Evaluating Inheritance and Coupling Metrics

Sonia Chawla [#1], Dr. Rajender Nath [*2]

1 M.Tech., Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, Haryana, India.

2 Professor, Department of Computer Science and Applications, Kurukshetra University, Kurukshetra , Haryana, India.

***Abstract*—Software metrics helps in estimating the quality of software. To meet the customer's demands, there is a competence in the software industries for the best quality product. Some metrics can be applied in the early stages of product development that helps in eliminating the complexity at later stages. This paper helps in predicting software quality with the help of object oriented metrics.**

***Keywords*— Software quality, object-oriented metrics, quality attributes, complexity.**

1. Introduction

Measurement is required to assess quality and improvements in performance of the software products. Software industries are striving to improve productivity and quality of software to meet ever increasing demands of users. Metrics give information regarding the status of an attribute of the software and help to find opportunities of improvements in the software. Object oriented metrics measures the characteristics of object oriented designs. These measures allow the designer to access the software early in the process making changes which reduces complexity and improve the capability of the product. Quality of a software system can be predicted by evaluating the attributes of software with the help of metrics. In this paper, an attempt is made to use object oriented metrics to predict software quality. Rest of the paper is organized as follows: Section 2 gives an overview of the existing studies in object oriented metrics. Section 3 formulates the problem. Section 4 tests effectiveness of inheritance and coupling metrics with the help of a sample project. Section 5 presents conclusion.

2. Related Work

Among all the metric suites, Chidamber and kemerer [4] metric suite is the most referenced one. They defined six metrics-Weighted Methods per Class (WMC), Depth of Inheritance (DIT), Coupling Between Objects (CBO), Response For a Class (RFC), Lack of Cohesion in Methods (LCOM), Number of Children (NOC). Various studies have been done on their validation by many researchers. Li et al. [10] validated CK metrics using statistical analysis on two commercial systems. Five of the six metrics (except CBO) helped predict maintenance effort and they proposed many other metrics to evaluate maintainability like Data Abstraction Coupling (DAC), Message Passing Coupling (MPC), Number of methods (NOM) and two size metrics. Lorenz and Kidd [6] divided metrics into project level and design level metrics. They divided design metrics into four categories: size, internals, externals, inheritance. MOOD (Metrics for Object Oriented Design) metric set proposed by Abreu et al. [2] measure the object oriented mechanisms such as inheritance (Method Inheritance Factor, Attribute Inheritance Factor), encapsulation (Method Hiding Factor, Attribute Hiding Factor), polymorphism (Polymorphism Factor), message passing (Coupling Factor). Rosenberg [9] proposed nine metrics to evaluate attributes like efficiency, complexity, reusability, testability, understandability. Three of them were the traditional metrics viz. LOC, Comment percentage, Cyclomatic Complexity and the rest were same as those of CK metrics.

W.Li et al [7]described another Object-Oriented syntactic metrics suite that addressed certain shortcomings in CK's metrics suite. They proposed certain metrics-Number of Ancestor Classes (NAC), Number of Descendant Classes (NDC), Number of Local Methods (NLM), Class Method Complexity (CMC) Coupling Through Abstract Data Types (CTA) and Coupling Through Message Passing (CTM). Bansiya J. et al. [3] defined Quality Model for Object Oriented Design (QMOOD) metrics which provided formulae to compute quality attributes.

3. Problem Formulation

There are certain factors to assess the quality of the software. McCall, Richards and Walters [8] divided product into three aspects: revision, transition and operation and proposed quality factors based on this

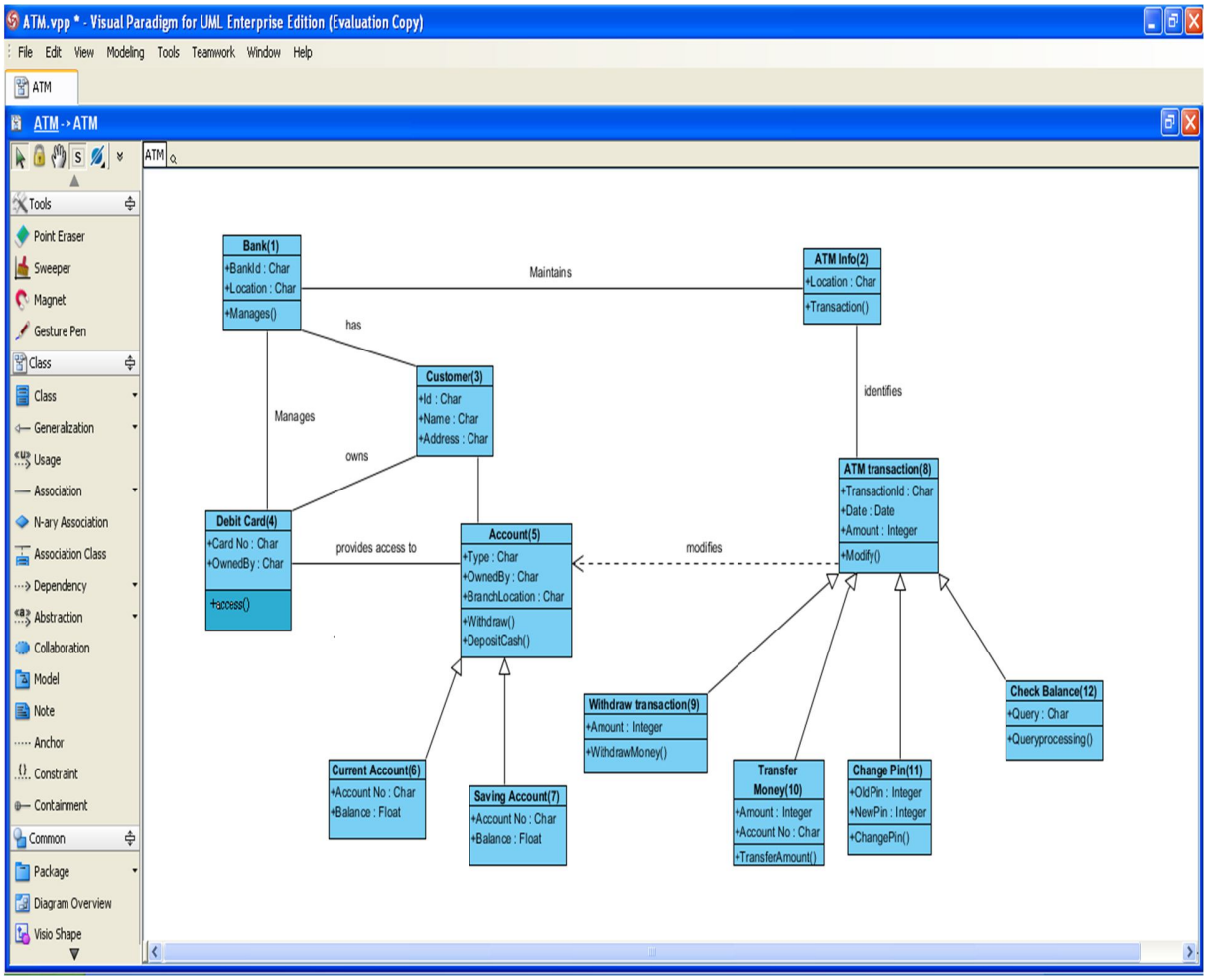

Figure 1 ATM Class Diagram

.

categorization. ISO 9126 standard also defines six attributes of quality- functionality, reliabilty, usability, efficiency, maintainability, portability.

Faults must be checked at the early stages of the product development as it saves time and cost. It also helps in reducing complexity at early stages, which also affects other quality attributes. If complexity is less it implies that the effort required to test a program would be less and product would be more reliable. The object oriented paradigm provides strong support for software reuse. Inheritance helps in reusability as well as it affects other factors like complexity, testability etc.

These metrics are tested on ATM class diagram (selected for this study) in order to evaluate the effectiveness of these metrics. The following inheritance metrics (DIT, MIF, AIF, NOC, MFA, NAC, NDC) and coupling metrics (DAC, RFC, CBO, CF) are identified to compute complexity, reliability, testability and reusability.

4. Testing effectiveness of Inheritance and Coupling Metrics

The metrics chosen for evaluating the class diagram are divided into two categories viz. Inheritance Metrics and Coupling Metrics. Both Class level as well as system level metrics from different metric suites have been considered. Metric values are analyzed to predict the software quality attributes like reusability, testability, reliability.

4.1. Inheritance Metrics

Inheritance promotes reusability but it should be used in a proper range so that the project doesn't become complex. In this section seven inheritance metrics are considered for estimating the software quality.

Attribute Inheritance Factor (AIF): The AIF is the ratio of sum of the inherited attributes in all classes of the system to the total number of available attributes for all classes. It is a system level metric and measures the extent of attribute inheritance in a

METRIC	CLASS	1	2	3	4	5	6	7	8	9	10	11	12
AIF	ATTRIBUTES INHERITED ($A_i(C_i)$)	0	0	0	0	0	3	3	0	3	3	3	3
	ATTRIBUTES AVAILABLE($A_A(C_i)$)	2	1	3	2	3	5	5	3	4	5	5	4
MIF	METHODS INHERITED ($M_i(C_i)$)	0	0	0	0	0	2	2	0	1	1	1	1
	METHODS AVAILABLE($M_A(C_i)$)	1	1	0	1	2	2	2	1	2	2	2	2

Table 1: System Level Inheritance Metrics

system. The number of attributes inherited and attributes available are computed for each class and shown in the table 1.Mathematically AIF is computed as:

$$AIF=\sum A_i(C_i)/\sum A_a(C_{i)},$$

where i is from 1 to total number of classes.

Value of numerator and denominator comes out to be 18 and 42 respectively. Hence, the value of AIF comes out to be 0.42 as follows:

AIF = 18/42
= 0.42

Interpretation: For the class diagram shown in figure 1, the computed value of AIF is 0.42. According to one source, the acceptable range of AIF is from 0 to 48% [5]. The value computed is in this range and this shows that the project is not complex. As the complexity is lesser, it implies more reliability. It further implies that the testability effort required will also be lesser.

Method Inheritance Factor (MIF): The MIF is the ratio of sum of the inherited methods in all classes of the system to the total number of available methods for all classes. MIF is a system level metric. The values for number of methods inherited and methods available are computed for each class and shown in the table 1. Mathematically MIF is computed as:

$$MIF =\sum M_i(C_i) / \sum M_a(C_i)$$

where i is from 1 to total number of classes.
Value of numerator and denominator comes out to be 8 and 18 respectively. Hence the value of MIF comes out to be 0.44 as follows:

MIF=8/18
=0.44

Interpretation: MIF acceptable range is within 20% to 80% [5]. For the class diagram in figure 1, the computed value of MIF is 0.44 which is in the acceptable range. This shows that the inheritance used will not make the system complex. So, the reliability will be more and the required testability effort will be lesser.

Depth of Inheritance Tree (DIT): DIT is the maximum length from node to the root of the tree. It is a class level metric. Values are normalized for each class and are shown in table 2.

Interpretation: Sum of normalized values of DIT of all the classes comes out to be 0.54 and the maximum possible value could have been 12. As compared to the maximum value, it is very small. So, in this project the use of inheritance is very less which implies that the complexity and reusability of project will be lesser. It further implies that as complexity is less, the amount of effort required to test will also be lesser. Therefore, more depth should be avoided for a project to be reliable and effective.

Number of Children (NOC): NOC is defined as the number of immediate subclasses subordinated to a class in the class hierarchy. Values are normalized w.r.t. 11 for each class as shown in table 2. After the summation of the values for all classes, the total value comes out to be 0.54.

Interpretation: Value computed for NOC i.e. 0.54 is very less as compared to the maximum value of 12. This shows that reusability is less due to low inheritance and the amount of testing required will be less due to less children. Therefore, it shows that the project is not complex and hence more reliable.

METRIC	VALUES	CLASSES												
		1	2	3	4	5	6	7	8	9	10	11	12	TOTAL
DIT	COMPUTED	0	0	0	0	0	0.09	0.09	0	0.09	0.09	0.09	0.09	0.54
	MAXIMUM	1	1	1	1	1	1	1	1	1	1	1	1	12
NOC	COMPUTED	0	0	0	0	0.18	0	0	0.36	0	0	0	0	0.54
	MAXIMUM	1	1	1	1	1	1	1	1	1	1	1	1	12
MFA	COMPUTED	0	0	0	0	0	1	1	0	0.5	0.5	0.5	0.5	4
	MAXIMUM	1	1	1	1	1	1	1	1	1	1	1	1	12
NAC	COMPUTED	0	0	0	0	0	0.09	0.09	0	0.09	0.09	0.09	0.09	0.54
	MAXIMUM	1	1	1	1	1	1	1	1	1	1	1	1	12
NDC	COMPUTED	0	0	0	0	0.18	0	0	0.36	0	0	0	0	0.54
	MAXIMUM	1	1	1	1	1	1	1	1	1	1	1	1	12

TABLE 2 CLASS LEVEL INHERITANCE METRICS

Measure of Functional Abstraction (MFA): MFA is the ratio of number of methods inherited by a class to the total number of methods of the class. Its range is from 0 to 1.The summation of MFA values for each class comes out to be 4 as shown in table 2.

Interpretation: Maximum value of MFA could have been 12and the computed value is 4 that is very less and this shows that inheritance used in the project is very less and therefore the reusability is lesser. It helps in predicting that the inheritance used is not making the project complex and is reliable as well this also proves that the testability will be lesser.

Number of Ancestor Classes (NAC): This metric measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy. It is also a class level metric. Values of NAC for each class are normalized w.r.t. 11.

Interpretation: From the table 2, the value of NAC is 0.54 which is very small compared to the maximum value of 12. It implies that the complexity is less. As the complexity is lesser, reliability is more and the testability effort is lesser.

Number of Descendant Classes (NDC): This metric measures the number of classes that may potentially be influenced by the class because of inheritance relations. Li proposed this metric addressing a problem in CK's NOC that measures only the immediate subclasses and not the further classification. In this project, computed value of NDC is same as that of NOC that is 0.54.

Interpretation: The computed value of 0.54 is very less than the maximum value of 12. NDC measures inheritance better than NOC as it takes into account the grandchildren. Also, it tells better about the testability, complexity etc. In class diagram, computed value is same as NOC.

4.2. Coupling Metrics

Coupling indicates the relationship or interdependency between modules. Strong coupling complicates a system. Coupling should be kept as low as possible. The following four coupling metrics are used for the quality estimation of the project.

Coupling between Objects (CBO): CBO for a class is a count of number of classes to which it is coupled. It is a class level metric and the couplings due to

METRIC	VALUES	CLASSES												
		1	2	3	4	5	6	7	8	9	10	11	12	TOTAL
CBO	COMPUTED	0.02	0.15	0.02	0.02	0.03	0.007	0.007	0.04	0.007	0.007	0.007	0.007	0.28
	MAXIMUM	1	1	1	1	1	1	1	1	1	1	1	1	12
DAC	COMPUTED	0.27	0.18	0.27	0.27	0.18	0.09	0.09	0.18	0.09	0.09	0.09	0.09	1.89
	MAXIMUM	1	1	1	1	1	1	1	1	1	1	1	1	12
RFC	COMPUTED	0.3	0.3	0.4	0.4	0.3	0.2	0.2	0.4	0.2	0.2	0.2	0.2	3.3
	MAXIMUM	1	1	1	1	1	1	1	1	1	1	1	1	12

Table 3 Class Level Coupling Metrics

inheritance are also computed in CBO.

Interpretation: Value of CBO is 0.28 which is very small compared to the maximum value of 12 as shown in Table 3. This shows that the classes are not much dependent on each other .This independency of classes show that the project is not complex and the testability effort required will be lesser as well as reusability will be easier.

Data Abstraction Coupling (DAC): DAC is the number of abstract data types (ADT) defined in a class. The metric which measures the coupling complexity caused by ADT's is DAC. The more ADT's in a class indicates large amount of coupling.

Interpretation: DAC's value computed is 1.89 and the value possible could have been 12.This shows that there are not many ADT's defined in a class. Therefore, the couplings are less and indicate that the project need less effort to test and is reliable also.

Coupling Factor (CF): It is the number of couplings to the total number of couplings possible in a system. Coupling due to inheritance are not included in it. It is a system level metric.

Coupling Factor =15/132
=0.11

Numerator comes out to be 15 by summing the values of CF for each class as shown in Table 4.

Interpretation: Value of CF comes out to be 0.11.This value is very less from the maximum value i.e. 132. It shows that the system is not complex and therefore the required testability effort will be lesser.

Response for a Class (RFC): The response set of a class is defined as set of methods that can be potentially executed in response to a message received by an object of that class. RFC= $|RS|$,where RS is the response set for the class.
It can be expressed as

$$RS=\{M\} U_{all\ i} \{R_i\},$$

Where $\{R_i\}$ = set of methods called by method I
{M}=set of methods in the class.
Values computed are shown in Table 3.

Interpretation: Computed value of RFC comes out to be 3.3 and the possible could have been 12. This shows that the project is not complex and the effort required to test would be lesser.

METRIC	CLASS	1	2	3	4	5	6	7	8	9	10	11	12	TOTAL
CF	COUPLINGS	3	2	3	3	2	0	0	2	0	0	0	0	15

Table 4: Coupling Factor

Coupling in a system increases complexity, reduces reusability. In the class diagram, RFC tells better about coupling as it tells about methods of the same class to be executed.

5. Conclusion

Inheritance promotes reusability but it should be used in a proper range so that the project doesn't become complex**.** Inheritance metrics at class level - DIT, NOC, NDC, NAC, MFA and system level - AIF, MIF have been evaluated in this work. For evaluation, class diagram of the ATM system has been used. It has been found that MFA is more predictive than MIF about inheritance. Secondly, it has been observed that single metric is not fully indicative of inheritance rather multiple metrics should be used. For instance, DIT gives depth of inheritance and NOC gives information about the width of inheritance tree.

Coupling indicates the relationship or interdependency between modules. Strong coupling complicates a system. Coupling should be kept as low as possible. Coupling Metrics at class level - CBO, RFC, DAC and at system level - CF have been evaluated and CF value found was lesser as compared to CBO because CF did not include inheritance. RFC value was found higher and more predictive of telling the effectiveness of the system as it also tells about methods of the same class to be executed.

References

[1] Abreu, B.F. and W.L. Melo (1996): "Evaluating the impact of Object-Oriented Design on Software Quality", Proceedings of METRICS '96, IEEE, 1996.pp. 90-99.

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

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

[4] Chidamber, S. R. and C. F. Kemerer (1994):"A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, 20(6), 1994, pp. 476–493.

[5] E Da-wei: "Analysis and Implementation of Software Metric for Object-Oriented ",IEEE International Conference, pp.1-4, 2009

[6] Lorenz, M. and J. Kidd (1994): "Object-Oriented Software Metrics", Prentice Hall, 1994.

[7] Li, W."Another Metric Suite for Object-Oriented Programming"Joumal of Systems and Software, Vol.44 Issue 2, pp. 155- 162, 1998**.**

[8] Pressman,R.S., "Software Engineering-A Practitioner's Approach", The McGraw-Hill ,Fifth Edition, 2001.

[9] Rosenberg L. H. and L. Hyatt (1995): "Software Quality Metrics for Object-Oriented Environments", SATC, NASA Technical Report SATC-TR-95-1001, 1995.

[10] W.Li and S.Henry, "Object Oriented Metrics that Predict Maintainability,"J.Systems and Software, vol.23, pp.111-122, 1993.