Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD

Nasib S. Gill
Professor & Head, Department of Computer Science & Applications
Maharshi Dayanand University (India)

Sunil Sikka
Research Scholar, Department of Computer Science & Applications
Maharshi Dayanand University (India)

Abstract— **Reuse and reusability are two major aspects in object oriented software which can be measured from inheritance hierarchy. Reusability is the prerequisite of reuse but both may or may not be measured using same metric. This paper characterizes metrics of reuse and reusability in Object Oriented Software Development (OOSD). Reuse metrics compute the extent to which classes have been reused and reusability metrics computes the extent to which classes can be reused. In this paper five new metrics namely- Breadth of Inheritance Tree (BIT), Method Reuse Per Inheritance Relation (MRPIR), Attribute Reuse Per Inheritance Relation (ARPIR), Generality of Class (GC) and Reuse Probability (RP) have been proposed. These metrics help to evaluate reuse and reusability of object oriented software. Four extensively validated existing object oriented metrics, namely- Depth of Inheritance Tree (DIT), Number of Children (NOC), Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF) have been selected and investigated for comparison with proposed metrics. All metrics can be computed from inheritance hierarchies and classified according to their characteristics. Further, metrics are evaluated against a case study. These metrics are helpful in comparing alternative inheritance hierarchies at design time to select best alternative, so that the development time and cost can be reduced.**

Keywords- Inheritance Hierarchy, Inheritance Metrics, Reuse, Reusability

I. INTRODUCTION

Software reuse is the use of existing artifacts to create new software and reusability is the degree to which the artifacts can be reused. Software reuse reduces the development efforts and increases the quality of software. It is well recognized by both researchers and software industries that Object Oriented Software Development (OOSD) approach is capable of developing software by reusing existing classes and for developing reusable classes. OOSD reuse the existing classes to fulfill the current requirements of customer and also promise to develop the reusable classes to fulfill the future or changing requirements of customer. OOSD supports reuse in three ways (1) Verbatim reuse through instantiation and use of previously defined classes (2) Generic reuse through generic templates (3) Leveraged reuse through inheritance [1]. Among these, inheritance is the foremost technique of reuse. Inheritance is the relationship among classes, wherein an object in a class acquires characteristics from one or more other classes [2]. By organizing classes into “classification hierarchy”, inheritance gives an extra dimension to the encapsulation of the abstract data types because it enables classes to inherit attributes and methods from other classes [3].

Different inheritance hierarchies may be designed to implement inheritance for a problem therefore, it becomes essential to assess inheritance hierarchy from reuse and reusability point of views for increasing its potential benefits. Many object oriented metrics are available in literature for assessing inheritance hierarchy but these metrics do not distinguish reuse and reusability. Reuse and reusability metrics evaluates two different aspects of inheritance hierarchy. Reuse metrics answers the question “What is the amount of reuse among classes in the software?” and reusability metrics answers the question “Whether the classes are reusable in future”. This paper investigates four existing metrics- Depth of Inheritance Tree (DIT), Number of Children (NOC), Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF) and also proposes five new metrics- Breadth of Inheritance Tree (BIT), Method Reuse Per Inheritance Relation (MRPIR), Attribute Reuse Per Inheritance Relation (ARPIR), Generality of Class (GC) and Reuse Probability (RP). All inheritance based metrics are classified into two categories: Reuse Based Metrics (RBM) and Reusability Prediction Metrics (RPM). RBM are further classified into two categories- Reuse Indicator Metrics (RIM) and Reuse Estimation Metrics (REM).

Rest of the paper is organized in six sections. Section 2 presents the related work and motivation of this research work. Section 3 explains the concept of reuse and reusability. Section 4 presents the taxonomy of inheritance based metrics, investigates the four existing inheritance based metrics and proposes five new inheritance based metrics. Section 5 conducts an analysis of metrics by applying metrics on two alternative inheritance hierarchies of same problem and interprets the result. Finally Section 6 presents conclusion and future directions.

II. RELATED WORK AND MOTIVATION

Since the proposal of six object oriented metrics by Chidamber and Kemerer's[4] many metrics have been proposed by various researchers. Many of the work focus on metrics of reuse and reusability. Abreu et al. [5] define many metrics among them two metrics- MIF and AIF measures the reuse using inherited methods and attributes. Sheldon et al. [6] propose metrics for understandability and modifiability of a class inheritance hierarchy and compare the proposed metrics with Chidamber & Kemerer's and Handerson-Sellers's metrics. Bhatia et al. [7] define reusability of a class as a function of DIT, NOC and Coupling Between Object classes (CBO). Authors state that reusability in whole class diagram is equal to the reusability of class having maximum reusability. DIT and NOC have positive effect on reusability where as CBO has negative effect on reuability of a class. Gandhi et al. [8] propose four metrics (Number of Template Children (NTC), Depth of Template Tree (DTT), Method Template Inheritance Factor (MTIF) and Attribute Template Inheritance Factor (ATIF)) based on template and state that proposed metrics as reusability metrics. Sandhu et al. [9] define reusability in terms of tuned version of CK-metrics suite and propose a neuro-fuzzy based model for automatic identification reusability of object-oriented software components. Rajnish et al. [10] propose three metrics namely- Derive Base Ratio Metric (DBRM), Average Number of Direct Child (ANDC) metric and Average Number of Indirect Child (ANIC) metric for measuring class inheritance tree. Authors compare proposed metrics with metrics proposed by Sheldon et al.[6] and Henderson-Seller. Suri et al. [11] define reusability of a component in terms of its independency. Higher independency is treated as indicator of more reusability.

Most of the work in this filed is related to proposal of inheritance based metrics and models. For better understanding of these metrics proper classification of metrics required. This triggers the approach used in this paper to differentiate the inheritance based metrics and to propose five new metrics.

III. EXPLORING REUSE AND REUSABILITY

Software reuse is the key activity in software development for improving the software productivity and quality by using existing software artifacts or knowledge to develop the new software. Effective software reuse can be only achieved through systematic quantified process. Effective reuse pays maximum productivity and quality benefits at fewer cost. Reusability is the property of any artifact that makes it reusable. Reusability of any artifact can be defined as the degree to which it can be reused [12]. Morisio et al. defines software reuse as a systematic practice of developing software from a stock of building blocks[13]. Building blocks for reuse in OOSD are classes. OOSD includes both reuse of the existing classes as well as design of new classes that can be reused in future. Reuse is achieved by establishing various relationships among classes such as inheritance or composition. Features of class such as high generality level (i.e. less application specificity) and less coupling increases its reusability. Introducing reusability features in the classes must be the goal of inheritance hierarchy at design time.

Concept of reuse and reusability can be understood from following example. Suppose a web based software development company wants to analyze the optimization of its sites due to code, content and user interface. Code optimization deals with programming techniques to load and index the pages quickly. Effective contents of website deal with the keywords which can drive the visitors to the website. Good user interface makes the website interactive and user friendly. It is decided to design three classes: CodeOptimizationAnalyzer, ContentOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer to measure the performance of website due to code, content and user interface respectively. It is found that some of the features of these three classes are same therefore, by following the basic principle of designing of inheritance hierarchy that common features should be contained in superclass, one more class OptimizationAnalyzer is designed which contains that common features. CodeOptimizationAnalyzer, ContentOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer classes are derived from OptimizationAnalyzer class as shown in Fig. 1.

OptimizationAnalyzer class is reused by CodeOptimizationAnalyzer, ContentOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer classes. This inheritance hierarchy is reasonable as per the current requirements. However, in future the company may analyze the optimization from more different point of views such as graphics optimization, search engine optimization , database optimization etc. Therefore, the inheritance hierarchy should be more generalized so that more classes can be inherited in future. Another alternative of same problem is show in Fig. 2. Three new classes: PerformanceOptimizationAnalyzer, BusinessOptimizationAnalyzer and DesignOptimizationAnalyzer are introduced to increase the reusability of inheritance hierarchy. Newly added classes in alternative-2 generalize the optimization analyzers and contain the common features of all the optimization analyzers which fall into same categories. Altternative-2 provides more opportunity for reuse in future.

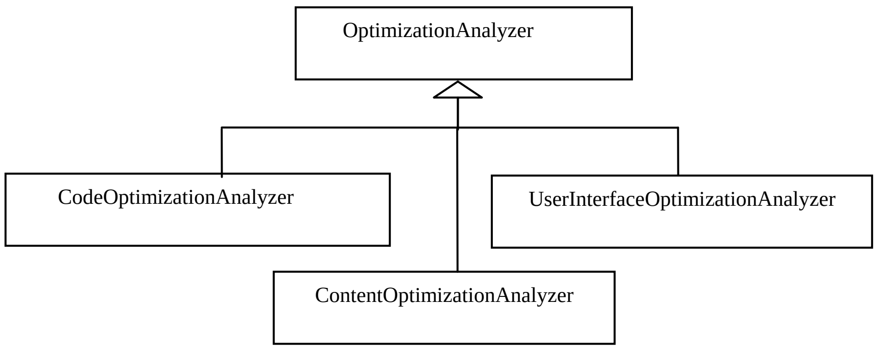
OptimizationAnalyzer
CodeOptimizationAnalyzer
UserInterfaceOptimizationAnalyzer
ContentOptimizationAnalyzer

Figure 1. Inheritance Hierarchy for Website Optimization Problem - Alternative1

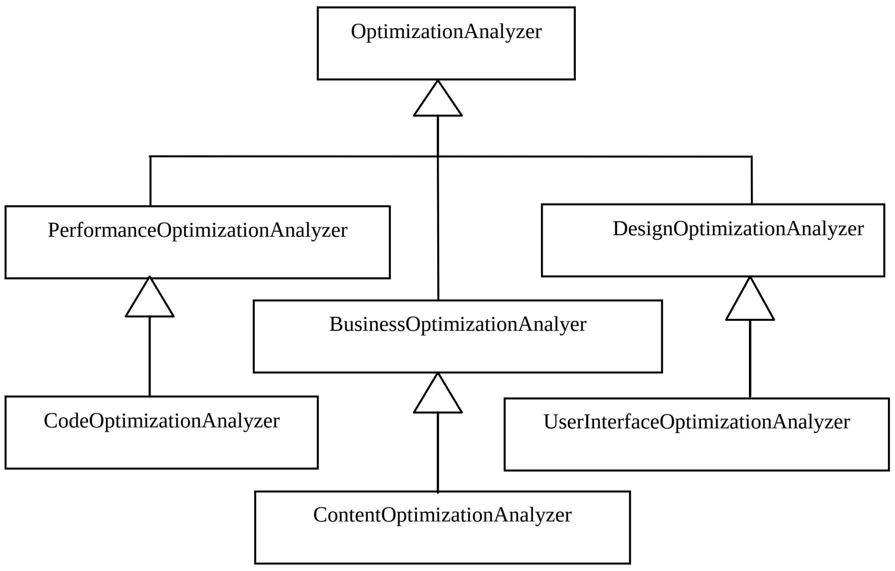
OptimizationAnalyzer
PerformanceOptimizationAnalyzer
DesignOptimizationAnalyzer
BusinessOptimizationAnalyer
CodeOptimizationAnalyzer
UserInterfaceOptimizationAnalyzer
ContentOptimizationAnalyzer

Figure 2. Inheritance Hierarchy for Website Optimization Problem - Alternative2.

IV. INHERITANCE HIERARCHY BASED METRICS

Designing inheritance hierarchy must be supported by reuse and reusability metrics to measure its effectiveness. Metrics can be used as indication of improvement of reuse and reusability in inheritance hierarchy. Reuse metrics are used to compute the amount of reuse among classes and reusability metrics are used to predict the extent to which classes can be reused. For better understanding of inheritance based metrics it is necessary to classify them. This paper categorizes inheritance based metrics into two categories- RBM and RPM as shown in Fig. 3. RBM is further classified into two categories RIM and REM. RIM of a class just gives the idea of reuse amount whereas REM actually computes the amount of reuse on the basis of method/attribute reused. RPM of a class computes the extent up to which it can be reused in future. RPM computes reusability only on the basis dimension and characteristics of inheritance tree. RPM don’t take account of actual number of method and attribute inherited in the computation.

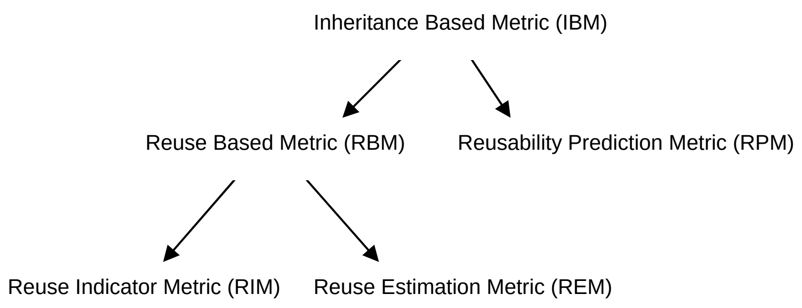
Inheritance Based Metric (IBM)
Reuse Based Metric (RBM)
Reusability Prediction Metric (RPM)
Reuse Indicator Metric (RIM)
Reuse Estimation Metric (REM)

Figure 3. Taxonomy of Inheritance Based Metrics.

Inheritance based metrics investigated and proposed in this paper are listed in Table-1.

TABLE 1. REUSE AND REUSABILITY METRICS

Metric Name	Acronym	Source	Category
Depth of Inheritance Tree	DIT	CK Metrics[4]	RIM+RPM
Number of Children	NOC	CK Metrics[4]	RIM
Method Inheritance Factor	MIF	MOOD Metrics[5,15]	REM
Attribute Inheritance Factor	AIF	MOOD Metrics[5,15]	REM
Breadth of Inheritance Tree	BIT	Proposed	RIM
Method Reuse Per Inheritance Relation	MRPIR	Proposed	REM
Attribute Reuse Per Inheritance Relation	ARPIR	Proposed	REM
Generality of Class	GC	Proposed	RPM
Reuse Probability	RP	Proposed	RPM

A. EXISTING METRICS INVESTIGATED

Many inheritance metrics are available in literature out of which four metrics which have been widely validated in literature are selected for further investigation and comparison with proposed metrics. Selected metrics are described as follows.

1) Depth of Inheritance Tree (DIT)

DIT of a class computes the maximum path length from that class to the root of the inheritance tree. Higher the DIT of a class, greater the number of methods it is likely to inherit, i.e. higher reuse. The class having higher DIT is less reusable as compare to class having lower DIT due to class at higher depth is more specialized as compare to class at lower depth in inheritance hierarchy. More specialized class provides less opportunity to inherit it. Therefore, DIT is RIM as well as RPM metric.

2) Number of Children (NOC)

NOC of a class is the number of immediate sub-classes of that class. Greater number of subclasses/children is indicator of greater reuse. NOC indicates the reuse among classes therefore, it is RIM metric.

3) Method Inheritance Factor (MIF)

MIF computes the system level reuse in terms of total number of methods inherited and declared in the classes. It is computed as follows
MIF = (total number of methods inherited in all classes) / (total number of methods declared and inherited in all classes)
Higher the value of MIF more is the amount of reuse in the software. Zero value of MIF indicates there is no method inheritance in the classes, this may be due to the scope of methods don’t permits method inheritance. MIF measures the amount of reuse in the inheritance hierarchy therefore, it is REM.

4) Attribute Inheritance Factor (AIF)

Like MIF, AIF is also system level metric and computes reuse in terms of total number of attributes inherited

and declared in the classes. It is computed as follows
AIF = (total number of attribute inherited in all classes) / (total number of attributes declared and inherited in all classes)
AIF is also categorized in REM category.

B. NEW METRICS PROPOSED

Following are five proposed metrics.

1) Breadth of Inheritance Tree (BIT)

Breadth of inheritance tree is equal to the total number of leaf nodes in the inheritance hierarchy

BIT= Number of Leaf Nodes

For example BIT of inheritance hierarchy given in Fig. 4 is 6 because its has six leaf nodes (E, I, J, C, G, H)

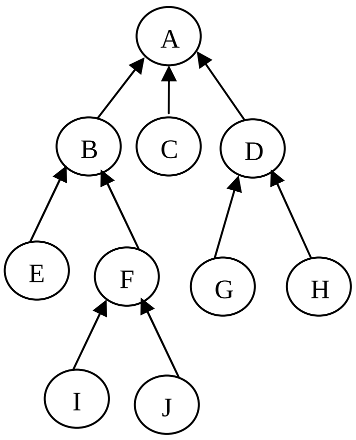

Figure 4. Inheritance Hierarchy for computing BIT.

BIT is indicator of reuse. Higher BIT means higher number of methods/attributes reused in the derived class. It don't computes actual amount of inheritance but only indicates the reuse therefore, it is classified as RIM metric.

Comparison with existing Metrics
As compare to NOC metric which computes number of immediate sub classes of a class BIT measures the breadth of whole inheritance tree. As compare to DIT it gives another dimension to inheritance tree i.e. breadth. As compare to MIF and AIF, it doesn't computes the actual number of method and attribute inherited therefore, categorized into RIM category.

2) Method Reuse Per Inheritance Relation (MRPIR)

MRPIR computes the total number of methods reused per inheritance relation in the inheritance hierarchy. It applies on whole inheritance hierarchy in the system. It can be computed as follows

$$\text{MRPIR} = \frac{\sum_{k=1}^{r} MI_k}{r}$$

Where r= Total number of inheritance relationships
MI_k=No. of methods inherited through kth inheritance relationship
If same method is inherited through different inheritance relationships then it is computed separately in each relationship.

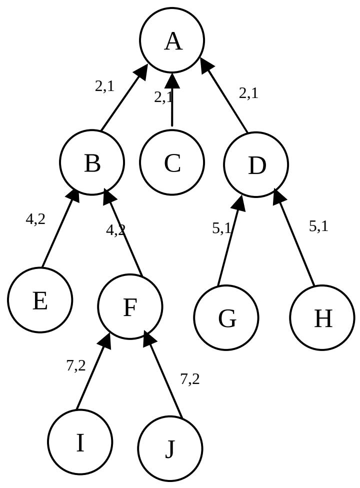

Figure 5. Weighted Inheritance Hierarchy for computing MRPIR

Consider an inheritance hierarchy given in Fig. 5. Weight of each arc is the number of methods and attributes inherited from base class to derived class. First number of weight represents number of methods inherited and second number of weight represents number of attributes inherited. MRPIR of this inheritance hierarchy is (2+2+2+4+4+5+5+7+7)/9=4.22. This metrics is classified in REM category because it computes the amount of reuse by taking account of number of methods inherited in the inheritance hierarchy.

Comparison with existing metrics
DIT and NOC are two dimensions of inheritance hierarchy and are useful as indicator of reuse. However, MRPIR actually computes average number of method reused in the inheritance hierarchy. It gives clearer picture of reuse due to inheritance. As compare to MIF which considers method declared and inherited in all classes, MRPIR considers only reused methods and inheritance relationships only.

3) Attribute Reuse Per Inheritance Relation (ARPIR)

It computes the total number of attributes reused per inheritance relation in the inheritance hierarchy. Like MRPIR It is also categorized into REM category.

$$ARPIR = \frac{\sum_{k=1}^{r} AI_k}{r}$$

Where AI_k=No. of attributes inherited through kth inheritance relationship
For example ARPIR of inheritance hierarchy given in Fig. 5 is (1+1+1+2+2+1+1+2+2)/9=1.44.

Comparison with existing metrics
ARPIR is similar to MRPIR and can be compared with DIT, NOC and AIF in same way as MRPIR is compared with DIT, NOC and MIF.

4) Generality of Class (GC)

Generality of a class is the measure of its relative abstraction level. Higher the generality of a class more it is likely to be reused. GC can be computed as follows

$$GC = \frac{a}{al}$$

Where a = Abstraction level of the class
al = Total number of possible abstraction levels

Higher the value of GC of a class means higher reusability. For example if OptimizationAnalyzer class has three abstraction level (OptimizationAnalyzer, DesignOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer) and the abstraction level of OptimizationAnalyzer, DesignOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer is 3 , 2 and 1 respectively then their GC is 3/3,2/3 and 1/3 respectively. OptimizationAnalyzer class is having the highest value of GC therefore; it is more reusable as compared to other two classes.

Comparison with existing Metrics
DIT metric considers only depth of a class in inheritance tree and takes higher depth as indicator of higher reuse. DIT does not consider characteristics of the class whereas GC considers the generality of the class a feature of reusability. However the relationship may exist between GC and DIT metric. A class with a small DIT has much potential for reuse it tends to be a general abstract class[14]. As classes at higher depth are more specific as compare to the classes at lower depth therefore, higher depth indicates less abstraction level.

5) Reuse Probability (RP)

It is the probability of reusing classes in the inheritance hierarchy. It can be computed as follows

$$RP = \frac{N_i - N_{\lg}}{N}$$

Where

N_i = Total number of classes that can be inherited

N_{lg} = Total number of classes that can be inherited but having lowest possible generic level
N = Total number of classes in the inheritance hierarchy

The final/sealed classes can’t be inherited. The class having lowest generic level is most specialized class and assumed non inheritable. Higher the number of such classes in the software lower is the probability of reuse therefore, less reusability.

In best case RP=1, in this case all the classes are inheritable and their generic level allow them to inherit. In worst case RP=0, in this case all classes are non inheritable and having lowest generic level. Higher probability indicates more reusability of classes in inheritance hierarchy therefore, it is categorized into RPM category.

Comparison with existing Metrics
GC is depth and breadth independent metric. It computes reuse probability by taking account of number of inheritable and non inheritable classes only. As it is RPM metric it do not need to compute method/attributed inherited and declared like in MIF and AIF metric.

V. CASE STUDY

One case study is designed to assess the metrics discussed in Section 4. This case study is small and simple but shows the significance of metrics. The case study presents two alternative inheritance hierarchies of software for educational institute as shown in Fig. 6 and Fig. 7. It is hypothesized that second alternative have more reuse and reusability as compare to first alternative. Generalization scale for classes is shown in Fig. 8. It is assumed that all methods are inheritable and attributes are non inheritable in both the alternatives. Results are presented in Table-2 and Table-3. In Table-2 class number 1, 2, 3, 4 and 5 represents Student, Bcastudent, Mcastudent, Bbastudent and Mbastudent respectively. In Table-3 class number 1, 2, 3, 4, 5, 6 and 7 represents Student, ITstudent, Mngstudent, Bcastudent, Mcastudent, Bbastudent and Mbastudent respectively.

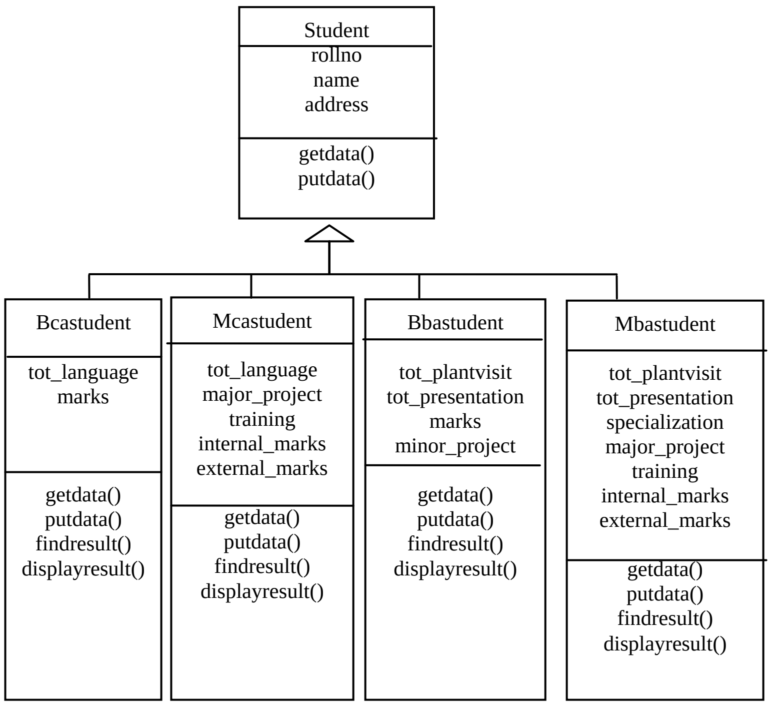

Figure 6. Inheritance Hierarchy for Educational Institute - Alternative1.

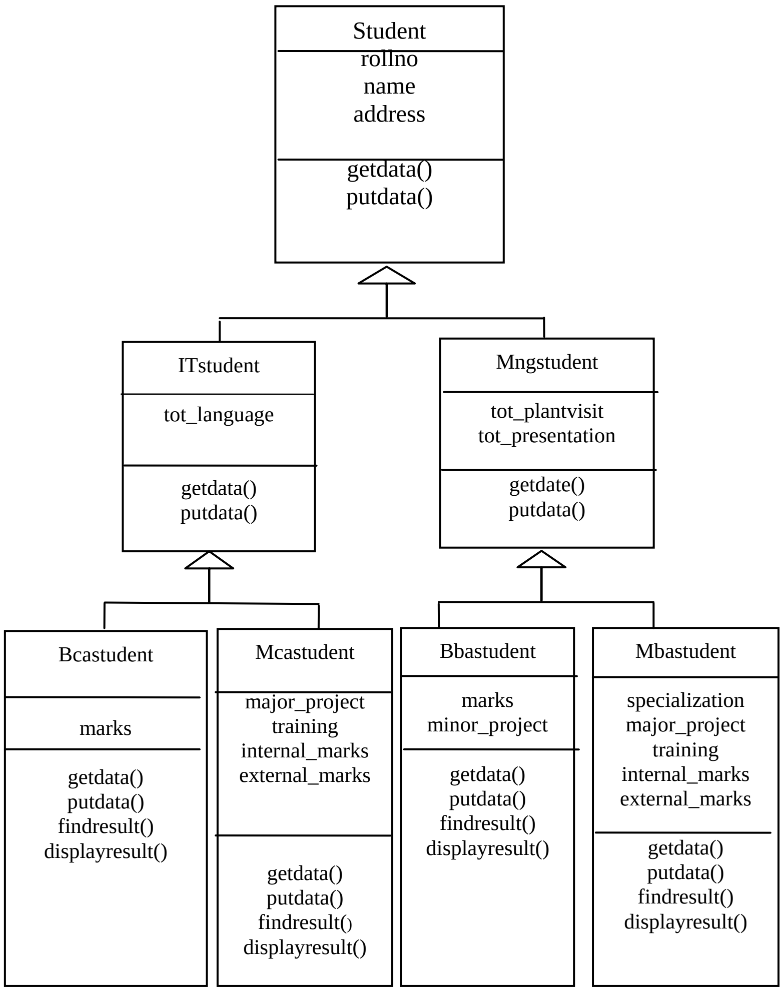

Figure 7. Inheritance Hierarchy for Educational Institute – Alternative2.

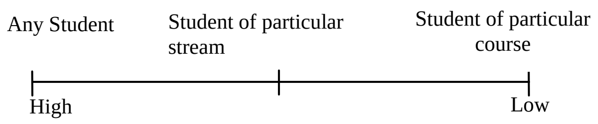

Figure 8. Generalization scale for Student Class

TABLE 2. RESULTS OF ALTERNATIVE-1

Metric	**Class No.**				
	1	**2**	**3**	**4**	**5**
DIT	0	1	1	1	1
NOC	4	0	0	0	0
GC	1	0.33	0.33	0.33	0.33
MIF	0.31				
AIF	0				
BIT	4				
MRPIR	2				
ARPIR	0				
RP	0.2				

TABLE 3. RESULTS OF ALTERNATIVE-2

Metric	**Class No.**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
DIT	0	1	1	2	2	2	2
NOC	2	2	2	0	0	0	0
GC	1	0.67	0.67	0.33	0.33	0.33	0.33
MIF	0.35						
AIF	0						
BIT	4						
MRPIR	3.33						
ARPIR	0						
RP	0.43						

Discussion on Results

Maximum DIT of alternative-2 is higher than alternative-1 therefore, indicates higher reuse. BIT of both the alternative is same however alternative-2 has more number of inherited methods as compare to alternative-1.Maximum NOC of alternative-1 is higher as compare to alternative-2 but overall total number of subclasses in alternative-1 is 4 which is less than total number of subclasses of alternative-2.MIF of alternative-2 is higher as compare to alternative-1. Therefore, higher amount of methods are reused in alternative-2. MRPIR of alternative-2 is higher MRPIR of alternative -1. Therefore, alternative-2 has more reuse as compare to alternative-1. AIF and ARPIR of both alternatives are 0 because no attribute inheritance is considered. RP of alternative-2 is higher than as compare to RP of alternative-1. Therefore, alternative-2 is more reusable. Classes of alternative-1 have only two distinct values of GC (1, 0.33). Whereas classes of alternative-2 have three different values of GC (1, 0.67, 0.33). Average value of GC alternative-2 (0.52) which is higher than average value of GC (0.46) of alternative-1. Therefore, alternative-2 is more reusable.

VI. CONCLUSION AND FUTURE DIRECTIONS

This paper assesses inheritance hierarchy from two different views i.e. reuse and reusability, which are helpful for increasing reuse and reusability and comparing two alternative inheritance hierarchies of same problem. Taxonomy of inheritance based metrics is given for its better understanding. Four existing inheritance based metrics (DIT, NOC, MIF and AIF) have been investigated and five new metrics (BIT, MRPIR, ARPIR, GC and RP) have been proposed for assessing reuse and reusability. Analysis of metrics on a case study shows that these metrics are helpful for assessing reuse and reusability. This study is supported by simple and small case study however the same study can be replicated empirically with industrial projects to generalize results.

REFERENCES

[1] Santhi Karunanithi, James M Bieman “Candidate Reuse Metrics For Object Oriented and Ada Software” Proc. IEEE International Software Metrics Symposium, pp. 120-128, May 1993.

[2] Dr. Linda H.Roshanberg “Applying and Interpreting Object Oriented Metrics”. In Software Technology Conference (April 1998) http://satc.gsfc.nasa.gov/support/STC_APR98/apply_oo/apply_oo.html

[3] Dr. Kadhim M.Breesam “Metrics for Object-Oriented Design Focusing on Class Inheitance Metric” IEEE 2nd International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX’07) 2007.

[4] Shyam R.Chidamber, Chris F.Kemerer “A Metrics Suite For Object Oriented Designs” IEEE Transaction on Software Engineering. Vol.20 No.6, June 1994.

[5] F. B. Abreu, R. Carapuça(1994) "Object-Oriented Software Engineering: Measuring and Controlling the Development Process" Proceedings of the 4th International Conference on Software Quality, McLean, Virginia, USA, October 1994.

[6] Frederick T.Sheldon, Kshamta Jerath and Hong Chung "Metrics for Maintainability of Class Inheritance Hierarchies",Journal of Software Maintenance and Evolution: Research and Practice, Vol. 14, pp. 1-14, 2002.

[7] Pradeep Kumar Bhatia, Rajbeer Mann, “An Approach to Measure Software Reusability of OO Design” Proceedings of 2nd National Conference on Challenges & Opportunities in Information Technology (COIT-2008) RIMT-IET, Mandi Gobindgarh. March 29, 2008

[8] Parul Gandhi & Pradeep Kumar Bhatia “Reusability Metrics for Object-Oriented System: An Alternative Approach” International Journal of Software Engineering (IJSE), Volume (1): Issue (4) December 2010.

[9] Parvinder S. Sandhu and Harpreet Kaur “A Reusability Evaluation Model for OO-Based Software Components”. International Journal of Computer Science Volume 1 Number 4, 2006, pp.259-264.

[10] Kumar Rajnish, Arbind Kumar Choudhary and Anand Mohan Agrawal “Inheritance Metrics for Object-Oriented Design” International Journal of Computer Science & Infomation Technology, Volume 2, Number 6, December 2010.

[11] P. K. Suri and Neeraj Garg “ Software Reuse Metrics: Measuring Component Independence and its applicability in Software Reuse” IJCSNS International Journal of Computer Science and Network Security, Vol.9 No.5, May 2009.

[12] Frakes, William and Terry, Carol “Software Reuse: Metrics and Models”; ACM Computing Surveys, 28, 2 (1996), pp. 415-435.

[13] Maurizio Morisio, Michel Ezran, Colin Tully “Success and Failure Factors in Software Reuse” IEEE Transactions on Software Engineering, Volume 28 Number 4, pp: 340–357,2002.

[14] Dr. E. Chandra 1, P. Edith Linda “Class Break Point Determination Using CK Metrics Thresholds” Global Journal of Computer Science and Technology Vol.10 Issue 14 (Ver.1.0) November 2010.

[15] K.K.Aggarwal, Yogesh Singh, Arvind Kaur, Ruchika Malhotra “Empirical Study of Object-Oriented Metrics” Journal of Object Technologies, vol. 5 no.8 November-December 2006, pp 149-173.

AUTHORS PROFILE

Dr. Nasib S. Gill is currently working as Professor & Head, Department of Computer Science & Applications, Maharshi Dayanand University, Rohtak, Haryana (India). He has more than 20 years of experience in teaching and research. He is the recipient of Commonwealth Fellowship Award availed at Brunel University, West London (United Kingdom) for the year 2001-2002. His major current research interests include designing and development of Component-Based Metrics and Software Complexity Metrics.

Sunil Sikka is a research scholar with the Department of Computer Science & Applications, Maharshi Dayanand University, Rohtak, Haryana (India). His area of research is Object Oriented Software Measurement. His other area of interest includes Software Engineering, Artificial Intelligence and Object Oriented Programming.