International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)
Volume 2, Issue 3, March 2015, PP 59-62
ISSN 2349-4840 (Print) & ISSN 2349-4859 (Online)
www.arcjournals.org

New Metrics for System Understandability of Inheritance Hierarchies

Mr. D.N.V.Syma Kumar[1], Dr. R.Satya Prasad[2]

Research Scholar of Krishna University, Machilipatnam, A.P., India
Assoc. Professor, Dept.ofCSE, AcharyaNagarjuna University, Guntur, A.P., India
Syam.researchscholar@gmail.com[1], profresp@gmail.com[2]

Abstract: *Understandability is a key concept in every system maintenance on the design phase. In the classification of the Inheritance multiple inheritance hierarchy is a typical task to identify the class understandability and system understandability. Some of the understandability metrics were already existed but they are giving the large complexity values while measuring a system or a class. In this research paper we proposed new metrics for class and system understandabilities. These metrics poses the lowest complexity values when comparing with existed metrics. This helps the designers of the system in easy way of understanding the system and get better designs of the systems in future.*

Keywords: *software metrics, object-oriented, inheritance hierarchy, DAG, system understandability, system maintenance, weyker's properties.*

1. INTRODUCTION

Inheritance is a good object-oriented mechanism which can use the features of parent class into child class. In this inheritance concept reuse technique will be utilized. Inheritance hierarchy is a class level structure which can be utilized to understand the given object oriented program in effective manner. Previous studies on the Inheritance through classes reduce the redundancy and system maintenance necessary that increases the efficiency of the system [3, 17, 19, 20, 21, 22]. Inheritance is classified into several types i.e., single, Multiple, Multi-level … etc. Understanding of the inheritance hierarchy in the view of system is a typical task because increasing the number of class levels leads to typical design of the system. So, evaluation of the inheritance hierarchy is needed.

Object oriented metrics were used in evaluation of the inheritance hierarchy of the system. Basically software metrics are surveyed and classified into several types for software quality [1,2].Object oriented metrics were applied in various programming languages and fields [18]. There are several well known object – oriented inheritance metrics were existed [3, 4, 5, 6, 7, 8, 24]. In this inheritance metrics used attributes are number of classes, number of methods inherited or overridden and depth of inheritance hierarchy ...etc.

Every software metric has to show the theoretical and mathematical foundation behind the metric to evaluate the program. To measure the complexity metrics Weyker [9] proposed set of properties which have to be satisfied to give the good metrics. Several researchers evaluated their proposed metrics [3, 4, 10, 11, 12, 13, 23, 25] with the help of Weyker's proposed properties. Most of the properties were satisfied by the well known inheritance metrics as DIT, NOC, NAC, NDC and AID. Some of the weyker's properties were not satisfied by these metrics [23] because those properties were worked on the traditional programming. Here, in the most of the object oriented metrics mainly focuses only on class not the inside information of the class.

In this paper we proposed two more metrics namely Average Class Understandability (ACU) and Average System Understandability (ASU). ACU and ASU are utilized to find the complexity values on class wise and system wise. Our main concentration is to reduce the understandability as much as possible for the better understanding of the given system design.

The frame of the proposed paper as follows: Section 2 presents the related work in the field of inheritance hierarchy metrics and also the techniques like DIT, NOC, NAC, NDC, AID and AU. Section 3 focuses on our proposed metrics ACU and ASU. In Section 4 Weyker's properties are evaluated with our proposed metrics. Section-5 deals with the Comparison with other existed inheritance metrics. In Section-6 Reduced complexity values of the understandability are discussed. Section 7 and Section 8 were focused on the conclusion and future scope of the paper respectively.

2. RELATED WORK

In this section we stress the importance of understandability metrics along with other well known inheritance metrics like DIT, NOC, NAC, NDC, AID and AU.

Depth of Inheritance Tree (DIT) suggested by chidamber-kerner [14, 15], DIT can be measured as the depth in the hierarchy. Later it was modified as maximum distance from node to the root of the tree [3]. DIT shows the ambiguity problem in inheritance hierarchy. W.Li [6] introduces Number of Ancestor Classes (NAC) is a solution for the ambiguity problem faced in the DIT mechanism. NAC measures the total number of ancestor classes inherited by a class in the inheritance hierarchy.

Consider the breadth of the inheritance hierarchy rather than depth, chidamber-kerner [14] introduced new mechanism called Number Of Childs (NOC). It was modified as number of immediate sub classes in the class hierarchy [3]. By NOC only immediate sub classes have to be taken into account but not all the sub classes which are influenced by the specified class. To solve this problem W.Li [6] introduced Number of Descendant Classes (NDC) of a class. It states that the total number of subclasses of that class. Henderson-Sellers [5] introduced a new metric called Average Depth of Inheritance (AID) state that ratio between the sum of individual depths of the classes to the number of classes. This AID metric takes more time for simplified systems comparing with other systems.

For evaluating the system understandability Sheldon-Jerath [8] proposed a new metric called Average Understandability [AU].This AU mainly focused on the predecessors of the given class. The resultant AU metric gives the largest complexity values. This metric takes more time to understand and design the system. Hence we need new metrics in this area to reduce the understandability complexity value for better design and understandability of the given system.

3. PROPOSED INHERITANCE METRICS

In this paper we propose two metrics ACU and ASU for measuring the class and system understandability purposes. ACU stands for Average Class Understandability and ASU stands for Average System Understandability. We are taken the class diagrams as the Directed Acyclic Graph (DAG) with no loops [16] in the situation of the multiple inheritance hierarchies.

In these metrics we used the average complexity method. For good results rather than best and worst complexities. Average complexity values can be used in various software features.ACU metric states that understandability of the class includes the super classes which are inherited in the present class and sub classes which are inherited the present class.ASU is the average of the total classes individual ACUs with number of the classes in the system.

Average Class Understandability is

$ACU = (_{i\,/}\, n_1) + (_{i\,/}\, n_2)$

Sup_i=Depth from super class i.

n1= Number of super classes.

Sub_i=Height from sub class i.

n2= Number of sub classes.

Average System Understandability is

$ASU = {}_{i\,/}\, n$

ACU_i=Average Class Understandability of Class i.

n= Number of classes.

It is the easiest way to find the understandability of the given inheritance hierarchy and understandability complexity value is low when compared to existing metrics.

4. PROPOSED METRICS EVALUATION WITH WEYUKER'S PROPERTIES

The statistical measurement of the every software metric can be evaluated by the Weyker's properties [9]. The Weyker's properties state that the measuring software metric is how much effective. It is very important measure for every software metric, even some of the researchers are criticizing the Weyker's properties. Our proposed object oriented metrics were mainly focused on the class only, not the inside information of the given class. Hence most of the Weyker's properties which can be observed on software metrics obtained good results, but some of the properties (7, 9) which cannot work on the object oriented metrics properly [23].

Property-1:-Non-Coarseness-

Suppose our proposed metric M, work on two different classes A and B always the Non-Coarseness found that $M(A) \neq M(B)$.

Property-2:- Granularity –

This property states that there are a finite number of cases of the program having the finite metric values. The metric value must be non-negative value.

Property-3:- Non-Uniqueness-

For two different classes A and B, proposed metric values must be same as M (A) =M (B).

Property-4:- Design Implementation –

The class utilized in the metric has to present the two different metric values designed by the two different designers.

Property-5:- Monotonicity –

This property states that two different classes A, B combination metric (A+B) is greater than or equal to the individual classes metrics values. It mean M (A+B) ≥M (A) and M (A+B) ≥M (B). A+B means combination of two different classes A and B. The combination of two classes may lead to the possible cases in the object oriented design.

Every metric has to follow three possible cases to satisfy the Monotonicity property.

1. If class A and class B are siblings.
2. If one class is the child of another class.
3. If class A and class B are neither siblings nor children of each other.

Property-6:- Non-Equivalence of Interaction-

If two classes A and B having same metric values, then combine the third class C with the existing A and B classes. The result of A and C not equal to B and C.

Property-7:- Significance of Permutation-

If program A and B such that B is formed with the A's permuting order of statements then evaluated both metric values need not be equal.

Property-8:- No change of remaining

If class A is rename with class B then the metric value not to be changed.

Property-9:-Interaction Complexity-

This property sates that interacting of two classes A and B will be greater than the sum of class A and class B.

M(A)+M(B)< M(A+B)

This ninth property is not satisfied by any object oriented design [23] with classes because with the combination of two classes leads to the highest value. This gives bad design of the system. So, well-known techniques of the inheritance hierarchy (DIT, NOC, NDC, NAC, AID) also not followed the weyker's 9thproperty.Hence our proposed metrics ACU and ASU also not satisfied by the weyker's 9th property.

5. COMPARISON WITH OTHER INHERITANCE METRICS

Here our metrics are compare to well known metrics like DIT, NOC, NAC, NDC, AID and AU in the view of the Weyker's properties. Previously existing results for DIT, NOC, NAC, NDC and AID were taken with the evaluation of weyker's properties. Now we have done the evaluation of AU metric with respect of weyker's properties. Our metrics got the good results rather than the previous results. Some of the properties were not supported by our metrics also, because those will be applicable for only traditional programming purpose only not for the object oriented purpose.

Results of the metrics against weyker's properties are give in the following table1.

Table1. *Measurement of Inheritance Metrics in view of Weyker's properties*.
√ - *weyker's property satisfied by the metric.*
× - *weyker's property not satisfied by the metric.*

Property	DIT	NOC	NAC	NDC	AID	AU	ACU	ASU
1	√	√	√	√	√	√	√	√
2	√	√	√	√	√	√	√	√
3	√	√	√	√	√	√	√	√
4	√	√	√	√	√	√	√	√
5	×	√	×	×	×	×	√	√
6	√	√	√	√	√	√	√	√
7	×	×	×	×	×	×	×	×
8	√	√	√	√	√	√	√	√
9	×	×	×	×	×	×	×	×

Here some of the properties should not be satisfied by the inheritance metrics [23] because the inheritance hierarchy metrics mostly focus on the system outer view not the inside details of the system.

In the results discussion we mainly comparing with our metrics ASU with the existing AU metric. In this comparison our metrics got the lowest complexity values rather than the others. Hence by using our metrics the designers can easily understand the design of the system within optimum time.

6. CONCLUSION

In this paper we discussed so many useful and well known inheritance metrics and especially we compared our proposed metrics ACU and ASU with the existing metric AU. Our proposed metrics obtain lowest understandability complexity values rather than AU. Here we tested our metric with the well-knownweyker's properties. Most of the weyker's properties are satisfied by our proposed metrics. Some of the properties which were mainly worked on traditional programming and inside data of the system those were not supported by our proposed metrics because our metrics were not focused on the inside of the class. By using our metrics we reduced the understandability complexity values drastically. This will help the designers for understanding the system design in easiest manner and design the system in effective manner.

7. FUTURE SCOPE

With the reduced results of our proposed metrics we identified that the designers overhead must be reduced up to some extent .In future we want to focus on the inside data of the classes and use the class information in the understandability metrics. We want to develop an innovative model for the understandability by considering the data and methods in the class. This may be utilized effectively in the system maintenance and observe the behavior of the system with the data and methods of the classes in the inheritance hierarchy.

REFERENCES

[1]. AmjanShaik, C. R. K. Reddy, BalaManda ,Prakashini.

[2]. C,Deepthi. KMetrics for Object Oriented Design Software Systems: A Survey Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS) 2010, 1 (2): 190-198.

[3]. M.S.Ranwat,A.Mittal,S.K.Dubey Survey on impact of software metrics on software quality (IJACSA)International journal of Advanced Computer Science and Applications, Vol.3,No.1,2012.

[4]. Chidamber, S.R.—Kemerer, C.F.: A Metrics Suite for Object Oriented Design.IEEE Transactions on Software Engineering, Vol. 20, 1994, No. 6, pp. 476–493.

[5]. Abreu, F.B.—Carapuca, R.: Candidate Metrics for Object Oriented Software within a Taxonomy Framework. Journal of System Software, Vol. 26, 1994, pp.87–96.

[6]. Henderson-Sellers, B.: Object Oriented Metrics: Measures of Complexity. Pren-tice Hall PTR: Englewood Cliffs, NJ, 1996; pp. 130–132.

[7]. Li, W.: Another Metric Suite for Object-Oriented Programming. Journal of Systems and Software, Vol. 44, 1998, pp. 155–162.

[8]. Lorenz, M.—Kidd, J.: Object-Oriented Software Metrics. Prentice Hall 1994, ISBN: 013179292X.

[9]. Sheldon, F.T.—Jerath, K.—Chung, H.: Metrics for Maintainability of Class In-heritance Hierarchies. Journal of Software Maintenance 14, 3 May 2002, pp. 147–160.

[10]. Weyuker, E. J.: Evaluating Software Complexity Measures. IEEE Transactions on Software Engineering, Vol. 14, 1988, No. 9, pp. 1357–1365.

[11]. Cherniavsky, J.—Smith, C.: OnWeyukers Axioms for Software Complexity Mea-sures. IEEE Transaction on Software Engineering, Vol. 17, 1991, No. 6, pp. 636–638.

[12]. Gursaran, G.R.: On the Applicability of Weyuker Property Nine to Object-Oriented Structural Inheritance Complexity Metrics. IEEE Transaction on Software Engineering, Vol. 27, 2001, No. 4, pp. 361–364.

[13]. Sharma, N.—Joshi, P.—Joshi, R.K.: Applicability of Weyuker's Property 9 to Object-Oriented Metrics. IEEE Transaction on Software Engineering, Vol. 32, 2006, No. 3, pp. 209–211.

[14]. Deepti Mishra: New Inheritance complexity metricsfor object – oriented software systems:An evaluation with weyker's properties Computing and Informatics, Vol. 30, 2011, 267–293.

[15]. Chidamber, S.R.—Kemerer, C.F.: Towards A Metrics Suite for Object Oriented Design, OOPSLA'91, pp. 197-211, 1991.

[16]. Chidamber, S.R.—Kemerer, C.F.: A Metrics Suite for Object Oriented Design, M.I.T.Solan School of Management 1993.

[17]. wang CC, shih TK ,paiWC An automatic approach to object –oriented software testing and metrics for c++ inheritance hierarchies, proceedings International Conference on Automated Software Engineering (ASE'97), IEEE Computer Society press 1997;934-938

[18]. BasiliVR,Biand LC Melo WL A validation of object-oriented metrics as quality indicators, Technical Report,University of Maryland, Department of computer science,1995; 242-249.

[19]. Darcy, D.P.—Kemerer, C. F.: OO Metrics in Practice. IEEE Softw. 22, 6 November 2005, pp. 17–19. DOI: http://dx.doi.org/10.1109/MS.

2005

[20]. Ghassanalkadi, Application of a revised DIT metric to Redesign an OO Design, Journal of Object technology , Vol. 2,Issue 3,pp 897-910,2005.

[21]. Basili, V.R.:Viewing Maintenance As Reuse Oriented Software Development. IEEE Software, Vol. 7, 1990, No. 1, pp. 19–25.

[22]. Cartwright, M.—Shepperd, M.: An Empirical Analysis of Object Oriented Soft-ware in Industry. In: Bournemouth Metrics Workshop, April, Bournemouth, UK1996.

[23]. Li, W.—Henry, S.: Object-Oriented Metrics That Predict Maintainability. Journal of Systems and Software, Vol. 23, 1994, No. 2, pp. 111–122.

[24]. Sanjay Misra and Ibrahim Akman :Applicability of Weyuker's Properties on OO Metrics: Some Misunderstandings ,ComSIS Vol. 5, No. 1, June 2008

[25]. K. Rajnish, A. K. Choudhary, A. M. Agrawal, "Inheritance Metrics for Object-Oriented Design", *IJCSIT*, Vol. 2 No.6, December 2010, pp.13-26.

[26]. K. Rajnish and V. Bhattacherjee, "Class Inheritance Metrics-An Analytical and Empirical Approach", *INFOCOMP-Journal of Computer Science*, Federal University of Lavras, Brazil, Vol. 7 No.3, pp. 25-34, 2008.