Testability Assessment of Object Oriented Software Using Static Metric Model and Analytic Hierarchy Process

Article · May 2015

CITATIONS
0

READS
161

2 authors, including:

Harsha Singhani Ratnani
Jagannath International Management School
6 PUBLICATIONS **17** CITATIONS

SEE PROFILE

Testability Assessment of Object Oriented Software Using Static Metric Model and Analytic Hierarchy Process

Dr. Pushpa R. Suri[1], Harsha Singhani[2]
[1]Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, Haryana, India

[2]Research Scholar, Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, Haryana, India

Abstract

Based on existing software testability models for object oriented software, we have proposed a new testability assessment model for object oriented software. The model is based on those six important internal programming features of object oriented design which are not used before together at the same time in-spite of being highlighted in some or other research. These factors are assessed using popular static object oriented metrics and their link with testability is established. The model is further analysed using Multi Criteria Decision Making (MCDM) approach. The model would be validated using Analytic Hierarchy Process (AHP). The proposed model and evaluation technique helps software engineering practitioners to choose the best alternative amongst available options by analysing the Testability.

Keywords: *Software Testability Assessment Model, Object Oriented Testability, Static Metric, AHP.*

1. Introduction

Testability is one of the qualitative factors of software engineering which has been accepted in McCall and Boehm software quality model, which built the foundation of ISO 9126 software quality model. Formally, Software testability has been defined and described in literature from different point of views IEEE [1] defines it as "The degree to which a system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met" and ISO [2] has defined software testability as functionality or "attributes of software that bear on the effort needed to validate the software product".

The testability research actually is done from the prospect of reducing testing effort and testing cost which is more than 40% of total development cost of any software [3]. Still, the research in the field of testability has not been done in much detail. As discussed in our previous work about testability and testability metrics [4], [5], it has been found that testability research has taken a speed up in past few years only. Tough, much of the work has been done using various object oriented featured metrics only. In this paper we have proposed a testability model for assessment during design time and evaluated the same using AHP technique.

This paper is organized as follows: Section2 gives brief overview of software testability related work. Section3 showcases the proposed testability assessment model from design perspective. Section4 provides overview of material and methodology used during this research. Section5 presents the details of testability evaluation based on proposed model using AHP. It is followed by result and findings in section with conclusion drawn in section 7.

2. Related Work

2.1 Software Testability

Software testability measurement refers to the activities and methods that study, analyze, and measure software testability during a software product life cycle. Unlike software testing, the major objective of software testability measurement is to find out which software components are poor in quality, and where faults can hide from software testing. Now these measurements can be applied at various phases during software development life cycle of a system. In past number of research efforts were made addressing software testability measurement. The focus of past studies was on how to measure software testability at various software development phases like Design Phase [6]–[13] and Coding Phase[14]–[17]. Lot of stress has been given upon usage of object oriented metrics for object oriented software testability evaluation during these researches. The metrics investigated related to object oriented software testability assessment mostly belong to static software metrics category. These metrics were mostly adapted from CK, MOOD, Brian, Henderson-Sellers metric suite [18]–[21]. Furthermore, Lot of empirical studies has been done in showing the correlation of these metrics with unit testing effort [22]–[25]. Also found that few studies have been focussed on UML diagram features from software testability improvisation prospect during review of these design diagrams [26]–[29]. All this work has been explained in depth in our previous research work [4],[5]. But still very less work has been found in testability analysis using MCDM techniques, in

spite of the fact that the testability factor depends on multiple criteria which is explained next.

2.2 Analytical Hierarchy Process

In context with software engineering problems, very few studies related to multi-criteria decision making (MCDM) approach has been done and published. Saaty [30] proposed AHP as one of the most practical method based on MCDM. There are other popular methods such as Fuzzy-AHP and preference ranking organization method of enrichment evaluations (PROMETHEE-2), all capable of solving logistics as well as technical systems. Now, when it comes to testability very less of it is validated ever using any MCDM techniques.

AHP technique is proposed by Saaty, which based on pair-wise matrix to determine indistinctiveness in MCDM problems. It helps in decision making on the basis of needs and understanding of the problem [30]. P. Khanna [31] have proposed primitive work in this field using AHP for testability which is not supported by any empirical study on the data. Dubey et. al. [32] have done study on object oriented usability. Though some work have been found for aspect oriented software testability and reusability assessment using MCDM technique done by Singh and Sangawan [33], [34], which has been technically found useful in how AHP needs to be applied in other software's too for the study of other quality features. Yang [35] have also used this technique for analysing and calculating hardware testability using comprehensive weighted method and AHP.

3. Proposed Model

Our proposed testability model is based on Dromey's software quality model [36] which has been a benchmark in use for various quality features as well as many testability models so far. We have followed the steps as mentioned below to formalize the model:

- Identification of internal design features for object oriented software testability assessment.
- Identification of static metrics out of many popular metrics for each.
- Establishing link between testability and these identified factors.
- Followed by Model Evaluation using AHP technique.

On the basis of our previous research work and surveys we have identified six factors to assess testability for object oriented software at design level [4], [5]. All these are internal quality characteristics – Encapsulation, Inheritance, Coupling, Cohesion, Polymorphism and Size & Complexity as explained in Table 1. Out of six identified features four features have been proposed in MTMOOD testability model [10], which does not cover the polymorphism and size & complexity feature, which have also been found as essential internal features by many researchers in testability study [15], [22], [36], [37].

These six object oriented features play a very significant role in testability improvisation directly or indirectly. This relation has been build based on thorough study of publications [2], [20], [35], [38], [39]etc.

Table 1: Object Oriented Design Feature Affecting Testability

OO Feature Testability	Definition
Encapsulation	It is defined as a kind of abstraction that enforces a clean separation between the external interface of an object and its internal implementation
Inheritance	It is a measure of the 'is-a' relationship between classes.
Coupling	It is defined as the interdependency of an object on other objects in a design.
Cohesion	It defines as the internal consistency within the parts of design.
Size & Complexity	It's the measure of size of the system in terms attributes or methods included in the class and capture the complexity of the class.
Polymorphism	Polymorphism allows the implementation of a given operation to be dependent on the object that "contains" the operation.

The studies indicate encapsulation promotes efficiency and complexity. Inheritance has a significant influence on the efficiency, complexity, reusability and testability or maintainability. While low coupling is considered good for understandability, complexity, reusability and testability or maintainability, whereas higher measures of coupling are viewed to adversely influence these quality attributes. Cohesion is viewed to have a significant effect on a design's understandability and reusability. Size & Complexity has a significant impact on understandability, and testability or maintainability. Polymorphism reduces complexity and improves reusability. Though these features can be measured by many metrics options available as discussed earlier [5]. Most of these metrics are accepted by practitioners on 'heavy usages and popularity' and by academic experts on empirical (post

development) validation. But to keep study simple from AHP evaluation aspect we have chosen the few basic but popular metrics amongst testability researchers.

So, the proposed testability assessment model with respect to internal design features using static metrics is as shown in Fig1.It is based on six above mentioned object oriented features from testability perspective as pointed in Binders research too [6]. Out of all the popular metrics suites discussed in our previous work [41] six of these static metrics as explained below in Table2 have been identified for the evaluation of each of these feature and their effects on any object oriented software testability at design time.

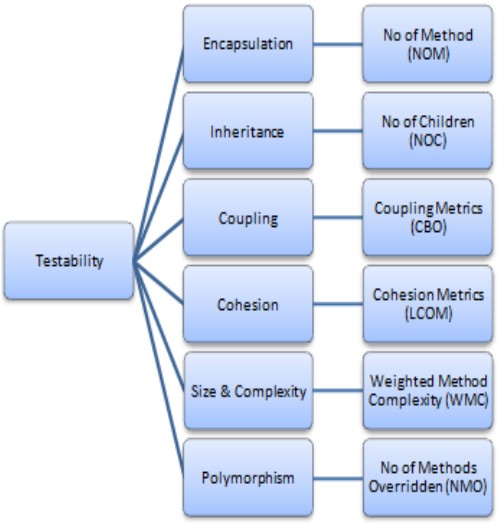

Fig 1: Proposed Software Testability Assessment Model with Static Metrics

As described in Table2 below for Encapsulation evaluation number of methods metrics (NOM) is being chosen by many researchers for the effect of information hiding on testability [10], [38]. So we kept it for encapsulation evaluation for our model too. Inheritance is evaluated using Number of Children metrics (NOC), one of the most popular and efficient inheritance metrics [22], [36], [41], [42]. For Coupling we chose coupling between objects (CBO) and for Cohesion we opted cohesion metrics (Li & Henry version) (LCOM). These two were the most sought after and unparalleled metrics available for assessing coupling and cohesion effect on testability as per literature study and popularity amongst industry practitioners [10], [20], [22] ,[24], [37], [43].Though Size & Complexity can be easily measured by other metrics in this category but we chose weighted method complexity (WMC) metrics due to its significant role and association in number of test case indication pointed [6], [22], [42]. Polymorphism is one of the underlying factors affecting testability but as quite stressed by early researchers like Binder and others [6], [45] as it results in testability reduction ,we chose polymorphism factor metrics (POF/PF) for testability assessment.

Table2: Testability Model Metrics Details

Testability Factor	Metrics Name	Description
Encapsulation	No of Method (NOM)	This metric is the count of all the methods
Inheritance	No of Children (NOC)	This metric is the count of children of super-class in the design.
Coupling	Coupling Between Object (CBO)	This metric count of the different number of other classes that a class is directly coupled to. (Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class)
Cohesion	Cohesion Metric (LCOM)	This metric computes the relatedness among methods of a class based upon the parameter list of the methods.
Size & Complexity	Weighted Method Complexity (WMC)	It s the count of sum of all methods complexities in a class
Polymorphism	No of methods overridden (NMO)	It is count of overridden method in a subclass

4. Material And Methodology

4.1 AHP Methodology

It initially requires the goal objective to be divided in to hierarchy of factors and sub-factors, which can be easily analysed individually. Once the hierarchy is build the decision maker's job is to evaluate the problem as follows:

Step1. **Reciprocal Matrix Formation:** First, a pair-wise comparison matrix has been constructed based on the factors. Every factor needs to compare with the immediate next factor. A common scale by Saaty as shown in Table3 below is used for the same.

The matrix thus formed somewhat look likes this, Suppose for n number of factors, F_1, F_2….F_n are considered, which are to be compared. Relative weight of F_i relating to F_j denoted as m_{ij} and a

square matrix A = [m_{ij}] of order n will be formed as given in equation (1) below.

$$A = \begin{matrix} F_1 \\ [m_{ij}] \\ F_n \end{matrix} = \begin{pmatrix} F_1 & F_1 & . & F_1 \\ 1 & m_{12} & . & m_{1n} \\ F_2 1/m_{12} & 1 & . & m_{2n} \\ . & . & . & . \\ 1/m_{1n} & 1/m_{2n} & . & 1 \end{pmatrix} \quad (1)$$

Here, $m_{ij} = 1/m_{ji}$ and i does not equal to j and $m_{ii} = 1$ for all i. Hence the calculated matrix is known as reciprocal matrix.

Table 3: Satty Rating Scale [30]

Intensity of Importance	Definition	Description
1	Equal Importance	Elements C_i and C_j are equally important
3	Weak Importance of C_i over C_j	Experience and Judgment slightly favor C_i over C_j
5	Essential or Strong Importance	Experience and Judgment strongly favor C_i over C_j
7	Demonstrated Importance	C_i is very strongly favored over C_j
9	Absolute Importance	The evidence favoring C_i over C_j is of the highest possible order of affirmation
2,4,6,8	Intermediate	When compromise is needed, values between two adjacent judgments are used
Reciprocals of the above judgments	If C_i has one of the above judgments assigned to it when compared with C_j, then C_j has the reciprocal value when compared with C_i	A reasonable assumption

Step2: **Eigen Vector Calculation:** Next, we have to evaluate the relative weights of the factors, which are relevant to the problem is called an eigen vector ω.

$$A\ \omega = \lambda_{max}\ \omega\ ,\ \lambda_{max} = n \quad (2)$$

Where, ω is eigen vector and λ_{max} is eigen value. For a consistent matrix, $\lambda_{max} >= n$.

Step3: **Consistency Index Calculation:** Now, we have to evaluate Consistency Index (CI) for that matrix using

$$CI = \frac{(\lambda_{max} - n)}{n-1} \quad (3)$$

Step4: **Consistency Ratio:** Finally, we have to evaluate consistency ratio (CR) using saaty average consistency index (RI) values as shown in Table4.

$$CR = \frac{CI}{RI} \quad (4)$$

Table 4: Saaty Scale of Average Consistency Index (RI) [30]

1	2	3	4	5	6	7	8	9	10
0.0	0.0	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

Saaty also proposed that if the CR > 0.1, the judgements may not be consistent and unreliable. In such a case, a new comparison matrix is needed to set up until CR < 0.1. This way we can apply the AHP for predicting a decision based on available choices at hand.

4.2 Testability Study

In order to conduct testability study based on above model and AHP technique. The hierarchical model with factors – Encapsulation (F1), Inheritance (F2), Coupling (F3), cohesion (F4), Size & complexity (F5) and polymorphism (F6) has been shown below in fig2. In order to assign weights to these factors a survey form was being sent to 10 professional which are either academicians doing research in object oriented testing related subjects or having good knowledge of object oriented concepts or from industry professional practicing these methods. On basis of eigen value, eigen vector, consistency ratio and consistency index calculations, we have been able to evaluate weights for all these factors which is shown in detail in next section.

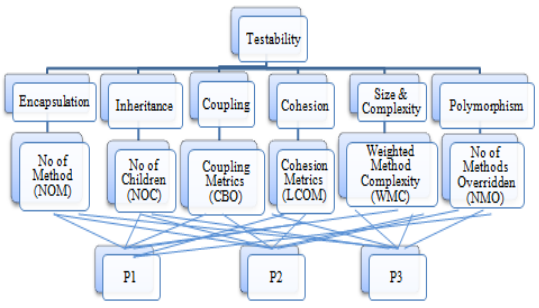

Fig 2: AHP Hierarchy for Evaluation of Software Testability Based on Above Model

5. Evaluation of Testability Model Using AHP

5.1 Proposed Model Evaluation

A square matrix of 6X6 is sent for pair-wise weight filling to 10 experts as discussed above. The mean matrix thus formed using these 10 samples on six testability factors is given below:

$$\begin{pmatrix} F1 & F2 & F3 & F4 & F5 & F6 \\ 1.00 & 1.35 & 4.50 & 3.20 & 0.95 & 1.55 \\ 1.23 & 1.00 & 4.30 & 3.30 & 1.07 & 1.40 \\ 0.23 & 0.24 & 1.00 & 0.63 & 0.22 & 0.26 \\ 0.32 & 0.31 & 1.95 & 1.00 & 0.27 & 0.31 \\ 1.55 & 1.60 & 4.70 & 3.80 & 1.00 & 1.40 \\ 0.95 & 1.10 & 3.90 & 3.30 & 1.10 & 1.00 \end{pmatrix} \quad (5)$$

There are many methods for calculating the eigenvector. We have used spreadsheet based approximate calculations for local priorities of criteria. The Eigen value thus calculated are as shown in table 5 below. The eigenvector of the relative importance of F1, F2, F3, F4, F5 and F6 is (0.22, 0.22, 0.04, 0.06, 0.25, 0.20), which is given in Table-5. These values are weights of main factors i.e. Encapsulation (0.22), Inheritance (0.22), Coupling (0.04), Cohesion (0.06), Size & Complexity (0.25) and Polymorphism (0.20) in testability assessment.

Now the six eigen values calculated for each of these factors is (6.62, 6.61, 6.58, 6.47, 6.63, 6.60) with λ_{max}=6.59 which is >= 6 (total no of factors), which is consistent. Using this we calculate the CI and CR values as follows:

$$CI = \frac{(\lambda_{max}-n)}{n-1} = \frac{6.59-6}{6-1} = 0.12 \quad (6)$$

$$CR = CI/RI = 0.12/1.24 = 0.09 \quad (7)$$

Table 5: Eigen Vector and Eigen Value for main factors

	F1	F2	F3	F4	F5	F6	Eigen Value
F1	1.00	1.35	4.50	3.20	0.95	1.55	0.22
F2	1.23	1.00	4.30	3.30	1.07	1.40	0.22
F3	0.23	0.24	1.00	0.63	0.22	0.26	0.04
F4	0.32	0.31	1.95	1.00	0.27	0.31	0.06
F5	1.55	1.60	4.70	3.80	1.00	1.40	0.25
F6	0.95	1.10	3.90	3.30	1.10	1.00	0.20

λ_{max} =6.59, CI=0.12, CR=0.09

We found the calculated value of CR<0.1 in all the samples of matrices, which indicates that the estimate is consistent and acceptable.

5.2 Testability Evaluate of Sample OO Projects:

We have applied the above testability assessment on three object oriented programs the data for which is taken from [46] which consists of three standard object oriented projects. Table 6 below shows the gathered metric value for each of the above mentioned programming features. Here the prime motivation is to show the applicability of the proposed scheme, irrespective of the size of the considered project. The AHP technique is applied on pair-wise comparison matrix of OO projects for each testability factor individually.

Table 6: Three Project Metrics Values[46]

	NOM	NOC	CBO	LCOM	WMC	NMO
P1	6	3	1	0.5	6	1.5
P2	10	1	2.2	0.5	10	8
P3	8.8	1	2.2	1	8.8	1.5

The eigen vector value for all three projects with respect to six testability assessment factors- Encapsulation (Table7), Inheritance (Table8), Coupling (Table9), Cohesion (Table10), Size& Complexity (Table 11) and Polymorphism (Table12) are shown below. The solution with respective eigen vector values and respective CR (0.07, 0.07, 0.07, 0.06, 0.07, 0.08) values are also below in these tables. All CR values are below 0.1. Hence, the judgements are consistent and acceptable.

This matrix eigen vector values are utilised in evaluating global utility of each project and its overall rank.

Table 7: Pair-wise Comparison Matrix of three OO Projects for Encapsulation

	P1	P 2	P 3	Eigen Values
P1	1.00	5.00	3.00	0.62
P2	0.20	1.00	0.25	0.10
P3	0.33	4.00	1.00	0.28

λmax=3.09,CI=0.04,CR=0.07

Table 8: Pair-wise Comparison Matrix of three OO Projects for Inheritance

	P1	P2	P3	Eigen Values
P1	1.00	0.33	4.00	0.28
P2	3.00	1.00	5.00	0.62
P3	0.25	0.20	1.00	0.10

λ_{max}=3.09, CI=0.04, CR=0.07

Table 9: Pair-wise Comparison Matrix of three OO Projects for Coupling

	P1	P2	P3	Eigen Values
P1	1.00	4.00	7.00	0.69
P2	0.25	1.00	4.00	0.23
P3	0.14	0.25	1.00	0.08

λ_{max} =3.08, CI=0.04, CR=0.07

Table 10: Pair-wise Comparison Matrix of three OO Projects for Cohesion

	p1	p2	p3	Eigen Values
P1	1.00	3.00	0.33	0.27
P2	0.33	1.00	0.25	0.12
P3	3.00	4.00	1.00	0.61

λ_{max} =3.07, CI=0.04, CR=0.06

Table 11: Pair-wise Comparison Matrix of three OO Projects for Size

	P 1	P 2	P 3	Eigen Values
P1	1.00	5.00	2.00	0.57
P2	0.20	1.00	0.25	0.10
P3	0.50	4.00	1.00	0.33

λ_{max}=3.02, CI=0.01, CR=0.02

Table 12: Pair-wise Comparison Matrix of three OO Projects for Polymorphism

	P 1	P 2	P 3	Eigen Values
P1	1.00	6.00	3.00	0.63
P2	0.17	1.00	0.20	0.08
P3	0.33	5.00	1.00	0.29

λ_{max}=3.10,CI=0.05,CR=0.08

Now finally we have to construct a matrix of the eigenvectors for three selected projects P1, P2 and P3 and six testability assessment factors weights F1, F2, F3, F4, F5, and F6 as mentioned below.

The overall global utility of each project is calculated using the summation of the products of the weight of OO Project with reference to each factor by the weights of corresponding factor yields the global utility of each OO Project.

OOS Testability = $\sum_{i=1}^{n} Weight\ value\ of\ Fi * Comparative\ value\ of\ Pi$ (8)

For example: U (P1) = 0.22*0.62+0.22*0.28+0.04*0.69+0.25*0.62+0.2*0.63 = 0.52 (9)

The best OO Project is the one which is having the highest overall testability index values. Accordingly, ranking of OO Project is done which are shown in Table 13 and P1 found to be the best choice as its testability index value is highest amongst three.

Table 13: Global overall utility and Rank of all Three projects w.r.t. Testability

	F1	**F2**	**F3**	**F4**	**F5**	**F6**	Global Utility	Rank
W_i	0.22	0.22	0.04	0.06	0.25	0.20		
P1	0.62	0.28	0.69	0.27	0.62	0.63	0.52	1
P2	0.10	0.62	0.23	0.12	0.10	0.08	0.21	3
P3	0.28	0.10	0.08	0.61	0.28	0.29	0.25	2

6. Result and Findings

The above technique has shown that role of encapsulation (22%), inheritance (22%), coupling (4%), Cohesion (6%), size& complexity (25%) and Polymorphism (20%) in overall testability assessment of any OO project as per sample survey based on AHP technique. The result here is utilised for three medium sized projects for overall testability index (TI) calculation. In actual situation, comparative values of characteristics can be gathered from running projects, which are developed using object oriented technology. Though, the projects, which are compared here, are medium size projects but still good enough to support the model. However, our motive is to show the applicability of proposed scheme for the testability estimation of object oriented software. Proposed schemes can be applied on real life software based on the values of identified six factors and it will determine the Testability Index (TI) for the considered software. It can be applied on each module (method, class, package, module etc) in order to know their testability or it can also be applied on whole developed system to know its overall testability.

7. Conclusion and Future Scope

In this paper we have proposed an object oriented testability model depending on internal object oriented software design features. The six OO factors affecting testability are – Encapsulation, Inheritance, Coupling, Cohesion, Size & Complexity and Polymorphism found and identified as per literature survey. We linked each of these features with suitable popular OO metrics only at design level. Now, in order to evaluate testability using above model we used analytical hierarchical process (AHP). The weights of each of these factors thus obtained using this technique was

being applied on three medium sized projects for testability assessment.

In future the assessment of many core runtime testability factors and metrics may be analysed using AP technique along with large scale industrial survey. Later, this model can be cross validated using other techniques and help practitioners in testability estimation and improvisation first at design and later at source code level, which has not been covered in our study. Software practitioners can use the proposed approach for selecting the appropriate program in term of software testability for OO software.

References

[1] J. Radatz, A. Geraci, and F. Katki, "IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)," 1990.

[2] ISO, "ISO/IEC 9126: Software Engineering Product Quality," 2002.

[3] A. P. Mathur, *Foundations of Software Testing*, Second. Pearson, 2013.

[4] P. R. Suri and H. Singhani, "Object Oriented Software Testability Survey at Designing and Implementation Phase," *International Journal of Science and Research*, vol. 4, no. 4, pp. 3047–3053, 2015.

[5] P. R. Suri and H. Singhani, "Object Oriented Software Testability (OOSTe) Metrics Analysis," *International Journal of Computer Applications Technology and Research*, vol. 4, no. 5, pp. 359–367, 2015.

[6] R. V Binder, "Design For Testabity in Object-Oriented Systems," *Communications of the ACM*, vol. 37, pp. 87–100, 1994.

[7] S. Jungmayr, "Testability during Design," pp. 1–2, 2002.

[8] B. Pettichord, "Design for Testability," in *Pacific Northwest Software Quality Conference.*, 2002, pp. 1–28.

[9] E. Mulo, "Design for Testability in Software Systems," 2007.

[10] R. A. Khan and K. Mustafa, "Metric based testability model for object oriented design (MTMOOD)," *ACM SIGSOFT Software Engineering Notes*, vol. 34, no. 2, p. 1, 2009.

[11] M. Nazir, R. A. Khan, and K. Mustafa, "Testability Estimation Framework," *International Journal of Computer Applications*, vol. 2, no. 5, pp. 9–14, 2010.

[12] D. Esposito, "Design Your Classes For Testbility." 2008.

[13] J. E. Payne, R. T. Alexander, and C. D. Hutchinson, "Design-for-Testability for Object-Oriented Software," *Object Magazine*, vol. 7, no. 5, pp. 34–43, 1997.

[14] Y. Wang, G. King, I. Court, M. Ross, and G. Staples, "On testable object-oriented programming," *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 4, pp. 84–90, 1997.

[15] B. Baudry, Y. Le Traon, G. Sunye, and J. M. Jézéquel, "Towards a ' Safe ' Use of Design Patterns to Improve OO Software Testability," *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*, pp. 324–329, 2001.

[16] M. Harman, A. Baresel, D. Binkley, and R. Hierons, "Testability Transformation: Program Transformation to Improve Testability," in *Formal Method and Testing, LNCS*, 2011, pp. 320–344.

[17] M. Badri, A. Kout, and F. Toure, "An empirical analysis of a testability model for object-oriented programs," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 4, p. 1, 2011.

[18] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[19] A. Fernando, "Design Metrics for OO software system," *ECOOP'95, Quantitative Methods Workshop*, 1995.

[20] L. C. Briand, J. Wust, S. V. Ikonomovski, and H. Lounis, "Investigating quality factors in object-oriented designs: an industrial case study," *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*, 1999.

[21] B. Henderson and Sellers, *Object-Oriented Metric*. New Jersey: Prentice Hall, 1996.

[22] M. Badri, "Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes," *Journal of Software Engineering and Applications*, vol. 05, no. July, pp. 513–526, 2012.

[23] M. Bruntink and A. Vandeursen, "An empirical study into class testability," *Journal of Systems and Software*, vol. 79, pp. 1219–1232, 2006.

[24] L. Badri, M. Badri, and F. Toure, "An empirical analysis of lack of cohesion metrics for predicting testability of classes," *International Journal of Software Engineering and its Applications*, vol. 5, no. 2, pp. 69–86, 2011.

[25] Y. Singh and A. Saha, "Predicting Testability of Eclipse: Case Study," *Journal of Software Engineering*, vol. 4, no. 2, pp. 122–136, 2010.

[26] B. Baudry, Y. Le Traon, and G. Sunye, "Improving the testability of UML class diagrams," *First International Workshop onTestability Assessment, 2004. IWoTA 2004. Proceedings.*, 2004.

[27] M. Genero, M. Piattini, and C. Calero, "A survey of metrics for UML class diagrams," *Journal of Object Technology*, vol. 4, no. 9, pp. 59–92, 2005.

[28] B. Baudry and Y. Le Traon, "Measuring design testability of a UML class diagram," *Information and Software Technology*, vol. 47, no. 13, pp. 859–879, 2005.

[29] B. Baudry, Y. Le Traon, and G. Sunye, "Testability analysis of a UML class diagram," *Proceedings Eighth IEEE Symposium on Software Metrics*, 2002.

[30] T. L. Saaty, "Decision making with the analytic hierarchy process," *International Journal of Services Sciences*, vol. 1, no. 1, p. 83, 2008.

[31] P. Khanna, "Testability of Object-Oriented Systems : An AHP-Based Approach for Prioritization of Metrics," in *International Conference on Contemporary Computing and Informatics(IC3I)*, 2014, pp. 273–281.

[32] S. K. Dubey, A. Mittal, and A. Rana, "Measurement of Object Oriented Software Usability using Fuzzy AHP," *International Journal of Computer Science and Telecommunications*, vol. 3, no. 5, pp. 98–104, 2012.

[33] P. K. Singh, O. P. Sangwan, A. Pratap, and A. P. Singh, "Testability Assessment of Aspect Oriented Software Using Multicriteria Decision Making Approaches," *World Applied Sciences Journal*, vol. 32, no. 4, pp. 718–730, 2014.

[34] P. K. Singh, O. P. Sangwan, A. P. Singh, and A. Pratap, "A Quantitative Evaluation of Reusability for Aspect Oriented Software using Multi-criteria Decision Making Approach," *World Applied Sciences Journal*, vol. 30, no. 12, pp. 1966–1976, 2014.

[35] C. Yang, Y. Zheng, M. Zhu, Z. Zuo, X. Chen, and X. Peng, "A Testability Allocation Method Based on Analytic Hierarchy Process and Comprehensive Weighted," in *IEEE 9th Conference on Industrial Electronics and Applications (ICIEA)*, 2014, pp. 113–116.

[36] R. G. Dromey, "A Model for Software Product Quality," *IEEE Transactions on Software Engineering*, vol. 21. pp. 146–162, 1995.

[37] S. Khalid, S. Zehra, and F. Arif, "Analysis of object oriented complexity and testability using object oriented design metrics," in *Proceedings of the 2010 National Software Engineering Conference on - NSEC '10*, 2010, pp. 1–8.

[38] M. Nazir and K. Mustafa, "An Empirical Validation of Testability Estimation Model," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 9, pp. 1298–1301, 2013.

[39] L. Rosenberg and L. Hyatt, "Software quality metrics for object-oriented environments," *Crosstalk Journal, April*, vol. 10, no. 4, pp. 1–6, 1997.

[40] M. Nazir and R. A. Khan, "Software Design Testability Factors: A New Perspective," in *Proceedings of Third National Conference INDIACOM*, 2009, pp. 1–6.

[41] H. Singhani and P. R. Suri, "Object Oriented SoftwareTestability (OOSTe) Metrics Assessment Framework," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 5, no. 4, pp. 1096–1106, 2015.

[42] M. Bruntink, "Testability of Object-Oriented Systems : a Metrics-based Approach," Master's thesis, Faculty of Natural sciences, Mathematics, and Computer science, University of Amsterdam, 2003.

[43] M. Genero, M. Piattini, and C. Calero, "An Empirical Study to Validate Metrics for Class Diagrams," in *Proc. of International Database Engineering and Applications Symposium (IDEAS'02), Edmonton, Canada.*, 2002, pp. 1–10.

[44] M. Patidar, R. Gupta, and G. Chandel, "Coupling and Cohesion Measures in Object Oriented Programming," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 3, pp. 517–521, 2013.

[45] S. Mouchawrab, L. C. Briand, and Y. Labiche, "A measurement framework for object-oriented software testability," *Information and Software Technology*, vol. 47, no. April, pp. 979–997, 2005.

[46] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical study of object-oriented metrics," *Journal of Object Technology*, vol. 5, no. 8, pp. 149–173, 2006.

Dr. Pushpa R. Suri received her Ph.D. Degree from Kurukshetra University, Kurukshetra. She had recently retired as a Professor from the Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, Haryana, India. She has many publications in International and National Journals and Conferences. Her teaching and research activities include Discrete Mathematical Structure, Data Structure, Information Computing and Database Systems.

Harsha Singhani received her Master of Computer Application degree from Maharishi Dayanand University, Rohtak, Haryana, India. She has got experience of over 12 years of teaching in field of I.T. At present, she is pursuing Ph.D. (Computer Science) from Kurukshetra University, Kurukshetra, Haryana, India. Her teaching and research areas include database systems, automata theory, object oriented programming and software testing.