Volume 8, No. 5, May-June 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

ISSN No. 0976-5697

Dynamic Software Metric Estimation (DSME): Tool using ArgoUML

P.L. Powar
Department of Mathematics and Computer Science
R. D. University, Jabalpur, India

M.P. Singh
Department of Computer Science
Dr. B. R. Ambedkar University, Agra, India

Samar Upadhyay
Department of Computer Application
JEC, Jabalpur, India

Bharat Solanki
Department of Mathematics and Computer Science
R. D. University, Jabalpur, India

Abstract: Software cost estimation had been a challenge for the researchers. Due to various technologies and further researches in software development, the field of cost estimation gained an enormous scope for studies. Moreover, the growth of internet based technology and distribution made this problem quite popular. Component based software development strategies have been found to be advantages for software development companies. Cost estimation using static metric has been found to be helpful in pre decisions whereas dynamic metrics are helping for estimating cost of maintenance, system loads and suggests the improvement if required in the technology. The present paper proposes and provides an implementation of the DSME tool for evaluating the software metrics using dynamic metrics. For estimation of dynamic metrics current focus is on time sequence diagram processed using ArgoUML software tool.

Keywords: Software Metrics, Static and dynamic Metrics, Component-based software systems.

1. INTRODUCTION

Implementation methodology of software has been changing in every decade or in few years. The revolutions in software and hardware engineering and devices have imposed the need for same. In last few years Component Based Software Engineering (CBSE) has been adapted in the industry. CBSE is a process that emphasizes the design and construction of computer based systems using reusable software components. It provides the way of developing very large software systems. Component based software engineering has been widely accepted as a new and latest approach to software development. Today's the software systems are very difficult, bulky and unmanageable. This causes in lesser productivity, higher risk management and meagre software quality.

Software metrics measure different aspects of software complexity and therefore play a key role in analyzing and improving the quality of software. Metrics provide important information on external quality aspects of software such as its maintainability, reusability and reliability. These metrics are helpful in achieving the quality and in managing risk in the component based system by checking the factors that affect risk and quality.

In CBSE, component is an independent and replaceable part of a system that performs a clear function in the context of a well defined architecture. It results in better productivity, improved quality, reduction in time spent and cost to develop. Metrics used in component based software engineering are helpful in achieving the quality and managing risk in component based system by checking the factors that affect risk and quality. Metrics help the developer in identifying the probable risks so that proper corrective action can be taken. Various metrics have been proposed to measure the different attributes of a component like functionality, interactivity, complexity, reusability etc. Figure 1.1 shows the CBSE Metrics for Software Component.

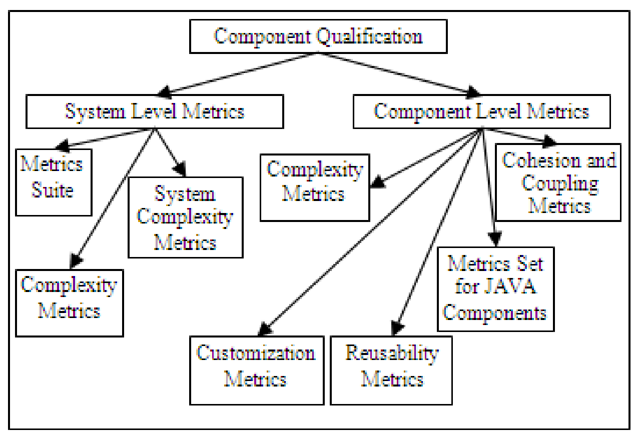

Figure 1.1: Software Component Measurement Metrics

The paper is organized as follows: Section 1 describes the available technology and methods in processing CBSE and their utilization. Section 2 covers the details of the various metrics used in software engineering especially in component based software. In order to use metrics for processing of CBSE, we have referred the work of Narasimhan, Parthasarathy, Das [8] and Narasimhan, Hendradjaya [9]. Validation of metrics using Weyuker Properties have been discussed in section 3. Existing work by various researchers have been discussed in Section 4. Research problem of the present paper has been discussed in Section 5. Section 6 provides the proposed methodology. Section 7 provides application of DSME tool. The discussions on the results are given in section 8. Conclusion of the work has been discussed in Section 9.

2. SOFTWARE METRICS

In general metrics are functions which are evaluated to generate some measurement or degree by which any software possesses some property. A software metric has goal in obtaining objective, reproducible and quantifiable measurements, which may be used in quantifiable assessment of various valuable applications in schedule and budget planning, cost estimation, software debugging, software performance optimization, quality assurance testing, and optimal personnel task assignments. Recently, many researcher have used different software metrics and the metrics have been upgraded and extended in accordance with the need of software user. In software engineering software metrics have been defined and clustered according to the process, product, quality specification, software design, software architectures, software complexity, code metrics, testing metrics etc.

By static metric, one meant to the metrics which can be evaluated before software implementation and execution, whereas the metrics which are evaluated during the execution of the software are dynamic metrics.

In CBSE various metrics have been proposed using the graph connectivity as a medium to represent a system of integrated components.

Metrics may play an important role in quality assurance, especially in the acquisition of components and in deciding whether they should be used or not. Metrics should provide a basis for deciding whether reuse is sensible, whether it is cost effective to adapt existing component or build a component from scratch. In short, metric which address cost savings on component basis are needed. Metrics can see as part of the topics acquisition and usage.

In this section, we describe some software metric which are quite popular from implementation point of view.

- Object Oriented Metrics: Object-oriented measurements are being used to evaluate and predict the quality of software. A growing body of empirical results supports the theoretical validity of these metrics. The validation of these metrics requires convincingly demonstrating that (1) the metric measures what it purports to measure (for example, a coupling metric really measures coupling) and (2) the metric is associated with an important external metric, such as reliability, maintainability and fault-proneness. Often these metrics have been used as an early indicator of these externally visible attributes, because the externally visible attributes could not be measures until too late in the software development process.(See Table 2.1)
- Reusability metrics: The reusability assets are different in different contexts. However, there are some characteristics that generally contribute to the reusability of assets. Although many of these characteristics apply to assets in general, we focus in this section on components as assets. (see table 2.2)
- Direct Metrics: We need a set of direct metrics (i.e., metrics computed directly from the source code) to describe a system in simple, absolute terms. The metrics describing the size and complexity are probably some of the simplest and widely used metrics. They count the most significant modularity units of an object-oriented system, from the highest level (i.e., packages or namespaces), down to the there is one metric in the overview pyramid that measures it. The metrics are placed one per line in a top-down manner. (see Table 2.3)
- Static and dynamic metrics : Narasimhan, Parthasarathy, Das [8] and Narasimhan, Hendradjaya [9] has defined two suites of metrics, which cover static and dynamic aspects of component assembly. The static metrics measure complexity and criticality of component assembly, wherein complexity is measured using Component Packing Density and Component Interaction Density metrics. Further, four criticality conditions namely, Link, Bridge, Inheritance and Size criticalities have been identified and quantified. The complexity and criticality metrics are combined to form a Triangular Metric, which can be used to classify the type and nature of applications. Dynamic metrics are collected during the runtime of a complete application. Dynamic metrics are useful to identify super-component and to evaluate the degree of utilization of various components. In this paper both static and dynamic metrics are evaluated using Weyuker's set of properties. (cf. Table 2.4)

Table 2.1 : Object Oriented Metrics

Metric		**Object oriented Feature**	**Measurement Method**	**Concept**	**Interpretation**
CC	Cyclomatic complexity	Method	Algorithmic test paths	Complexity	Low => decisions deferred through message passing Low not necessarily less complex
SIZE	Lines of code	Method	Physical lines , statements , and/or comments	Complexity	Should be small
COM	Comment percentage	Method	Components divided by the	Usability Reusability	20 to 30 %

			total line countless blank lines		
WMC	Weighted methods per class	Class/ method	1)Methods implemented within a class 2)Sum of complexity of methods	Complexity Usability Reusability	Larger => greater complexity and decreased understandability ; testing and debugging more complicated
LCOM	Lack of cohesion of methods	Class/ Cohesion	Similarity of methods within a class by attributes	Design Reusability	High=> good class subdivision Low=> Increased complexity – subdivide
CBO	Coupling between Objects	Coupling	Distinct non inherited related classes inherited	Design Reusability	High=> poor design , difficult to understand , decreased reuse , increased maintenance
DIT	Depth of Inheritance tree	Inheritance	Maximum length from class node to root	Reusability Understandability Testability	Higher=> more complex , more reuse
NOC	Number of children	Inheritance	Immediate Subclass	Design	Higher=> more reuse ; poor design increasing testing

Table 2.2 : Reusability Metrics

Metric	Definition
Reuse level (RP)	Ratio of the number of reused lines of code to the total number of lines of code Reuse Level (RL) Ratio of the number of reused items to the total number of items.
Reuse Frequency(RF)	Ratio of the references to reused items to the total number of references
Reuse size & Frequency(RSF)	Similar to Reuse Frequency , but also considers the size of items in the number of lines of code
Reuse Ratio(RR)	Similar to Reuse percent, but also considers partially changed items as reused .
Reuse Density	Ratio of the number of reused parts to the total number of lines of code

Table2.3 : Direct Metrics

Metric	Definition
NOP	Number of Packages, i.e., the number of highlevel packaging mechanisms, e.g., packages in Java, namespaces in C++, etc.
NOC	Number of Classes, i.e., the number of classes defined in the system, not counting library classes.
NOM	Number of Operations, l i. e., the total number of user defined operations within the system, including both methods and global functions (in programming languages that allow such constructs).
LOC	Lines of Code, i. e., the lines of all user-defined operations. In the Overview Pyramid only the code lines containing functionality (i.e., lines of code belonging to methods) are counted.
CYCLO	Cyclomatic Number, i.e., the total number of possible program paths summed from all the operations in the system. It is the sum of McCabe‘s Cyclomatic number for all operations.
CALLS	Number of Operation Calls, i. e., this metric counts the total number of distinct operation calls (invocations) in the project, by summing the number of operations called by all the user-defined operations. If an operation fo () is called three times by a method f1() it will be counted only once. If it is called by methods f1(), f2() and f3(), three calls will be counted for this metric.
FANOUT	Number of Called Classes, this is computed as a sum of the FANOUT metric (i.e., classes from which operations call methods) for all user defined operations. This metric provides raw information about how dispersed operation calls are in classes.
System coupling	computed proportions. Again, the numbers above describe the total coupling amount of a system, but it is difficult to use those numbers to characterize a system with respect to coupling. We can compute, using the number of operations (NOM), two proportions that better characterize the coupling of a system.
Coupling intensity (CALLS/ Operation)	This proportion denotes the level of collaboration (coupling) between the operations, i .e., how many other operations are called on average from each operation. Very high values suggest that there is excessive coupling among operations, i .e., a sign that the calling operation does not.

Table 2.4 : Dynamic metrics

NAME	FORMULAE	DESCRIPTION
Number of Cycle (NC)	NC = # cycles	Where, #cycles is the number of cycles within the graph
Average Number of Active Components	$ACD = \frac{\#activecomponent}{T_e}$	#activecomponents is the number of active component and T_e is time to execute the application (in seconds)
Active Component Density (ACD)	$ACD = \frac{\#activecomponent}{\#components}$	#activecomponent is the number of active components and #component is the number of available components.
Average Active Component Density	$AACD = \frac{\Sigma_n ACD_n}{T_e}$	$\Sigma_n ACD_n$ is the sum of ACD and T_e is time to execute the application (in seconds). Execution time can be any of execution of a function, between functions or execution of the entire program.
Peak Number of Active Components	$AC_{\Delta t}$ = max { AC_1,..,AC_n}	$\#AC_n$ is the number of active component at time n and Δt is the time interval in seconds.

3. VALIDATING THE METRICS USING WEYUKER PROPERTIES

Weyuker has proposed an axiomatic framework for evaluating complexity measures [14]. The properties are not without critique and these have been discussed in [3] and [4] by Fenton, Pfleeger and Henderson-sellers. The properties, however, have been used to validate the C-K metrics by Chidamber and Kemerer [2] and, as a consequence, we will employ the same framework for compatibility's sake. The properties are:

Property 1: There are programs P and Q for which M (P) ≠ M (Q)

Property 2: If c is non-negative number, then there are only finitely many programs P for which M (P) =c

Property 3: There are distinct programs P and Q for which M (P) =M (Q)

Property 4: There are functionally equivalent programs P and Q for which M (P) ≠ M (Q)

Property 5: For any program bodies P and Q, we have M (P) ≤ M (P; Q) and M (Q) ≤ M (P; Q)

Property 6: There exist program bodies P, Q and R such that M (P) =M (Q) and M (P; R) ≠ M (Q; R)

Property 7: There are program bodies P and Q such that Q is formed by permuting the order of statements of P and M (P)≠ M (Q)

Property 8: If P is a renaming of Q, then M (P) = M (Q)

Property 9: There exist program bodies P and Q such that M (P) +M (Q) < M (P; Q)

4. EXISTING WORK

Recently Pandey and Shareef [10] proposes a UML based tool, which can derive static metrics for Component-based software systems. This tool has the ability to extract static metrics for component assembly and it can be used generally for assessing the details of a component assembly diagram. Software developers may use CAME to extract various metrics for components as which are displayed through snapshots presented in [10].

Pandey and Shareef [11] proposes an upgraded UML-based "CAME" tool, which can derive structural complexity metrics from component-based system specifications represented in UML. This upgraded "CAME" tool enables software developers and system analysts to extract metrics related to the interfaces of components at an early stage of the SDLC, helping them in identifying complex components requiring more attention. The complexity numbers calculated guide them as to where they should concentrate their testing efforts, resulting in a more reliable component-based system. This tool can be modified to extract metrics for other artifacts like composite and use case diagrams.

Ali, et al. [1] describes software behavioral models that derive from early requirements specifications such as use-ease scenarios and properties have proven useful in early analysis and checking of the design correctness of individual components or whole system.

Sun [12] present a coal gebraic model for behavioral adaptation in component-based systems. Dissimulation equivalence and refinement relationship are used to ensure that a component can replace another one. When the behavior of two components can not be matched perfectly, behavioral adaptation might be needed to allow substitution of components.

Khalilzad, et al. [5] describes complexity in the real-time embedded software domain has been growing rapidly.

In [5] authors designed an adaptive framework for scheduling component-based distributed real-time systems.

Khalilzad, et al. [6] proposed component-based software development provides a modular approach to develop complex software systems. In this paper authors focus on periodic interface models.

Mahajan, et al. [7] developed and proved the necessity of Component-Based Software testing prioritization framework which plans to uncover more extreme bugs at an early stage and to enhance software product deliverable quality utilizing Genetic Algorithm (GA) with java decoding technique. For this, authors propose a set of prioritization keys to plan the proposed Component-Based Software java framework.

5. RESEARCH PROBLEM

Component B ased S oftware E ngineering is t he w idely us ed concept in th e s oftware in dustry. M etrics p lay an important role in determining the various characteristics of a component to find out which components are reusable and what particular function they will perform. Metrics help in providing the data to the system and improve the quality of system. Metrics are also helpful in managing risk in the component based system.
To find out the solutions of the problems in existing system in various areas of software is quite popular field of research for the co mputer ex perts. CBSE i s o ne o f t he ar eas of software development w hich h ad b een i ntroduced q uite ear lier and it becomes the essential requirement for the software industry in view of en ormous co mputerization i n ev ery s ector. Today, when no field is untouched f rom s oftware us es, C BSE i s creating revolution in the software industry. Use of CBSE has explicit advantages along with some challenges. Software requires to be evaluated before the development to avoid the wastage o f r esources i f s oftware f ails an d also requires evaluation d uring th eir lif e to manage the software maintenance cost and match the technology available. Different software evaluation strategies h ave b een ev aluated with software metric m easurement an d ef fort es timation models have been introduced [8][9].
In this paper, a study has been made on how dynamic metrics are used in component based development that concentrates on the factors lik e c omplexity, s ize, r eliability, r eusability, understandability, maintainability etc. The software metrics in use have been categorized in static and dynamic as described in section 2. Evaluation of the software has been done by using dynamic m etrics and it can be v isualized b efore s oftware development using sequence diagram in UML. Such mapping and testing of the metric values is a major challenge which has been taken into consideration in this work.
DSME tool developed in this paper has been implemented on the E-learning system.

6. PROPOSED METHODOLOGY

The proposed methodology is given as following:

- Step1: Design the time sequence diagram of any proposed software using Argo UML tool according to requirements of clients.
- Step 2: Create XMI file of given time sequence diagram with the help of option Export XMI given in Argo UML. This XMI f ile c ontains all th e in formation o f tim e sequence diagram like unique xmi.id, call action, return action, association role etc.
- Step 3: Using Java based software and Netbeans tool the XMI files is then parsed for extracting information related to various dynamic metrics such a s Number o f cy cles (NC) and utilization of components in CBSE.

For Calculating N C a nd u tilization o f c omponents in CBSE, the following algorithms have been used:

```
Algorithm EvaluateCycles()
Begin
  Implement Time Sequence Diagram for the case study
  Use ArgoUML Tool to Generate the XMI file
  Use J ava t o P rocess t he X MI f ile and create a list of
  Components and their associations
  CL:=Blank List of Cycles

  For Each Component in List
    C:=Component;
    B:=Search C in CL

    If B=False Then
      Temp:=C;
      Flag:=False;
      TempCycle:= BlankList of Cyle
      TempCycle:=Temp;

   For Each Component+1 in List
        C1:= NextComponent;
        If Temp != C1 Then
          Temp:=C1;
          TempCycle:=TempCycle+Temp;
        End if;
          If C ==  Temp Then
            Flag:=True;
             Break;
          End if;
    End

      If Flag=True then
        Add TempCycle in CL
      End If;
  End;

  Return CL
End;
```

```
Algorithm EvaluateUtilization()
Begin
  CL:= EvaluateCycles();
  For Each Component in List
  Begin
    C:= Component Name
    CC:=0;
    For Each Component in Cycles
    Begin
      If C = Component then
        CC:=CC+1;
      End if
    End;
    Add C & CC in List UL
  End;
  Return UL
End;
```


7. APPLICATION OF DSME TOOL

In this section, we consider the model of E-learning system which has been designed with the help of Argo UML tool.
Our aim is to implement the DSME tool on the time sequence diagram of the five modules of E-learning system. (cf. Fig 7.1 to 7.5).
To facilitate the use of the existing suite of component assembly metrics a parser based tool, DSME (Dynamic software metric estimation tool), developed in JAVA using Netbeans 7.1.2, is used to analyze UML component assembly diagrams represented in XMI. This tool extracts existing dynamic metrics. This tool works only with XMI files that contain information like xmi.id, call action, return action, association role etc. UML component diagrams with only the elements provided by the *Argo*UML tool has been drawn. For parsing the XMI file, SAX [13] – a Java API for XML to parse the XMI file is used. The version implemented in the DSME tool is SAX 2.0.1 as the SAX parser is an easy-to-use forward parser. The flow of process of how the DSME tool works is depicted in Figure-7.6.
The component-based metrics implemented in the DSME tool are: NC, utilization of component, all defined by Narasimhan, et al. [8] (see also [9]). Table-2.4 shows some of the dynamic metrics currently obtained by using DSME tool, derived through XMI file. The coding of XMI parser for evaluating NC and Utilization of components is shown in Figure-7.7, and Figure 7.8. The XMI representation of UML component diagrams is illustrated in Figure 7.9.
Figure-7.1 shows a simple component assembly diagram i.e. Time sequence diagram created with the help of ArgoUML 0.34 (UML Modelling tool). The diagram consists of components and a Dependency indicator. XMI assigns each model element a unique xmi.id. This also defines a namespace for each element in the model. These unique IDs allow elements to reference associated elements, as (*xmi.idref*) values and also provides an access method to the data structure.

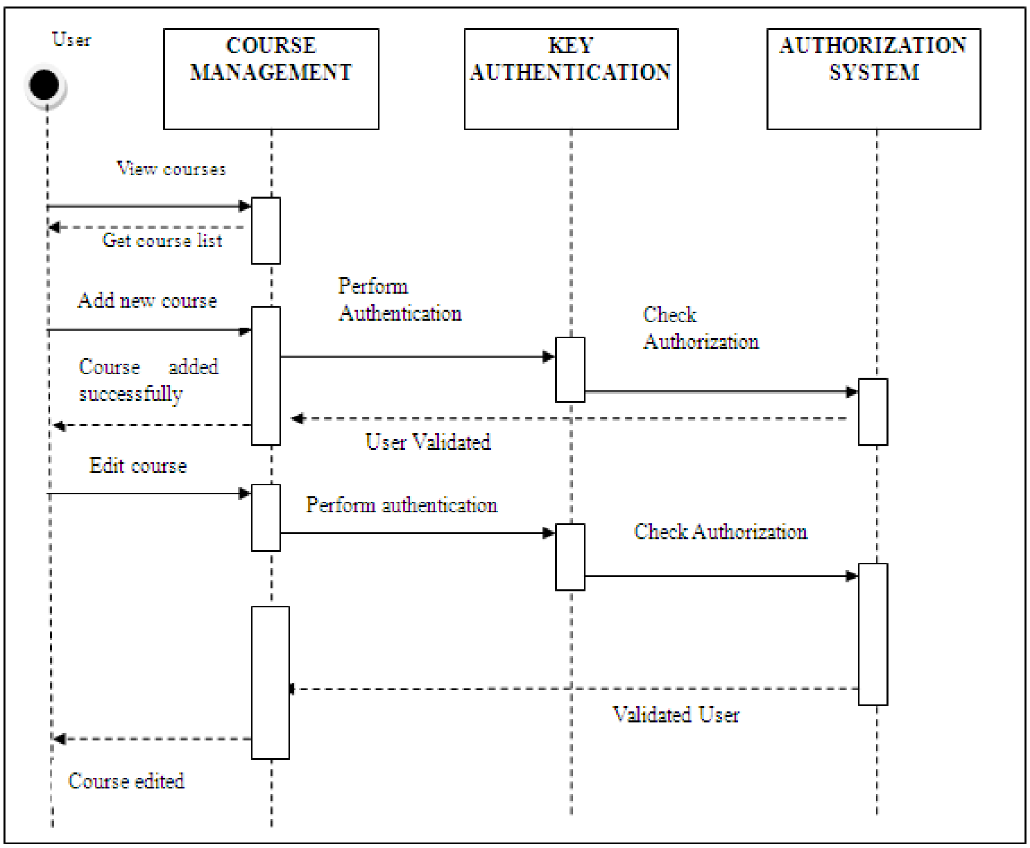

Figure -7.1 Time sequence diagram for course management

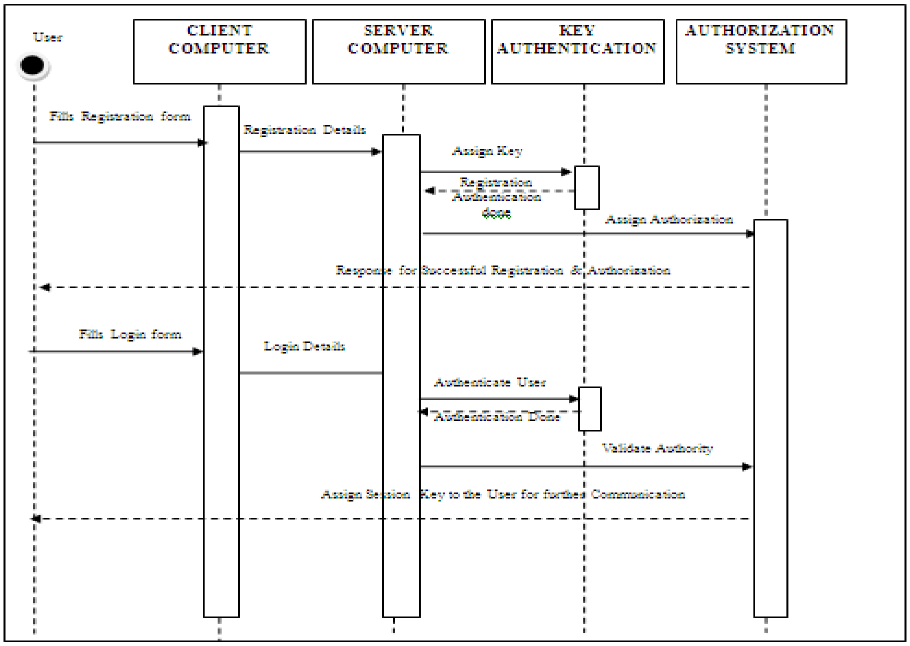

Figure -7.2 Time sequence diagram for login register

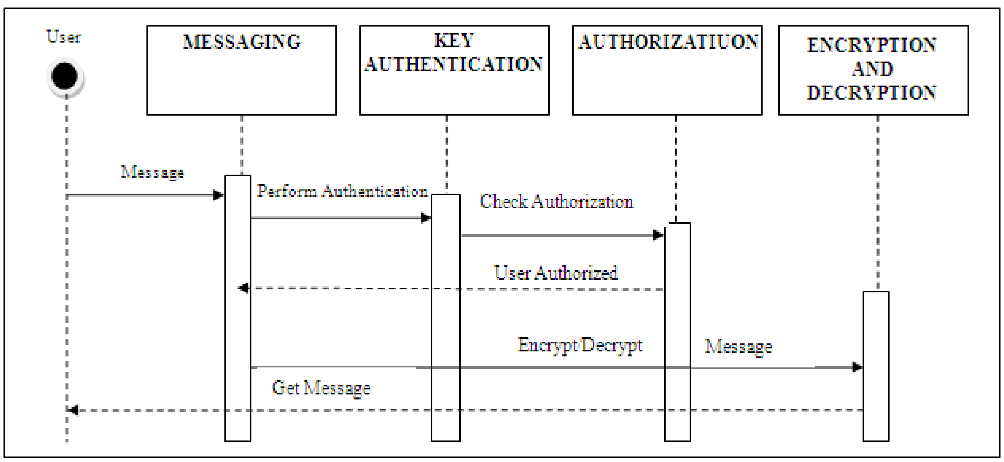

Figure -7.3 Time sequence diagram for messaging

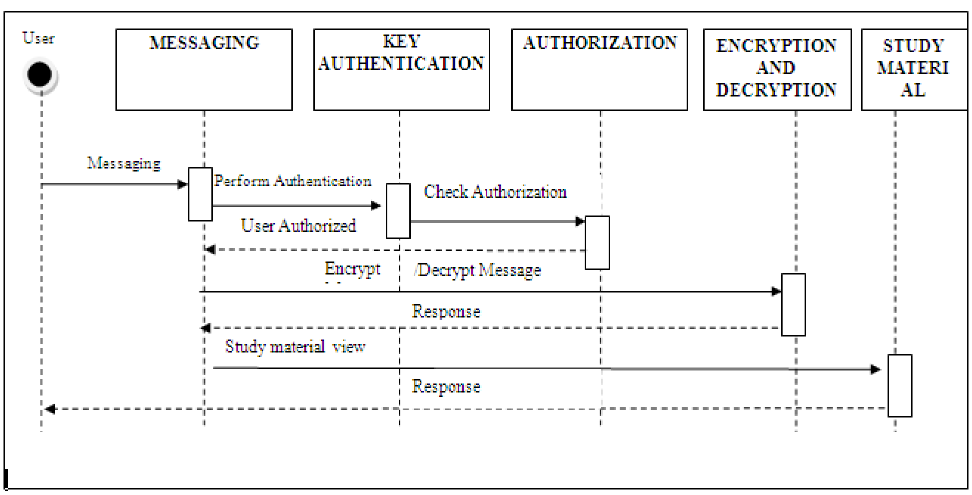

Figure -7.4 Time sequence diagram for report generation

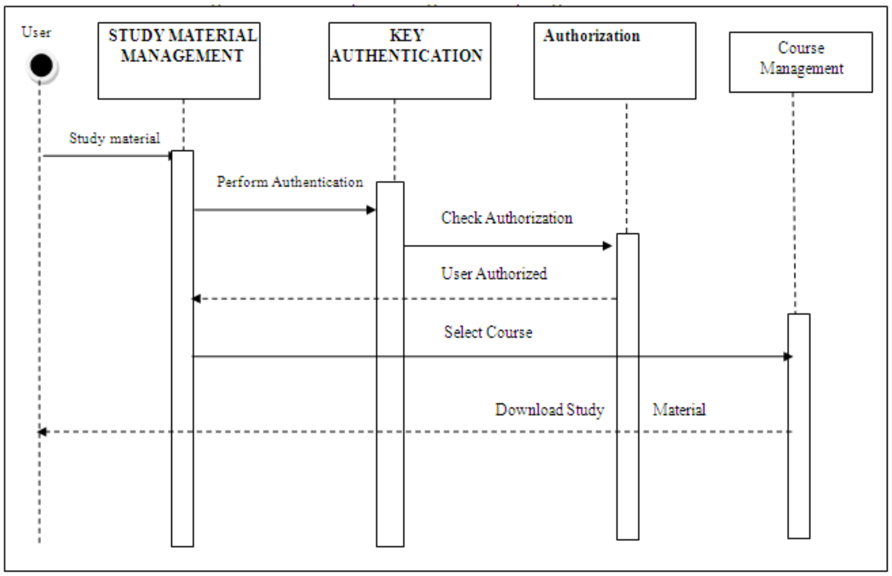

Figure -7.5 Time sequence diagram for study material management

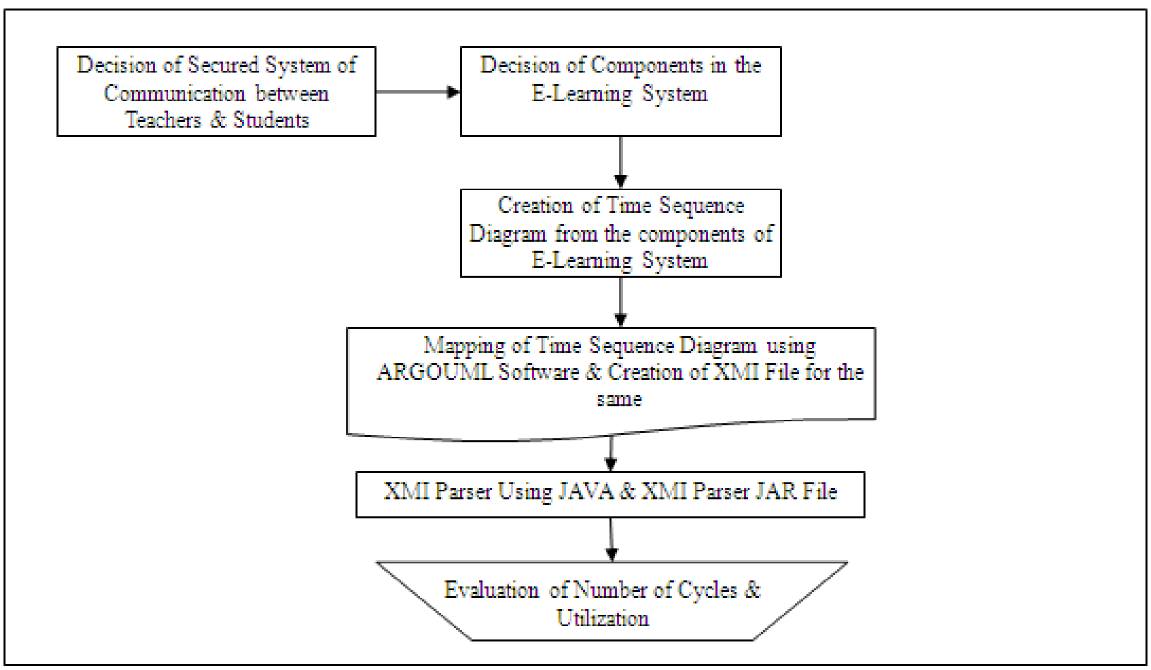

Figure 7.6: Working of DSME Tool for Project E-Learning System

```
public void showNC() {
    String loopStart = "";
    String from = "";
    String loop = "";
    for (int i = 0; i < GlobalLists.lstMessages.size(); i++) {
        GlobalLists.lstMessages.get(i).setIsUsed(false);
    }
    for (int i = 0; i < GlobalLists.lstMessages.size(); i++) {
                                                        Message msg = GlobalLists.lstMessages.get(i);
        if (msg != null && msg.getSenderId() != null && !msg.getSenderId().equals("")) {
            from = getClassifierRole(msg.getSenderId());
            String to = getClassifierRole(msg.getReceiverId());
            if (loopStart.equals("")) {
                loopStart = from;
                loop = msg.getName() + "::" + from + "=>" + to + ",";
            }
            else {
                if (to.equals(loopStart)) {
                    loop += from + "=>" + to + ",";
                    GlobalLists.lstLoopDetails.add(loop);
                    loop = "";
                    from = "";
                    to = "";
                    loopStart = "";
                }
                else {
                    loop += from + "=>" + to + ",";
                }
```

Figure 7.7 : Java implementation function for calculating show NC

```
void showUtilization() {
    for(int i=0;i<GlobalLists.lstClassifierRoles.size();i++) {
        ClassifierRole cr = GlobalLists.lstClassifierRoles.get(i);
        if(cr!=null && !cr.getName().equals("")) {
            int uc = 0;
            for(int j=0;j<GlobalLists.lstLoopDetails.size();j++) {
                String loop = GlobalLists.lstLoopDetails.get(j);
                if(loop.indexOf(cr.getName())>=0) { uc++; }
            }
            lmUtilization.addElement(cr.getName()+" :: "+ uc);
        }
    }
}
```

Figure 7.8 : Java implementation function for calculating show utilization of component

```
<UML:Model xmi.id = '-64--88--23-1--64f815cc:155b4aebd88:-8000:0000000000000865'
   name = 'SDForLogin' isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
   isAbstract = 'false'>
   <UML:Namespace.ownedElement>
    <UML:Collaboration xmi.id = '-64--88--23-1--64f815cc:155b4aebd88:-8000:000000000000087C'
     name = 'LoginRegistration' isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
     isAbstract = 'false'>
     <UML:Namespace.ownedElement>
      <UML:ClassifierRole xmi.id = '-64--88--23-1--64f815cc:155b4aebd88:-8000:0000000000000880'
       isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
       <UML:ClassifierRole.multiplicity>
        <UML:Multiplicity xmi.id = '-64--88--23-1--64f815cc:155b4aebd88:-8000:0000000000000882'>
         <UML:Multiplicity.range>
          <UML:MultiplicityRange xmi.id = '-64--88--23-1--64f815cc:155b4aebd88:-000:0000000000000881'
```

Figure-7.9: XMI representation of a component dependency diagram

8. RESULT AND DISCUSSION

Results obtained for NC and utilization of components has been depicted in screen shots Figure 8.1 to Figure 8.4 for module course management displayed in Figure 7.1. Their significance in evaluating the software is as follows:

Number of Cycle: The NC is number of cycles within an integrated component in a graph representation

$$NC = \#cycles$$

Where # cycles is the number of cycles or loops within the graph.

When an application is executed, components call other components through the provided interfaces. Components with similar purposes create a cycle within the component's graph representation. More cycles typically indicate more special purposes within a component assembly.

Identifying cycles creates clustering in the whole component assembly. Each cluster might indicate a super component, i.e., a component that consists of other components.

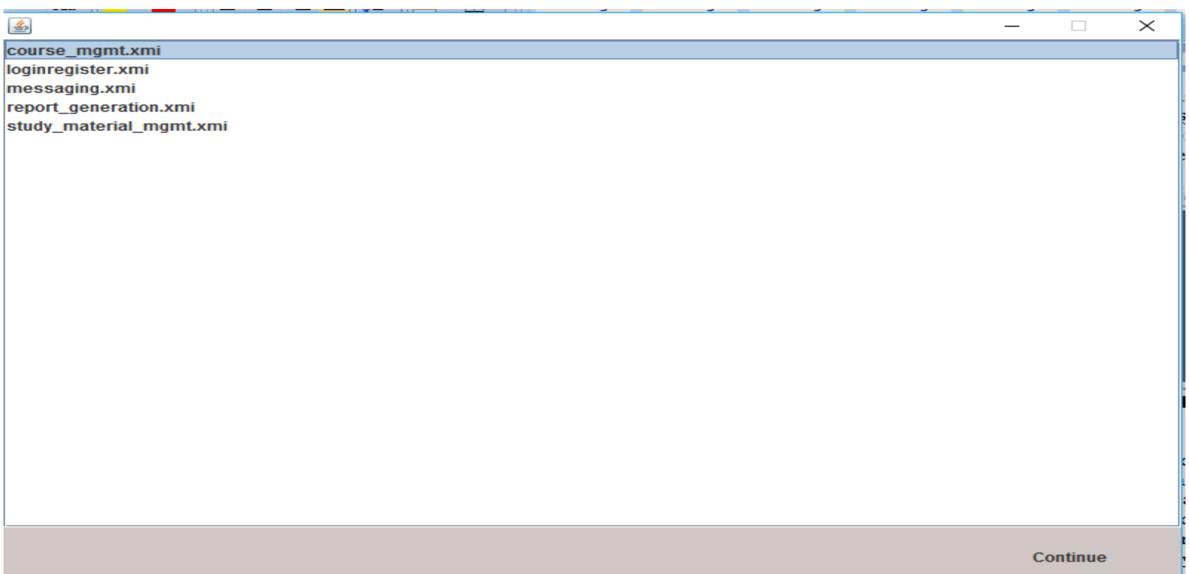

Figure 8.1: GUI for displaying information of component assembly using DSME tool

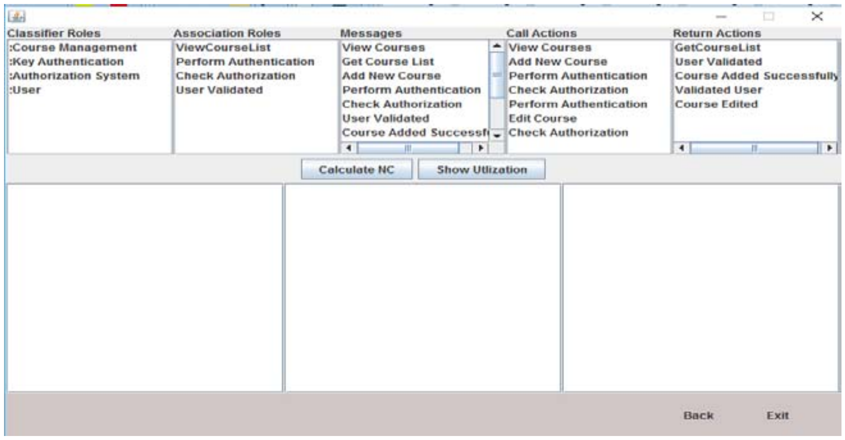

Figure 8.2: GUI for displaying information of course management

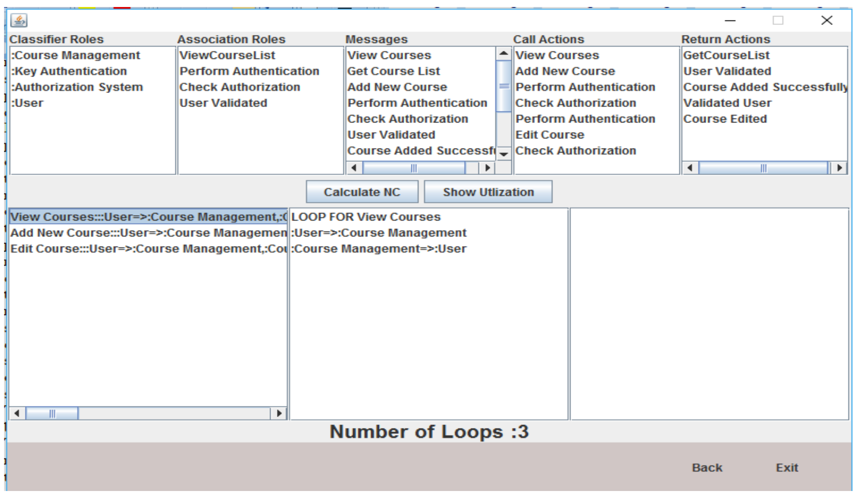

Figure 8.3: GUI for displaying information calculate NC for course management

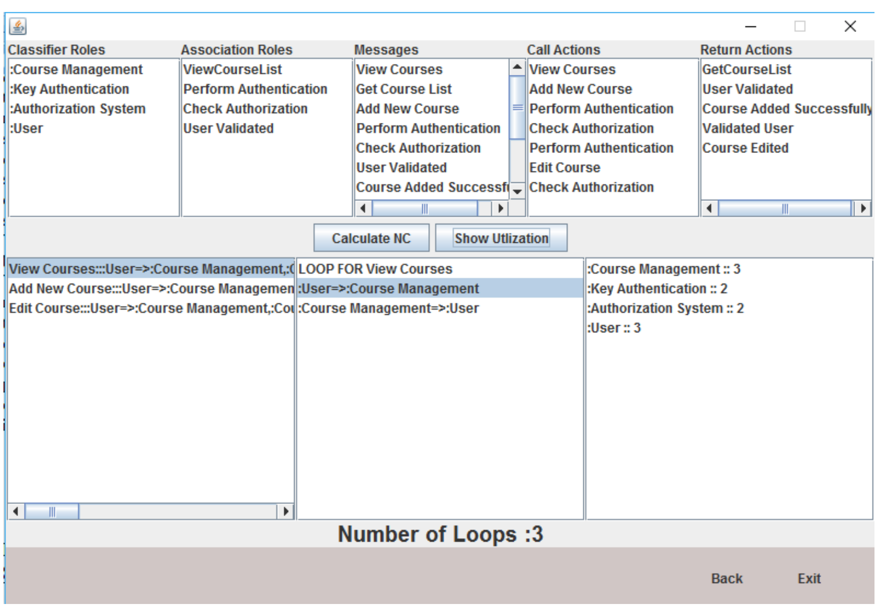

Figure 8.4: GUI for displaying information show utilization for course management

Component Utilization: It is the count of components used in a module in different cycles. This shows the importance of the component in any module. Utilization of the component in all modules must also be considered for more accuracy.

The following table shows the NC and utilization of components of for all module course management of E-learning system:

Module	NC (No. of cycles)	utilization of component
course management	3	user : 3 course management : 3 key authentication : 2 authorization system : 2
Login Register	2	user : 2 client computer : 2 server computer : 2 key authentication : 2 authorization system : 2
Messaging	1	user : 1 messaging : 1 key authentication : 1 authorization : 1 Encryption & Decryption : 1
Report generation	1	user : 1 messaging : 2 key authentication : 1 authorization : 1 Encryption & Decryption : 1 Study material management :1
Study material management	1	User :1 Study material mgmt : 1 key authentication : 1 authorization : 1 course mgmt : 1

9. CONCLUSION AND FUTURE WORK

The evaluation of number of component cycles in DSME tool helps in deciding the super components, which in turn can be used in decision of effort required, reducing the overall cost and reducing the complexity of the software system in early stage i.e. at designing stage. Similarly utilization of components in a module helps in deciding most usable components in the system and hence importance of the components during the implementation of software system. Overall, both of the evaluated component metrics in early stage of software development can be used in reducing the software cost, reduces the glue code cost, integration complexity and distinguishing the components as per their importance.

Finally, it may be concluded that the Number of cycles and degree of utilization is one of the key component for the cost estimation of CBSE. Hence, by computing NC and degree of utilization, basically we would be in a position to predict the approximate cost of CBSE.

This tool may also be modified to extract other dynamic metrics for component-based systems, which will be implemented in a future version.

10. REFERENCES

[1] Ali A., Jawawi D. N. A., Ibrahim A. O. Isa M.A, Deriving behavioural models of component-based software systems from requirements specifications, Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), 2015 International Conference on, Khartoum, 2015, pp. 260-265. doi: 10.1109/ICCNEEE.2015.7381373

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

[3] Fenton N.E., Pfleeger S.L., Software Metrics: A igorous & Practical Approach, second ed., PWS Publishing Company, Boston, 1997.

[4] Henderson-Sellers B., Object-Oriented Metrics: Measures of Complexity, Prentice-Hall PTR, Upper Saddle River, NJ, 1996.

[5] Khalilzad N., Ashjaei M., Almeida L., Behnam M., Nolte T., Adaptive multi-resource end-to-end reservations for component-based distributed real-time systems, Embedded Systems For Real-time Multimedia (ESTIMedia), 2015 13th IEEE Symposium on, Amsterdam, 2015, pp. 1-10. doi: 10.1109/ESTIMedia.2015.7351772.

[6] Khalilzad N., Behnam M., Nolte T., On Component-Based Software Development for Multiprocessor Real-Time Systems, Embedded and Real-Time Computing Systems and Applications (RTCSA), 2015 IEEE 21st International Conference on, Hong Kong, 2015, pp. 132-140. doi: 10.1109/RTCSA.2015.27.

[7] Mahajan S., Joshi S. D., Khanaa V., Component-Based Software System Test Case Prioritization with Genetic Algorithm Decoding Technique Using Java Platform, Computing Communication Control and Automation (ICCUBEA), 2015 International Conference on, Pune, 2015, pp. 847-851. doi: 10.1109/ICCUBEA.2015.169.

[8] Narasimhan L.V., Parthasarathy P.T., Das M, Evaluation of a Suite of Metrics for Component Based Software Engineering (CBSE), issues in Informing Science and Information Technology Volume 6, 2009.

[9] Narasimhan L.V., Hendradjaya B., Some theoretical consideration for a suite of metrics for the integration of software components, issues of information Science and Information Technology, 177 (2007) 844-864.

[10] Pandey R. K., Shareef J.W., CAME: Component Assembly Metrics Extraction using UML, ACM SIGSOFT Software Engineering Notes, July 2013, Volume 38 Number 4, pg. 1-12.

[11] Pandey R. K., Shareef J.W., Design of a Component Interface Complexity Measurement Tool for Component-Based Systems, ACM SIGSOFT Software Engineering Notes, July 2015, Volume 40 Number 1, pg. 1-12.

[12] Sun M., Towards a Coalgebraic Semantics of Behavioral Adaptation in Component-Based Software Systems, Computer Science and Mechanical Automation (CSMA), 2015 International Conference on, Hangzhou, 2015, pp. 41-44. doi: 10.1109/CSMA.2015.15.

[13] SAX, Retrieved on March 5, 2011 http://sax.sourceforge.net.

[14] Weyuker E.J., Evaluating software complexity, IEEE Transaction on Software Engineering 14 (9) (1988) 1357–1365.