*International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)*
*Volume-3 Number-3 Issue-12 September-2013*

# Reverse Engineering: An Analysis of Static Behaviors of Object Oriented Programs by Extracting UML Class Diagram

**Mrinal Kanti Sarkar[1], Trijit Chatterjee[2], Dipta Mukherjee[3]**

## Abstract

***The Unified Modeling Language (UML) has been accepted as a standard for modeling object oriented system. It helps the designer to understand a problem well by focusing on one aspects of a problem at a time. In this paper we present a novel approach in which reverse engineering is performed and we have chosen UML as the modeling language to achieve a representation of the implemented system. In this work we have considered java programs. After a brief introduction to the subject, we present some analyses which go beyond mere enumeration of methods and fields. We sketch a method which determines classes and their attribute, operation and relationship: generalization, aggregation, association and various kind of dependencies in form of a simple class diagram that can be understood by a programmer when inspecting the source code of a given java programs. To fully understand the behavior of a system, it is crucial to have efficient techniques to reverse static views of the system. In this paper, we focus on the reverse engineering to find UML class diagram from an object oriented system and analysis of its static behavior.***

## Keywords

***Reverse Engineering; Static; OOP; UML; Class Diagram;***

## 1. Introduction

Reverse-engineering can help to understand a complex system by retrieving models and documentation from a program. Unified Modeling Languages (UML) offers different types of diagram so it is a good language for the reverse engineering.

**M. K Sarkar**, Department of Computer Science & Engineering, University of Engineering & Management Jaipur, Rajasthan, India.
**T. Chatterjee**, Department of Computer Science & Engineering, University of Engineering & Management Jaipur, Rajasthan, India.
**D. Mukherjee**, Department of Computer Science & Engineering, University of Engineering & Management Jaipur, India.

The static structure of a system comprises of a number of class diagrams and their relationship.
After analyzing lots of related works, it shows that there seems to be no solution for the reverse engineering of the more complex class diagram. In the present work, we outline a reverse engineering approach for UML specification in the form of class diagram from java programs to analyze its static behaviors.

Traditional software engineering research and development focuses on increasing the productivity and quality of systems under development or being planned. Without diminishing the importance of software engineering activities focusing on initial design and development, empirical evidence suggests that significant resources are devoted to reversing the effects of poorly designed or neglected software systems. In a perfect world, all software systems, past and present, would be developed and maintained with the benefit of well-structured software engineering guidelines. There are several systems which had negated their structured design but there must be tools and methodologies to handle these cases. Reverse Engineering is a methodology that greatly reduces the time, effort and complexity involved in solving the program comprehension problem. Reverse Engineering is best defined by Chikofsky and Cross [1] as "the process of analyzing a subject system

- To identify the system's components and their inter-relationships and
- To create representations of the system in another form or at a higher level of abstraction."

Reverse Engineering is a well-established practice in that there are numerous CASE tools available to map source code to good quality structural models. There are several tools which have been implanted using reverse engineering of UML static diagram. Static views of the system allow engineers to understand its structure but it does not show the behavior of the software. If we want to understand its actual behavior, we need dynamic models such as sequence diagrams or state charts diagram. Many works has been done on reverse engineering of UML class diagrams. Automatic reverse engineering of UML dynamic models [2] are describe by Y-G Guhenneuc and T Ziadi. Another approach by using state vectors through trace analysis [3] can get the dynamic views of a system. These CASE tools were being used alongside sequential programming languages like COBOL to maintain the design of documentation. The concept of Reverse Engineering emerged from the hardware world where hardware circuits were reverse-engineered to create clones. When the software engineers adopted the same term to describe some software engineering practices, there was a dearth of well-defined terminology to use for both technical and market-place discussions. A Library Information System (LIS) application which is being used extensively in all university's library in the world was chosen as a case study for the purpose of this work. The eLib program is a small Java program that supports the main functions operated in a library. It is not so easy to understand how the classes are organized, how they interact with each other to fulfill the main functions, how responsibilities are distributed among the classes, what is computed locally and what is delegated.In this paper we extract the static behavior of an object oriented system. We have chosen UML class diagram to understand the static behavior of an OOP.Our paper is organized as follows. In section 2, we present the concept of reverse engineering and analysis the difficulties of reverse engineering. Section 3 explores the class diagram and its perspective. In section 4, we address the design issues to extract class diagram from a java source code. We present our result in section 5. Finally, we give the concluding remarks of the whole paper and future scope in Section 6.

## 2. Reverse Engineering

### A. Reverse Engineering Defined

Chikofsky and Cross [1] made a very successful attempt at providing some precise and long standing definitions for much of the terminology used to this day in the field of Reverse Engineering. The following terms and definitions are adapted from the canonical taxonomy given in [1].

Forward Engineering: Forward engineering takes sequences from feasibility study through designing its implementation. We can conclude that the forward engineering is opposite of reverse engineering.

Reverse Engineering: It is a process by which one can identify and analysis the component of software's system, their inter-relationships and the representation of their entities at a higher level of abstraction [1]. Reverse engineering by itself involves only analysis; it does not involve changing the subject system or not create any new system [1]. It has been found that there are many sub-areas of reverse engineering. But re-documentation and design recovery are widely used in reverse engineering.

Re-documentation: It is another of reverse engineering and it involves creating or revising of system documentation at the same level of abstraction [1]. It is the simple and oldest form of reverse engineering and it gives you the alternate views of the system. The primary goals of these tools are to provide easier ways to visualize relationship among program component.

Design recovery: Using domain knowledge and other external information, system component can be re-documents where possible to create a model of the system at a higher level of abstraction [1]. According to Ted Bigger Staff "Design recovery recreates design abstraction from a combination of code, existing design documentation, personal experience, and general knowledge about problem and application domains. It must reproduce all of the information required for a person to fully understand what a program does, how does it, why does it, and so forth."

Re-structuring: The engineering process of transforming the system from one representation to another at the same relative abstraction level, while preserving the subject's systems external functional behavior [1]. Using re-structuring, we can maintain the code's structure in the sense of structured design.

Re-engineering: Re-engineering changes the functionality and direction of the system [1]. It involves a combination of reverse engineering for comprehension, and a re-application of forward engineering and maximize which functionalities that need to be retained, deleted or added [1]. It includes some form of reverse engineering followed by some form of forward engineering or restructuring.

### B. Objectives of Reverse Engineering

The primary purpose of reverse engineering a software system is to increase the overall comprehensibility of the system for both maintenance and new development. When we try to characterize reverse engineering in terms of its objectives, there are six key objectives (Chikofsky and Cross II 1990) in reverse engineering [1].Cope with complexity: We must develop method to better deal with the share volume and complexity of the system. A key to controlling these attributes is automated support.

Reverse Engineering methods and tools, combined with case environments, will provide a way to extract relevant information so decision makers can control the process and the product in the system.Generate alternate views: Graphical representation have long accepted as comprehension aids. However, creating and maintaining them continuous to be a bottleneck in the process. Reverse-engineering tools facilitate the generation or regeneration of graphical representation from other forms. Many designers work from a single, primary perspective (like dataflow diagrams), reverse-engineering tools can generate additional views from other perspective (like control flow diagram, structure chart, and entity relationship diagrams) to aid the review and verification process.

Re-cover lost information: Its recover the lost information of a large system by performing continuous evaluation. Modifications are frequently not reflected in the documentation, particularly at a higher level than the code itself. We cannot substitute for preserving design history in the first place; Design recovery is our way to salvage whatever we can form the existing system.

Detect side effect: Both haphazard initial design and successive modification can lead to unintended ramification and side effects that impede a system's performance in subtle ways.

Synthesize higher abstraction: Reverse-engineering requires methods and techniques for creating alternate views that transcend to higher abstraction levels. There is a debate in the software community as to how completely process can be automated. Clearly, expert-system technology will play a major role in achieving the full potential of generating high-level abstraction.

Facilitate reuse: A significant issue in the movement toward software reusability is the large body of existing software assets. Reverse-engineering can help detect candidates for reusable software components from present system.

### C. Difficulties in Reverse Engineering

Michael L. Nelson [5] has discussed detailed of the practical difficulties involved in the reverse engineering. The difficulties of reverse engineering show that there is different level of abstraction, and that is the formal/ cognitive distinction. Computers and programming languages are formal, while the human cognitive capabilities [6] are non-formal. Therefore, the result of any reverse engineering work could be very subjective. Any program is "understood to the extent that the reverse engineer can build up correct high level chunks from the low level details available in the program."

A discussion of how these difficulties are manifested to the reverse engineer follows. The choice of methodology, representation and tools used will define the usefulness of the derived reverse engineering information. The work done on integrating the top-down and bottom-up approaches to understanding a program to develop an approach, called the Synchronized Refinement is described. This approach is based on the detection of design decisions in the source code and the organization of the information into an information structure suitable for browsing by software maintainers. However, the process suggested is labor intensive, though the paper suggests that automating the individual tasks in the process can reduce the rigor involved. The author suggests that many of the activities described in the Synchronized Refinement methodology are automatable. He suggests that if a comprehensive information structure is populated with information about a program, different views of the system can be extracted from the database as required for understanding a particular part of the source code.

Currently, reverse engineering is currently heavily dependent on human interaction and memorization. While there are tools to assist the reverse engineer in program comprehension, it is not a fully automated process. The human element present in program comprehension is the subject of another field, software psychology [6]. Software psychology measures human performance while interacting with computer and information systems.

## 3. Class Diagram

Static behavior of a system can be represented by a class diagram. So, we represent the static view of an application using class diagram. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages. The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram.

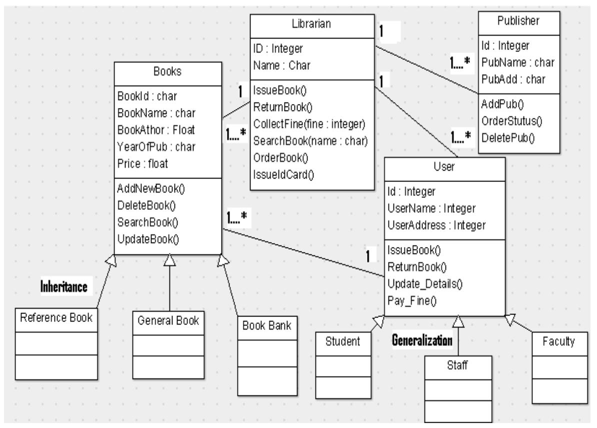


Fig 1: Class Diagram

Now the above diagram is an example of a Library Management System. So it describes a particular aspect of the entire application. A typical class diagram looks like the one in Fig 1. Here Class Library has three association relationship with classes User, Book and publisher. Class student, staff and faculty are inherited from base class User. Class reference book, general book and book bank are also inherited from Books, so there is a generalization.

## 4. Approach to Create Class Diagram

In this chapter we will address the design issues of the class diagram from a java source code. Some generic ideas imbibed from previous work and intuitive notation were decided upon initially to set up the environment for the reverse engineering effort. There are different steps to find out the class diagram from a java source code. So, we have different types of several artifacts in any software system like the source code, design documents, specification documents and the developer knowledge/experience that are the most vital importance for the Reverse Engineering effort. These are gathered together in an effort to build the knowledge base for the software system.

### A. Design of Class Diagram

Recovery of the class diagram from the source code is a difficult task. The decision about what elements to show/hide profoundly affects the usability of the diagram. A basic algorithm for the recovery of the class diagram can be obtained by a purely syntactic analysis of the source code, provided that a precise definition of an interclass relationship is given.

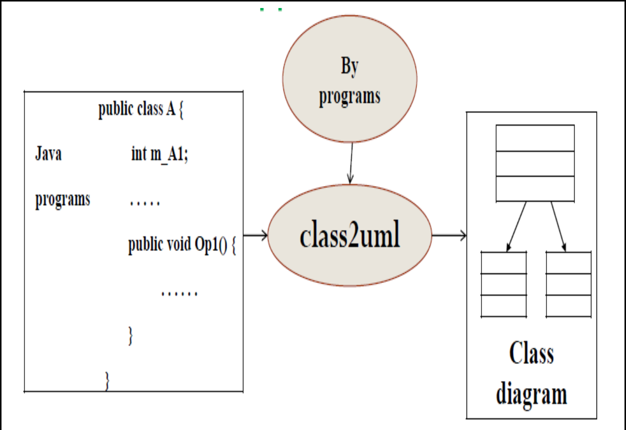


Fig 2: OOP source code to class diagram

For example, an association can be inferred when a class attribute stores a reference to another class. One problem of the basic algorithm for the recovery of the class diagram is that declared types are an approximation of the classes actually instantiated in a program, due to inheritance and interfaces. Aggregation, association and dependency relationships are displayed in a class diagram to indicate that a class has access to resources (attributes or operations) from another class. Now see how the design can be extended to get a class diagram from a java source code.

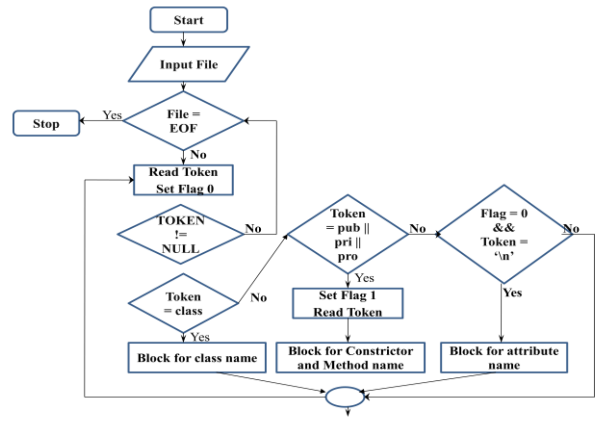
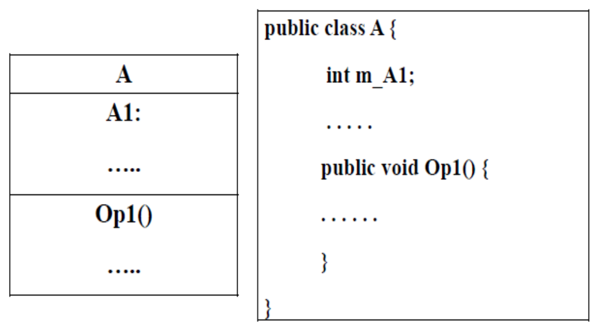


Fig 3: Control flow diagram to find class diagram

### B. Class with Attribute and Operation

Analyzing the syntax of the source code of our example Library, when we find the keyword class in the source code, after the class the name coming it must be the class name. Then first compartment below the class name shows the attributes. Class operations are in the bottom compartment. Here we also give a sample code and its class name, attribute and operation. If we get the keyword abstract before the keyword class, then the desired class is the abstract class. Here Fig 4, we give a sample example.



```
public class A {
        int m_A1;
        .....
        public void Op1() {
        ......
        }
}
```

**Fig 4: Model and code for class with attribute and operation**

### C. Association Relationship

Two classes are connected by a (bidirectional) association if there is the possibility to navigate from an object instantiating the first class to an object instantiating the second class (and vice versa). Unidirectional associations exist when only one-way navigation is possible. Navigation from an object to another one requires that a stable reference exists in the first object toward the other one. In this way, the second object can be accessed at any time from the first one. There are two types of association 1 to 1 and 1 to many. After analyzing the source code, here we give a sample code to finding 1 to 1 and 1to many associations.

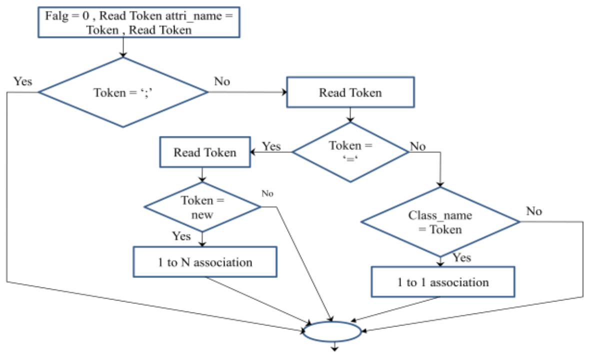


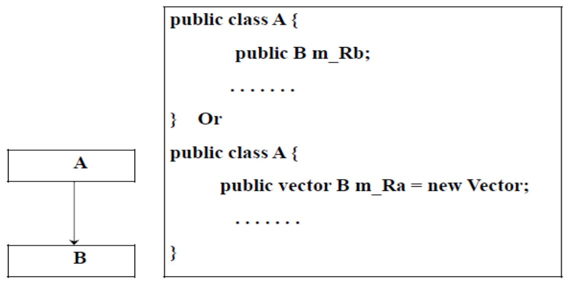
**Fig 5: Control flow diagram to find association**

### D. 1 to 1 and 1 to N Association

If a class attributes referencing other object then we find association relationship and if a class attribute referencing other object and we get the keyword new then we find 1 to N association or multiplicity relationship.

### E. Aggregation Relation

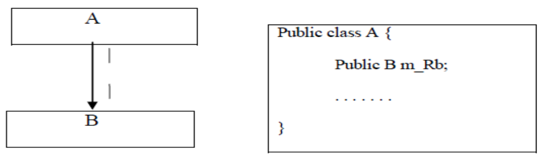
A class is related to another class by an aggregation relationship if the latter is a part-of the former. So, there is no substantial difference between aggregation and association. Both relationships are typically implemented as a class attributes referencing other objects.



```
public class A {
        public B m_Rb;
        .......
}   Or
public class A {
        public vector B m_Ra = new Vector;
        .......
}
```

**Fig 6: Model and code for 1 to 1Association**

Attributes of container type are used whenever the multiplicity of the target objects is greater than one. In principle, there would be the possibility to approximately distinguish between composition and aggregation, by analyzing the life time of the referenced objects. However, in practice implementations of the two relation variants have a large overlap.



```
Public class A {
        Public B m_Rb;
        .......
}
```

**Fig 7: Model and code for 1 to 1 Aggregation**

### F. Generalization and Realization Relation

Generalization is easily determined from the class declaration, by looking for the keywords extends. If we get extends keywords after the class name then class name is the subclass which got after extends, we get another class name that is the super class. The Fig 9 gives a sample code and its model of class diagram.

Realization is easily determined from the class declaration, by looking for the implements extends. If we get extends keywords after the class name then class name is the subclass and after the implements, we get another class name that is the super class. The Fig 9 gives a sample code and its model of class diagram.

## 5. Implementation Result of Class Diagram

The design of the class diagram tool has been implemented in g++ and jdk 1.6.0_16 in LINUX platform. We have implemented a C programs names main.c which creates a class diagram from any object oriented program.

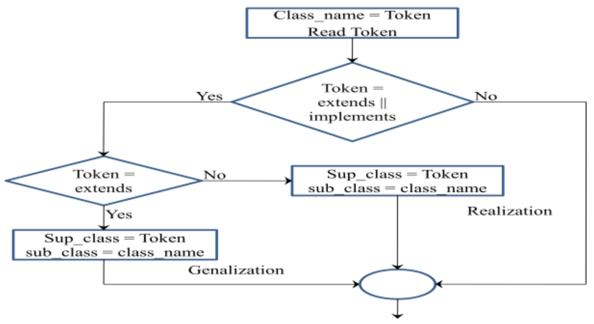
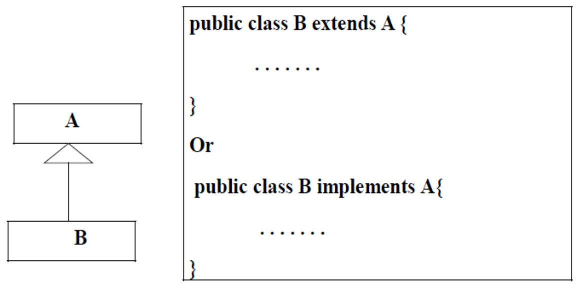


**Fig 8: Control flow diagram to find generalization and realization**

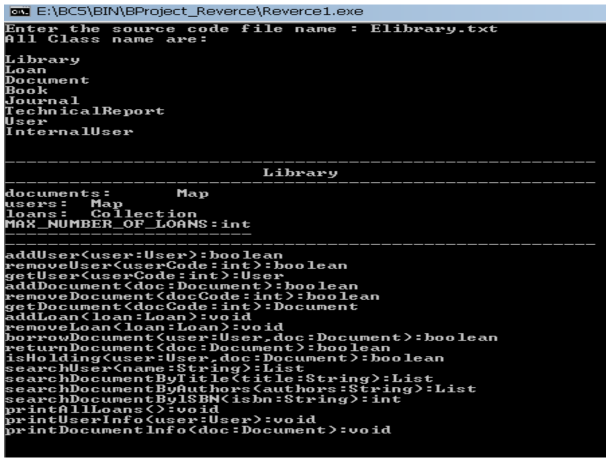


```
public class B extends A {
        .......
}
Or
public class B implements A{
        .......
}
```

**Fig 9: Model and code for Generalization and Realization**

We have been able to handle a simple class diagram. This class diagram only show you the various class name and their attribute name and operation. Our C program also finds the relation like association, generalization, and realization of the classes. Till now our program does not handle sequence, state diagram and other model of the UML diagram. Thus our reverse engineering tool can work only simple programs.

**Results**



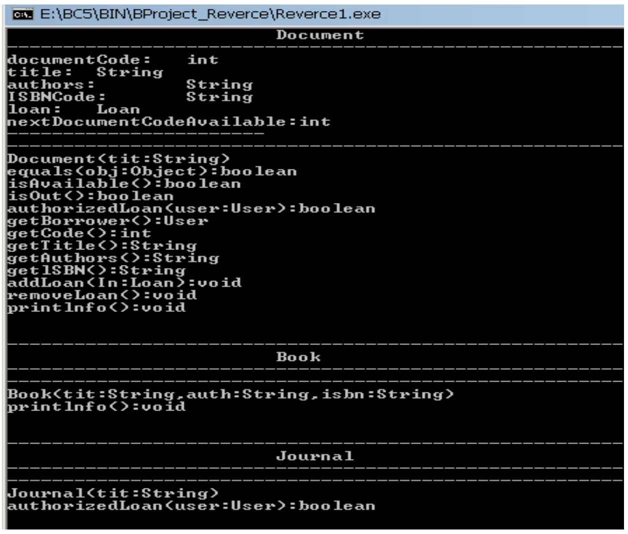
```
E:\BC5\BIN\BProject_Reverce\Reverce1.exe
Enter the source code file name : Elibrary.txt
All Class name are:

Library
Loan
Document
Book
Journal
TechnicalReport
User
InternalUser

------------------------------------------------------------
                          Library
------------------------------------------------------------
documents:        Map
users:   Map
loans:   Collection
MAX_NUMBER_OF_LOANS:int
------------------------------------------------------------

addUser(user:User):boolean
removeUser(userCode:int):boolean
getUser(userCode:int):User
addDocument(doc:Document):boolean
removeDocument(docCode:int):boolean
getDocument(docCode:int):Document
addLoan(loan:Loan):void
removeLoan(loan:Loan):void
borrowDocument(user:User,doc:Document):boolean
returnDocument(doc:Document):boolean
isHolding(user:User,doc:Document):boolean
searchUser(name:String):List
searchDocumentByTitle(title:String):List
searchDocumentByAuthors(authors:String):List
searchDocumentByISBN(isbn:String):int
printAllLoans():void
printUserInfo(user:User):void
printDocumentInfo(doc:Document):void
```

**Fig. 10: Result 1**

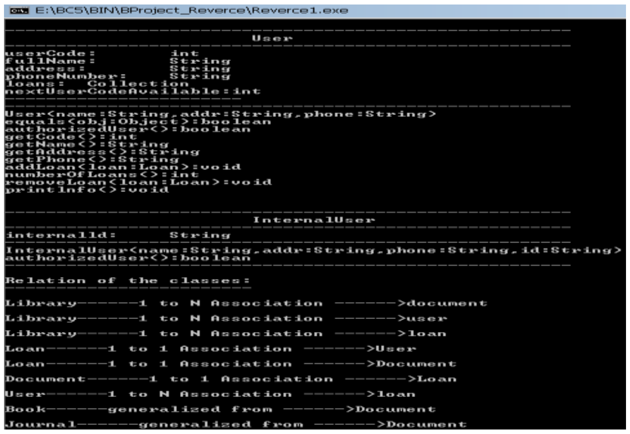


```
E:\BC5\BIN\BProject_Reverce\Reverce1.exe
                          Document
------------------------------------------------------------
documentCode:    int
title:  String
authors:         String
ISBNCode:        String
loan:    Loan
nextDocumentCodeAvailable:int
------------------------------------------------------------
Document(tit:String)
equals(obj:Object):boolean
isAvailable():boolean
isOut():boolean
authorizedLoan(user:User):boolean
getBorrower():User
getCode():int
getTitle():String
getAuthors():String
getISBN():String
addLoan(In:Loan):void
removeLoan():void
printInfo():void

------------------------------------------------------------
                          Book
------------------------------------------------------------
Book(tit:String,auth:String,isbn:String)
printInfo():void

------------------------------------------------------------
                          Journal
------------------------------------------------------------
Journal(tit:String)
authorizedLoan(user:User):boolean
```

**Fig. 11: Result 2**



```
E:\BC5\BIN\BProject_Reverce\Reverce1.exe
------------------------------------------------------------
                          User
------------------------------------------------------------
userCode:          int
fullName:          String
address:           String
phoneNumber:       String
loans:   Collection
nextUserCodeAvailable:int
------------------------------------------------------------
User(name:String,addr:String,phone:String)
equals(obj:Object):boolean
authorizedUser():boolean
getCode():int
getName():String
getAddress():String
getPhone():String
addLoan(loan:Loan):void
numberOfLoans():int
removeLoan(loan:Loan):void
printInfo():void

------------------------------------------------------------
                          InternalUser
------------------------------------------------------------
internalId:      String
------------------------------------------------------------
InternalUser(name:String,addr:String,phone:String,id:String)
authorizedUser():boolean
------------------------------------------------------------
Relation of the classes:
------------------------
Library-------1 to N Association ------->document
Library-------1 to N Association ------->user
Library-------1 to N Association ------->loan
Loan-------1 to 1 Association ------->User
Loan-------1 to 1 Association ------->Document
Document-------1 to 1 Association ------->Loan
User-------1 to N Association ------->loan
Book-------generalized from ------->Document
Journal-------generalized from ------->Document
```

**Fig. 12: Result 3**

## 6. Conclusion

Obviously, it is very hard to implement UML structural view from an object oriented programming languages. Software reverse engineering, or program comprehension, is the difficult task of recovering design and other information from a software system. The formal process described but does not have any automated approaches involved at this point, but the emergence of such tools is expected from the tools industry soon. Now our work consists in generating a simple class diagram. Till now we are able to handle a simple class diagram. Though static analysis is not sufficient to analyze or understand a java source code properly. So we need dynamic analysis like sequence diagram, state chart diagram, activity diagram. While implementing our design we have only handled a small subset of the languages. Here escape character are not recognized. Method calls within a method calls also ignored. The more complicated features of the programs have to be implemented. A good graphical user interface is to needed to view the all diagrams. In future extracting of aggregation, composition, dependency which require more advanced technology. The main key issue of reverse engineering is to discover the abstraction of java source code.

## References

[1] E. J. Chikofsky and J. H. Cross, II, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, vol. 7, no. 1, pp. 13-17, January 1990.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Michael L. Nelson "A Survey of reverse Engineering and Program comprehension" April 19, 1996.

[3] Yves Le Traon, Benoit Baudry and Romain Delamare "Reverse Engineering of UML 2.0 Sequence Diagram from execution Traces" French national institute for research in computer science April, 2006.

[4] Stan Jarzabek and Guosheng Wang, "Model Based Design of Reverse Engineering Tools," Software Maintennance: Research and Practice, J. Softw. Maint: Res. Pract.10, 353-380 (1998).K. Elissa, "Title of paper if known," unpublished.

[5] Frank Tip, "A survey of Program Slicing Techniques", Journal of Programming languages, Sept. 1995.

[6] Ben Shneiderman , "Software Psychology: Human Factor in Computer and Information System", Little Brown and Co., Boston,Massachusetts, 1980.

[7] "Application Reengineering", Guide Pub, GPP-208, Guide Int'l Corp., Chicago, 1989.

[8] R. Mall, "Fundamentals of Software Engineering", Prentice-Hall, India, 2nd Edition, August, 2008.

[9] Nicola Howarth, "Abstract Syntax Tree Design" ANSA Phase III, 23rd August, 1995

[10] Romain Delamare, Benoit Baudry, Yves LeTraon, "Reverse engineering of UML 2.0 Sequence Diagrams from Execution Traces" " French national institute for research in computer science and control, April, 2006

[11] D. Kornack and P. Rakic, "Cell Proliferation without Neurogenesis in Adult Primate Neocortex," Science, vol. 294, Dec. 2001, pp. 2127-2130, doi:10.1126/science.1065467.

[12] F.G. Pagan, "Partial Computation and the Construction of Language Processors", Prentice Hall, 1991

[13] Y.-G. Gu´eh´eneuc and T. Ziadi, "Automated reverse-engineering of UML 2.0 dynamic models", Proceedings of the 6th ECOOP Workshop on Object-Oriented Reengineering, 2005.

[14] Object Management Group website http://www.omg. org/uml

[15] Rational Corporation website http://www.rational.com

[16] H. Goto, Y. Hasegawa, and M. Tanaka, "Efficient Scheduling Focusing on the Duality of MPL Representatives," Proc. IEEE Symp. Computational Intelligence in Scheduling (SCIS 07), IEEE Press, Dec. 2007, pp. 57-64, doi:10.1109/SCIS.2007.357670.

**Mrinal Kanti Sarkar** received his B.Tech degree in Computer Science & Engineering from Govt. College of Engineering & Ceramic Technology and M.Tceh degree in Computer Science & Engineering from Indian Institute of Technology, Kharagpur. He has worked as a Senior Lecturer at The ICFAI University Tripura. Now, he is working as an Assistant Professor at University of Engineering and Management Jaipur, India.

**Trijit Chatterjee** received his B.Sc. degree in Computer Science(Hons) from Calcutta University and M.Sc. degree in Computer Science from St. Xavier's College, Kolkata, under Calcutta University, Calcutta, India. Now, he is working as an Assistant Professor at University of Engineering and Management Jaipur, India.

**Dipta Mukherjee** received his B.Tech and M.Tech degree in Computer Science & Engineering from Kalyani University, West Bengal. He has worked as an Assistant Professor at Institute of Engineering & Management Kolkta. Now, he is working as an Assistant Professor at University of Engineering and Management Jaipur,India.