Journal of Computer Science

Original Research Paper

A VF-IMF Cohesion Metric for Object-Oriented Classes

Joy Christy Antony Sami and Umamakeswari Arumugam

Department of Computer Science Engineering, School of Computing,
SASTRA Deemed to be University, Thirumalaisamudram, Thanjavur, Tamil Nadu, India, 613 401

Article history
Received: 10-07-2019
Revised: 21-09-2019
Accepted: 09-04-2020

Corresponding Author:
Umamakeswari Arumugam
Department of Computer Science Engineering, School of Computing, SASTRA Deemed to be University, Thirumalaisamudram, Thanjavur, Tamil Nadu, India, 613 401
Email: umamakeswari.arumugam@gmail.com

Abstract: Cohesion in Object Oriented (OO) modules impact reusability, efficiency and complexity of software. OO Programmers are mandated to create software with high cohesion. The testing phase in Software Development Life Cycle (SDLC) is not only concerned about creating error free software but also assess quality of code through software metrics. The metric'Lack of Cohesion in Methods (LCOM)' is one of the significant OO metric for measuring level of cohesion in software modules. LCOM and its improvised versions of cohesion metrics output degree of cohesion in software modules rather than providing solutions to reconstruct the poorly cohesive modules. Further, the traditional cohesion metrics do not differentiate the possible levels such as high, medium and low cohesions. Thus, in this paper a novel, Variable Frequency – Inverse Method Frequency (VF-IMF) based machine learning metric is proposed to assess the level of cohesion in modules and also to group module methods to instill high cohesion. The proposed metric is experimented over three sample modules represents each level of cohesion. The experimental results show that the proposed metric clearly differentiates the three levels of cohesion and offers a compromised solution for building high cohesive modules than traditional LCOM metrics. The metric is also validated against Weyuker's properties and is proven to be a valid metric as it satisfies all the 9 properties.

Keywords: TF, IDF, LCOM, MALCOM, TCC, LC

Introduction

Software metric is a standard measure of a degree to which software system or process possesses some property. In other words, metrics estimate the quality of the software in the testing phase of software development life cycle. The traditional classification of software metrics can be divided into procedure and object oriented programming. Object oriented programming is now a widely adopted approach in software engineering because software built using this technique is usually easier to maintain. OO metrics are a division of software metrics that evaluates the features of OO programming such as modularity, reusability, inheritance, abstraction and encapsulation and so on. In modularity, cohesion is an important feature for exhibiting the intra module communications. Good modular designs maximize cohesion and promote encapsulation. High cohesion classes do not share their attributes with other classes and can easily be reused. Thus, high cohesion promotes reusability by reducing complexity. Software testing researchers have come up with many object oriented metrics such as LCOM, Tight Class Cohesion (TCC) and Loose Class Cohesion (LCC) to evaluate software module with respect to cohesion. But, all those metrics tend to highlight only the existence/non-existence of cohesion in modules, without specifying intermediate levels such as Low, Medium and High. Modified Approach on LCOM (MALCOM) takes a different turn by addressing the issue (Ganesh *et al*., 2015). But, the usefulness of software metrics should go beyond the expectations on what action the programmer ought to take with the obtained metric value or when the class has poor cohesion. To answer these questions, machine learning techniques come into picture. Machine learning techniques consist series of statistical algorithms that take input data to analyze the hidden pattern or useful knowledge.

The paper combines the traditional concepts of software metrics with machine learning algorithms to provide a concrete solution for constructing highly cohesive modules in object oriented software. Unsupervised algorithms in machine learning techniques describe the nature or characteristics of data. Clustering is one of the unsupervised learning algorithms that groups similar objects as clusters. Clustering algorithms

are widely used in image analysis, document clustering, medical diagnosis, customer segmentation and pattern recognition and so on. The application of clustering algorithms in software metrics is yet another novel approach of the paper. Initially, the methods in a module are transformed to a structure that gets processed by the clustering algorithm is same as how tokens are parsed in document clustering using Term Frequency-Inverse Document Frequency (TF-IDF). In this work, we use K-Means clustering algorithms for generating clusters as it is straight forward, most popular and widely used algorithm. The methods that share module variables are clustered with the assumption, if all methods grouped in one cluster shows high cohesion, grouped in different clusters equal to number of methods in module as low cohesion. If the clusters formed using the algorithm neither puts all methods in one cluster nor forms clusters as equal to the number of methods in the module tends to fall in the medium cohesion.

The remaining section of the paper organized as follows: Section II reviews the existing literature in software metrics, section III describes the methodology, section IV explicates the experimentation and result discussion and finally section V concludes the findings and future works of the paper.

Review of Literature

Over a detailed survey on the Cohesion, new metrics were introduced ranging from syntactic to semantic analysis (Gu *et al*., 2017; Alzahrani *et al*., 2019; Cinnéide *et al*., 2017) and software metrics tools were developed e.g. VizzAnalyzer, Analyst4j, Understand and Ckjm. A study on comparing these tools was carried out by (Purao and Vaishnavi, 2003). However, implementation of cohesion metrics from these tools is not enough for the investigation of cohesion due to several relationships being overlooked. In this paper, the authors tried to look at the relationships that make up cohesion by examining existing frameworks and selecting metrics that correlate to the respective relationships. The selected metrics are then formalized to show how these metrics are implemented. Another detailed survey On Chidamber and Kemerer (Santos *et al*., 2017) introduced the primary cohesion metric for object-oriented systems. The authors defined LCOM metric as measure of cohesion which is the difference of null intersections with non-null intersections of methods in module and concluded that higher cohesion results in higher quality. The authors later revised their definition of LCOM to incorporate Loose and Tight class cohesions (Wu *et al*., 2015; Kerdoudi *et al*., 2016), where LCC is the fraction of number of direct connections and number of indirect connections with maximum number of possible connections and TCC is the fraction of number of direct connections with maximum number of possible connections. Goel and Gupta (2017) propose a dynamic approach to measure cohesion of software systems. The paper concluded that the conventionally used measurement systems were statically headed and provided the unfinished dynamic behaviour of the system. This is due to static systems that provide measures only for syntactic program analysis and so they have tiny ability. They insisted that the dynamic measures should always be utilized in combination with their static counterparts. Vincent proposed MALCOM metric by introducing a third variable ‘R’ in addition to ‘P’ and ‘Q’ in LCOM. The variable ‘P’ in MALCOM is set to 1 if the total number of resultant variables equal to the fraction of total number of variables with total number of methods. ‘Q’ variable in MALCOM is set to -1 if the total number of variables equal to total number of variables declared in the module and Else the variable ‘R’ is set to 0. The authors defined ‘P’ as high cohesion, ‘Q’ as poor cohesion and ‘R’ as partial cohesion. Kansal *et al*. (2019) formulated Improved Sensitive Class Cohesion Metric (ISCOM). The authors have reduced the intensity of pair of methods as the fraction of shared variables between method-pairs by the average cardinalities of the two sets of variables accessed by the methods. Rathee *et al*. (2018) presented FUPClust based on Frequent Usage Pattern based cohesion metric by initially considering class as a module and subsequently group them as packages to improve cohesion. The metric extracted the frequent usage patterns from member functions interactions to capture the cohesiveness of the module, further performed clustering to increase cohesion and decrease coupling among modules. Zhang *et al*. (2018) developed a directed and weighted software dependency network model to measure high cohesion-low coupling nature of object oriented software systems. The authors denoted classes as nodes, their relationships with other classes as edges and the number of occurrences of the relationships as weights for the accurate categorization of dependency strength between software components.

The review clearly insists the invention of newer cohesion metrics as the exiting metrics fails to differentiate the possible levels of cohesions such as low, medium, high and boundaries to marginalize them. Elucidation of cohesion levels in module initially helps the programmers in decision making with respect to acceptance or rejection of modules eventually aids to build reusable and maintainable software. Thus, in this paper an effort has been made to analyze cohesion levels and their inferences.

Methodology

This section explicates the methodology of the proposedVF-IMF cohesion metric by means of clustering the variables used across the module methods. Following are the few important terms used to construct the VF-IMF metric:

- OO Module: The software is broken into modules. Each module is further tokenized as sets of methods having variables used by them as members.
- VF-IMF Dataset: A numerical dataset with row as methods and column as variable is constructed from the categorical sets. The dataset consists value '1', if method 'i' uses variable 'j' in the intersecting 'ij' position or '0' otherwise. This process is similar to the process of document clustering. Here, a variable's frequency within and between the methods are analysed to measure its significance in software module.
- K-means Algorithm: This algorithm is used to group the variables in the form of clusters.
- VF-IMF Cohesion Metric: The output of clustering algorithm plots shared and unshared variable in separate classes and suggests possible grouping of methods into different modules

Figure 1 depicts the process flow of the proposed VF-IMF metric.

Calculating VF-IMF

The datasets created from the OO module is given as input to VF-IMF matrix. VF-IMF is the customized version of popular concept Term Frequency – Inverse Document Frequency (TF-IDF). TF-IDF is the most common and widely used method in text mining and document clustering to group documents of same type. The statistical measure aids to evaluate the importance of a word with respect to corpus of documents, where, the term frequency refers to significance of a term in a document and inverse document frequency refers to the significance of the term across multiple documents. In this work, the terms are assumed as variables and methods are denoted as documents. Computation of VF-IMF with variables and methods in a module is a significant contribution in the proposed work as they enable K-Means algorithm to create clusters of methods that actually shares variables across them. The computation of VF-IMF with respect to proposed cohesion metric is shown.

Variable- Frequency (VF)

VF calculation slightly varies from its base TF. The value of TF in document clustering is the fraction of number of occurrences of term '*i*' in document '*d*' with the total number of terms in the document. On the other hand, *VF* value for variable '*i*' is set to '1', if the variable present in method '*j*' or '0' otherwise. Equation 1 denotes the notion of *VF* for variable '*i*':

$$\forall_{j=1}^{m} VF_{i=1}^{n} = \begin{cases} 1 \; if \;\; variable \; i \; is \; present \; in \; method \; j \\ 0 \;\; otherwise \end{cases} \quad (1)$$

Where:
VF = The frequency of variable *i* in method *j*
n = The number of variables in method *j*
m = The number of methods in module

Inverse Method Frequency (IMF)

The inverse method frequency highlights the importance of a variable '*i*' within module methods, by comparing its presence in other methods. Specifically, the Inverse Method Frequency is shown in Equation 2:

$$IMF_i = \log \frac{|M|}{M_m} \quad (2)$$

Here:
$|M|$ = Gives the number of methods in the module
M_m = The number of methods containing the variable *i*

Another notation of Equation 2 is shown in Eq. 3:

$$IMF_i = \log \frac{total \; number \; of \;\; methods}{number \; of \;\; methods \; with \; variable \; i \; in \; it} \quad (3)$$

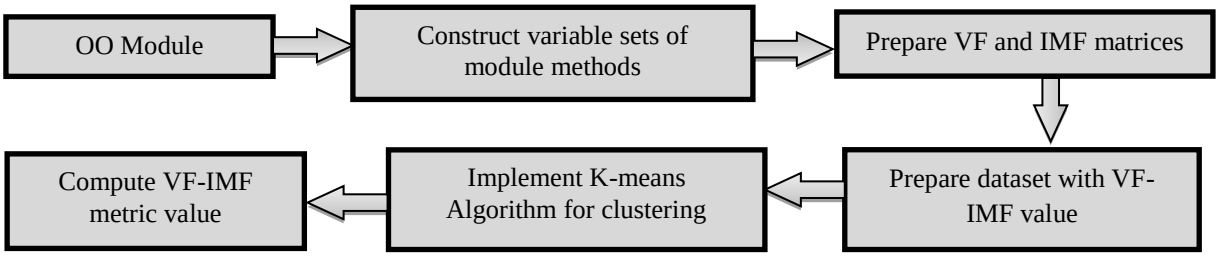

Fig. 1: Process Flow of VF-IMF metric

Algorithmic steps for VF-IMF Transformation of Class Module:
Input: An Object Oriented Class
Output: VF-IMF Dataset

1. for each inverse frequency, F_i, i=1,2,3,…. n for module variables do
2. Set IF_i as 0
3. end for
4. for each method M_j, j=1,2,3,…m in a module do
5. for each variable V_i, i=1,2,3,….n in a module do
6. if variable V_i is found in method M_j
7. Set VF_{ji} as 1
8. Increment IF_i by 1
9. else
10. set VF_i as 0
11. end if
12. end for
13. end for
14. for each inverse method frequency MFi, i=1,2,3,… n for module variables do
15. set IMF_i as m/IF_i
16. end for
17. for each method M_j, j=1,2,3,…m in a module do
18. for each variable V_i, i=1,2,3,….n in a module do
19. set VF-IMF_{ji} as $VF_{ji} \times IMF_i$
20. end for
21. end for

Clustering by K-MEANS

The inputs to K-means algorithm are the cohesion value calculated using VF-IMF. The main goal is to build clusters of methods that have common attributes. Thus, number of clusters to be created by the proposed work is the number of methods that the module has. The interpretation of K-means clustering should be made according to the following findings:

- F1: if only one cluster is created with remaining m-1 empty clusters, the module is said to have high cohesion
- F2: if m clusters created with one instance in each cluster, the module is said to have poor cohesion
- F3: if k clusters created with arbitrary instances in each cluster, m-1 non-empty clusters, the module is said to have moderate cohesion

Taking F3 into account, the non-empty clusters can be reformed to create new modules with high cohesion. K-means algorithm initially chooses ‘m’ random clusters centroids, where ‘*m*’ is the number of methods in the module. The distance between cluster centroids and the instances are computed and grouped into the cluster with the nearest distance. The process gets repeated until the cluster formation of the current iteration is same as the cluster formation of previous iteration (i.e.) when no new cluster formation is possible to reduce the root mean squared error. The formulation of K-Means algorithm is denoted in Equation 3:

$$RMSE = \sum_{i=1}^{n} \sum_{j=1}^{m} (\|x_i - c_j\|)^2 \quad (3)$$

Where:
$||x_i - c_j||$ = The Euclidean distance between instances and cluster centroids
i = The number of instances
m = The number of cluster centroids

Algorithmic steps for K-Means Clustering
Input: VM-IMF Matrix, cluster centroids
Output: Cluster of methods sharing common attributes

1. for each row R_j, j=1,2,3,….m in the VM-IMF Matrix do
2. for each cluster centroids C_i, i=1,2,3…. m do
3. compute $RMSE_{ji}$ as Euclidean Distance of R_j,C_i
4. end for
5. assign R_j to C_i whose distance is minimum
6. end for
7. recompute cluster centroids as $C_i = (\frac{1}{E_i}) \sum_{k=1}^{E_i} x_i)$) where E is the number of elements in the cluster
8. repeat steps 1 to 5 until there is no change in the cluster centroids

Once when the result of K-Means clustering is obtained, the VF-IMF metric value is computed as shown in Equation 4:

$$VFIMF = \frac{total\ number\ of\ methods\ in\ the\ module}{number\ of\ clusters\ created} \quad (4)$$

Experimentation and Result Discussion

The theoretical and empirical validation of the proposed VF-IMF metric is carried out in this section using three cohesion versions of OO programs with the interpretation of low cohesion as High Complexity, medium cohesion as medium complexity and high cohesion as low Complexity.

Empirical Analysis

The empirical analysisis conducted with the intention to demonstrate the methodology of the proposed work with respect to complexity of cohesion in modules. Table 1 denotes the variable sets of three experimental OO modules namely GeoLow, GeoMedium and GeoHigh.

Table 1: Variable set Description of Experimental Modules

S. No	Module Name	Number of Methods in module	Variable Set
1	GeoLow	3	axisX={x1,x2,x3,x4} axisY={y1,y2,y3,y4} axisZ={z1,z2,z3,z4}
2	GeoMedium	6	setXaxis={x1,x2,x3,x4} setYaxis={y1,y2,y3,y4} setZaxis={z1,z2,z3,z4} getXaxis={x1,x2,x3,x4} getYaxis={y1,y2,y3,y4} getZaxis={z1,z2,z3,z4}
3	GeoHigh	3	getAxis={x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,z3,z4} setAxis={x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,z3,z4} printAxis={x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,z3,z4}

Table 2: Cluster results for geolow cohesion complexity version

Cluster number	Total number of methods in the module	Number of methods assigned in the cluster		Cluster percentage
0		1	axisX	33%
1	3	1	axisY	33%
2		1	axisZ	33%

The number of clusters to be created in the proposed work is always the number of methods in the module. The interpretation of experimentation is done with the textual description of the cluster results through Weka data mining tool for better understanding the advantages of VF-IMF metric. Table 2 denotes the cluster alignments of the low cohesion complexity version of the experimental program 'GeoLow'. From the results it has been observed that the K-Means algorithm creates clusters as equal to the number of methods in the module and does not generate any empty clusters. This scenario implies that the methods in the module neither share any data member (variable) nor have any common data member in their variable sets, which means there is no cohesion present in the module. As cohesion strongly impose variable sharing within module methods to discover what method is relevant to a module and how the module can further be used, it is inevitable to revisit and reconstruct the module with sharable data members. Once the attainment of cluster results is over, VF-IMF metric is computed as the fraction of total number of methods with number of clusters created. Thus, VF-IMF metric for the experimental program - Geo Low is computed as '1' as shown below.

$$VFIMF = \frac{total\ number\ of\ methods\ in\ the\ module}{number\ of\ clusters\ created} = \frac{3}{3} = 1$$

Table 3 depicts the cluster results obtained for medium cohesion complexity version of the experimental program for demonstrating the performance of VMIMF metric for medium cohesion. Unlike low cohesion version of the OO program, K-Means algorithm for this experimental program 'GeoMedium' creates three variable groups and three empty clusters. As the number of empty clusters lies between the ranges of medium cohesion from 2 to m-1 clusters, the module is said to have medium cohesion. The results clearly depict that the VF-IMF metric creates clusters of variables and empty clusters when there is a partial sharing of variable between module methods. Medium cohesion means that the module makes up some functionality that is spreading out of the scope of the module. Thus, those methods should be identified to be repaired and reconstructed. Thus, VF-IMF metric for the experimental program GeoMedium is computed as '2' as shown below.

$$VFIMF = \frac{total\ number\ of\ methods\ in\ the\ module}{number\ of\ clusters\ created} = \frac{6}{3} = 2$$

Finally, the proposed VF-IMF metric is computed for high cohesion complexity version of the experimental program for cohesion and shown in Table 4. According to the proposed metric, when the cohesion complexity of a module is low, the variable sets of all module methods should form a single cluster and m-1 empty clusters, which means that all methods in the module share all the variables within module methods and the variable sets of each method is identical from one another. Having executed K-Means algorithm for the low cohesion program, it has been proven that the proposed metric generates a single cluster with m-1 cluster, thus, denotes low complexity cohesion module.

Table 3: Cluster results of geomedium cohesion complexity version

Cluster number	Total number of methods in the module	Number of methods assigned in the cluster	Cluster members	Cluster percentage
0	6	2	getYaxis setYaxis	33%
1		2	getYaxis setYaxis	33%
2		2	getYaxis setYaxis	33%

Table 4: Cluster results of GeoHigh cohesion complexity version

Cluster Number	Total number of methods in the module	Number of methods assigned in the Cluster	Cluster Members	Cluster Percentage
0	3	3	getAxis, setAxis, printAxis	100%

Table 5: Metric value analysis

Complexity version	LCOM1	LCOM2	LCOM3	TCC	LCC	VF-IMF	VF-IMF Description
GeoLow	3	0.667	1	0	0	1	Clusters having Variable sets as Members with Non empty clusters
GeoMedium	8	0.337	0.8	0.533	0.65	2	2….m-1 clusters with empty clusters
GeoHigh	0	0	0	6	6	3	Single Cluster with m-1 empty clusters

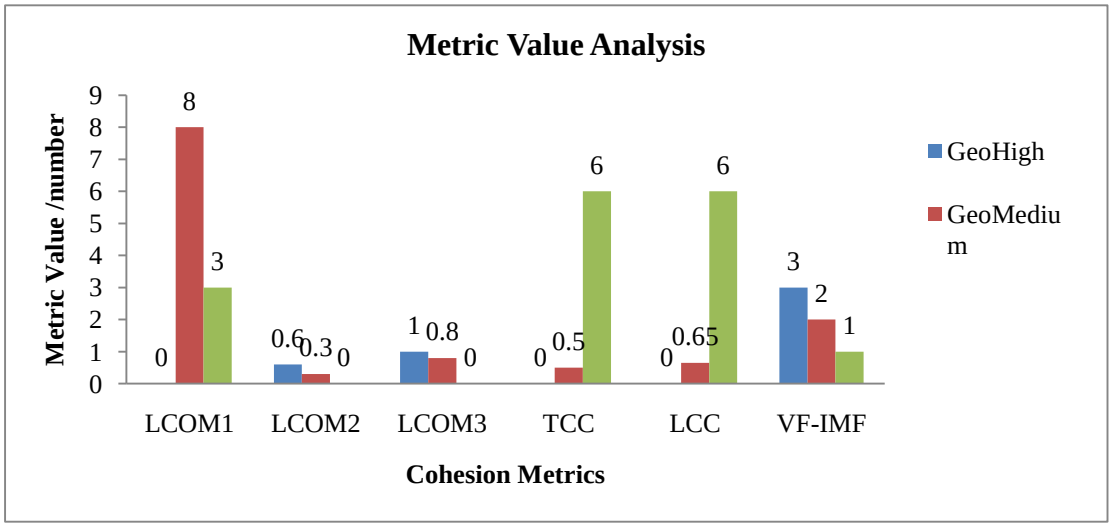

Fig. 2: Metric value analysis

When the module is designed by sharing all variables amongst module methods, the maintenance of a module becomes easier and the variables within the module are directly related to the functionality that module is meant to provide and reduces duplication of knowledge in modules. VF-IMF metric for the experimental program GeoHigh is computed as '3' as shown below.

The comparison of proposed VF-IMF metric and other traditional cohesion metric is shown in Table 5.

The pictorial representation of the metric value analysis of the traditional cohesion metrics LCOM1, LCOM2, LCOM3, TCC and LCC with VF-IMF is shown in Fig. 2. The figure shows less meaningful interpretation of traditional metrics with respect to low, medium and high cohesions. The distribution of traditional cohesion metric values from low cohesion to high is not linear and confuses the programmer in making reconstruction decisions. On the other hand, the

VF-IMF metric values for the low, medium and high cohesions are 1, 2 and 3 respectively. The metric clearly measures the low cohesion complexity program as 1, medium cohesion complexity program as 2 and high cohesion complexity program as3, thus, showing a linear relationship on the distribution of metric values. This kind of representation indeed helps the testers for the effective reconstruction of the module. In addition, the proposed work visualizes cluster results through graphical notations that helps the programmer to identify the possible construction of methods to create high cohesion modules.

Theoretical Validation

Weyuker suggested nine properties to evaluate any software metric to be proven as valid (Mal and Rajnish, 2013; Rana and Singh, 2018; Xiang *et al*., 2019). Hence, this section theoretically validates the proposed VF-IMF metric against the properties of Weyuker's:

1. Non-Coarseness: VF-IMF metric does not rank all modules with same metric value
2. Non-Negativity: VF-IMF metric does not result a negative number as metric value
3. Non-uniqueness: VF-IMF metric may result same metric value for finite number of classes, if the modules are equally as complex
4. Design Details are Important: VF-IMF metric does not result same metric value when the implementation of two modules are different even if they provide same functionality
5. Monotonicity: VF-IMF metric value for two combined modules can never be less than the metric value for either of modules
6. Non-equivalence of Interaction: VF-IMF metric does not result the same value for two modules pertaining same functionality when interacting with a different module
7. Permutation: VF-IMF metric is not sensitive to the permutations of modules, even though the property does not suitable for object oriented programming
8. Renaming: VF-IMF metric value does not affect by renaming module.
9. Interaction increases complexity: The combined VF-IMF metric value of two modules is higher than the sum of metric values of individual modules as the interaction increases complexity.

The proposed VF-IMF metric satisfies all the properties of Weyuker. Thus, VF-IMF is proven to be valid software metric.

Conclusion

In this paper, through VF-IMF metric, the variables are clustered to depict what level of cohesion exist in a module among the three low, high and medium cohesions, so that the role of a developer is made easier to develop a module that ensures reusability of the code. The quality of the metric is also validated against a number of modules through theoretical and empirical analysis. The result shows that the performance of VF-IMF metric is good in elucidating the existence of low, medium and high level cohesions in a module than that of LCOM along with the graphical representation of variable clusters. If the module is refined by grouping the variables with the clusters created by K-Means algorithm will lead to high cohesion. In addition, the theoretical validation of VF-IMF metric also conforms to all the properties of Weyuker to be proven as a valid metric. In future, the approach can also be extended to identify the usefulness of machine learning techniques to measure the levels of coupling in a module, as the modules are expected to be designed with high cohesion and low coupling.

Acknowledgement

The authors would like to thank SASTRA Deemed to be University for the facilities used in this work

Author's Contributions

All authors have equally contributed to the final version of the manuscript

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and there are no ethical issues involved

References

Alzahrani, M., S. Alqithami and A. Melton, 2019. Using client-based class cohesion metrics to predict class maintainability. Proceedings of the IEEE 43rd Annual Computer Software and Applications Conference (COPSAC' 19), IEEE Xplore press, pp: 72-80.

Cinnéide, M.O., I.H. Moghadam, M. Harman, S. Counsell and L. Tratt, 2017. An experimental search-based approach to cohesion metric evaluation. Empirical Software Eng., 22: 292-329.

Ganesh, S.H. and H.B. Vincent Raj, 2015. Performance based analysis on MALCOM-A software metric. Proceedigs of the International Conference on Circuits, Power and Computing Technologies (ICCPCT' 2015), IEEE xplore press, pp. 1-5.

Goel, B.M. and S.B. Gupta, 2017. Dynamic coupling based performance analysis of object oriented systems. Int. J. Advanced Res. Computer Sci.

Gu, A., X. Zhou, Z. Li, Q. Li and L. Li, 2017. Measuring object-oriented class cohesion based on complex networks. Arabian J. Sci. Eng., 42: 3551-3561.

Kansal, D., T. Aher and R.K. Joshi, 2019. Sensitivity and Monotonicity in Class Cohesion Metrics. Proceedings of the 12th Innovations on Software Engineering Conference (formerly known as India Software Engineering Conference), ACM.

Kerdoudi, M.L., C. Tibermacine and S. Sadou, 2016. Opening web applications for third-party development: A service-oriented solution. Service Oriented Computing Applications, 10: 437-463.

Mal, S. and K. Rajnish, 2013. Applicability of weyuker's property 9 to inheritance metric. Int. J. Computer Applications.

Purao, S. and V. Vaishnavi, 2003. Product metrics for object-oriented systems. ACM Comput. Surveys, 35: 191-221.

Rana, P. and R. Singh, 2018. Evaluation and applicability of weyuker's properties on software metrics for component based software system. Int. J. Engineering Sci. Mathematics, 7: 495-505.

Rathee, A. and J.K. Chhabra, 2018. Improving cohesion of a software system by performing usage pattern based clustering. Procedia Computer Sci., 125: 740-746.

Santos, L.D.C., R.M. Saraiva, M. Perkusich, H.O. Almeida and A. Perkusich, 2017. An empirical study on the influence of context in computing thresholds for Chidamber and Kemerer metrics. SEKE.

Wu, D., L. Chen, Y. Zhou and B. Xu, 2015. A metrics-based comparative study on object-oriented programming languages. State Key Laboratory for Novel Software Technology at Nanjing University.

Xiang, Y., W. Pan, H. Jiang, Y. Zhu and H. Li, 2019. Measuring software modularity based on software networks. Entropy, 21: 344.

Zhang, J., J. Wu, Y. Xia and F. Ye, 2018. Measuring cohesion of software systems using weighted directed complex networks. Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS' 18), IEEE Xplore press, pp: 1-5.