October 2014

OCL FORMAL SPECIFICATION BASED METRICS AS A MEASURE OF COMPLEXITY AND FAULT-PRONENESS

A. JALILA

Department of Computer Applications, Thiagarajar College of Engineering, Tamilnadu – 625 015,
mejaila@gmail.com

D. JEYA MALA

Department of Computer Applications, Thiagarajar College of Engineering, Madurai, India, djmce@tce.edu

S. BALAMURUGAN

Department of Computer Applications, Thiagarajar College of Engineering, Tamilnadu – 625 015,
balamsg4u@gmail.com

K. SABARI NATHAN

Department of Computer Applications, Thiagarajar College of Engineering, Tamilnadu – 625 015,
sabarinathan4you@gmail.com

Follow this and additional works at: <https://www.interscience.in/ijcsi>

Part of the [Computer Engineering Commons](#), [Information Security Commons](#), and the [Systems and Communications Commons](#)

Recommended Citation

JALILA, A.; MALA, D. JEYA; BALAMURUGAN, S.; and NATHAN, K. SABARI (2014) "OCL FORMAL SPECIFICATION BASED METRICS AS A MEASURE OF COMPLEXITY AND FAULT-PRONENESS," *International Journal of Computer Science and Informatics*: Vol. 4 : Iss. 2 , Article 6.

DOI: 10.47893/IJCSI.2014.1182

Available at: <https://www.interscience.in/ijcsi/vol4/iss2/6>

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Computer Science and Informatics by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.

OCL FORMAL SPECIFICATION BASED METRICS AS A MEASURE OF COMPLEXITY AND FAULT-PRONENESS

A.JALILA¹, D. JEYA MALA², S.BALA MURUGAN³, K.SABARI NATHAN⁴

^{1,2,3,4}Department of Computer Applications, Thiagarajar College of Engineering, Madurai, India
E-mail: ¹mejaila@gmail.com, ²djmce@tce.edu, ³balams4u@gmail.com, ⁴sabarinathan4you@gmail.com

Abstract- Formal specification of UML models in OCL is essential to improve software quality. Owing to the use of OCL in precise model specification, its application has been looked in different perspectives such as early measurement of module complexity. Moreover, when UML class diagrams are complemented with OCL, the metrics collected from OCL specification can serve as an indicator of fault-prone components. In the proposed approach an empirical study has been conducted on five soft real time case study applications. In this paper, existing metrics which are applicable to OCL expression are validated using module complexity. Moreover, a new metrics suite, which can be extracted from OCL expressions, has been devoted to quantify module complexity. The proposed metrics suite can be directly extracted from OCL expressions. Relative weight has been assigned to each metric which is selected for the proposed study, based on its importance in fault-prone components identification. The study shows that an analysis on OCL formal specification based metrics is effective in identifying fault-prone components of the system. Furthermore, it helps to distribute efforts required for software development and testing activities.\

Keywords- Critical Components, UML (Unified Modeling Language), Formal Specification, OCL (Object Constraints Language), Design Metrics.

I. INTRODUCTION

The static structure of software system can be represented using UML class diagram which describes classes, interfaces and collaboration among components in an object oriented system. However, a software specification based only on UML diagram is not sufficient to explore system structure. Thus, software industries exercise model-based formal specification languages such as OCL, Z etc., for precise software specification, to improve software quality. OCL is based on mathematical set theory and predicate logic which is used to express constraints on UML models. Though OCL specification has found its application in precise software modelling its usage can also be extended to support effective testing. Fault-prone component based testing is an effective technique to improve software quality. Fault-prone components are the components which are central to the system's operation.

They may not be operationally critical but are critical from a resilience perspective.

The importance of fault-prone components identification in software quality improvement has been widely studied [1][2][3][4][5]etc., Earlier research works have widely acknowledged that object oriented design [1] [2] [3] and source code based metrics [4], [5], [6], [7] are the quantifying factors to predict fault-prone modules. However, the proposed study insists that OCL formal specification based metrics also can predict fault-prone modules of the system effectively during the initial stages of software life cycle. Measuring design metrics from OCL is preferable due to the following reasons.

- Prediction of fault-prone components during the design phase is more effective than its prediction after the coding or pretesting phases.
- UML design models represent more abstract level of the system. Hence, it is difficult to directly measure and automate metrics extraction from UML model.
- OCL is platform independent and it defines semantics of the system with specific standard
- OCL formal specification supports simple and well structured code development.
- When OCL is executed it does not change the state of the system.

The following are the major contributions of this paper

1. Recognition of the existing object oriented design metrics which are applicable to OCL expression.
2. Investigation of the significance of existing OCL based object-oriented metrics in critical components identification across fault severity.
3. Proposing new metrics which can be extracted directly from OCL expressions.
4. Assigning relative weight to each test metrics based on its importance in complexity estimation.
5. Comparing the effectiveness of the test metrics and the proposed OCL based metrics in fault-prone components prediction.

There are three major findings as observed from this study. First, few of the existing object oriented design metrics which are applicable to OCL specification are significant to measure the complexity of a system. Second, the proposed OCL based metric can also be used to measure the complexity of a system. Third, the proposed OCL specification based metrics are highly significant to identify fault-prone components of the system than the test metrics. For the purpose of this work, the terms component, class and module have been used interchangeably. Also, fault-prone component would also mean critical component or

error prone component. The remainder of this paper is organized into the following sections. Section II starts deals with a description of related earlier work. Five different real-time case studies identified for this work have been introduced in Section III. An empirical data analysis method has been elaborated in Section IV. The process of metrics extraction has been explained in Section V. Section VI and VII discuss the existing and proposed metrics respectively. The performance evaluation study has been detailed in Section VIII and Section IX has drawn conclusions out of this study.

II. RELATED WORK

Earlier research works have used statistical and machine learning approaches to classify components as fault-prone and non- fault prone [8], [9] and [10]. Ray and Mohapatra [1] have proposed an analytical method for reliability-based risk assessment of a software system at the architectural level which is based on UML sequence diagram and state chart diagram. In their work they have considered risk associated with various states of a component, message criticality and business risk to identify high risk components. El Emam et al. [4] have validated object-oriented design metrics on a commercial Java system for detecting faulty components at the early stage of software development. Also, they have depicted that an inheritance metric and an Export Coupling (EC) metric were strongly associated with error-prone modules prediction. Ray et al. [11] have proposed a software component prioritization method as a pretesting activity to prioritize critical components. They have predicted critical components in source code of the system based on two criteria such as influence metric and operational profile. According to their approach, the influence metric value is derived from the extended system dependency graph (ESDG) by applying forward slicing technique. Tang et al. [8] have validated object-oriented design metrics suite which was proposed by Chidamber and Kemerer (CK) by using three types of faults namely object-oriented faults, object management faults and traditional faults. They have proposed a set of new metrics that can serve as an indicator of fault- prone components of a system. Arisholm Erik et al. [9] have compared many data mining and machine learning techniques to build fault-proneness models, compared the impact of using different metric sets including source code structural measures and change/fault history types of measures as predictors, based on different evaluation criteria. Malhotra et al. [10] have applied statistical and machine learning methods to predict fault-prone components of a system. They have analyzed the results using Area Under the Curve (AUC). The results showed that the model predicted using the random forest and bagging methods provides more accurate result.

Olague et al. [12] have done study which has indicated that, the CK and QMOOD object oriented class metrics suites are useful in developing quality classification models to predict defects in both traditional and highly iterative, or agile, software development processes for both initial delivery and for multiple, sequential releases. WMC (Weighted Method Per Class), RFC(Response for a Class), CIS (Class Interface Size), and NOM (Number of Methods) are consistent predictors of class quality and error-proneness. CK metrics have been shown to be better and more reliable predictors of fault-proneness than the MOOD or QMOOD metrics. Shatnawi et al. [13] have experimented the effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. In their study they have tested software metrics namely CBO (Coupling Between Objects), CTA (Coupling through Abstract Data Type), CTM (Through Message Passing), RFC, WMC, DIT, NOC etc. They proved that software metrics are used to identify error- prone classes even after the software release evolution process With respect to the existing literature, the main contribution of this work is to prove that, the OCL formal specification based metrics are expressive enough for classification of the components as fault-prone and non- fault prone. \

III. DESCRIPTION OF THE CASE STUDY APPLICATIONS

For the experimental purpose of this research work, the specifications of five real-time applications have been selected. These five case study projects are developed in Java language and are referred as Case study 1, Case study 2, and Case study 3. Case study 4, Case study 5. The Case study 1 is a Blood Bank Management System (BBMS), Case study 2 is a Patient Monitoring System (PMS), Case study 3 is a Library Management System (LMS). Case study 4 is a Banking Management System (BMS) and Case study 5 is an E-commerce System (E-com). These five case study applications are neither very large nor small but of medium size. By means of rigorous analysis of several real-time applications related to these five case study applications, the components are classified into two types namely fault-prone and non-fault prone. The classification scheme is based on the module complexity and its dependability. A brief summary of these case study applications is presented in Table 1.

TABLE 1
BRIEF SUMMARY OF CASE STUDY APPLICATIONS

Particulars	Case Study 1 (BBMS)	Case Study 2 (PMS)	Case Study 3 (LMS)	Case Study 4 (BMS)	Case Study 5 (E-com)
No of components	19	22	15	19	32
No of fault- prone components	3	5	4	4	6
Fault-prone Components List	Admin Bstoc Donor	Appointment Diagnosis Treatment Billing Registration	Admin Membership Transaction Issuebook	Checkbalance Deposit Withdraw Admin	Admin Order Stock Bank Sales Payment

IV. DATA ANALYSIS METHODOLOGY

The purpose of data analysis is to empirically analyze the significance of the test metrics which are applicable to OCL formal specification and the

proposed metrics, in predicting fault-prone components of the system. The technique used to analyze the data collected for each measure is described in three stages:

Expression 1:

```

context person::personinfo(pid:Integer)
pre: self.isStaic(pid)
pre: name.isDefined()
pre: dod.isDefined()
pre: gender.isDefined()
pre: bgroup.isDefined()

```

Expression 2:

```

context person::personid()
pre: self.pid > 0
pre: pid.isDefined()
post: self.pid = self.pid+pre + pid

```

Expression 3:

```

context donar::donarinfo(pid:Integer, bloodgroup:String)
pre: medicalFitness.isDefined()

```

Expression 4:

```

context donar:: Donate(d:donor)
pre DonarDefined: d.isDefined()
pre isAPositive: bgroup=#APositive
pre isBPositive: bgroup=#BPositive
...

```

Expression 5:

```

context receipient::receipientinfo(pid:Integer, bloodgroup:String)
pre: reason.isDefined()
pre: empid.isDefined()

```

Expression 6:

```

context bloodstock ::updatestock(r: bloodrequest)
pre: stkid.isDefined()
pre: date.isDefined()
pre: bloodgroup.isDefined()
pre: Apositive.isDefined()
pre: self.isPrivate(stkid)
pre: self.Apositive > 0
post: Apositive=Apositive-r.Apositive

```

Expression 7:

```

context bloodstock::stockinfo(stkid:Integer)
pre: self.isPrivate(stkid)
pre: date.isDefined()
pre: bloodgrpoup. isDefined()
Pre: Apositive.isDefined()

```

Expression 8:

```

context administrator::checkstock(s: bloodstock, r: bloodrequest)
pre: s.Apositive > 0
pre: s.Bpositive > 0
post: s.Apositive > r.Apositive
post: s.Bpositive > r.Apositive
...

```

Expression 9:

```

context donor
inv inv1: self. bloodstock ->size()>0

```

Expression 10:

```

context bloodstock
inv VerifyRequest2: self.bloodrequest.Bpositive<=self.Bpositive
inv VerifyRequest3: self.bloodrequest.ABpositive<=self.ABpositive

```

Expression 11:

```

context administrator::acceptd(p : donor)
pre acceptPre1: p.isDefined()
pre acceptPre2: donor->excludes(p)
post acceptePost: donor->includes(p)

```

Expression 12:

```

context administrator::acceptr(r : recipient)
pre rejectPre: recipient ->includes(r)
post rejectPost: recipient->excludes(r)

```

Fig.1 Example for metrics extraction from OCL expressions

- Descriptive statistics methods which include minimum, maximum, mean, median, standard deviation, 25% quartile, 50% quartile, 75% quartile
- Correlation analysis based on Spearman's Rho method is used to relate each metrics in complexity estimation.
- Performance measurement based on the performance measures such as sensitivity, specificity and precision [10] to find the significance of existing design metrics and OCL based metrics in fault-prone components prediction.

Sensitivity=

$$\frac{\text{Number of components correctly predicted as fault - prone}}{\text{Total number of actual fault - prone components}} \quad (1)$$

Specificity =

$$\frac{\text{Number of components correctly predicted as non - fault prone}}{\text{Total number of actual non fault - prone components}} \quad (2)$$

Precision =

$$\frac{\text{Number of components predicted correctly as fault - prone or non - fault prone}}{\text{Total number of components}} \quad (3)$$

V. METRICS EXTRACTION FROM OCL EXPRESSIONS

In general, there are two major segments in OCL specification; they are method definition (which includes pre, post-conditions) and invariant

declaration. In this research work, it has been endeavoured that the metrics related to the size of a component can be estimated based on its method definition. Let there be a component namely C, containing a

method M. This method M can be defined using OCL expression as follows.

```
context C :: M(v1 : T1, ..., vn : Tn) : RT
pre : PreC
post : PostC
```

where, $V = \{v1...vn\}$ are the formal parameters of the method M. $T = \{T1...Tn\}$ are the parameter types, RT is the return type for the method M. PreC is the pre-conditions, PostC is post conditions of method M. PreC and PostC are optional.

Moreover, the association between classes can be estimated by means of both method definition and invariants declaration segments in OCL expression. The invariant of a component C can be defined using OCL expression as follows.

```
context C inv invName : Ct or
context C inv invName : self.Dc.Dt → Ct
```

Where, Dc is the associated class name, Dt is the associated class attribute and Ct is the condition as applicable to the class C.

VI. EXISTING OBJECT ORIENTED DESIGN METRICS

In this proposed study, 15 object-oriented design metrics have been selected from among multitude of such metrics. All these metrics have received reasonable attention from researchers to realise module complexity, given the source code and design documents. As a novel approach, these metrics have been extracted from OCL formal specification of the system. Among these fifteen metrics, the metrics such as Depth of Inheritance (DOI) and Number of Children (NOC) were proposed by Chidamber and Kemerer (CK) [14]. The three metrics such as Direct Class Coupling (DCC), Data Access Metric (DAM) and Number of methods (NOM) were proposed by Bansiya et al. [15] and metrics such as NOF, NOP were listed by Zimmermann et al. [16] as complexity measures which are related to the field failures and dependencies.

The metrics such as Fan-In and Fan-Out and Information Flow (IF) were used to measure the structural complexity of the program which were proposed by Hendry et al. [17]. The other three metrics such as Number of methods defined in a subclass (NMA), Number of method inherited by a subclass (NMI), The Average Parameters Per Method metric (APPM) were proposed by Lorenz et al. [18]. Martin [19] has proposed Instability (IM) Metric. Metrics such as NOC, DOI, NOM, NOF, NOP, NSFM, NMA, APPM and NMI can be derived from method declaration of OCL expressions and the other metrics such as DCC, Fan-In and Fan-out, IF, IM can be extracted from pre and post condition and invariant expressions in OCL expressions.

In the remainder of this section, definition for each of the test metrics has been provided, along with its corresponding mathematical notation. Also, the process of test metrics extraction from OCL specification has been explained.

A) Test metrics extraction from OCL expressions

This section describes the new way of extracting existing design metrics from OCL expressions.

1) *NOM metric*: This metric indicates the total number of methods defined in a class. This metric can be extracted from method specification segment in the OCL expression. Let a class C contain methods $m = \{1...n\}$, then the total number of methods in class C is denoted as

$$NOM(C) = \sum_{m=1}^n C_m \quad (4)$$

Example 1: Expressions 1 and 2 of Figure 1 represent that, there are two methods such as *personinfo()* and *personid()* defined for class *Person*. Hence its $NOM = 2$.

2) *NOF metric*: This metric specifies the total number of attributes defined in a class. This metric can be extracted from method specification section in the OCL expression. Let a class C contains attributes $f = \{1...n\}$, then the total number of attributes in a class C is expressed by

$$NOF(C) = \sum_{f=1}^n C_f \quad (5)$$

Example 2: Expressions 1 of Figure 1 depicts that there are five attributes defined for class *bloodstock* and hence its $NOA = 5$.

3) *NOP metric*: This metric denotes the total number of parameters defined in each method of a class. Let a class C contain methods $M = \{1...n\}$ and each method pass parameters $p = \{0...k\}$, then the total number of parameters in a class C is given by

$$NOP(C) = \sum_{M=1}^n \sum_{p=0}^k M_p(C) \quad (6)$$

Example 3: Expressions 5 and 6 of Figure 1 represent that class *bloodstock* has two methods (*stockinfo*, *updatestock*) of which the method *stockinfo* has a parameter *stkid* and the method *updatestock* passes a parameter *r: bloodrequest*. Hence, NOP of class *bloodstock* is 2.

4) *NSFM metric*: This metric signifies the total number of static attributes and static methods in a class. This metric can be extracted from pre-condition definition part of the OCL expression. Let a class C contain static fields $sf = \{0...n\}$ and static methods $sm = \{0...m\}$, then the total number static fields and methods in a class is given by

$$NSMF(C) = NSF(C) + NSM(C) \quad (7)$$

$$NSF(C) = \sum_{sf=0}^n C_{sf} \quad (8)$$

$$NSM(C) = \sum_{sm=0}^m C_{sm} \quad (9)$$

Example 4: Expression 1 of Figure 1 depicts that class person has a static variable named pid which is accessed by a static method named personid(). Hence NSF of class Person is 2. 5) DAM metric: This metric is the ratio of the number of private attributes to the total number of attributes declared in a class. The pre-condition expressions for a method in OCL specification is used to extract this metric. Let a class C contain attributes $a = \{1 \dots n\}$ which includes private attributes $p = \{0 \dots m\}$, then DAM of class C is expressed as

$$DAM(C) = \frac{\sum_{p=0}^m (c_p)}{\sum_{a=1}^n (c_a)} \quad (10)$$

Example 5: Expression 6 of Figure 1 shows that class bloodstock has four attributes namely stkid, date, bloodgroup, Apositive. Hence NOF of bloodstock = 4. Moreover, attribute stkid is defined as a private attribute of class bloodstock. Hence, its DAM of class bloodstock is $\frac{1}{4} = 0.25$.

6) APPM metric: This metric is the ratio between the total number of parameters defined for all methods and the total number of methods defined for a class. This metric value can be extracted from the method definition part for a class in OCL expression. Let a class C contain methods $m = \{1 \dots n\}$ and the total number of methods parameter defined for a class C is $p = \{0 \dots k\}$, then APPM metric value of class C is given by

$$APPM(C) = \frac{\sum_{p=0}^k (c_p)}{\sum_{m=1}^n (c_m)} \quad (11)$$

Example 6: Expression 1 of Figure 1 shows that there are two methods defined for the class Person such as personinfo and personid. Also, it depicts that method personinfo() has a parameter pid. Hence, APPM of class Person is given by $\frac{1}{2} = 0.5$.

7) DOI metric: This metric denotes the level number of the given class in the inheritance tree hierarchy. DOI metric value of a class can be calculated from its method definition and invariant declaration part in the OCL expression. Let a class C contain predecessors $d = \{0 \dots n\}$, then the depth of inheritance tree of class C is given by

$$DOI(C) = \sum_{d=0}^n c_{d1} \quad (12)$$

Example 7: Expressions 1, 2 and 3 of Figure 1 depict that classes such as donor and recipient have inherited an attribute (pid) from class person. Hence their predecessor is person. Thus, their DOI is 1.

8) NOC metric: This metric denotes the total number of immediate successors of a class. Let a class C contain immediate successors $nc = \{0 \dots n\}$, then the total number of children for a class C is given by

$$NOC(C) = \sum_{nc=0}^n c_{nc} \quad (13)$$

Example 8: Expressions 1, 2 and 3 of Figure 1 depict that class person has two immediate successors such as donor and recipient. Hence these classes have inherited the attribute pid of the class person. Thus, its NOC=2.

9) NMA metric: This metric specifies the total number of methods defined in a subclass or subclasses of a given class. Let a class C contain immediate successors $S = \{0 \dots n\}$ and each successor has methods $m = \{1 \dots k\}$, then the total number of methods defined in subclasses of a class C is given by

$$NMA(c) = \sum_{s=0}^n \sum_{m=1}^k s_m(C) \quad (14)$$

Example 9: Expressions 1, 2 and 3 of Figure 1 depict that there are three methods such as donorinfo(), Donate() and recipientinfo() are defined in subclasses of class Person. Hence its NMA=3.

10) NMI metric: This metric specifies the total number of methods inherited by a subclass. This metric is a measure of the complexity of a method. Let a class C contain immediate successors $S = \{0 \dots n\}$ and each successor has inherited methods $mi = \{0 \dots k\}$ which are defined in their parent class C, then the total number of methods inherited by subclasses of class C is given by

$$NMA(c) = \sum_{s=0}^n \sum_{mi=1}^k s_{mi}(C) \quad (15)$$

Example 10: Expressions 1, 2 and 3 of Figure 1 depict that class Person has two immediate successors namely donor and recipient which have inherited a method personid() from the class Person. Hence, NMI of class Person is 1.

11) DCC metric: This metric points out the non-inheritance based relation for a class. A class can

reference another class by accessing its attribute or method or instance. Let us consider a class C reference other classes such as $j = \{0..n\}$, then the total number of direct class coupling measure of class C is given by

$$DCC(C) = \sum_{j=0}^n c_j(C) \quad (16)$$

Example 11: Expressions 8, 11 and 12 of Figure 1 show that class *administrator* is associated with four classes such as *bloodstock*, *bloodrequest*, *donor* and *recipient*. Hence, its $DCC=4$.

12) Fan-In metric: This metric is used to measure the total number of other classes that reference a class. Let us consider a- number components that call class C, b- number of parameters passed to class C from a component higher in the hierarchy, c- number of parameters passed to class C from a component lower in the hierarchy and d- number of elements read by component C. Then

$$\text{Fan-In (C) [20]} = a + b + c + d \quad (17)$$

Example 12: Expression 8 of Figure 1 shows that class *administrator* calls two classes such as *bloodstock* and *bloodrequest*. Hence, its Fan-In value is 2.

13) Fan-Out metric: Fan-out of a class is used to measure the total number of other classes referenced by a class. Let us consider e – number of components called by class C, f – number of parameters passed from class C to a component higher in the hierarchy g- number of parameters passed from a class C to a component lower in the hierarchy and h- number of data elements returned to class C. Then

$$\text{Fan-Out (C) [20]} = e + f + g + h \quad (18)$$

Example 13: Expressions 11 and 12 of Figure 1 show that class *administrator* calls two classes such as *recipient* and *donor*. Hence, its Fan-Out value is 2.

14) INF metric: This metrics is used to measure the data flow in and out of a method through attribute access and parameter passing. INF metric value of a component C can be derived from its Fan-In and Fan-out values as follows.

$$\text{INF (C) [17]} = (\text{Fan-In (C)} * \text{Fan-Out(C)})^2 \quad (19)$$

Example 14: Expressions 8, 11 and 12 of Figure 1 indicates that Fan-In=2, Fan-out=2 for the class *administrator*. Hence, INF of class *administrator* is $= (2*2)^2 = 16$.

15) IM metric: It has been viewed that Effluent Coupling (EC) is equal to Fan out and Afferent Coupling (AC) is equal to Fan In [21]. IM metric value of a component C can be derived from its Fan-In and Fan-out values as follows.

$$\text{IM(C)} = \frac{\text{Fan - out(C)}}{\text{Fan - In(C)} + \text{Fan - out(C)}} \quad (20)$$

Example 15: Expressions 8, 11 and 12 of Figure 1 describe that Fan-In=2, Fan-out=2 for the class *administrator*. Hence, IM of class *administrator* is $= (2/(2+2)) = 1/2=0.5$.

B) Analysis on test metrics

In this section the significance of the test metrics in complexity estimation is analyzed using descriptive data analysis and correlation analysis techniques.

1) Descriptive Data Analysis: The descriptive statistics of the existing design metrics which are extracted from OCL expressions for the five case study applications are shown in Table 2.

From Table 2 the following observations are made.

- 75% of classes of a system have shown zero values for inheritance-related metrics such as NSFM, DAM, NMA, NMI and NOC. It depicts that these metrics are not significant for complexity estimation and are not considered for further analysis.
- Compared to NOC, DOI metric has value slightly above 0 for the case studies namely 3, 4 and 5. Hence, it can be considered for complexity estimation.
- The size-related metrics such as NOM, NOF, NOP, APPM and association-related metrics namely DCC, Fan-In, Fan-Out are significant for complexity estimation.

2) Correlation Analysis: In order to analyze the relation between each existing design metrics with total test metrics values, correlation analysis is performed, the results of which are shown in Table 3.

From Table 3 it has been inferred that, metric DOI is negatively correlated with total metric values. Hence, DOI can be ignored for estimating module complexity. Furthermore, if the correlation coefficient of a metric is below 0.5 it indicates that the particular metric is not related to complexity estimation [22]. For the five case study applications, the correlation co-efficient of metric APPM is below 0.5. Hence, it also will not be considered for complexity estimation. Table 3 shows that, eight metrics such as INF, DCC, Fan-In, Fan-Out, NOP, NOM, IM and NOF are highly significant in complexity estimation.

According to Jubire et al. [23], all design metrics are not equal and each metric has unique dimension. However, according to Binkley and Schach [24], coupling-based metrics are the good predictors of fault-prone components.

BBMS									
Metrics	Descriptive statistics					Percentiles			
	Min	Max	μ	Median	σ	25%	50%	75%	
NOF	1	13	4.84	4	3.79	2	4	6	
NOM	1	6	1.79	1	1.20	0.5	1	1.5	
NOP	0	12	1.89	1	2.51	0.5	1	1.5	
NSFM*	0	2	0.42	0	0.82	0	0	0	
DAM*	0	1	0.11	0	0.31	0	0	0	
APPM	0	2	0.95	1	0.48	0.5	1	1.5	
DOI	0	1	0.42	0	0.49	0	0	0	
NOC*	0	4	0.42	0	1.04	0	0	0	
NMA*	0	4	0.53	0	1.23	0	0	0	
NMI*	0	1	0.16	0	0.36	0	0	0	
DCC	1	11	2.79	2	2.21	1	2	3	
Fan-In	1	6	1.53	1	1.23	0.5	1	1.5	
Fan-Out	0	5	1.26	1	1.29	0.5	1	1.5	
IF	2	22	5.58	4	4.43	2	4	6	
IM	0	0.8	0.39	0.5	0.25	0.25	0.5	0.75	

PMS									
Metrics	Descriptive statistics					Percentiles			
	Min	Max	μ	Median	σ	25%	50%	75%	
NOF	3	21	5.57	12.5	12.75	6.25	12.5	18.75	
NOM	1	5	1.19	3	2.65	1.5	3	4.5	
NOP	3	17	4.67	12	11.35	6	12	18	
NSFM*	0	1	0.49	0	0.4	0	0	0	
DAM*	0	0	0	0	0	0	0	0	
APPM	3	8.5	1.49	4	4.513	2	4	6	
DOI	0	1	0.5	0	0.45	0	0	0	
NOC*	0	4	1.12	0	0.45	0	0	0	
NMA*	0	1	0.36	0	0.15	0	0	0	
NMI*	0	6	1.94	0	0.8	0	0	0	
DCC	1	6	1.43	2.5	3.05	1.25	2.5	3.75	
Fan-In	1	4	0.81	1	1.5	0.5	1	1.5	
Fan-Out	0	4	0.92	1	1.55	0.5	1	1.5	
IF	0	81	21.87	2.5	12.7	1.25	2.5	3.75	
IM	0	1	0.8	1	0.4	0.5	1	1.5	

BBMS									
Metrics	Descriptive statistics					Percentiles			
	Min	Max	μ	Median	σ	25%	50%	75%	
NOF	2	11	5.8	6	2.76	3	6	9	
NOM	1	4	1.47	1	0.81	0.5	1	1.5	
NOP	1	6	1.8	1	1.33	0.5	1	1.5	
NSFM*	0	2	0.27	0	0.68	0	0	0	
DAM*	0	1	0.07	0	0.25	0	0	0	
APPM	0.5	2	1.23	1	0.48	0.5	1	1.5	
DOI	0	1	0.33	0	0.47	0	0	0	
NOC*	0	3	0.33	0	0.87	0	0	0	
NMA*	0	5	0.47	0	1.31	0	0	0	
NMI*	0	1	0.13	0	0.34	0	0	0	
DCC	8	18	2.8	2	1.56	1	2	3	
Fan-In	1	4	1.33	1	0.81	0.3	1	1.3	
Fan-Out	0	4	1.27	1	0.85	0.5	1	1.5	
IF	0	32	4.93	2	7.44	1	2	3	
IM	0	1	0.39	0.5	0.25	0.25	0.5	0.75	

BMS								
Metrics	Descriptive statistics					Percentiles		
	Min	Max	μ	Median	σ	25%	50%	75%
NOF	2	15	6.21	4	3.64	2	4	6
NOM	1	4	1.74	2	0.85	1	2	3
NOP	2	16	4.37	3	3.34	1.5	3	4.5
NSFM*	0	0	0.00	0	0.00	0	0	0
DAM*	0	0	0.00	0	0.00	0	0	0
APPM	1	5	2.43	2	0.97	1	2	3
DOI	0	1	0.63	1	0.48	0.5	1	1.5
NOC*	0	7	1.00	0	1.92	0	0	0
NMA*	0	7	1.00	0	1.92	0	0	0
NMI*	0	1	0.26	0	0.44	0	0	0
DCC	1	8	2.95	2	1.88	1	2	3
Fan-In	1	6	1.47	1	1.14	0.5	1	1.5
Fan-Out	0	6	1.47	1	1.39	0.5	1	1.5
IF	0	24	4.63	2	5.77	1	2	3
IM	0	12	2.70	2	2.70	1	2	3

E-com								
Metrics	Descriptive statistics					Percentiles		
	Min	Max	μ	Median	σ	25%	50%	75%
NOF	0	10	4.56	4	2.46	2.00	4	6
NOM	1	4	1.56	1	0.69	0.50	1	1.5
NOP	0	5	1.68	1	1.25	0.50	1	1.5
NSFM*	0	2	0.85	1	0.69	0.50	1	1.5
DAM*	0	2	0.88	0	0.99	0.00	0	0
APPM	0	2.5	0.65	1	1.07	0.50	1	1.5
DOI	0	2	0.59	1	0.62	0.50	1	1.5
NOC*	0	2	0.62	1	0.6	0.50	1	1.5
NMA*	0	6	1.75	0	0.85	0.00	0	0
NMI*	0	1	0.4	0	0.21	0.00	0	0
DCC	1	18	3.74	3	3.08	1.5	3	4.5
Fan-In	0	8	1.94	1.5	1.45	0.75	1.5	2.25
Fan-Out	0	10	1.79	1	1.80	0.5	1	1.5
IF	0	160	11.12	4	26.82	2	4	6
IM	0	1	0.43	0.5	0.23	0.25	0.5	0.75

TABLE 3

CORRELATION OF EACH TEST METRIC WITH TOTAL TEST METRIC VALUES

Metrics	BBMS	PMS	LMS	BMS	E-com
NOF	0.277	0.657	0.024	0.524	0.083
NOM	0.916	0.467	0.744	0.445	0.59
NOP	0.912	0.795	0.618	0.547	0.524
APPM	0.251	0.362	0.085	0.407	0.316
DOI	-0.401*	-0.328*	-0.232*	-0.443*	-0.055*
DCC	0.960	0.788	0.699	0.879	0.949
Fan-In	0.792	0.525	0.894	0.500	0.875
Fan-Out	0.852	0.569	0.939	0.776	0.920
IF	0.963	0.889	0.980	0.820	0.990
IM	0.364	0.117	0.839	0.685	0.182

* Least significant

Based upon the analysis results a weight is assigned to each test metric in a scale of 1 - 4 to reflect its importance in complexity estimation, which is presented in Table 4.

TABLE 4

RELATIVE WEIGHT OF EACH TEST METRICS

Metrics	Weight
INF	4
DCC	3.8
Fan-Out	3.5
Fan-In	3
NOP	2.5
NOM	2
IM	1.6
NOF	1.3

Thus, the total complexity of a components C based on the test metrics is given by

$$\text{Total complexity (C)} = \text{INF(C)} * 4 + \text{DCC(C)} * 3.8 + \text{Fan-Out(C)} * 3.5 + \text{Fan_In(C)} * 3 + \text{NOP(C)} * 2.5 + \text{NOM(C)} * 2 + \text{IM(C)} * 1.6 + \text{NOF(C)} * 1.3 \quad (21)$$

VII. PROPOSED OCL BASED DESIGN METRICS

A) Proposed metrics

This section presents a new set of proposed metrics which can be extracted directly from OCL expressions based on the invariants, pre-conditions

and post-conditions to measure the complexity of each module of the system. Five such metrics are proposed, and they have been classified into two types, namely measures of class size and measures of dependency between classes.

1) Number of pre-conditions specified for a class (NPRS):

The NPRS metric measures the total number of pre-conditions defined for a class. In general, a pre condition denotes the conditions which must be true before the function is called. In this proposed approach, it has been investigated that pre-conditions can be used as an appropriate measure of attribute defined in a class. NPRS metric is a measure of size of a class. Therefore, higher the NPRS for a class, larger is its complexity. Let a class C contain pre-conditions $pc = \{0..n\}$ then, the total number of pre-conditions specified for a class C is given by

TABLE 5

DESCRIPTIVE STATISTICS FOR THE FIVE CASE STUDY APPLICATIONS USING PROPOSED OCL BASED METRICS

BBMS								
Metrics	Descriptive statistics					Percentiles		
	Min	Max	μ	Median	σ	25%	50%	75%
NPRS	1	13	4.79	5	3.43	2.5	5	7.5
NPOS	0	8	1.53	1	1.85	0.5	1	1.5
NLOS	2	18	8.63	6	5.87	3	6	9
NOPP	0	12	1.42	0	2.74	0	0	0
NOIN	0	8	2.42	1	3.00	0.5	1	1.5

PMS								
Metrics	Descriptive statistics					Percentiles		
	Min	Max	μ	Median	σ	25%	50%	75%
NPRS	1	11	5.09	4.5	3.15	2.25	4.5	6.75
NPOS	0	2	0.91	1	0.42	0.5	1	1.5
NLOS	2	17	7.68	7	4.18	3.5	7	10.5
NOPP	0	4	0.77	0	1.24	0	0	0
NOIN	0	5	1.27	0.5	1.63	0.25	0.5	0.75

LMS								
Metrics	Descriptive statistics					Percentiles		
	Min	Max	μ	Median	σ	25%	50%	75%
NPRS	2	12	5.87	5	2.96	2.5	5	7.5
NPOS	0	3	0.67	0	0.87	0	0	0
NLOS	3	26	10.27	8	6.09	4	8	12
NOPP	0	4	0.73	0	1.12	0	0	0
NOIN	0	7	3.07	2	2.14	1	2	3

BMS								
Metrics	Descriptive statistics					Percentiles		
	Min	Max	μ	Median	σ	25%	50%	75%
NPRS	1	6	3.58	4	1.43	2	4	6
NPOS	0	2	0.95	1	0.60	0.5	1	1.5
NLOS	1	14	6.37	7	2.99	3.5	7	10.5
NOPP	0	3	1.16	1	1.14	0.5	1	1.5
NOIN	0	8	1.84	1	2.13	0.5	1	1.5

E-com								
Metrics	Descriptive statistics					Percentiles		
	Min	Max	μ	Median	σ	25%	50%	75%
NPRS	1	12	4.61	4	2.77	2	4	6
NPOS	0	2	0.39	0	0.60	0	0	0
NLOS	1	14	4.97	5	3.19	2.5	5	7.5
NOPP	0	1	0.15	0	0.36	0	0	0
NOIN	0	5	1.36	1	1.43	0.5	1	1.5

$$\text{NPRS (C)} = \sum_{pc=0}^n c_{pc} \quad (22)$$

Example 1: Expressions 1 and 2 of Figure 1 show that there are seven pre-conditions specified for the class Person. Hence, its NPRS= 7.

2) Number of post-conditions specified for a class (NPOS):

NPOS metric measures the total number of post-conditions specified for a class. In general, a post condition denotes the conditions that must be true after the function has been called. The NPOS metric too is a measure of the size of a class. If NPOS metric

value is high then the complexity of a class will also be high. Let a class C contain post-conditions $po = \{0 \dots n\}$ then, the total number of post-conditions specified for a class C is given by

$$NPOS(C) = \sum_{po=0}^n c_{po} \quad (23)$$

Example 2: Expressions 1 and 2 of Figure 1 show that there is one post-condition specified for class Person. Hence, its NPOS= 1.

3) Number of lines of specification for a class (NLOS): NLOS metric measures the total number of lines used to specify a class. This denotes the total number of specification lines used to define methods of a class which include context, invariants, pre and post-conditions used to define the class. Number of lines of specification is directly related to complexity of a component. Higher the NLOS value, more the complexity of a class is. Let a class C is specified with the number of lines using OCL $ls = \{1 \dots n\}$, then the total number of lines is used to specify a class C is given by

$$NLOS(C) = \sum_{ls=1}^n c_{ls} \quad (24)$$

Example 3: Expressions 6, 7 and 10 of Figure 1 show that there are 14 lines used to specify class Bloodstock hence, its NLOS= 14.

4) Number of objects passed as method parameter for a class (NOPP): NOPP metric measures the total number of class objects passed as a parameter to a method of a class. By passing class as a parameter to the method, all members of a class, including attributes and methods can be inferred. The complexity of a class will be high if more number of class instances are passed as a parameter to the methods of that class. Let a class C contain methods $M = \{1 \dots n\}$ and each method pass class objects as parameter $ps = \{0 \dots k\}$, then the total number of parameters in a class C is given by

$$NOPP(C) = \sum_{M=1}^n \sum_{ps=0}^k M_{ps}(C) \quad (25)$$

Example 4: It is evident from expression 8 that the class administrator receives two object parameters

namely s and r, belonging to bloodstock and bloodrequest classes respectively. Hence, its NOPP=2.

5) Number of invariants specified for a class (NOIN): The NOIN metric defines the total number of invariants specified for a class. Invariant of a class indicates those conditions which are always true for a class. The relationships between classes can be inferred using invariants. Let a class C contain methods $M = \{1 \dots n\}$ and each method is specified using invariants $inv = \{1 \dots k\}$ then, the total number of invariants specified for class C is given by

$$NOIN(C) = \sum_{M=1}^n \sum_{inv=0}^k M_{inv}(C) \quad (26)$$

Example 5: Expressions 1 and 2 of Figure 1 show that there are seven pre-conditions specified for the class Person and hence its NOIN= 7.

TABLE 6
CORRELATION OF THE PROPOSED OCL BASED METRICS WITH TOTAL METRICS VALUES

Metrics	Correlation Analysis				
	BBMS	PMS	LMS	BMS	E-com
NPRS	0.828	0.910	0.605	0.968	0.905
NPOS	0.533	0.584	0.861	0.569	0.613
NLOS	0.987	0.992	0.994	0.998	0.994
NOPP	0.586	0.696	0.558	0.651	0.556
NOIN	0.862	0.846	0.763	0.832	0.743

B) Analysis on the proposed OCL-based metrics

In order to realize the significance of the proposed OCL-based metrics in complexity estimation, descriptive data analysis and correlation analysis are performed.

Table 5 shows the descriptive data of the proposed OCL-based metrics for the five case study applications. From Table 5, the following observations are made.

- NLOS metric is the most significant metric in complexity estimation.
- Among the five proposed metrics, NPOS metric is the least significant.

2) Correlation Analysis: Correlation analysis is performed to analyze the relation between each proposed metric and total metric values, the results of which is presented in Table 6.

From Table 6 it has been inferred that all the five proposed metrics are positively correlated with the total metric values. Based on the experimental results, it is observed that all the five metrics are strong measures of module complexity.

Though NLOS metric is highly significant in complexity estimation dependency-based metrics [24] such as NOPP, NOIN are given higher weightage while measuring module complexity.

Thus, a relative weightage for each of the proposed metrics in a scale f 1 to 4 is assigned as shown in Table 7. Using these weightage the total complexity of a component C is given by

$$\text{Total complexity (Ci)} = \text{NPRS(Ci)} * 2 + \text{NPOS(Ci)} * 1 + \text{NLOS(Ci)} * 2.5 + \text{NOPP(Ci)} * 4 + \text{NOIN(Ci)} * 3 \quad (27)$$

TABLE 7
RELATIVE WEIGHT OF THE PROPOSED OCL BASED METRICS

Metrics	Weight
NPRS	2
NPOS	1
NLOS	2.5
NOPP	3
NOIN	4

VIII. PERFORMANCE EVALUATION: PROPOSED METRICS VS TEST METRICS

Formal specifications represent blue print of source of the software system. Hence, metrics extracted from OCL specification provides more accurate results for

complexity estimation. Furthermore, there is positive correlation between complexity of components and its fault-proneness [6], [25]. Thus, complexity estimation based on OCL specification enables prediction of fault-prone modules during design time.

This section compares the significance of the test metrics with the proposed OCL-based metrics in predicting fault-prone components during design stage of software development. Table 8 shows the total complexity value of each component of the five case study applications measured using both test metrics and the proposed OCL based metrics.

TABLE 8
TOTAL COMPLEXITY ESTIMATION AND FAULT-PRONE COMPONENTS PREDICTION FOR FIVE CASE STUDY APPLICATIONS USING THE TEST METRICS VS PROPOSED OCL BASED METRICS

BBMS			PMS			LMS			BMS			E-com		
Components	Existing Metrics	Proposed Metrics	Components	Existing Metrics	Proposed Metrics	Components	Existing Metrics	Proposed Metrics	Components	Existing Metrics	Proposed Metrics	Components	Existing Metrics	Proposed Metrics
Admin*	246.07#	132	In_Patient	37.40	21.5	Admin*	191.1#	97.5@	Account	114.4#	37.5	Admin*	208.73#	61@
Blood stock*	57.43	119	Out-Patient	31.40	14	Membership*	201.8#	86@	Admin*	110.5#	106.5@	User	21.97	39
Person	34.70	43	Operation	58.40	36.5	Staff	45.22	25.5	Check Balance*	182.4#	61.5@	Customer	31.97	9
Transaction	48.77	57	Medicine	38.90	36	Student	33.4	29	Cheque Book	58.4	60	Supplier	31.97	11
Blood issue	35.10	59	Injection	37.60	25.5	Books	27.5	44	Current A/c	40.5	40	Bank*	100.55#	58@
Blood request	38.80	38	Diet advice	16.90	8.5	Digital library	23.9	36	Customer	50.36	26	Order*	110.05#	92.5@
Report	109.16#	21.5	Diagnosis report	40.20	2.5	Book Purchase	42.5	62@	Deposit*	64.3	105.5@	Purchase Order	50.16	21.5
Empdetail	41.60	91#	Lab report	24.50	2.5	Payment	130.82#	33.5	Fixed A/c	22.6	37.5	Transaction	115.70#	47
Camp	46.50	22.5	Patient report	65.12	2.5	Transaction*	72.2#	50@	Fund Transfer	71.5	68@	Purchase	60.70	47
Blood collection	83.50#	91.5#	Billing*	36.10	63.5@	Issue book*	53.67#	55@	Month	36.3	18.5	Sales*	93.87#	74@
Donor*	50.47	98.5#	Nurse	69.97	15	Return book	59.22#	63.5@	Online Services	37.9	18	Purchase return	48.61	41
Equipment	41.60	12.5	Lab technician	26.60	15	Renew book	46.1	47	Profile Updation	71.3	34	Sales return	48.61	38
Recipient	36.20	41.5	Doctor	100.07#	36.5	Collect fine	50.02#	37	Recurring A/c	22.6	27.5	Stock*	129.88#	64@
Salary	34.90	22.5	Appointment*	107.52#	54@	Search	20	13.5	Saving A/c	22.6	27.5	Shipment	104.44#	43
Attendance	26.20	22.5	Diagnosis*	198.18#	72.5@				Statements	29.4	21.5	Confirmshipment	86.78#	35.5
Donor report	13.60	8	Patient	136.66#	26				Transaction	71.2	29.5	Product	42.98	35
Camp report	13.60	8	Registration*	92.50#	59@				User	106.3#	31	Payment*	77.58	57.5@
Recipient report	13.60	8	Report	61.00	11				Withdraw*	102.4#	67@	Invoice	100.44#	31
Stock report	13.60	8	Staff	61.60	37.5				Year	35.3	12.5	Receipt	123.30#	31
			Test	81.65#	34.5							Report	110.70#	22.5
			Treatment*	39.80	62.2@							Purchase report	20.40	9
			Ward	67.60	33.5							Sales report	20.40	9
												Purchaseorderreport	24.40	9
												Stock report	18.40	9
												Salesorde rreport	24.40	9
												Department	47.40	11
												Purchase dept	37.77	11
												Sales dept	57.00	40
												Guest	40.30	18
												Person	47.80	13.5
												Reorder level	31.97	15.5
												Employee	42.98	57@

* Actual fault-prone Component

Component predicted as fault-prone using the test metrics

@ Component predicted as fault-prone using the proposed OCL based metrics

Based on the data collected from Table 8 a performance evaluation has been carried out and its results are presented in Table 9.

TABLE 9
PERFORMANCE EVALUATION RESULTS

Test Metrics					
Performance Measure	BBMS	PMS	LMS	BMS	E-com
Sensitivity	0.33	0.80	1.00	0.75	0.83
Specificity	0.93	0.77	0.63	0.93	0.80
Precision	0.89	0.77	0.73	0.89	0.81
Proposed OCL based Metrics					
Performance Measure	BBMS	PMS	LMS	BMS	E-com
Sensitivity	1.00	1.00	1.00	0.89.00	1.00
Specificity	0.93	1.00	0.72	0.93	0.96
Precision	1.00	1.00	0.80	0.94	0.96

Furthermore, It has been already proved by many researchers [4] [5] [6] that source code metrics are used to measure the complexity of a component.

However, corrective measures at the early stages of software development are more cost effective than at the later stages of software development [26]. From Table 9 it is observed that metrics obtained from OCL specification are highly significant in complexity estimation and fault-prone components prediction than the test metrics. Thus, it is concluded that OCL formal specification-based metrics are directly related to complexity estimation and they support more accurate prediction of fault-prone components.

IX. CONCLUSIONS

The proposed study has shown that there are eight object-oriented design metrics extracted from OCL specification that are significant in complexity estimation. Moreover, as a novel approaches this work has proposed the five OCL based metrics as a

measure of complexity and fault-proneness. Statistical analysis has been performed on the five case study applications both on the existing design metrics and the proposed OCL based metrics. From the experimental results it is evident that, the OCL based metrics are highly significant in complexity estimation. The proposed study showed that the fault-prone components of the system can be predicted based on the analysis on its OCL formal specification. This enables prioritization of components in inspection, defect detection, debugging, coding and testing activities to improve software quality.

As a future work more OCL formal specification based metrics can be analysed with other statistical and machine learning approaches to reveal the contribution of OCL based metrics in complexity estimation which would predict the fault-prone components of a software system.

ACKNOWLEDGMENT

The proposed paper is a part of the UGC major research project supported by University Grants Commission (UGC), New Delhi, India

REFERENCES

- [1] Ray. M and Mohapatra D.P., "A Novel Methodology for Software Risk Assessment at Architectural Level using UML Diagrams", SETLabs Briefings, vol. 9, no. 4, pp. 41-60, 2012.
- [2] Goseva-Popstojanova Katerina, Hassan Ahmed, Guedem Ajith, Abdelmoez Walid, M.Nassar Diaa Eldin, Ammar Hany, Mili Ali, "Architectural Level Risk Analysis", IEEE Transactions on Software Engineering, vol. 29, no.10, pp. 946 – 960, 2003.
- [3] Cortelessa Vittorio, Goseva-Popstojanova Katerina, Appukkutty Kalaivani, R.Guedem Ajith, Hassan Ahamed, Elnaggar Rania, Abdelmoez Walid, H.Ammar Hany, "Model Based Performance Risk Analysis", IEEE Transactions on Software Engineering, vol. 31, no.1, pp.3 – 20, 2005.
- [4] Emam.A, Khaled El, Melo B Walcelio, C. Machado Javam, "The Prediction of Faulty Classes using Object-Oriented Design Metrics", Journal of Systems and Software, vol. 56, no.1, pp.63-75, 2001.
- [5] Osamu Mizuno and Hideaki Hata , "Prediction of Fault-prone Modules using a Text Filtering Based Metric", International Journal of Software Engineering and Its Applications , vol. 4, no. 1, 2010.
- [6] Liguu Yu and Alok Mishra, "Experience in Predicting Fault-Prone Software Modules using Complexity Metrics", Quality Technology & Quantitative Management, vol. 9, no. 4, pp. 421-433, 2012.
- [7] Jiang Yue , Cukic Bojan, Menzies Tim, "Fault Prediction using Early Lifecycle Data", Proceedings of the 18th IEEE International Symposium on Software Reliability , IEEE Computer Society, Washington, DC, USA , pp.237-246, 2007.
- [8] M.H. Tang, M.H. Kao, and M.H. Chen , "An Empirical Study on Object-Oriented Metrics," In Proceedings Sixth International Symposium on Software Metrics, pp. 242-249, 1999.
- [9] Erik Arisholm, Lionel C. Briand, Eivind B. Johannessen, "A Systematic and Comprehensive Investigation of Methods to Build and Evaluate Fault Prediction Models", Journal of Systems and Software, vol.83, pp. 2–17, 2010.
- [10] Malhotra Ruchika, Jain Ankita, "Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality", Journal of Information Processing Systems, vol.8, no.2, pp.241-262, 2012.
- [11] Ray.M and Mohapatra D.P., "Prioritizing Program Elements: A Pretesting Effort to Improve Software Quality", ISRN Software Engineering, vol. 2012, Article ID 598150, doi:10.5402/2012/598150
- [12] M. Olague Hector, H. Etzkorn Letha, Gholston Sampson, Quattlebaum Stephen, "Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed using Highly Iterative or Agile Software Development Processes", IEEE Transactions on Software Engineering, vol. 33, no. 6, pp. 402 – 419. 2007.
- [13] Shatnawi A Raed, Li Wei, "The Effectiveness of Software Metrics in Identifying Error-Prone Classes in Post-Release Software Evolution Process", Journal of Systems and Software, vol. 81, pp.1868–1882, 2008.
- [14] S.R.Chidamber and C.F.Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol.20, no.1, pp. 476 – 493, 1994.
- [15] Bansiya, J, Davis, C, "Automated Metrics and Object-Oriented Development", Dr. Dobb's Journal, vol. 272, pp. 42-48, 1997.
- [16] Zimmerman, T; Nagappan, N; Herzig, K; Premraj, R; L. Williams, "An Empirical Study of the Factors Relating Field Failures and Dependencies", IEEE Fourth International Conference on Software Testing, Verification and Validation, 2011, pp. 347-356
- [17] Hendry, Sallie, Kafura Dennis, "Software Structure Metrics Based on Information Flow", IEEE Transactions on Software Engineering, vol. 7, no. 5, pp. 510-518, 1981.
- [18] Lorenz M. and Kidd J, "Object-Oriented Software Metrics a Practical Guide", Prentice Hall, Englewood Cliffs, 1994. New Jersey.
- [19] R. Martin, "OO Design Quality Metrics - An Analysis of Dependencies", In Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94. 1994.
- [20] Goodman P, "Software Metrics: Best Practices for Successful IT Management", Rothstein Associates Inc, 2004.
- [21] Online:<http://www.virtualmachinery.com/jhawkmetricsclass.htm>.
- [22] K. K. Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra, "Empirical Study of Object-Oriented Metrics", Journal of Object Technology, vol.5, no.8, pp. 149-173, 2006.
- [23] Jubair, J, Al-Ja'afar, Khan, Eddin, Sabri, M, "Chidember Kemerer (CK) and Lorenze-Kidd(LK) Metrics to Assess Java Programs", King AbdullahII School for Information Technology, University of Jordan, Jordan.
- [24] Binkley, A.B. and Schach, S.R, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures", Proceedings of the 20th International Conference on Software Engineering, pp. 452–455, 1998.

- [25] W. Wong Eric, "Identify Fault-Prone Software Modules in Telecommunications Systems", Motorola 2006 System, Software and Simulation Symposium, Chicago, Illinois, 2006
- [26] Frank Elberzhager, Alla Rosbach, Jürgen Münch, Robert Eschbach, "Reducing test effort: A Systematic Mapping Study on Existing Approaches", Journal of Information & Software Technology, vol. 54, no.10, pp. 1092-1106, 2012.