Compliance of Analysis and Design Models in Object-Oriented Systems: A Metrics Based approach

Sabnam Sengupta
B.P.Poddar Institute of Management & Technology
137, V.I.P Road, Kolkata 700052
Tel: +919433076680
sabnam_sg@yahoo.com

Ananya Kanjilal
B.P.Poddar Institute of Management & Technology
137, V.I.P Road, Kolkata 700052
Tel: +919830513854
ag_k@rediffmail.com

Swapan Bhattacharya
National Institute of Technology
Durgapur 713205.
Tel: +919830128721
bswapan2000@yahoo.co.in

ABSTRACT
In an object-oriented environment, it is necessary to ensure that all the requirements are addressed in the analysis and design phase, and modeled consistently in UML diagrams, for visual depiction of the behavioral and structural aspects of the system. Metrics, which measure the extent of compliance between related models, will be a powerful tool for developers to have a quantitative feedback about the correctness of a system. In this paper we propose a metrics based approach to ensure compliance between UML analysis models like Activity diagrams and Design models like Class and State chart diagrams. We have proposed a Design Compliance Metrics II (DCM-II) to quantitatively measure the extent of consistency between activity, class and state chart diagrams implementing the use cases for the same requirement. The basis of measuring consistency is the relationship between the artifacts based on the common attributes. It also helps in measuring progress of a project and thus helps in project management. DCM-II verifies whether the requirements that have been covered in design have been consistently realized in activity, class and state charts. Two case studies have been considered and calculation of DCM-II has been done for illustration of our approach.

Categories and Subject Descriptors
D.2.8 [**Metrics**]: *Process Metrics*

D.2.10 [**Design**]: *Methodologies*

General Terms
Measurement, Design, Verification.

Keywords
Metrics based analysis, Requirement analysis, Design Compliance, Design Consistency

1. INTRODUCTION
In an object-oriented environment, requirements are modeled as use cases; use case events are depicted using activity diagrams; activity/action states along with decision blocks of activity diagrams and they are implemented as methods of various classes defined in the class diagram and often cause change of states of objects. It is necessary to ensure that these UML models used different phases of software development, are consistent. However, the process of verifying is very much manual, even today and there is hardly any way to quantitatively measure the degree of consistency among different models. 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.

In this paper, we have proposed a metrics based methodology to verify that requirements are consistently implemented in the analysis models like – statecharts, activity diagrams, class diagram etc. for a particular use case. We have proposed a new set of metrics named Design Compliance Metrics-II (DCM-II) and presented methods to derive the metrics from a given set of requirements and UML analysis and design models – use case, activity, statechart and class diagrams. Based on these metrics values, a quantitative feedback can be provided regarding 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. Observations for DCM-II on two case studies have been presented to illustrate our work.

2. RELATED WORK
This section presents a review of some of the research work that has been done in the area of verification of UML designs, especially related to consistency verification and metrics that captures certain attributes of the software system based on UML models.

Kim et al. in [17] 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 [21]. 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 [23], [22] have developed metrics to ensure coverage of requirements. In [23] 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 [22], a specification based coverage metrics has been defined to evaluate test sets.

In [20] a metrics suite is defined to measure the quality of design at an early development phase. The suite consists of 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 [1], [4], [5], [6], [7], [9], [10], [11], [12], [13], [14], [15] have used formal techniques for verification. Formal techniques range from Object-Z in [7], algebra in [9], attributed graph grammars in [15] focusing mainly on class diagrams and behavioral diagrams. An algorithmic approach to a consistency check between UML Sequence and State diagrams is described in [8] while [10] proposes a declarative approach using process algebra CSP for consistency checking between sequence and statecharts. In [2] an approach for automated consistency checking named

VIEWINTEGRA has been developed and in [3] 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 [6], [14] and [15]. 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 among the diagrams. This work is an extension of our previous work [24], where we proposed a set of metrics to measure the extent of coverage of requirement through use case, sequence and class diagram. It differs from [17] 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 degree of consistency within the design.

3. SCOPE OF WORK

In this paper we have proposed a set of metrics based on requirements and UML analysis & design models for an object oriented system which will help in measuring the degree of compliance and consistency of these models with respect to requirements. In an object-oriented system, use case diagrams of UML form the basis of requirements activity, class and statechart diagrams model the implementation of use cases (requirements) showing the static and dynamic aspects. We have proposed DCM-II, which is extension of DCM-I, as proposed in [24] and will address an important issue-
Measuring the extent of consistency between the activity, statechart and class diagrams will ensure that the requirements have been consistently implemented in design.
We have considered an ATM system and a library management system as our example and our approach has been applied to these case studies and metrics have been calculated.

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, activity, statechart and class diagrams that show the requirements and their implementations within the analysis and design phase. Use cases model requirements and ideally one requirement may be mapped to one or more use cases. Each use case is described textually as the flow of events of a use case. These textual flows of events are diagrammatically depicted using activity diagram as action or activity states, decision blocks, swimlanes and object flow. The states of the objects are depicted using statechart diagram. The transitions among the different object states map with the methods. The action or activity states map with the methods, too. Each method is defined in the Class diagram that defines the structural aspects of the system.
The relationship between Use case, Activity, Statechart and Class diagrams are based on the existence of common elements between the diagrams. This forms the basis of definition of DCM-II discussed in next section.

Conference'04, Month 1–2, 2004, City, State, Country.

5. PROPOSED WORK

Our proposed metrics for analyzing requirements is based on the design of the system captured in the UML models. We have formulated the metrics based on the relationships that have been identified in earlier sections. Metrics are measurements based on project parameters that serve to give a quantitative measurement of various aspects. We first define some terms related to DCM-II.

5.1 Definition: Design Compliance

A design is compliant with requirements if the behavioral model (here activity and statechart 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-II) is thus a measure of the extent of consistency among activity, statechart and class diagrams for a use case.

5.2 Notations used

In this section the set of 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:
U – Set of Use cases
UC– Set of Use case diagrams
E – Set of Events
ACT – Set of Activity Diagrams
Ac – Set of Action/Activity States
C – Set of Classes
CL– Set of Class diagrams
STD – Set of Statechart diagrams
ST – Set of States
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.

U_R: The set of unique use cases defined in use case diagram corresponding to a particular requirement
$U_R = \{u_i \mid u_i \varepsilon U \text{ and } U \varepsilon UC, u_i \text{ implements } r_i, r_i \varepsilon R\}$
If this set is empty, it indicates that requirements have not been captured in the use case diagrams of UML.

A use case is described using a flow of events.
E_U: The set of unique events describing a particular use case
$E_U = \{e_i \mid e_i \varepsilon E, e_i \text{ describes } u_i, u_i \varepsilon U\}$

The textual flow of events for a particular use case is diagrammatically represented as action or activity states of activity diagram, along with decision blocks, object flow and swimlanes. These swimlanes represent the names of the objects. A single activity diagram or a set of activity diagrams (showing alternate flow of events) is defined as -

A_U: The set of activity diagrams used to represent flow of events of a particular use case u_i in UR, UUC

$A_U = \{ac_i \mid ac_i \varepsilon AC, ac_i \text{ represents } e_i, e_i \varepsilon E_U\}$
If this set is empty it means that none of the events of use case U has been implemented and represented in any activity diagrams.

U: The set of implemented use cases i.e. those use cases that have at least one corresponding activity diagram for implementation (i.e. for which $A_U \neq \varphi$)

OBJ: The set of unique objects used in all the active ity diagrams, either as object flow or as swimlane.
OBJ = {obj_i, st_i | obj_i ε CL and st_i ε ST} | { obj_i | obj_i ε CL}

ST: The set of states of different objects used to implement and design all the use cases for a particular requirement.
A statechart diagram is used to depict the different states of an object and transitions among the states.

STD: The set of all the statechart diagrams used to implement a particular requirement.

STD= {obj_i, tr_i | obj_i ε CL, tr_i ε TR, maps with ac_i, ac_i ε AC}

TR: The set of all the transitions among different states of all the objects, whose state transitions are depicted, in designing for a particular requirement.

TR = {$fromSt_i$, m_i, $toSt_i$ | $fromSt_i$ ε ST, $toSt_i$ ε ST, m_i ε M}

MC: The set of methods defined in a class diagram
MC = {m_j | m_j ε M, m_i ε c_i , c_i ε C, C ε CL}

5.3 Design Compliance Metrics (DCM-II)

We measure the extent of consistency achieved in the implementation of a requirement. The DCM value is calculated which determines whether the requirement is consistently implemented in the design and measures the extent of consistency between activity, statechart 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 $U \neq \phi$

A. Usecase-Activity consistency(U-AC)

Euc: Set of Events describing a usecase
Euc = {e_i | e_i ε U}
Eac: Set of events handled as action/activity states in Activity Diagram
Eac = {e_i | e_i ε A}

ED (Event Differential)

The event differential is computed for every events captured in all the activity diagrams as action or activity states used for implementing a particular use case. It is defined as –

ED = i – j
(where i = 1 if e_i ε E_{uc}, else i = 0
and j = 1 if e_j ε E_{ac}, else j = 0)
Here i and j are used as indicators and can assume values 0 and 1. This simply indicates whether an event is nonexistent or existent in a use case diagram or an activity diagram respectively. Therefore for every event E, The event differential is a measure of existence of a particular event in both the diagrams – structural and behavioral.

Significance of ED

If ED= 0, events present/absent in both use case and activity diagrams (consistent events)
If ED=1, events present in use case but, not in activity (unimplemented events)
If ED=-1, Events present in activity, not in usecase (irrelevant events)

B. Activity-Class consistency(AC-CL)

C_{ac}: Set of Classes/objects used in all Activity diagrams for a particular use case/
Cac = {ci | ci ε AC}
C_{cl}: Set of Classes defined in Class diagram
Ccl = {cj | cj ε CL}

CD_{AC-CL} (Class Differential)

The class differential is computed for every class C of objects used in all the activity diagrams used for implementing a particular use case. It is defined as –

CD_{AC-CL} = i – j
(where i = 1 if ci ε Cac, else i = 0
and j = 1 if cj ε Ccl, 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 an activity 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_{AC-CL}

If CD_{AC-CL} =0, class is either present in activity as well as class diagram or else absent in both the diagrams. (consistent classes)
If CD_{AC-CL} =1 then class is used in activity diagram but not defined in class diagram. (unimplemented classes)
If CDAC-CL =-1 then it means that class is defined in class diagram but not used in activity diagram. (Unused classes)

Mac: Set of methods/actionstate/decisionBlock used in Activity diagram which corresponds to activity events.
Mcl: Methods defined in Class diagram

MD_{AC-CL}(Method Differential)

The method differential is computed for every method M belonging to class C of the system. It is defined as –

MD_{AC-CL} = i – j
(where i = 1 if m_i ε M_{ac}, else i = 0
and j = 1 if m_j ε 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 an activity 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_{AC-CL}

If MD_{AC-CL}=0, method is present is activity as well as class diagram or absent in both. (consistent methods). This is same as M_{A-C} where
M_{A-C}: The set of implemented action or activity states in a activity diagram as well as defined in any class of class diagram.
i.e. M_{A-C} = {mac | mac ε Ac and mac ε Mcl }

If MD_{AC-CL} =1 then method is used in activity diagram but not defined in class diagram. (unimplemented method)
If MD_{AC-CL} =-1 then it means that method is defined in class diagram but not used in activity diagram (unused methods)

C. Statechart-Class consistency(ST-CL)

$\mathbf{C_{st}}$: Set of Classes/objects used in all statechart diagrams for a particular use case/
$C_{st} = \{c_i \mid c_i \varepsilon ST\}$
$\mathbf{C_{cl}}$: Set of Classes defined in Class diagram
$C_{cl} = \{c_j \mid c_j \varepsilon CL\}$

$\mathbf{CD_{ST\text{-}CL}}$ (Class Differential)

The class differential is computed for every classes of objects C used in all the statechart diagrams used for implementing a particular use case. It is defined as –

$\mathbf{CD_{ST\text{-}CL}} = i - j$

(where i = 1 if $c_i \varepsilon C_{st}$, else i = 0
and j = 1 if $c_j \varepsilon C_{cl}$, 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 an statechart 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 $\mathbf{CD_{ST\text{-}CL}}$

If $\mathbf{CD_{ST\text{-}CL}}$ =0, class is either present in statechart as well as class diagram or else absent in both the diagrams. (Consistent classes)
If $\mathbf{CD_{ST\text{-}CL}}$ =1 then class is used in statechart diagram but not defined in class diagram. (unimplemented classes)
If $\mathbf{CD_{ST\text{-}CL}}$ =-1 then it means that class is defined in class diagram but not used in statechart diagram. (Classes not changing states: steady state classes)

M_{st}:Set of methods used in statechart diagram as a part of transition
M_{cl}: Methods defined in Class diagram

$\mathbf{MD_{ST\text{-}CL}}$ (Method Differential)

The method differential is computed for every method M belonging to class C of the system. It is defined as –

$\mathbf{MD_{ST\text{-}CL}} = i - j$

(where i = 1 if $m_i \varepsilon Mst$, else i = 0
and j = 1 if $m_j \varepsilon M_{cl}$, 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 statechart 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 $\mathbf{MD_{ST\text{-}CL}}$

If $\mathbf{MD_{ST\text{-}CL}}$ =0, method is present is statechart as well as class diagram or absent in both. (consistent methods)
If $\mathbf{MD_{ST\text{-}CL}}$ =1 then method is used in statechart diagram but not defined in class diagram. (unimpl. method)
If $\mathbf{MD}_{ST\text{-}CL}$ =-1 then it means that method is defined in class diagram but not used in statechart diagram. (unused methods)

D. Activity-Statechart consistency(AC-ST)

$\mathbf{C_{ac}}$: Set of Classes of objects defined in activity diagram either as object flow, or as swimlanes
$C_{ac} = \{c_j \mid c_j \varepsilon AC\}$
$\mathbf{C_{st}}$: Set of Classes/objects used in all statechart diagrams for a particular use case/
$C_{st} = \{c_i \mid c_i \varepsilon ST\}$

$\mathbf{CD_{AC\text{-}ST}}$ (Class Differential)

The class differential is computed for every classes of objects C used in all the activity diagrams and statechart diagrams used for implementing a particular use case. It is defined as –

$\mathbf{CD_{AC\text{-}ST}} = i - j$

(where i = 1 if $c_i \varepsilon C_{ac}$, else i = 0
and j = 1 if $c_j \varepsilon C_{st}$, 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 an activity diagram or a statechart. Therefore for every class C, the class differential is a measure of existence of a particular class in both the diagrams, both of them are behavioral.

Significance of $\mathbf{CD_{AC\text{-}ST}}$

If $\mathbf{CD_{AC\text{-}ST}}$ =0, class is either present in activity, as well as statechart or else absent in both the diagrams. (consistent classes)
If $\mathbf{CD_{AC\text{-}ST}}$ =1 then class is used in activity diagram but not defined in statechart diagram. (steady state. classes)
If $\mathbf{CD_{AC\text{-}ST}}$ =-1 then it means that class is defined in statechart diagram but not used in activity diagram. (unused classes)

However, all objects used in activity may or may not undergo state changes and hence may not be present in statecharts. The method differential does not hold any significance as far as activity and statecharts are concerned.

1) UE (Undefined events)

The undefined event metrics gives a measure of number of events used in use case diagrams but not defined in activity diagram.

$UE = \sum i$ (i stands for i^{th} class whose $\mathbf{ED} = 1$)

This is the summation of all the events having positive event differentials.

2) CE (Consistent events)

The consistent event metrics gives a measure of number of events used in use case diagrams as well as implemented in activity diagram.

$CE = \sum i$ (i stands for i^{th} class whose $\mathbf{ED} = 0$)

This is the summation of all the events having zero class differentials.

3) ECF (Event Consistency Factor)

This factor gives a measure of consistency of the events used for describing and depicting a use case.

ECF = Consistent events / Total events
CE + UE = Total number of events
Therefore, ECF = CE / (CE+UE)

This factor can be computed for every event used for description and depiction of a use case.

Significance of ECF

If ECF = 1, it indicates that all the events that have been used in the use case have been described in activity diagram i.e. UE = 0.
If ECF < 0, it indicates that there are events used in activity diagrams which are not defined in the use case diagram i.e. UE >0.

4) UC (Undefined classes)

The undefined class metrics gives a measure of number of classes used in activity or statechart diagrams but not defined in class diagram. It is computed by considering $CD_{AC\text{-}CL}$, $CD_{ST\text{-}CL}$ and $CD_{AC\text{-}ST}$

$UC = \sum i$ (i stands for i^{th} class whose $\mathbf{CD_{AC\text{-}CL}} = 1$ or $\mathbf{CD_{ST\text{-}CL}} = 1$)

5) HC (Helper class)

This metrics gives a measure of the number of classes defined in the class diagram but not used in either activity diagram or statechart diagram for any use case. This may be a possible case of redundant or unused classes but not necessarily as some of them may be helper classes useful for the overall system.

HC = ∑i (i stands for i^{th} class whose $CD_{AC\text{-}CL}$ = -1 and $CD_{ST\text{-}CL}$ = -1)

6) CC (Consistent class)

The consistent class metrics gives a measure of number of classes, which have been defined in class diagram as well as used in activity and statechart diagram.

CC = ∑i (i stands for i^{th} class whose $\mathbf{CD_{AC\text{-}CL}} = 0$, $\mathbf{CD_{ST\text{-}CL}} = 0$ and $\mathbf{CD_{AC\text{-}ST}} = 0$)

This is the summation of all the classes having zero class differentials.

7) 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

Total number of classes used = UC + CC + HC

Therefore, CCF = CC / (UC + CC + HC)

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 means that UC =0 as well as HC=0

It indicates that all the classes that have been used in the activity and statechart diagrams have been defined in class diagram i.e. UC = 0. Moreover there are no classes that are defined but not used i.e. HC=0.

This indicates that the analysis and design is consistently compliant with requirements as far as class definition and usage is concerned

If CCF < 1, it means that

a) UC > 0 i.e. there are objects of certain classes used in activity and statechart diagrams which are not defined in the class diagram.
b) HC > 0 i.e. there are some classes that are not used in activity and statechart diagrams.

Degree of consistency of Requirement Implementation

- If CCF=1 or CCF<1 and UC=0, then this is a case of consistent implementation of requirements in analysis and design phase.
- Other values of CCF indicates the level or degree of consistency achieved in analysis and design

8) UM (Undefined methods – for a class)

The undefined method metrics gives a measure of number of methods used in activity diagram and/or statechart diagram but not defined in class diagram. It is computed by considering $M_{AC\text{-}CL}$, $M_{ST\text{-}CL}$ and $MD_{AC\text{-}ST}$

UM = ∑i (i stands for i^{th} method whose $\mathbf{MD_{AC\text{-}CL}} = 1$, $\mathbf{MD_{ST\text{-}CL}}$=1)

This is the summation of all the methods having positive method differentials.

9) HM (Helper method –for a class)

This metrics gives a measure of the number of methods defined in the class diagram but not used activity or statechart diagram for any use case. This may be a possible case of redundant or unused methods but not necessarily as some of them may be helper methods useful for the overall system.

HM = ∑i (i stands for i^{th} method whose $MD_{SQ\text{-}CL}$ = -1)

10) 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 activity and statechart diagram.

CM = ∑i (i stands for i^{th} method whose $\mathbf{MD_{AC\text{-}CL}} = 0$, $\mathbf{MD_{ST\text{-}CL}}$= 0)

This is the summation of all the methods having zero class differentials.

9) 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

Total number of methods used = UM + CM + HM

Therefore, MCF = CM / (UM + CM + HM)

This factor can be computed for every method in each class used for implementation of a use case i.e. for the set M_U.

Significance of MCF

If MCF = 1, it means that UM =0 as well as HM=0

It indicates that all the methods that have been used in the activity and statechart diagrams have been defined in class diagram i.e. UM = 0. Moreover there are no methods that are defined but not used i.e. HM=0.

This indicates that the analysis and design is consistently compliant with requirements as far as method definition and usage is concerned

If MCF < 1, it means that

a) UM > 0 i.e. there are certain methods used in activity and statechart diagrams which are not defined in the class diagram.
b) HM > 0 i.e. there are some methods that are not used in activity diagram.

Degree of consistency of Requirement Implementation

If MCF=1 or MCF<1 and UM=0, then this is a case of consistent implementation of requirements in analysis and design phase.

Other values of MCF indicates the level or degree of consistency achieved in analysis and design

By taking average of MCF values for all the classes of the class diagram,

$$MCF_{av} = \frac{\sum MCF_i}{n}$$ where i=1..n are classes used

Design Compliance Metrics II (DCM II)

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

$$DCM_U = \frac{ECF + CCF + MCFav}{3}$$

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.

$$DCM = \frac{\sum DCM_i}{n}$$ where i=1..n are implemented use cases for a requirement.

A value of 1 indicates that the requirement has been consistently implemented in activity, statechart and class diagrams. This implies that all methods and classes (objects) used in activity and statechart 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 two examples, one of an ATM management System where a user can deposit to or withdraw from his bank account and the other one is a library management system, where the library member can issue and return books. Our metrics is applied to both and results are presented in the following sub sections.

A. ATM System

The use case diagram is shown in Fig 1 where each requirement maps to a use case. The events of the use case "Withdraw" is shown in Fig 2. The activity diagram corresponding to the basic flow of event of use case "Issue Book", and the statechart diagram of the "Member" object is shown in Fig 3 and Fig 4, respectively. The class diagram is shown in Fig 5.

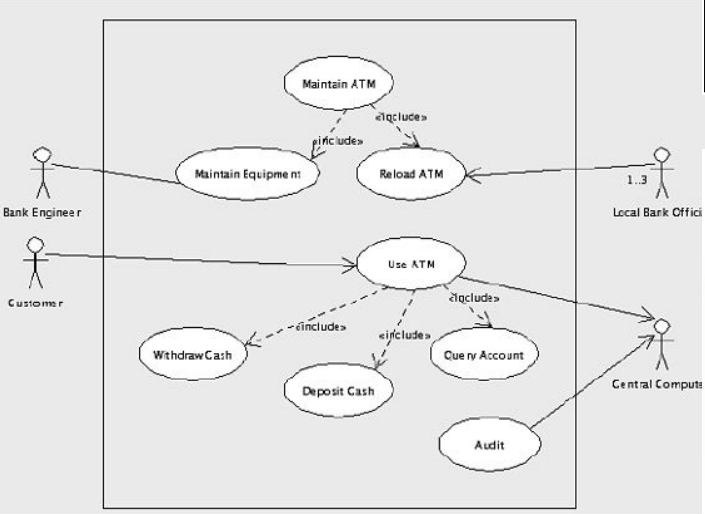

Fig 1: Use case Diagram of ATM System

Use Case Name	Event ID	Event Description
WithdrawCash	01	Customer inserts Card
	02	Customer enters PIN
	03	Authorization of card done
	04	If PIN is invalid, card ejected
	05	If PIN is valid, enter amount to withdraw
	06	Check account balance
	07	If sufficient balance not available, display account balance
	08	Debit balance by withdrawAmount
	09	Dispense Withdaw Amount as cash through ATM slot
	10	Display account balance
	11	Eject card
	12	Customer collects card

Fig 2: Basic Flow of Events for "Withdraw Cash" use case

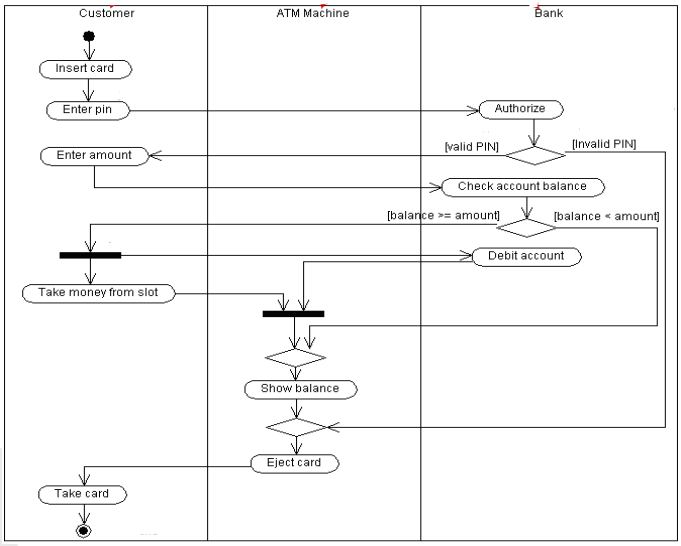

Fig 3: Activity Diagram depicting Flow of Events for "Withdraw Cash" use case

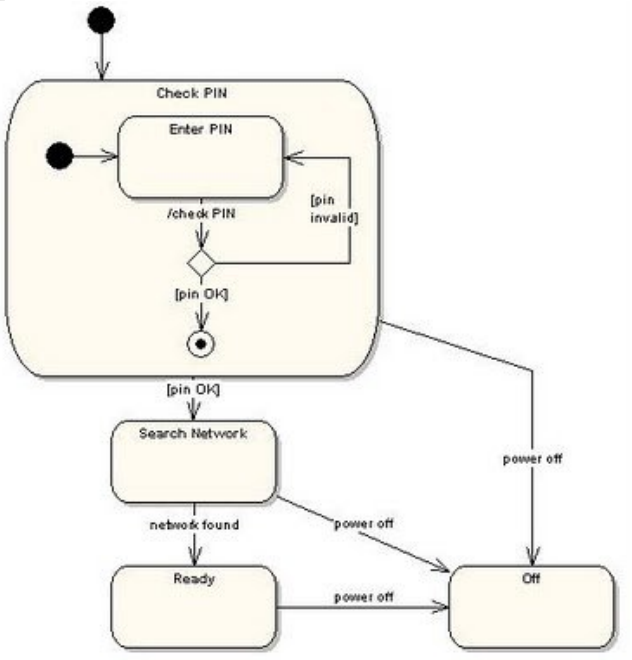

Fig 4: State chart diagram of "ATM Machine" object

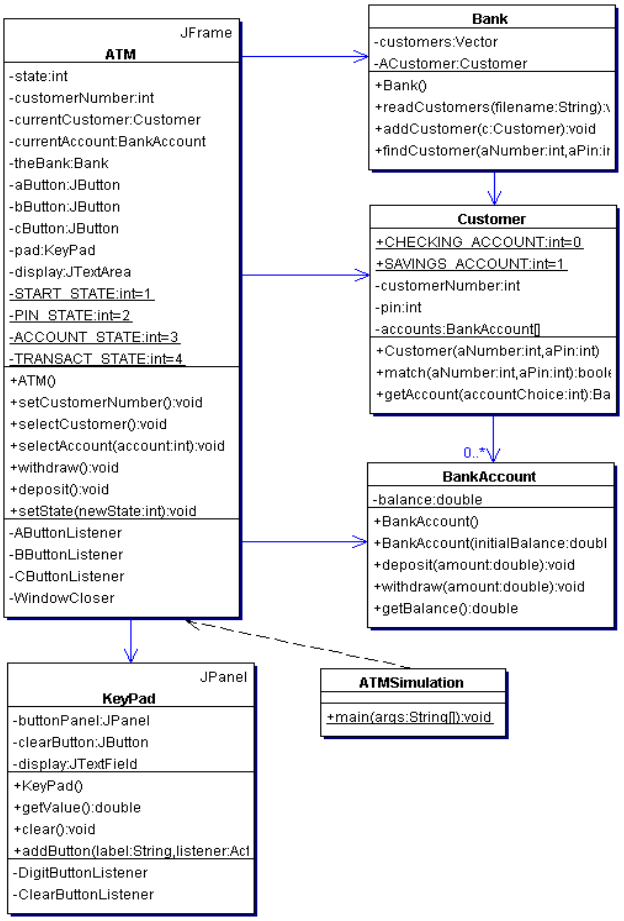

Fig 5: Class Diagram of ATM System

- **Requirement-Design Metrics for ATM**

In this section we show the results of application of our metrics on the ATM case study.

Design Compliance Metrics - DCM

This metrics calculates ECF, CCF, MCF and hence DCM for 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 "Withdraw Cash" as an example.

Table-1 shows the computation of Event Differential (ED) and Event Consistency Factor (ECF) between use case & activity events. Table 2, 3, 4 show Class & Method differentials between activity-class, statechart-class and activity statechart diagrams, respectively.

Use Case Diagram		Activity State Diagram		ED
UsecaseID	EventID	Activity Event	Class	
Withdraw Cash	01	Customer inserts Card	Customer	0
	02	Customer enters PIN		0
	05	If PIN is valid, enter amount to withdraw		0
	09	Dispense Withdaw Amount as cash through ATM slot		0
	12	Customer collects card		0
	10	Display account balance	ATM	0
	11	Eject card		0
	03	Authorization of card done	Bank	0
	06	Check account balance		0
	08	Debit balance by withdrawAmount		0
	04	-	-	1
ECF= (CE/(CE+UE)) = 10/11 = 0.90				

Table 1: Metrics (Event Differential)

Activity State Diagram		Class Diagram		MD AC-CL	CD AC-CL
Activity Event	Class	Method	Class		
Insert Card	Customer	Custome	Customer	0	0
Enter PIN		Custome		0	
Enter amount		getAccount		0	
Take money from slot		match		0	
Take card		-		1	
Show balance	ATM Machine	-	ATM	1	0
Eject card		-		1	
-		setCustomerNumber		-1	
-		selectCustomer		-1	
-		selectAccount		-1	
-		withdraw		-1	
-		deposit		-1	
-		setState		-1	
Authorize	Bank	readCustomer	Bank	0	0
Check account balance		-		1	
Debit account		withdraw		0	
-	-	deposit	BankAccount	-1	-1
-		withdraw		-1	
-		getBalance		-1	

Table 2: Metrics AC-CL (Class and Method Differential)

Statechart Diagram		Class Diagram		MD ST-CL	CD ST-CL
Method	**Class**	**Method**	**Class**		
checkPin	ATM-Machine	ATM ()	ATM-Machine	0	0
Pin-ok		-		1	
Network-found		-		1	
Power-off		-		1	

Table 3: Metrics ST-CL (Class and Method Differential)

Statechart Diagram		Activity Diagram		MD ST-AC	CD ST-AC
Class	**Method**	**Class**	**Method**		
ATM-Machine	checkPin	ATM-Machine	-	1	0
	Pin-ok		-	1	
	Network-found		-	1	
	Power-off		-	1	

Table 4: Metrics ST-AC (Class and Method Differential)

Based on the values of Class and Event Differentials from Table 2, 3 and 4, Class Consistency Factor (CCF) and Method Consistency Factor (MCF) are calculated as shown in Table 5 and 6, respectively.

Class	CD_{AC-CL}	CD_{ST-CL}	Type	CCF
Customer	0	-	CC	3/4=0.75
ATM-Machine	0	0	CC	
Bank	0	0	CC	
BankAccount	-1	-1	HC	

Table 5: Calculation of CCF for LMS

This indicates that for the ATM system, about 75% of the classes is consistent

Similarly, MCF can be calculated as given below-

Class	Method	MD AC-CL	MD ST-CL	Type	MCF CM / (UM + CM + HM)
Customer	customer	0	-	CM	1
	getAccount	0	-	CM	
	match	0	-	CM	
ATM machine	ATM	-1	-	HM	0
	setCustomerNumber	-1	0	HM	
	selectAccount	-1	0	HM	
	selectCustomer	-1	0	HM	
	Withdraw	-1	0	HM	
	Deposit	-1	0	HM	
	setState	-1	0	HM	
Bank	bank	0		CM	0.25
	readCustomer	-1		HM	
	addCustomer	-1		HM	
	findCustomer	-1		HM	
BankAccount	bankAccount				0
	deposit	-1		HM	
	Withdraw	-1		HM	
	getBalance	-1		HM	
MCFav					0.32

Table 6: Calculation of MCF for ATM

Thus overall Design Compliance Metrics for "Withdraw Cash" use case,

DCM = (ECF + CCF + MCFav)/3 = (0.9 + 0.75 + 0.32)/3 = 0.656

This value of DCM indicates that the level of consistency of implementation of "Withdraw Cash" use case is 65.6%.

Likewise DCM for other use cases may be computed.

B. Library Management System

The use case diagram is shown in Fig 6 where each requirement maps to a use case. The events of the use case "Issue Book" is shown in Fig 7. The activity diagram corresponding to the basic flow of event of use case "Issue Book", and the statechart diagram of the "Member" object is shown in Fig 8 and Fig 9, respectively. The class diagram is shown in Fig 10.

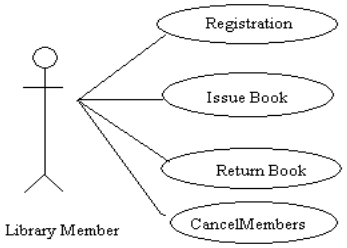

Fig 6: Use case Diagram of LMS

Use Case Name	Event ID	Event Description
Issue Book	01	The librarian enters the member ID
	02	Member validation
	03	Checking is made if issue limit for the member has exceeded.
	04	The librarian enters book id.
	05	Validation of the book takes place.
	06	Checking is made if book needs to be re-issued

Use Case Name	Event ID	Event Description
	07	Checking is made to see if any demand is pending on the book
	08	If no demand is pending re-issue done
	09	If book is not for re-issue, checking is made to see if book is available
	10	If available, issue book
	11	If not available, place demand on book
	12	If demand placed, ask for any other book of the same author/subject

Fig 7: Basic Flow of Events for "Issue Book" use case

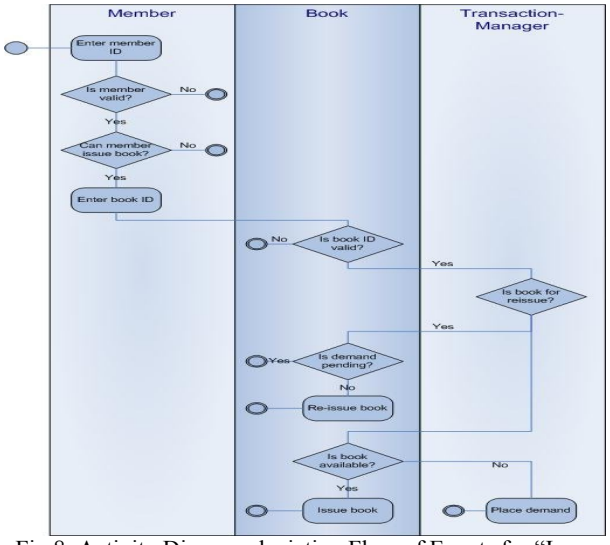

Fig 8: Activity Diagram depicting Flow of Events for "Issue Book" use case

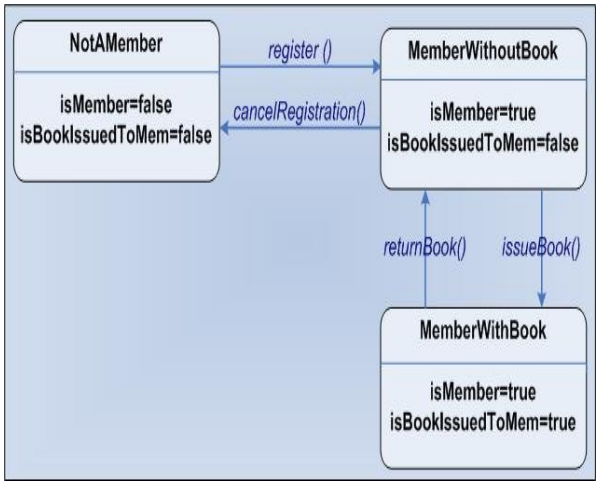

Fig 9: State chart diagram of "Mamber" object of LMS

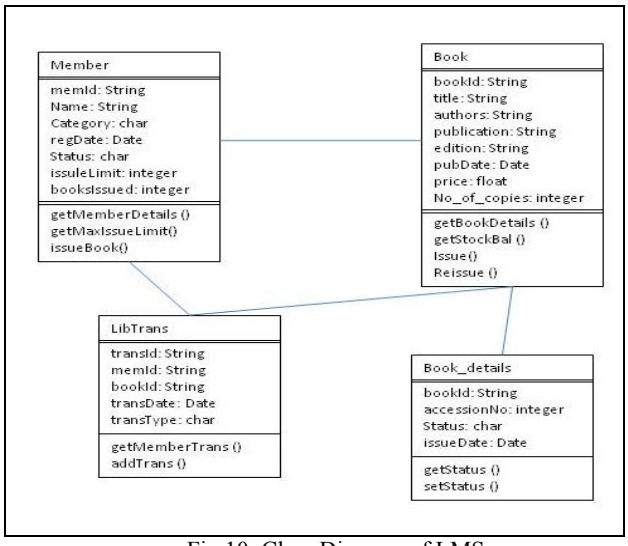

Fig 10: Class Diagram of LMS

- **Requirement-Design Metrics for LMS**

In this section we show the results of application of our metrics on the Library Management case study.

Design Compliance Metrics - DCM

This metrics calculates ECF, CCF, MCF and hence DCM for 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 7 shows the computation of Event Differential (ED) and Event Consistency Factor (ECF) between use case & activity events. Table 8, 9, and 10 show Class & Method differentials between activity-class, statechart-class and activity statechart diagrams, respectively.

Use Case Diagram		Activity State Diagram		ED
Use Case ID	**Event ID**	**Activity Event**	**Class**	
Issue Book	1	Enter Member ID	Interface Class	0
	4	Enter Book ID		0
	2	Is Member valid	Member	0
	3	Can member issue book		0
	-	-		-
	5	Is BookID valid	Book	0
	9	Is book available		0
	6	Is book for re-issue	LibTrans	0
	11	Place Demand		0
	10	Issue Book		0
	8	Re-issue book		0
	11	Place demand	Book_Details	0
	10	Issue Book		0
	8	Re-issue book		0
	7	Is demand pending		0

Use Case Diagram		Activity State Diagram		ED
Use Case ID	**Event ID**	**Activity Event**	**Class**	
Issue Book	1	Enter Member ID	Interface Class	0
	4	Enter Book ID		0
	12	-	-	1
ECF = CE/(CE+UE) = 16/17				0.94

Table 7: Metrics (Event Differential)

Activity State Diagram		Class Diagram		MD AC-CL	CD AC-CL
Activity Event	**Class**	**Method**	**Class**		
Enter Member ID	Interface Class	-	-	-	1
Enter Book ID		-		-	
Is Member valid	Member	getMemberDetails	Member	0	0
Can member issue book		getMaxIssueLimit		0	
-		bookIssue		-1	
Is BookID valid	Book	getBook Details	Book	0	0
Is book available		GetStockBal		0	
Is book for re-issue	LibTrans	-	LibTrans	1	0
Place Demand		addTrans		0	
Issue Book		addTrans		0	
Re-issue book		addTrans		0	
Place demand	Book_Details	setStatus	Book_Details	0	0
Issue Book		Issue		0	
Re-issue book		reissue		0	
Is demand pending		getStatus		0	
-	-	-	-	-	

Table 8: Metrics AC-CL (Class and Method Differential)

Statechart Diagram		Class Diagram		MD ST-CL	CD ST-CL
Method	**Class**	**Method**	**Class**		
issueBook	Member	issueBook		0	0
issue	Book	issue	Book	0	
return		return		0	

Statechart Diagram		Class Diagram		MD ST-CL	CD ST-CL
Method	**Class**	**Method**	**Class**		
placeDemand		placeDemand		0	

Table 9: Metrics ST-CL (Class and Method Differential)

Statechart Diagram		Activity Diagram		MD ST-AC	CD ST-AC
Class	**Method**	**Class**	**Method**		
Member	issueBook	Member	Issue book	0	0
Book	issue	Book	issue	0	0
	placeDemand		Place Demand	0	

Table 10: Metrics ST-AC (Class and Method Differential)

Based on the values of Class and Event Differentials from Table 7, 8, 9 and 10, Class Consistency Factor (CCF) and Method Consistency Factor (MCF) are calculated as shown in Table 11 and 12, respectively.

Class	CD_{AC-CL}	CD_{ST-CL}	Type	CCF
Interface	1	-	UC	3/5=0.6
Member	0	0	CC	
Book	0	0	CC	
Book_details	-1	-1	HC	
LibTrans	0	0	CC	

Table 11: Calculation of CCF for LMS

This indicates that for the Library management system, about 60% of the classes are consistent and present in activity, class, statechart diagram.

Similarly, MCF can be calculated as given below-

Class	**Method**	**MD AC-CL**	**MD ST-CL**	**Type**	**MCF CM / (UM + CM + HM)**
Interface	isMemberValid	1	0	UM	0
	isBookValid	1	0	UM	
	IsBookAvl	1	0	UM	
Member	getMemberDetails	0	0	CM	3/ (3+0+0) =1
	getMaxIssueLimit	0	0	CM	
	IssueBook	0	0	CM	
Book	getBookDetails	0	-	CM	4 / (4+0+0)= 1
	getStockBal	0	-	CM	
	Issue	0	0	CM	
	reissue	0	-	CM	

LibTrans	getMemberTrans	0	0	CM	1
	addTrans	0	0	CM	
Book_details	getStatus	-1	-	HM	0
	setStatus	-1	-	HM	
MCFav					3/5=0.6

Table 12: Calculation of MCF for LMS

Thus overall Design Compliance Metrics for "Issue Book" use case,

DCM = (ECF + CCF + MCFav)/3

= (0.94+0.6+0.6)/3 = .713

This value of DCM indicates that the level of consistency of implementation of "Issue Book" use case is 71.3%.

7. CONCLUSION

The adoption of UML as a standard for modeling design specifications of object-oriented systems has made modeling simpler and easy to understand with lots of tools supporting it. However, UML being a visual language, is semi-formal in nature and hence verification of design in UML has triggered challenging opportunities of research in this domain. In this paper we present a metrics based analysis of requirements. We propose a new set of metrics based on UML models namely –Design Compliance Metrics II (DCM - II), which is an extension of our earlier work to measure extent of coverage of a requirement in design using class and sequence diagram. DCM-II is a unique method for studying consistency between activity, statechart and class diagrams of UML implementing the events of use cases of use case diagram by providing quantitative feedback on the level of consistency in design at any point of time. In this paper we have considered only use case, activity, statechart and class diagrams and in our future work we intend to extend this concept further and fine-tune the metrics by including other commonly used UML diagrams. Application of these metrics on various case studies would enable us doing a comparative analysis of the consistency among analysis and design models and take appropriate actions.

REFRENCES

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

[2] Alexander Egyed, "Scalable Consistency Checking Between Diagrams-The ViewIntegra Approach", 16th IEEE International Conference on Automated Software Engineering (ASE'01**)**, **,** San Diego, California, pp 387, November 26 - 29, 2001.

[3] Martin Wolf, Evgeni Ivanov, Rainer Burkhardt and Ilka Philippow, "UML Tool Support: Utilization of Object-Oriented Models", Technology of Object-Oriented Languages and Systems (TOOLS 34'00), Santa Barbara, California, pp 529, July 30 – August 3, 2000.

[4] Hazem Hamed and Ashraf Salem, "UML-L: An UML Based Design Description Language", ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'01), Beirut, Lebanon, pp 0438, June 25 - 29, 2001.

[5] Andrea Zisman and Alexander Kozlenkov, "Knowledge Base Approach to Consistency Management of UML Specifications", 16th IEEE International Conference on Automated Software Engineering (ASE'01), San Diego, California, pp 359, November 26 - 29, 2001.

[6] Jing Liu, Zhiming Liu, Jifeng He and Xiaoshan Li, "Linking UML Models of Design and Requirement", 2004 Australian Software Engineering Conference (ASWEC'04) Melbourne, Australia, pp 329, April 13 - 16, 2004.

[7] Soon-Kyeong Kim and David Carrington, "A Formal Object-Oriented Approach to defining Consistency Constraints for UML Models", 2004 Australian Software Engineering Conference (ASWEC'04), Melbourne, Australia,pp 87, April 13 - 16, 2004.

[8] Boris Litvak, Shmuel Tyszberowicz and Amiram Yehudai, "Behavioral Consistency Validation of UML Diagrams", First International Conference on Software Engineering and Formal Methods (SEFM'03), Brisbane, Australia, pp 118, September 22 - 27, 2003.

[9] Pascal André, Annya Romanczuk and Jean-Claude Royer, "Checking Consistency of UML Class Diagrams using Larch Prover", Electronic Workshops in Computing (eWiC), Rigorous object-oriented Methods, York, UK, 17th Jan, 2000.

[10] Jochen M. Küster and Jan Stehr, "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.

[11] Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel and Stefan Sauer, "Testing the Consistency of Dynamic UML diagrams", Sixth International conference on Integrated Design and Process Technology, Pasadena, California, June 23-28, 2002.

[12] Tom Mens, Ragnhild, Van der Straeten and Jocelyn Simmonds, "Maintaining Consistency between UML Models with Description Logic Tools, Fourth International Workshop on Object-oriented Reengineering (WOOR2003), Darmstadt, Germany, July 21, 2003.

[13] Alexander F. Egyed, "Automatically Validating Model Consistency during Refinement", 23rd International Conference on Software Engineering (ICSE 2001), Toronto, Ontario, Canada, May 12-19, 2001.

[14] Franck Xia and Gautam S. Kane, "Defining 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'2003), National Taiwan University, December 10-13, 2003.

[15] Aliki Tsiolakis and Hartmut Ehrig, "Consistency Analysis of UML Class and Sequence Diagrams using Attributed Graph Grammars", Proc. Of Joint APPLIGRAPH and GETGRATS workshop on graph transformation systems, pp 77-86, March 25, 2000.

[16] OMG standard, XMI Specification 2.0, http://www.omg.org

[17] Hyoseob Kim and Cornelia Boldyreff, "Developing Software Metrics Applicable to UML Models", Proceedings of 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, June 11th, 2002.

[18] Holly Parsons-Hann, Kecheng Liu, "Measuring Requirements Complexity to Increase the Probability of Project Success", Proceedings of 7th Intl. Conf. On Enterprise Information Systems (ICEIS), Miami, Florida, USA, Vol-III, pp 434-438, May 24-28, 2005.

[19] Hana Chockler , Orna Kupferman , Robert P. Kurshan ,

Moshe Y. Vardi, "A Practical Approach to Coverage in Model Checking", Proc. of 13th Intl. Conf. on Computer Aided Verification, p.66-78, July 18-22, 2001.

[20] Ahmed Hassan, Walid Rabie Abdel Moez, and Hany H. Ammar , "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.

[21] Chidamber, S, R, Kemerer, C, F, "A Metrics Suite for Object-Oriented Design", In IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 476-493, 1994.

[22] Paul Ammann, Paul E. Black, A Specification-Based Coverage Metric to Evaluate Test Sets, Proc. of 4th IEEE Intl. Symposium on High-Assurance Systems Engineering, p.239-248, Nov 17-19, 1999.

[23] Michael W. Whalen, Ajitha Rajan, Mats P.E. Heimdahl, Steven P. Miller, "Coverage metrics for requirements-based testing**",** Proc. of the 2006 Intl. symposium on Software testing and analysis, ISSTA'06, Portland, Maine, USA, pp 25-36.

[24] Ananya Kanjilal, Goutam Kanjilal and Swapan Bhattacharya, "Metrics based analysis of requirements for object-oriented systems: An empirical approach", Infocomp Journal of Computer Science, Vol. 7, No. 2, page 26-35, 2008.