An Access Control Metric Suite for Class Hierarchy of Object-Oriented Software Systems

Rajender Singh Chhillar, Parveen Kajla, Usha Chhillar, and Narender Kumar, *Member, IACSIT*

***Abstract*—During last two decades, object-oriented paradigm has arisen as a prevailing software engineering practice for solving software problems. A large number of object-oriented metrics are proposed during that period to measure the properties like abstraction, encapsulation, inheritance, polymorphism, coupling, cohesion, information hiding and reusability. Most of the researchers mainly depict attributes like class, object, and method to measure the properties in their research. In this paper metrics are defined to depict Member Access Control mechanism and then employed in the class hierarchy. By successfully implemented the proposed metrics in object-oriented programing, we quantify the derived classes which directly results to reduce code, time and complexity of the object-oriented software systems. The proposed metrics provides a new way to understand and imply these concepts in research and development of the software using object-oriented approach.**

***Index Terms*—Attributes, access control specifiers, class hierarchy, inheritance, methods, object-oriented paradigm.**

I. Introduction

The Object oriented approach prevails over the function oriented approach. The class is the fundamental unit of OO development system. The class encapsulates methods and data members. Object oriented product metrics measure the effectiveness of object oriented technique. The OO approach metrics get its pace after the proposal of metric suite by Chidamber&Kemerer [1], [2] in 1994 and its exploratory analysis in 1998 [3]. Large research works have been conducted to validate CK Metric suite. Basili [4], Briand [5], Li [6], Tang [7] made empirical, theoretical, real-time system study on CK Metric suite. Harrison, Counsell and Nithi [8] in MOOD metric suite proposedmetrics on method and attribute hiding factor, inheritance factor, polymorphism and coupling factor. Lorenz and Kidd [9] proposed class based metrics into size, inheritance, internal and external categories of software development.

The six metrics proposed by Chidamber&Kemerer states

- Weighted Method per Class (WMC): WMC is the sum of complexities of all the methods in a class.
- Depth of Inheritance Tree (DIT): DIT is the maximum number of classes from the node to the root of the tree.
- Number of Children (NOC): NOC is the number of immediate subclasses subordinated to a class in the class hierarchy.
- Coupling between Objects (CBO): CBO is the number of other classes to which it is coupled.
- Response for a Class (RFC): RFC is a set of methods that can be executed in response to a message received by an object of that class.
- Lack of Cohesion of Methods (LCOM): LCOM is a count of the inter-relatedness between portions of a program.

This paper is organized into five sections. Section II describes the class hierarchy of object-oriented software systems proposed by the various researchers. Section III states proposed Member Access Control Metrics (MACM) for object-oriented systems. Section IV indicates the implementation of MACM by usinginheritance. Section V depicts observation and results. Section VI refers concluding remarks and future scope. At last references and biographies are portrayed.

II. Class Hierarchy of Object-Oriented Software Systems

A key feature of object-oriented programming languages like C++ is inheritance. Inheritance allows us to create classes which are derived from other classes. The derived class inherits the methods and data members of base class. Chidamber&Kemerer [2] defines DIT is a measure of how many ancestor classes can potentially affect this class. The deeper the class in the hierarchy, the greater the number of methods it is likely to inherit. MOOD metrics [8] states class hierarchy as Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF). Lorenz and Kidd [9] proposed class hierarchy as Number of Operations Added by a subclass (NOA). If the value of NOA is increased, it drifts away from the abstraction implied by the superclass. And with the increase of class hierarchy, the value of NOA at lower levels in the hierarchy should go down.

III. Proposed Member Access Control Metrics (MACM) for Object-Oriented Systems

In Object-oriented approach, most of the researchers define metrics for classes, objects, methods, coupling, cohesion, reusability, testing, inheritance and polymorphism [10]-[17]. Some of the metrics depicts the behavior of the methods but no such metrics are designed particularly for the Member Access Control which gives total control over the reusability and access of data and methods in the class hierarchy. This section defines Member Access Control Metrics (MACM) for Object-oriented systems.

Manuscript received May 20, 2014; revised August 29, 2014.

R. S. Chhillar and P. Kajla are with the Department of Computer Science and Applications, Maharshi Dayanand University, Rohtak, India (e-mail: chhillar02@gmail.com, pkajla77@gmail.com).

U. Chhillar is with the Computer Science Department, A. I. J. H. M. (PG) College, Rohtak, India (e-mail: chhillarusha01@gmail.com).

N. Kumar is with the Department of Mathematics, Govt. College for Women, Lakhanmajra, Rohtak, India (e-mail: nkhooda1970@gmail.com).

These metrics are further classified into

1) Member Function Access Control Metrics (MFACM)
2) Data Member Access Control Metrics (DMACM)
3) Member Access Control Factor Metrics (MACF)

A. Classes in a System

If a system is denoted as *S* and finite number of classes are referred as $C_i(i = 1 \dots n)$, then *S* is defined as

$$S = \{ C_1, C_2, \dots\dots, C_n\}$$

or

$$S = \sum_{i=1}^{tc} C_i$$

where *tc* is the total number of classes in the system

B. Methods of the Class

For the i^{th} class C_i, Methods of the class are referred as $NM(k = 1 \dots m)$, then

$$NM(C_i) = \{ NM_{i1}, NM_{i2}, \dots\dots, NM_{im}\}$$

C. Attributes of the Class

For the i^{th} class C_i, Attributes of the class are referred as $NA\ (k = 1 \dots m)$, then

$$NA(C_i) = \{ NA_{i1}, NA_{i2}, \dots\dots, NA_{im}\}$$

In object-oriented language, Inheritance is a process of creating a new class from an existing class. While deriving the new classes, the access control specifiers gives the total control over the data members (attributes) and member functions (methods) of the base classes. A derived class can be defined with one of the access specifier, viz. private, public and protected. These access specifiers during inheritance are responsible for reusability of the data members and member functions of the base class.

Considering these access specifiers, number of member functions (Methods) are quantified which belongs to a particular access specifier.

D. Private Methods of the Class

For the i^{th} class C_i, Private Methods of the class are referred as $NM_{Pri}\ (k = 1 \dots m)$, then

$$NM_{pri}(C_i) = \{ NM_{pri_{i1}}, NM_{pri_{i2}}, \dots\dots, NM_{pri_{im}}\}$$

E. Protected Methods of the Class

For the i^{th} class C_i, Protected Methods of the class are referred as $NM_{Pro}\ (k = 1 \dots m)$, then

$$NM_{pro}(C_i) = \{ NM_{pro_{i1}}, NM_{pro_{i2}}, \dots\dots, NM_{pro_{im}}\}$$

F. Public Methods of the Class

For the i^{th} class C_i, Public Methods of the class are referred as $NM_{Pub}\ (k = 1 \dots m)$, then

$$NM_{pub}(C_i) = \{ NM_{pub_{i1}}, NM_{pub_{i2}}, \dots\dots, NM_{pub_{im}}\}$$

So, Member Function Access Control Metrics (MFACM) for the i^{th} class can be defined as

$$MFACM\ (C_i) = NM_{pri}(C_i) + NM_{pro}(C_i) + NM_{pub}(C_i)$$

where C_i = i^{th} class in the system,
$NM_{pri}(C_i)$ = number of private member functions;
$NM_{pro}(C_i)$ = number of protected member functions;
$NM_{pub}(C_i)$ = number of public member functions.

Finally, Member Function Access Control Metrics (MFACM) for the system can be defined as

$$MFACM\ = \sum_{i=1}^{tc} NM_{pri}(C_i) + NM_{pro}(C_i) + NM_{pub}(C_i)$$

where tc = total number of classes in the system,
$NM_{pri}(C_i)$ = number of private member functions;
$NM_{pro}(C_i)$ = number of protected member functions;
$NM_{pub}(C_i)$ = number of public member functions.

Considering these access specifiers, number of data member (Attributes) which belongs to a particular access specifier.

G. Private Attributes of the Class

For the i^{th} class C_i, Private Attributes of the class are referred as $NA_{Pri}\ (k = 1 \dots m)$, then

$$NA_{pri}(C_i) = \{ NA_{pri_{i1}}, NA_{pri_{i2}}, \dots\dots, NA_{pri_{im}}\}$$

H. Protected Attributes of the Class

For the i^{th} class C_i, Protected Methods of the class are referred as $NM_{Pro}\ (k = 1 \dots m)$, then

$$NA_{pro}(C_i) = \{ NA_{pro_{i1}}, NA_{pro_{i2}}, \dots\dots, NA_{pro_{im}}\}$$

I. Public Attributes of the Class

For the i^{th} class C_i, Public Methods of the class are referred as $NM_{Pub}\ (k = 1 \dots m)$, then

$$NA_{pub}(C_i) = \{ NA_{pub_{i1}}, NA_{pub_{i2}}, \dots\dots, NA_{pub_{im}}\}$$

So, Data Member Access Control Metrics (DMACM) for the i^{th} class can be defined as

$$DMACM\ (C_i) = NA_{pri}(C_i) + NA_{pro}(C_i) + NA_{pub}(C_i)$$

where
C_i = i^{th} class in the system,
$NA_{pri}(C_i)$ = number of private data members;
$NA_{pro}(C_i)$ = number of protected data members;
$NA_{pub}(C_i)$= number of public data members.

Finally, Data Member Access Control Metrics (DMACM) for the system can be defined as

$$DMACM\ = \sum_{i=1}^{tc} NA_{pri}(C_i) + NA_{pro}(C_i) + NA_{pub}(C_i)$$

where tc = total number of classes in the system,
$NA_{pri}(C_i)$ = number of private member functions;
$NA_{pro}(C_i)$ = number of protected member functions;

$NA_{pub}(C_i)$ = number of public member functions.

IV. Implementation of Member Access Control Metrics by Using Inheritance

Most of the researchers describe theoretically the concept of inheritance in object-oriented programming. Inheritance results to the reusability of the code. To implement member access control metrics in class hierarchy number of programs illustrating Simple, Multilevel, Multiple, Hierarchical and Hybrid Inheritance are created using object-oriented programming language C++. Then based upon these programs some new metrics are also designed and results are associated. A derived class extends its features by inheriting the properties of another class (base class) and adding features of its own. The declaration of derived class specifies its relationship with the base class in addition to its own features. The access mechanism of the individual members of a class is based on the use of visibility mode as private, public and protected.

A. Case I: Private Inheritance

As Fig. 1, in a privately derived class, the visibility mode is private, in which

- Each public member in the base class is private in the derived class
- Each protected member in the base class is private in the derived class
- Each private member in the base class remains private in the base class and hence not accessible in the derived class.

```
//example of Private Inheritance

class base
{
    private:
        int x;
        voidfun_x();
    protected:
        int y;
        voidfun_y();
    public:
        int z;
        voidfun_z();
};

class derived : private base
{
    private:
        int d;
        void display();
};
```

Fig. 1. Private inheritance.

B. Case II: Protected Inheritance

As Fig. 2, ina protected derived class, the visibility mode is protected, in which

- Each public member in the base class is protected in the derived class
- Each protected member in the base class is protected in the derived class
- Each private member in the base class remains private in the base class and hence not accessible in the derived class.

```
//example of Protected Inheritance

class base
{
    private:
        int x;
        voidfun_x();
    protected:
        int y;
        voidfun_y();
    public:
        int z;
        voidfun_z();
};

class derived : protected base
{
    private:
        int d;
        void display();
};
```

Fig. 2. Protected inheritance.

C. Case III: Public Inheritance

As Fig. 3, ina publically derived class, the visibility mode is public, in which

- Each public member in the base class is public in the derived class
- Each protected member in the base class is protected in the derived class
- Each private member in the base class remains private in the base class and hence not accessible in the derived class.

```
//example of Public Inheritance

class base
{
    private:
        int x;
        voidfun_x();
    protected:
        int y;
        voidfun_y();
    public:
        int z;
        voidfun_z();
};

class derived : public base
{
    private:
        int d;
        void display();
};
```

Fig. 3. Public inheritance.

The Table I shows the visibility scope of private, protected and public inheritance.

TABLE I: VISIBILITY SCOPE

Base Class visibility	Derived Class Visibility		
	Private Derivation	Protected Derivation	Public Derivation
Private	Not accessible	Not accessible	Not accessible
Protected	Private	Protected	Protected
Public	Private	Protected	Public

To determine the scope of reusability the proposed metrics are used quantitatively. Further new metrics are also depicted from the existing metrics.

To determine the reusability scope of members in the class hierarchy a new Member Access Control Factor Metrics for visibility modes are produced.

$$MACFM_{Pri} = \frac{\sum_{i=1}^{tc}(NM_{Pri}(C_i) + NA_{Pri}(C_i))}{\sum_{i=1}^{tc}(NM(C_i) + NA(C_i))}$$

$$MACFM_{Pro} = \frac{\sum_{i=1}^{tc}(NM_{Pro}(C_i) + NA_{Pro}(C_i))}{\sum_{i=1}^{tc}(NM(C_i) + NA(C_i))}$$

$$MACFM_{Pub} = \frac{\sum_{i=1}^{tc}(NM_{Pub}(C_i) + NA_{Pub}(C_i))}{\sum_{i=1}^{tc}(NM(C_i) + NA(C_i))}$$

V. OBSERVATION AND RESULTS

The observations of proposed metric suite are

- MFACM is used to determine total number of methods in a class
- MFACM is also used to determine total number of methods in the whole system
- MFACM also determines the number of private, protected and public member functions in the system.
- DMACM is used to determine total number of attributes in a class
- DMACM is used to determine total number of attributes in the whole system
- DMACM also determines the number of private, protected and public attributes in the system.
- MACFM provides percentage of private member
- MACFM provides percentage of protected member
- MACFM provides percentage of public member

The proposed metric suite results in

- Determining the scope of reuse of code using inheritance.
- Determines effectiveness of private, public and protected members to inherit base class features in the system.
- Evaluating estimates/improves execution time of the system.
- Evaluating estimates/improves size and effort of the system.
- It in turn results in cost effectiveness.

VI. CONCLUSION

We proposed a number of member access control metrics or object-oriented software systems by using inheritance. These metrics are useful for estimating time, cost and effort for object-oriented software development. In future work we will implement these proposed metrics for large and complex object-oriented software systems and may play an important role for controlling complexity and thus improving quality of object-oriented systems.

REFERENCES

[1] S. R. Chidamber and C. F. Kemerer, “Towards a metric suite for object-oriented design,” in *Proc. the Conference on Object-Oriented Programming Systems, Languages and Applications,* ACM Press: NY, 1991, pp. 197-211.

[2] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object orienteddesign,” *IEEE Transactions on Software Engineering*, pp. 476-492, 1994.

[3] S. R. Chidamber and C. F. Kemerer, “Managerial use of metrics for object-oriented software: An exploratory analysis,” *IEEE Transactions on Software Engineering,* vol. 24, issue 8, pp. 629-639, 1998.

[4] V. R. Basili, L. Biand, and W. L. Melo, “A validation of object-oriented design metrics as quality indicators,” *IEEE Transactions on Software Engineering*, vol. 22, pp. 751-761, 1996.

[5] L. C. Braind and S. Morasoa, “Defining and validating measures for object-based high level design,” *IEEE Transactions on Software Engineering,* vol. 25, pp. 722-743, 1999.

[6] W. Li and S. Henry, “Object-oriented metrics that predict maintainability,” *Journal of Systems and Software*, vol. 23, 1993.

[7] M. H. Tang, M. H. Kao, and M. H. Chen, “An empirical study on object-oriented metrics,” in *Proc. 23rd Annual International Computer Software and Application Conference, IEEE Computer Society,* 1999, pp. 242-249.

[8] R. Harrison, S. J. Counsell, and R. V. Nithi, “An evaluation of MOOD set of object oriented software metrics,” *IEEE Trans. Software Engineering*, vol. SE-24, no. 6, pp. 491-496, 1998.

[9] M. Lorenz and J. Kidd, “Object-Oriented software metrics: A practical guide,” PHI, A Pearson Education Company, 1994.

[10] F. B. Abreu, “The MOOD Metrics Set,” in *Proc. the 9th European conference on Object-Oriented Programming, Workshop on Metrics,* Springer: Berlin, 1995.

[11] K. Morris, “Metrics for object oriented software development,” Master thesis, M.I.T., Sloan school of management, Cambridge, MA, 1998.

[12] R. Singh and P. S. Grover, “A new program weighted complexity metric,” in *Proc. International Conference on Software Engg. (CONSEG’97)*, Chennai, India, 1997, pp. 33-39.

[13] K. K. Aggarwal, Y. Singh, and R. Malhotra, “Empirical study of object-oriented Metrics,” *Journal of Object Technology,* vol. 5, pp. 149-173, 2006.

[14] S. Mishra, “An object oriented complexity metric based on cognitive weights,” in *Proc. 6th IEEE International Conference on Cognitive Informatics*, 2007.

[15] R. S. Chhillar and P. Kajla, “Metrics to study constructor in class hierarchy,” in *Proc. National Conference on Advanced Computing Technologies-2013 (NCACT-2013),* vol. 2, pp. 923-926.

[16] R. S. Chhillar, P. Kajlaand, and U. Chhillar, “Developing a nested class complexity metric for nested classes,” presented in 6th International Conference on Computer and Electrical Engineering (ICCEE2013), Paris, October 12-13, 2013

[17] R. S. Chhillar, P. Kajlaand, and U. Chhillar, “Developing a nested class complexity metric for nested classes,” *International Journal of Electrical Energy (IJOEE),* vol. 1, no. 4, pp. 244-248, 2013.

Rajender Singh Chhillaris is working as a professor and the head of Department of Computer Science and Applications, Maharshi Dayanand University (MDU), Rohtak, Haryana, India. He acted as a director of University Institute of Engineering and Technology (UIET), M. D. University, Rohtak from April, 2006 to August 2007 and remained to be the head of Department of Computer Science and Applications, M. D. University, Rohtak earlier also from March 2003 to March 2006.

He also worked as a director of Computer Centre, MDU from 2003 to 2010. He was a member, a monitoring committee of campus wide networking, M. D. University, Rohtak. He obtained his Ph.D. degree in computer science from Maharshi Dayanand University, Rohtak and master’s degree from Kurukshetra University, Kurukshetra.

Dr. Chhillar’s research areas include software engineering, software testing, computer network security, software metrics, component and aspect based metrics, data ware housing and data mining, information and network security and it management. He had published more than 150 publications in International and National journals/ conferences. Professor Chhillar has also authored two books–*Software Engineering: Metrics, Testing and Faults, Excel Books House*, New Delhi; and *Application of Information Technology to Business, Ramesh Books House*, Jaipur. He is senior member of various National and International academic bodies/associations and reviewer of various international journals.

Parveen Kajla is a research scholar in the Department of Computer Scienceand Applications, Maharshi Dayanand University (MDU), Rohtak, Haryana, India. He is a coordinator of PG Courses at Vaish Mahila Mahavidyalya, Rohtak and senior lecturer in Department of Computer Science and Applications, Vaish Mahila Mahavidyalya, Rohtak. He obtained his master’s degree in computer science from Maharshi Dayanand University, Rohtak and M.Phil. degree (computer science) from Chaudhary Devi Lal University (CDLU), Sirsa. His research interests include software engineering focusing on object-oriented and component-based metrics.

Usha Chhillar is working as the head of Department of Computer Science, A.I.J.H.M. PG College, Rohtak, Haryana, India. She obtained her Ph.D. degree in computer science from Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, Haryana, India. She pursued her master degree in computer science from Maharshi Dayanand University (MDU), Rohtak and M.Phil. degree (computer science) from Ch. Devi Lal University (CDLU), Sirsa. She has total more than thirteen years teaching experience. Her research interests include software engineering, object-oriented and component-based software metrics.

Narender Kumar is working as an assistant professor in Mathematics at Govt. College for Women, Lakhanmajra, Rohtak, Haryana, India. He obtained his M.Phil. degree in mathematics from Kurukshetra University, Kurukshetra.