International Journal of Computing and Digital Systems

ISSN (2210-142X)

Int. J. Com. Dig. Sys. #, No.# (Mon-20..)

On the Relations among Object-Oriented Software Metrics: A Network-Based Approach

Marwah M.A. Dabdawb[1], Basim Mahmood[2]

[1]Software Engineering Department, University of Mosul, Mosul 4100, Iraq
[2]Computer Science Department, University of Mosul, Mosul 41002, Iraq
[2]BioComplex Laboratory, Exeter, UK

E-mail address: Marwa_marwan21@uomosul.edu.iq, bmahmood@biocomplexlab.org/ bmahmood@uomosul.edu.iq

Received ## Mon. 20##, Revised ## Mon. 20##, Accepted ## Mon. 20##, Published ## Mon. 20##

Abstract: Recent years have witnessed a great revolution in software applications. The quality of the software is important insofar as it contributes to providing better services for users. Software metrics are mainly used to obtain feedback on the quality of software design. These metrics enable developers to identify the potential weaknesses in their designs. Furthermore, software metrics may have correlations with each other and may impact the outcome of each other. This specific case may lead to a misleading interpretation of the design, which eventually affects the quality of software and waste the time and effort during the design phase. Therefore, selecting the appropriate metric during software design should be carefully performed. In this work, we use network science concepts in deeply investigating the relations among object-oriented software metrics. The analysis approach is based on network visualization and measurements. The dataset of this work was collected from accredited references in the field of Software Engineering. This study involves the main 104 metrics that are basically used during software design. The findings depict interesting facts on the relations among metrics. This work is considered as a comprehensive analysis and assessment that takes into account different dimensions of the relations among software metrics. We believe that the analysis and the results can make it easy for developers in selecting the appropriate metrics during the design phase.

Keywords: Complex Network, Data Analysis, Object-Oriented Metrics, Software Engineering

1. INTRODUCTION

In recent decades, many software measurements have been developed and used extensively in assessing software quality. These measurements are considered crucial factors in determining the quality of software products [1]. Software measurements are also referred to as *Metrics*. These metrics provide managers and developers with feedback in a form of quantitative observations on the property of software in terms of the following [2]:

- Quality of software and system complexity.
- Deduction of effort in the design and development.
- Ease of use and the difficulty of testing.
- Understanding, tracking, and progress during the various stages of software development life cycle.

Nowadays, the software development process relies massively on object-oriented models. To assess the quality of object-oriented software, several metrics can be involved in the process. In a software development environment, object design is a crucial aspect according to "IEEE Software Engineering Standards", and it is essential to measure software design quality by using these metrics, which assist in verifying the quality of software properties [3].

Software metrics have been classified into Traditional metrics and Object-oriented metrics. The latter, which is our scope in this study, has become a dominant approach to structure software requirements, designs, and implementations. It applies the concepts of understandability, reliability, reusability, and maintainability in a better manner than the traditional approach. Object-oriented designs are constructed using classes, each of which contains a set of attributes and methods, and the implementations of these classes are objects [4][5].

The software design phase is considered one of the most crucial aspects of software quality. Practically, it is important for software developers to evaluate their designs at each step using appropriate metrics. The decision of selecting a metric for the software model should be carefully performed. However, using inappropriate metrics may lead to a misleading interpretation of software evaluation, which eventually causes a loss in time and effort. In fact, a well-understanding of software metrics plays a key role in obtaining an optimal evaluation for a design. Understanding a metric depends on the concept of that metric as well as on its relations to other metrics.

Although of its importance, the problem of understanding the relations among metrics has not received enough attention in the literature. Few studies considered this issue in their works and showed the influence of metrics relations on the consequent decisions of the design. Recent work in 2020 was performed by Kuk et al. [6] and presented an analytical study of the object-oriented metrics to find the relationship between the value of the metrics with the level of security of the software. These relationships were determined by examining software vulnerabilities with code-level metrics. The authors demonstrated a relationship between the considered metrics and software security issues by using CWE/OWASP tools that were used for the classification of software vulnerabilities. In the same year, Schnoor and Hasselbring [7] conducted a study to find a relation between the dynamic weighted metrics and their corresponding static metrics. Their data were collected from four diverse experiments and the results showed that there was a strong relationship between dynamic and static metrics. They also found a difference in the analysis between class level and package level metrics. In the same context and for assessing system architecture, Etzel et al. [8] in 2020 used metrics from code level and object-oriented design analysis to enrich system architects with assessment information about the quality of the system architecture. The collected metrics were examined with Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL) model. The study provided important suggestions for software developers.

Moreover, Aggarwal et al. [9] conducted an empirical study on the relations among 22 software metrics that were developed by several researchers. The authors applied these metrics on three projects and presented descriptive statistics, analysis of components, and correlation analysis. They relate these metrics to each other and deduced one group of metrics that can provide sufficient information for use. Another study was performed by Prasad and Nagar [10] on the relation between two types of metrics. The study examined the correlation between the current object-oriented metrics (Coupling and Cohesion) and procedurally oriented metrics (Cyclomatic Complexity, Line of Code, and Knot metric). The authors also performed an empirical study to determine a new set of metrics that capture new dimensions in coupling measurement, which are used in the conceptual coupling of classes. Although many metrics were introduced in the literature, there is still a lack of understanding of how these measurements relate to each other. In this context, Ó Cinnéide et al. [11] proposed an empirical technique to estimate software metrics and to seek the relationships among them. This technique was based on search-based refactoring. To achieve this, the researchers implemented their approach on 5 common Cohesion metrics. They used eight Java systems (real world), involving 300,000 lines of code and more than 3,000 refactorings. Their results revealed significant insights into the chosen software metrics.

Furthermore, Chong and Sai [12] introduced an approach to integrate and harmonize current metrics that evaluate the complexity of object-oriented software systems based on 3 levels of metrics: code level, system-level, and graph level. First, an object-oriented source code was turned into UML class diagrams. Then, using the concepts of complex networks, classes were converted into nodes while edges represented the relationships among the nodes. Based on code-level and system-level metrics, nodes and edges were weighted based on the complexity of classes and the relationships among them. After that, they analyzed the software system using graph-level metrics to collect patterns that reflected particular characteristics of maintainability and reliability of software systems. Another study performed by Bhardwaj and Ajay [13] assumed that although software projects are different and maybe unique; but still have features in common such as software size, duration, effort, and productivity. The authors clarified the relationship among these main (key) metrics and how they affect each other. They also illustrated how these metrics can be applied in predicting the total number of defects in software. The study concluded that the software size metric is the most important and has the most influence over other metrics.

According to the literature, we still have the problem of understanding the relations among metrics in terms of the following:

- Most of the studies considered a few numbers of metrics when analyzing the relations among them.
- Most of the studies did not provide a clear definition of how to relate a metric to another one.

In this work, we try to fill the aforementioned gaps and hence our contributions are:

- Perform a comprehensive study on the relations among 104 metrics that belong to 11 object-

oriented software properties.
- Propose a definition for relating metrics to each other and makes it useful in this study.

The advantages of this work lie in the following:

- Supports software developers in making suitable decisions on the object-oriented metric(s) they use during the design phase.
- Enriches developers with useful knowledge on the relations among software metrics, and;
- Provides developers with a wider and a deep view of the evaluation of their designs, which saves the consumed time and effort.

The rest of this paper is organized as follows: the next section presents the research method that is used in our study including the dataset collection and network creation. In section 3 we present the obtained results and discuss them. Finally, we conclude our work in Section 4.

2. RESEARCH METHOD

This study represents an analysis and evaluation of the relations among object-oriented metrics. This kind of works uses network measurements as the main tool for obtaining technical results. In this section, we describe the evaluation measurements that are used in the analysis. We also describe the process of data collection and network creation method.

A. Network Measurements

The measurements of this work are inspired from the field of complex networks. This kind of measurement is considered the most suitable when investigating relations-related issues. The field of complex networks is one of the modernist fields of study that was started in the 2000s. The structure of a complex network can be formed as a *Graph (G)* that contains *Vertices (V)* and *Edges (E)* connecting them. A *Weight (W)* can be assigned to each edge within a graph (network) based on the nature of the relation between two nodes. Practically, several measurements can be used for evaluating the performance of a network in two levels, Network-Level and Node-Level as follows:

Network-Level Measurements [14] can measure a particular feature in the whole network structure and include the following:

- Average Degree (Avg_D) of a network. This measurement reflects the average number of relations for all the nodes in the network.
- The density (D_S) of a network is another measurement that shows the ratio of the potential connections to the actual number of connections in the network.
- The clustering Coefficient (C_O) of a network reflects the tendency of a node to cluster with other network nodes and can be formalized as follows:

$$C_{O(i)} = \frac{2\left|\{l_{ik}: n_j, n_k \in N_i, l_{ik} \in E\}\right|}{ki(ki-1)} \quad (1)$$

Where l_{jk} is the edge between the nodes nj and nk. N_i is the total number of nodes and ki is the neighbors of node i in the network. The average clustering coefficient (Avg_C) is the mean of all the C_O values in the network.

- The shortest path length can also be considered as an indicator of network structure. The average shortest path length (P) reflects the average shortest paths among all network nodes.
- The diameter (D_T) of a network represents the path length between the farthest nodes in that network.

Node-Level Measurements [14] can evaluate the performance of nodes in a network and include the following:

- Degree Centrality (C_d) of a node represents the number of connections of a node in a network.
- Betweenness Centrality (C_b) reflects how well-positioned a node in the flow of information within a network. This means high values of C_b reflect a high level of importance for a node in a network. The C_b of node j can be defined as follows:

$$C_b(j) = \sum_{i \neq j \neq k} \frac{\sigma_{ik}(j)}{\sigma_{ik}} \quad (2)$$

Where σ_{ik} is the shortest path between the nodes i and k. $\sigma(j)$ is the number of paths that pass-through node j.

- Closeness Centrality (C_c) shows how close a node from other network nodes and can be formalized by the following:

$$C_c(i) = \frac{N-1}{\Sigma_j d(ji)} \quad (3)$$

Where $d(ij)$ is the distance between the nodes i and j.

- Eigen Centrality (E), this measurement evaluates the influence of a node in a network in terms of its connections to the well-connected nodes in that network. To calculate E, consider a graph $G(V, E)$, where V is a set of Vertices and E is a set of Edges among these vertices and an adjacency matrix $A = (a\ v, t)$ for the vertices v and t such that $a\ v,t = 1$ if both nodes are connected and 0 otherwise. The X score for node v can be as follows:

$$X(m) = \frac{1}{\lambda}\sum_{t\in M_{(v)}} x_t = \frac{1}{\lambda}\sum_{t\in G_{(v)}} a_{m,t}x_t \quad (4)$$

Where $x(m)$ is the neighbors of node m and λ is the Eigenvalue. As a vector notation, the equation above can be rewritten as follows:

$$A_x = \lambda_x \quad (5)$$

This term represents the Eigen centrality *E* of a node (metric).

B. Dataset Collection

We collected a dataset that includes 11 design properties that were presented by the distinguished work of Bansiya et al. [15], namely, Design size, Hierarchy, Abstraction, Encapsulation, Coupling, Cohesion, Composition, Inheritance, Polymorphism, Messaging, and Complexity. For each property, we collected the most popular related metrics that contribute to assessing that property. To this end, we collected 104 object-oriented design metrics that were presented in the literature (see the Appendix). These metrics were at different levels (Object, Class, Package, and System) and different states (Static or Dynamic [5]). Thereafter, each metric was assigned to its corresponding design property.

C. Network Creation

In this section, we present the proposed strategies for forming the dataset (nodes and edges) and making it suitable for generating the network. In general, the dataset of a network should define the nodes and edges. As mentioned, to calculate a metric, parameters (or formulas) should be involved in the calculation process. Therefore, each metric was represented as a node, and two metrics were considered to be connected by an edge if and only if there existed parameters (or formulas) in common (see Fig. 1). This means, metrics from different properties could be connected, which was desired since this work digs deeply into the relations among different metrics that belong to different properties. The weight of the edge for a pair of metrics was driven by the number of parameters in common. The more parameters in common between the two metrics, the more weight was set. The dataset in this case will be used to generate what we call MTR Network (**MeTRics** Network).

3. RESULTS AND DISCUSSIONS

In this section, we present the visualizations and analysis of the results of the MTR network. Fig. 2 depicts the visualization of the MTR network, which shows how dense the relations among different properties.

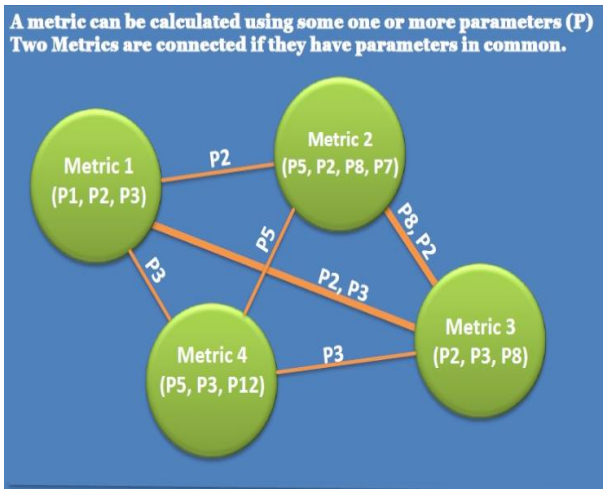

Figure 1. The creation of nodes and edges in the Metrics Network.

The main characteristics of the MTR network are presented in Table I. The average degree in the MTR network reflects the average frequency of connections among the metrics from different properties, which was high compared to network size (nodes and edges). Also, the average shortest path between any given two metrics in the network is approximately 2 edges meaning that the distances among different metrics are short regardless of the property they belong to. This result is also confirmed when we observe the diameter of the network. Since the average degree of the network is high, the density of network relations is also high taking into consideration the number of nodes in the network. Moreover, the MTR network reflects a strong average tendency of its metrics to cluster together and form a community. In this regard, we tested the communities in MTR using the Girvan-Newman algorithm [44]. This algorithm finds the edges with high betweenness centrality values, then, it removes these edges leaving the nodes (metrics) themselves. The steps of the Girvan-Newman algorithm can be as follows:

- Step 1: For a given network, calculate the betweenness centrality for all network edges.
- Step 2: Remove the edges that have the highest betweenness levels (edges that connect the communities).
- Step 3: Re-Calculate the betweenness centrality for all network edges.
- Step 4: Repeat Step 2 and Step 3 until all the edges removed from the network.

The experiments show that MTR metrics tend to form 36 communities of metrics when reaching a modularity level of 0.174 considering the number of edges in the MTR network (see Fig. 3).

The figure shows that when decreasing the modularity level in the algorithm, the number of detected edges is significantly increased. These preliminary findings lead us to dig into the network and analyze the relations among the metrics. The better way to perform such an analysis is to involve network centrality measurements and analyze the obtained results. To this end, we decide to use the centrality measurements that were described in Section 3.

TABLE I: THE CHARACTERISTICS OF THE MTR NETWORK IN TERMS OF NETWORK MEASUREMENTS

# of Nodes	# of Edges	Average Degree	Average Clustering Coefficient	Average Path Length	Diameter	Density
104	1122	22.65	0.88	1.98	4	0.22

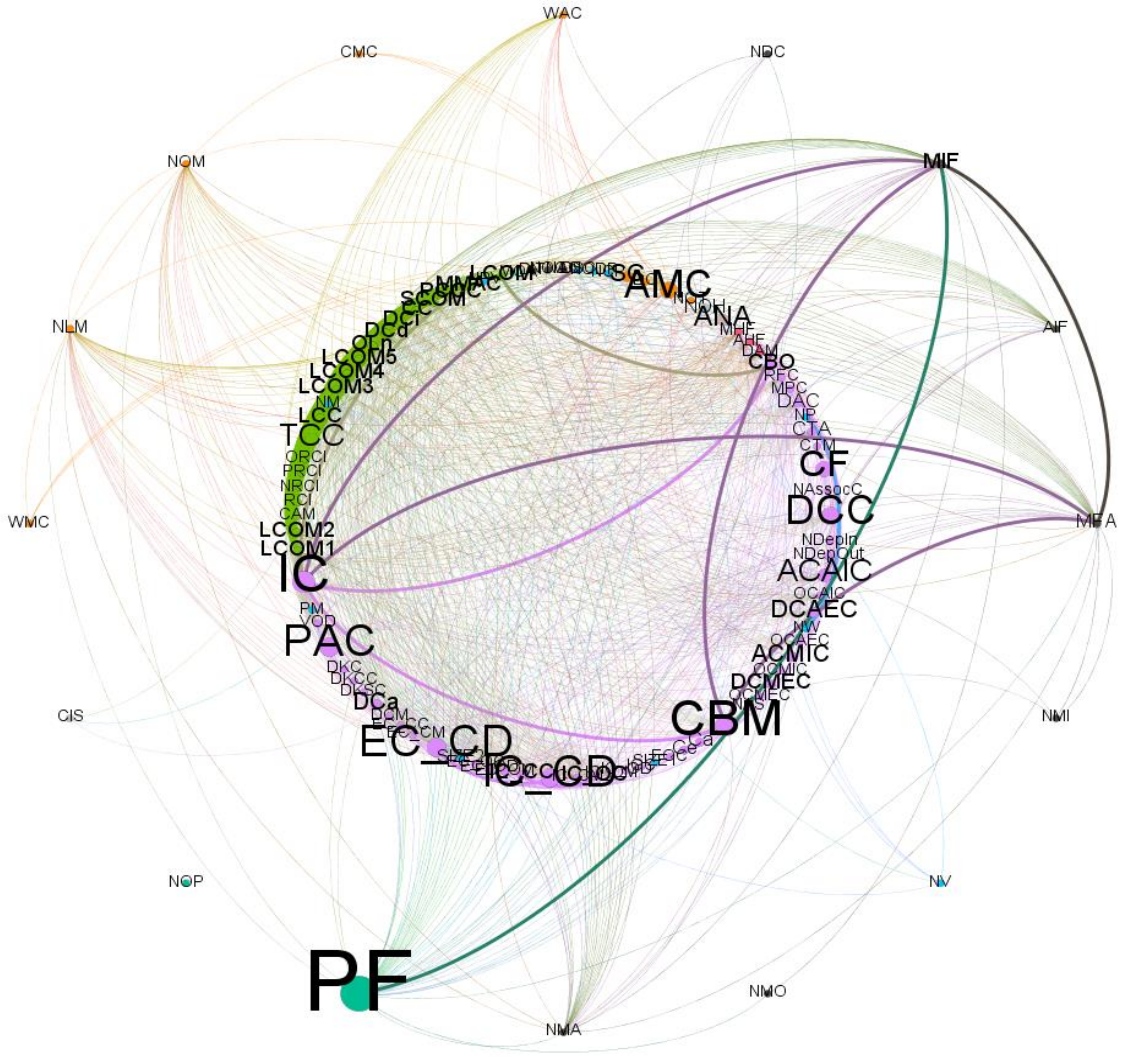

Figure 2. The visualization of the MTR network. Different colors reflect different properties in the MTR network. Node size reflects the dense of the connections (degree).

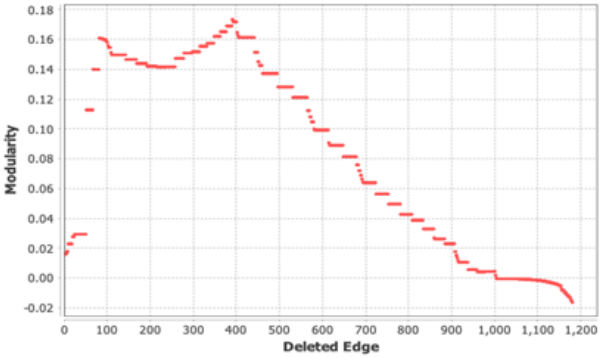

Figure 3. Performance of Girvan-Newman algorithm and the modularity level

The first step in testing network centrality measurements on the MTR network is to show the impact of the Collective Centralities (C_L) of a metric m. Therefore, we performed calculations based on the following equation:

$$C_L = C_b(m) + C_c(m) + C_d(m) + E(m) \quad (6)$$

Since the clustering coefficient of a metric can explain the tendency of a metric to cluster with the other metrics, we propose to raise the power of C_L to the clustering coefficient and calculate the strength of each metric compared to the other metrics in the MTR network. Hence, the strength *S* of a metric *m* is calculated as follows:

$$S(m) = (C_L)^{C_O(m)} \quad (7)$$

Fig. 4 shows the mean values of the strength of each property. The strength of a property is calculated by averaging the strength of each metric classified under that property. The figure shows that Coupling is the strongest property in terms of its relations to other metrics in the MTR network followed by Encapsulation and Composition properties. This finding is important and tells us that software developers should give enough attention to the strength of properties during the design phase. Moreover, we analyze the variations of each property in the MTR network (see Fig. 5). According to this analysis, some properties have a few numbers of metrics; therefore, they show a low level of variations. Fig. 5 also shows interesting results, for instance, the Coupling property reflects a high level of variations. It means that the Coupling property has metrics with weak relations, which drives this behavior in the variations. This phenomenon has appeared in many properties in the MTR network such as Complexity, Polymorphism, and Inheritance. Therefore, we decided to visualize the strength of all the metrics in the MTR network.

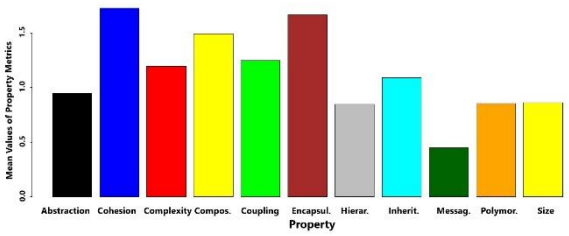

Figure 4. Strength level for properties

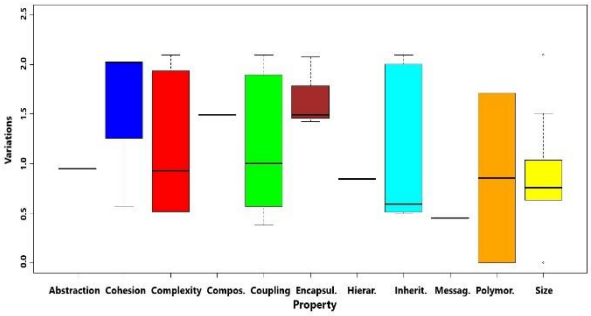

Figure 5. Variation levels for each property in the MTR network.

Fig. 6 depicts the strength of each metric along with its property. Based on this visualization, it is clear that Coupling and Cohesion properties have a high number of strong metrics with a few weak metrics that led to the variations.

The stated results were performed using Equation 7. However, for the sake of the analysis to be more in-depth, we plan to investigate the impact of each network centrality measurement in *CL*. In this context, we test the four centrality measurements by involving them in a regression model where the measurements represent the independent variables and *CL* is the dependent as follows:

$$CL \sim Betweenness + Eigen + Closeness + Degree \quad (8)$$

Using this model, our hypothesis testing as follows:

Null Hypothesis:

$$H_0: \mu(C_b) = \mu(E) = \mu(C_c) = \mu(C_d) \quad (9)$$

Alternative Hypothesis:

$$H_1: \mu(C_b) \neq \mu(E) \neq \mu(C_c) \neq \mu(C_d) \quad (10)$$

Table II presents the one-way **AN**alysis **O**f **VA**riance (ANOVA) table of our model. Given this output, we cannot accept the null hypothesis of equal means of the four centrality measurements, and there exist differences in the measurements according to the alternative hypothesis.

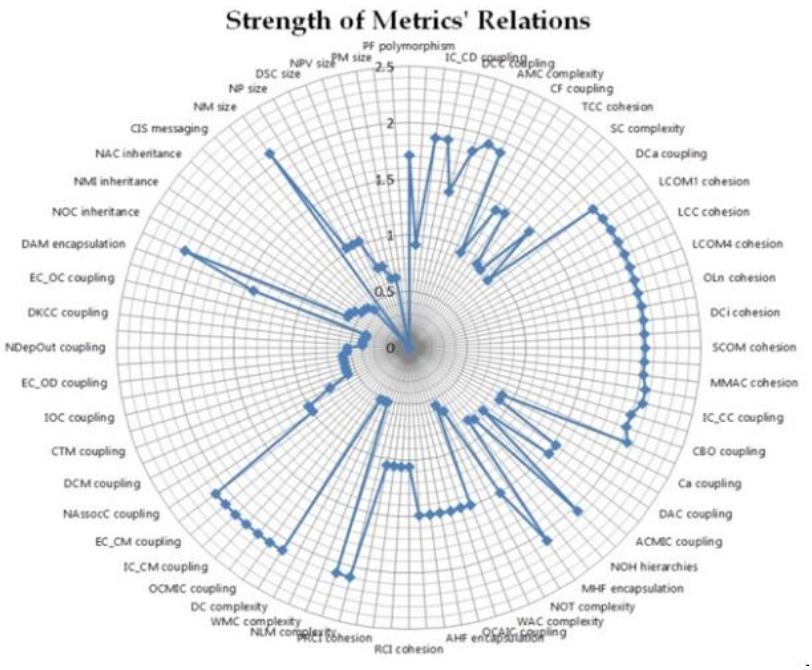

Figure 6. Strength of metric relations.

TABLE II: ONE-WAY ANOVA FOR THE CL MODEL

	Sum of Squared Errors	Mean of Squared Errors	F-Statistics	P-Value
Measurements	17.43	5.809	75.88	0.00002
Residuals	31.54	0.077		

Furthermore, we evaluate the measurements in terms of their impact on the model. To this end, we use the Akaike Information Criterion (AIC) method and created several models. These models are created using combinations of the centrality measurements that are used to calculate *CL*. The results show that when considering the betweenness centrality measurement as an independent variable in the models, AIC produces minimum outputs (better models). This specific case could not be obtained using the other measurements. According to that, the betweenness centrality measurement can be considered as the most significant contributor to the collective centralities. Also, most of the variations in *CL* were explained by the betweenness as well as it is considered as a central concept when it comes to relations. Hence, this result can be used as an indicator in our further analysis in this work. Now, based on the aforementioned finding, Fig. 7 shows the levels of betweenness centrality for all the metrics in the MTR network. It can be observed that Cohesion metrics have the highest level of betweenness centrality, which means they are well-positioned in the MTR network and the metrics under this property are considered as bridges across different properties. Another interesting result is that the PF metric has the highest level of betweenness centrality, which means it is a well-positioned metric within all the shortest paths in the MTR network. This is because PF parameters contribute to calculating several metrics in MTR. Since the MTR network has influential metrics, developers should be aware of this fact and take it into account during the design phase. In addition to the presented analysis and results, we performed a correlation analysis for all the properties in the MTR network. Fig. 8 shows the correlation matrix (1 (strongest) to -1 (weakest)) and the relations among all the pairs of properties. The strongest pairs in MTR were (Inheritance-Complexity), (Size-Complexity), and (Size-Inheritance). Interestingly, it can be seen that three properties gained the strongest relations in MTR. This outcome is due to the common concepts that are associated with them (e.g., internal interactions). The correlations among other properties are also shown in the figure. In software engineering, Coupling is usually contrasted with Cohesion; therefore, they correlate with each other. Furthermore, we perform more visualization for the MTR network.

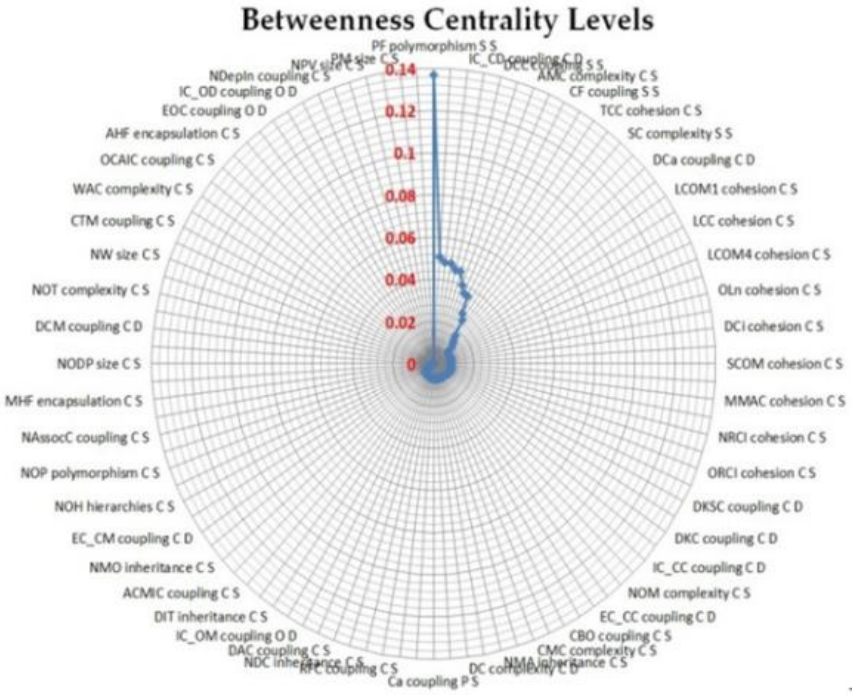

Figure 7. Level of betweenness centrality

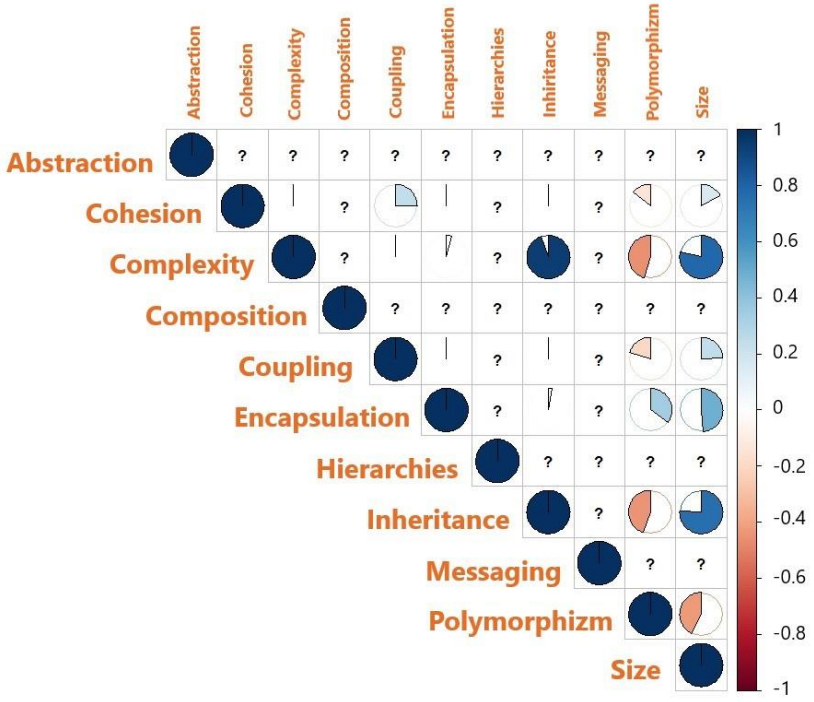

Figure 8. Correlation matrix of properties in the MTR network

Fig. 9 depicts a property-based visualization for all the metrics. This visualization tells us many facts on the MTR network. For instance, there is a strong relation between Coupling and Cohesion through the metrics pairs (IC-LCOM) and (CBO-LCOM). In fact, Coupling contains measurements that are related to the degree of association and dependence of one module with another, while Cohesion represents the correlation of the parts of one module [45]. Another reason behind this relation is that they use the same parameters but with an external concept in Coupling and an internal concept in Cohesion. Furthermore, the coupling property contains 6 strong within-pairs (DAC-CTA), (Ca-Ce), (IC_CC-IC_OC), (IC_CD-EC_CD), (IC-CBO), and (IC-CBM), which makes the clustering coefficient of this cluster to be 0.87. Also, the relation between Coupling and Inheritance through the metrics ((IC-MFA), (CBM-MFA) and (CBM-MIF)) is due to the fact that there are three types of coupling: interaction coupling, component coupling, and Inheritance coupling, which is generated from inheritance [46]. The other relation is between Polymorphism and Inheritance. This relation is originated because polymorphism arises from inheritance [47]. Finally, it was observed that all the metrics in Cohesion property are connected with strong relations except the metrics NRCI, CAM, RCI, and LCOM; they are weakly connected to the other metrics in their property. This is because they use unique parameters that are not frequently used by the other metrics in the Cohesion property.

The other visualization that was performed on the MTR network is Level-based, which can be Class, System, Object, and Package levels. Fig. 10 portraits the classes in the form of clusters, each of which has a different color. The visualization also shows that the biggest cluster is the Class level metrics, while the smallest one is the Package level. We can observe that there is a strong relation between Class and System levels through the pair (MIF-PF) as well as between Class and Object levels through the pair (IC_OC-IC_CC). Therefore, these two pairs of metrics play as bridges across different levels.

The last visualization of the MTR network is performed based on the state of the metrics, which can be Static or Dynamic. Fig. 11 shows that the relation between Static and Dynamic state metrics exists and the Static metrics have strong relations with each other. In fact, static and dynamic metrics have parameters in common; therefore, there is a relationship between them but the circumstances when collecting these metrics is different. Dynamic metrics usually collected in run-time, while static metrics don't need execution. That means static metrics deal with the structural aspects of the software system while dynamic metrics deal with the behavioural aspects of the system. Also, the major dynamic metrics proposed for Coupling and Cohesion.

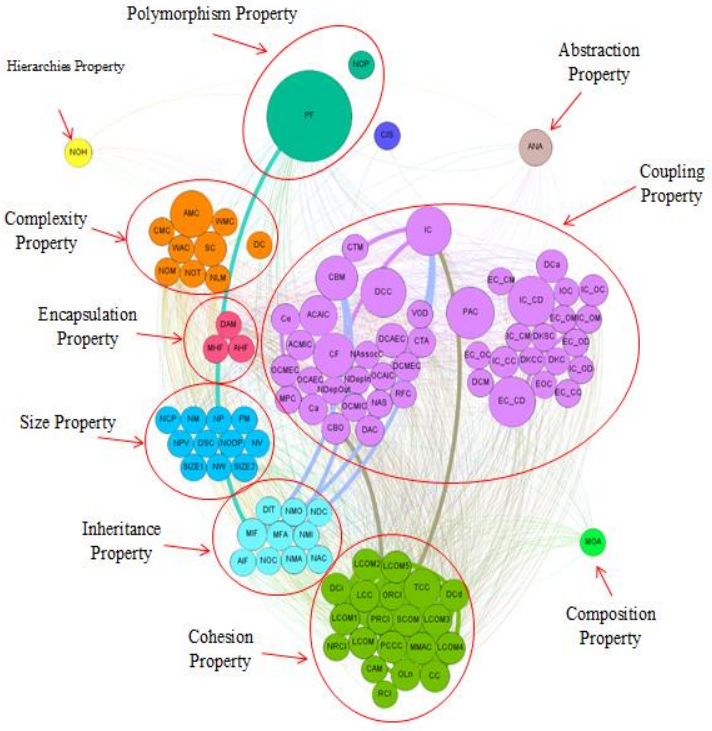

Figure 9. Property-Based visualization of the MTR network. Each property has a different color and the size of nodes reflects the levels of Betweenness Centrality (high levels reflected by the big size nodes).

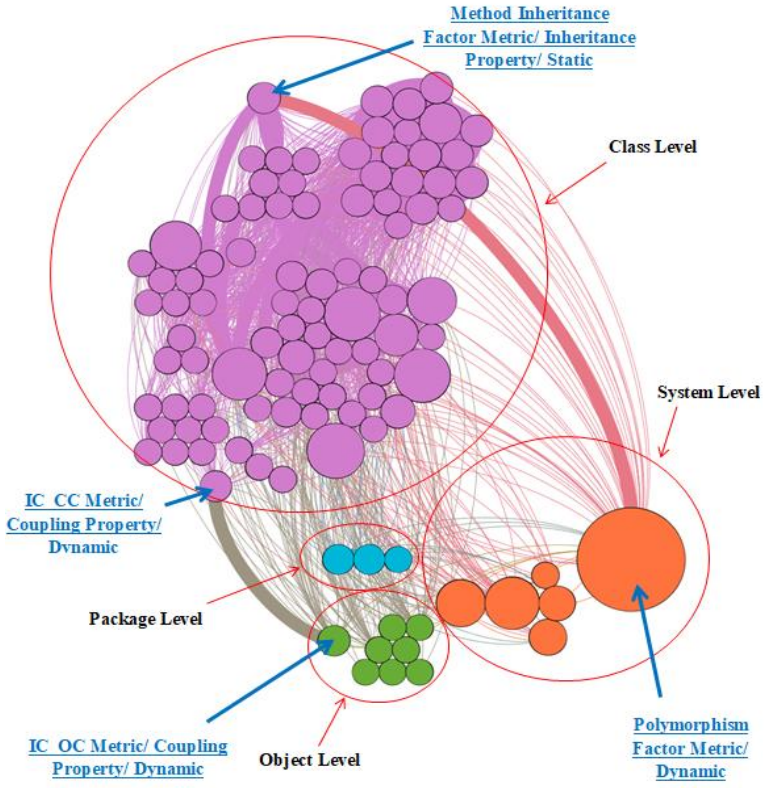

Figure 10. Level-Based Visualization of the MTR network. Each level has a different color and the size of nodes reflects the levels of Betweenness Centrality (high levels reflected by the big size nodes).

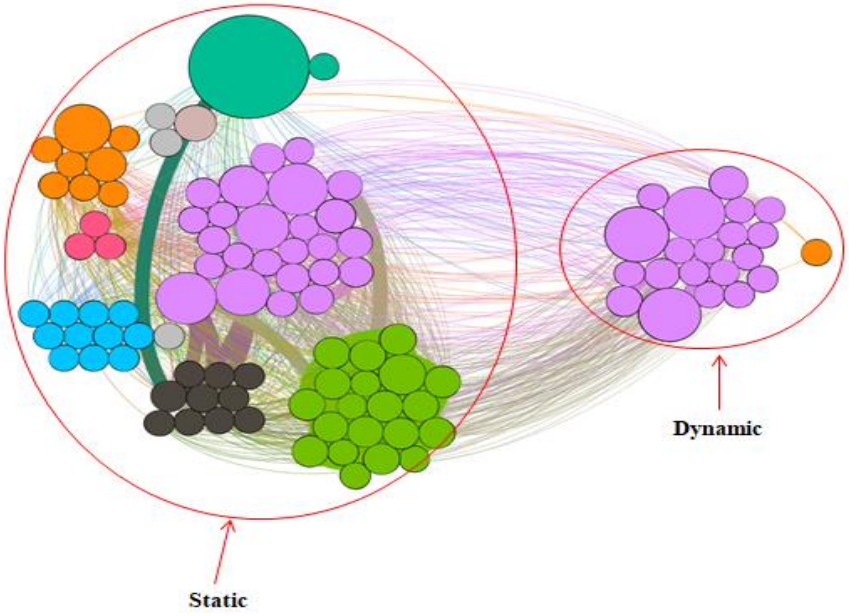

Figure 11. State-Based Visualization of the Metrics Network. The Static and Dynamic state metrics are shown in two clusters. Different colors reflect different properties. The size of nodes reflects the levels of Betweenness Centrality (high levels reflected by the big size nodes).

4. CONCLUSIONS

This article presented a comprehensive study on software object-oriented metrics. We generated what has been called the MTR network, which contained metrics and relations among them. We visualized, analyzed, and evaluated the network using concepts and measurements inspired from the Complex networks field.

Our results could be of interest to software developers during the design phase. We believe that investigating the relations among software metrics is an important aspect that should be of focus by software literature. It could be concluded that network measurements can be considered as strong tools for analyzing the relations among software metrics. The use of such techniques provided us with a deep view from different angles and different dimensions to the data because the nature of this approach digs into the relations among objects and how they relate to each other.

Moreover, we found many interesting facts on the metrics and they can be useful when it comes to software design assessment. Also, the results showed that several metrics should be given more attention by software developers since most of them have relations with each other.

Finally, understanding the relations among software metrics, as we strongly believe, plays a significant role in producing well-designed software and makes it easier when making a decision on using a particular metric, which eventually reduces the time and efforts that can be consumed in the assessment of the design.

As future work, we are working on extending our dataset to include more metrics in addition to the currently considered object-oriented metrics. We also plan to generate a Giant component that includes the majority of software metrics aiming at having different views on the relations among different kinds of metrics.

ACKNOWLEDGMENT

We are grateful to the Software and Computer Science departments/University of Mosul for making this work achieved. We also would like to thank the departments of Software and Computer Science for providing us with all the possible support in performing this research.

REFERENCES

[1] Suresh, Yeresime, Jayadeep Pati, and Santanu Ku Rath. "Effectiveness of software metrics for object-oriented systems." Procedia technology 6 (2012): 420-427.

[2] Tahir, Amjed, and Stephen G. MacDonell. "A systematic mapping study on dynamic metrics and software quality." 2012 28th IEEE International Conference on Software Maintenance (ICSM). IEEE, 2012.

[3] Deshpande, Mrs Bhagyashri Sunil, Binod Kumar, and Ajay Kumar. "Object Oriented Design Metrics for Software Defect Prediction: An Empirical Study." (2020).

[4] Fenton, Norman, and James Bieman. Software metrics: a rigorous and practical approach. CRC press, 2014.

[5] Goel, Brij Mohan, and Satinder Bal Gupta. "A Comparative Study of Static and Dynamic Object-Oriented Metrics." International Journal of Information Technology & Systems 5.1 (2016).

[6] Kuk, Kristijan, Petar Milić, and Stefan Denić. "Object-oriented software metrics in software code vulnerability analysis." 2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA). IEEE, 2020.

[7] Schnoor, Henning, and Wilhelm Hasselbring. "Comparing static and dynamic weighted software coupling metrics." Computers 9.2 (2020): 24.

[8] Etzel, Christoph, Florian Hofhammer, and Bernhard Bauer. "Towards metrics for analyzing system architectures modeled with EAST-ADL." (2020).

[9] Aggarwal, K. K., et al. "Empirical Study of Object-Oriented Metrics." J. Object Technol. 5.8 (2006): 149-173.

[10] Prasad, Lalji, and Aditi Nagar. "Experimental analysis of different metrics (object-oriented and structural) of software." First International Conference on Computational Intelligence, Communication Systems and Networks. IEEE, 2009.

[11] Ó Cinnéide, Mel, et al. "Experimental assessment of software metrics using automated refactoring." Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement. 2012.

[12] Chong, Chun Yong, and Sai Peck Lee. "Analyzing maintainability and reliability of object-oriented software using weighted complex network." Journal of Systems and Software 110 (2015): 28-53.

[13] Bhardwaj, Mridul, and Ajay Rana. "Key software metrics and its impact on each other for software development projects." ACM SIGSOFT Software Engineering Notes 41.1 (2016): 1-4.

[14] Albert, Réka, and Albert-László Barabási. "Statistical mechanics of complex networks." Reviews of modern physics 74.1 (2002): 47.

[15] Bansiya, Jagdish, and Carl G. Davis. "A hierarchical model for object-oriented design quality assessment." IEEE Transactions on software engineering 28.1 (2002): 4-17.

[16] Genero, Marcela. Defining and validating metrics for conceptual models. Diss. Universidad de Castilla-La Mancha, 2001.

[17] Li, Wei, and Sallie Henry. "Object-oriented metrics that predict maintainability." Journal of systems and software 23.2 (1993): 111-122.

[18] Lorenz, Mark, and Jeff Kidd. "Object-oriented software metrics: a practical guide". Prentice-Hall, Inc., 1994.

[19] Reißing, Ralf. "Towards a model for object-oriented design measurement." 5th International ECOOP workshop on quantitative approaches in object-oriented software engineering. 2001.

[20] e Abreu, F. Brito. "The MOOD metrics set." proc. ECOOP. Vol. 95. 1995.

[21] Chidamber, Shyam R., and Chris F. Kemerer. "A metrics suite for object oriented design." IEEE Transactions on software engineering 20.6 (1994): 476-493.

[22] Li, Wei. "Another metric suite for object-oriented programming." Journal of Systems and Software 44.2 (1998): 155-162.

[23] Briand, Lionel, Prem Devanbu, and Walcelio Melo. "An investigation into coupling measures for C++." Proceedings of the 19th international conference on Software engineering. 1997.

[24] Harrison, Rachel, Steve Counsell, and Reuben Nithi. "Coupling metrics for object-oriented design." Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No. 98TB100262). IEEE, 1998.

[25] Tang, Mei-Huei, Ming-Hung Kao, and Mei-Hwa Chen. "An empirical study on object-oriented metrics." Proceedings sixth international software metrics symposium (Cat. No. PR00403). IEEE, 1999.

[26] Martin, Robert. "OO design quality metrics." An analysis of dependencies 12.1 (1994): 151-170.

[27] Yacoub, Sherif M., Hany H. Ammar, and Tom Robinson. "Dynamic metrics for object-oriented designs." Proceedings Sixth International Software Metrics Symposium (Cat. No. PR00403). IEEE, 1999.

[28] Arisholm, Erik, Lionel C. Briand, and Audun Foyen. "Dynamic coupling measurement for object-oriented software." IEEE Transactions on software engineering 30.8 (2004): 491-506.

[29] Hassoun, Youssef, Roger Johnson, and Steve Counsell. "A dynamic runtime coupling metric for meta-level architectures." Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. IEEE, 2004.

[30] Singh, Paramvir, and Hardeep Singh. "Class-level dynamic coupling metrics for static and dynamic analysis of object-oriented systems." International Journal of Information and Telecommunication Technology 1.1 (2010): 16-28.

[31] Sharble, Robert C., and Samuel S. Cohen. "The object-oriented brewery: a comparison of two object-oriented development methods." ACM SIGSOFT Software Engineering Notes 18.2 (1993): 60-73.

[32] Briand, Lionel, Sandro Morasca, and Victor R. Basili. "Defining and validating high-level design metrics." (1994).

[33] Bieman, James M., and Byung-Kyoo Kang. "Cohesion and reuse in an object-oriented system." ACM SIGSOFT Software Engineering Notes 20.SI (1995): 259-262.

[34] Li, Wei, and Sallie Henry. "Maintenance metrics for the object-oriented paradigm." [1993] Proceedings First International Software Metrics Symposium. IEEE, 1993.

[35] Hitz, Martin, and Behzad Montazeri. Measuring coupling and cohesion in object-oriented systems. na, 1995.

[36] Henderson-Sellers, Brian. Object-oriented metrics: measures of complexity. Prentice-Hall, Inc., 1995.

[37] Yang, X. Research on class cohesion measures. Diss. MS Thesis, Department of Computer Science and Engineering, Southeast University, 2002.

[38] Badri, Linda, and Mourad Badri. "A Proposal of a new class cohesion criterion: an empirical study." Journal of Object Technology 3.4 (2004): 145-159.

[39] Bonja, Challa, and Eyob Kidanmariam. "Metrics for class cohesion and similarity between methods." Proceedings of the 44th annual Southeast regional conference. 2006.

[40] Fernández, Luis, and Rosalía Peña. "A sensitive metric of class cohesion." (2006).

[41] Al Dallal, Jehad. "A design-based cohesion metric for object-oriented classes." International Journal of Computer Science and Engineering 1.3 (2007): 195-200.

[42] Al Dallal, Jehad, and Lionel C. Briand. "A precise method-method interaction-based cohesion metric for object-oriented classes." ACM Transactions on Software Engineering and Methodology (TOSEM) 21.2 (2012): 1-34.

[43] Kim, E. M. An experimental evaluation of OOP complexity metrics: SOMEFOOT. Diss. Master thesis, Chonbuk National University, 1993.

[44] Girvan, Michelle, and Mark EJ Newman. "Community structure in social and biological networks." Proceedings of the national academy of sciences 99.12 (2002): 7821-7826.

[45] Yadav, Sushma, S. Sunil, and S. Uttpal. "A review of object-oriented coupling and cohesion metrics." International Journal of Computer Science Trends and Technology 2.5 (2014): 45-55.

[46] Virdi, Harjot Singh, and Balraj Singh. "Study of the Different Types of Coupling Present in the Software Code." International Journal of computer Science and Information Technology 3.3 (2012): 4153-4156.

[47] Rodriguez, Daniel, and Rachel Harrison. "An overview of object-oriented design metrics." (2001).

Marwah M. A. Dabdawb received her Bachelor degree in Software Engineering in 2008 from the University of Mosul. Her M.Sc. degree was from the same university in 2018. She currently works as a faculty member at Software Department, University of Mosul, Iraq. Her main research interests include Software Engineering and Artificial Intelligence.

Basim Mahmood received his M.Sc. degree in Computer Science from the University of Mosul in 2009. His Ph.D. degree was in 2015 from the college of Engineering at Florida Institute of Technology, USA. He is working as a faculty member at the Computer Science Dept., University of Mosul, Iraq. He also works as a researcher in the BioComplex Laboratory, Exeter, UK. His current research interests include Complex Networks, Big Data Analysis and Data Mining.

APPENDIX: Table showin the detailes of metrics used in our work alongside with their parameters and references.

#	Acronym	Title	Property	Level (System, Package, Class, Object)	State (Static, Dynamic)	References	Parameters
1	DSC	"design size in classes"	Size	S	S	[15]	1- No. of classes in the design.
2	NODP	"The Number of Direct Parts"	Size	C	S	[16]	1-Number of "direct part" of a "whole" class.
3	NP	"The Number of Parts"	Size	C	S	[16]	1-Number of "direct and indirect parts" of a "whole" class.
4	NW	"The Number of wholes"	Size	C	S	[16]	1-The number of "direct or indirect whole" of a "part" class.
5	SIZE1 (classical LOC)	"Size of procedures or functions"	Size	C	S	[17]	1-No of semicolons in a class.
6	SIZE2	"Size of properties defined in a class"	Size	C	S	[17]	1-Attributes in class. 2-Private (local) methods.
7	PM	"Number of Public Methods"	size	C	S	[18]	1-Public methods in a class.
8	NM	"Number of Methods"	size	C	S	[18]	1-Number of methods in a class (private, public and protected).
9	NPV	"Number of Public Variables per class"	size	C	S	[18]	1- Counts the number of public variables in a class.
10	NV	"Number of Variables per class"	size	C	S	[18]	1-Counts the total number of variables in a class.
11	NCP	"Number of all classes in the package"	size	P	S	[19]	1-Number of all classes in the package.
12	NOH	"Number of Hierarchies"	Hierarchies	C	S	[15]	1-No. of ancestor. 2-no. of Descendants.
13	ANA	"Average Number of Ancestors"	Abstraction	S	S	[15]	1-Number of Ancestors. 2-Total Number of Classes.
14	MHF	"Method Hiding Factor"	Encapsulation	C	S	[20]	1-Private(hidden) methods. 2-Total methods (visible+ hidden).
15	AHF	"Attribute Hiding Factor"	Encapsulation	C	S	[20]	1- Private (hidden) attributes. 2- Total attributes (visible+ hidden).
16	DAM	"Data Access Metric"	Encapsulation	C	S	[15]	1-No. of private attributes. 2- No. of attributes.
17	CBO	"Coupling Between Objects"	Coupling	C	S	[21]	1- Instance variable. 2-Class methods.
18	RFC	"Response For a Class"	Coupling	C	S	[21]	1-Class methods.
19	MPC	"Message Passing Coupling"	Coupling	C	S	[17]	1- Message passing.
20	DAC	"Data Abstracting Coupling"	Coupling	C	S	[17]	1-Data type of attributes (instance)= other class.
21	CTA	"Coupling Through Abstract" Data Types	Coupling	C	S	[22]	1-Data type of attributes r (instance)= other class.
22	CTM	"Coupling Through Message Passing"	Coupling	C	S	[22]	1- Message passing in class.
23	CF	"Coupling Factor"	Coupling	S	S	[20]	1- Total no of classes. 2-Client class. 3- Server class.
24	NAssocC	"The Number of Association per Class"	Coupling	C	S	[16]	1- Class associations.
25	DCC	"Direct Class Coupling"	Coupling	S	S	[15]	1-Number of Attribute and Parameter. 2- Total Number of Classes.
26	NDepIn	"The Number of Dependencies In"	Coupling	C	S	[16]	1-Class dependency.
27	NDepOut	"The Number of Dependencies Out"	Coupling	C	S	[16]	1-Class dependency.
28	ACAIC	"A: coupling to ancestor classes. D: Descendants O: other CA: class attributes CM: class method IC: import coupling, the measure counts for a class c all interactions where c is using another class. EC: export coupling, count interactions where class d is the used class"	Coupling	C	S	[23]	1- Ancestor classes. 2- Type of class attributes.
29	OCAIC	"OCAIC"	Coupling	C	S	[23]	1- Class attributes.
30	DCAEC	"DCAEC"	Coupling	C	S	[23]	1- Descendants class. 2-Type of attributes in descendants classes.
31	OCAEC	"OCAEC"	Coupling	C	S	[23]	1- Class attributes.
32	ACMIC	"ACMIC"	Coupling	C	S	[23]	1-Ancestor classes. 2-Type of method parameters in the class.
33	OCMIC	"OCMIC"	Coupling	C	S	[23]	1- Class method.
34	DCMEC	"DCMEC"	Coupling	C	S	[23]	1- Descendants classes. 2- Type of method parameters in Descendants classes.
35	OCMEC	"OCMEC"	Coupling	C	S	[23]	1- Class method.
36	NAS	"Number of Associations"	Coupling	C	S	[24]	1-number of association lines.
37	CBM	"Coupling between methods"	Coupling	C	S	[25]	1-No. of original methods. 2-Inherited method.

							3- Coupling between original and inherited method .through parameters or variables etc.
38	Ca	"Afferent Coupling (incoming coupling)"	Coupling	P	S	[26]	1- no of classes. 2-class dependency.
39	Ce	"Efferent Coupling (outgoing coupling)"	Coupling	P	S	[26]	1- No. of classes. 2-Class dependency.
40	EOC	"Export Object Coupling"	Coupling	O	D	[27]	1- No. of message exchange between two obj.
41	IOC	"Import Object Coupling"	Coupling	O	D	[27]	1- No. of message exchange between two obj.
42	IC_OD	"IC: import coupling EC: export coupling O: object C: class C = counts the number of distinct classes that a method in a given class or object uses or is used by. M = counts the number of distinct methods invoked by each method in each class or object. D = counts the total number of dynamic messages sent or received from one class/object to or from other classes or objects"	Coupling	O	D	[28]	1- No. of messages between obj.
43	IC_OM	"IC_OM"	Coupling	O	D	[28]	1-Object method.
44	IC_OC	"IC_OC"	Coupling	O	D	[28]	1-No. Of server classes. 2-Obj. method.
45	IC_CM	"IC_CM"	Coupling	C	D	[28]	1- Methods in obj.
46	IC_CD	"IC_CD"	Coupling	C	D	[28]	1- Total no of messages. 2- Methods in obj.
47	IC_CC	"IC_CC"	Coupling	C	D	[28]	1- No. of server classes. 2- Methods in obj.
48	EC_OM	"EC_OM"	Coupling	O	D	[28]	1- Distinct method.
49	EC_OD	"EC_OD"	Coupling	O	D	[28]	1- Messages between obj.
50	EC_OC	"EC_OC"	Coupling	O	D	[28]	1- No of client classes.
51	EC_CD	"EC_CD"	Coupling	C	D	[28]	1-Total no of messages. 2- Methods in obj.
52	EC_CM	"EC_CM"	Coupling	C	D	[28]	1- Methods in obj.
53	EC_CC	"EC_CC"	Coupling	C	D	[28]	1-Total no of client classes.
54	DCM	"Dynamic Coupling Metric"	Coupling	C	D	[29]	1- Program execution steps. 2-No. of obj.
55	DCa	"Dynamic Afferent Coupling	Coupling	C	D	[30]	1- No. of classes accessing the methods of a class. 2- Total number of classes.
56	DKSC	"Dynamic Key Server Class"	Coupling	C	D	[30]	1- Number of calls sent to a class.
57	DKCC	"Dynamic Key Client Class"	Coupling	C	D	[30]	1-Number of calls sent by a class.
58	DKC	"Dynamic Key Class"	Coupling	C	D	[30]	1- Total number of static calls sent and received by all the classes.
59	PAC	"Percentage Active Classes"	Coupling	C	D	[30]	1- Number of classes sending or receiving at least one method calls. 2- Total number of classes.
60	VOD	"Violation of the law of Demeter"	Coupling	C	S	[31]	1-Class method.
61	IC	"Inheritance Coupling (IC)"	Coupling	C	S	[25]	1-No of parent classes. 2- Class original methods. 3- Class inherited method. 4-Coupling between original and inherited method .through parameters or variables etc.
62	LCOM1	"Lack of Cohesion of Methods"	Cohesion	C	S	[21]	1- Class methods. 2- Class attributes.
63	LCOM2	"Lack of Cohesion of Methods"	Cohesion	C	S	[21]	1- Class methods. 2- Class attributes.
64	CAM	"Cohesion Among Methods of Class"	Cohesion	C	S	[15]	1- Method's parameters types.
65	RCI	"Ratio of Cohesive Interactions"	Cohesion	C	S	[32]	1- Set of cohesion interaction in a module.
66	NRCI	"Neutral Ratio of Cohesive Interactions"	Cohesion	C	S	[32]	1- Set of cohesion interaction in a module.
67	PRCI	"Pessimistic Ratio of Cohesive Interaction"	Cohesion	C	S	[32]	1- Set of cohesion interaction in a module.
68	ORCI	"Optimistic Ratio of Cohesive Interactions"	Cohesion	C	S	[32]	1- Set of cohesion interaction in a module.
69	TCC	"Tight Class Cohesion"	Cohesion	C	S	[33]	1- Public methods of the class. 2- Attributes.
70	LCC	"Loose Class Cohesion"	Cohesion	C	S	[33]	1-Class methods. 2-Attributes.
71	LCOM3	"Lack of Cohesion of Methods"	Cohesion	C	S	[34]	1- Class methods. 2- Class attributes.
72	LCOM4	"Lack of Cohesion of Methods"	Cohesion	C	S	[35]	1- Class methods. 2-Class attributes.

73	LCOM5	"Lack of Cohesion of Methods"	Cohesion	C	S	[36]	1-Class attributes. 2- Class methods.
74	OLn	"OLn"	Cohesion	C	S	[37]	1- Class methods. 2- Class attributes.
75	DCd	"Degree of Direct Cohesion"	Cohesion		S	[38]	1-Methods of the class. 2- Class attribute..
76	DCi	"Degree of Indirect Cohesion"	Cohesion		S	[38]	1-Methods of the class. 2- Attributes.
77	CC	"Class Cohesion"	Cohesion	C	S	[39]	1- Methods of the class. 2- Class attribute.
78	SCOM	"Class Cohesion metric"	Cohesion	C	S	[40]	1- Methods of the class. 2- Class attribute.
79	PCCC	"Path Connectivity Class Cohesion"	Cohesion	C	S	[41]	1- No. of attributes. 2-No. of methods. 3-Number of simple paths in graph.
80	MMAC	"Method-Method through Attributes Cohesion"	Cohesion	C	S	[42]	1- No. of attributes. 2- No. of methods.
81	LCOM	"Lack of cohesion Metric"	Cohesion	C	D	[21]	1- Methods. 2-Instance variables.
82	MOA	"Measure of Aggregation"	Composition	C	S	[15]	1- Attributes in class.
83	DIT	"Depth of Inheritance Tree"	Inheritance	C	S	[21]	1- Number of ancestor classes.
84	NOC	"Number of Children"	Inheritance	C	S	[21]	1- Class direct children.
85	NAC	"Number of Ancestor Classes"	Inheritance	C	S	[22]	1- Number of ancestor classes.
86	NDC	"Number of Descendant Classes all subclasses"	Inheritance	C	S	[22]	1- Descendant classes.
87	MIF	"Method Inheritance Factor"	Inheritance	C	S	[20]	1- Inherited methods. 2- Available methods (new method+ inherited method-overriding method).
88	AIF	"Attribute Inheritance Factor"	Inheritance	C	S	[20]	1- Inherited attributes. 2- Available attributes.
89	MFA	"Measure of Functional Abstraction"	Inheritance	C	S	[15]	1- Inherited methods. 2- All methods (origin and inherited) for a class.
90	NMI	"Number of Methods Inherited by a subclass"	Inheritance	C	S	[18]	1-Number of methods inherited by a subclass.
91	NMO	"Number of Methods Overridden by a subclass"	Inheritance	C	S	[18]	1- Total number of methods overridden by a subclass.
92	NMA	"Number of Methods Added by a subclass"	Inheritance	C	S	[18]	1- Total number of methods defined in a subclass.
93	PF	"Polymorphism Factor"	Polymorphism	S	S	[20]	1-Overriding methods in class. 2-New methods in class. 3- Number of descendants of class. 4- Total no of classes.
94	NOP	"Number of Polymorphic Methods"	Polymorphism	C	S	[15]	1-Overriding method. 2-Overloading method.
95	CIS	"Class Interface Size"	Messaging	C	S	[15]	1- Public methods.
96	WMC	"Weighted Method Per Class"	Complexity	C	S	[21]	1- Method complexity.
97	NLM	"Number of Local Methods"	Complexity	C	S	[22]	1- No. of local methods.
98	NOM	"Number of Methods"	Complexity	C	S	[15], [17]	1- No. of methods.
99	CMC	"Class Methods Complexity"	Complexity	C	S	[22]	1-Method complexity.
100	WAC	"Weighted Attribute per Class"	Complexity	C	S	[31]	1-Weighted attributes Of class.
101	SC	"Static Complexity"	Complexity	S	S	[43]	1- Methods complexity in class. 2- All classes.
102	DC	"Dynamic Complexity"	Complexity	C	D	[43]	1- Methods reused.
103	AMC	"Average method complexity"	Complexity	C	S	[25]	1-No of methods. 2-Methods complexity.
104	NOT	"Number of Tramps"	Complexity	C	S	[31]	1- Parameters in method.