USING BRIAND'S PROPERTIES TO THEORETICALLY VALIDATE LI INHERITANCE METRICS

Amany Mohammed Al Luhaybi
Computer Science Department, Faculty of Computing & Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
E-mail: amanymuhammed91@gmail.com

Wajdi Aljedaibi
Computer Science Department, Faculty of Computing & Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
E-mail: waljedaibi@kau.edu.sa

Abstract: Theoretical validation of metrics meant to ensure that the metrics work as it should be and measure what it intended to measure. There are a lot of validation techniques, most Object Oriented metrics where validated against Weyuker's property, However, Weyuker's property been criticized by number of researchers when it used to validate OO metrics. This paper will present theoretical validation using Briand's et al length and size properties of Li inheritance metrics which are Number of Ancestor Classes (NAC) and Number of Descendent Classes (NDC), who came up with new OO metrics, which measure different attributes such as Coupling, Complexity and Size, after finding out the limitation in Chidamber- Kemerer measures suite of object-oriented design.
Keywords —Object-Oriented, Inheritance Metrics, Briand's Properties, Metric Evaluation.

I- Introduction

Software measuring gained a lot of attention as it's a good way to predict, manage and ensure its quality. Software metrics been used widely, a new metrics are generated duo the limitations of some other metrics, so this field is always growing as it is important to improve the development of software. There are a number of objectives of these metrics [1][2] like understanding, software examination, planning, optimization and quality improvement. Based on the uses of these metrics they can be classified into different categories [3] like metrics for analysis model, metrics for design model, metrics for source code, metrics for testing, metrics for quality assurances etc.

There are different types of metrics as it is illustrated in Fig.1 below, this paper will discuss Li metrics of inheritance metrics a type of object oriented metrics, and the validation of the two inheritance metrics against Braind's et al properties of length and size. which are [4] Number of Ancestor Classes (NAC) and it measures total number of ancestor classes from which a class inherits in the class inheritance hierarchy. The other metric is Number of Descendent Classes (NDC) and it is the total number of descendent classes (subclasses) of a class. These OO metrics where proposed by Li after finding some deficiencies in Chidamber-Kemerer metrics [7] after they been validated using Kitchenham technique, and here is an important advantage of the theoretical evaluation as it may lead to creation of new metrics as it happened with Li.

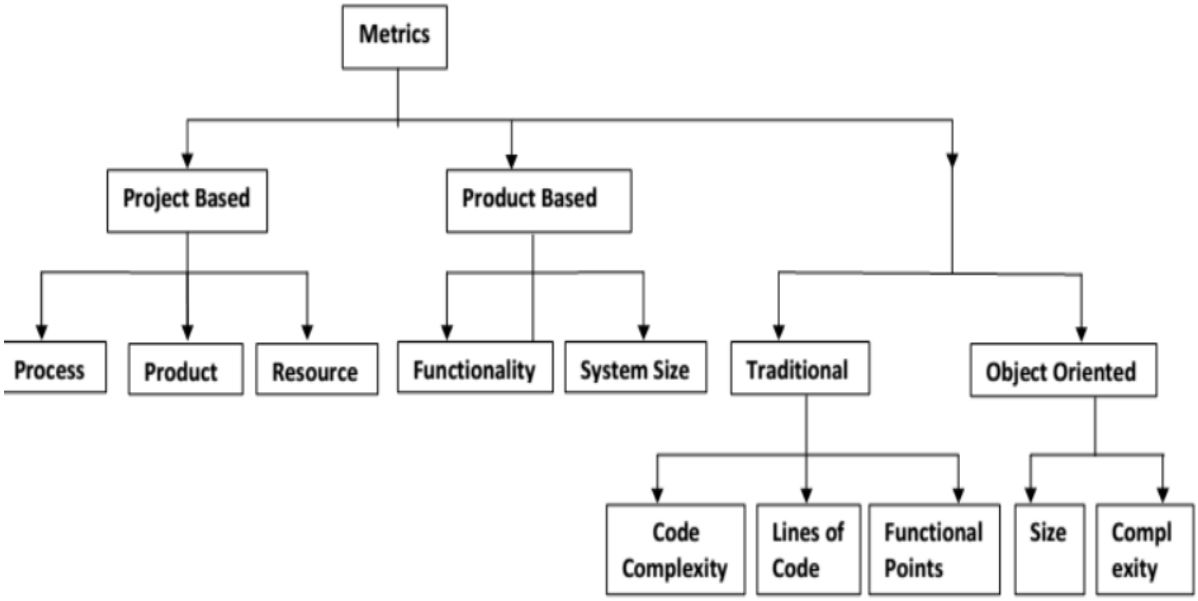


Fig.1. Types of Metrics [5].

The paper's organization is as follows: section II is the background divided into multi subsections which are: A. Li Metrics, B. Weyuker's Properties Critiqued by Other Researchers, C. Metrics that Been Validated Using Briand's Properties. Section III contains the Briand's et al Length and Size Properties that is used for validating inheritance metrics. Section IV is where the details Validation of NAC and NDC is taking place. Section V is the Conclusion.

II. Background

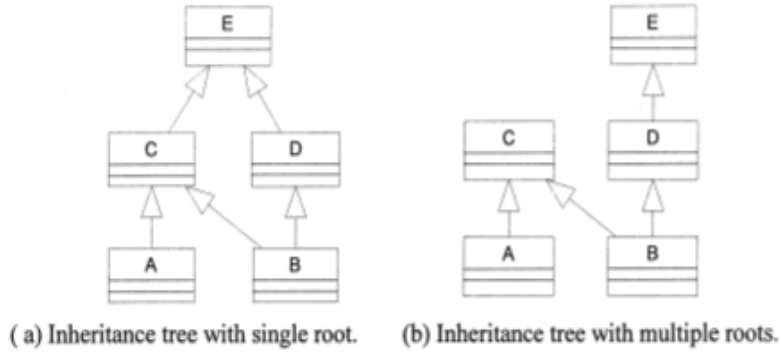
A. Li Metrics

In order to repair the deficiencies been found in some of the famous OO Chidamber-Kemerer metrics which can be found in [4]. Wi Li created six OO metrics as well, which will be briefly discussed below [4]:

- Number of Ancestor Classes (NAC)

Definition: NAC measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy.

The NAC values for class A and B in both Fig. 2(a) and (b). In Fig. 2(a), class A inherits from classes C and E, so the NAC values is NAC(A)= 2 ; class B inherits from three classes (C, D, and E), thus, yielding NAC(B)= 3. In Fig. 2(b), class A inherits from one class (C), thus, yielding NAC(A)= 1; class B inherits from three classes (C, D, and E), therefore, yielding NAC(B)= 3.



(a) Inheritance tree with single root. (b) Inheritance tree with multiple roots.

Fig. 2. Inheritance trees with single and multiple roots. [4]

- Number of Descendent Classes (NDC)

Definition: The NDC metric is the total number of descendent classes (subclasses) of a class.

- Number of Local Methods (NLM)

Definition: The NLM metric is the number of the local methods defined in a class which are accessible outside the class (e.g. public methods in C++).

- Class Method Complexity (CMC)

Definition: The CMC metric is the summation of the internal structural complexity of all local methods, regardless whether they are visible outside the class or not (e.g. all the public and private methods in C++).

- Coupling Through Abstract Data Type (CTA)

Definition: The CTA metric is the total number of classes that are used as abstract data types in the data- attribute declaration of a class.

- Coupling Through Message Passing (CTM)

Definition: The CTM metric measures the number of different messages sent out from a class to other classes excluding the messages sent to the objects created as local objects in the local methods of the class.

B. Weyuker's property critiqued by researchers

Weyuker suggested nine properties as Table 1 illustrated to validate metrics they can be found in details in [8], but they were for traditional programming or as it known procedural programming, therefore, the Weyukers never meant to be used for OO metrics validation so it may not be applicable to some of OO metrics, and it's only reasonable to search for another validation method.

Moreover, Weyuker's properties been used to validate the famous OO metrics Chidamber-Kemerer [7] so the Weyuker properties were generalized to be used or got accepted as validation method for all object oriented metrics.

Table 1. Weyuker's Properties

Property 1	**Noncoarseness**
Property 2	**Granularity**
Property 3	**Nonuniqueness (Notion of Equivalence)**
Property 4	**Design Details are Important**
Property 5	**Monotonicity**
Property 6	**Nonequivalence of Interaction**
Property 7	**Permutation**
Property 8	**Renaming Property**
Property 9	**Interaction Increases Complexity**

The generality of the first four properties of Weyuker leads to the assumption that they can be meet in any reasoning metrics. The fifth property is monotonic. It states that the metric value for the combination of classes/components is always greater than the metric value for any of components/classes. This is logical to happen, as if we increase the number of methods in the code the complexity is supposed to increase as well. But the Number of Catch Blocks per Class (NCBC) metric proposed by Aggarwal et al [9] doesn't satisfy this property [10]. Weyuker's property seven states that permutations of program statements can change the metric value. However, it may get important if the class complexity is calculated by the adding the method complexities since the order of statements are important in this case. Weyuker's ninth property saying that the intersection between two classes/components can increase the value of the complexity metrics

C. Metrics that Been Validated Using Briand's Properties.

There are a number of OO metrics that been validated against Briand's et al properties such as inheritance metrics of Chidamber-Kemerer. [6] which are Depth of Inheritance Tree Class (DITC) metric and Class Inheritance Tree (CIT) metric. Also some other metrics that the final result of their evaluation can be found in Table 2 [6]. Such as Lines of Code (LOC), Number of Concrete Classes defined in a system (NOC), Number of Methods defined in a class(NOM), Number of Attributes defined in a class (NOA), Number of occurrences of a keyword in a program (NOK), Number of occurrences of arithmetic operators in a program(NOAOP) and so on.

Table 2. Theoretical validation results of some metrics against Briand's et al size and length properties. [√: means metrics satisfy the property, ×: when metric doesn't satisfy the property]

Metric	S1, L1	S2, L2	S3	L3	L4	L5
LOC	√	√	√	√	√	×
NOC	√	√	√	√	√	×
NOM	√	√	√	√	√	×
NOA	√	√	√	√	√	×
SIZE2	√	√	√	√	√	×
NOK	√	√	√	√	√	×
NOAOP	√	√	√	√	√	×
Class-Leaf Depth (CLD)	√	√	×	√	√	√
Reuse Ratio (RR)	√	√	×	√	√	×
Specialization Ratio (SR)	√	√	×	√	×	×
DIT	√	√	×	√	√	√
Fandown	√	√	√	√	√	×
Fanup	√	√	√	√	√	×
NIA	√	√	√	√	√	×
NIM	√	√	√	√	√	×
NoVM	√	√	√	√	√	×
DITC	√	√	√	√	√	×
CIT	√	√	×	√	√	×
ICC	√	√	×	√	√	×
ICT	√	√	×	√	×	×

III. Briand's et al length and size properties

A. Size property

There are three properties lies under it which are non-negativity so the size is always positive, the second property is null value, when there are no elements in the system we expect to have zero as its size, the last property is Module Additivity when modules do not have elements in common, we expect size to be additive. Properties can be found summarized in Table 3 [6].

Table 3. Size properties

Property	*Description*
S1 (Non-negativity)	The size of a system is nonnegative.
S2 (Null value)	The size of a system is *0* if the set of elements which constitute the system is empty.
S3(Module Additivity)	The size of a system cannot be more than the sum of the sizes of its modules. In the case of disjoint modules, the size of a system is equal to the sum of the sizes of the modules.

B. Length property

It has 5 properties which can be found summarized in Table 4 [6]. Property 1 and 2 are the same as size properties which are the length is either positive value or zero. Property 3 indicates when we add new relationships between the component of the systems like attributes and methods doesn't increase the system size. Property 4 if the system is made of two modules then adding relationships between these two modules will increase its length. Property 5 is if we have two modules that will create the system then the system's length can be calculated as the maximum length of these two modules.

Table 4. Length properties

Property	Description
L1 (Non-negativity)	The length of a system is nonnegative.
L2 (Null value)	The length of a system is *0* if the set of elements which constitute the system is empty.
L3(Non Increasing Monotonicity)	Adding relationships between the elements of a module m in a system does not increase the length of the system.
L4 (Non Decreasing Monotonicity)	A system having modules *m1* and *m2* such that they are represented by separate connected components in the system. Adding relationships from elements of *m1* to elements of *m2* does not decrease the length of system.
L5 (Merger)	The length of a system made of union of two disjoint modules *m1* and *m2* is equal to the maximum of the lengths of *m1* and *m2*.

IV. Theoretical validation

A. Number of ancestor classes (NAC)

The NAC metric calculated as counts the number of nodes reachable from a node within the tree (i.e., a class), or the number of all ancestor classes that affect that particular class [11].

- Properties S1, S2, L1 and L2

It can be clearly seen from the definition that these properties are satisfied by the NAC metrics. As the fig2 (a) or (b) shows at particular class it either has a positive value if it inherits from other classes or it could be a zero value if it doesn't.

- Property S3 and L5

NAC satisfy S3 property, as it can be proven in fig.3 as the following:
NAC(D)=2
NAC(C)=1
NAC(C+D) = 3
Therefore, NAC(D)+ NAC(C)= 2+1=NAC(C+D) = 3
So the size of the system is equal to the sum of its modules. Therefore, the system which is made of the joining of the two modules M1 and M2 doesn't equal to the maximum of the lengths of its modules so it doesn't satisfy the property L5.

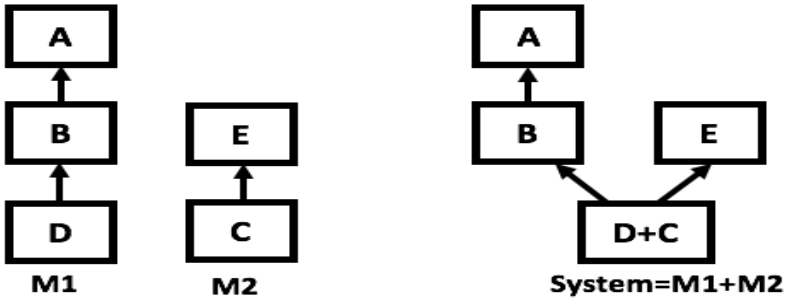


Fig.3. Two disjoint modules M1 and M2 and system after joining the modules.

- Property L3

NAC satisfy this property. As adding new relationships between the inner components of one single module will not effect the metric value like adding new connections between the attributes or methods within the class will not cause to increase the length of the module.

- Property L4

NAC satisfy this property. As adding new relationships from element in M1 to element in M2 will cause to increase the length of the system therefore, the value of the metric will be effected, mainly will be increased.

B. Number of Descendent Classes (NDC)

The metric counts the number of nodes reachable from a class, and considers all descendent classes that affect that particular predecessor class [11].

- Properties S1, S2, L1 and L2

It can be clearly seen from the definition that these properties are satisfied by the NDC metrics. As the fig2 (a) or (b) shows at particular class it either has a positive value if it affects other subclasses or it could be a zero value if it doesn't.

- Property S3 and L5

NDC satisfy S3 property, as it can be proven in fig.4 as the following:
NDC(A)=2
NDC(E)=1
NDC(A+E) = 3
Therefore, NDC(A)+ NDC(E)= 2+1=NDC(A+E) = 3
So the size of the system is equal to the sum of its modules. Therefore, the system which is made of the joining of the two modules M1 and M2 doesn't equal to the maximum of the lengths of its modules so it doesn't satisfy the property L5.

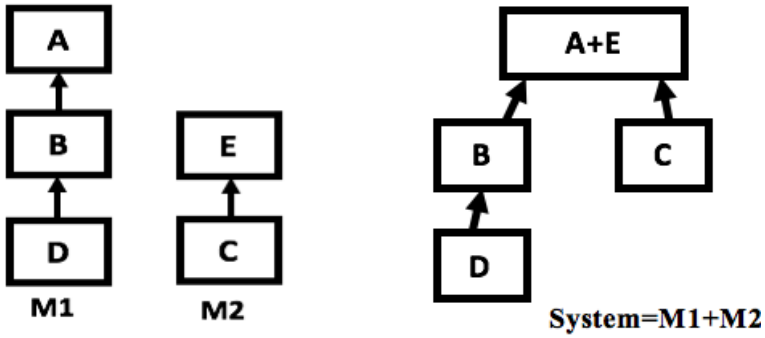


Fig.4. Two disjoint modules M1 and M2 and system after joining the modules.

- Property L3

NDC satisfy this property. As adding new relationships between the inner components of one single module will not effect the metric value like adding new connections between the attributes or methods within the class will not cause to increase the length of the module.

- Property L4

NDC satisfy this property. As adding new relationships from element in M1 to element in M2 will cause to increase the scope of inheritance, therefore, the length of the system will be increased so the value of the metric will be effected, mainly will be increased.

V. Conclusion

This paper contains a theoretical validation of Inheritance Metrics proposed by Li against the length and size properties of Briand *et al*. And it shown that for both NAC and NDC the metrics satisfy all size properties, but for the length properties it satisfies all except the L5 property. Table 5 shows the results and the contribution of this paper (the mark √ means satisfy and X means it doesn't satisfy it).

The future work will cover the whole Li suite metrics to be validated using other prosperities of Briand et al

Table 5. Evaluation result.

Metric	**S1, L1**	**S2, L2**	**S3**	**L3**	**L4**	**L5**
NAC	√	√	√	√	√	X
NDC	√	√	√	√	√	X

References

Manik Sharma , Gurdev Singh "Static and Dynamic metrics- A Comparative Analysis", Emerging Trends in Computing and Information Technology 2011.

Tu Honglei , Sun wei, Zhang Yanan, "The Research of Software metric and software complexity metrics" International Forum on Computer Science Technology and Applications(2009) Publisher: IEEE, Pages: 131-136

Roger S. Pressman "Software Engineering-A Practitioner's Approach" 6th Edition, McGraw Hill International Edition pp 466-472

Wei Li .(1998). Another metric suite for object-oriented programming. The Journal of Systems and Software 44, 155-162

P.Ashok Reddy, Dr.K.Rajasekhara Rao & Dr.M.Babu Reddy.(2015). Performance Evaluation of Procedural Metrics and Object Oriented Metrics. International Journal of Research Studies in Computer Science and Engineering (IJRSCSE), PP 69-72.

Rajnish, K., 2014, "Theoretical Validation of Inheritance Metrics for Object-Oriented Design against Briand's Property," International Journal of Information Engineering and Electronic Business, 3, pp. 28-33.

Chidamber, S.R and Kemerer, C.F.:A Metric Suite for Object Oriented Design,, IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, (1994).

Weyuker, E. J.: Evaluating Software Complexity Measure. IEEE Transaction on Software Complexity Measure, 14(9), pp. 1357-1365,(1988).

Aggarwal K.K., Singh,Y. Kaur, A. and Melhotra, R.: Software Design Metrices for object orinted Software, Journal of Object Technology, vol.6.no.1. pp. 121-138, (2006).

Misra S., Akman I. (2008) Applicability of Weyuker's Properties on OO Metrics: Some Misunderstandings, Journal of Computer and Information Sciences, 5(1) pp. 17-24

Sheldon, Frederick T., and Hong Chung. "Measuring the complexity of class diagrams in reverse engineering." *Journal of Software Maintenance and Evolution: Research and Practice* 18.5 (2006): 333-350.