Theoretical Validation of Inheritance Metric in QMOOD against Weyuker's Properties

Mariam Alharthi & Wajdi Aljedaibi

Computer Science Department, Faculty of Computing and Information Technology
FCIT, King Abdul-Aziz University KAU
Jeddah, Saudi Arabia SA
E-mail: waljedaibi@KAU.EDU.SA

Abstract
Quality Models are important element of the software industry to develop and implement the best quality product in the market. This type of model provides aid in describing quality measures, which directly enhance the user satisfaction and software quality. In software development, the inheritance technique is an important mechanism used in object-oriented programming that allows the developers to define new classes having all the properties of super class. This technique supports the hierarchy design for classes and makes an "is-a" association among the super and subclasses. This paper describes a standard procedure for validating the inheritance metric in Quality Model for Object-Oriented Design (QMOOD) by using a set of nine properties established by Weyuker. These properties commonly using for investigating the effectiveness of the metric. The integration of two measuring methods (i.e. QMOOD and Weyuker) will provide new way for evaluating the software quality based on the inheritance context. The output of this research shows the extent of satisfaction of the inheritance metric in QMOOD against Weyuker nine properties. Further results proved that Weyker's property number nine could not fulfilled by any inheritance metrics. This research introduces a way for measuring software that developed using object-oriented approach. The theoretical validation of the inheritance metric presented in this paper is a small step taken towards producing quality software and in providing assistance to the software industry.

Keywords; *software measurement, Theoretical validation, QMOOD, Weyuker's properties, Inheritance metrics.*

1. Introduction

The software measurement is a significant discipline in software engineering for controlling and understanding the development of products. Therefore, the validation of software measures is crucial to the software measurement success to measure and quantify the attributes accurately [1, 2]. Over the past years, many ways for software measurement validation have emerged. Schneidewind [3] shows that software metric is valid empirically if it is related with different measures of interest. [4] identifies a set of properties by focusing on complexity metrics and evaluate the metric according to these properties through determining whether the metric verifying each property. Fenton [5] and Melton [6] recommended that a valid measuring variable should follow measurement theory of representation condition, where the understanding of attributes not changed while mapping to the numerical system. Fenton and Kitchenham [7] proposed two distinct approaches to validation: the first determines the measure usefulness for predictive purposes, where the second determining to what extent a measure distinguishes a declared attribute.

With regard to software measures, there are several available measures for various aspects of the software and various levels of granularity. The software engineers or maintainers apply these analysis techniques to get an overview of particular characteristics of the software product. In this paper, a systematic investigation of QMOOD that was not investigated formally before in the literature is proposed by using Weyuker's nine properties. As discussed by [8] there are two types of validation (i) empirical validation, and (ii) analytical validation. Analytical validation is the same as theoretical validation, which implements the measuring procedure by employing predefined properties. This research is focused on theoretical validation. Weyuker's set of nine properties is specifically designed for measuring the software complexity using the theoretical validation process [4]. In addition, this investigation provides a new way by integrating the traditional Weyuker's software complexity measures with object-oriented design metrics suit that is QMOOD. As the idea proposed in this paper is to validate inheritance metrics, whereas the QMOOD metrics suite is used for evaluating the object-oriented designs for different properties such as; inheritance, polymorphism, complexity, and others.

The paper explains some basic concepts related to validation methodologies in software engineering field, Weyuker's properties and QMOOD metrics in Section II. In Section III, an overview of some inheritance

Manuscript received July 5, 2021
Manuscript revised July 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.7.33

metrics that are validated against Weyuker's properties is presented. Sections IV and V present the research problem and evaluation of the Measures of Functional Abstraction (MFA) metric against the nine properties of Weyuker. In section VI, the results were summarized in comparison to the evaluation results of the other inheritance metrics. Finally, the last two sections summarize the overall work and future work respectively.

II. BACKGROUND

Software quality metrics are used to measure the performance and successful implementation of a software [9]. There are number of techniques used for validating the software performance and quality [10, 11]. The subsequent section provides the detail on different methodologies and mechanism used for evaluating software engineering.

A. Validation Methodologies in Software Engineering

In software evaluation, two types of measuring metrics are very common, which are theoretical and empirical validation, see Fig. 1. In theoretical validation, it verifies that a measuring factor does not break any important properties for the estimating factor. The theoretical methods validating a measure with respect to certainly characterized criteria. While in empirical validation, prove that the values measured for attributes are compatible with the predicted values of attributes in the models. The empirical methods are considered as confirmed evidence of whether a measure is valid or not. In order to prove a metric validity, two kinds of validation are required. Moreover, theoretical validation analyzes the properties of attributes for a metric to be measured and provides information related to the fundamental statistical and mathematical operations that performed with the measure. The theoretical validation can perform using two different approaches as shown in Fig. 1[12].

In representational theory, Pfleeger, S.L., Kitchenham, B., and Fenton, N., explained the measure properties [13]. It depends on transformation or mapping between numerical and empirical worlds, where the characteristics of attributes in the real situation must be preserved by measures in a numerical world [12]. The property-based approach or axiomatic approach is proposed by [4] and [14]. It is called axiomatic approach because it uses a group of axioms for software attribute for theoretical validation, other names for this approach also are: algebraic and analytical validation, three popular types are shown in Fig. 1 [12]. Weyuker, E.J., [4] consists of nine properties that is commonly using for evaluating the appropriateness of object-oriented measures and properties solely important to demonstrate the measure validation. Kitchenham, B., et al. [13] gives a wider approach of theoretical validation. [14] depicts properties of measures for complexity, size, coupling, length and cohesion, where the representation condition is a prerequisite to measuring validation [14].

Turning to empirical validation, there are three types, which are experiments, case studies, and surveys as shown in Fig. 1. Based on the goal of measure, there are different techniques for empirical validation, such as prediction or evaluation, the amount of collected data, and the type [12].

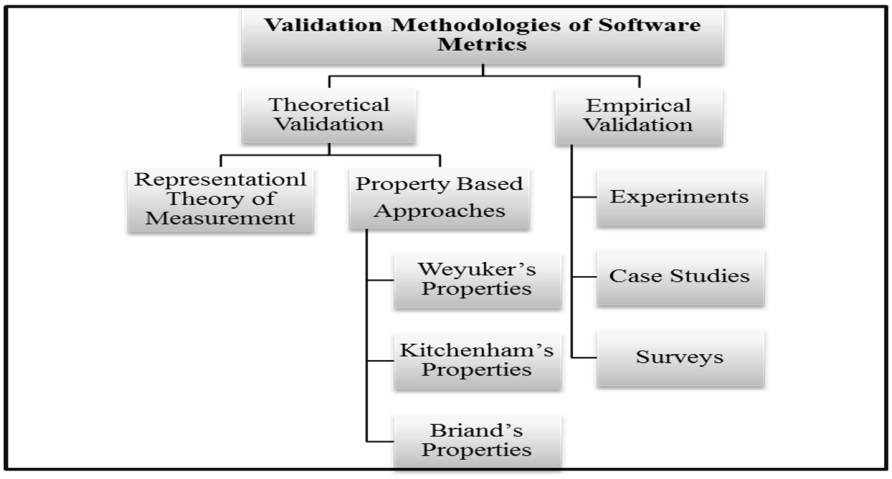


Figure 1 Validation methodologies of software metrics [12]

B. Weyuker's Properties

The most popular and used properties of software complexity measures, which has been used several time for validating object-oriented metrics, for instance: C-K metrics [15], variable categorization metrics [16], and DepDegree [17]. The assessment can be accomplished by determining set of properties that measures, whether a metric violated any property or not.

The nine properties of Weyuker are shown in Table 1. In property 1, suppose there are two classes P and Q, $|P|$ and $|Q|$ shows the complexity for classes P and Q respectively. This property shows that not each class can have similar value for metric. Property 2 suggests that a finite number of programs produce the similar value for a metric. Each of these programs has a finite number of classes; consequently, this feature will be satisfied when measuring at the class level for any metric. Property 3 shows that distinct programs P and Q can have the similar complexity or metric value. In property 4, suppose there are two programs P and Q, the two programs have the same functionality but during to the differences in the design details of each one, the metric value or complexity differs. Property 5 shows that the complexity or metric value of any two classes P and Q should be less than or equal to the metric value of a combination of the two classes together P;Q or P+Q. Property 6 shows that when there are three classes P, Q, and R, where the metric value or complexity for P and Q is the same $|P| = |Q|$, but the metric values for P;R and Q;R are different.

It confirms that the contact between P and R classes can be distinct from the contact between Q and R classes. The same can be proved when the combinations are R;P and R;Q. Property 7 proved the order of statements in any program can matter the complexity or the metric value for that program. Suppose there is a program P, from permuting P statements the program Q is found. This property proves that in this case, metric values for P and Q are not same. Furthermore, property 8 denotes that the program P changed its name to Q, the metric remains and should not change. The last property which is the property 9 proves that the difficulty of a combination of two programs P and Q is larger than the sum of the two complexity values for each program. This shows that the interaction increases the complexity

Table 1 Weyuker's nine properties [4]

Property No.	Property Measure	Formula
Property 1	Non-Coarseness	$\exists P, \exists Q \ \& \ \|P\| \neq \|Q\|$
Property 2	Granularity	-
Property 3	Non-Uniqueness (Notion of equivalence)	$\|P\| = \|Q\|$
Property 4	Design Details are Important	$P \equiv Q \ \& \ \|P\| \neq \|Q\|$
Property 5	Monotonicity	$\|P\| \leq \|P; Q\| \ \& \ \|Q\| \leq \|P; Q\|$
Property 6	Non-Equivalence of Interaction	$\|P\| = \|Q\| \ \& \ \|P; R\| \neq \|Q; R\|$ $\|P\| = \|Q\| \ \& \ \|R; P\| \neq \|R; Q\|$
Property 7	Permutation	$\|P\| \neq \|Q\|$
Property 8	Renaming property	$\|P\| = \|Q\|$
Property 9	Interaction Increases Complexity	$\|P\| + \|Q\| < \|P; Q\|$

From the listed 9 properties, property 2 and property 8 are satisfied for all object-oriented metrics. And property 7 is not required for the object-oriented programs; it is for the traditional programs [12].

C. QMOOD metrics

The Quality Model for Object Oriented Design (QMOOD) metrics was developed by Bansiya and Davis in 2002 [18]. It is used to assess or quantify multiple quality factors for example, understandability, extendibility, reusability, flexibility, functionality, and effectiveness [19]. These attributes can be quantified by using a formula for each attribute as shown in Table 2 [19].

The QMOOD hierarchy model involves four levels and three mappings or links, as shown in Fig. 2. The design properties that are involved in QMOOD set are listed in Table 3 [19].

Table 2 Formulas of quality attributes [19]

The Quality	**Computation Formula**
Understandability	-0.33*Abstraction+0.33*Encapsulation- 0.33*Coupling+0.33*Cohesion- 0.33*Polymorphism 0.33*Complexity-
Extendibility	0.5*Abstraction- 0.5*Coupling+0.5*Inheritance+0.5*polymorphism
Reusability	-0.25* Coupling +0.25*coupling+0.5*Messaging+0.5*Design Size
Flexibility	0.25*Encapsulation- 0.25*Coupling+0.5*Composition+0.5*Polymorphism
Functionality	0.12*Cohesion+0.22+Polymorphism+0.22*Messaging +0.22*Design Size+0.22*Hierarchies
Effectiveness	0.2*Abstraction+0.2* Encapsulation +0.2*Composition+0.2*inheritance+0.2*Polymorphism

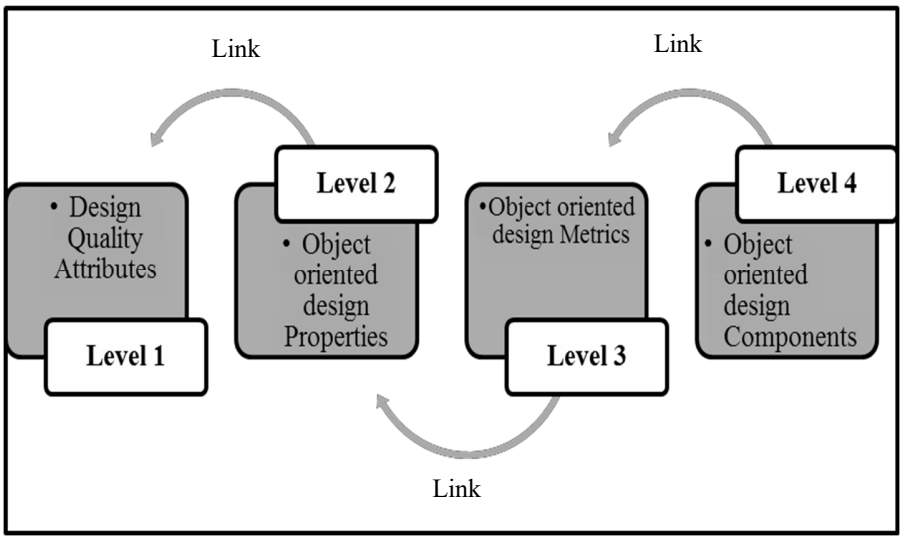


Figure 2 The model of QMOOD metrics [19]

III. RELATED WORKS

DIT and NOC are the inheritance metrics proposed by [15]. DIT measures the Depth of Inheritance Tree for a class. And NOC counts the Number of Children (subclasses) for a super class in the inheritance hierarchy. Also, TPC, TPAC, and TAC are the inheritance metrics proposed by Brito and Carapuca [20]. Total Progeny Count (TPC) counts the direct and indirect subclasses for a particular class. Total Parent Count (TPAC) counts the number of the parent or super-classes for a specified subclass. Total Ascendancy Count (TAC) counts the number of direct and indirect super-classes for a particular subclass. Li's [21] have proposed alternative inheritance metrics for DIT and NOC which are Number of Ancestor Class (NAC) and Number of Descendant Class (NDC). NAC measures the number of ancestor (super) classes for a specified subclass. NDC calculates the number of descendant classes (subclasses) for a specific class. The Depth of Inheritance Tree of a Class (DITC) is proposed by Rajnish and Bhattacherjee [22] [23] [24]. DITC calculates all the kinds of attributes and methods for a class, which includes the protected, private, public and inherited attributes and methods by using a particular formula.

Table 3 QMOOD design properties and metrics [19]

Design Property	Metrics	Description of Metrics
Design size	Design Size in Classes (DSC)	Produces a number of classes in the design.
Hierarchies	Number of Hierarchies (NOH)	Produces a number of class hierarchies in the design.
Abstraction	Average Number of Ancestors (ANA)	Produces the average number of classes that a specified class inherits information from them.
Encapsulation	Data Access Metrics (DAM)	Divides the number of private attributes in a class to the total number of attributes, and return the result (ratio).
Coupling	Direct Class Coupling (DCC)	Calculates the number of classes that a specified class depends on them.
Cohesion	Cohesion among Methods of Class (CAM)	Calculates the methods relatedness of a class with respect to the number of parameters.
Composition	Measure of Aggregation (MOA)	Computes the part-whole relationship extent that arises from using attributes.
Inheritance	Measures of Functional Abstraction (MFA)	Divides the number of inherited methods by a class to the number of accessible methods through the member methods of a class, and return the result (ratio).
Polymorphism	Number of Polymorphic Methods (NOP)	Computes how many methods can exhibit polymorphic behavior.
Messaging	Class Interface Size (CIS)	Computes how many public methods are in a class.
Complexity	Number of Methods (NOM)	Returns the number of methods in a class.

IV. RESEARCH PROBLEM

This paper aims to validate theoretically the Measures of Functional Abstraction (MFA) metric in QMOOD that is not validated in the literature before, which represent the inheritance design property. This validation process will be carried against the nine Weyuker's properties. The main purpose of this validation is to observe the object-oriented QMOOD metrics with traditional Weyuker's software quality measures. Most importantly, Weyuker's properties designated for theoretical validation, as this paper is evaluating the inheritance metric using theoretical validation.

V. EVALUATION

The Measures of Functional Abstraction (MFA) can be defined as follows in equation (1):

$$\frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} \qquad (1)$$

And

$$M_a(C_i) = M_d(C_i) + M_i(C_i) \qquad (2)$$

Where

$M_d(C_i)$ = The number of declared methods in a class C_i

$M_i(C_i)$ = The number of methods that can be inherited in a class C_i

$M_a(C_i)$ = all methods that can be invoked in a class C_i

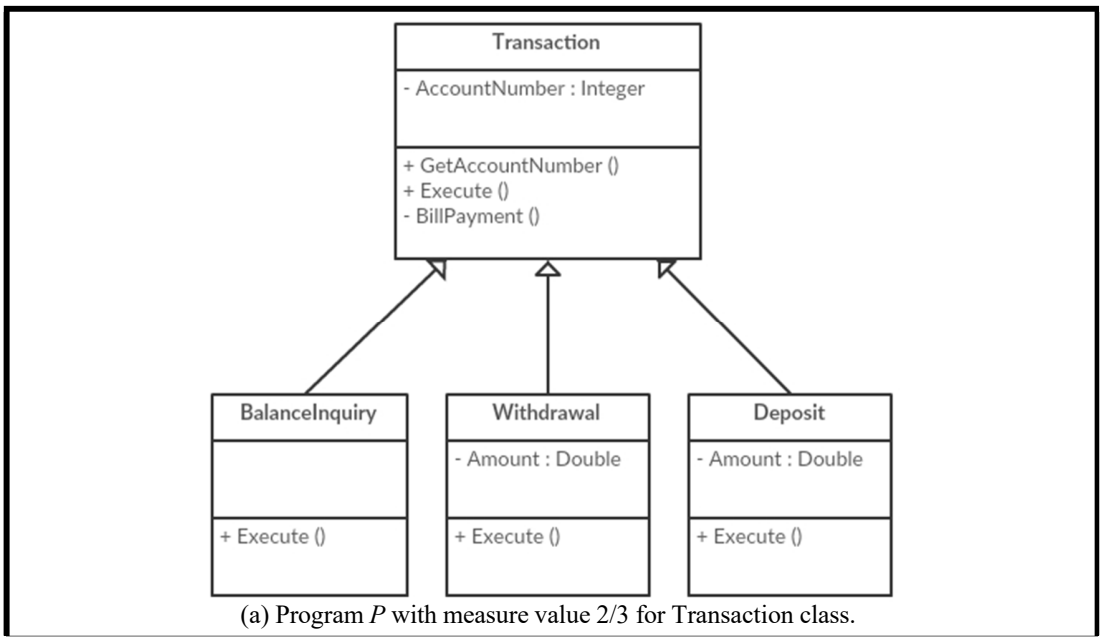
TC = Total Count

With MFA, for each class C1, C2, …, Cn, a method counts as 0 if cannot be inherited and 1 if can be inherited. The total number of inherited methods is divided by the total number of methods for the system; the total number of methods involves the inherited and not inherited methods as defined in equation (2). The result represents the ratio or proportion of inheritance for the system. Note that, the constructors and the java.lang.Object (as a parent) are ignored in computation.

In another hand, the complexity that going to calculated here is the degree or ratio of inheritance, which can be denoted by this symbol $|P|$ (suppose the program name is P).

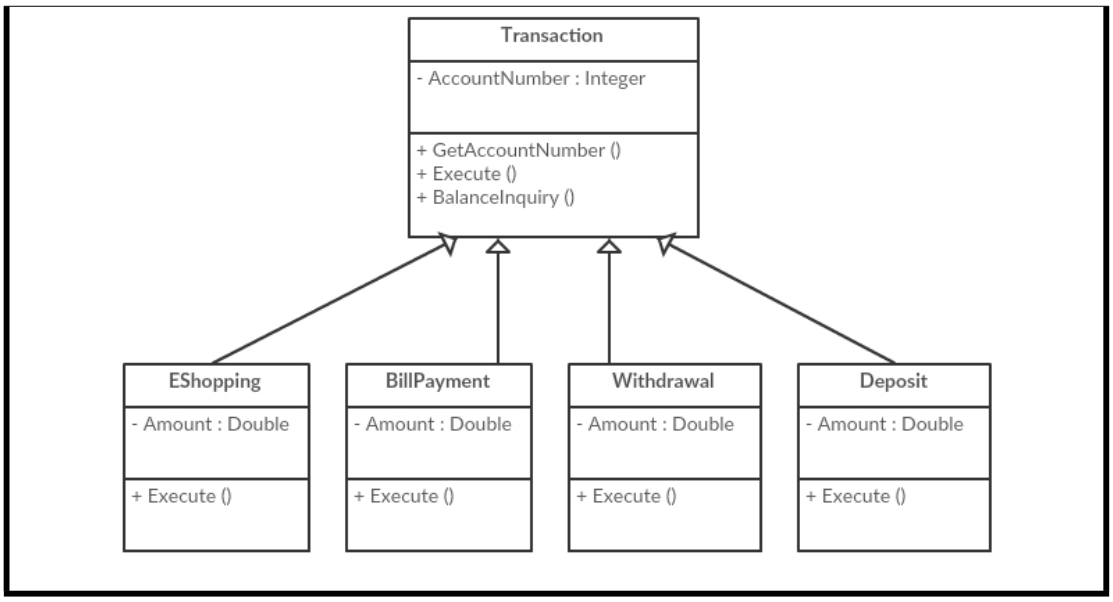
Now, each property will be proved separately as following:

Property 1: there are two programs P and Q exist in a specified domain where $|P| \neq |Q|$. Fig. 3 (a, b) shows the two programs with different measurement values. Thus, this property is satisfied.



(a) Program *P* with measure value 2/3 for Transaction class.

Figure 3(a) Different designs of Transaction System

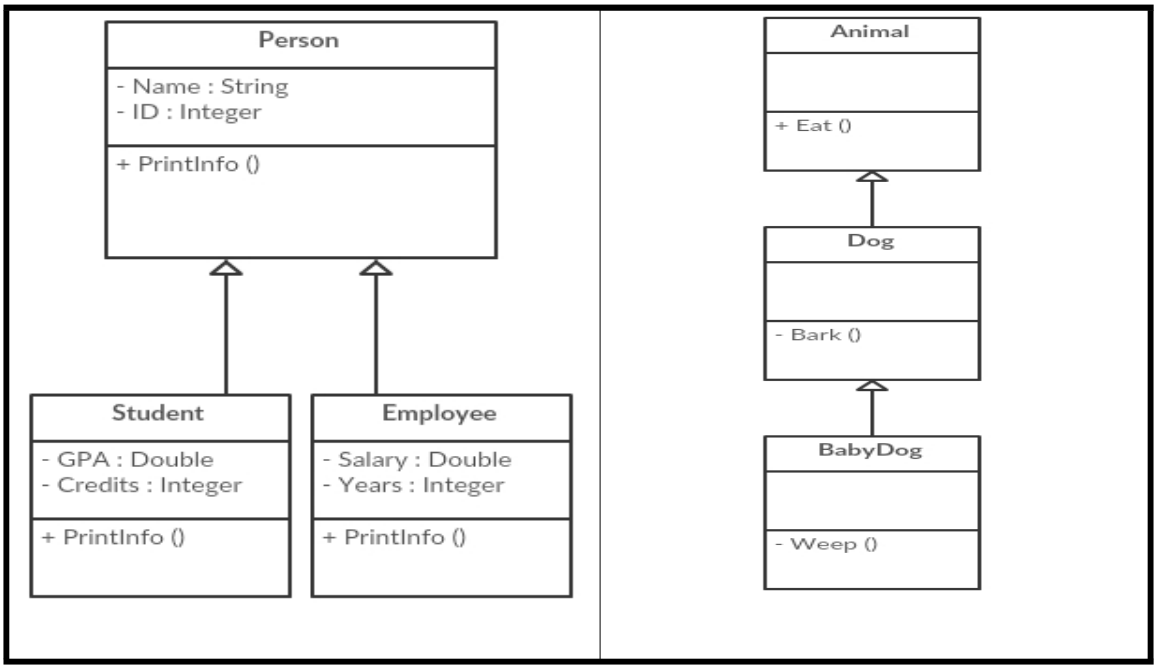


(b) Program *Q* with measure value 3/3 or 1 for Transaction class.

Figure 3(b) Different designs of Transaction system.

Property 2: if c is a non-negative number, then there is a finite set of programs with a measurement value c. This property is satisfied for all object-oriented metrics as mentioned above (in the background section).

Property 3: there are two distinct programs *P* and *Q* where $|P| = |Q|$. Fig. 4 shows the two programs with the equal measurement value 1/1 or simply 1 for the mentioned classes in the figures. Therefore, this property is satisfied also.

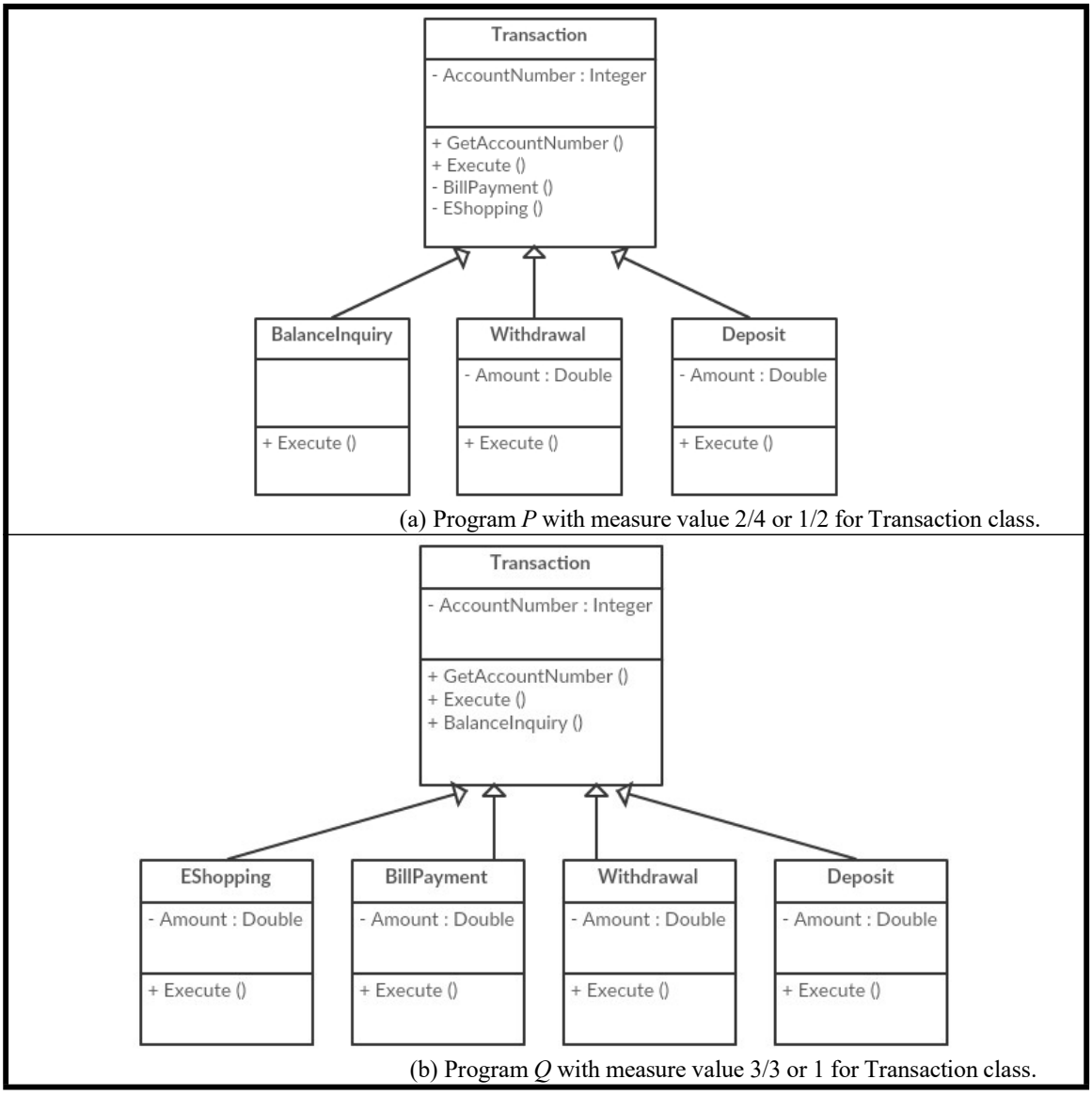


(a) Program *P* with measure value 1/1 or 1 for Person class. (b) Program *Q* with measure value 1/1 or 1 for Animal class.

Figure 4 Different distinct programs with different designs.

Property 4: if there are two programs P and Q are exist, where the two programs have an equivalent functionality ($P \equiv Q$), then $|P| \neq |Q|$. Fig. 5 shows different implementations of the same program (transaction program), where the degree of inheritance for a transaction class differs according to the implementation. Therefore, this property is satisfied for MFA.

In addition, the metrics that based on measuring an interface rather than the implementation will not satisfy this property because if there are two programs with the same functionality (means the same interface in this case), the measured value will be the same, but the implementations can differ. And because these metrics are only sensitive to interfaces not to implementations, they will have the same parameters in each interface; hence, the two programs have the same measurement value. These types of metrics can be called Interface Sensitive and Implementation Insensitive metrics.

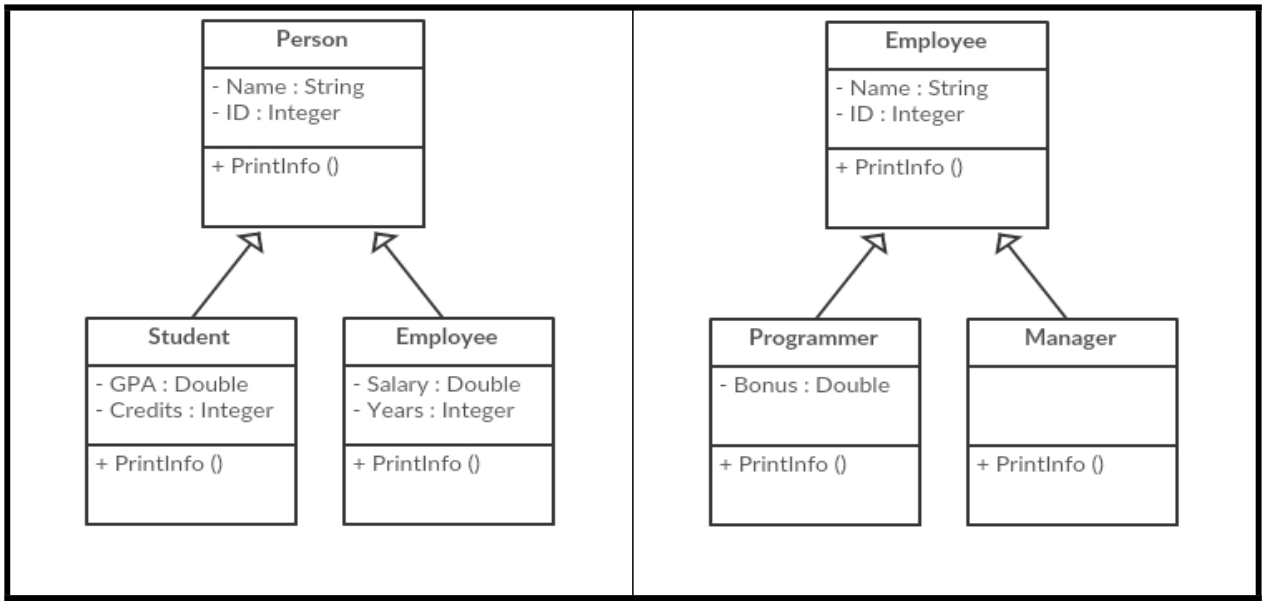


(a) Program *P* with measure value 2/4 or 1/2 for Transaction class.

(b) Program *Q* with measure value 3/3 or 1 for Transaction class.

Figure 5 Different implementations of Transaction program.

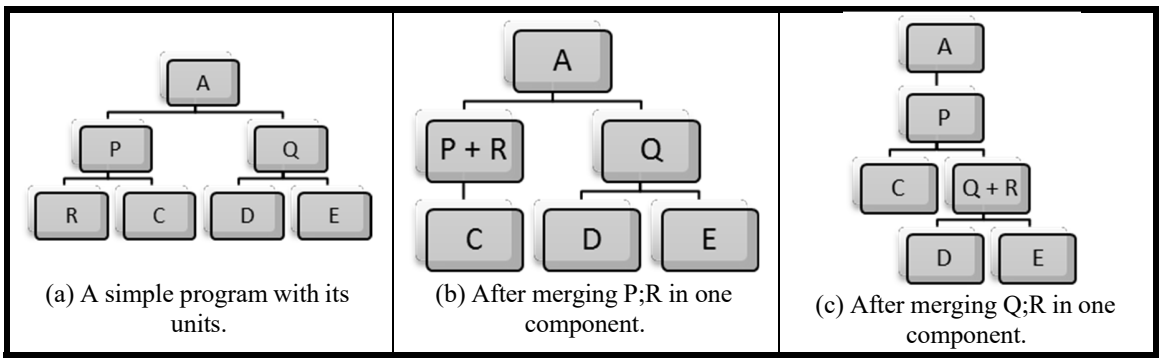
Property 5: if there are two programs *P* and *Q*, then $|P| \leq |P;Q|$ and $|Q| \leq |P;Q|$. The (; operator) denotes concatenating operation and it can place also by (+ operator). The meaning of concatenating operation is merging the two programs together. Fig. 6 shows two programs with their measure value for a parent or super class of each program. Merging or concatenation of two programs (*P;Q* or *P*+*Q*) yields to compute the degree or ratio of inheritance for the inherited string at all of the merged program. For *P*, the ratio is 1/1, also the ratio for *Q* is 1/1, hence the ratio for *P;Q* is 1/1 which is equal, and that means this property is satisfied for MFA.



(a) Program *P* with measure value 1/1 or 1 for Person class. (b) Program *Q* with measure value 1/1 or 1 for Employee class.

Figure 6 Example of different two programs

Property 6: if there are three programs *P*, *Q*, and *R*, where $|P| = |Q|$, then property 6 consists from two parts: (a) $|P;R| \neq |Q;R|$, and (b) $|R;P| \neq |R;Q|$. Fig. 7 shows an example of a particular hierarchal scenario of inheritance. Suppose *P* and *Q* have the same inheritance ratio. Because *R* is a child of *P*, that means there is a relation between these classes. Therefore, the interaction between *P* and *R* is differing than the interaction between *Q* and *R*, which yields different measurement values for each concatenation case and it concludes that MFA is satisfying property 6.



(a) A simple program with its units.

(b) After merging P;R in one component.

(c) After merging Q;R in one component.

Figure 7 A specified scenario of inheritance.

Property 7: when there are two programs *P* and *Q*, where the statements of program *P* are permuted to form the body of program *Q*, then the $|P| \neq |Q|$. This property has been developed for traditional programs; hence, it is not required to the object-oriented programs as mentioned above (in the background section).

Property 8: if there are two programs *P* and *Q* where the program *P* is a renaming only from the program *Q*, then $|P| = |Q|$. In inheritance metric, the semantic preserves that renaming of variables, methods or program will not affect the degree of inheritance for the program. That concludes this property is satisfied for MFA.

Moreover, this property can affect the understandability metrics because the names of classes, methods, and variables are important, and renaming them to any poorly chosen names will cause information confusion. As well as, this property can affect the metric that measures the length of names and uniqueness of the name. Therefore, the metrics that can be affected by this property can be called Name Sensitive metrics.

Property 9: there are two programs *P* and *Q* exist, where $|P| + |Q| < |P;Q|$. This property needs to ensure that the interaction between the two programs in order to form a concatenated program will increase the complexity of the new program. Means, the complexity of the new program will be greater than the sum of its parts. Otherwise, if this property is not satisfied by the metrics, property 9' can be satisfied, which states that $\forall P, \forall Q : |P| + |Q| \geq |P;Q|$. In inheritance, the program has a set of inherited methods for its subclasses. When this program is split into two programs that mean each one of the new super-classes in both programs will have a copy of the inherited methods for the old super class. Therefore, the methods of the old super class will be found at both new super-classes. Due to this duplication, the sum of complexities of the two programs is greater than the old program (concatenated program). Hence, MFA does not satisfy property 9 but it is satisfying property 9'. For example, Fig. 6 shows two programs *P* and *Q* with a measurement value 1/1 for each. The sum of these measurement values is 1/1 + 1/1 which is 2/1. As said before (in property 5 proof), the measured value for *P;Q* is 1/1. Hence, 2/1 is greater than 1/1. That concludes MFA satisfy $\forall P, \forall Q : |P| + |Q| \geq |P;Q|$.

Property 7: when there are two programs *P* and *Q*, where the statements of program *P* are permuted to form the body of program *Q*, then the $|P| \neq |Q|$. This property has been developed for traditional programs; hence, it is not required to the object-oriented programs as mentioned above (in the background section).

Property 8: if there are two programs *P* and *Q* where the program *P* is a renaming only from the program *Q*, then $|P| = |Q|$. In inheritance metric, the semantic preserves that renaming of variables, methods or program will not affect the degree of inheritance for the program. That concludes this property is satisfied for MFA.

Moreover, this property can affect the understandability metrics because the names of classes, methods, and variables are important, and renaming them to any poorly chosen names will cause information confusion. As well as, this property can affect the metric that measures the length of names and uniqueness of the name. Therefore, the metrics that can be affected by this property can be called Name Sensitive metrics.

Property 9: there are two programs *P* and *Q* exist, where $|P| + |Q| < |P;Q|$. This property needs to ensure that the interaction between the two programs in order to form a concatenated program will increase the complexity of the new program. Means, the complexity of the new program will be greater than the sum of its parts. Otherwise, if this property is not satisfied by the metrics, property 9' can be satisfied, which states that $\forall P, \forall Q : |P| + |Q| \geq |P;Q|$. In inheritance, the program has a set of inherited methods for its subclasses. When this program is split into two programs that mean each one of the new super-classes in both programs will have a copy of the inherited methods for the old super class. Therefore, the methods of the old super class will be found at both new super-classes. Due to this duplication, the sum of complexities

of the two programs is greater than the old program (concatenated program). Hence, MFA does not satisfy property 9 but it is satisfying property 9'. For example, Fig. 6 shows two programs *P* and *Q* with a measurement value 1/1 for each. The sum of these measurement values is 1/1 + 1/1 which is 2/1. As said before (in property 5 proof), the measured value for *P;Q* is 1/1. Hence, 2/1 is greater than 1/1. That concludes MFA satisfy

$$\forall P, \forall Q : |P| + |Q| \geq |P; Q|.$$

VI. DISCUSSION

As shown that Measures of Functional Abstraction (MFA) metric fulfills the first eight Weyuker's properties, the only property that has not been satisfied is the ninth property. Table 4 shows the evaluation results of some inheritance metrics that are mentioned above (in related works section) against Weyuker's properties. The symbol ✓ denotes that a property is satisfied, where × denotes that a property is not satisfied. The formal evaluation of MFA is also summarized in the table for using it as a program complexity indicator.

Table 4 The properties fulfilled by the inheritance metrics

Weyuker property no.	DIT	NOC	TPC	TPAC	TAC	NAC	NDC	DITC	MFA
1	✓	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓	✓	✓	✓	✓
5	×	✓	×	✓	×	×	✓	×	✓
6	✓	✓	✓	✓	✓	✓	✓	✓	✓
7	✓	✓	✓	✓	✓	✓	✓	✓	✓
8	✓	✓	✓	✓	✓	✓	✓	✓	✓
9	×	×	×	×	×	×	×	×	×

VII. CONCLUSION & FUTURE WORK

An analytical evaluation of MFA metric has been conducted in this paper against Weyuker's properties. As observed in table 4, property 9 does not fulfilled by none of the inheritance metrics. This is due to the fact that complexity of a class cannot be reduced by splitting that class into other classes, the complexity may increase or remains the same. However, Zhang and Xie [25] proposed an inheritance metric that satisfied property 9; but in fact, there is no practical example for this metric. Also, Rajnish and Bhattcaherjee [26] gave an inheritance method that satisfied property 9. Besides that, Mal and Rajnish [27] proposed two new inheritance metrics as well, which are Inheritance Complexity of Class (ICC) and Inheritance Complexity of Tree (ICT). ICC metric measures at the class level, hence it did not satisfy property 9. Whereas ICT metric measures at tree level, and this is why this metric is satisfied property 9. Accordingly, the applicability of Weyuker's property 9 to inheritance metrics is still under discussion in the previous work. The future scope includes the validation of the remaining metrics in QMOOD against Weyuker's properties in order to produce a systematic validation of the entire QMOOD metrics theoretically.

References

[1] H. Zuse, “Software complexity: Measures and Methods”. Vol. 4. *Walter de Gruyter GmbH & Co KG*, 2019.

[2] N. Padmalata, K. Vithal Nori, and R. Reddy. "Software Quality Models: A Systematic Mapping Study." *IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pp. 125-134. IEEE, 2019.

[3] N. Schneidewind, “Methodology for validating software metrics,” *IEEE Transactions on Software Engineering,* vol. 18, no. 5, pp. 410-422, May 1992.

[4] E. Weyuker, “Evaluating software complexity measures,” *IEEE Transactions on Software Engineering,* vol. 14, no. 9, pp. 1,357-1,365, Sept. 1988.

[5] N. Fenton, “Software measurement: A necessary scientific basis,” *IEEE Transactions on Software Engineering,* vol. 20, no. 3, pp. 199-206, Mar. 1994.

[6] A. Melton, D. Gustafson, J. Bieman, and A. Baker, “A mathematical perspective for software measures research,” *J. of Software Eng.,* vol. 5, no. 5, pp. 246-254, 1990.

[7] N. Fenton and B. Kitchenham, “Validating software measures,” *J. of Software Technology, Verification und Reliability,* vol. 1, no. 2, pp. 27- 42, 1991.

[8] M. Sharma, G. S. Nasib Gill, and S. Sunil. "Survey of Object-Oriented Metrics: Focusing on Validation and Formal Specification." *ACM SIGSOFT Software Engineering Notes* 37, no. 6: 1-5, 2012.

[9] P. Ramesh, R. C. Reddy, “Object Oriented Dynamic Metrics in Software Development: A Literature Review”, *International Journal of Applied Engineering Research* Vol. 14, No. 22: 4162-4172, 2019.

[10] K. Anil, “Analysis of Object-Oriented System Quality Model Using Soft Computing Techniques”, *International Journal of Advance Research*, Ideas and Innovations in Technology, Vol. 5, No. 2, 2019.

[11] C. Gemma, F. Palomba, F. Fontana, A. De Lucia, A. Zaidman, and F. Ferrucci. "Improving change prediction models with code smell-related information." *Empirical Software Engineering*, 1-47, 2019.

[12] K. Srinivasan and T. Devi, "Software Metrics Validation Methodologies in Software Engineering", *International Journal of Software Engineering & Applications,* vol. 5, no. 6, pp. 87-102, 2014.

[13] Kitchenham, B., Pfleeger, S.L., and Fenton, N., “Towards a Framework for Software Measurement Validation,” *IEEE Transactions on Software Engineering,* Vol. 21, No.12, December, pp. 929-943, 1995.

[14] Briand, L.C., Morasca, S., Basili, V.R., “Property-Based Software Engineering Measurement,” *IEEE Transactions on Software Engineering,* Vol. 22, No.1, January, pp. 68-85, 1996.

[15] Chidamber, S.R., and Kemerer, C.F., “A Metrics Suite for Object-Oriented Design,” *IEEE Transactions on Software Engineering,* Vol. 20, No. 6, June, pp. 476-493, 1994.

[16] Radhika Raju, P., and Ananda Rao, A., “A Metrics Suite for Variable Categorization to Support Program Invariants,” *International Journal of Software Engineering & Applications,* Vol.5, No.5, September, pp. 65-83, 2014.

[17] D. Beyer and P. Häring, "A formal evaluation of DepDegree based on weyuker's properties", *Proceedings of the 22nd International Conference on Program Comprehension* - ICPC 2014, pp. 258-261, 2014.

[18] Bansiya J. and C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment, *IEEE Transactions on Software Engineering,* pp. 4-17, 2002.

[19] P. Goyal and G. Joshi, "QMOOD metric sets to assess quality of Java program", *International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT),* 2014.

[20] A. F. Brito and R. Carapuca, “Candidate Metrics for Object-Oriented Software within a Taxonomy Framework, *Journal of System Software*, vol. 26, 87-96, 1994.

[21] W. Li,”Another metric suite for object-oriented programming”, *The Journal of Systems and Software*; 44(2): pp.155-162, 1998.

[22] K. Rajnish and V. Bhattacherjee, “A New Metric for Class Inheritance Hierarchy: An Illustration”, *proceedings of National Conference on Emerging Principles and Practices of Computer Science & Information Technology*, GNDEC, Ludhiana, pp 321-325, 2006.

[23] K. Rajnish and V. Bhattacherjee, “Class Inheritance Metrics and development Time: A Study”, *International Journal Titled as PCTE Journal of Computer Science*, Vol.2, Issue 2: pp. 22-28, December 2006.

[24] 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.

[25] L. Zhang and D. Xie, “Comments on „On the applicability of Weyuker Property Nine to Object-Oriented Structural Inheritance Complexity Metrics, *IEEE Transaction Software Engineering*, Vol.28, no.5, 526-527, 2002.

[26] K. Rajnish and V. Bhattacherjee, “Applicability of Weyuker Property 9 to Object- Oriented Inheritance Tree Metric-A Discussion”, *proceedings of IEEE 10th International Conference on Information Technology* (ICIT-2007), published by IEEE Computer Society Press, pp. 234-236, December-2007.

[27] S. Mal and K. Rajnish, “Applicability of Weyuker’s Property 9 to Inheritance Metric”, *International Journal of Computer Application, Foundation of Computer Science*, USA, vol. 66, no. 12, 2013.