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*, mejalila@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**

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

This Article is brought to you for free and open access by 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

¹,²,³,⁴Department of Computer Applications, Thiagarajar College of Engineering, Madurai, India
E-mail: ¹mejalila@gmail.com, ²djmce@tce.edu, ³balamsg4u@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 at from 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.

1. 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.

Olaugé 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. \n
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>
<thead>
<tr>
<th>TABLE 1 BRIEF SUMMARY OF CASE STUDY APPLICATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>No of components</td>
</tr>
<tr>
<td>No of faulty components</td>
</tr>
<tr>
<td>Fault-prone Components</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

International Journal of Computer Science and Informatics, ISSN (PRINT): 2231 –5292, Volume.4, Issue.2
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:

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, v2 : T2) : RT
pre : PreC
post : PostC

where, \(V = \{v_1, v_2\}\) are the formal parameters of the method
\(M, T = \{T_1, T_2\}\) 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,

\[
\text{context C inv invName : Ct or context C inv invName : self.De Di } \rightarrow \text{ Ct}
\]

Where, De is the associated class name, Di 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

\[
\text{NOM}(C) = \sum_{m=1}^{n} C_m
\]

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 \(f = \{1...m\}\), then the total number of attributes in a class C is expressed by

\[
\text{NOF}(C) = \sum_{f=1}^{m} C_f
\]

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

\[
\text{NOP}(C) = \sum_{m=1}^{n} \sum_{p=1}^{k} M_p(C)
\]

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 srid 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 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

\[
\text{NSFM}(C) = \text{NSF}(C) + \text{NSM}(C)
\]

\[
\text{NSF}(C) = \sum_{m=1}^{n} C_{sf}
\]

\[
\text{NSM}(C) = \sum_{m=1}^{n} C_{sm}
\]
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 NSFM 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 \( a = \{1…n\} \) which includes private attributes \( p = \{0…m\} \), then DAM of class C is expressed as

\[
\text{DAM}(C) = \frac{\sum_{p=0}^{m} (C_p)}{\sum_{q=1}^{n} (C_q)}
\]

(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 \( \frac{4}{4} = 0.25 \).

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 \( m = \{1…n\} \) and the total number of methods parameter defined for a class C is \( p = \{0…k\} \), then APPM metric value of class C is given by

\[
\text{APPM}(C) = \frac{\sum_{p=0}^{k} (C_p)}{\sum_{m=1}^{n} (C_m)}
\]

(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 \( d = \{0…n\} \), then the depth of inheritance tree of class C is given by

\[
\text{DOI}(C) = \sum_{d=0}^{n} C_{d1}
\]

(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 \( n_c = \{0…n\} \), then the total number of children for a class C is given by

\[
\text{NOC}(C) = \sum_{n_c=0}^{n} C_{n_c}
\]

(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 \( S = \{0…n\} \) and each successor has methods \( m = \{1…k\} \), then the total number of methods defined in subclasses of a class C is given by

\[
\text{NMA}(C) = \sum_{m=1}^{n} \sum_{k=1}^{m} S_{m}(C)
\]

(14)

Example 9: Expressions 1, 2 and 3 of Figure 1 depict that there are three methods such as donorinfo(), Donate() and recepientinfo() 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 \( S = \{0…n\} \) and each successor has inherited methods \( mi = \{0…k\} \) which are defined in their parent class C, then the total number of methods inherited by subclasses of class C is given by

\[
\text{NMI}(C) = \sum_{m=1}^{n} \sum_{k=1}^{m} S_{m}(C)
\]

(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 \( j = \{0 \ldots n\} \), then the total number of direct class coupling measure of class C is given by

\[
DCC(C) = \sum_{j=0}^{n} C_j(C)
\]  

(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

\[
Fan-In(C) [20] = a + b + c + d
\]  

(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 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

\[
Fan-Out(C) [20] = e + f + g + h
\]  

(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.

\[
INF(C) [17] = (Fan-In(C))^*Fan-Out(C))^2
\]  

(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 Efferent Coupling (EC) is equal to Fan out and Affluent 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.

\[
IM(C) = \frac{Fan-Out(C)}{Fan-In(C) + Fan-Out(C)}
\]  

(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 text 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, APFM 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 APFM 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.
OCL Formal Specification Based Metrics As A Measure of Complexity And Fault-Proneness

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 3**

<table>
<thead>
<tr>
<th>Metrics</th>
<th>ARM</th>
<th>PMS</th>
<th>LMS</th>
<th>RMS</th>
<th>From</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF</td>
<td>0.277</td>
<td>0.867</td>
<td>0.824</td>
<td>0.524</td>
<td>0.018</td>
</tr>
<tr>
<td>NOF</td>
<td>0.946</td>
<td>0.467</td>
<td>0.744</td>
<td>0.485</td>
<td>0.058</td>
</tr>
<tr>
<td>NPOS</td>
<td>0.382</td>
<td>0.540</td>
<td>0.618</td>
<td>0.524</td>
<td>0.108</td>
</tr>
<tr>
<td>NPRS</td>
<td>0.251</td>
<td>0.162</td>
<td>0.085</td>
<td>0.407</td>
<td>0.356</td>
</tr>
<tr>
<td>DCC</td>
<td>-0.400*</td>
<td>-0.378*</td>
<td>-0.272*</td>
<td>-0.443*</td>
<td>-0.055*</td>
</tr>
<tr>
<td>Fan-In</td>
<td>0.906</td>
<td>0.781</td>
<td>0.659</td>
<td>0.979</td>
<td>0.340</td>
</tr>
<tr>
<td>Fan-Out</td>
<td>0.820</td>
<td>0.117</td>
<td>0.819</td>
<td>0.685</td>
<td>0.182</td>
</tr>
<tr>
<td>IM</td>
<td>0.304</td>
<td>0.117</td>
<td>0.819</td>
<td>0.685</td>
<td>0.182</td>
</tr>
</tbody>
</table>

* Least significant

**Table 4**

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>INF</td>
<td>4</td>
</tr>
<tr>
<td>DCC</td>
<td>3.8</td>
</tr>
<tr>
<td>Fan-Out</td>
<td>3.5</td>
</tr>
<tr>
<td>Fan-In</td>
<td>3</td>
</tr>
<tr>
<td>NOP</td>
<td>2.5</td>
</tr>
<tr>
<td>NOF</td>
<td>2</td>
</tr>
<tr>
<td>IM</td>
<td>1.6</td>
</tr>
<tr>
<td>NPOS</td>
<td>1.3</td>
</tr>
</tbody>
</table>

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

\[
\text{Total complexity (C)} = (INF(C)^* + DCC(C)^*3.8 + Fan-Out(C)^*3.5 + Fan-In(C)^*3 + NOP(C)^*2.5 + NOF(C)^*2 + IM(C)^*1.6 + NPOS(C)^*1.3)
\]

**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 \(p = \{0, ..., n\}\), then, the total number of pre-conditions specified for a class C is given by

\[
\text{NPRS} = \sum_{p \in C} p
\]

Example 1: Expressions 1 and 2 of Figure 1 show that there are seven pre-conditions 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 \( p_{0}\) \( \{0...n\} \) then, the total number of post-conditions specified for a class C is given by

\[
\text{NPOS}(C) = \sum_{p_{0}=0}^{n} C_{p_{0}}
\]  \hspace{1cm} (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 \( l_{s} = \{1...n\} \), then the total number of lines is used to specify a class C is given by

\[
\text{NLOS}(C) = \sum_{l_{s}=1}^{n} C_{l_{s}}
\]  \hspace{1cm} (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 \( M = \{1...n\} \) and each method pass class objects as parameter \( ps = \{0...k\} \), then the total number of parameters in a class C is given by

\[
\text{NOPP}(C) = \sum_{m=1}^{n} \sum_{ps_{m}=0}^{k} M_{ps_{m}}(C)
\]  \hspace{1cm} (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 \( M = \{1...n\} \) and each method is specified using invariants \( inv = \{1...k\} \) then, the total number of invariants specified for class C is given by

\[
\text{NOIN}(C) = \sum_{m=1}^{n} \sum_{inv_{m}=0}^{k} M_{inv}(C)
\]  \hspace{1cm} (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.
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>
<thead>
<tr>
<th>TABLE 8</th>
<th>TOTAL COMPLEXITY ESTIMATION AND FAULT-PRONE COMPONENTS PREDICTION FOR FIVE CASE STUDY APPLICATIONS USING THE TEST METRICS VS PROPOSED OCL BASED METRICS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BDMS</strong></td>
<td><strong>PMS</strong></td>
</tr>
<tr>
<td>AdmEx*</td>
<td>246.07 0</td>
</tr>
<tr>
<td>Blood stock*</td>
<td>57.5</td>
</tr>
<tr>
<td>Person</td>
<td>74.50</td>
</tr>
<tr>
<td>Transaction</td>
<td>48.77</td>
</tr>
<tr>
<td>Blood issue</td>
<td>35.10</td>
</tr>
<tr>
<td>Blood request</td>
<td>38.80</td>
</tr>
<tr>
<td>Report</td>
<td>109.16</td>
</tr>
<tr>
<td>Equipment</td>
<td>41.40</td>
</tr>
<tr>
<td>Comp</td>
<td>48.30</td>
</tr>
<tr>
<td>Blood collection</td>
<td>63.16</td>
</tr>
<tr>
<td>Drug*</td>
<td>50.47</td>
</tr>
<tr>
<td>Equipment</td>
<td>41.40</td>
</tr>
<tr>
<td>Recipient</td>
<td>34.20</td>
</tr>
<tr>
<td>Salary</td>
<td>34.00</td>
</tr>
<tr>
<td>Attendance</td>
<td>20.20</td>
</tr>
<tr>
<td>Decision report</td>
<td>13.60</td>
</tr>
<tr>
<td>Kum report</td>
<td>13.60</td>
</tr>
<tr>
<td>Request report</td>
<td>13.60</td>
</tr>
<tr>
<td>Stock report</td>
<td>13.40</td>
</tr>
</tbody>
</table>

* 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>
<thead>
<tr>
<th>TABLE 9</th>
<th>PERFORMANCE EVALUATION RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance Measure</strong></td>
<td><strong>BDMS</strong></td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.53</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.03</td>
</tr>
<tr>
<td>Precision</td>
<td>0.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance Measure</th>
<th><strong>BDMS</strong></th>
<th><strong>PMS</strong></th>
<th><strong>LIMS</strong></th>
<th><strong>BDMS</strong></th>
<th><strong>PMS</strong></th>
<th><strong>LIMS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>0.69</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Specificity</td>
<td>0.00</td>
<td>1.00</td>
<td>0.72</td>
<td>0.72</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>Precision</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>0.72</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

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
