RTM: A Relation based Testability Metric for Object Oriented Systems

Sukhdip Singh
D C R University of Science & Technology, Murthal, Sonepat Haryana-131039, INDIA

Rajive Kumar
D C R University of Science & Technology,Murthal, Sonepat, Haryana-131039, INDIA

Raghuraj Singh
Harcourt Butler Technological Institute, Kanpur UP-208002, INDIA

ABSTRACT
The trend of day for object oriented software is highly complex objects, interacting with each other very rapidly. As a result it is becoming more and more difficult to manage and measure the complexity of the systems being developed. Relation-based testability measure is a metric, to provide the highly desirable insight to the inherent complexity of any object oriented system. We apply relation-based testability measure (RTM) to a University Automation product to measure the complexity of the system at an earlier stage of the development. Based on the approach we have developed an algorithm to measure the overall relational complexity of any object oriented system. The algorithm is very generic, accommodating both the flavors of traditional procedural approach and the modern object oriented approach. Applied at an early development stage, it can be very helpful for design, development and testing teams to co-ordinate their efforts and produce a much better and easy way to handle software product.

Keywords
Cyclomatic Complexity(CC), Structural Complexity (SC), Total Cyclomatic Complexity of Module (TCCM).

1. INTRODUCTION
The increasing dependence on software and the role of software in almost every field is also creating new challenges every day. In today's scenario the task of testing any software product is becoming increasingly complex and resource consuming. To make these inherently complex systems more manageable, the object oriented methodology is the flavor of the day for software professionals. The increased complexity of the systems poses many new challenges like infinitely many input combinations and corresponding outputs. It is almost impossible to cover all the paths of such systems with the available means, manual and automated.

Relation-based testability measure is metric, which will help to measure the overall testability of the complex object oriented system. It is found that metrics available till date are not sufficient for measuring the complexity of the systems at an early stage with the available testability measures for object oriented systems. Most of the available metrics are applied after the design has been finalized, so they are of little use for the designers of the software systems. More over the complexity information is computed so late that it is very costly to re-consider some design decisions.

The software complexity has been computed by many researchers using different attributes like control flow graphs [1], number of operators and operands used [2], number of identifiers per unit program area [3], cognitive complexity [4], [5], [6], [7] and spatial complexity [8], [9], [10], [11]. Voas et.al. [19] define software testability as the probability that the software will fail on its next execution, provided it contains fault. Vaos has proposed a technique for implementation of the different phases of their sensitivity analysis like introduction of flags at various points of the source code of the program. The program is run many times to find whether a particular piece of code has been executed or not and for how many times.

Freedman [21] proposes "domain testability", based on the notions of observability and controllability as adopted in hardware testing. Observability captures the degree to which a component can be observed to generate the correct output for a given input. The notion of 'controllability' relates to the possibility of a component generating all values of its specified output domain. Adapting a component such that it becomes observable and controllable can be done by introducing extensions. Observable extensions add inputs to account for previously implicit states in the component. Controllable extensions modify the output domain such that all specified output values can be generated. Freedman proposes to measure the number of bits required to implement observable and controllable extensions to obtain an index of observability and controllability, and consequently a measure of testability.

McGregor et. al. [20] has also given an approach to determine the testability of an object oriented system. They proposed a visibility component (VC), which is sensitive to the object oriented features like encapsulation, inheritance, exceptions and collaborations.

The VC depends upon following terms:-

1. Explicit Parameter: - An Object is an Explicit parameter of a method iff it is named in the method's signature.

2. Implicit Parameter: - An Object is an Explicit parameter of a method iff it is 'visible' from within the method.

3. Constant Object: An Object is said to be constant with respect to a method iff it is not modified with the execution of the method.

4. Constant Method: A Method is constant iff its execution does not affect any object of the system.

Jungmayr [22] takes an integration testing point of view, and focuses on dependencies between components. He proposes the notion of test-critical dependencies as well as metrics to identify them and subsequently removing them, using dedicated refactoring.

In [26] the available object oriented metrics have been studied and compared with each other and their relative usefulness has been discussed. The available metrics may be classified as under:-

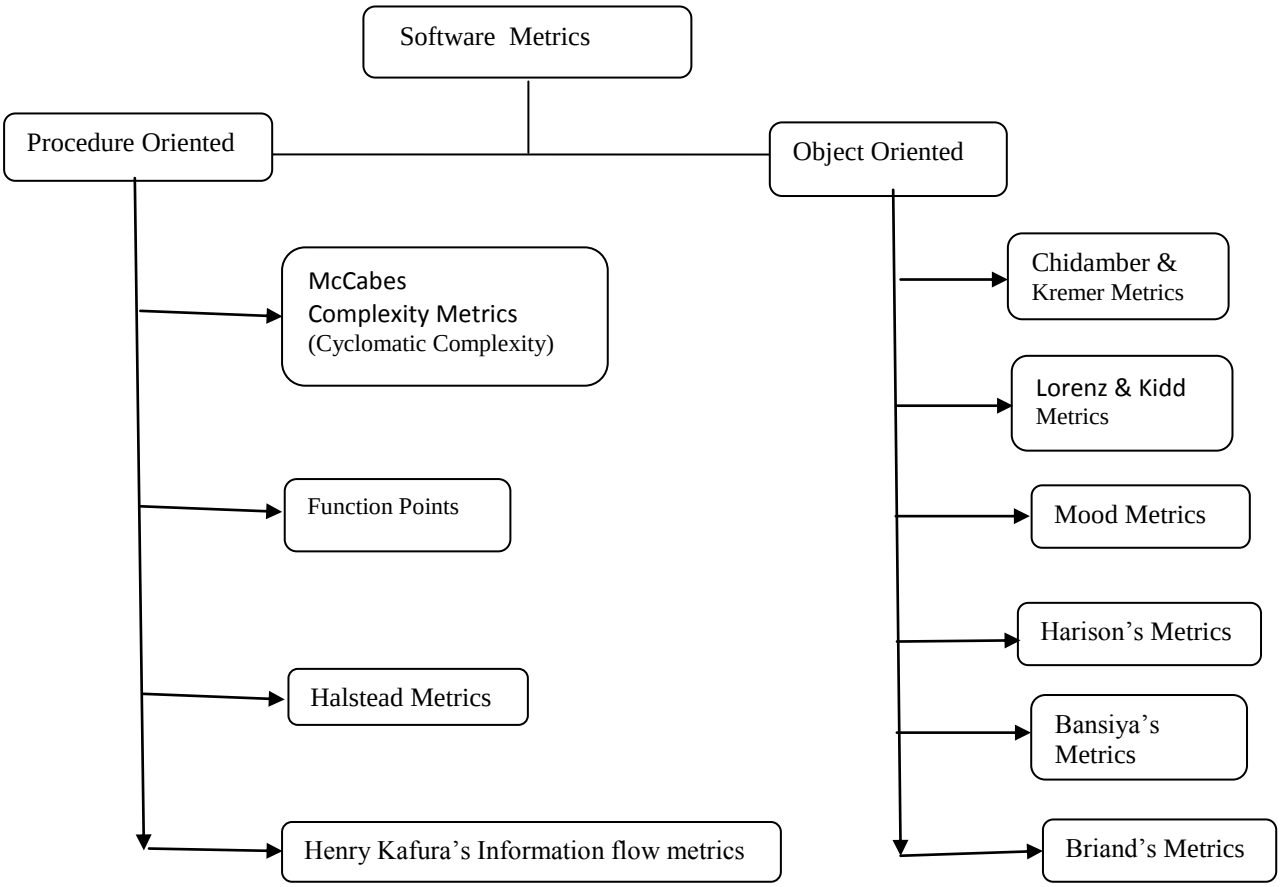

Fig 1. Software Metrics

2. EXISTING METRICS

The objective of this paper is to develop an approach using well known existing metrics for procedural as well as object-oriented systems, to predict the testability of the object oriented system. In section 2.1 and section 2.2, the relevant procedural metrics and object oriented metrics have been identified. Then, in section 2.3, an algorithm to calculate the Cyclomatic Complexity of the system has been proposed. The algorithm has been extended to calculate the complexity of the system based on object oriented metrics. After that the total relation based complexity of the system is calculated. The testability metric has been applied on three Java based projects and the results are analyzed in section 3.

2.1 Procedure Oriented Metric

Out of the available metrics, we have chosen the Cyclomatic Complexity [18] as the metric to measure the complexity of the individual method. To compute the Cyclomatic complexity (CC) we need to draw the control flow graphs for the program. Control flow graphs can be drawn from the flow charts of the program directly. CC provides a quantitative measure of the logical complexity of the method. The Cyclomatic Complexity can be computed in three ways:-

1. The number of regions of flow graph corresponds to Cyclomatic complexity.

2. Cyclomatic complexity, V(G) for a flow graph , G, is defined as

V(G)= E-N+2, where E is the number of flow graph edges. N is the number of flow graph nodes.

3. Cyclomatic complexity, V(G) for a flow graph, G is also defined as

V(G)=P + 1, where P is the number of Predicate nodes contained in flow graph G.

According to McCabe's [1] structured testing criterion, unreachable control flow paths need to be constructed. Theoretically, the control flow graphs can be constructed, but the actual efforts needed are influenced by the source code factors. The class-under-test will need to be initialized such that effective testing can be done. In our case, this entails that the fields of (an object of) a class are set to the right values before a test case can be executed. Furthermore, if the class-under-test depends on other classes, as it uses members of those classes, needs to be initialized. A class which deals with external interfaces (hardware, etc.) will typically require the external components to be initialized as well.

2.2 Object-Oriented Metrics

To select the object oriented metrics which are suitable candidates for characterizing the testability of the system, is really a tedious task. We have used the well-known metrics suite provided by Chidamber and Kemerer [14], as basis for this experiment. CK Metrics have direct impact on the testability of the system. The metrics have been discussed below. They are used in our experiment to implement and understand.

2.2.1 Inheritance Based Metrics

Inheritance is a powerful mechanism in an Object-Oriented (OO) programming. This mechanism supports the class hierarchy design and captures the IS-A relationship between a parent class and its child class. The two metrics for measuring inheritance, which we have considered are:-

2.2.1. 1 Depth of Inheritance Tree (DIT)

It is "the maximum length from the node to the root of the tree". It gives a measure of ancestor classes that can potentially affect this class. Higher value of DIT shows a higher potential for reuse but increased complexity.

2.2.1. 2 Number of Children (NOC)

It is the number of immediate sub-classes subordinated to a class in the class hierarchy. It is a measure of the child classes which inherits the methods of the parent class. High NOC value indicates high potential for reuse and likelihood of improper abstraction.

2.2.2 Coupling Based Metrics

These indicate the degree of interdependence among the modules of a software system. Low value of coupling is considered to be good for any software design. High value of coupling makes a system more complex because due to interdependency of modules on each other, it becomes very difficult to understand, control and modify the module. A class is said to be coupled to another class if it uses variables or methods of another class.

2.2.2. 1 Response for Class (RFC)

It is number of methods that can be invoked in response to a message sent to an object of a class. Accordingly the larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class. If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes complicated since it requires a greater level of understanding on the part of the tester.

2.2.2. 2 Coupling Between Objects (CBO)

It is a count of the number of other classes to which a class is coupled. CBO counts the non-inheritance related calls. According to this property larger the number of couples, the

higher the sensitivity to changes in other parts of the design and therefore maintenance is more difficult. Strong coupling complicates a system since a class is harder to understand, and modify. Two classes are said to be coupled if member functions declared in one class make use of methods or instance variables defined by other class. In fig 2 the value of CBO for class Ration is 2 and CBO is 0 for class TobePurchased and Available.

2.2.3 Cohesion Based Metrics

It is a measure of how strongly-related or focused the responsibilities of a single module are. As applied to object oriented programming, if the methods that serve the given class tend to be similar in many aspects, then the class is said to have high cohesion. Low cohesion modules for example are modules with coincidental cohesion, are indicative of a module that performs two or more basic functions. High cohesion modules have functional cohesion which indicates that modules perform only one basic function.

2.2.3. 1 Lack of Cohesion in Methods (LCOM)

It is a measure for the number of not connected method pairs in a class representing independent parts having no cohesion. It represents the difference between the number of method pairs not having instance variables in common, and the number of method pairs having common instance variables. High value of LCOM indicates that the class should probably be split into two or more classes.

2.2.4 Complexity Based Metrics

2.2.4.1 Weighted methods per Class (WMC)

It may be defined as the sum of complexity of all the methods defined in a class. If complexity of all the methods in class is assumed to be unity, then the WMC for that class will be equal to the number of methods defined in the class.

The following class diagram is used to show how the value of a particular metric is computed for a module, class or object

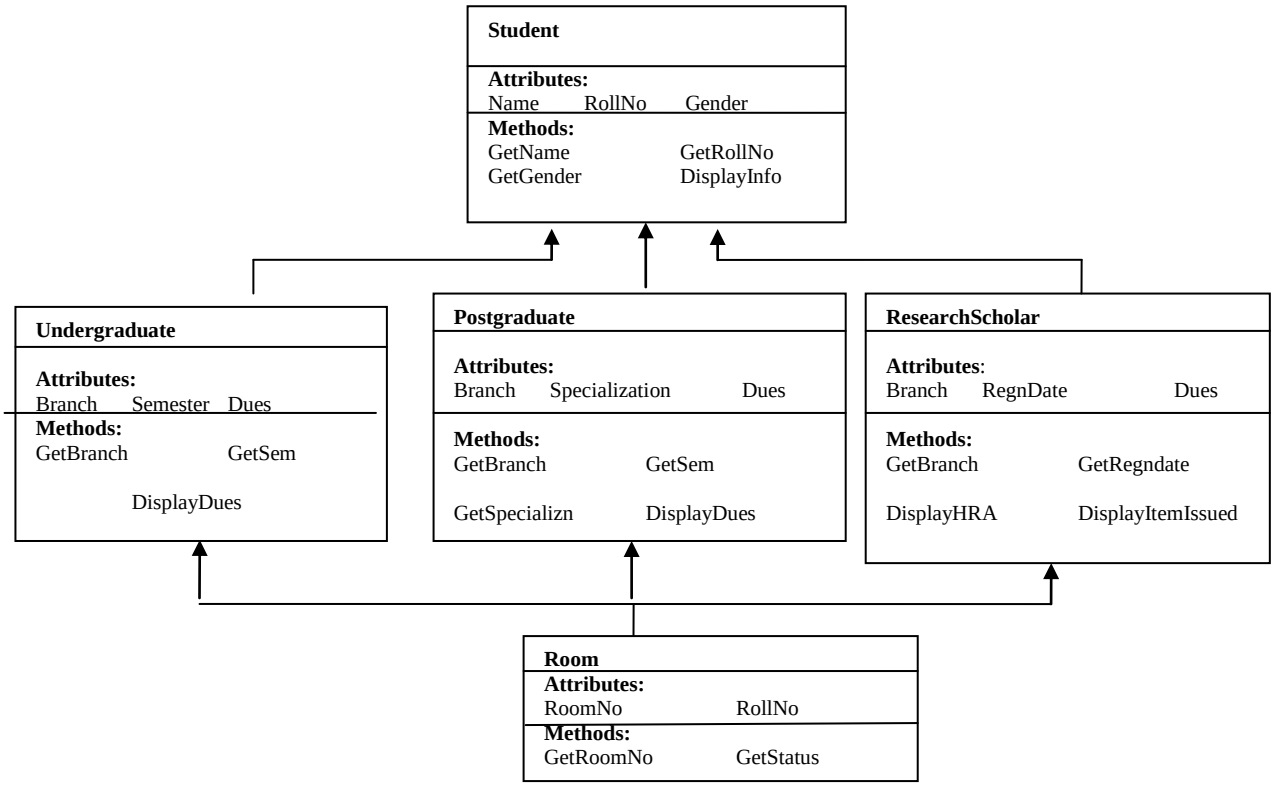

Fig 2: Class Diagram of Hostel management System

Metric/Class	Student	Undergraduate	Postgraduate	ResearchScolar	Room
DIT	0	1	1	1	2
NOC	3	1	1	1	0
WMC(Assuming complexity of each method to be unity)	4	3	3	3	2

Table1. Object oriented metric values for class diagram

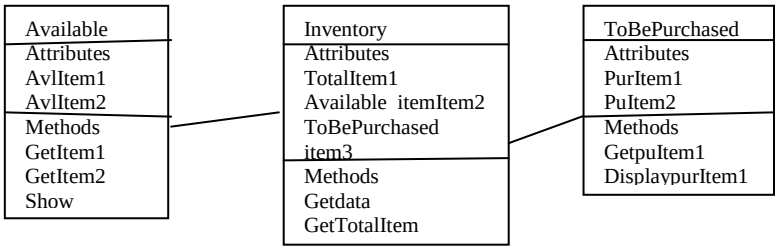

Fig 3: Class Diagram for Inventory System of Hostel

The following algorithm is used to compute the Cyclomatic complexity of all the methods used in all the classes of all the modules of the system first and then the overall relation based complexity of the system is computed :

3. ALGORITMH

RelationBasedTestabilityMetric : Compute RTM

Input: Class and Use-case diagrams of Object Oriented System, Total No of Modules

Output: Relation based Testability Metric value of Object Oriented System

RTM ←0

Step1 :- While (Module_Number ≠ Null) do

Step 2: For each class Ci in module MODi do

Step3 :- For each method Mi in class do

Step4 :- Compute Cyclomatic Complexity of the method Mi ϵ Ci , CCi

End for

Step5 :- Compute the total Cyclomatic Complexity of Class Ci , $$TCCi = \sum_{i=0}^{n} CCi$$

End for

Step6:- Compute Total Cyclomatic Complexity of Module $$TCCM = \sum_{J=1}^{M} TCCj$$

End While

Step7:- While (Module_Number ≠ Null) do

Step 8:- For each class Ci in module MODi do

Step 9:- $$SCi = DITi + NOCi + RFCi + CBOi + LCOMi + WMCi$$ End for

Step10:- $$TSC = \sum_{m=0}^{m=M} SCm$$

End While

Step11:- RBT=TCCM+TSC

Table 2. Notations

Abbreviation	Full Form
RBT	Relation Based Testability
C	Class
MOD	Module
M	Method
CC	Cyclomatic Complexity
SC	Structural Complexity
TSC	Total Structural Complexity
TCCM	Total Cyclomatic Complexity of Module

4. ANALYSIS

The algorithm has been applied on three Java projects. Project1 is a small project with low degree of modularity. Each module has a small number of classes. The complexity of the individual methods of the class is low. The methods in the classes are related to each other, the degree of dependence is also low. The second project is of moderate size and the values of metrics are also moderate. The third project is relatively larger in size and the metric values are also large. Project1 was developed during our experiment and the feedback provided to the developers proved to be very useful.

As is evident from the table below and the chart thereafter ,the RBT metric gives an indication of the complexity of the system to be developed at the earlier stage and may be used to guide the team to keep the complexity of the system as low as possible without compromising the functional structure of the system . The low value of RTM suggests that the system will be more testable at the later stages and it will help the testing team to perform their work more efficiently. In the projects considered for this study Project1 consisted of 3 packages and 20 classes, project2 consisted of 5 packages and 30 classes and project3, which is an open source project . All the projects have packages ,sub-packages and classes ,the proposed metrics take into account diff erent types of connections between two diff erent packages such as class −class, sub-package−sub-package, subpackage −class and class−sub-package . The logical structure of methods of the classes and their interaction and direction of connection between methods have also been taken into consideration during the measurement of complexity. The proposed metrics have been validated theoretically as well as empirically. The empirical validation of the proposed metric has been provided by evaluating three Java projects. Our Study clearly reflect that the proposed metric is a valid indicator of the complexity of the system under consideration.

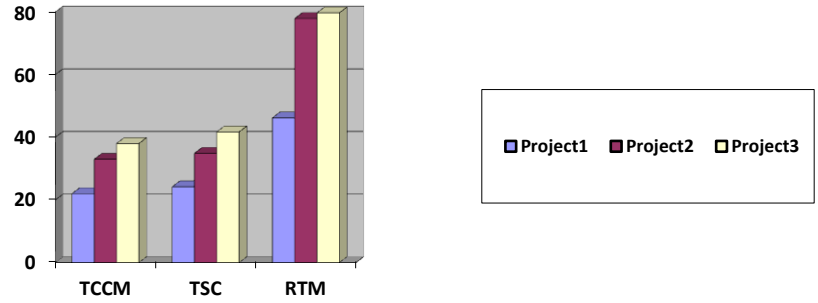

Fig 3 Histogram of the RTM for projects

Table 3: Analysis of results

	Project1	**Project2**	**Project3**
No. of Modules	5	7	10
TCCM	3+4+6+5+4=22	2+4+3+6+2+4+5+6=33	3+4+3+6+4+4+3+5+2+4= 38
TSC	3.6+4.3+6.2+5.4+4.7=24.2	2.3+4.2+3.7+6.3+2.3+ 4.7+5.1+6.3=34.9	3.4+4.3+3.3+6.4+4.5+ 4.6+3.3+5.4+2.2+4.3=41.7
RBT	46.2	77.9	79.7

5. CONCLUSION

In this paper efforts have been made on measuring the relation based complexity of the object oriented system. The paper presents an algorithmic approach to measure the complexity of the system considering its procedural and object oriented features simultaneously. The results obtained by applying this approach on the experimental projects have shown that the new approach has an edge over the existing approaches and is helpful for the developers. It provides an early indication of the complexity of the system and the developers may improve their design. As the metrics may be used in early stages it will help to evaluate the complexity of the system at an early stage. By using this approach better products may be designed in terms of understandability, maintainability and testability.

6. REFERENCES

[1] McCabe, T.J., 1976. A Complexity Measure. IEEE Transactions on Software Engineering, SE-2 (4), 308-320.

[2] Halstead, M.H. 1997. Elements of Software Science, Elsevier NorthHolland, New York.

[3] Harrison. W. 1992. An Entropy-based Measure of Software Complexity. IEEE Transactions on Software Engineering, 18(11), 1025-1029.

[4] Misra, S. 2006. Modified Cognitive Complexity Measure, In: 21st ISCIS'06, LNCS, vol. 4263,.1050-59.

[5] Misra, S. 2006.A Complexity Measure based on Cognitive Weights. International Journal of Theoretical and Applied Computer Science, 1(1), 1-10.

[6] Wang, Y., and J. Shao 2003. Measurement of the Cognitive Functional Complexity of Software, In: IEEE International Conference on Cognitive Informatics, ICCI'03, 67-71.

[7] Wang, Y. and Shao, J. 2003 A New Measure of Software Complexity based on Cognitive Weights. Canadian Journal of Electrical & Computer Engineering, 28(2), 69-74.

[8] Chhabra, J.K, K.K. Aggarwal, and Y. Singh 2003. Code and Data Spatial Complexity: Two Important Software Understandability Measures. Information and Software Technology. 45(8), 539-546.

[9] Chhabra, J.K, K.K. Aggarwal, and Y. Singh 2004. Measurement of Object Oriented Software Spatial Complexity. Information and Software Technology, 46(10), 689-699.

[10] Douce, C.R., P.J. Layzell, and J. Buckley, 1999. Spatial Measures of Software Complexity. In 11th Meeting of Psychology of Programming Interest Group, http://www.ppig.org/workshops/11th-programme.html

[11] Chhabra, J.K, K.K. Aggarwal, and Y. Singh , 2004, A Unified Measure of Complexity of Object-Oriented Software. Journal of the Computer Society of India, 34(3), 2-13

[12] Weyuker, E.J., 1988. Evaluating Software Complexity Measure. IEEE Transaction on Software Engineering, 14(9), 1357-136

[13] Briand, L.C., S. Morasca, and V.R. Basili, 1996. Property based Software Engineering Measurement.

IEEE Transactions on Software Engineering, 22(1), 68-86.

[14] Chidamber, S., and Kemerer, C., 1994. A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, 20(6),

[15] Tegarden, D.P., S.D. Sheetz, and D.E. Monarchi, 1992. The Effectiveness of Traditional Metrics for Object-Oriented Systems. In Twenty-Fifth Hawaii International Conference on System Sciences, Vol IV, IEEE Computer Society Press.

[16] Tian, J., and M.V. Zelkowitz, 1992. A Formal Program Complexity Model and its Application. J. Systems Software, 17, 253-266.

[17] Lakshmanian, K.B., S. Jayaprakash, P.K. Sinha, 1991. Properties of Control-Flow Complexity Measures. IEEE Transaction on Software Engineering, 17(2), 1289-1295.

[18] Pressman Roger S. , Software Engineering :A Practitioner's Approach, 5th Edition, 448-450

[19] J. vaos , L Morrel and K Miller , 1991. Predicting Where faults Can hide from Testing, IEEE Software, 8, 41-48 .

[20] [20] J. McGregor and S. Srinivas,1996. A Measure of Testing Eff ort. In Proceedings of the Conference on Object-Oriented Technologies, USENIX Association, 129–142.

[21] R. Freedman, 1991. Testability of Software Components. IEEE Transactions on Software Engineering, 17(6), 553–564.

[22] S. Jungmayr,2002. Identifying test-critical dependencies. In Proceedings of the International Conference on Software Maintenance, IEEE Computer Society, 404–413.

[23] R. Binder, 1994. Design for testability in object-oriented systems. Communications of the ACM, 37(9), 87–101.

[24] B. Henderson-Sellers, 1996. Object Oriented Metrics. Prentice Hall, New Jersey.

[25] L. C. Briand, J. W. Daly, and J. K. W¨ust, 1999. A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering, 25(1), 91–121.

[26] Sukhdip Singh , 2008 . Testability of Object Oriented Software : A Review. Proceedings of National Conference on Information Security & Mobile Computing 174-177.