International Journal on Future Revolution in Computer Science & Communication Engineering
Volume: 4 Issue: 3
ISSN: 2454-4248
195 – 200

Optimal Approach to Compute Metrics for Structural and Behavioural Diagrams of UML using Program Slicing Techniques

Er. Daljeet Singh
Department of Computer Science and Engineering
Guru Nanak Dev Engineering College
Ludhiana, Punjab, India
E-mail:diljitsingh007@gmail.com

Dr. Harmaninder Jit Singh Sidhu
Department of Computer Applications
Desh Bhagat University
Mandi Gobindghar, Punjab, India
E-mail:jeetsinder@gmail.com

***Abstract*—** We have purposed an optimal approach for computing the various complexity metrics for different UML diagrams by using the program slicing techniques. Firstly, we draw the UML diagrams in Argo UML software then XML file is generated then by using the SD Metrics Tool different parameters were calculated automatically. The dependency graph is drawn then the slicing criteria is adopted by using our purposed algorithm then complexity metrics were calculated. After applying the program slicing techniques the complexity of any diagram will be decreased and will easier for testability, maintainability, readability, and modularity.

Keywords*-UML diagrams;metrics;dependency graph;program slicing;XML*

*****

I. INTRODUCTION TO UML DIAGRAMS

A Unified Modeling Language (UML) diagram is a graphical representation of a model of a system which partially signifies the design and implementation. UML diagrams contain the elements: - UML nodes that are connected with edges and also known as paths. The UML model might also contain other documentation like use cases. The diagram is defined by the graphical symbols represented on the diagram. A diagram where the primary symbols in the contents area are classes is a class diagram. A diagram which shows use cases and actors is use case diagram. A diagram shows the sequence of message exchanges between objects is called sequence diagram. UML specification classified different kinds of diagrams, e.g. the combination of structural and behavioural elements helps to show a state machine nested inside a use case diagram. At the same time, some UML Tools do restrict set of available graphical elements which could be used when working on a specific type of diagram. Consequently, the boundaries between the varieties of diagrams are not strictly enforced.

Structural diagrams show the static structure of the system and its parts on different abstraction and implementation levels and show us how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system and may include implementation concepts, abstract and real world. Structure diagrams are not utilizing time-related concepts it does not show the details of the dynamic behaviour of the circumstances. However, they may show relationships to the behaviours of the classifiers permitted in the structure diagrams.

The class diagram is a static structure diagram which represents the structure of a system at the level of classifiers. The UML diagrams could be categorized hierarchically as shown in figure 1, Classification of UML diagrams. Some classifiers of the system, components or subsystem, different relationships between their attributes, classifiers, operations and constraints.

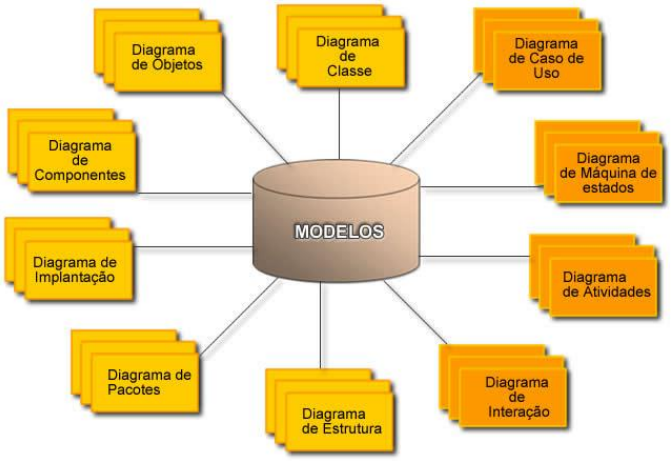

Figure 1 Classification of UML diagrams.

Behavioural diagrams show the dynamic behaviour of the objects in a system, which is used to represent as a series of changes to the system over the time. Use cases diagrams and Behavior diagrams are used to describe a set of use cases or actions that some system should perform in collaboration with one or more external users of the system or to provide some observable and valuable results to the actors. A number of changes to the system over the time implemented according to the need.

II. RELATED WORK

In this section, we briefly review the related work on the importance of UML metrics. Most of our work reported in metrics importance, technique and need for the UML structural and behavioural diagrams of UML. Garg Sushil et al. [15] purposed the aspect-oriented programming which is a new paradigm for improving the system's features such as modularity, readability and maintainability. Aspect-oriented software development is a new technique to support the concerns in software development. Coupling is an internal software attribute that can be used to indicate the degree of interdependence among the components of a software system.

 ISSN: 2454-4248
195 – 200

R. Mall et al. [16] propose a technique for static slicing of UML models. First, transform software architecture specified using UML into an intermediate representation named Model Dependency Graph (MDG). Model Dependency Graph (MDG) combines information of sequence diagrams along with the relevant information of the model. For a given slicing criterion, slicing algorithm traverses the constructed MDG to identify the relevant model elements. Algorithm's novelty lies in its computing a slice-based UML model. Sikka Preeti et al. [17] program slicing is proved in breaking down the large program into the small relevant parts that is needed as per the specified criteria. The objective of this paper is the betterment of existing program slicing techniques. Method / statistical analysis the methodology is proposed to start slicing the software from designing the various levels of the model and continue it with source code level. Verma Pretty et al. [12] for measuring the software, appropriate metrics are needed. To attain the various qualitative and quantitative aspects of software. To measure the software in terms of quality, size, efforts, efficiency, and reliability, performance etc. there are different metrics available in software engineering and it has been an area of interest for the various researchers. Measures of specific attributes of the process, project and product are used to compute Software metrics. This work proposes a similar approach of measuring software using various UML diagrams and applied Software size metric to evaluate the size of the Software.

All of the above authors and number of other authors had proposed and implemented the techniques for calculating the metrics for one of UML diagram but we had purposed as well as implemented the best optimal technique other than the above authors for structural as well as behavioral diagrams to reduce the testability as well as maintainability of the software.

III. UML METRICS

There are different types of UML Metrics used for computations for calculating selected qualities, complexities and sizes as described below:-

A. UML Design Complexity Metrics
B. UML Design Quantity Metrics
C. UML Design Size Metrics

A. UML Design Complexity Metrics

Design complexity metrics are computations for calculating selected complexities. Complexity is defined as the relation of entities to relationships. The size of a set is determined by the number elements in that set. The complexity of a set is a question of the number of relationships between the elements of that set. The more connections or dependencies there are relative to the number of elements, the greater the complexity. Various types of UML Design Complexities are:

a) Object Interaction Complexity
b) Class Hierarchical Complexity
c) Class Data Complexity
d) Class Functional Complexity
e) Object State Complexity
f) State Transition Complexity
g) Activity Control Flow Complexity
h) Use Case Complexity
i) Actor Interaction Complexity
j) Overall Design Complexity

The complexity of a single entity is determined by the number of sub-entities relative to the number of relationships between those sub-entities.

B. UML Design Quality Metrics

The Design Quantity Metrics counts of the diagram and model the types contained in the UML model. The model types are further subdivided into two counts:-

- The Design Entity Counts.
- The Design Relationship Counts.

The design diagram type counts are: Use case diagrams, activity diagrams, class diagrams, sequence diagrams, interaction diagrams, state diagrams, component diagrams, distribution diagrams. Various types of UML design quality metrics are:

a) Degree of Class Coupling
b) Degree of Class Cohesion
c) Degree of Modularity
d) Degree of Portability
e) Degree of Reusability
f) Degree of Testability
g) Degree of Conformity
h) Degree of Consistency
i) Degree of Completeness
j) Degree of Compliance

The design entity type counts are: Sub-systems, use cases, actors, components, interfaces, classes, base/superclasses, methods, parameters, attributes, activities, objects, states, rules, stereotypes, design entities, and design entities referenced.

The design relationship type counts are Usages, associations, generalizations, interactions, class hierarchy levels, method invocations, activity flows, state transitions, and test cases. The design quality metrics are computations for calculating selected qualities. The closer the model is to fulfilling that standard, the higher is its quality.

Quality is defined here as the relation of that state the model. Quality measurement presupposes a standard for the UML model. The actual state of the model is then compared with that standard. In German the overall design quality can be simply expressed by the ratio: the upper bound of the metric is 1. If the IST exceeds the SOLL then the quality goal has been surpassed. A quotient coefficient of 0.5 indicates median quality. It should be remembered that quality is relative. However, in comparison with the ratio derived from another design in exactly the same way, it indicates that the one design has a better or lower quality than the other, at least in respect to the quality characteristic measured. Since there is no absolute quality scale, the quality of a system design can only be assessed in relation to the quality.

ISSN: 2454-4248
195 – 200

C. *UML Design Size Metrics*

The design size metrics are computed values for representing the size of a system. Of course what is being measured here is not the system itself, but a model of the system. The system itself will only be measurable when it is finished. Those size measures can be derived from the requirements by analyzing the required texts or at design time by analyzing the design diagrams. Both measurements can, of course, be only as good as the requirements and/or the design is measured. That means the original cost estimation has to be based on the requirements. One needs size measures at an early stage in order to predict the effort that will be required to produce and test a system. Since the design is more detailed and more likely to be complete, the design size metrics will lead to a more reliable estimate. However, the design is complete much later than the requirements. If the design based estimation surpasses the original one, it will be necessary to delete functionality, i.e. to leave out less important use cases and objects. If the design based estimation varies significantly from the original one, it will be necessary to stop the project and to renegotiate the proposed time and costs. In any case, the project should be recalculated when the design is finished. When estimating a project one should always estimate with at least three different methods. For that reason, five measures are taken to give the estimator a choice. The five size measurements taken are:

a) Data-Points
b) Function-Points
c) Object-Points
d) Use Case-Points
e) Test-Cases

IV. EXAMPLE FOR DESIGNING XML SCHEMAS USING UML DIAGRAM

Unified Modeling Language (UML) is an industry that is used in modelling business concepts when building the software systems in an object-oriented manner. XML schemas, constrain the nature of XML exchanged.XML has gained ground in becoming a key for these systems in terms of sending the information and commands.

In Figure 2, BALTIC Shipping which is a shipping company that transports shipments from the one country to other. It needs to create a system for tracking shipments from its head office from one country to its regional offices which were in other countries. All the orders and confirmation data is exchanged in XML format and schemas have to be designed to outline the structure of the documents for all the orders for the company. The business used to model orders is also used to exchange information with the "Inventory Tracking System" which knows which packages the company is holding for delivery.

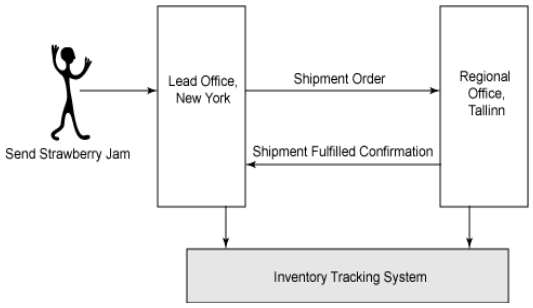

Figure 2 BALTIC Shipping workflow

A. UML and its object-oriented modelling can be used to building XML schemas. The business supervisor of BALTIC Shipping comes and asks to model the XML schema that will formalize the information that is transmitted between different systems in the BALTIC Shipping company.

In the "UML diagram", the business of a Shipping Order is represented in figure 3. In addition, the UML diagram is used to represent what constitutes an origin place or an order place. Origin place and destination place types are shown to be the same as type address of the client, and BALTIC Shipping stores an address in its database with the following fields:

- Name
- Street
- City
- Country

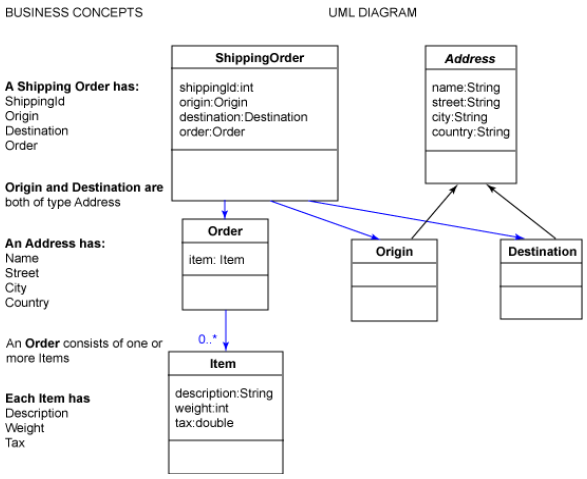

Figure 3 UML diagram

BALTIC Shipping Company defines a Shipping Order that must consist of a

- Shipping Id
- Origin
- Destination
- **Order**

It considers this information whenever any data regarding a Shipping Order is exchanged with one another. These are business techniques and they have been used in their database models, in their software programs, and in their documents

 ISSN: 2454-4248
195 – 200

that are read by supervisor and business team members. The schemas represent the mapping of the UML diagram to XML schemas are as follows:-

XML CODE1: SHIPPINGORDER.XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
         <xs:element name="shippingId"type="int"/>
          <xs:element name="origin" type="Origin"/>
            <xs:element                    name="destination"
type="Destination"/>
             <xs:element name="order" type="Order"/>
                  <xs:include
schemaLocation="DataTypes.xsd"/>
                     <xs:element name="shippingOrder">
                 <xs:complexType>
          <xs:sequence>
          </xs:sequence>
        </xs:complexType>
     </xs:element>
</xs:schema>
```

The Shipping Order class in UML is assembled and resented by the complex type "shipping order", in the schema. According to the business requirements, a Shipping Order consists of a

- Shipping Id
- Origin
- Destination
- Order

One thing to note is that input the `Origin` type along with other generic types in a Data Types schema in XML code 2. A Data Types are convenient for storing reusable types, such as the definition of an Address, which are used in XML documents.

In the UML diagram, "`Address`" is an abstract type as indicated. The types

- Name
- Street
- City
- Country

XML CODE 2. DATATYPES.XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
<xs:complexType name="Destination">
<xs:complexContent>
<xs:extension base="Address"/>
 </xs:complexContent>
 </xs:complexType>
<xs:complexType name="Order">
<xs:seque
<xs:element                name="item"                type="Item"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Item">
  attributeFormDefault="unqualified">
 <xs:complexType name="Address" abstract="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
          <xs:sequence>
        <xs:element name="description" type="xs:string"/>
        <xs:element name="weight" type="xs:double"/>
        <xs:element name="tax" type="xs:double"/>
```

If we had started designing the XML schema from starting, it would have been difficult to just write down all the object types in XML. In addition, it would be nearly difficult and impossible to explain them to the business supervisor who is not familiar with the terminology of XML code. The UML diagrams, you could translate the business concepts of the company and then create the XML schemas with this representation. The instance document for a parcel that is shipped from one country to another is given the following schemas.

XML CODE 3. SHIPPINGORDER.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<shippingOrder
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\schemas\ShippingOrde
r.xsd">
destination>
                    <name>Mai Madar</name>
                    <street>Liivalaia 33</street>
                    <city>Tallinn</city>
                    <country>Estonia</country>
                    </destination>
                    <order>
                    <item>
                    <description>Ten    Strawberry    Jam
bottles</description>
                    <weight>3.141</weight>
                    <tax>7.60</tax>
                    </item>
                    <shippingId>09887</shippingId>
                    <origin>
                    <name>Ayesha Malik</name>
                    <street>100 Wall Street</street>
                    <city>New York</city>
                    <country>USA</country>
                    </origin>
                    </shippingOrder>
```

XML schemas provide a great deal of richness in formalizing XML documents and a preliminary analysis shows that most of the fundamental functionalities of XML schemas can be represented by UML diagrams.

 ISSN: 2454-4248
195 – 200

V. PROGRAM SLICING

Program slicing is the computation of the set of programs or program statements, the program slice that may affect the values at some point of interest, referred to as a slicing of that program criterion. Program slicing can be used in debugging to locate the source of bugs more easily as compared to the find the bugs in full program. Other applications of slicing include software optimization, maintenance and program analysis. There are two types of Program slicing:-

a) Static slicing
b) Dynamic Slicing

(a) Static Slicing

Static slicing is based on the original definition of Weiser, formally, a "static program slice" S consists of all statements in program "p" that may affect the value of variable "v" at some point "p".

(b) Dynamic Slicing

To make the use of information about a particular execution of a program. A dynamic slice contains all statements that actually affect the value "v" of a variable at a program point "p" for a particular execution of the program rather than all statements that may have affected the value "v" of a variable at a program point "p" for any arbitrary execution of the program.

VI. THE UML CLASS DIAGRAM

The UML class diagram as an example of the Hasp Software with various attributes and functions in which representation of main part of the building block.

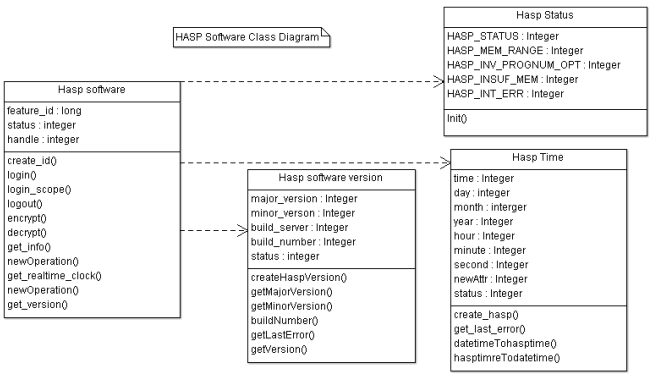

Figure 4 Class diagram of UML for Hasp Software

In figure 4, the class diagram of Hasp Software is used for both generals as well as conceptual modelling for the development of the applications. In this example, the class diagram for the Hasp Software is represented with the required attributes and functions with their associations.

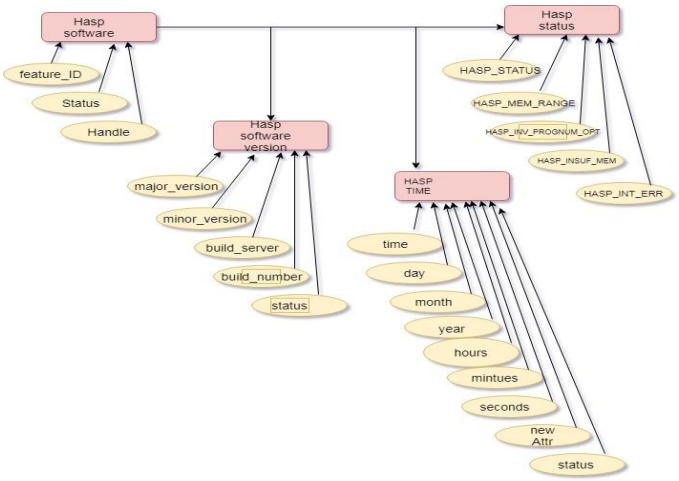

Figure 5 Class Dependency Graph (CDG)

In figure 5 Class Dependency Graph (CDG), classes and their attributes, methods and their call parameters, together with method return values are represented as different types of nodes. The data dependencies arise when the class methods, its parameters and return values directly or indirectly make use of the class attributes.

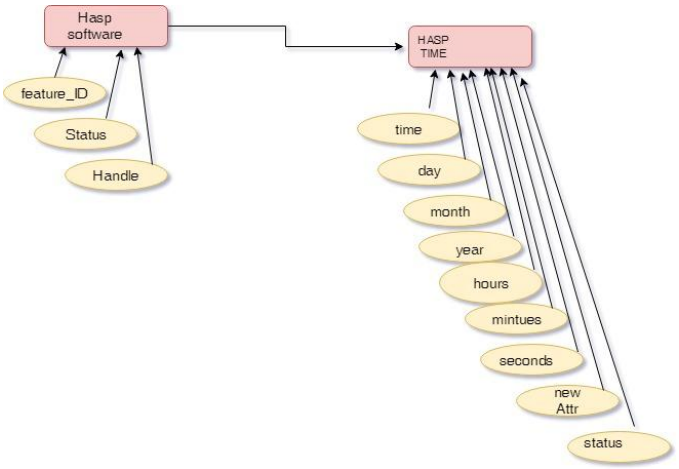

Figure 6 Slicing Graph of Class Dependency Graph (CDG)

In figure 6, Slicing Graph of Class Dependency Graph (CDG) of Hasp Software is generated according to the requirement, in which customer and teller class are taken as a sliced criterion. At the same time, Class Dependency Graph (CDG) of Hasp Software represents class relations in Slice-based cohesion metrics. Worked examples, illustrating the calculation of slice-based cohesion metrics for a class dependency graph (CDG) of Hasp Software, Degree of Cohesion (DCH), Number of Attributes Used (NAU), Total Number of Attributes (TNA), Degree of Cohesion (DCH) = NAU/TNA

For computing the Degree of Cohesion (DCH), the connectivity between the classes and the number of attributes used by the method of a class. In figure 6, "Slicing Graph of Class Dependency Graph (CDG)" where two classes were taken as sliced criteria. The cohesion is interaction within a class and coupling is the interaction with the other classes.

 ISSN: 2454-4248
195 – 200

Always this high cohesion and low coupling are recommended in the software to develop successful software. In table 1 the results for the degree of cohesion after the slicing is displayed.

Table 1 Degree of cohesion

Class name	NAU	TNA	DCH
Hasp time	0	9	**0**
Hasp Software	0	4	**0**

The degree of coupling in table 2, the ratio of a number of messages received to the number of the message passed. For finding the degree of coupling, message received coupling (MRC) and the message passed coupling (MPC) is used, it is the number of messages received and passed by a class. (MRC) Message received coupling is the complexity of message received by the classes, as MRC is the number of messages received by a class from the other classes. (MPC) Message Passed Coupling is denied as the number of the message passed among objects of the classes. The degree of coupling is given
Degree of coupling (DC) = MRC/ MPC.

Table 2 Degree of coupling

Class name	MRC	MPC	DC
Hasp time	0	3	**0**
Hasp Software	1	0	**1**

U=
((Nr_Classes * Minimun_Nr_Attributes)/(Nr_Class_Attributes))
(2)

Class Data Complexity for customer and teller = 2*3/13 = **0.46**

VII. FUTURE WORK

The Metrics are used for identifying the design of the software to be developed for improving the system features such as readability, testability, modularity and maintainability. We have proposed a technique to decreasing the complexity of the software by using the program slicing techniques. We calculated with the effective utilization of optimized algorithm implemented in programs slicing techniques. This work can be extended as automation for the whole process by building a tool.

REFERENCES

[1] Kumar. S.V and Santosh, "Impact of coupling and cohesion in object-oriented technology," Journal of software engineering and applications, vol. 5, pp. 671-676, 2012.

[2] Kambow. Lavleen and Singh. Daljeet, "Visualizing the software metrics of state chart diagram using program slicing," International journal of applied information system (IJAIS), ISSN: 2249-0868, foundation of computer science, New York, USA, Vol. 2, 2013, pp. 9.

[3] Rani. Tincy, Sanyal. Manish and Garg. Sushil, "Measuring Software Design Class Metrics:- A Tool Approach," International journal of engineering research & technology (IJERT), ISSN: 2278-0181, vol. 1, Issue 7, September 2012.

[4] Virtual Machinery, "Object-Oriented Software Metrics - Introduction and overview", Virtual Machinery, Link: http://www.virtualmachinery.com/jhawkmetrics.htm

[5] Kumar. Akhilesh and Khalsa. sunnit kaur, " Determining cohesion and coupling for class diagram through slicing techniques", IJACE, Vol. 4, No.1, Jan-June 2012,pp. 19-24.

[6] Singh. Daljeet and Kamra. Amit, " Measuring Software design metrics of UML Struictural and Behaviourl Diagrms," Internationl Journal of computer & Mathematicl Sciences (IJCMS), ISSN : 2347-8527, Vol. 6, Issue.5, May 2017.

[7] Genero. M.," Defining and validating metrics for conceptual models," [Ph.D. thesis]. University of Castilla-La Mancha, 2002.

[8] Weyuker, "Evaluating software complexity measures," IEEE Transactions on Software Engineering, 14(9),1998, pp.1357-1365.

[9] Chidamber Shyam, "A metrics suite for object-oriented design", IEEE Transactions on Software Engineering, June, 1994.

[10] Seyyed. Mohsen, "Object-Oriented Metrics", Sharif University of Technology, International journal of science and research, Department of Computer Engineering, January 2006.

[11] Mythili. Thirugnanam, "Quality Metrics Tool for Object-Oriented Programming", International Journal of computer theory and engineering, vol. 2, No. 5, October 2010.

[12] Verma. Pretty, "Effect of different UML diagrams to evaluate the size metrics for different software projects", Global Journal of computer science and technology software and engineering, vol. 15, issue. 8, version 1.0, February. 2015.

[13] Ana. Nicolaescu, "Evolution of Object-Oriented Coupling Metrics: A Sampling of 25 Years of Research," RWTH Aachen Univ., Aachen, Germany Horst Lichter ; Yi Xu, May 2015, pp. 16-18.

[14] Alshammari, "A Hierarchical Security Assessment Model for Object-Oriented Programs," Fac. of Science & Technol., Queensland Univ. of Technol., Brisbane, QLD, Australia, Colin Fidge, Diane Corney, July 2011, pp. 13-14.

[15] Garg. Sushil, Kahlon K. S. and Bansal P. K, "How to Measure Coupling in AOP from UML Diagram" International Journal of Computer Science and Telecommunications, Volume 2, Issue 8, November 2011.

[16] Jaiprakash. T. Lallchandani, R. Mall: Static Slicing of UML Architectural Models, in Journal of Object Technology, vol. 8, no. 1, January– February 2009, pp. 159–188.

[17] Sikka. Preeti and Kaur. Kulwant, "Mingling of Program Slicing to Designing Phase" Indian Journal of Science and Technology, Vol 9(44), DOI: 10.17485/ijst/2016/v9i44/105091, November 2016.