"Reverse Engineering": Extracting Information from C++ code

Ati Jain
Dept. of Information Tech.
Mahakal Inst. Of Tech.
Ujjain, INDIA
swapnil.soner@gmail.com

Swapnil Soner
Dept. of Information Tech.
Mahakal Inst. Of Tech.
Ujjain, INDIA
jain.ati@gmail.com

Aishwarya Holkar
Dept. of Computer Sc & Engg
Swami Vivekanand College of Engg
Indore, INDIA
er.aishwary_holkar@yahoo.in

***Abstract*— This paper presents an experiment to reverse engineer a legacy source code. Here, it is used as a part of a project to develop and upgrade the existing system written in C++ language. To accomplish this task some information's must be need to be extracted from that legacy C++ code. The extracted information now used to implement a new version of the design documentation. This experiment revealed issues about recovering design information, such as, knowing variables, functions and classes. The reverse engineering process used to recover the Data flow diagrams, Control flow diagrams and class diagrams and the experience gained during the study are reported.**

Keywords— Extraction, Legacy code, Data Flow Diagrams, Control Flow Diagrams, Class Diagrams

I. INTRODUCTION

Due to rapid advancements in software industry, structure of a software system progressively degrades. Thus maintenance of large and a critical legacy system have become hard. A legacy system is one, which is extremely valuable to an organization, performing key strategic functions. But maintaining such systems as to incorporate new functionalities or due to organizational policy changes become hard in the absence of proper documentation, qualified staff and other resources [5].

The cost of re-engineering a system is generally less than developing a new system. Sometimes what is required is, add some functionalities, change some policies, change the structure of system without changing functionalities, changing the architecture of the system to add some non functional requirements and for making these changes, it is worthless to develop a new system. But still it is a mater of debate that whether to go for forward engineering or for re-engineering.

The focus of this paper is intended to develop a Reengineering method to automate the extraction of DFDs, CFDs and class diagrams from any legacy C++ code. Extracted information will be classified as structural, behavioral and constraint rules through which such information can be produced. In this, a C++ program will be taken as input and than blanks and comments are removed and the variables are identified. In next step we try to find various classes and its dependency [6].

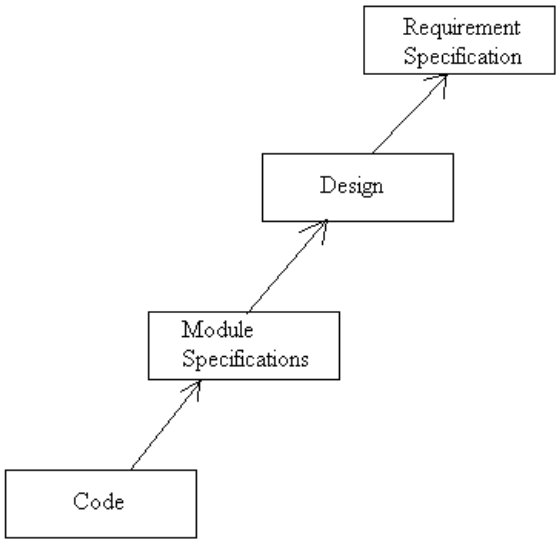

Figure 1. Reverse Engineering

II. LEGACY SYSTEM AND CODE

With age, legacy systems represent significant corporate knowledge, including requirements, design decisions, and business rules. The number of large systems being built from scratch decreases, the number of legacy systems increases. These assets can be used effectively if a systematic strategy for the continued evolution of legacy systems to meet changing mission, technology and user needs is evolved [4].

The software portion of a legacy system may have been written many years ago, developed using primitive and unplanned methodologies, and subjected to elongated maintenance. The result is a legacy system that lacks the ability to acquire the ever-changing demands in a cost-

978-1-4244-8666-3/10/$26.00 © 2010 IEEE

effective manner. Here we are trying to use legacy code in order to get the design documentation.

III. REVERSE ENGINEERING PHASES

THERE ARE FOUR PHASES MAINLY [5]:
Component ANALYZING
DESIGN recover
DESIGN reconstructing

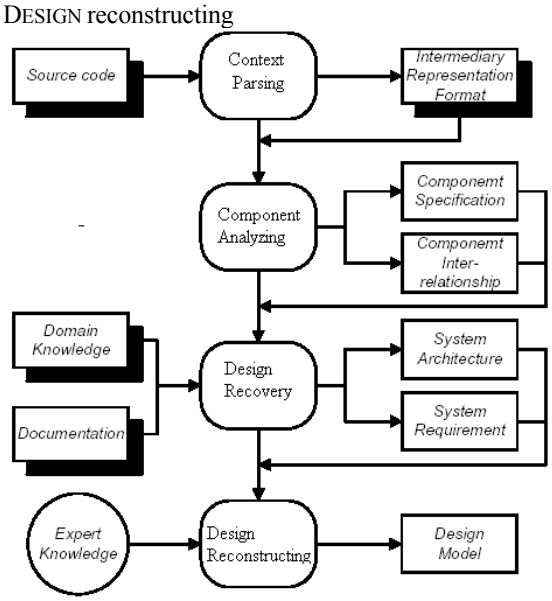

Figure 2. Re-engineering Process

A. Context Parsing Phase

This is the first phase, analyses the source code along with which extracts syntactic and semantic program information. These techniques generally used in compilers such as token parsing and syntax analysis can be adopted. In this context parsing phase, it generally parses source code and then transforms the parsed code into a more structured representation as an intermediate form.

B. Component Analyzing Phase

Various tools are available for this phase, which can easily extract artifacts like structure chart, variables, attributes, functions, program slices, call graph, data flow graph and control dependencies.

C. Design Recovery phase

Extracting original requirements is a tough job and this requires domain knowledge. So, generally all available techniques use structural and knowledge representation to get some high level information. Recovered domain knowledge plays an important role in:

- Quick understanding of the background of the application in case of lost or inconsistent document.
- And can be reused to develop new application.

D. Design Reconstructing phase

To full fill the task completely, the process of reverse engineering introduces a fourth phase, the design reconstruction phase. In this phase, system models and design specifications gained from the last phase can be further examined and integrated to reconstruct a short view of the model.

This model offers not only little functionality and system behavior, but also the correct architecture and design that could be in the original implementation that results in re-development and destined work.

System model and design specification will be examined and integrated to reconstruct precise view of the design model, which will help in maintenance and re-development process. For component analysis we will require design, architecture, specification and program understanding, as we know the function of the component. But to recover the design we first have to understand the functionality of the system and then only we can proceed further. As we know what to re design so for reconstruction of design recovery tool is not required.

IV. REVERESE ENGINEEERING PROCESS

The reverse engineering process begins by extracting detailed design information that extracting a high-level design abstraction. Detailed (low-level) design information is extracted from the source code and existing design documents. This information includes variables, Class Diagrams, Class dependencies, structure charts, data descriptions and PDL to describe processing details. This high-level design representation is extracted from recovered detailed design and expressed by data-flow diagrams and control-flow diagrams. The system understanding is the most important application in the field of reverse engineering [2]. The procedure steps are discussed below:

A. Removing Header files and Comments

C++ program can be made which basically performs the parsing function it extracts the comments along with the line from the code and stores it into a separate file so that it could be used further. It also removes the header file and stores rest of code in a file with each statement along with a unique identifier.

Code class shows the contents of extracted file. Comment class is used to display the comments associated with a line. If no comment found displays the message accordingly.

If there is no comment associated with the line it will show “NO COMMENTS FOUND”. This an important feature of the tool as comments if they are there in code helps in better understanding of code. This will help a user in identifying a best candidate variable for extracting the code. But in the absence of comments associated with lines and no naming convention used it will be difficult for user to guess a right variable.

This can be found for example by:
void CommentAndHeaderRemoval();

1) Removing comment of type (//)
2) Removing comment of type (/* */)

3) Removing header file (#)

These are some techniques that can be used in order to identify headers and comments by finding their identifiers.

B. Variable Identification

First of all, Input output variables should be identified those are involved in implementation of a business logic. Though identification of a right output variable is not an easy task, but how good, we can make about the output variable more accurate results we will get. Arguments passed to a procedure are the best candidate variables for extraction of a business rules as generally these arguments contain information on which that particular procedure perform or implement a business rule [1].

Identification of variables through the C++ code can be done as:

```
void FindVariables()
{
char ch,s[10]; /*variables : ch=current
          character,s:temp string that
          stores INT,FLOAT,CHAR keywords
          for comparison*/
char ia[100];   // integer array (stores
integer variables)
char ca[100];   // character array
char fa[100];
int flag=0,j,fch=0,ff=0;     //variables
for counting
/*Here
 flag: counts integer tokens & indexes
array ia(integer array) declared in the
beginning.
 fch :    counts character tokens &
indexes ca
 ff  :  counts Float tokens & indexes fa
*/
int m,k,out[3];
//out[3]: stores all three flag, fch, ff
& is passed to function output.
float g;
float asdf;
int abc;char def;float ghi;
}
```

As in above code Integer, Float and Character variables can be founded out by writing the appropriate C++ code, where any C++ file can be given as input and identification of the variables can be found, which can be further used to implement a business rule. But best candidate variable out of these is one, which contains a result of some calculation. However these data items may come from any source-derived from database or from user input.

The biggest problem in identifying a variable is that it is not necessary that target system is well structured and all the naming conventions have been used. In such circumstances we will provide the user with a list of all probable input and output variables so that user or expert can select one and try to find out if there is any rule implemented with that data or not. Along with that we also provide user a facility to store the description with a variable if needed.

Other information with variable lists can be found which may be:

- Data Flow Diagram
- Control Flow Diagram

C. Data Flow Analysis

The recovered variables can be analyzed to identify data transformation in the C++ code. This transformation shows the data processing done in a program. These transformations are used to develop a set of hierarchical data flow diagrams that model the software, which is step to develop a data-flow model.

Structured design gives guidelines for transforming data-flow diagrams into structure charts. To some extent, these guidelines can be reversed. The idea is to identify transformations on data expressed in the detailed design and use these transformations to create processes in a data-flow model. The order of transformations in the detailed design determines the order in the data-flow model. The data-flow model consists of a hierarchy of data-flow diagrams. The top-level diagram shows a single process representing the program and its input sources and output destinations. The process in the top-level diagram is then expanded in a separate diagram. This second level diagram shows processes that together do the transformation attributed to the top-level process. Each of these processes can be expanded into lower level diagrams that show even more detailed transformations.

Data-flow diagrams were constructed from the recovered detailed design using a procedure. In this procedure, data transformations in the detailed design are identified using a two-step process. In the first step, each node in the structure chart becomes a process in a data-flow diagram. The names given to processes were not the names from the structure chart nodes. Instead, descriptive process names that suggested the task of a process were used. Here step creates a hierarchy of data-flow diagrams. In the second step, the PDL associated with each structure chart node is examined and data transformations are identified and such transformations are then summed to the appropriate data-flow diagram as new processes with descriptive names. In this fashion all data transformations present in the detail design are captured in the data-flow model.

Recovery of the DFDs forced an analysis of the data items exchanged between nodes in the structure chart. Individual data items carry either data information or control information, or in a poorly designed system both. A data-flow model is concerned with modeling transformations on data and does not express control information. Data items used solely to carry control information were extraneous to the model and therefore were not recovered.

D. Control Flow Analysis

Identify the high-level control structure of the program and record it using control-flow diagrams. This refers to high-level control that affects the overall operation of the software, not to low-level processing control.

This step was to identify the control structure of the software. The task of uncovering this structure occurred while creating the data-flow diagram. What remained was to record the control structure and explain its purpose. The control structure was recorded using control-flow diagrams, which were overlaid on the data flow diagrams. This step allowed both data and control-flow to be visible on the same diagram. In addition, a state control model was defined and recorded using a state diagram.

A problem during this step was distinguishing between low-level control structure that involved the implementation of a routine and high-level control structures that served to control the software operation. The former should be included as part of the processing described in the detailed design, the latter needs to be recorded in a control flow diagram and its control specification. The temptation is to recover too much of the control structure.

E. Identifying Functions

After finding all the above steps, now find out the number of functions that are present in the code. Functions are nothing but these are the methods, which is generally declared in the source code. These functions can be VOID, INT, and CHAR type. These are the basically methods which is declared and used in the programs. This step puts an effort to know the methods by identifying its return type and "( )". These are the two main identifiers or keywords through which identification of the methods can be made. This identification and retrieval method of helps in deriving UML (Unified Modeling Language) diagrams like Use Case diagrams which consist of actors and functions that are present in any system and class diagrams which shows the different classes that are present in the system's source code and the various types of the dependencies between those classes.

Function identification is not an easier task in case if INT type. Due to the fact that return type INT can be return type of variable as well as functions. So sometimes it overrides the function.

F. Class Diagrams and its dependencies

Similarly identifying the "class" keyword from the input program code can redesign class diagrams for any C code. Finding of classes is an easier task but identify the relationship between two classes i.e. knowing the class dependency is not an easier task [3]. Class diagrams includes:

Name of the class at the top of the block, number of methods and number of variables that are present in that particular class. In same manner blocks are made similarly for all the classes and then further class diagrams can be made to complete by showing the class dependencies. These dependencies are based on some relationships.

Various types of class relationships can be:

1) *Association*
2) *Aggregation*
3) *Inheritance*
4) *Dependency*
5) *Multiplicity*

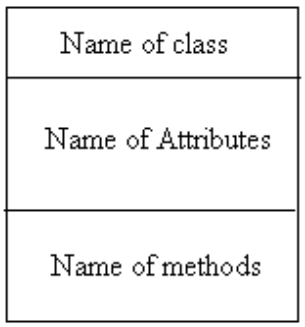

Figure 3. Block Representing Class

Now, the question arises how to find out class dependencies in any C++ code. Since the number of classes can be identified by:

void getclass(); //identify class keywords

The solution for this question can be given by identifying various certain methods.

Inheritance: Inheritance between these classes if exist can be taken out by matching and showing PUBLIC, PRIVATE, PROTECTED keyword with respect to the class. In the similar manner relationship like association, aggregation can be finding out by some ways.

Aggregation: In C++ aggregation is achieved by calling the object of one class in another class. For en example if there is a class 'A' and another class B, then aggregation is achieved bydeclaring object of class B refer to the class 'A'.

```
class A
{
/* Body of class A*/
 };
class B
{
       /*Body of class B*/
     A obj=new A();  /*Declaration of
                 object of class A*/
};
```

The logic to find aggregation can be implemented by storing the names of the classes in a table with the help of the SQL-LITE. Now we know that the class body starts with '{' and class definition ends with the '}' and ';'.Now each line is scanned and compared with the values of the table. Now if the scanned statements if matches with classes names stored in the table that means that the object of the stored class is called within this class. So it confirms that aggregation exits.

Association: An association between two classes is a type of a link between the corresponding objects. A (two-way) association between classes A and B describes a relationship between each object of class A and some objects of class B, and vice versa.While in association object's lifetime are not dependent they are more over interdependent to each other. One more thing, which we can say that same employee object cannot assign to different company.

For example:

```
class A
{
private:
```

```
            B* itsB;
    }
```

ACKNOWLEDGMENT

We would like to thank the all faculty members of the institute, our head of the department Prof. Anshu Tripathi, my project guide Prof. Abhishek Singh Rathore who helped us lot in calculating the facts and figures related to our paper. I would also like to thank Prof. J. K. Khatwani, Mr. Samay Mahajan and the anonymous reviewers who provided helpful feedback on my manuscript..

REFERENCES

[1] X. P. Chen, W. T. Tsai, J. Joiner, H. Gandamaneni and J. Sun, "Automatic Variable Classification for COBOL Programs", Proc. of IEEE COMPSAC, 1994, pp. 432-437.

[2] J. K. Joiner, W. T. Tsai, X. P. Chen, ***S.*** Subramanian, J. Sun and H. Gandamaneni, "Data-Centered Program Understanding", Proc. of IEEE Software Maintenance, 1994,

[3] C. L. Ong and W. T. Tsai, "Class and Object Extraction From Imperative Code", Journal of Object-Oriented Programming, March/April 1993, pp. 58-68.

[4] Joseph M Scandura ―Converting Legacy Code into Ada University Of Pennysilvania and intelligent Micro Systems, Inc. April 1994.

[5] Chih-Wei Lu, William C. Chu, Chih-Hung Chang, Yeh-Ching Chung, Xiaodong Liu, Hongji Yang: ―Reverse engineering, Handbook of Software Engineering and Knowledge Engineering, 2002.

[6] Chia-Chu Chiang; Bayrak, C. ―Legacy software Modernization Systems, Man and Cybernetics, 2006. SMC apos; 06. IEEE International Conference on Volume 2, Issue , 8-11 Oct. 2006 Page(s):1304 – 1309.