International Journal of Scientific Engineering and Technology (ISSN : 2277-1581)
www.ijset.com, Volume No.1, Issue No.2 pg:14-16 01 April 012

REVERSE ENGINEERING APPROACH TO INSTATEMENT OF DESIGN ARTIFACTS

Ms.Parul Dongre *, Mr.Arvind Upadhyay *, Mrs Namrata tapsavi *

* Department of Computer science, IES IPS Academy Indore

parul.dongre@gmail.com
upadhyayarvind10@gmail.com
namrat_tapsavi09@gmail.com

ABSTRACT

Software engineering concerned with improving the productivity of the software development process and the quality of the systems it produces. However, in current scenario, the most of the development effort is spent on maintaining existing systems rather than developing new ones. This paper describes wide research program which is present in the area of reverse engineering, tools developed for reversing and also discuss the concepts of new tool generation.
Fjeldstad and Hamlen report says that 47% to 62% of time spent on actual enhancement and correction tasks respectively, and devoted to comprehension activities. These involve reading the documentation, scanning the source code, and understanding the changes to be made. The implications are that if we want to improve software development .we should look at maintenance, and if we want to improve maintenance, we should facilitate the process of comprehending existing programs. Reverse engineering provides a direct attack on the program comprehension problem.
Here, we are discussing and study the various tools evolve in the field of reverse engineering and discussing of concept to create a new tool.

Keyword: Forward engineering, legacy code, Reengineering, Reusability, Software maintenance.

I. Reverse engineering- a brief overview

Reverse engineering is a process where an engineered artifact (such as a car, a jet engine, or a software program) is deconstructed in a way that reveals its innermost details, such as its design and architecture. This is similar to scientific research that studies natural phenomena, with the difference that no one commonly refers to scientific research as reverse engineering, simply because no one knows for sure whether or not nature was ever engineered.

In the software world reverse engineering boils down to taking an existing program for which source-code or proper documentation is not available and attempting to recover details regarding its design and implementation. In some cases source code is available but the original developers who created it are unavailable. Therefore we can say that Reverse engineering is usually conducted to obtain missing knowledge, ideas, and design philosophy when information is unavailable. In some cases, the information is owned by someone who isn't willing to share them. In other cases, the information has been lost or destroyed.

Software reverse engineering requires a combination of skills and a thorough understanding of computers and software development, but like most worthwhile subjects, the only real prerequisite is a strong curiosity and desire to learn. Software reverse engineering integrates several arts: code breaking, puzzle solving, programming, and logical analysis.

The process is used by a variety of different people for a variety of different purposes.
Chikofskys and Cross [8] give the following definition. "Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction." The purpose of reverse engineering is to understand a software system in order to facilitate enhancement, correction, documentation, redesign, or reprogramming in a different programming language.

Figure 1 contains a graphical depiction of a process model for reverse engineering and reengineering [1]. The process model is captured by two sectioned, where each section represents a different level of abstraction. The higher levels in the model are *concepts* and *requirements*. The lower levels include *designs* and *implementations*. Entry into this reengineering process model begins with system, where *Abstraction* (or reverse engineering) is performed to a level of detail appropriate to the task being performed. For instance, if a system is to be reengineered in response to efficiency constraints, then abstraction to the design level may be appropriate. The next step is *Alteration*, where the system is configured into a new form at a different level of

abstraction. Finally, *Refinement* of the new form into an implementation can be performed to create system.

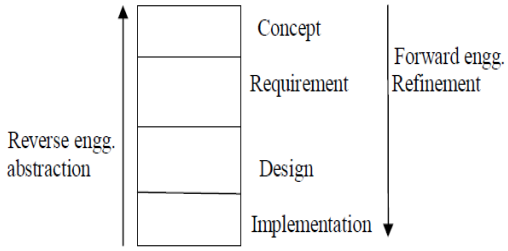

Figure 1
Level of abstraction

Several terms are frequently used in the discussion of software reengineering [ChiBO]. Software reengineering is the alternation of a system to reconstitute it in a new form, which potentially involves changes at the requirements, design, and coding phases. The activity generally includes reverse engineering followed by forward engineering or restructuring. Reverse engineering is the process of analyzing a system to extract design artifacts or abstractions that are less implementation-dependent.

Forward engineering is the traditional software development process of moving from requirements, design and coding. Restructuring is the transformation from one representation to another at the same abstraction.

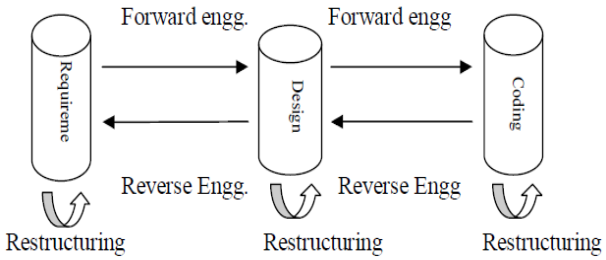

Figure 2

II. Reversing tools

Reversing is impossible without the right tools. There are hundreds of different software tools available out there that can be used for reversing. Understanding the differences between these tools and choosing the right ones is critical.

C++Code Crawler, is there which is used by many major platforms. The important thing is that this is free, language independent reverse engineering tool which combine metrics with software. Visualization Source Publisher is also there which works on windows and UNIX also produces PDF and HTML documentation from C/C++ and Ada source code. It Colorizes code, adds structure bars for easy sorting out of nested control structures.

There is jGRASP but restricted to java Produces Control Structure Diagrams for Java, C, C++, Complexity Profile Graph diagrams for Java and Ada, UML diagrams for Java and has an integrated debugger and workbench for Java. RoboHelp is also there for windows generates documentation from source code (C, C++, Visual Basic, Java, JavaScript, Delphi), Word documents, etc. to WebHelp, HTML Help, HTML, JavaHelp, WinHelp. WinA&D, MacA&D/Translator is there on Mac, HP and Sun UNIX, Windows platform for reverse engineering. Rational Rose is there on to work for SPARC, RS/6000, Windows, OS/2 to use for reverse engineering C/C++, Ada code. Other than this SoDa, Imagix 4D, CDOC-suite used respectively for automatically creates documents from Rational Rose OO models, C/C++ reverse engineering, C/C++ reverse engineering: control-logic, caller/called hierarchy.

Concept

III. Analysis for tool development

We will actually design a parser that will implement the concept of reverse engineering and accordingly reversely engineer the input java code & generate the respective design artifacts like class, sequence, activity and use case diagram. In order to implement the concept of reverse engineering we firstly allow the user to import the desired input java code into the interface (Input Screen where he can import the java code from the stored path) and then step by step generate the class diagram first after that we proceed towards the sequence, activity and use case diagram.

Here we define the source path of the input java code file. After defining the source path of the java code and importing it the actual process starts.

Class Diagram Generator: The first solution came in the form of "Class Diagram". Here there is another interface by which the user can define the setting parameters for the class diagram.

Sequence Diagram Generator: After successfully generating the class diagram the next will be to generate the desired sequence diagram. Here we used parsing technique to generate the sequence diagram.

Activity Diagram Generator: The next step is for the generation of activity diagram. For this we need to extract the loops and operators used in various functions of the classes. Then we can find the various activities.

Use Case Diagram Generator: After activity diagram, the next step is to generate the use case diagram of the java code. This is a very crucial step. Use Case diagram successfully depicts all the functionalities as well as the interaction between various actors.

IV. System architecture

Our parser is Java based software to recover the high level UML design patterns (design artifacts) from java source file only. Through these design models the code analysis becomes easier. Our complete application is implemented in three phases.

A. Source code analysis
B. Tokenizing source codes
C. Generation of design artifacts

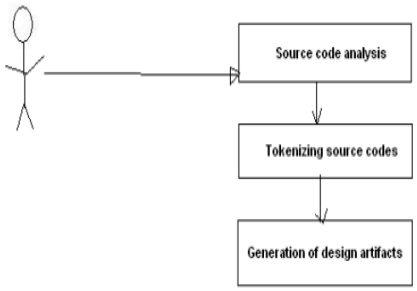

Figure 3

A. Source Code Analysis: In this phase we get the user input in the form of the java source code file and after that we start analysis of this input source code. By reading the source code file and prepare list of all files related to the particular source code file we the found dependencies, functions, various child classes, member variables and their return types.

B. Tokenizing Source Codes: In this phase we get the input from the system (the output of the pervious phase). We get a list of all members and list of relationships between them. Here we are creating a list of nodes and edges that will describe the actual dependency between the members.

C. Generation of Design Artifacts: The result of the second phase acts as an input in this phase to generate design artifacts. For that we use the graphics to map the members name, operations and the attributes as node.

Conclusion

Large software system requires more maintenance effort then smaller system. Large system is more complex in terms of variety of function they perform, so it requires great program understanding capabilities. For program understanding we can use various kinds of tools which support reverse engineering to recover lost information, for improve documentation, to provide alternate view, to extract reusable components and to detect side effects. Here we discuss a new tool for java source code which at higher-level of abstraction visualise the complete software by making class diagram, sequence, activity and use case diagram.

References

1. Chia-chu chiang and Roger Y. Lee, Developing tool for reverse engineering in a software product line architecture, IEEE conference 2004.

2. Leon moonen and Tarja systa, International workshop on reverse engineering models from software artifacts, 16th working conference on reverse engineering, 2009.

3. G. C. Gannod, Y. Chen, and B. Cheng. An automated approach for supporting software reuse via reverse engineering. Technical Report MSUCPS: TR98-16, Michigan State University, Department of Computer Science and Engineering, May 1998.

4. Barry W. Boehm, Software Engineering Economics, Prentice Hall, 1981.

5. R. K. Fjeldstad and W. T. Hamlen., ‘‘Application Program Maintenance Study: Tutorial on Software Maintenance, G. Parikh and N. Zvegintozov, editors, IEEE Computer Society, April 1983, IEEE Order1 No. EM453.

6. E. J. Chikofsky and J. H. Cross. Reverse Engineering and Design Recovery: Taxonomy. IEEE Software 7(1):13– 17, January 1990.

7. Reversing: The secret of reverse engineering, Eldad EilamCopyright 2005 by Wiley Publishing, Inc., Indianapolis, Indiana ISBN-10: 0-7645-7481-7 ,ISBN-13: 978-0-7645-7481-8.

8. Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software, Lionel C. Briand, Senior Member, IEEE, Yvan Labiche, Member, IEEE, and Johannes Ledu ,IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 32, NO. 9, SEPTEMBER 2006