Estimating the Quality of an Object-Oriented Software System Using Graph Algorithms

N. Khanahmadliravi and H. R. Khataee

***Abstract*—This paper proposes a new set of structural software metrics to estimate the quality of an object-oriented software system using graph algorithms. In the proposed technique, firstly, the class diagram of the ultimate system is mapped into its graph-based structural model. In the developed graph-based structural model the weights of edges are the dependence degrees of relationships among the classes of the class diagram. Then, the graph algorithms are applied based on the longest path algorithm to propose a new set of structural software metrics. The proposed structural software metrics can estimate the quality of the ultimate software system based on the possible maximum relational complexities between the classes of its class diagram. Our proposed structural software metrics can detect some classes which have the most complex relationships with other classes in the class diagram. The detected classes may need more time and effort to maintain in the next phases of the software development compared to other classes.**

***Index Terms*—Class diagram, graph algorithm, software metric, software quality.**

I. Introduction

Classes are the foundation of the object-oriented software design and development. Unified Modeling Language (UML) class diagram is a standard modeling language for describing, visualizing, and documenting of the employed classes and relationships among them in an object-oriented software system before development of the system. In this way, with measuring the quality of a class diagram in the early stages of the software system development, many unnecessary revisions at later stages will be deleted [1]. The quality of a class diagram is measured using software metrics which map a software development domain to a numerical domain, usually the real numbers [2]. It means that with measuring the complexity of a class diagram using software metrics, we can estimate the quality of its ultimate software system in the early stages of the development. Most of the previously reported works on the complexity of class diagrams have not strictly focused on the relationships among classes, especially indirect relationships [3]-[6]. Therefore, in this paper, we apply graph algorithms and introduce a methodology to propose a new set of structural software metrics to measure the complexity of a class diagram with regarding all the direct and indirect relationships among its classes. In the proposed methodology, firstly, we develop a graph-based structural model of a class diagram based on the different relationships among its classes. Then, we employ this graph-based structural model of the class diagram and graph algorithms to propose a new set of structural software metrics. The proposed structural software metrics can estimate the quality of an ultimate software system based on all types of direct/indirect relationships among its classes more objectively.

This paper has been organized as follow: the next section represents the relational concepts of a class diagram. Then, in the third section, our proposed structural software metrics of UML class diagram have been presented. The fourth section presents the elaboration of the proposed structural software metrics through an example. Finally, the last section concludes the paper.

Manuscript received June 15, 2012; revised July 24, 2012.

N. Khanahmadliravi is with the Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia (e-mail: nasim_khanahmad@yahoo.com).

H. R. Khataee is with the Research Group of Intelligent Computing, Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia (e-mail: hr_khataee@yahoo.com).

II. UML Class Diagram: Structural Concepts

There are six types of primary relationships among classes of a class diagram including association, aggregation, composition, generalization, dependency, and realization [7] (Fig. 1). One of the main structural features of a class diagram is the dependence degree of its classes which characterizes the semantics of relationships among the classes [8]. The dependence degree indicates the level of dependence among the classes which have relationship. The six types of primary relationships among classes in a class diagram define different values of dependence degrees. The smallest dependence degree belongs to the dependency relationship as it is a kind of association relationship that does not need to be instantiated. The association is a more general relationship, while does not explicitly describe the concrete relationship. Therefore, dependence degree of an association relationship should not be less than that of a dependency relationship.

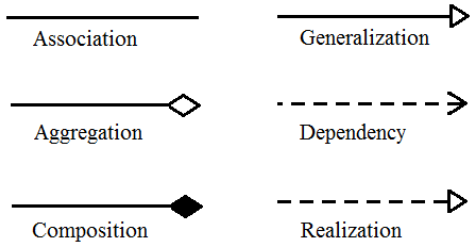

Fig. 1. A schematic of six kinds of primary relationships among classes of a class diagram.

As the aggregation relationship indicates the whole-part relationship, its dependence degree is more than that of the dependency relationship. Accordingly, dependence degree of a composition relationship is more than that of an aggregation relationship. In a composition relationship, a super-class can access only the public interface of its sub-class, while in a generalization relationship a sub-class can define new attributes and methods in addition to the attributes and methods which can inherit from super-class. Hence, the dependence degree of a generalization relationship should be bigger than that of a composition relationship. Finally, as the realization relationship defines the most concrete relationship among classes, its dependence degree is the biggest [7]. In general, let r_i with $1 \leq i \leq 6$ be dependency, association, aggregation, composition, generalization, and realization relationships respectively. The comparison of the dependence degrees of these six kinds of primary relationships among classes of a class diagram is as follows [7]:

$$d(r_1) < d(r_3) < d(r_4) < d(r_5) < d(r_6) \quad (1)$$

$$d(r_1) \leq d(r_2) \leq d(r_4) \quad (2)$$

where $d(r_i)$ denotes the dependence degree of r_i $(1 \leq i \leq 6)$. Therefore, it can be concluded that the structural complexity of a class diagram relies on the different dependence degrees of the relationships among its classes and the distribution of these relationships.

TABLE I: DEFINITIONS OF ADJACENCY, DISTANCE AND SHORTEST DISTANCE MATRIXES OF A RAPH *G* WHIT SUPPOSING $|V(G)| = n$.

Matrix	Definition
Adjacency $A\,[a_{ij}]_{n\times n}$	If (an edge joins vertices *i* and *j*) then $a_{ij} = 1$, zero otherwise. Obviously, if (*i* equals to *j*) then $a_{ij} = 0$.
Distance $D\,[d_{ij}]_{n\times n}$	If (a_{ij} equals to 1) then d_{ij} = dependence degree of the relationship between classes *i* and *j*, otherwise $d_{ij} = 0$. Obviously, if (*i* equals to *j*) then $d_{ij} = 0$.
Longest distance $L\,[l_{ij}]_{n\times n}$	l_{ij} = the longest path between vertices *i* and *j*. Obviously, if (*i* equals to *j*) then $l_{ij} = 0$.
Each of the a_{ij}, d_{ij}, l_{ij} values is related to the value on line *i* and column *j* of their respective matrixes that $1 \leq i,\ j \leq n$.	

III. STRUCTURAL SOFTWARE METRICS

In this section, we have used graph algorithms and proposed a new set of structural software metrics to estimate the quality of an ultimate software system based on all types of direct/indirect relationships among its class diagram's classes. In order to apply graph algorithms to propose our objective structural software metrics, firstly, we map the class diagram into its graph-based structural model where the nodes and edges of the graph are the classes and relationships among classes of the class diagram respectively. The weights of the edges among the nodes are the dependence degrees of the relationships among the respective classes. Let *G* be a connected, directed, acyclic, and weighted respective graph of a class diagram with vertex set *V (G)* and edge set *E (G)*. We use $|V(G)| = n$ and $|E(G)| = e$ to denote the cardinality of *V (G)* and *E (G)* respectively. Our applied graph algorithms employ:

- Adjacency matrix,
- Distance matrix,
- Longest distance matrix,

To propose a new set of structural software metrics. Table I illustrates the definitions of these matrixes.

The first employed graph algorithm is the critical path method [9] which is used to compute the longest paths between all pairs of vertices in G with the following pseudo-code:

```
Algorithm 1.
Input: A [aij]n ×n, D [dij]n ×n.
Output: L [lij]n ×n.
  begin
    for i = 1 to n do
     for j = 1 to n do
      replace the longest path between
      vertices i and j to lij;
  end
```

As the longest path algorithm has been solved for acyclic graphs with nonnegative weights [9, 10], in this paper we consider the class diagrams that their respective graph-based structural models can be mapped to a directed, acyclic, and weighted graphs. Based on the output of the "Algorithm 1", $L\,[l_{ij}]_{n\times n}$, our proposed software metrics are defined.

Metric 1: Maximum Relational Complexity (MaxRC) between classes *u* and *v* can be defined as:

$$MaxRC(u, v) = l_{uv} \quad (3)$$

where l_{uv} is the value of the $L\,[l_{ij}]_{n\times n}$. The MaxRC indicates a relationship (direct/indirect) with the maximum dependence degree between two classes. Therefore, MaxRC shows the maximum possible relational complexity between two certain classes of a class diagram.

Metric 2: Maximum Relational Outgoing Complexity (MaxROC) of a class v can be defined as:

$$MaxROC(v) = \sum_{\substack{i=1,\\ 1\leq v\leq n}}^{n} l_{vi} \quad (4)$$

where $v \in V(G)$ in the graph-based structural model of the class diagram. The MaxROC (*v*) equals to the sum of the dependence degrees of outgoing relationships (direct/indirect) from a class *v* to all other classes with maximum complexities. According to (4), the following algorithm is used to compute the MaxROC (v):

```
Algorithm 2.
Input: L [lij]n ×n.
Output: MaxROC (v).
  begin
    MaxROC (v) = 0.0;
```

for i = 1 to n do
add lvi into MaxROC (v);
end

Metric 3: Maximum Relational Incoming Complexity (MaxRIC) of a class *v* defines the sum of the dependence degrees of incoming relationships (direct/indirect) from all other classes to a class *v* with maximum complexities. The MaxRIC of a class *v* can be defined as:

$$\text{MaxRIC}(v) = \sum_{\substack{i=1,\\ 1 \le v \le n}}^{n} l_{iv} \quad (5)$$

where $v \in V(G)$ in the graph-based structural model of the class diagram. In order to compute the MaxRIC (*v*) the following algorithm is used according to the proposed (5):

Algorithm 3.
Input: L [lij]n ×n.
Output: MaxRIC (v).
begin
MaxRIC (v) = 0.0;
for i = 1 to n do
add liv into MaxRIC (v);
end

According to the definitions of our proposed structural software metrics, a general view point of these software metrics can be explained. Then in next section, we elaborate these view points using a simple example. The proposed first software metric can measure the maximum relational complexity between two classes of a class diagram with regarding all types of direct/indirect relationships between them. In this way, the software metric 1 can estimate the maximum relational complexities among the classes of an ultimate software system more objectively.

Software metrics 2 and 3 can measure the sum of the maximum relational complexities of a class with all other classes of a class diagram. These two software metrics can reveal and predict the bottleneck classes of an ultimate software system which have the most complex relationships with other classes. This is due to that the biggest value of all MaxROCs in a class diagram indicates a class that has the biggest relational outgoing complexity. Thus, this class needs more time and effort for testing and maintenance compared to other classes. In this way, this class can be a bottleneck class in the ultimate software system as it has the most complex transactions with other classes. Similarly, this interpretation can be described for the biggest value of all MaxRICs in a class diagram. Next section demonstrates the above mentioned view points through an example.

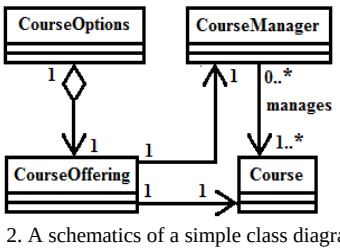

Fig. 2. A schematics of a simple class diagram.

IV. Elaboration of Structural Software Metrics

In this section, we elaborate the proposed methodology and structural software metrics of this work through an example class diagram depicted in Fig 2.

According to the proposed methodology in previous section, the class diagram of Fig. 2 is mapped to its graph-based structural model (see Fig. 3). In this example, according to (1) and (2), it is assumed that $d(r_1) = 1$, $d(r_2) = 2$, $d(r_3) = 3$, $d(r_4) = 4$, $d(r_5) = 5$, and $d(r_6) = 6$ as well as the "CourseOptions", "CourseManager", "CourseOffering", and "Course" classes are equivalent to the nodes 1 to 4 respectively. Thus, the example class diagram of Fig. 2 is mapped to its respective graph-based structural model shown in Fig. 3.

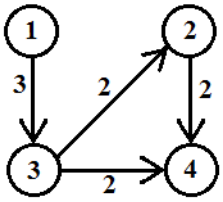

Fig. 3. The graph-based structural model of the class diagram.

Based on the developed graph-based structural model of the class diagram in Fig. 3, the distance and adjacency matrixes are computed and depicted in Fig. 4 (a, b).

$$A[a_{ij}]_{4*4} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(a)

$$D[d_{ij}]_{4*4} = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(b)

Fig. 4. The adjacency (a) and distance (b) matrixes of the graph-based structural model of the example class diagram.

$$L[l_{ij}]_{4*4} = \begin{bmatrix} 0 & 5 & 3 & 7 \\ 0 & 0 & 0 & 2 \\ 0 & 2 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 5. The longest distance matrix of the graph-based structural model of the example class diagram.

Then, we apply the "Algorithm 1" to compute the $L[l_{ij}]_{4\times4}$ as illustrated in Fig. 5. According to the computed $L[l_{ij}]_{4\times4}$ of example class diagram, the outputs of our proposed software metrics for this example are summarized in Tables II, III. Tables II and III can clearly describe the view points of our proposed software metrics for this example. Table II illustrates the possible maximum relational complexities (i.e. metric 1) between all classes of the example class diagram. In Table I, the proposed software metric 1 demonstrates that the possible maximum relational complexity between classes "CourseOptions" and "Course" (nodes 1 and 4) can be 7, MaxRC (1, 4) = 7. This indicates that in the ultimate software system the relationship between classes "CourseOptions" and "Course" through classes "CourseOffering" and

"CourseManager" can be more complex than of their relationship through class "CourseOffering". Similarly, Table I indicates that the possible maximum relational complexity between "CourseOffering" and "Course" (nodes 3 and 4) can be 4, MaxRC (3, 4) = 4. This means that the relationship between classes "CourseOffering" and "Course" through class "CourseManager" can be more complex than of their direct relationship.

On the other hand, Table III shows the proposed MaxROC (v) and MaxRIC (v) metrics where $1 \leq v \leq 4$ which denotes the classes of the class diagram. Table III indicates that the software metrics 2 and 3 measure MaxROC (3) = 6 and MaxRIC (3) = 3. These results represent that the sum of the maximum outgoing and incoming relational complexities of class "CourseOffering" with all other three classes can be 6 and 3 respectively. Moreover, as the biggest value of all MaxROCs is MaxROC (1) = 15, it can be detected that the class "CourseOptions" may be a bottleneck class in the ultimate software system. It means that the class "CourseOptions" has the most complex relationships with other classes. In this way, the class "CourseOptions" needs more time and effort to maintain in the next phases of the development. Another bottleneck class of the ultimate software system can be class "Course". This is due to that the MaxRIC (4) = 13 is the biggest value of the all MaxRICs.

TABLE II: THE COMPUTED VALUES OF METRIC 1.

MaxRC (u, v)	**v = 1**	**v = 2**	**v = 3**	**v = 4**
u = 1	0	5	3	7
u = 2	0	0	0	2
u = 3	0	2	0	4
u = 4	0	0	0	0

TABLE III: THE COMPUTED VALUES OF METRICS 2 AMD 3.

	v = 1	**v = 2**	**v = 3**	**v = 4**
MaxROC (v)	15	2	6	0
MaxRIC (v)	0	7	3	13

V. CONCLUSIONS

In this paper, we have proposed a new set of structural software metrics to estimate the quality of an ultimate software system using graph algorithms. We have applied graph algorithms to the graph-based structural model of a class diagram and introduced a set of structural software metrics including MaxRC (u, v), MaxROC (v), and MaxRIC (v), where v, w V (G). The advantages of the proposed new structural software metrics were as follows: (1) the metrics considered all direct and indirect relational complexities among classes, (2) the metrics highlighted some classes which were needed to be considered more in maintenance phase, and (3) the metrics realized the bottleneck classes of the ultimate software system.

REFERENCES

[1] V. Krishnapriya and K. Ramar, "Comparison of Class Inheritance and Interface Usage in Object Oriented Programming through Complexity Measures," *International Journal of Computer Science & Information Technology*, vol. 2, pp. 28-36, 2010.

[2] M. Marchesi, "OOA metrics for the United Modeling Languages," in: *Proc. 2nd* Euromicro *Conference on Software Maintenance and Reengineering*, Palazzo degli Affari, 1998, pp. 67-73.

[3] S. R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design*," IEEE. T. Software. Eng*, vol. 20, pp. 476-493, 1994.

[4] M. Genero, M. Piattini, and C. Calero, "Early Measures For UML class diagrams," *L'Objet., Hermes Science Publications*, vol. 6, pp. 489-515, 2000.

[5] D. Berardi, D. Calvanese, and G.D. Giacomo, "Reasoning on uml class diagrams," *Artif. Intell*, vol. 168, pp. 70-118, 2005.

[6] F. S. C. Tseng and C. Chen, "Enriching the class diagram concepts to capture natural language semantics for database access," *Data. Knowl. Eng.*, vol. 67, pp. 1-29, 2008.

[7] Y. Zhou and B. Xu, "Measuring structural complexity for Class Diagrams: An Information Theory Approach," in *Proc. the 2005 ACM symposium on Applied computing*, New York, 2005, pp. 1679-1683.

[8] R. M. Pitrik and J. Kaasboll, "Part-Whole Relationship Categories and Their Application in Object-Oriented Analysis," *IEEE. T. Knowl. Data. En*, vol. 11, pp. 779-797, 1999.

[9] E. Horowitz, S. Sahni and D. Mehta, *Fundamentals of Data Structures in C++*, Computer Science Press, USA, 1995, pp. 320-400.

[10] Y. Chen, D. Rinks, and K. Tang, "Critical path in an activity network with time constraints," *Eur. J. Oper. Res*., vol. 100, pp. 122-133, 1997.