# Metrics-based Analysis of Requirements for Object-Oriented systems: An empirical approach

ANANYA KANJILAL[1]
GOUTAM KANJILAL[2]
SWAPAN BHATTACHARYA[3]

[1]Dept. of Information Technology, B. P. Poddar Institute of Management and Technology,
137, V.I.P Road, Kolkata - 700052, India
ag_k@rediffmail.com
[2]Cognizant Technology Solutions, Plot GN-34/3, Sector V,
Salt Lake Electronic Complex, Kolkata - 700091, India
goutam.kanjilal@cognizant.com
[3]National Institute of Technology, Durgapur - 713209, India
bswapan2000@yahoo.co.in

**Abstract.** In an object-oriented environment, it is necessary to ensure that all the requirements are addressed in the design as well as implemented in a consistent manner in UML diagrams like sequence and class diagrams, which model the behavioral and structural aspects of the system. Metrics, which measures the degree of coverage of requirements and the extent of consistency between related models, will be a powerful tool for developers to have a quantitative feedback about the correctness of a system. We have proposed a new set of metrics namely Requirement Coverage Metrics (RCM) and Design Compliance Metrics (DCM) based on UML design models. RCM indicates the extent of coverage of requirements in design and highlights any missing requirements or inadequate coverage in design. It also helps in measuring progress of a project and thus helps in project management. DCM verifies whether the requirements that have been covered in design have been consistently realized in sequence and class diagrams or not. A case study has been considered and calculation of RCM and DCM has been done for illustration of our approach. We have also discussed implementation methodology using an XML based approach and in this paper we present implementation of a part of the metrics suite (DCM) for substantiation of our approach.

**Keywords:** Metrics based analysis, Requirement metrics, Requirement analysis, Design Compliance, Consistency, Requirement coverage.

(Received February 01, 2008 / Accepted March 26, 2008)

## 1 Introduction

In an object-oriented environment, requirements are modeled as use cases and they are implemented as methods of various classes defined in the class diagram and used in the behavioral diagrams. It is necessary to ensure that each and every requirement is addressed as use case and every event of the use cases are implemented as methods of classes and used in behavioral diagrams in a consistent manner. In this paper, we have proposed a metrics based methodology to ensure requirement coverage and consistency of its implementation in design. Metrics act as indicators that provide a quantitative feedback to software developers about various aspects of the software and pinpoint problem areas in their systems. We have proposed a new set of metrics named Requirement Coverage metrics (RCM) and Design Com-

pliance metrics (DCM) and presented methods to derive the metrics from a given set of requirements and UML design models - use case, sequence and class diagrams. We have presented analysis based on them to provide a quantitative feedback regarding coverage of requirements and consistency of its implementation such that developers can take steps before coding starts. Since changes are less expensive the earlier in the development lifecycle they are made, this can save the project considerable time and money. An XML based prototype has been developed that implements this approach. Observations for one of the metrics- DCM have been presented to illustrate our work.

## 2 RELATED WORK

This section presents a review of some of the research works that have been done in the area of coverage of requirements and consistency verification of UML designs. Kim et al. in [11] proposes a set of metrics applicable for UML models. They have defined a large set of metrics separately for model, classes, messages, use case, etc and made a comparison with the more commonly used CK metrics [4]. The metrics suite has been developed on the elements used in the UML models and can be use to predict various characteristics of a project during early phases of software development. Some works as in [19], [2] has developed metrics to ensure coverage of requirements. In [19] high-level requirements expressed formally have been used to define structural coverage metrics as well generate requirement based test cases that can be directly traceable to requirements. In [2], a specification based coverage metrics has been defined to evaluate test sets. They focus on test coverage, however we focus on coverage of requirements in design models. In [9] a metrics suite is defined to measure the quality of design like dynamic complexity and object coupling based on measures from UML architectural specification diagrams. Several works have proposed methodologies for verification of consistency within the UML models. Some like [13], [8], [21], [10], [12], [16], [14], [7], [17], [6], [20], [18] have used formal techniques for verification. Formal techniques range from Object-Z in [12], algebra in [16], attributed graph grammars in [18] focusing mainly on class diagrams and behavioral diagrams. An algorithmic approach to a consistency check between UML Sequence and State diagrams is described in [3] while [14] proposes a declarative approach using process algebra CSP for consistency checking between sequence and statecharts. In [5] an approach for automated consistency checking named VIEWINTEGRA has been developed and in [15] strategies to ensure consistency in object-oriented models has been developed by integrating elements in UML Tool Object Technology Workbench. Our work is closely related to some these works as in [10], [20] and [18] as far as domain of work i.e. class and sequence diagrams are concerned. However, most of these works focus on verifying consistency whereas our work focuses on quantitative analysis and measurement of design to indicate the degree of consistency between these two diagrams apart from determining extent of requirement coverage. It differs from [11] in the sense that here they have defined metrics separately for each UML artifact like message, use case, etc whereas we have defined metrics that consider related UML models from the perspective of requirement analysis. Our metrics will be able to measure the extent of requirement implementation in design and also the degree of consistency within the design.

## 3 SCOPE OF WORK

In this paper we have proposed a new set of metrics based on requirements and UML design models for an object oriented system which will help in measuring the degree of coverage of requirements in design and the degree of compliance and consistency of design models with respect to requirements. In an object-oriented system, use case diagrams of UML form the basis of requirements and Class and Sequence diagrams model the implementation of use cases (requirements) in the design showing the static and dynamic aspects respectively. We have proposed RCM and DCM, which will address two important issues -

1) Measuring coverage of requirements and ensuring that all use cases are at least implemented in a sequence diagram i.e. requirements are captured in the design

2) Measuring the extent of consistency between the Class and Sequence diagrams that will ensure that the requirements have been consistently implemented in design and design is compliant with the given set of requirements.

We have used XML as a standard for expressing the UML use case, class and sequence diagrams in a structured manner based on the XMI standard so that the metrics can be automatically calculated. We have considered a library system as our example and our approach has been applied to this case study and metrics have been calculated. A prototype has been described that shows implementation of one of the metrics.

## 4 UML DIAGRAM RELATIONSHIPS

The UML model consists of several diagrams that depict overlapping aspects of an object-oriented system.

In our work we have considered Use case, Class and Sequence diagrams that show the requirements and their implementations within the design. We have used E-

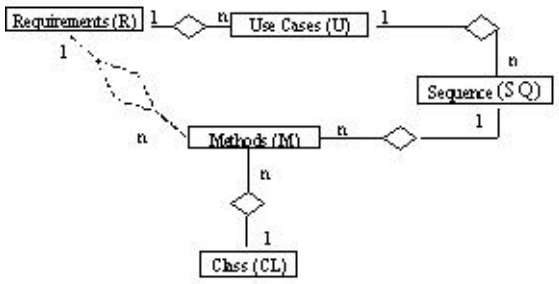


**Figure 1:** Requirement to UML model mapping: UML Relationships

R representation in Fig 1 to show these relationships between Requirements, Use cases, Sequence, Methods and Classes.
The relationship between Use case, Sequence and Class diagrams are based on the existence of common elements between the diagrams. Thus every requirement is eventually implemented through a set of methods defined in Classes (as shown by dotted line). This forms the basis of definition of RCM & DCM discussed in next section.

## 5 PROPOSED WORK

In this section we define metrics RCM and DCM, which will be useful in requirement management as well as project management of object-oriented software projects. We have formulated the metrics based on the relationships that has been identified in earlier section. Metrics are measurements based on project parameters that serve to give a quantitative measurement of various aspects about the system and can be effectively used to control and manage projects and processes.
**Definition: Requirement Coverage**
A Requirement is fulfilled or realized through a number of methods defined in classes (as indicated by the dotted line in Fig 1). We define coverage as the entire path of a requirement from Requirement document to use case and through sequence diagram to class methods. If the path is incomplete at any point, coverage is not 100 per cent. The Requirement Coverage metrics (RCM) is thus a measure of the extent of traceability of requirements.
**Definition: Design Compliance**
A design is compliant with requirements if the behavioral model (here sequence diagram) uses methods which are identically defined (i.e. signatures are same) in the structural model (here Class diagram) for realization of a set of use cases modeling a requirement. The Design Compliance Metrics (DCM) is thus a measure of the extent of consistency between sequence and class diagrams for a use case.

### 5.1 Notations used

In this section the set of three metrics is defined which will be useful in requirement management as well as project management of object-oriented software projects. The following notations are used during metrics definitions:
R - Set of Requirements
U - Set of Use cases
UC - Set of Use case diagrams
C - Set of Classes
CL - Set of Class diagrams
SQ - Set of Sequence diagrams
M - Set of Methods (Methods include name and parameter)
N(S) - Cardinality or Size of a set i.e. number of elements in the set S.

### 5.2 Requirement Coverage Metrics (RCM)

$U_R$: The set of unique use cases defined in use case diagram corresponding to a particular requirement
$U_R = \{u_i \mid u_i \in U, U \in UC, u_i$ implements $r_i, r_i \in R\}$
If this set is empty, it indicates that requirements have not been captured in the use case diagrams of UML.
$S_U$: The set of sequence diagrams used to implement a particular use case $u_i$ in $U_{UC}$
$S_U = \{sq_i \mid sq_i \in SQ, sq_i$ implements $u_i, u_i \epsilon U_R\}$
If this set is empty it means that the use case U has not been implemented in any sequence diagrams.
$U$:The set of implemented use cases i.e. those use cases that have at least one corresponding sequence diagram for implementation (i.e. for which $S_U \neq \phi$)
$M_S$: The set of methods used in all the sequence diagrams for all use cases for a particular requirement.
$M_S = \{m_i \mid m_i \in sq_i, sq_i \in S_U\}$
If this set is empty it means that the sequence diagram S has not used any methods.
$M_C$: The set of methods defined in a class diagram
$M_C = \{m_j \mid m_j \in M, m_i \in c_i, c_i \in C, C \in CL\}$
$M_{S-C}$: The set of implemented methods i.e. methods that are used in a sequence diagram as well as defined in any class of class diagram.
$M_{S-C} = \{m_i \mid m_i \in M_S$ and $m_i \in M_C\}$
The coverage of a requirement i.e. the path of its definition in requirement document till its implementation as

methods in Class diagram has several parts. Thus RCM consists of

**1) Requirements-Use Case coverage (RUC)**

This factor indicates whether a requirement has been mapped to use cases or not. It can be either 0 or 1. If there is at least one use case for a requirement, RUC is 1 otherwise 0.

This measures trace of requirements into use cases.

**2) Use Case-Sequence coverage (USC)**

The ratio of number of use cases implemented in sequence diagrams to the total number of use cases for a particular requirement

$USC = N(U)/N(U_R)$

This measures trace of use cases into sequence diagrams.

**3) Sequence-Class coverage (SCC)**

The ratio of number of implemented methods (i.e. methods used in sequence as well as defined in class) to total number of methods used in a sequence diagram

$SCC = N(M_{S-C})/N(M_S)$

This measures trace of sequence methods into class diagrams.

**Significance of RUC, USC and SCC**

The coverage factors assume values within 0 and 1. A value of 1 indicates 100 per cent implementation and 0 indicates no implementation.

If RUC is 0, it is understood that USC and SCC will be zero. This signifies that if no requirement has been implemented as use cases (RUC = 0), naturally, use case-sequence coverage and successively sequence-class coverage will have no meaning.

Likewise, if USC is 0, it is understandable that SCC will also be zero. This signifies that if no use case has been implemented as sequence diagrams, naturally, sequence-class coverage will have no meaning.

**RCF (Requirement Coverage Factor)**

This defines requirement coverage factor i.e. the extent of coverage of a requirement. We define RCF as

$$RCF = \frac{RUC + USC + SCC}{3}$$

This depends upon the coverage of requirement in each of the successive phases of its implementation in use case, sequence and class. We give equal importance to all the trace paths (Requirements-Use case, Use Case-Sequence, Sequence-Class) and hence it is defined as an average of all the coverage values in each of the phases. Thus $0 \geq RCF \leq 1$

**Significance of RCF**

The value of RCF varies from 0 to 1.

A value of 0 indicates that all the coverage factors RUC, USC and SCC are 0 i.e. requirements have not been captured in the design (as use cases) at all. Since if RUC=0, USC and SCC are 0

A value of 1 indicates that RUC, USC and SCC are all 1. It means that all requirements have been implemented as use cases and all the use cases have sequence diagram implementations and all the methods used in the sequence diagrams are defined in classes.

If RCF is neither 0 nor 1 that means there is incomplete coverage. There may be three cases -

1) If SCC = 0, requirements can be traced to sequence diagram methods but there is no corresponding method definition in class for all the methods used.

2) If USC = 0, requirements can be traced to use cases only but there is no corresponding sequence diagram implementation for all the use cases.

3) If neither SCC nor USC is zero, then it indicates the overall coverage of requirements.

**Requirement Coverage Metrics**

The RCM for a system is defined as the average of all the RCF values for all the requirements taken together.

$$RCM = \frac{\text{Sum Total of RCF for all requirements}}{\text{Total Number of requirements}}$$

$$= \frac{\sum_{Rid=1}^{n} RCF}{N(R)}$$

Thus RCM gives a quantitative measurement of extent of coverage of requirements of a system in design.

### 5.3 Design Compliance Metrics (DCM)

Once the coverage of a requirement is determined, we next consider those requirements that have been implemented in sequence diagram i.e. U $\neq \phi$ and naturally, RUC $\neq$ 0 and USC $\neq$ 0. We measure the extent of consistency achieved in the implementation of these requirements. The DCM value is calculated which determines whether the requirement is consistently implemented in the design and measures the extent of consistency between sequence and class diagrams i.e. between the structural and behavioral design. This is computed for a particular use case and only for those use cases where U $\neq \phi$

$C_{SQ}$: The set of unique classes used in all the sequence diagrams (for a use case) taken together (Classes whose objects are used in sequence diagram)

$C_{SQ} = \{c_i \mid c_i \in SQ\}$

$C_{CL}$: The set of unique classes defined in class diagram.

$C_{CL} = \{c_j \mid c_j \in CL\}$

**CD (Class Differential)**

The class differential is computed for every class C used in all the sequence diagrams used for implementing a particular use case. It is defined as -

CD = i-j

(where i = 1 if $c_i \in C_{SQ}$, else i = 0
and j = 1 if $c_j \in C_{CL}$, else j = 0)
Here i and j are used as indicators and can assume values 0 and 1. This simply indicates whether a class is nonexistent or existent in a sequence diagram or a class diagram respectively. Therefore for every class C, The class differential is a measure of existence of a particular class in both the diagrams - structural and behavioral.

**Significance of CD**

If CD=0, class is either present in sequence as well as class diagram or else absent in both the diagrams.
If CD=1 then class is used in sequence diagram but not defined in class diagram.
If CD=-1 then it means that class is defined in class diagram but not used in sequence diagram.
$M_{SQ}$: The set of unique methods of a class used in all the sequence diagrams used for realizing a particular use case.
$M_{SQ} = \{m_i \mid m_i \in SQ\}$
$M_{CL}$: The set of unique methods defined for a specific class in the class diagram.
$M_{CL} = \{m_i \mid m_i \in CL\}$

**MD (Method Differential)**

The method differential is computed for every method M belonging to class C of the system. It is defined as -
MD = i-j
(where i = 1 if $m_i \in M_{SQ}$, else i = 0
and j = 1 if $m_i \in M_{CL}$, else j = 0)
Here i and j are used as indicators and can assume values 0 and 1. This simply indicates whether a method is nonexistent or existent in a sequence diagram or a class diagram. Therefore for every method m of class C, the method differential is a measure of existence of a particular method of a class in both the diagrams.

**Significance of MD**

If MD=0, method is present in sequence as well as class diagram or else absent in both.
If MD=1 then method is used in sequence diagram but not defined in class diagram.
If MD=-1 then it means that the method is defined in class diagram but not used in sequence diagram.

**1) UC (Underfined Classes)**

The undefined class metrics gives a measure of number of classes used in sequence diagrams but not defined in class diagram.
UC = $\Sigma i$ (i stands for $i^{th}$ class whose CD = 1)
This is the summation of all the classes having positive class differentials.

**2) CC (Consistent classes)**

The consistent class metrics gives a measure of number of classes, which have been defined as well as used in sequence diagram.
CC = $\Sigma i$ (i stands for $i^{th}$ class whose $CD_i$= 0)
This is the summation of all the classes having zero class differentials.

**3) CCF (Class Consistency Factor)**

This factor gives a measure of consistency of the classes used for implementation of a use case.
CCF = Consistent classes / Total Classes used
CC + UC = Total number of classes used
Therefore, CCF = CC / (CC+UC)
This factor can be computed for every class used for implementation of a use case i.e. for the set $C_U$.

**Significance of CCF**

If CCF = 1, it indicates that all the classes that have been used in the sequence diagram have been defined in class diagram i.e. UC = 0. This indicates that the design is consistently compliant with requirements as far as class definition and usage is concerned.
If CCF < 0, it indicates that there objects of certain classes used in sequence diagram which are not defined in the class diagram i.e. UC >0. This is a measure of the level of consistency of classes used for implementation of a use case.

**4) UM (Undefined methods - for a class)**

The undefined method metrics gives a measure of number of methods used in sequence diagrams but not defined in class diagram.
UM = $\Sigma i$ (i stands for $i^{th}$ method whose $MD_i$ = 1)
This is the summation of all the methods having positive method differentials.

**5) CM (Consistent methods -for a class)**

The consistent method metrics gives a measure of number of methods, which have been defined in class diagram as well as used in sequence diagram.
CM = $\Sigma i$ (i stands for $i^{th}$ method whose $MD_i$ = 0)
This is the summation of all the methods having zero class differentials.

**6) MCF (Method Consistency Factor)**

This gives a measure of consistency of the methods used of a class for implementing a use case.
MCF = Consistent methods / Total methods used
CM + UM = Total number of methods used
Therefore, MCF = CM / (CM+UM)
This factor can be computed for every method used for implementation of a use case i.e. for the set $M_U$.

**Significance of MCF**

If MCF = 1, it indicates that all the methods that have been used in the sequence diagram have been defined in class diagram i.e. UM = 0. This indicates that the design is consistently compliant with requirements as far as method definition and usage is concerned.
If MCF < 0, it indicates that there certain methods used

in sequence diagram which are not defined in the class diagram i.e. UM >0. This is a measure of the level of consistency of methods used for implementation of a use case.

**Design Compliance Metrics (DCM)**

The design compliance metrics (DCM) is computed from CCF and MCF for each use case as follows:

For all classes, DCM is calculated and finally an average is taken for a particular use case.

$$MCF_{av} = \frac{\Sigma MCF_i}{n}$$

where i=1..n are classes used

$$DCM_U = \frac{CCF + MCF_{av}}{2}$$

The value of DCM will be between 0 and 1 and we compute the average of all DCM's for all classes used for use case in the set U (implemented use case).

Finally the DCM for a requirement is calculated as the average of DCM value for all the use cases used to realize a requirement.

$$DCM = \frac{\Sigma DCM_i}{n}$$

where i=1..n are implemented use cases for a requirement.

A value of 1 indicates that the requirement has been consistently implemented in sequence and class diagrams. This implies that all methods and classes (objects) used in sequence diagrams are defined in class diagram.

A value less than 1 indicates the level of inconsistency in the behavioral and structural design.

## 6 CASE STUDY

We have considered an example of a Library management System (LMS) where a member can register, issue and return books from the library or cancel membership. The requirements document is shown in Fig 2.

| Requirement ID (rq) | Description |
|---|---|
| 01 | Members can issue books |
| 02 | Members can return books |
| 03 | Person can register to become a new member |
| 04 | Member can cancel membership |
| 05 | Members must return books within 15 days or pay late fine |

**Figure 2:** Requirements Document

### 6.1 UML Diagrams

The use case diagram is shown in Fig 3 where each requirement maps to a use case. The class diagram is shown in Fig 4. The sequence diagrams corresponding to the use cases "Issue Book", "Return Book" are shown in Fig 7 and Fig 8 respectively in the Appendix. Table-7 in Appendix shows the relationship between the

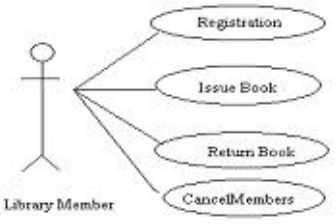


**Figure 3:** Use case Diagram of LMS

requirements and the design artifacts for the case study based on the definition in Fig 1 in the form of a trace table. As evident from the table, we have highlighted areas where certain methods and classes are not defined as well as the use cases that are not implemented in sequence diagrams.

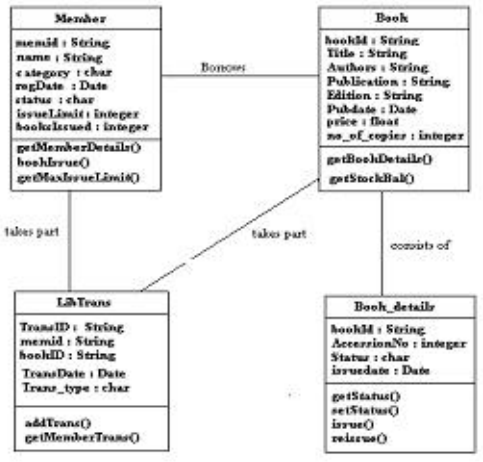


**Figure 4:** Class Diagram of LMS

### 6.2 Measuring Requirement-Design Metrics for LMS

In this section we show the results of application of our metrics on the case study we have chosen. The Table-7 in Appendix lists the trace of requirements to use cases, sequence diagrams methods and class methods. This table is referred in this section for calculation of the metrics.

**Requirement Coverage Metrics - RCM**
This consists of RUC, USC and SCC as defined earlier. For our case study, R = 5 (set of requirements)
**Calculation of RUC**
We calculate $U_R$ for each requirement, i.e. set of use cases of each requirement and then RUC as in Table-1.
**Calculation of USC**

**Table 1:** Calculation of RUC

| Requirement | $U_R$ | RUC |
|---|---|---|
| 01 | Issue Book | 1 |
| 02 | Return Book | 1 |
| 03 | Registration | 1 |
| 04 | Cancel Memebers | 1 |
| 05 | None | 0 |

We calculate $S_U$ for each use case, i.e. set of sequence diagrams corresponding to each requirement and then U as shown in Table 2.

**Table 2:** Calculation of $S_U$ and U

| $R(r_{id})$ | $U_R$ | $S_U$ | U |
|---|---|---|---|
| 01 | Issue Book | Issue (Fig 5) | Issue Book |
| 02 | Return Book | Return (Fig 6) | Return Book |
| 03 | Registration | None | $\Phi$ |
| 04 | Cancel Memebers | None | $\Phi$ |

For Requirements '01' and '02', there is only one use case and they have one sequence diagram implementation. Hence U is 1. For requirements '03' and '04', the use cases have not been implemented and hence U is 0. Therefore, USC for each requirement can be calculated as shown in Table-3. This indicates that only

**Table 3:** Calculation of USC

| Requirement | $N(U_R)$ | N(U) | USC $N(U)/N(U_R)$ |
|---|---|---|---|
| 01 | 1 | 1 | 1 |
| 02 | 1 | 1 | 1 |
| 03 | 1 | 0 | 0 |
| 04 | 1 | 0 | 0 |

"Issue Book" and "Return Book" use cases have been captured in design and the rest are missing from design models.
**Calculation of SCC**
We calculate MS i.e. the set of methods used in all the sequence diagrams for each requirement. Referring to Table-7, the number of methods used for every requirement in sequence diagrams and the methods that are defined in a class can be identified and tabulated as in Table-4 The USC value is zero for requirements

**Table 4:** Calculation of SCC

| Requirement | $N(M_S)$ | $N(M_{S-C})$ | SCC |
|---|---|---|---|
| 01 | 11 | 7 | 0.636 |
| 02 | 8 | 5 | 0.625 |

(03,04) and hence they are not considered for calculation of SCC. Finally from Table 1, Table 3 and Table 4, we calculate the RCF value for each requirement as shown in Table-5. Thus

**Table 5:** Calculation of RCF

| Requirement | RUC | USC | SCC | RCF |
|---|---|---|---|---|
| 01 | 1 | 1 | 0.636 | 0.878 |
| 02 | 1 | 1 | 0.625 | 0.875 |
| 03 | 1 | 0 | 0 | 0.33 |
| 04 | 1 | 0 | 0 | 0.33 |
| 05 | 0 | 0 | 0 | 0 |

$RCM = (0.878 + 0.875 + 0.33 + 0.33)/5 = 0.48$
This indicates that only 48% of requirements have been implemented in design.
**Design Compliance Metrics-DCM**
This metrics calculates CCF, MCF and hence DCM for

**Table 6:** Metrics (Class and Method Differential)

| Class Diagram | | Sequence Diagram | | | CD |
|---|---|---|---|---|---|
| Class Method | Class | Method | MD | | |
| | | InterfaceClass | isMemberValid | 1 | 1 |
| | | | isBookIssued | 1 | |
| | | | isOnHold | 1 | |
| | | | isBookAvl | 1 | |
| | | | issuAllowd | 1 | |
| | | | isBookidValid | 1 | |
| Member | getMember Details | Member | getMember Details | 0 | 0 |
| | BookIssue | | bookIssue | 0 | |
| | | | bookReturn | 1 | |
| | getMaxIssueLimit | | | -1 | |
| Book | getBook Details | Book | getBook Details | 0 | 0 |
| | getStockBal | | getStockBal | 0 | |
| Book Details | getStatus | BookDetails | getStatus | 0 | 0 |
| | setStatus | | setStatus | 0 | |
| | | | | | |
| | Issue | | Issue | 0 | |
| | Reissue | | Reissue | 0 | |
| | | | Return | 1 | |
| LibTrans | getMemberTrans | LibTrans | getMemberTrans | 0 | |
| | addTrans | | addTrans | 0 | |

each Use case. In this case only two use cases have been further implemented in design and we show the results for one use case “Issue Book" as an example.
Table-6 lists the class and method differential of each class and method used to implement the use case. From this we show the results of calculation of CCF, MCF and DCM.
From Table-6, the following metrics can be deduced-
UC=1,CC=4
CCF=Class Consistency Factor
=CC/(CC+UC)=4/5=0.8
This indicates that for the Library management system, about 80% of the classes are consistent and present in both class and sequence diagram.
Similarly, MCF can be calculated as given below-
For class InterfaceClass, MCF = 0
For class Member, MCF = 0.67
For class LibTrans, MCF = 1
For class Book DEtails, MCF = 0.8
For class Book, MCF = 1
$MCF_{av} = (0 + 1 + 0.67 + 0.8 + 1)/5 = 0.694$
Thus overall Design Compliance Metrics for “Issue Book" use case,
$DCM = (CCF + MCF_{av})/2 = (0.8 + 0.694)/2 = .75$
Since Requirement ‘Members can issue books’ (Rid = 01) implements only one use case ‘Issue Book’, this value of DCM indicates that the level of consistency of implementation of this requirement is 75%.
Likewise DCM for other requirements may be computed.

## 7 IMPLEMENTATION METHODOLOGY

UML is semi-formal nature and hence not suited for the process of automation or computerization, which requires well-defined precise formal semantics. XML bridges part of the gap by providing building blocks for serializing UML data textually. XMI (XML Metadata Interchange) is an open industry standard that applies XML to abstract systems such as UML [1].
We base our UML model transformations to XML on this standard. The XML schemas for these artifacts are shown in Schema 1, Schema 2 and Schema 3. For brevity, XML schemas for UML models are shown and of requirements document is omitted.

```
<xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="URI"
.     xmlns:xmi="http://www.omg.org/XMI"
.....
<xsd:complexType name="ud">
<xsd:complexType name="u"»
<xsd:sequence>
<xsd:element name="uid" type="xsd:integer"/>
<xsd:element name="udesc" type="xsd:string"/>
<xsd:element name=”actor” type=”xsd:string”
maxOccurs=”unbounded”/>
<xsd:element name=”rid” type=”xsd:integer” minOccurs=”1”/>
</xsd:sequence>
</xsd:complexType>
</xsd:complexType>
```

**Schema 1: XML schema for Use Case Diagrams**

```
<xsd:complexType name="sequence">
<xsd:sequence>
<xsd:element name=”sid” type=”xsd:string”>
<xsd:element name=”uid” type=”xsd:string”>
<xsd:complexType name="message"»
<xsd:element name="Torder" type="xsd:integer"/>
<xsd:choice>
<xsd:element name="fromClass" type="xsd:string"/>
<xsd:element name="fromActor" type="xsd:string"/>
</xsd:choice>
<xsd:choice>
<xsd:element name="toClass" type="xsd:string"/>
<xsd:element name="toActor" type="xsd:string"/>
</xsd:choice>
<xsd:choice>
<xsd:element name=”method” type=”xsd:string”/>
<xsd:element name=”text” type=”xsd:string”/>
</xsd:choice>
</xsd:complexType>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

**Schema 2: XML schema for Sequence Diagram**

```
<xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="URI"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsd="http://www.w3.org/2001/
XMLSchema" xmlns:p="URI">
<xsd:import namespace=http://www.omg.org/XMI
schemaLocation="xmi20.xsd"/>
<xsd:complexType name="cl">
<xsd:complexType name="c"»
<xsd:sequence>
<xsd:element name="name" type="xsd:string" />
<xsd:element name="attr" type="xsd:string"
minOccurs=”0” maxOccurs=”unbounded”/>
<xsd:element name=”method” type=”xsd:string” minOccurs=”0”
maxOccurs=”unbounded”/>
</xsd:sequence>
</xsd:complexType>
</xsd:complexType>
</xsd:schema>
```

**Schema 3: XML schema for class diagram**

Once the diagrams are represented in a structured manner as above, it would be easy to verify the rules by comparing the XML documents. For brevity, we present only some of them belonging to the Design Compliance metrics set.

**1) Set of unique classes used in all the sequence diagrams**

Refer to Schema 2, the contents of the tag <fromClass> and <toClass> constitute the set of all classes used in the sequence diagram. This is repeated if there are more than one sequence diagrams.

**2) Set of unique classes defined in class diagram.**

Refer to Schema 3, the contents of <name> tag constitute the set of all classes defined in class diagram.

**3) Class Differential (CD)**

This can be easily computed by comparing the sets (set1 and set2 say) obtained from metrics 1 and metrics2. Any class occurring only in set 1 will have CD=1, classes occurring in both the sets will have CD=0 and those occurring in only set 2 will have CD=-1.

Proceeding in the same manner, we can also implement the other metrics based on the XML documents. For brevity, examples for all the metrics have been omitted. The results if applied on the case study will yield the same results as discussed in the previous section.

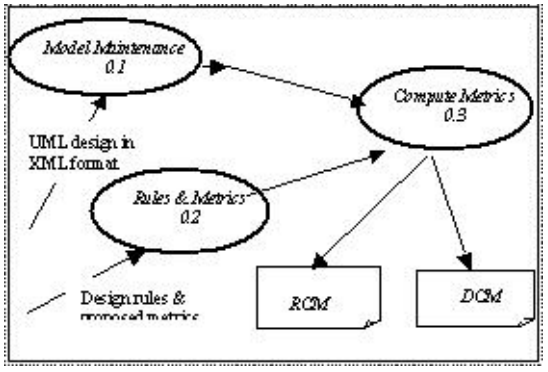


**Figure 5:** DFD of the Prototype

## 8 PROTOTYPE

A prototype has been developed which is XML based for implementing our approach. It accepts UML diagrams in XML format as input and calculates the metrics. The 1st level DFD is shown in Fig 5.A sample screen calculating CCF is shown in Fig 6.

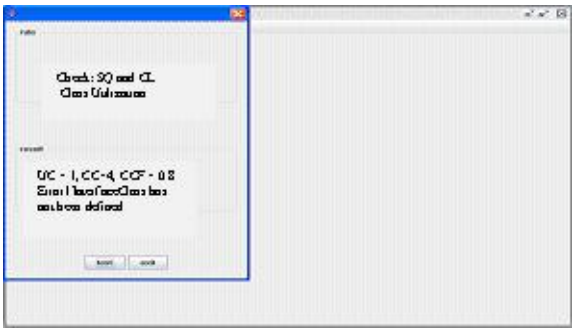

**Figure 6:** CCF-Class consistency Factor

## 9 CONCLUSION

UML has become a common standard for modeling design specifications of object-oriented systems, but being a visual language, is semi-formal in nature and hence verification of design in UML is necessary. In this paper we present a metrics based analysis of requirements. We propose two new set of metrics based on UML models namely - Requirement Coverage metrics and Design Compliance Metrics. RCM gives a measure of the degree of coverage of a requirement in use case, sequence and class diagrams. This will help in identifying missing requirements, or incomplete implementation of requirements as well as the progress of a project at any point of time. This would prove a valuable input for quality assessment of software systems. DCM proposes a unique method for studying consistency between Class and sequence diagrams of UML by providing quantitative feedback on the level of consistency in design at any point of time. We have also proposed an XML based implementation and presented a prototype. In our future work we intend to extend this concept further and fine-tune the metrics by including other UML diagrams like collaboration, activity and state charts.

## References

[1] Omg standard xmi specification 2.0. *http://www.omg.org*, 2008.

[2] Ammann, P. and Black, P. E. A specification-based coverage metric to evaluate test sets. *Proc. of 4th IEEE Intl. Symposium on High-Assurance Systems Engineering*, pages 239–248, Nov 17-19 1999.

[3] Boris Litvak, S. T. and Yehudai, A. Behavioral consistency validation of uml diagrams. *First International Conference on Software Engineering*

*and Formal Methods (SEFM'03), Brisbane, Australia*, page pp 118, September 22-27 2003.

[4] Chidamber, S. R. and Kemerer, C. F. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[5] Egyed, A. Scalable consistency checking between diagrams-the viewintegra approach. *16th IEEE International Conference on Automated Software Engineering (ASE'01), , San Diego, California*, November 26-29 2001.

[6] Egyed, A. F. Automatically validating model consistency during refinement. *23rd International Conference on Software Engineering (ICSE 2001), Toronto, Ontario, Canada*, May 12-19 2001.

[7] Gregor Engels, R. H., Jan Hendrik Hausmann and Sauer, S. Testing the consistency of dynamic uml diagrams. *Sixth International conference on Integrated Design and Process Technology, Pasadena, California*, June 23-28 2002.

[8] Hamed, H. and Salem, A. Uml-l: An uml based design description language. *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'01), Beirut, Lebanon*, page pp 0438, June 25-29 2001.

[9] Hassan, A., Rabie, W., Moez, A., and Ammar, H. H. An approach to measure the quality of software architectures from uml specifications. *5th World Multi-Conference on Systems, Cybernetics and Informatics and the 7th international conference on information systems, analysis and synthesis ISAS*, July 2001.

[10] Jing Liu, J. H., Zhiming Liu and Li, X. Linking uml models of design and requirement. *2004 Australian Software Engineering Conference (ASWEC'04) Melbourne, Australia*, page pp 329, April 13-16 2004.

[11] Kim, H. and Boldyreff, C. Developing software metrics applicable to uml models. *Proceedings of 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, June 11th 2002.

[12] Kim, S.-K. and Carrington, D. A formal object-oriented approach to defining consistency constraints for uml models. *2004 Australian Software Engineering Conference (ASWEC'04), Melbourne, Australia*, page pp 87, April 13-16 2004.

[13] Krishnan, P. Consistency checks for uml. *Seventh Asia-Pacific Software Engineering Conference (APSEC'00), Singapore*, page pp 162, December 05-08 2000.

[14] Küster, J. M. and Stehr, J. Towards explicit behavioral consistency concepts in the uml. *Second International Workshop on Scenarios and State Machines : Models, Algorithms and Tools, Portland, Oregon, USA*, May 3 2003.

[15] Martin Wolf, R. B., Evgeni Ivanov and Philippow, I. Uml tool support: Utilization of object-oriented models. *Technology of Object-Oriented Languages and Systems (TOOLS 34'00), Santa Barbara, California*, page pp 529, July 30-August 3 2000.

[16] Pascal André, A. R. and Royer, J.-C. Checking consistency of uml class diagrams using larch prover. *Electronic Workshops in Computing (eWiC), Rigorous object-oriented Methods, York, UK*, 17th Jan 2000.

[17] Tom Mens, R., der Straeten, V., and Simmonds, J. Maintaining consistency between uml models with description logic tools. *Fourth International Workshop on Object-oriented Reengineering (WOOR2003), Darmstadt, Germany*, July 21 2003.

[18] Tsiolakis, A. and Ehrig, H. Consistency analysis of uml class and sequence diagrams using attributed graph grammars. *Proc. Of Joint APPLIGRAPH and GETGRATS workshop on graph transformation systems*, pages pp 77–86, March 25 2000.

[19] Whalen, M. W., Rajan, A., P.E., M., Steven, H., and Miller, P. Coverage metrics for requirements-based testing. *Proc. of the 2006 Intl. symposium on Software testing and analysis, ISSTA, Portland, USA*, pages 25–36, 2006.

[20] Xia, F. and Kane, G. S. Definign the semantics of uml class and sequence diagrams for ensuring the consistency and executability of oo software specification. *First International Workshop on Automated Technology for Verification and Analysis ATVA, National Taiwan University*, pages pp 77–86, December 10-13 2003.

[21] Zisman, A. and Kozlenkov, A. Knowledge base approach to consistency management of uml specifications. *16th IEEE International Conference on*

*Automated Software Engineering (ASE'01), San Diego, California*, page pp 359, November 26-29 2001.

## APPENDIX

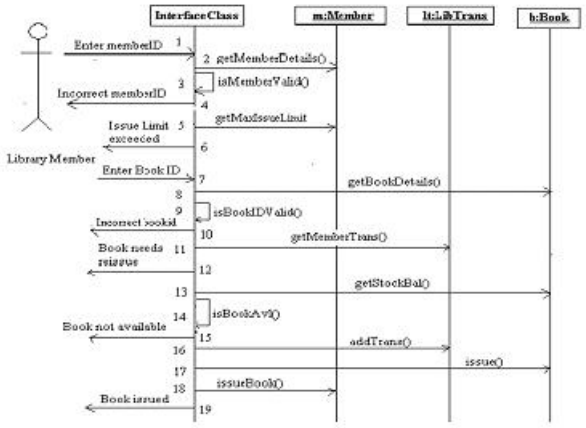


**Figure 7:** Sequence Diagram for "Issue Book" use case

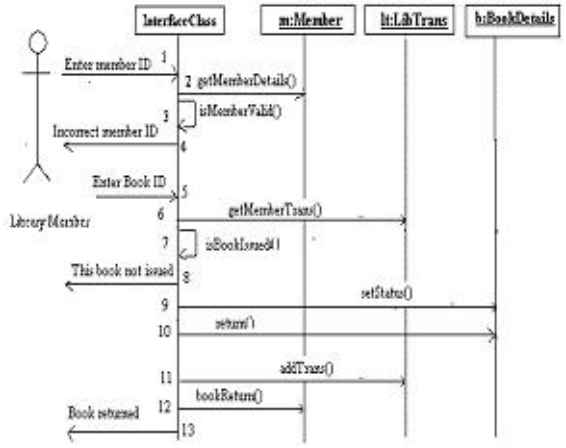


**Figure 8:** Sequence Diagram for "Return Book" use case

**Table 7:** Requirements to Methods trace table

| Requirements ID | Use Case Name | Sequence Diagram | | Class Name | Class Diagram Methods |
|---|---|---|---|---|---|
| | | Name | Methods | | |
| 01 | Issue Book | Issue (Fig 7) | getMember Details | Member | getMember Details |
| | | | isMember Valid | Interface Class | -(Class not defined) |
| | | | getMaxIssue Limit | Member | getMaxIssue Limit |
| | | | getBook Details | Book | getBook Details |
| | | | isBookID Valid | Interface Class | -(Class not defined) |
| | | | getMember Trans | LibTrans | getMember Trans |
| | | | getStockBal | Book | getStockBal |
| | | | isBookAvl | Interface Class | -(Class not defined) |
| | | | addTrans | LibTrans | addTrans |
| | | | Issue Book Details | | issue |
| | | | issueBook | Member | -(Method undefined) |
| 02 05 | Return Book | Return (Fig 8) | getMember Details | Member | getMember Details |
| | | | isMember Valid | Interface Class | -(Class not defined) |
| | | | getMember Trans | LibTrans | getMember Trans |
| | | | isBook Issued | Interface Class | -(Class not defined) |
| | | | setStatus | Book Details | setStatus |
| | | | Return | Book Details | return |
| | | | addTrans | LibTrans | addTrans |
| | | | bookReturn | Member | -(Method undefined) |
| 03 | Registration | No Sequence Diagrams | - | - | - |
| 04 | Cancel Members | No Sequence Diagrams | - | - | - |