International Journal of Scientific & Engineering Research, Volume 6, Issue ł,)ÜÕÌ-2015
ISSN 2229-5518

Testability Assessment of Object Oriented Software Using Internal & External Factor Model and Analytic Hierarchy Process

Harsha Singhani, Dr. Pushpa R. Suri

Abstract —In this paper we have proposed a new testability assessment model for object oriented software based on existing software testability models. The model is based on those six important internal programming features of object oriented design and six external quality factors which are not used before together at the same time in-spite of being highlighted in some or other research. These design features are assessed using popular static object oriented metrics and their link with testability is established indirectly through the affected quality factors. The model is further analysed using Multi Criteria Decision Making (MCDM) approach. The model is 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 not only at internal level but also at external quality level too.

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. These models had 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 and much of the work has been done using various object oriented featured metrics. In this paper we have proposed a testability model for assessment at during design time and evaluated the same using AHP technique.

• *Harsha Singhani is currently pursuing Ph.D. (Computer Science) from Kurukshetra University, Kurukshetra, Haryana, India. PH-95400899999. E-mail: harshasinghani@gmail.com*

• *Dr. Pushpa R. Suri is a retired Professor from the Department of Computer Science and Applications Kurukshetra University, Kurukshetra, Haryana, India. E-mail: pushpa.suri@yahoo.com*

This paper is organized as follows: Section2 gives brief overview of software testability and AHP related work. Section3 showcases the proposed testability assessment model from design perspective based on significant object oriented programming features and external quality factors. Section4 provides overview of methodology applied for evaluation and implementation of the proposed model. Section5 presents the details of testability evaluation based on proposed model using AHP. It is followed by conclusion drawn in section 6.

2. Related Work

2.1Software Testability

Software Testability actually acts as a software support characteristic for making it easier to test. As stated by Binder [6] and Freedman [7] "a Testable Software is one that can be tested easily, systematically and externally at the user interface level without any ad-hoc measure". Voas [8] describes it as complimentary support to software testing by easing down the method of finding faults within the system by focusing more on areas that most likely to deliver these faults. Hence, over the years Testability has been diagnosed as one of the core quality indicators, which leads to improvisation of test process. The insight provided by testability at designing, coding and testing phase is very useful as this additional information helps in product quality and reliability improvisation [9], [10]. All this has led to a notion amongst practitioners that testability should be planned early in the design phase though not necessarily so. As stated by experts like Binder [6] that testability involves factors like controllability and observability

.Where controllability is the ability to control software input and state and observability is the possibility to observe the output and state changes that occur in software. So, overall testable software has to be controllable and observable [6]. But during our research we have found more such quality factors like complexity, traceability, understandability, and test–support capability have equally contributed to testability of a system[11].

Now any quality factor measurement refers to the activities and methods that study, analyze, and measure it during a software product development life cycle. Similar is the case with software testability measurement also. Unlike software testing, the major objective of software testability measurement is to find out where these faults are hiding from testing and highlighting specific components which are poor in quality. Now these measurements can be applied at various stages during software development life cycle of a software system.

In the past, there were a number of research efforts addressing software testability measurement. The focus of past studies was on how to measure software testability at various phases like Design Phase[6], [10], [12], [13], [14], [15], [16], [17] and Coding Phase [18], [19], [20], [21], [22]. Lot of stress has been given upon usage of object oriented metrics for object oriented software testability evaluation during these researches. It came out during the study that the metrics used related to object oriented software testability assessment mostly belong to static software metrics category. These metrics were mostly adapted from CK [23], MOOD [24], Brian [25], Henderson-Sellers [26] metric suite along with others [27]. Lot of empirical study has been done in showing the correlation of these metrics with unit testing effort [27], [28], [29], [30]. Few studies have been focused on UML diagram features from software testability improvisation prospect during review of these design diagrams [31], [32], [33], [34]. All this work has been explained in depth in our previous work [4], [5]. But very less work has been found using MCDM techniques, which is explained next.

2.2 Analytical Hierarchical Process

In context with software engineering problems, very few studies related to multi-criteria decision making (MCDM) approach has been done and published. Saaty [36] proposed AHP as one of the most practical method based on MCDM. AHP is mostly used when the criteria or factors and decision makers are small in number. 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 as proposed by Saaty [36], is 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. P. Khanna [37] 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. [38] have done study on object oriented usability with AHP. Though some work have been found for aspect oriented software testability and reusability assessment using MCDM technique done by Singh and Sangawan [39], [40] which has been technically found useful in how AHP needs to be applied in other software too, for study of other quality features. Yang [41] have also used this technique for analyzing and calculating hardware testability using comprehensive weighted method and AHP.

3. Testability Evaluation Model

Our testability model is based on Dromey's software quality model [42] 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.
- Identification of external factors affecting software testability
- Establishing link between theses external quality factors and internal features which are evaluated through selected object oriented metrics.
- Establishing link between testability and these identified external factors which indirectly link it to identified internal features.
- The Model is followed Evaluation using AHP technique.

On the basis of our previous research work and surveys we have identified six object oriented core features 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 defined below in Table 1.

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. Out of six identified features four features have been proposed in MTMOOD testability model [15], 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 as illustrated below in table 2. 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 Affecting Testability	Definition	Testability Relation
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	Encapsulation provides explicit barriers among different abstractions and thus leads to a clear separation of concerns. Thus if not used appropriately it makes system more complex and difficult to trace and test. But yes separation of concerns is good for testability.
Inheritance	It is a measure of the 'is-a' relationship between classes.	Inheritance has a significant influence on complexity, understandability, reusability and testability. Inheritance is one of the major test generation factors[28].
Coupling	It is defined as the interdependency of an object on other objects in a design.	Strong coupling complicates a system since a module is harder to understand, change, or correct by itself if it is highly interrelated with other modules. Thus low coupling is considered good for understandability, complexity, reusability and testability or maintainability
Cohesion	It defines as the internal consistency within the parts of design.	Cohesion is one of the measures of goodness or good quality in the software as a cohesive module is more understandable and less complex. Low cohesion is associated with traits in programming such as difficult to maintain, test, reuse, and even understand.
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.	Size & Complexity has a significant impact on understandability, and thus testability or maintainability of the system.
Polymorphism	Polymorphism allows the implementation of a given operation to be dependent on the object that "contains" the operation.	Polymorphism reduces complexity and improves reusability. More use of polymorphism leads more test case generation [28].

So, the proposed testability assessment model with respect to internal design features using static metrics is based on six above mentioned object oriented features from testability perspective as pointed in Binders research too [6]. 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. Out of all the popular metrics suites discussed in our previous work [48] 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.

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[15], [44]. 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], [28], [49]. Polymorphism is one of the underlying factors affecting testability but as quite stressed by early researchers like Binder and others [6], [25] as it results in testability reduction ,we chose polymorphism factor metrics (POF/PF) for testability assessment.

TABLE 2:
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

Keeping in mind our previous research work and surveys, we have identified six external quality factors to assess testability for object oriented software [4], [5]. These factors are –Controllability, Observability, Complexity, Understandability, Traceability and Built-in-Test. Most of these factors were pointed in Binder's [6] research work on testability. Many other researchers established these factors relation too with testability as mentioned below in table 3. These factors get directly or indirectly affected by all of the above mentioned internal features and further complicate or reduce the task of testing hence reducing or increasing overall testability of the software.

TABLE 3:
EXTERNAL SOFTWARE QUALITY FACTORS AFFECTING TESTABILITY

External Factors Affecting Testability	Definition	Significant Testability Related Work
Controllability	During software testing, some conditions like disk full, network link failure etc. are difficult to test. Controllable software makes it possible to initialize the software to desired states, prior to the execution of various tests.	High Complexity of the system is actually an indicator of decreased system testability [43], [44], [53], [54].
Observability	In the process of testing, there is a need to observe the internal details of software execution, to ascertain correctness of processing and to diagnose errors discovered during this process. Observable software makes it feasible for the tester to observe the internal behaviour of the software, to the required degree of details.	Observable software makes it feasible for the tester to observe the internal behaviour of the software, to the required degree of details, Hence observability increases testability in the system [7], [55], [56].
Complexity	It is basically is the difficulty to maintain, change and understand software.	BIT actually provides extra test capability within the code for separation of test and application functionality which makes software more testable by better controllability and improved observability [6], [18], [57], [58].
Understandability	It is the degree to which the component under test is documented or self-explaining.	Controllability is an important index of testability as it makes testing easier [7], [55], [56], [59].
Traceability	It is the degree to which the component under test is traceable.	A non-traceable software system cannot be effectively tested, since relations between required, intended and current behaviours of the system cannot easily be identified[6], [49].
Built In Test	Built in testing involves adding extra functionality within system components that allow extra control or observation of the state of these components.	An understandable system is easily testable and [13], [60]–[62].

We had identified the link between all the internal object oriented programming features which directly affect testability and all external quality factors which are also indicators of testable software too. The table given below actually elaborates the contribution of each of these internal programming features towards the six major quality factors which are directly linked to testability.

TABLE 4:
RELATION BETWEEN OBJECT ORIENTED PROGRAMMING FEATURES AND EXTERNAL SOFTWARE QUALITY FACTORS

External Factors Affecting	Internal OO Features Linked	Significant Effect On Testability
Controllability	Encapsulation	Encapsulation promotes controllability
	Coupling	Coupling makes controllability difficult
	Cohesion	Cohesion helps improving controllability
	Polymorphism	Polymorphism further reduces controllability
Observability	Encapsulation	Encapsulation reduces observability
	Inheritance	Inheritance help improving observability
	Polymorphism	Polymorphism & data hiding reduces observability
Complexity	Inheritance	Low Inheritance indicates more complex software
	Coupling	Highly coupled classes makes system more complex
	Cohesion	Cohesion amongst methods and classes help reducing complexity
	Size	Big size software and classes are more complex
	Polymorphism	Polymorphic data and methods helps reduce complexity
Understandability	Inheritance	Inheritance reduces understandability
	Coupling	Coupling makes system hard to understand
	Cohesion	Cohesion improves understandability
	Size	Large Size software are not easily understandable.
Traceability	Encapsulation	Encapsulation makes traceability difficult
	Coupling	Coupling makes traceability of test requirement hard
	Size	Increase in size reduces traceability
Built In Test	Encapsulation	More Encapsulation requires more built in test
	Coupling	Coupling increases built in test count
	Cohesion	Cohesion reduces the need of built in test

Hence we may say that Testability requires Low Coupling, Adequate Complexity, Good Understandability, High Traceability, Good Observability, Adequate Control and more Built in test. In spite of having lot of measurement techniques for testability evaluation using some or the factor using few of the above mentioned metrics, but testability has not yet been found to be evaluated from these factor perspectives. The study still does not show an elaborative impact of all of them together for testability improvisation or test effort reduction which is what motivated us for proposing this new model.
So, the proposed testability assessment model with respect to internal design features using static metrics is based on six above mentioned object oriented features from testability perspective as pointed in Binders research too [6]. The proposed model is as follows:

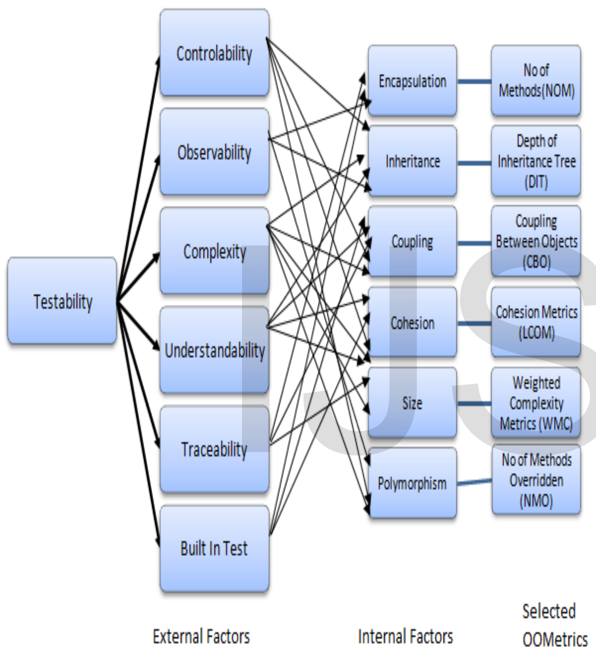

Fig 1. Testability Assessment Model

4. 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, F1, F2….Fn are considered, which are to be compared. Relative weight of Fi relating to Fj denoted as mij and a square matrix A = [mij] of order n will be formed as given in equation (1) below.

$$A = \overset{F_1}{\underset{F_n}{\left[m_{ij} \right]}} = \begin{array}{c} \\ \\ F_2 \\ \\ \\ \end{array} \begin{pmatrix} F_1 & F_1 & . & F_1 \\ 1 & m_{12} & . & m_{1n} \\ 1/m_{12} & 1 & . & m_{2n} \\ . & . & . & . \\ 1/m_{1n} & 1/m_{2n} & . & 1 \end{pmatrix} \tag{1}$$

Here, mij =1/mji and i does not equal to j and mii =1 for all i. Hence the calculated matrix is known as reciprocal matrix.

TABLE 5:
SATTY RATING SCALE [36]

Intensity of Importance	Definition	Description
1	Equal Importance	Elements Ci and Cj are equally important
3	Weak Importance of Ci over Cj	Experience and Judgment slightly favor Ci over Cj
5	Essential or Strong Importance	Experience and Judgment strongly favor Ci over Cj
7	Demonstrated Importance	Ci is very strongly favored over Cj
9	Absolute Importance	The evidence favoring Ci over Cj 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 Ci has one of the above judgments assigned to it when compared with Cj, then Cj has the reciprocal value when compared with Ci	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.

$$\mathbf{A \cdot \omega = \lambda_{max}\, \omega \;,\; \lambda_{max} = n} \tag{2}$$

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

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

$$\mathbf{CI} = \frac{(\lambda_{\max} - n)}{n-1} \qquad \textbf{(3)}$$

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

$$\mathbf{CR} = \frac{\mathbf{CI}}{\mathbf{RI}} \qquad \textbf{(4)}$$

TABLE 6:
SAATY SCALE OF AVERAGE CONSISTENCY INDEX (RI) [36]

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

Testability is an important attribute to the maintainability of software. Testable software is easy and less costly to maintain and testability represents an important software quality characteristics.

In order to conduct testability study based on above model and AHP technique. The hierarchical model has been made with six external factors- Controllability (F1), Observability (F2), Complexity (F3), Understandability (F4), Traceability (F5), Built-In-Test (F6) and six internal programming features as sub-factors where sub-factors contribute to testability directly as well as indirectly by affecting these quality factors.

As discussed above these six features are– Encapsulation (SF1), Inheritance (SF2), Coupling (SF3), Cohesion (SF4), Size & Complexity (SF5) and Polymorphism (SF6) has been shown below in fig2. In order to assign weights to these six major quality factors that affect testability a thorough study of object oriented software testability factors has been done along with discussion with concerned experts from industry and academia. The conclusive values are thus fed in 6x6 matrixes of these major factors as given below in Table 7. On the basis of this matrix, eigen value, eigen vector, consistency ratio and consistency index calculations, we have been able to evaluate weights for all these factors as shown below in detail.

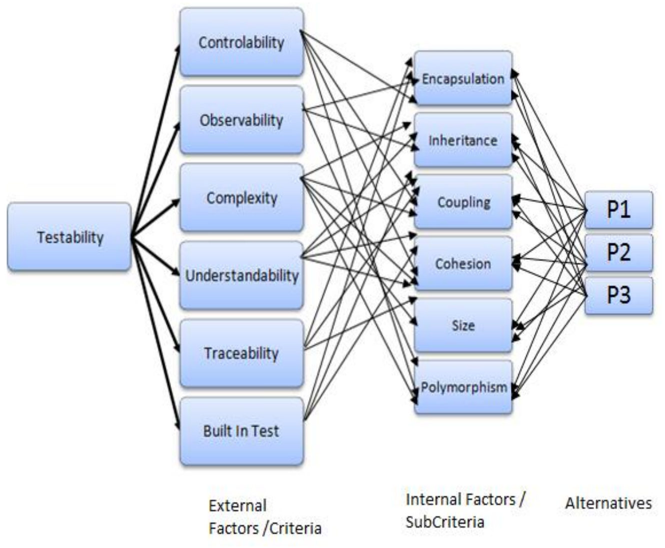

Fig 2. Object Oriented Software Testability Assessment Hierarchy for AHP Analysis

5. EVALUATION OF TESTABILITY MODEL USING AHP

5.1 Proposed Model Evaluation

The pair wise comparative value of all the six major factors affecting testability is given below in Table 7. There are many methods for calculating the eigenvector. We have used spreadsheet based approximate calculations for local priorities of criteria or factors. The Eigen value thus calculated are as shown below in table 7. The eigenvector of the relative importance of F1, F2, F3, F4, F5 and F6 is (0.08, 0.37, 0.04, 0.25, 0.14, 0.13). These values are weights of main factors i.e. Controllability (0.08), Observability (0.37), Complexity (0.04), Understandability (0.25), Traceability (0.14) and Built-In-Test (0.13) in testability assessment.

Now the six eigen values calculated for each of these factors is (6.19, 6.63, 6.19, 6.68, 6.56, 6.26) with λ_{max}=6.42 which is >= 6 (total no of factors), which is consistent. Using this we calculate the CI and CR values as follows:

$$\mathbf{CI} = \frac{(\lambda_{\max} - n)}{n-1} = \frac{6.42-6}{6-1} = \mathbf{0.08} \qquad \textbf{(5)}$$

$$\mathbf{CR=CI/RI=0.08/1.24=0.07} \qquad \textbf{(6)}$$

TABLE 7:
EIGEN VECTOR AND EIGEN VALUE FOR MAIN QUALITY FACTORS

	F1	F2	F3	F4	F5	F6	Eigen Vector
F1	1.00	0.25	3.00	0.33	0.50	0.33	0.08
F2	4.00	1.00	5.00	3.00	3.00	3.00	0.37
F3	0.33	0.20	1.00	0.17	0.33	0.25	0.04
F4	3.00	0.33	6.00	1.00	2.00	4.00	0.25
F5	2.00	0.33	3.00	0.50	1.00	2.00	0.14
F6	3.00	0.33	4.00	0.25	0.50	1.00	0.13

(λmax =6.42, CI=0.08, CR=0.07)

As the calculated value of consistency ratio is found to be CR<0.1 indicates that the estimate is consistent and acceptable.

Now all these factors are further dependent on one or more core object oriented features which act as sub-factors in the proposed model. These sub-factors as discussed above in table 4 affect the quality attributes which contribute to testability. The demonstrative pair-wise comparison matrix of respective sub-factors for each of these factors individually is given below from Table 8-Table 13.Table 8 reflects Controllability (F1) factor matrix analysis from four sub-factors perspective, similarly Table 9 reflects Observability (F2) and three respective sub-factors comparative effect on it. Table 10 and 11 gives pair-wise comparative values of four sub-factor affecting Complexity (F3) and Understandability (F4) respectively. In Table 12 and 13 we have given comparative values of three sub-factors each for Traceability (F5) and Built-in-Test (F6). The eigen value, λ_{max} , CI and CR values for each individual factor matrix is calculated as per above mentioned method.

TABLE 8:
EIGEN VECTOR & VALUES FOR SUB-FACTORS OF CONTROLLABILITY

	SF1	SF3	SF4	SF6	Eigen Vector
SF1	1.00	3.00	0.50	2.00	1.20
SF3	0.33	1.00	0.33	0.33	0.40
SF4	2.00	3.00	1.00	2.00	1.69
SF6	0.50	3.00	0.50	1.00	0.85

(λ_{max} =4.12, CI=0.04, CR=0.05)

TABLE 9:
EIGEN VECTOR & VALUES FOR SUB-FACTORS OF OBSERVABILITY

	SF1	SF2	SF6	Eigen Vector
SF1	1	0.33	2	0.268013
SF2	3	1	2	0.537374
SF6	0.5	0.5	1	0.194613

(λ_{max} =3.008, CI=0.005, CR=0.008)

TABLE10:
EIGEN VECTOR & VALUES FOR SUB-FACTORS OF COMPLEXITY

	SF2	SF3	SF4	SF6	Eigen Vector
SF2	1	0.25	3	2	0.23
SF3	4	1	4	3	0.43
SF4	0.33	0.25	1	0.5	0.07
SF6	0.5	0.33	2	1	0.16

(λ_{max} =5.18, CI=0.04, CR=0.04)

TABLE 11:
EIGEN VECTOR & VALUES FOR SUB-FACTORS OF UNDERSTANDABILITY

	SF2	SF3	SF4	SF5	Eigen Vector
SF2	1.00	3.00	0.50	0.50	0.21
SF3	0.33	1.00	0.33	0.33	0.10
SF4	2.00	3.00	1.00	2.00	0.41
SF5	2.00	3.00	0.50	1.00	0.29

(λ_{max} =4.12, CI=0.04, CR=0.05)

TABLE 12:
EIGEN VECTOR & VALUES FOR SUB-FACTORS OF TRACEABILITY

	SF1	SF3	SF5	Eigen Vector
SF1	1	4	3	0.62
SF3	0.25	1	2	0.22
SF5	0.33	0.5	1	0.16

(λ_{max} =3.04, CI=0.02, CR=0.03)

TABLE 13:
EIGEN VECTOR & VALUES FOR SUB-FACTORS OF BUILT-IN–TEST

	SF1	SF3	SF4	Eigen Vector
SF1	1	0.5	0.2	0.12
SF3	2	1	0.33	0.23
SF4	5	3	1	0.65

(λ_{max} =3.004, CI=0.002, CR=0.003)

The Eigen vector values for all of these six selected quality factors and their respective sub-factors matrix are found to be within acceptable limits. All six CR values are <0.1, hence result is consistent and applicable.

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 [52] which consists of three standard object oriented projects. Table 14 below shows the gathered metric value for each of the above internal object oriented programming features which affect the selected quality factors. 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 these object oriented projects for each testability factor individually.

TABLE 14:
THREE PROJECT METRICS VALUES

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

The eigen vector values for all three projects with respect to six testability assessment sub-factors - Encapsulation (Table15), Inheritance (Table16), Coupling (Table17), Cohesion (Table18), Size& Complexity (Table 19) and Polymorphism (Table20) 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. These eigen vector values are utilised in evaluating global utility of each project and its overall rank.

TABLE 15:
PAIR-WISE COMPARISON MATRIX OF THREE OO PROJECTS FOR ENCAPSULATION (SF1)

	P1	P 2	P 3	Eigen Vector
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

(λ_{max} =3.09,CI=0.04,CR=0.07)

TABLE 16:
PAIR-WISE COMPARISON MATRIX OF THREE OO PROJECTS FOR INHERITANCE (SF2)

	P1	**P2**	**P3**	**Eigen Vector**
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 17:
PAIR-WISE COMPARISON MATRIX OF THREE OO PROJECTS FOR COUPLING (SF3)

	P1	**P2**	**P3**	**Eigen Vector**
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 18:
PAIR-WISE COMPARISON MATRIX OF THREE OO PROJECTS FOR COHESION (SF4)

	P1	P2	P3	**Eigen Vector**
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 19:
PAIR-WISE COMPARISON MATRIX OF THREE OO PROJECTS FOR SIZE (SF5)

	P 1	P 2	P 3	**Eigen Vector**
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

(λ_{max} =3.09, CI=0.04, CR=0.07)

TABLE 20:
PAIR-WISE COMPARISON MATRIX OF THREE OO PROJECTS FOR POLYMORPHISM (SF6)

	P 1	P 2	P 3	**Eigen Vector**
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)

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 =

$\sum \sum_{i=1}^{n}$ Weight value of SFi $*$ Comparative value of Pi **(7)**

For example: U (P1) =

=(0.619*0.023+0.688*0.008+0.272*0.032+0.627*0.017) + (0.619*0.098+0.284+0.197+0.627*0.071) + (0.284*0.010+0.688*0.018+0.272*0.003+0.619*0.005+0.627*0.007) + (0.284*0.051+0.688*0.024+0.272.0.100+0.619*0.071) + (0.619*0.087+0.688*0.032+0.619*0.022) + (0.619*0.037+0.688*0.068+0.272*0.021) **(8)**

= 0.039 +0.162+0.023+0.098+ 0.088+0.075 = 0.485 **(9)**

The Analytical Hierarchy Process has shown in this study that project 1 is most testable project amongst all three projects followed by project 3. The three object oriented project testability study presented here.

TABLE 21:
EIGEN VECTOR AND WEIGHTS FOR OVERALL GLOBAL UTILITY

Factors	**Weights For Factors**	**Sub-Factors**	**Global Weights For Sub Factors**	**Testability Comparison At Sub-Factor Level**			**Overall Comparison Between Projects**		
				P1	**P2**	**P3**	**P1**	**P2**	**P3**
F1	0.08	SF1	0.023	0.619	0.096	0.284	0.039	0.009	0.032
		SF3	0.008	0.688	0.234	0.078			
		SF4	0.032	0.272	0.120	0.608			
		SF6	0.017	0.627	0.081	0.292			
F2	0.37	SF1	0.098	0.619	0.096	0.284	0.162	0.137	0.068
		SF2	0.197	0.284	0.619	0.096			
		SF6	0.071	0.627	0.081	0.292			
F3	0.04	SF2	0.010	0.284	0.619	0.096	0.023	0.012	0.008
		SF3	0.018	0.688	0.234	0.078			
		SF4	0.003	0.272	0.120	0.608			
		SF5	0.005	0.619	0.096	0.284			
		SF6	0.007	0.627	0.081	0.292			
F4	0.25	SF2	0.051	0.284	0.619	0.096	0.098	0.056	0.091
		SF3	0.024	0.688	0.234	0.078			
		SF4	0.100	0.272	0.120	0.608			
		SF5	0.071	0.619	0.096	0.284			
F5	0.14	SF1	0.087	0.619	0.096	0.284	0.088	0.018	0.035
		SF3	0.032	0.688	0.234	0.078			
		SF5	0.022	0.619	0.096	0.284			
F6	0.13	SF1	0.037	0.619	0.096	0.284	0.075	0.022	0.028
		SF2	0.068	0.688	0.234	0.078			
		SF3	0.021	0.272	0.120	0.608			
Total	1.00	**Overall Global Utility /Priority**					**0.485**	**0.254**	**0.261**
		Project Ranking					**1**	**3**	**2**

CONCLUSION

In this paper we have proposed an object oriented software Testability assessment model. The model has been based on six external quality factors identified from the previous research work which are further linked with basic object oriented programming features which indirectly contributes to testability too. The identified quality factors are Controllability, Observability, Complexity, Understandability, Traceability and Built-in Test which are further dependent on internal sub factors namely Encapsulation, Inheritance, Coupling, Cohesion, Size-Complexity and Polymorphism. Furthermore all these factors were linked to suitable static object oriented metrics. This is being done to evaluate the comparative values of factor and sub-factors matrix values to be used for evaluation the assessment model using AHP technique. The evaluation is further applied on three object oriented medium sized projects for identifying and ranking most testable project. The overall testability index is further calculated for all projects.

The study can be extended further by gathering comparative values of characteristics from running projects, which are developed using OO technology. Though, the projects, which are compared here, are not so large. 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.

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] R. S. Freedman, "Testability of software components - Rewritten," *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 553–564, 1991.

[8] J. M. Voas and K. W. Miller, "Software Testability : The New Verification," *IEEE Software*, vol. 12, no. 3, pp. 17–28, 1995.

[9] J. M. Voas and K. W. Miller, "Improving the software development process using testability research," in *Software Reliability Engineering,1992. Third International Symposium on. IEEE*, 1992, pp. 114–121.

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

[11] J. Fu, B. Liu, and M. Lu, "Present and future of software testability analysis," *ICCASM 2010 - 2010 International Conference on Computer Application and System Modeling, Proceedings*, vol. 15, no. Iccasm, 2010.

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

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

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] M. Joshi and N. Sardana, "An Enhanced Marking Target Statement Strategy of E-PIE for Testability Estimation," in *Contemporary Computing (IC3), 2014 Seventh International Conference on. IEEE*, 2014, pp. 346–350.

[23] 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.

[24] T. Mayer and T. Hall, "Measuring OO systems: a critical analysis of the MOOD metrics," *Proceedings Technology of Object-Oriented Languages and Systems. TOOLS 29 (Cat. No.PR00275)*, 1999.

[25] 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.

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

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

[28] 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.

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

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

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

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] 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.

[44] 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.

[45] 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.

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

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

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] J. M. Voas, J. M. Voas, K. W. Miller, K. W. Miller, J. E. Payne, and J. E. Payne, "An Empirical Comparison of a Dynamic Software Testability Metric to Static Cyclomatic Complexity," 1993.

[54] S. A. Khan and R. A. Khan, "Object Oriented Design Complexity Quantification Model," *Procedia Technology*, vol. 4, pp. 548–554, 2012.

[55] T. B. Nguyen, M. Delaunay, and C. Robach, "Testability Analysis of Data-Flow Software," *Electronic Notes in Theoretical Computer Science*, vol. 116, pp. 213–225, 2005.

[56] S. Kansomkeat, J. Offutt, and W. Rivepiboon, "INCREASING CLASS-COMPONENT TESTABILITY," in *Proceedings of 23rd IASTED International Multi-Conference*, 2005, pp. 15–17.

[57] T. Jeon, L. Sungyoun, and H. Seung, "Increasing the Testability of Object-Oriented Frameworks with Built-in Tests," in *Advanced Internet Services and Applications. Springer Berlin Heidelberg,* 2002, pp. 169–182.

[58] J. Vincent, G. King, P. Lay, and J. Kinghorn, "Principles of Built-In-Test for Run-Time-Testability in Component-Based Software Systems," *Software Quality Journal*, vol. 10, no. 2, pp. 115–133, 2002.

[59] A. Goel, S. C. Gupta, and S. K. Wasan, "COTT – A Testability Framework for Object- Oriented Software Testing," *International Jounal of Computer Science*, vol. 3, no. 1, pp. 813–820, 2008.

[60] M. Nazir, R. A. Khan, and K. Mustafa, "A Metrics Based Model for Understandability Quantification," *Journal of Computing*, vol. 2, no. 4, pp. 90–94, 2010.

[61] J. Bach, "Test Plan Evaluation Model," *Satisfies Inc.*, 1999. .

[62] J. Bach, "Heuristics of Software Testability," *Satisfice, Inc.*, 2003. .