International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN:2147-6799 www.ijisae.org Original Research Paper

A Novel Approach to Abstract the Design Information and Minimal Cover from the Restructured Java Program

Aparna K. S.[1] and Dr. R. N. Kulkarni[2]

Submitted: 15/12/2023 **Revised**: 20/01/2024 **Accepted**: 03/02/2024

Abstract: The software industry has undergone enormous development during the previous few decades. Java programming was used to create many of the software applications needed to run the current generic applications. To support the business operations, these applications have undergone changes based on the changing requirements of the customer or organization. Many changes performed during these need-based updates are made only to the applications and not in the relevant documents related to the software. Further altering these software systems could occasionally cause a software crash. It might be challenging to add new features to the old programs as there is a possibility of redundant code.

To overcome the problems related with either redundancy of the code or documentation, a novel methodology is proposed and a tool is developed to abstract the various components from the input java programming system such as control flow graph, data flow information in the control flow order and the various attributes which are participating in the program. Further the data flow graph and control flow graph are used to find the functional dependencies and attribute closures. The abstracted attributes are used to find the minimal cover, which results in the computation of the functionality of the program.

Keywords: *Restructuring, Data flow graph, Control flow graph, Referred Variable, Defined variable, functional dependency*

1. Introduction

The demand to use technology and software to automate organizational tasks has increased dramatically during the past few decades. The utilization of application systems constructed using Java programming has experienced significant demand. These application systems, which were created decades ago, have undergone several ongoing, need-based adjustments. As the original writers were unavailable, there was a chance of redundancy in the code, when these changes were implemented. Moreover, the programming language was flexible enough for the various programmers to use alternative names for the variables, operations, and methods. As, the success of the translation depends on the knowledge of the translator's expertise and experience, thus translating the complete application to a new programming language was equally challenging, as it needs to comprehend the organization's entire business rules, which gets modified periodically leading to unstructured code. This unstructured-ness can be handled with the help of restructuring techniques for abstracting the design information.

Taxonomy:

Control flow graph represents graph implying the execution flow of the overall program

Data flow graph represents graph implying the flow of data along the control flow.

Referenced attribute get referenced in a executable statement of a program without any modifications after execution

Defined attribute are attributes, changing during the execution of statement.

Restructuring is the transformation from one form to another without changing its functionality

2. Related Work

The various methods of restructuring Java applications are covered in [1], the preprocessing step for obtaining the design data necessary for static analysis of the input. Restructuring [2], benefits the memory optimization significantly contributing to the memory management issues required for reengineering. The study [6] presents a novel approach for generating a descriptive file from source code using comment lines. This method takes an existing Java source code program and adds a description file to it automatically. It contains the parameters functions, conditions, and description. All of this information will eventually be included in a text document file that the framework creates. In our approach we are eliminating these comment lines, blank lines and changing the multiple lines into a single line as these comment lines will not be updated periodically. The next step of this paper is to extract the design information. The different approaches are discussed. Firstly extracting the Data flow diagram from the given java program is discussed [4]and represented in the table format

[1]*Assistant Professor, Department of Computer science & Engineering, RYMEC, Ballari, Research scholar, VTU, Belagavi.*
[2]*Professor and HOD, Department of Computer Science & Engineering, Ballari Institute of Technology and Management, Ballari, aparna.vastrad@gmail.com, rnkulkarni@bitm.edu.in*

using which, the design of the software is extracted. The feature extractor method is to extract the data from the generated XML file. Information gathered from method names [7], variable names, and comments in the project source code that is being analyzed. Each Java class is presented as a document. Next, using Latent Dirichlet Allocation (LDA), a set of topics representing the different functionality offered by the project are identified. In approach [2], the source code goes through various stages of the static analysis and functionality is derived from program slicing. The value of data representation in tables is underlined in [3], which presents the information of the sequence diagram in the form of useful table. In [4], a table that serves as a static representation of the class diagram provides complete information. Meanwhile the activity and use case diagrams in [5] are produced using the source code, indicating that the focus should be on other UML diagrams to extract the design information. In [6], the Control Flow Graph (CFG) is used to represent the source code representing all pathways that an application will take throughout its execution. Further this CFG is then used to extract other related concepts. The Java programs are used for various degrees of analysis in [7] and are rendered as graphs. ProgQuery and Neo4J graph, among other tools, are effective, but complexity rises with larger programs. Models are made from the intermediate representation of the source code and are used to extract UML models in reverse method combining static and dynamic analysis[18,19]. The static and dynamic analysis of bytecode is done and represented in the form of table [9]. It emphasizes how the results of the static code analysis for the Java programs are extracted as graphs[10]. In[11,12], the static analysis of the tools and techniques for locating bugs, security vulnerabilities, code duplication, and code smells are addressed in detail. Opensource static code analysis is one of the tools that is accessible to find errors and different coding techniques. In [13], the static analysis of the code is carried out utilizing AST, assessing the many paths that the programs can take performing a control flow and programs flow analysis. The survey discussed above use graphical representations to illustrate static analysis for the different application, however these graphical representations have memory and maintenance issues that make it difficult to retrieve and preserve the data. These graphical representations of the programs, presented in a variety of ways, are useful for evaluating the program code, but they undoubtedly used a lot of memory and added overhead retrieving the data for later processing. As a result, alternative forms of graphical representation, such as tabular representation, proved to be more effective in terms of memory utilization, processing speed, throughput analysis. The proposed work discusses the control flow analysis of the input Java programs in a tabular representation. The control flow and data flow is derived from the control flow table and data flow table in our approach. From the data flow table, the functional dependencies are extracted and further these functional dependencies are minimized as discussed below. In [17], closure and the minimal cover algorithm were used to resolve the superfluous functional dependencies. The Armstrong axioms are used to create the F+ algorithm, which generates closure of functional dependencies (FDs). Finding functional relationships and minimizing them with the minimal cover method is done for various sets of functional dependencies, as discussed in detail [18]. A lifetime model [15] for individual source code lines or tokens can be used to guide preventive maintenance, estimate maintenance effort, and, in general, find characteristics that can increase software development efficiency. All components involved in the software system [14] should be able to easily comprehend and discern the necessary components of the requirements through the software design. In our methodology, the functional dependencies extracted from the data flow table have lot of redundant functional dependencies which have to minimized because there will be an unequal distribution of functional dependencies. Consequently, these functional dependencies might be mentioned in other statements explicitly or indirectly. This scenario complicates the design extraction process. In our methodology, this problem is resolved by applying the concept of minimal cover [20,21] and the storage will be conserved as a result, by removing redundant and unnecessary productions in the functional dependencies, resulting in database optimization

3. **Proposed Methodology**

A. ***Restructuring the Input Program***: Restructuring is the method of bringing modifications to the software structure explicitly without affecting its functionality. In our proposed methodology, comment line elimination, blank line elimination, and making multiple declaration statements into a single declaration statement is a part of the restructuring [3] process. Other steps include statements ending with a semicolon, curly braces insertion before the start and end part statements in the body of the loop, respectively, and assigning sequential line numbers to logic statements. This process of restructuring changes the program structure without affecting its function. This refactored code will be a input for extracting the design information.

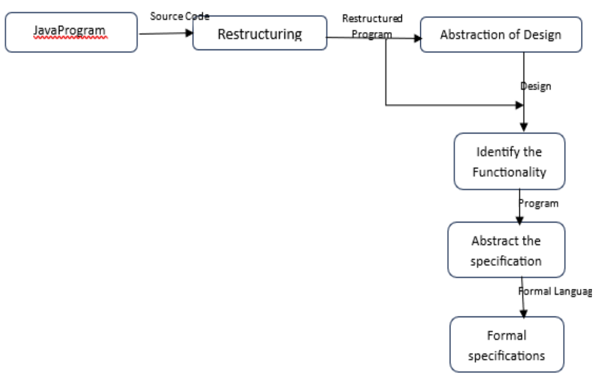


Fig 3.1: Block diagram of the proposed methodology

Consider the input java program in Figure 2 which is a non-structured code.

```
import java.util.Scanner;
public class Quad
{
public static void main(String[] args)
{
double a1,b1,c1;
//determinant is det
double det;
//r1=root1 and r2=root2
double r1,r2;
Scanner in =new Scanner(System.in);
//reading values of a1,b1,c1
System.out.println("Enter the value of a");
a1=in.nextDouble
System.out.println("Enter the value of b");
b1= in.nextDouble();
System.out.println("Enter the value of c")
c1= in.nextDouble();
/*comparing the values and chking for determinanat*/
if(a1==0|| b1==0||c1==0)
{
System.out.println("Invalid input:Enter non-zero co-effeceints");
System.exit(0);
else
{
d1=b1 *b1 – 4 *a1 * c1
if(d1>0)
{
//calculation of root1 and root2 for positive det values
r1 = (-b1+ Math.sqrt(d1)) / (2 * a1);
r2 = (-b1 - Math.sqrt(d1)) / (2 * a1);
System.out.println("root1 and root2 are", r1, r2);
}
else if(d1= =0)
{
//executed when roots are same
r1 = r2 = -b1 / (2 * a1);
System.out.println("Roots are real and equal" +r1);
System.out.println("Root1 = Root2 = "+ r1);
}
else
{
//calculating the negative part
double rpart = -b1/ (2 *a1);
System.out.println("Roots are complex and imaginary");
//printing values of roots
System.out.println("r1="+rpart+"+i"+ imaginaryPart+"r2="+rpart+"-i"+ imaginaryPart);
}//else  ends
}//if ends
Sc.close();
}//main ends
}//quadratic ends
```

Fig 3.2:Input java Program

From Figure 3.3, it is very clear that the CPU execution time varies a lot with the given input size of the program

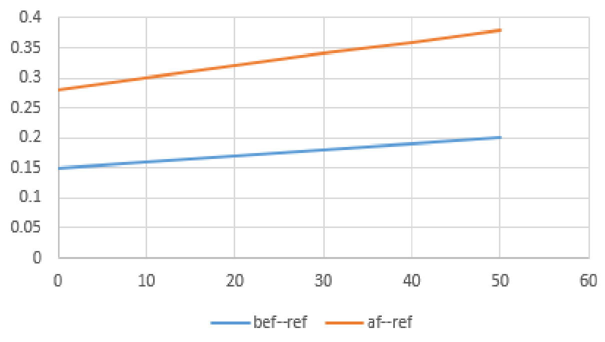


Fig 3.3:Graphical analysis of the Restructuring It is obvious that the varied execution time (in seconds) implies the effect of restructuring process. Considering the input size of 50 lines, the graph in Figure 3 illustrates that after restructuring the execution time has reduced a lot and for the same input size the execution time is comparatively less, which indicates the effect of restructuring the input code.

B. Abstraction of Control flow graph from Java Program: The program is represented in the form of a graph, where each node in the control flow graph represents either a conditional statement, computational statement, or the invocation of a member function. An edge connecting those vertices represents the control flow between the statement. The program execution control flow is shown on the Control Flow Graph (CFG). Each line in Java can include a computational statement, a predicate statement, or an invocation statement. When the program is represented as a graph with one vertex (or node) assigned to each statement, the joining edge between the two vertices represents the control flow. This representation is considered inefficient due to the bulk size of java applications, hence these CFG are represented in tabular form using the four columns, where column1 implies start of the program, column2 represents the point of occurrence of the first control statement, third column represents the point of true block(Trans1) and fourth column indicates the false block(Trans2). Usually Trans1=start of the program and Trans2= end of the program and the same procedure gets repeated for all the control statements like if-else, while, for, do-while, switch, break, continue, return. The function invocation present in the input program are handled by putting the point of callee function in trans1 and trans2 will be obviously empty as it is not having the true and false blocks. Further these true and false blocks are analyzed and the above procedure is followed for any nested control structures occurring in it. The concept of defining nodes and Usage/Referenced nodes are used to construct the control flow table.

Algorithm for abstraction of Design information

Input: java program after restructuring

Output: Tables depicting control flow, data flow and minimal cover of attributes

Step 1: [Java program preprocessing with restructuring]

- Input Java Program is scanned
- Appending main program with the imported packages
- Comment line elimination
- Blank lines elimination
- Single line statements got by converting multi statement line and multiline statements
- Each statement are given the physical line numbers

These restructuring steps help in the efficient representation of the input program, required for analyzing the design information in the form of control and data tables

Step2: [Control flow table abstraction from the Restructured Java program]

Scan the input Java program starting from the string 'Public static void main' and search for the control keys of the input program comparing with that of VERBS(control statements) as discussed above. If the scanned control keys belongs to VERBS, then represent that information in the table format representing statement number of the first line of the program in column1, column2 is the point where the first control key occurs column3 is **Tr1** representing the transition to the starting point of the true block and column4 is **Tr2** representing the transition to the starting point of the false block and this process gets repeated for all the control keys present in the input program and as well for the nested control structures

Step 3: [Data flow table abstraction from the Restructured Java program]: The flow of data variables in the program is based on the control flow order. These data variables are defined and referred at several points of the program. Identification of these referred and defined variables in every line of the input program is noted in the tabular format, where first column indicates input program line number, second column is for defined variables and third column is for referred variables .

Step4: [Finding the minimal cover of the functional dependencies]. From the data flow table, the functional dependency existing between these variables are extracted. These functional dependences are minimized to extract minimal cover of attributes eliminating redundant attributes. The minimal cover is done using these steps:

i. Representing the Functional Dependencies in a canonical form where RHS should have a single attribute.

ii. Remove the redundant FDs (which can be derived from the transitive principles).

iii. Remove the extraneous attributes, by finding the closure of all the LHS attributes of the Functional dependencies having non-single attribute. If CLOSURE(A)= ABC then BC are extraneous attributes and have to be removed.

iv. Apply the Armstrong principles and try to group FDs using common LHS

4. Results and Discussion

Refactored code :

```
1.  import java.util.Scanner;
2.  public class Quad
3.  {
4.  public static void main(String[] args)
5.  {
6.  double a1,b1,c1,d1,r1,r2;
7.  Scanner in =new Scanner(System.in);
8.  System.out.println("Enter the value of a");
9.  a1=in.nextDouble
10. System.out.println("Enter the value of b");
11. b1= in.nextDouble();
12. System.out.println("Enter the value of c")
13. c1= in.nextDouble();
14. if(a1==0|| b1==0||c1==0)
15. {
16. System.out.println("Invalid input:Enter non-zero co-effeceintsâ€);
17. System.exit(0);
18. else
19. {
20. d1=b1 *b1 - 4 *a1 * c1
21. if(d1>0)
22. {
23. r1 = (-b1 + Math.sqrt(d1)) / (2 * a1);
24. r2 = (-b1 - Math.sqrt(d1)) / (2 * a1);
25. System.out.println("root1 and root2 are", r1, r2);
26. }
27. else if(d1= =0)
28. {
29. r1 = r2 = -b1 / (2 * a1);
30. system.out.println("Roots are real and equal" +r1);
31. system.out.println("Root1 = Root2 = "+ r1);
32. }
33. else
34. {
35. double rpart = -b1 / (2 *a1);
36. System.out.println("Roots are complex andimaginary");
37. System.out.println("r1="+rpart +"+i"+ imaginaryPart+" r2= "+ rpart+"-i"+ imaginaryPart);
38. }


39. Sc.close();
40. }
41. }
```

Fig 4.1: Restructured Java program

In this program, the static analysis of the code is done. The execution starts from the main. In this methodology, the control statements are scanned from main. These control statements are analyzed based on their structure, the different block of statements get executed. In the above program, we have the control statement at line no 14, and true block starts at line 15 and false block from line 19. In true block, further we have exit statement at line 17, so the control jumps to line 42. In the false block, further there is another condition checking for determinant at line 21, where true block moves to line 22 and completes its execution at 26 and in false block, the control statements are checked, if true moves to 28 and the block from 28 to 32 gets executed and the else block gets executed from 34 to 38 and both the blocks get terminated at line 40. The control flow table is as shown in the Figure 4.1. The data flow table in Figure 4.2, represents the flow of the variables at different control points of the program.

S	E	TR1	TR2
1	14	15	19
15	17	42	
19	21	22	27
22	26	37	
27	27	28	34
28	32	42	
34	39	40	

Fig 4.2: Control flow table of Figure 4.1

Line No	Defined Vars	Referred Vars
2	Quad	
9		a1
11		b1
13		c1
14		a1 b1 c1
20	d1	a1 b1 c1
21		d1
23	r1	a1, b1, d1
24	r2	a1, b1, d1
27	-	d1
29	r1, r2	b1, a1
30	-	r1
31		'r1'
35	rpart	b1 a1

Fig 4.3: Data flow table of Figure 4.2

This proposed tool is tested for different combinations of inputs and the tool is compared for its efficiency. The existing tools are building the control flow order using the graphical representation. This graphical representation has lot of concerns with respect to memory issues. The proposed tool analyzes the static analysis of the code which is very easily traceable and consumes very less execution time and gives good throughput analysis as discussed below. From the Data flow table, rows in which there exists both referred and defined variables are identified and

such rows imply the functional dependencies between the variables. The following FunctionalDependencies(FD) are extracted from the data flow table.

a1 b1 c1→d1

a1 b1 d1→r1

a1 b1 d1→ r2

a1 b1 →r1 r2

a1 b1→rpart

After applying step1 of the minimal cover algorithm , we get

a1 b1 c1→d1

a1 b1 d1→r1

a1 b1 d1→ r2

a1 b1 →r1

a1 b1→r2

a1 b1→rpart

Applying step2, we get the same FDs, as we do not have redundant FDs. Applying step3 we consider the LHS having more than two attributes. The closure rule is applied and checked to see any extraneous attributes. All FDs are equally important and hence all FDs are retained.

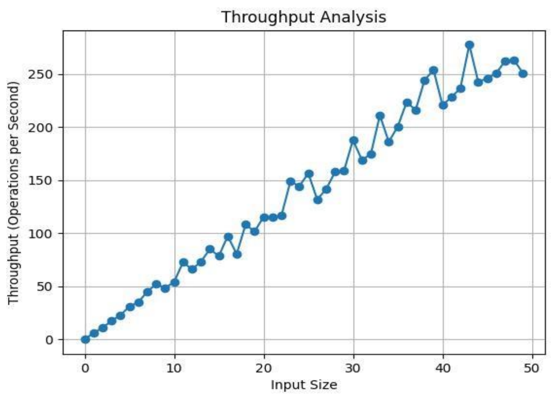
Applying the step4, group all FDs having the common LHS together.

a1 b1 c1 →d1

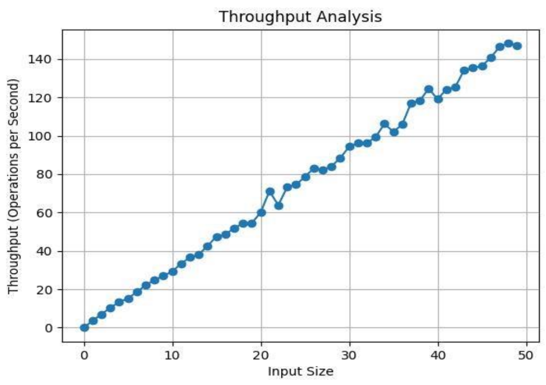
a1 b1 d1 → r1, r2

a1 b1 → r1 r2 rpart

This is the minimal cover obtained which is equivalent to the original set of FDs having fewer number of FDs. The minimization of the functional dependencies in a program code is done using the minimal cover process. These functional dependencies are minimized and the redundant and irrelevant attributes are removed, which are added by the different programmers during the maintenance phase to keep the structure of the application program intact with business policies.



Before restructuring



After restructuring

Figure 4.3: Throughput graph of the input program before and after restructuring

The graph (before restructuring) in Figure 4.3 varies a lot with the number of operations per second as the program is not restructured. The number of operations per second is competitively less after restructuring as shown in Figure4.3(after restructuring)

5. Conclusion and Future Work

This paper presents an automated tool for abstraction of the design information of the input Java program in the form of control flow graph and data flow graph and stored in the form of Control flow table and data flow table. The functional dependencies are extracted from the Data flow table and minimized to get the optimal set of attributes using which the code can be comprehended. The proposed tool is tested for its correctness and completeness by applying on different programs of varying complexities. The tool generates expected output for all the given programs and the functional dependencies abstracted from the Data flow table are further minimized. From these minimized attributes, the different program slices are extracted using backward slicing. Using these slices, the different functionalities present in the input Java program are identified and extracted.

Author contributions

The two authors have mutually discussed and analysed and worked on the methodology and conducted different tests for its effectiveness

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] Dr. R. N. Kulkarni and Aparna K.S, “A Novel Approach to Restructure the Input Java Program”, International. Journal of Advanced Networking and Applications Volume 12 ISSN: 0975-0290 4621, 2020.

[2] R. N. Kulkarni, Venkata Sandeep Edara, “Memory Optimization Using Distributed Shared Memory Management for Re-engineering”, International Journal of Intelligent systems and applications in Engineering, IJISAE, 2023.

[3] Dr. R. N. Kulkarni and Padma priya Patil, ”Abstraction of Information Flow Table from a Restructured Legacy ‘C’ Program to be amenable for Multicore Architecture”, 5th International conference on innovations in computer science and engineering.

[4] R. N. Kulkarni , Mr. P. Pani Rama Prasad, “Novel Approach to Abstract the Data Flow Diagram from Java Application Program”, International Journal of Intelligent systems and applications in Engineering, IJISAE, 2023.

[5] Dr. R. N. Kulkarni and C. K. Srinivasa, "Novel approach to transform UML Sequence diagram to Activity diagram", Journal of University of Shanghai for Science and Technology, Volume 23, Issue 7, July2021.

[6] Dr. R. N Kulkarni and P. Pani Rama Prasad, "Abstraction of UML Class Diagram from the Input Java Program, International Journal of Advanced Networking and Applications Volume 12 , ISSN: 0975-0290.

[7] RashaGh Alsarraj1, Atica M. Altaie2, Anfal A. Fadhil 3, “Designing and implementing a tool to transform source code to UM L diagrams”, Periodicals of Engineering and Natural Sciences ISSN 2303-4521 Vol, March 2021.

[8] Ali Hameed Mohsin , Mustafa Hammad “A Code Summarization Approach for Object Oriented Programs” , International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT),2019.

[9] Code Christos Psarras, Themistoklis Diamantopoulos, Andreas Symeonidis “A Mechanism for Automatically Summarizing Software Functionality from Source”. IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)

[10] Rafael R. Prado, Paulo S. L. Souza, George G. M. Dourado, Simone R. S. Souza, “Extracting Static and Dynamic Structural Information from Java Concurrent Programs for Coverage Testing”, 2015 XLI Latin American Computing Conference.

[11] Oscar Rodriguez-prieto1, Alan Mycroft, Francisco ortin, “An Efficient and Scalable Platform for Java Source Code Analysis Using Overlaid Graph Representations”, Received March 10, 2020.

[12] Umair Sabir, Farooqui Azam, Samiullah, Muhammad Waseem Anwar, Wasi Haider butt, and Anam Amjad, “A Model Driven Reverse Engineering Framework for Generating High Level UML Models From Java Source Code” , Published in IEEE access dated November 13, 2019.

[13] Shafiullah Soomro, Zainab Alansri, Mohammad Riyaz Belgaum, “Path Executions of Java Bytecode Programs” Advances in Intelligent Systems and computing, Springer, 2017.

[14] Gang Shu, Boya Sun, Tim A.D. Henderson, Andy Podgurski, “Java PDG: A New Platform for Program Dependence Analysis”, Sixth International Conference on Software testing, Verification and Validation. IEEE, 2013.

[15] Dusanka Dakin, Srđan Spasojevic, Sonja Ristic Danilo Nikolic, Darko Stefanovic, “Analysis of the tools for

static code analysis", 20th International Symposium March 2021.

[16] Eda Ozcan , Damla Topalli , Gul Tok Demir and Nergiz Ercil Cagily, "A user task design notation for improved software design", London, UK, PeerJ computer science, 2021

[17] Diomidis Spinelli's, Panos Louridas and Maria Kechagia, "Software evolution: the lifetime of fine-grained, elements", London, UK, PeerJ computer science, 2021

[18] Shafiullah Soomro, Zainab Alansri, Mohammad Riyaz Belgaum, "Control and Data flow execution of Java program", Asian Journal of Scientific Research DOI: 10.393/ajsr , 2017.

[19] Abdul vahab karuthedath Thrissur ,Sreekutty Vijayan, Vipin Kumar K.S. *"*System ependence Graph based test case generation for Object Oriented Programs*",* International Conference on Power, Instrumentation, Control and Computing (PICC), 2020.

[20] Jeong Yang ,Young Lee, Deep Gandhi, Sruthi Ganesan Valli, "Synchronized UML Diagrams for Object-Oriented Program Comprehension", The 12th International Conference on Computer Science & Education (ICCSE 2017) August 22-25, 2017. University of Houston, USA.

[21] Elliot Varey, John Burrows, Jing Sun† and Sathiamoorthy Manoharan "From Code to Design: A Reverse Engineering Approach", IEEE sponsored International Conference on Engineering of Complex Computer Systems 2016 IEEE DOI 10.1109/ICECCS.2016.1.

[22] Michael John Decker1 , Kyle Swartz , Michael L. Collard , and Jonathan I. Maletic, "A Tool for Efficiently Reverse Engineering Accurate UML Class Diagrams", International Conference on Software Maintenance and Evolution, 2016 IEEE.

[23] A.B Amoore, M.A. Amodu Longe, "Functional Dependency: Design and Implementation of a Minimal Cover Algorithm" , IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 5, Ver. I (Sep.- Oct. 2017).

[24] Dr. Shivanand M. Handigund " An Ameliorated Methodology for the Abstraction and Minimization of Functional Dependencies of legacy 'C' Program Elements ", International Journal of Computer Applications (0975 – 8887) Volume 16– No.3, February 2011.