reuse of component at a finer level of granularity needs standards to support the connection of these finer components. Existing code is one of the richness assets in legacy systems. OO is a rich method of providing objects that share state and consists of building blocks of software systems which could be reused. Both of classes and components are define the object behavior, and both have a high level of abstraction (interfaces) to define their internal functionality. The components can be implemented using one class or more [10]. Reusing of OO under the component based development approach was accepted as promising reuse methodology [11]. OO classes are hard to be reused, because they are usually depend on each other and don't work separately [12].

Chidamber and Kemerer (CK) metrics is one of the most popular and known software metrics suites. These metrics are Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between Objects (CBO), Response for a Class (RFC), and Lack of Cohesion in Methods (LCOM).CK suite was applied in this paper, to measure the reusability. There are rigid reasons for applying traditional CK metrics suite which are, CK metrics have been widely used by researchers and experimented many times, the relationships between these metrics and software quality were validate and proved [6] [13]-[17]. CK metrics are the base of most of the other proposed metrics [18], and CK metrics have a threshold values which have been investigated in literatures [19] [20].

Many studies introduced different methods to evaluate software quality attributes, based on software metrics. In [21] investigate if there is a relationship between the class growth and coupling and cohesion metrics values. Then

the authors used statistical methods to measure reusability of components (class) using fan-in and fan-out metrics. The empirical investigation based on the idea that a high value of fan-in coupling metric indicates that class called by many classes, so it is reusable. Low fan-in metric value indicates low coupling and encapsulation. [22] proposed a new metrics to measure the component's quality attributes, which are complexity, customizability, and reusability The study measures the reusability based on two approaches. The first approach is calculating Component Reusability (CR) based on its interface methods. CR calculates the component itself reusability. The second approach is measuring the reusability level by using Component Reuse level (CRL) metric. CRL metric is based on Line of Code (LOC) metric and the Functionality of the component. [23] proposed a new reusability model, inspired from the reusability model given by REBOOT [24] with the difference of criteria and used metrics. [25] used the factors of previous study and forms empirical evaluation using different java bean components. [13] described nine traditional metrics and define their applicability to define quality attributes. They define five of sex CK's metrics that are related to the reusability evaluation, which are WMC, LCOM, DIT, CBO, and NOC. This paper uses the traditional CK metric suite to predict the software reusability quality factor. The prediction based on using neural network (SOM) to find software reusability level. [4] considered the adaptability, compose-ability and complexity important factors for measuring the reusability. [26] proposed two new metrics relates with the effect of the templates with inheritance in object-oriented software quality The first proposed metric Generic Reusability Ratio (GRr) measure the effect of using templates on the program volume. The second is Effort Ratio (ER) measure the effect of the templates on the whole development efforts. The proposed metrics were motivated by Halstead's metrics, and were experimented using programs written with and without templates, to measure the impact of using the templates on reusability. [27] validated CK metrics suite using six different systems, to discover the relationship between the metrics. They found there is a relationship between WMC, LCOM, and CBO. The classes with high values of WMC are also have a high values of LCOM and CBO, which lead to high complexity of the system and have a negative impact on the quality attributes like reusability. Low values of NOC and DIT indicates a disregard of inheritance property. The most frequently used metrics are proposed by Chidamber and Kemmerer (CK metrics). Their metrics suite has been proven by researcher that they are targeting the object-oriented quality measurements, as mentioned in [28].

In this paper we applied SOM as clustering technique to help in categorizing the reusable software components using Chidamber and Kemerer (CK) metrics suite. The clustering process is based on software metrics values to discover the component reusability level (Reusable, Medium Reusable, and Low Reusable).

3. Self-Organizing Map (SOM)

Self-Organizing Map (SOM) is one of well-known algorithm in pattern recognition and classification. SOM is an ANN model that is based on competitive learning and is an unsupervised learning paradigm [29] [30].

Kohonen's Self-Organizing Map uses an arranged set of neurons usually in 2-D rectangular or hexagonal grid [31]. Data reduction into 2-D dimensionality from high dimensionality is effective in approximation of similarity relations [32] [33] and also useful for data visualization to help in determining classes or similar patterns [34]. SOM transfers the arbitrary dimensions of incoming input data signals into one or two-dimensional map [35], and learn or discover the underlying structure of the input data. SOM has two layers of neurons; an input layer and an output layer. Each input vector is fully connected to each neuron in the output layer. The choice of the SOM grid size is determined by the degree of details of the findings; more generalization of finding requires less grid size than more detailed one [36].

In SOM grid the similar clusters are neighbors, and different clusters are far from each other on the grid. This is called spatial autocorrelation.

SOM has been widely used in many applications ranging from image processing [37] [38], speech recognition [39] [40], condition monitoring of processes [41], cloud classification, and micro-array data analysis. In the field of software engineering, SOM has been used in many researches, but not many are concerned with how to identifying metric based reusable components. [42] apply SOM to the description of the software components, in order to categorize these components in repositories that derived from software manual and the results were very promising. [43] compare SOM and GHSOM in clustering software components in the repositories. [44] develop Software Self-Organizing Map (SSOM) that is based on SOM, to classify software modules. [45] proposed a Software Reuse Methodology based on SOM to promote UNIX commands software components reuse. [36] discussed the applicability of SOM in analysis and visualization of software modules, based on its metrics values,

to detect software quality measures. The proposed approach is useful for designer to examine and form a deep understating of the clusters to recognize the module measure. [7] define a Neuro-fuzzy model through two main step; the first step was based on SOM to analyzes, evaluates and optimizes reusability for Component Based Software Engineering using CK metrics values, and the second step was using supervised Back propagation Neural Network (BPNN) and fuzzy inference rules applied on CK metric values to categorize the data into Very Low, Low, Medium, High and Very High reusability classes.The original form of SOM that has been proposed by [46] was used in this paper rather any other update forms of SOM algorithm, this is because it is computationally the simplest and the lightest, and it is produce in practise useful results of mapping a high dimensional input vector [47].

4. The Proposed Methodology

The objective of this paper is to capture, analyze, and model the effects of software components metrics values on software reusability level. This work uses SOM to define the level of software reusability. The proposed methodology consists of the following phases that are shown in **Figure 1**.

4.1. Dataset Collection and Preprocessing

The required dataset for this work was collected from COMETS datasets that is available on the web [48]. We selected Chidamber and Kemerer (CK) metric suite (CBO, DIT, LCOM, NOC, WMC, and RFC) from seventeen metrics in COMETS datasets. The metrics values were collected from the last released versions of selected systems. Three different systems with their CK metrics were selected to be experimented using SOM. CK metrics values were used as attributes for each input vector to SOM. The selected systems are Eclipse JDT, Eclipse PDE, and Hibernate. The number of classes in each system is 1061, 1454, 1173 respectively. The statistical description of each selected system is shown in **Tables 1-3** respectively.

Figure 1. The proposed methodology.

Table 1. Statistical description of Eclipse JDT dataset.

	CBO	DIT	LCOM	NOC	RFC	WMC
Minimum	0	1	0	0	0	0
Maximum	222	8	80,982	26	661	2074
Standard deviation	21.36	1.72	2762.65	1.99	62.81	159.24
Mean	15.87	2.62	235.36	0.65	41.57	61.60
Median	9	2	5	0	22	21

Table 2. Statistical description of Eclipse PDE dataset.

	CBO	DIT	LCOM	NOC	RFC	WMC
Minimum	0	1	0	0	0	0
Maximum	129	9	1377	108	334	229
Standard deviation	17.38	1.54	80.67	3.82	38.68	28.77
Mean	17.14	2.22	19.79	0.66	34.16	20.57
Median	12	2	1	0	21	11

Table 3. Statistical description of hibernate dataset.

	CBO	DIT	LCOM	NOC	RFC	WMC
Minimum	0	0	0	0	0	0
Maximum	186	7	13,069	28	682	594
Standard deviation	14.93	1.18	747.92	1.80	48.99	38.64
Mean	11.78	1.85	115.40	0.50	26.70	18.19
Median	8	1	2	0	13	8

The statistical descriptions show that there is a low median and mean for DIT and NOC metrics. That means inheritance is not used much in the systems. This result was also found in [15] [19] [49] [50].

Metrics threshold used as quantitative method to define the good qualitative of software quality and useful to identify the high risk classes [51]. The threshold value forms good general view of software to help developer in review software classes [52]. Exceeding the upper bound of threshold value can be considered a problem and a sign of poor design of the software class, so less reusable. Threshold values provide a simple analysis method to define the risky software design. We assume that threshold represent the upper bound as in [53]. The adopted metrics threshold used in this paper is shown in **Table 4**.

In data preprocessing phase, data should be prepared to be used by any clustering techniques, in order to develop a unified form for all data instances [54]. The used dataset has many records (OO classes) with the value −1 in all metrics; which means that class did not existed in a given time. All records with the value of −1 were ignored and deleted from the dataset; the number of classes before and after cleansing in the selected systems is shown in **Table 5**.

Then Min-Max normalization method has been adopted in this paper to normalize the selected dataset, using Equation (1) [55].

$$v_i' = \frac{v_i - \min_A}{\max_A - \min_A}\left(\text{new\_}\max_A - \text{new\_}\min_A\right) + \text{new\_}\min_A \quad (1)$$

where: $\max_A, \min_A$ are the minimum and maximum values of an attribute A. Min-max normalization maps a value v_i of A to a value v_i' in the range $\left[\text{new\_}\max_A, \text{new\_}\min_A\right]$.

4.2. Reusability Classification Using SOM

SOM is based on Competitive Learning rule. **Figure 2** shows a simple competitive learning network. We can notice that the network is fully connected input neurons (i) to output neurons (o) with weight (w_{io}) [35]:

The algorithm of Kohonen's SOM can be summarizing as shown in **Figure 3**.

4.3. Clustering Validation

Silhouette method is used to measure the clustering validity. Silhouette uses combination of two criteria; separation and cohesion. Separation measures how well the clusters are separated from each other based on distance between clusters centroids. Cohesion measures how well the clusters are cohesion and vectors are closely related to each other in one cluster [50].

For a data set *D*, of n objects, suppose *D* is partitioned into k clusters, $C_1, \cdots, C_k$. Silhouette can be computed using the following steps [55]:

- For each object $o \in D$, find the average distance between o and all other objects in the cluster to which o belongs. Called its value $a(o)$.
- For each object $o \in D$, find the minimum average distance from o to all clusters to which o does not belong. Called its value $b(o)$.
- Calculate silhouette coefficient $s(o)$ using Equation 2 [50]:

$$s(o) = \frac{b(o) - a(o)}{\max\{a(o), b(o)\}}$$

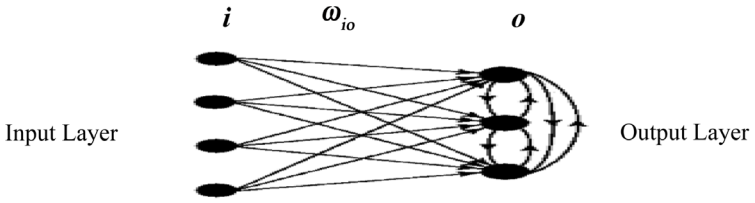

Figure 2. Simple competitive learning network.

1- Initialize the weights to small random values and the neighbourhood size large enough to cover half the nodes.
2- Select an input pattern x randomly from the training set and present it to the network.
3- Find the best matching or "winning" node ***k*** whose weight vector w_k is closest to the current input vector x using the vector distance (minimum-distance Euclidean criterion):

$$\|x - w_k\| = \min_i \|x - w_i\|$$

where ||. || represents the Euclidean distance.
4- Update the weights of nodes in the neighbourhood of ***k*** using the Kohonen's learning rule:

$$w_i^{new} = w_i^{old} + \alpha h_{i,k}\left(x - w_i\right) \text{ if } i \text{ is in } N_k$$

$$w_i^{new} = w_i^{old} \text{ if } i \text{ is not in } N_k$$

where:
α The learning rate parameter between 0 and 1.
$h_{i,k}$ Neighbourhood function centred on the winning neuron ***k***
5- Decrease the learning rate slightly
6- Repeat Steps 1-5 with a number of cycles and then decrease the size of the neighbourhood. Repeat until weights are stabilized.

Figure 3. SOM algorithm.

Table 4. Threshold values.

Metric	Proposed Threshold	Reference
CBO	13	[19]
WMC	20	[19]
LCOM	20	[49]
DIT	2	[49]
RFC	44	[19]
NOC	2	[49]

Table 5. Data cleansing.

System Name	Version Number	Number of original Classes	Number of Cleaned Classes
Eclipse JDT	183	1368	1061
Eclipse PDE	191	3083	1454
Hibernate	98	1216	1173

5. Experimental Results and Analysis

The experiments were applied for each system separately, using the same experiment settings, to standardize the evaluation for each. SOM Grid size and number of epochs were updated in all experiments. Learning rate values in all experiments were as follow:

Ordering phase learning rate = 0.9
Ordering phase steps = 1000

Tuning phase learning rate = 0.02

Tuning phase neighborhood distance = 1

While the number of epochs was increasing, the silhouette average values were increasing also. The epoch's number is then adjusted at 2000, because silhouette became almost stable in all selected systems at this point. Many experiments were obtained on different SOM grid sizes. The experiment with the highest silhouette value is selected. Then the metrics values were analyzed in each cluster to find their relationships with the class reusability level.

Eclipse JDT system classes were normalized, then they were imported to SOM network to be clustered. Classes were experimented many times in order to find the highest silhouette average value. Experiment number 8 in **Table 6** achieves the highest silhouette average value, so this experiment will be analyzed. The analysis was based mainly on CK metrics average values in each cluster, and also based on the percentage of classes that are exceeding CK metrics threshold values.

After analyzing each cluster in the Eclipse JDT System, we found that DIT and NOC metrics are dominating the clustering process, which was also found in [36]. There is no obvious trend found between classes in one cluster. Metrics values also were analyzed in each cluster and an obvious common relationship in the same cluster's classes couldn't be found. Based on previous observations, the distribution of NOC and DIT metrics values was analyzed in Eclipse JDT System, it was found that 76.2% of the NOC metric classes have 0 values, and 76.3% of classes have the values 1, 2, and 3 in DIT metric. In addition the experimental results show that eliminating NOC and DIT metrics from clustering process may enhance the results. Then the experiments were reapplied after eliminating NOC and DIT metrics from input.

In **Table 6**, the experimental attempts of Eclipse JDT system are shown. The SOM grid size was changed many times until find the best one, which was for this dataset is [2 × 2] grid size, because it has the highest average of silhouette. Also the Table shows that when the grid size is changed, the average of silhouette is changed also.

The silhouette plot of experiment 8 is shown in **Figure 4**. The X-axis illustrates the silhouette values. The value of silhouette coefficient is ranging from −1 to 1. Negative value of silhouette is not preferable because it means that object is related to other cluster more than the cluster that is belonging to. Values approaching to 1 are the desirable value; that means the average distance of object to points in the same cluster is greater than the minimum average distance of object to all other clusters. To measure the goodness of overall clustering process, the average silhouette can be used [55]. The Y-axis defines the number of clusters in the grid.

The classes in each cluster can be classified into one of three main categories. These categories are summarizing the relationship between CK metrics values and reusability quality factor. The categories are:

- Category One: A High Reusable cluster contains classes that are not exceeding threshold values and have lowest values of CBO, LCOM, RFC, and WMC metrics.

Table 6. Eclipse JDT experiments attempts.

Experiment No.	Grid Size	Average of Silhouette
1	[10 × 10]	0.4937
2	[8 × 8]	0.4607
3	[7 × 7]	0.4586
4	[6 × 6]	0.4437
5	[5 × 5]	0.4971
6	[4 × 4]	0.5426
7	[3 × 3]	0.6351
8	**[2 × 2]**	**0.7083**
9	[5 × 4]	0.5099
10	[4 × 3]	0.6303
11	[3 × 2]	0.6185

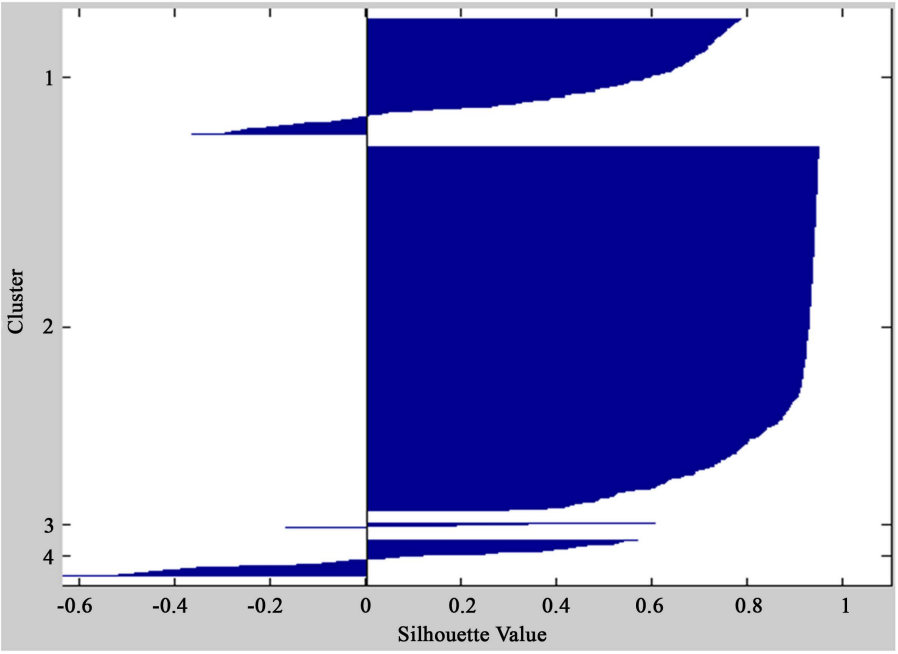

Figure 4. Silhouette result of experiment 8 of eclipse JDT system after eliminating NOC and DIT metrics.

- Category Two: A Medium Reusable cluster contains classes that are around threshold values and have the medium values of CBO, LCOM, RFC, and WMC metrics.
- Category Three: A Low Reusable cluster contains classes that are exceeding threshold values and have highest values of CBO, LCOM, RFC, and WMC metrics.

Based on that, the analysis of each cluster vectors in experiment 8 is shown in **Table 7**, where Avg. column is the metric average in each cluster. If the Avg. column of one metric in specific cluster is high, that means the classes that included in this cluster had high values of that metric, so they are become less reusable and vice versa. Ex.% column is the percentage of classes that are exceeding metric threshold values. When the percentage is high, it means that there are many classes in this cluster are exceeding the threshold value, so it become also low reusable and vice versa.

As shown in **Table 7** cluster 3 and 4 are Low Reusable clusters, because almost all of the classes in both clusters exceed the threshold values and their averages are very high. Cluster 2 contains the most properly classes to be reusable; it has the minimum values of all metrics, so it is High Reusable cluster. Cluster 1 is Medium Reusable cluster, because the percentage of the classes that exceeding the threshold and averages is less than clusters 3 and 4, and greater than cluster 2.

Eclipse PDE and Hibernate systems were experimented many times also in order to find the highest silhouettes average values as done in the Eclipse JDT system. The analysis of the results was based mainly on CK metrics average values for each clusters, and also based on the percentage of classes that are exceeding CK metrics threshold values. The same observations were also found in Eclipse PDE and Hibernate systems (no obvious trend between classes and no common relationship in the same clusters classes). In Eclipse PDE system, NOC metric values had 0 in 82.5% of the classes and 81.7% of classes have the values 1, 2, and 3 in DIT metric. In Hibernate system, NOC metric values have 0 in 83.1% of the classes and 77.6% of classes had the values of 1 and 2 in DIT metric. Therefore, the distribution of NOC and DIT were poor, similar to the results found in [15] [19] [49] [50]. Then NOC and DIT metrics were also discarded from input in the experiments as done in Eclipse JDT.

Table 8 and **Table 9** show the experimental attempts of Eclipse PDE and Hibernate systems respectively. The Tables show that the best grid size for both systems is also [2 × 2] grid size as Eclipse JDT system, because it has the highest silhouette average.

The silhouette plots of experiment 8 for both systems (Eclipse PDE and Hibernate systems) are shown in **Figure 5** and **Figure 6** respectively.

The results analysis of experiment 8 for Eclipse PDE system, are shown in **Table 10**. It illustrate that cluster 3

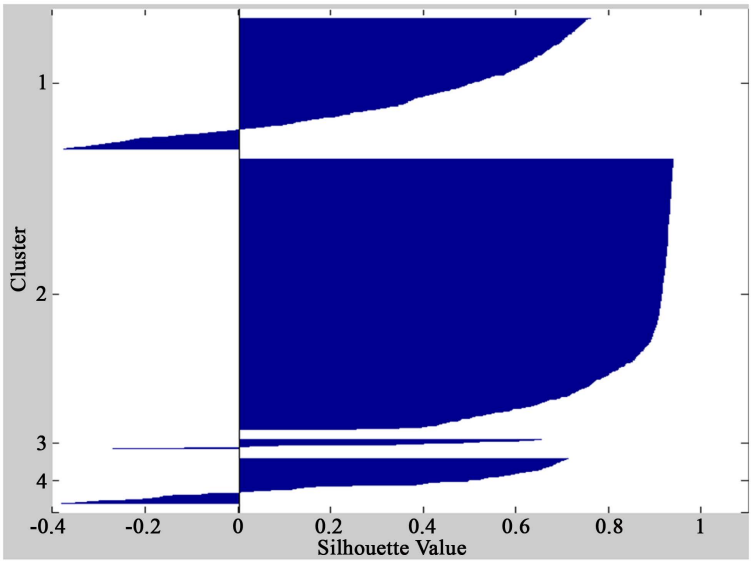

Figure 5. Silhouette result of experiment 8 of eclipse PDE system after eliminating NOC and DIT metrics.

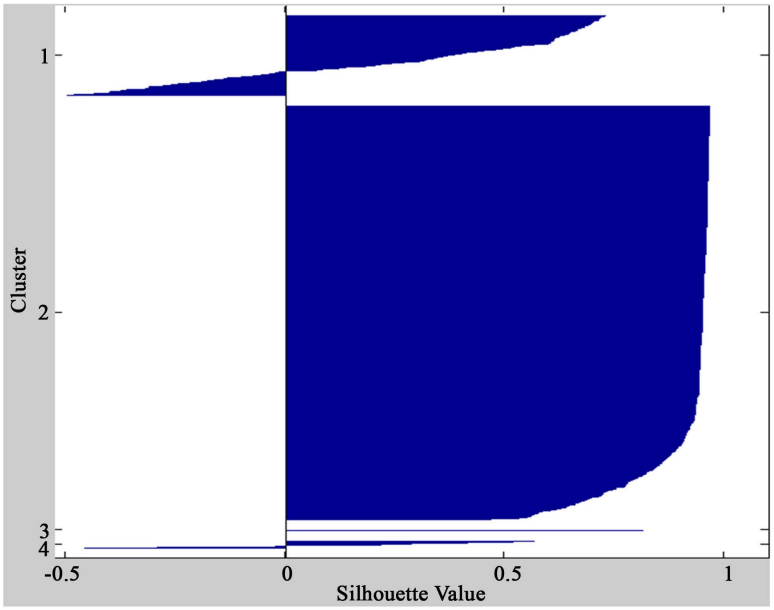

Figure 6. Silhouette result of experiment 8 of hibernate system after eliminating NOC and DIT metrics.

Table 7. Results analysis of experiment 8 of eclipse JDT system.

	CBO		LCOM		RFC		WMC		NO. OF CLASSES
	Avg.	Ex.%	Avg.	Ex.%	Avg.	Ex.%	Avg.	Ex.%	
Cluster 1	25.2	97.3%	100.4	48%	61.4	80.4%	79.9	96.4%	**225**
Cluster 2	6.7	8.8%	16.6	15%	16.4	3.4%	16.6	29.5%	**706**
Cluster 3	133.3	100%	9050	100%	4	100%	1092.6	100%	**13**
Cluster 4	54.7	98.8%	1210.3	77.8%	155.3	100%	258.9	100%	**72**
Threshold	**13**		**20**		**44**		**20**		

Table 8. Eclipse PDE experiments attempts.

Experiment No.	Grid Size	Average of Silhouette
1	[10 × 10]	0.4060
2	[8 × 8]	0.3840
3	[7 × 7]	0.3878
4	[6 × 6]	0.4182
5	[5 × 5]	0.4636
6	[4 × 4]	0.4839
7	[3 × 3]	0.5733
8	**[2 × 2]**	**0.6565**
9	[5 × 4]	0.4483
10	[4 × 3]	0.5688
11	[3 × 2]	0.6029

Table 9. Hibernate experiments attempts.

Experiment No.	Grid Size	Average of Silhouette
1	[10 × 10]	0.4666
2	[8 × 8]	0.4468
3	[7 × 7]	0.4250
4	[6 × 6]	0.4388
5	[5 × 5]	0.5260
6	[4 × 4]	0.5296
7	[3 × 3]	0.6131
8	**[2 × 2]**	**0.8003**
9	[5 × 4]	0.5200
10	[4 × 3]	0.5754
11	[3 × 2]	0.7090

Table 10. Results analysis of experiment 8 of eclipse PDE system.

	CBO		LCOM		RFC		WMC		NO. OF CLASSES
	Avg.	Ex.%	Avg.	Ex.%	Avg.	Ex.%	Avg.	Ex.%	
Cluster 1	22.5	91.6%	25.16	33.1%	44.5	44.4%	25.2	56.4%	**417**
Cluster 2	7	11%	5.5	7.1%	12.2	0%	7.56	3.9%	**861**
Cluster 3	85.5	100%	160.9	58%	198.3	100%	155.2	100%	**31**
Cluster 4	46.7	100%	58.7	44.8%	99.4	100%	55.5	97.2%	**145**
Threshold	**13**		**20**		**44**		**20**		

and 4 are Low Reusable clusters. Cluster 2 contains the most properly classes to be High Reusable. Cluster 1 is Medium Reusable cluster.

The analysis of cluster's vectors of experiment 8 for Hibernate system is shown in **Table 11**. The results show that Cluster 3 has only 2 vectors with very high values for all metrics. Cluster 3 and 4 are Low Reusable clusters. Cluster 2 is a High Reusable cluster, because it has the minimum values of all metrics. Cluster 1 is a Medium

Table 11. Results analysis of experiment 8 of hibernate system.

	CBO		LCOM		RFC		WMC		NO. OF CLASSES
	Avg.	Ex.%	Avg.	Ex.%	Avg.	Ex.%	Avg.	Ex.%	
Cluster 1	27.2	98.2%	252.8	59%	70.3	78.2%	44.6	81.4	**188**
Cluster 2	7.1	13.4%	11.3	12%	12.4	0.5%	8.3	6.6%	**964**
Cluster 3	126.5	100%	12954	100%	583	100%	466	100%	**2**
Cluster 4	83.9	100%	2824.8	88.9%	273.7	100%	220.6	100%	**18**
Threshold	**13**		**20**		**44**		**20**		

Reusable cluster, because the percentage of the classes that exceeding the threshold and its averages is less than clusters 3 and 4, and greater than cluster 2.

LCOM metric can be used to evaluate the software reusability using SOM, based on used threshold even it has a poor distribution. Unlike statistical methods which discard LCOM metric from finding a clear threshold, because of its poor distribution.

6. Conclusion and Future Work

This research is based on using the Kohonen's Self-Organizing Map (SOM) to cluster software metrics (CK metrics suite). The clustering of CK metrics was based on metrics threshold values that proposed in literature. We showed that SOM can be applied to clusters software metrics to visualize the relationship between software metrics and its reusability level (High Reusable, Medium Reusable and Low Reusable). SOM was used in this research according to its powerful ability in clustering data vectors and its property of spatial autocorrelation. This helps in discovering software metrics patterns and its relationship with reusability category. The clustering validity was based on the highest silhouette average value, after we applied many grids sizes and different number of epochs. Initially, we applied SOM on all CK metrics suite, in order to cluster classes to their suitable reusability category, but we found that NOC and DIT metrics dominated the clustering results because of their poor distribution, so it was helpless clustering. The solution was to eliminate NOC and DIT metrics from clustering process. The experimental results show that the clustering becomes more homogenous and meaningful.

In future, after more investigations in software metrics related to reusability quality factor, SOM can be tested again with different metrics to have better results. This will help the software designers to predict class component reuse level. Furthermore, we could use same datasets of three systems with eliminating three of CK metrics (LCOM, DIT, and NOC) because of its poor distribution; hence the datasets become 3-D, to better visualization.

We can also define more than three categories for reusability, for example, we can categorize classes into five categories: Not Reusable, Low Reusable, Medium Reusable, Reusable and High Reusable.

In our work, to gain better results we discarded NOC and DIT metrics from experiments because of their poor distribution. Another solution that can be applied is to change the initial parameters such as inputting initial weights and learning rate values, then reapplying experiments.

References

[1] Sommerville, I. (2011) Software Engineering. 9th Edition, Addison-Wesley, New York.

[2] Goel, B.M. and Bhatia, P.K. (2013) Analysis of Reusability of Object-Oriented Systems Using Object-Oriented Metrics. *ACM SIGSOFT Software Engineering Notes*, **38**, 1-5. http://dx.doi.org/10.1145/2492248.2492264

[3] Maggo, S. and Gupta, C. (2014) A Machine Learning Based Efficient Software Reusability Prediction Model for Java Based Object Oriented Software. *International Journal of Information Technology and Computer Science* (*IJITCS*), **6**, 1. http://dx.doi.org/10.5815/ijitcs.2014.02.01

[4] Rotaru, O.P. and Dobre, M. (2005) Reusability Metrics for Software Components. *Proceedings of the* 3*rd ACS/IEEE International Conference on Computer Systems and Applications*, Cairo, 2005, 24-29. http://dx.doi.org/10.1109/aiccsa.2005.1387023

[5] Singh, G. (2013) Metrics for Measuring the Quality of Object-Oriented Software. *ACM SIGSOFT Software Engineering Notes*, **38**, 1-5. http://dx.doi.org/10.1145/2507288.2507311

[6] Chidamber, S.R. and Kemerer, C.F. (1994) A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, **20**, 476-493. http://dx.doi.org/10.1109/32.295895

[7] Kumar, V., Kumar, R. and Sharma, A. (2013) Applying Neuro-Fuzzy Approach to Build the Reusability Assessment Framework across Software Component Releases—An Empirical Evaluation. *International Journal of Computer Applications*, **70**, 41-47. http://dx.doi.org/10.5120/12041-8047

[8] Caldiera, G. and Basili, V.R. (1991) Identifying and Qualifying Reusable Software Components. *IEEE Software*, **24**, 61-70. http://dx.doi.org/10.1109/2.67210

[9] Sametinger, J. (1997) Software Engineering with Reusable Components. Springer Verlag, New York. http://dx.doi.org/10.1007/978-3-662-03345-6

[10] Weinreich, R. and Sametinger, J. (2001) Component Models and Component Services: Concepts and Principles, Component-Based Software Engineering: Putting Pieces Together. In: Heineman, G.T. and Councill, W.T., Eds., *Component-Based Software Engineering*: *Putting Pieces Together*, Addison-Wesley, New York, 33-48.

[11] He, J., Chen, R. and Gu, W. (2009) A New Method for Component Reuse. *The* 2*nd IEEE International Conference on Computer Science and Information Technology* (*ICCSIT*), Beijing, August 2009, 304-307.

[12] Smaragdakis, Y. and Batory, D. (1998) Implementing Reusable Object-Oriented Components. *Proceedings of the* 5*th International Conference on Software Reuse*, Victoria, 2-5 June 1998, 36-45. http://dx.doi.org/10.1109/icsr.1998.685728

[13] Rosenberg, L.H. and Hyatt, L.E. (1997) Software Quality Metrics for Object-Oriented Environments. *Crosstalk Journal*, **10**, 1-16.

[14] Antony, P.J. (2013) Predicting Reliability of Software Using Thresholds of CK Metrics. *International Journal of Advanced Networking & Applications*, **4**, 6.

[15] Basili, V.R., Briand, L.C. and Melo, W.L. (1996) A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering*, **22**, 751-761. http://dx.doi.org/10.1109/32.544352

[16] Succi, G., Pedrycz, W., Stefanovic, M. and Miller, J. (2003) Practical Assessment of the Models for Identification of Defect-Prone Classes in Object-Oriented Commercial Systems Using Design Metrics. *Journal of Systems and Software*, **65**, 1-12. http://dx.doi.org/10.1016/S0164-1212(02)00024-9

[17] Hudiab, A., Al-Zaghoul, F., Saadeh, M. and Saadeh, H. (2015) ADTEM—Architecture Design Testability Evaluation Model to Assess Software Architecture Based on Testability Metrics. *Journal of Software Engineering and Applications*, **8**, 201-210. http://dx.doi.org/10.4236/jsea.2015.84021

[18] Gill, N.S. and Sikka, S. (2011) Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD. *International Journal on Computer Science and Engineering*, **3**, 2300-2309.

[19] Shatnawi, R. (2010) A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems. *IEEE Transactions on Software Engineering*, **36**, 216-225. http://dx.doi.org/10.1109/TSE.2010.9

[20] Shatnawi, R., Li, W., Swain, J. and Newman, T. (2010) Finding Software Metrics Threshold Values Using ROC Curves. *Journal of Software Maintenance and Evolution*: *Research and Practice*, **22**, 1-16. http://dx.doi.org/10.1002/smr.404

[21] Guo, X.L., Wang, H.Y. and Glass, D.H. (2012) A Growing Bayesian Self-Organizing Map for Data Clustering. *Proceedings of the International Conference on Machine Learning and Cybernetics* (*ICMLC*), **2**, 708-713. http://dx.doi.org/10.1109/icmlc.2012.6359011

[22] Cho, E.S., Kim, M.S. and Kim, S.D. (2001) Component Metrics to Measure Component Quality. *Proceedings of the* 8*th Asia Pacific Software Engineering Conference* (*APSEC*), Macau, 4-7 December 2001, 419-426.

[23] Washizaki, H., Yamamoto, H. and Fukazawa, Y. (2003) A Metrics Suite for Measuring Reusability of Software Components. *Proceedings of* 9*th International Symposium on Software Metrics*, Sydney, 3-5 September 2003, 211-223. http://dx.doi.org/10.1109/metric.2003.1232469

[24] Sindre, G., Conradi, R. and Karlsson, E.A. (1995) The REBOOT Approach to Software Reuse. *Journal of Systems and Software*, **30**, 201-212. http://dx.doi.org/10.1016/0164-1212(94)00134-9

[25] Sharma, A., Grover, P.S. and Kumar, R. (2009) Reusability Assessment for Software Components. *ACM SIGSOFT Software Engineering Notes*, **34**, 1-6. http://dx.doi.org/10.1145/1507195.1507215

[26] Gandhi, P. and Bhatia, P.K. (2011) Estimation of Generic Reusability for Object-Oriented Software an Empirical Approach. *ACM SIGSOFT Software Engineering Notes*, **36**, 1-4. http://dx.doi.org/10.1145/1968587.1968606

[27] Kulkarni, U.L., Kalshetty, Y.R. and Arde, V.G. (2010) Validation of Ck Metrics for Object Oriented Design Measurement. *Proceedings of the* 3*rd International Conference on Emerging Trends in Engineering and Technology* (*ICETET*), Goa, 19-21 November 2010, 646-651. http://dx.doi.org/10.1109/ICETET.2010.159

[28] Tang, M.H., Kao, M.H. and Chen, M.H. (1999) An Empirical Study on Object Oriented Metrics. *Proceedings of the 6th International Software Metrics Symposium*, Boca Raton, 4-6 November 1999, 242-249.

[29] Hsu, C.C. (2006) Generalizing Self-Organizing Map for Categorical Data. *IEEE Transactions on Neural Networks*, **17**, 294-304. http://dx.doi.org/10.1109/TNN.2005.863415

[30] Yin, H. (2008) The Self-Organizing Maps: Background, Theories, Extensions and Applications. In: Fulcher, J. and Jain, L.C., Eds., *Computational Intelligence*: *A Compendium*, Springer, Berlin, 715-762. http://dx.doi.org/10.1007/978-3-540-78293-3_17

[31] Van Hulle, M.M. (2012) Self-Organizing Maps. In: Rozenberg, G., Bäck, T. and Kok, J.N., Eds., *Handbook of Natural Computing*, Springer, Berlin, 585-622. http://dx.doi.org/10.1007/978-3-540-92910-9_19

[32] Kruskal, J.B. and Wish, M. (1978) Multidimensional Scaling. Sage University Paper Series on Quantitative Applications in the Social Sciences, No. 07-011, Sage Publications, Newbury Park. http://dx.doi.org/10.4135/9781412985130

[33] Cottrell, M., Fort, J.C. and Pagés, G. (1997) Theoretical Aspects of the SOM Algorithm. *Proceedings of the Workshop on Self-Organizing Maps* (*WSOM* 97), Helsinki University of Technology, Neural Networks Research Centre, Espoo, 4-6 June 1997, 246-267.

[34] Maimon, O. and Rokach, L. (2007) Soft Computing for Knowledge Discovery and Data Mining. Springer Science & Business Media, Berlin

[35] Haykin, S. and Network, N. (2004) Neural Networks: A Comprehensive Foundation. 2nd Edition, Pearson Education, Indian, Bangladesh.

[36] Pedrycz, W., Succi, G., Musílek, P. and Bai, X. (2001) Using Self-Organizing Maps to Analyze Object-Oriented Software Measures. *Journal of Systems and Software*, **59**, 65-82. http://dx.doi.org/10.1016/S0164-1212(01)00049-8

[37] Chang, C.H., Xu, P., Xiao, R. and Srikanthan, T. (2005) New Adaptive Color Quantization Method Based on Self-Organizing Maps. *IEEE Transactions on Neural Networks*, **16**, 237-249. http://dx.doi.org/10.1109/TNN.2004.836543

[38] Dreyfus, G. (2005) Neural Networks: Methodology and Applications. Springer, Berlin.

[39] Mäntysalo, J., Torkkola, K. and Kohonen, T. (1994) Mapping Content Dependent Acoustic Information into Context Independent Form by LVQ. *Speech Communication*, **14**, 119-130. http://dx.doi.org/10.1016/0167-6393(94)90003-5

[40] Lapidot, I., Guterman, H. and Cohen, A. (2002) Unsupervised Speaker Recognition Based on Competition between Self-Organizing Maps. *IEEE Transactions on Neural Networks*, **13**, 877-887. http://dx.doi.org/10.1109/TNN.2002.1021888

[41] Simula, O. and Kangas, J. (1995) Process Monitoring and Visualization Using Self-Organizing Maps. *Integrated Computer-Aided Engineering*, **6**, 3-14.

[42] Merkl, D. (1993) Structuring Software for Reuse—The Case of Self-Organizing Maps. *Proceedings of the International Joint Conference on Neural Networks* (*IJCNN*), **3**, 2468-2471. http://dx.doi.org/10.1109/ijcnn.1993.714224

[43] Veras, R.C., Meira, S.R., Oliveira, A.L. and Melo, B.J. (2007) Comparative Study of Clustering Techniques for the Organization of Software Repositories. 19*th IEEE International Conference on Tools with Artificial Intelligence* (*ICTAI* 2007), **1**, 210-214. http://dx.doi.org/10.1109/ICTAI.2007.162

[44] Lin, Y. and Ye, H. (2009) Input Data Representation for Self-Organizing Map in Software Classification. *Proceedings of the* 2*nd International Symposium on Knowledge Acquisition and Modeling* (*KAM*), **2**, 350-353. http://dx.doi.org/10.1109/kam.2009.151

[45] Acharya, S. and Sadananda, R. (1997) Promoting Software Reuse Using Self-Organizing Maps. *Neural Processing Letters*, **5**, 219-226. http://dx.doi.org/10.1023/A:1009603129257

[46] Kohonen, T. (1982) Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, **43**, 59-69. http://dx.doi.org/10.1007/BF00337288

[47] Kohonen, T. (2013) Essentials of the Self-Organizing Map. *Neural Networks*, **37**, 52-65. http://dx.doi.org/10.1016/j.neunet.2012.09.018

[48] Applied Software Engineering Research Group (ASERG) (2013) http://aserg.labsoft.dcc.ufmg.br/comets/

[49] Zhou, Y. and Leung, H. (2006) Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. *IEEE Transactions on Software Engineering*, **32**, 771-789. http://dx.doi.org/10.1109/TSE.2006.102

[50] Tan, P.N., Steinbach, M. and Kumar, V. (2006) Introduction to Data Mining. Pearson: Addison Wesley, Boston.

[51] El Emam, K., Goel, N. and Rai, S. (2000) Thresholds for Object-Oriented Measures. 11*th International Symposium on Software Reliability Engineering* (ISSRE 2000), San Jose, 8-11 October 2000, 24-38.

[52] Singh, S. and Kahlon, K.S. (2014) Object Oriented Software Metrics Threshold Values at Quantitative Acceptable Risk Level. *CSI Transactions on ICT*, **2**, 191-205. http://dx.doi.org/10.1007/s40012-014-0057-1

[53] Herbold, S., Grabowski, J. and Waack, S. (2011) Calculation and Optimization of Thresholds for Sets of Software Metrics. *Empirical Software Engineering*, **16**, 812-841. http://dx.doi.org/10.1007/s10664-011-9162-z

[54] Mirkin, B. (2012) Clustering: A Data Recovery Approach. CRC Press, Boca Raton. http://dx.doi.org/10.1201/b13101

[55] Jiawei, H. and Kamber, M. (2012) Data Mining: Concepts and Techniques. 3th Edition, Elsevier, San Francisco.