International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 3, ISSN 2249 –071X

Generates UML Diagrams From Java Code Through Reverse Engineering

Ms. Namrata Sharma
PG_Scholar SSSIT, Sehore
E-mail: namarata_28@yahoo.com

Mr. Gajendra Singh Chandel
Prof. (CSE/IT) SSSIT, Sehore
Email:gajendrasingh86@rediffmail.com

Abstract **- Reverse Engineering is focused on the challenging task of understanding legacy program code without having suitable documentation. Using a transformational forward engineering perspective, we gain the insight that much of this difficulty is caused by design decisions made during system development. Such decisions "hide" the program functionality and performance requirements in the final system by applying repeated refinements through layers of abstraction, and information-spreading optimizations, both of which change representations and force single program entities to serve multiple purposes. To be able to reverse engineer, we essentially have to reverse these design decisions. Following the transformational approach we can use the transformations of a forward engineering methodology and apply them "backwards" to reverse engineer code to a more abstract specification. Since most existing code was not generated by transformational synthesis, this produces a plausible formal transformational design rather than the original authors' actual design. A byproduct of the transformational reverse engineering process is a design database for the program that then can be maintained to minimize the need for further reverse engineering during the remaining lifetime of the system.**

Keywords **- Legacy Program, code, analysis, Reverse engineering, Forward engineering, Transformational Approach, software engineering.**

I. INTRODUCTION

Object-oriented software development methodology primarily has three phases Analysis, Design and Implementation. With the view of traditional waterfall model, reverse engineering thus is looking back to design from implementation and to analysis from implementation. Important thing is that it actually is a reverse forward engineering i.e. from implementation; analysis is not reached before design. Reverse Engineering is a methodology that greatly reduces the time, effort and complexity involved in solving these issues providing efficient program understanding as an integral constituent of re- engineering paradigm. Reverse engineering produces a high-level representation of a software system from a low-level one. This paper discusses about reverse engineering of java code & recovers the design artifacts of a software system from its source code and related documentation. Reverse engineering is a process of examination only: The software system under consideration is not modified (which would make it re- engineering). Software anti-tamper technology is used to deter both reverse engineering and re-engineering of proprietary software and software-powered systems. In practice, two main types of reverse engineering emerge. In the first case, source code is already available for the software, but higher-level aspects of the program, perhaps poorly documented or documented but no longer valid, are discovered. In the second case, there is no source code available for the software, and any efforts towards discovering one possible source code for the software are regarded as reverse engineering. This second usage of the term is the one most people are familiar with. Reverse engineering of software can make use of the clean room design technique to avoid copyright infringement.

A simple schematic diagram to elaborate reverse engineering is shown in "Fig.1". It is the process of analysis. The software system or program under study is neither modified nor re-implemented because of not bringing it under Re-engineering. Software Re-engineering is the area which deals with modifying software to efficiently adapt new changes that can be incorporated within as software aging is a well known issue. Reverse engineering provided cost effective solution for modifying software or programs to adapt change management through Re-engineering application.

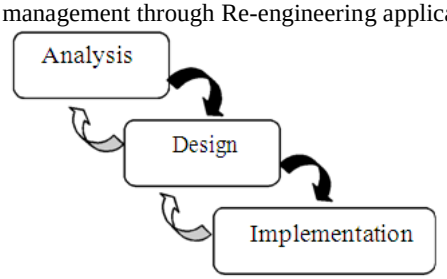

Fig.1. A simple representation to reverse engineering of object-oriented development.

SDLC Traditional flow
Reverse Engineering flow

Reverse engineering is a systematic form of program understanding that takes a program and constructs a high-level representation useful for documentation, maintenance, or reuse [1]. Reverse engineering can be used for a variety of purposes: to reconstruct or improve documentation; to facilitate software maintenance or conversion activities; or to redesign and re-engineer an existing system.

Unfortunately, source code does not contain much of the design information and additional information sources are required. Usually the scale of the software is often large; the maintainer also needs some automated support for the understanding and the recovery of the design artifacts. The design information from a combination of

code, existing design documentation and general knowledge about problem and application domain is required to recover the design artifacts.

II. Reverse Engineering

If all existing software had been developed using a transformation system recording the design and its rationale there would be no need for reverse engineering any code. However, we have to make a concession to reality [15]. A huge amount of conventionally developed software exists and such systems will continue to be developed for a long time.

These systems have errors and continual demand for the enhancement of their functional and performance requirements. Modification is a necessity. This means that there is a fundamental need to understand them. As they are often badly designed and have an incomplete, nonexistent, or, even worse, wrong documentation without any design information, this is a challenging task. Currently a major approach to understanding consists of trying to recognize plans (code fragments implementing programming concepts) bottom-up from the program to a more abstract description. Essentially, this is done by pattern matching on an internal representation of the code (e.g. an abstract syntax tree) which leads to detecting patterns of higher-level plans or concepts in a lower-level code.

This approach has a major drawback: It depends on having developed all (or at least all important) plan instances that may occur in the code that we want to understand. Unfortunately these plans can be (and have been) realized in many different ways because of the interactions between many possible design decisions (i.e. transformations). Imagine a logical array of structures; it can be implemented as an array of structures or a structure of arrays. Code to sort the array will look very different depending of the choice of implementation. A set of plans for sorting must somehow account for all the possible different representations of the data.

Thus, to detect a plan in legacy code we have to be able to detect all these different realizations of the same plan. But unfortunately we need different patterns to detect these different realizations; and all these patterns have to exist in advance to be able to reverse engineer the code, which is impossible because of the multitude of conceivable realizations all solving the same problem. Worse, we must take into account that the plans we want to recognize are interleaved, delocalized, and share resources. So we must find plan patterns in the code that are intertwined with other plan patterns both of which may be scattered throughout the whole program code.

With the help of Reverse Engineering we generate Class Diagram from java code than after the code analysis we get the output in form of class diagram shown in “fig.2”. So the first solution came in the form of “Class Diagram”. Here there is another interface by which the user can define the setting parameters of the class diagram like in terms of hierarchical degree of separation, in terms of functions and attributes of each sub classes (child classes) also the root (parent class) etc. There are some more properties are there which the user can set according to his need.

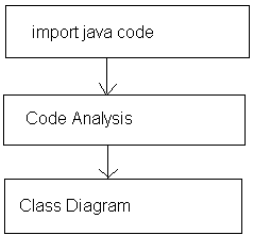

Fig. 2. Java code to class diagram

This is a description of how the system will interact with its users. It actually defines how the user will interact with the application which uses the concept of reverse engineering and extract the design artifacts from the input java code & helps us to understand the working as well as functionalities of the code in an easier manner and understandable manner.

III. Classification of Reverse Engineering

Reverse Engineering Techniques can be classified in two ways –

- Programmer’s view
- Analyzer’s view

A. Programmer’s view

It is based on the input to be provided to the tools or environments to analyze the system under study. A programmer has prime concern for three aspects : creational, structural and behavioral. Usually that affects two types of reverse engineering paradigms-

- Code Reverse Engineering
- Data Reverse Engineering

B. Analyzer’s view

It is based on the type of analysis conducted on software or program to inspect the properties to look into structural or behavioral aspects of the system under study.

Two major kinds of analysis can be done under reverse engineering- Structural Analysis and Behavioral Analysis [2]. The former is called static analysis whereas the later one is known as Dynamic Analysis.

Java is an object oriented language centered on Objects reflecting real time behavior. Structural analysis of java code can be used to identify code elements viz. Attributes, fields, methods and other code artifacts whereas Behavioral analysis is concerned with the runtime behavior of objects created and then garbage collected during the program execution.

IV. Purpose of Reverse Engineering

Reverse-engineering is used for many purposes: as a learning tool; as a way to make new, compatible products

that are cheaper than what is currently on the market; for making software interoperate more effectively or to bridge data between different operating systems or databases; and to uncover the undocumented features of commercial products etc [6]. The major motivations behind usage of this technology are as following:

1. Interoperability

Interoperability refers to production of a product which operates with the product to be analyzed. Examples of such products are applications which need to interoperate with operating systems; software controlled exchanges which need to operate with others. Sometimes software needs to be updated or corrected as according to the current need, at those times it is required specially in case of no or insufficient documentation.

2. Lost documentation

It is often done because the documentation of a system has been lost (or was never written), or the developer is no longer available.

3. Product analysis

To examine how a product works, what components it consists of, estimate costs, etc.

4. Competition

Competitors may want to produce a product which competes with the product to be analyzed or they may use it for the purpose of competitive technical intelligence. It means understand what your competitor is actually doing versus what they say they are doing.

5. Learning

It may be used for learning from others' mistakes. So that same mistakes that others have already made and corrected, are not made by the analyzer.

6. Military or commercial espionage

It may be used for learning about an enemy's or competitors latest research by stealing or capturing a prototype and dismantling it.

7. Creation of unlicensed/unapproved duplicates

It is used for producing illegal duplicates.

8. Cracking

Breaching security is a task performed by crackers in which it plays a vital role and is used by the crackers frequently.

V. Problem Statement

A common problem experienced by the software engineering community traditionally has been that of understanding legacy code. Legacy code is a semi- formal term that refers to the programs coded for typically industry strength projects, which become increasingly difficult to understand as they grow in size and complexity. Most legacy code and systems were designed before object -oriented development was used. So it is expensive and risky to replace the legacy code and system.

Software engineering has undergone a paradigm shift as the size of the software systems deployed increased dramatically and businesses began to rely increasingly on computers and information systems [3]. A substantial portion of the software development effort is spent on maintaining existing systems rather than developing new ones. An estimated 50% to 80% of the time and material involved in software development is devoted to maintenance of existing code. Crucial to the maintenance of existing systems is the task of program comprehension, an emerging area in software engineering. 47% of the time spent on enhancements to existing programs and 62% of that spent on program corrections involve program comprehension tasks like reading the documentation, scanning the source code, and understanding the changes to be made. All of these tasks pose a barrier in getting actual advantages of increasing business value [14].

VI. Solution

Reverse engineering is taking apart an object to see how it works in order to duplicate or enhance the object [7]. reverse engineering is very common in such diverse fields as software engineering, entertainment, automotive, consumer products, microchips, chemicals, electronics, and mechanical designs. For example, when a new machine comes to market, competing manufacturers may buy one machine and disassemble it to learn how it was built and how it works.

The major research issues involve the need for formalisms to represent program behavior and visualize program execution. Reverse engineering has many supporting aspects. It may focus on features such as control flows, global variables, data structures, and resource exchanges. At a higher semantic level, it may focus on behavioral features such as memory usage, uninitialized variables, value ranges, and algorithmic plans. At an even higher level of abstraction, it may focus on business rules, policies, and responsibilities. Each of these points of investigation must be addressed differently.

VII. Methodology

The Reverse Engineering Abstraction Methodology (REAM) which we used to discover the design artifacts is aimed at assisting the activities of reverse engineering to recover the design of the software at different levels of abstraction [9]. The methodology consists of (five phases) shown in “Fig.3”, high level model, functional model, architectural model, source code model and mapping model. REAM help engineers perform various software engineering tasks by exploiting the high-level, functional, architectural, source code and

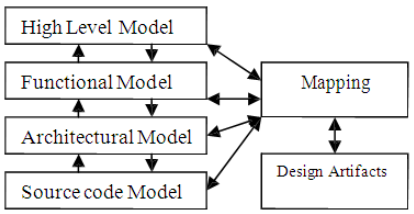

Fig. 3. Reverse Engineering Abstraction Methodology

Mapping models to recover the design artifacts. The

goal of this iterative approach is to enable a software engineer to produce, within a time constraints of the task being performed, a high-level, functional, architectural, source code and mapping model that is suitable to use for recovering the design artifacts and reasoning about the tasks at hand.

An engineer can interprets the models, as necessary, modifies the high-level model, functional model, architectural model, source code model, or mapping model to iteratively to recover and reasons about the systems artifacts.

VIII. Application Domain

The Reverse Engineering methodology permits the user to develop high-level, functional, architectural, source code and mapping models to recover the design artifacts at different levels of abstractions by exploiting various types of information (like available documents, experience and source code). The approach not only providing a choice to derive the high level model from the source code model but it also provide the approach to develop and abstract the high-level model, functional model and architectural model from the source code model and available sources (like documents and domain knowledge), and correlate them at different levels of abstraction [9]. The methodology is lightweight and iterative and can be used according to the tasks in hand at different levels of abstraction. The methodology also demonstrate that, high-level, functional, architectural and mapping models can be beneficial for planning, assessing, and executing tasks on an existing system to recover and abstract the design artifacts.

Identifying classes from source code and convert it into the UML diagram

The distinction between classes, interfaces, and data types is semantically important in UML. So this element is described as classifiers. We apply data filtering on classifiers [10]. Filtering is a process in which searching of classes occur according to its prefix and name. Now we find out the nodes and edges of classes with the help of their attributes and operation respectively. For example the node attributes can be node width, node height, node dependent etc. and edge operation can be dependent directories, source destination. After finding the dependent classes, the next target on the directories in which classes are lies. Load the classes and directories than apply the fragmentation process. In this process the tokens from loops are divided and resolve the error message and handling [11]. All of these possible through GIF filter (Graphics Interchange Format). GIF filter is nothing but the mapping process in which nodes are converted into GIF Format and than map on the graphics panel.

IX. A Comparison Case Study on Tools

Two tools were selected in this study as they support java reverse engineering. These were –

A. Rational Rose

Rational Rose [5] [13] is a widely used commercial UML modeling tool. Rational Rose offer reverse engineering capabilities, but their capabilities are very limited. Rational Rose supports reverse engineering of, Java software systems. When reverse engineering a Java program, Rose constructs a tree view that contains classes, interfaces, and association found at the highest level. Methods, variables etc. are nested under the owner classes. Representation of the extracted information and generates a default layout for it.

Additionally, Rose automatically constructs a package hierarchy as a tree view. Rose is able to reverse engineer the information from the source code (.java files), byte code (.class files), jar files, or packed zip files. In Rose, the Java reverse engineering module can be given instructions on files, directories, packages, and libraries to be examined.

B. Reverse

Reverse is non commercial tool to convert java code to class diagram developed by Neil Johan. [4] .User needs to select main java file and tool automatically displays class diagram. Tool has extracted limited classes, but no interfaces. Hence, realization relationships have not been extracted. It was successful in identifying most of the associations.

X. UML Properties

Number of Classes (NOC)

This is a general measure for the overall size of a software module. Therefore, high NOC values may indicate a more detailed representation.

Number of Associations (NOA)

NOA is a metric measure of interconnectedness in a module. In reverse engineering it is important to understand how classes are connected.

Number of Generalization relationship (NGR)

It models “is a” and “is like” relationships, enabling you to reuse existing data and code easily. It is a generalization / specialization relationship between classes, which helps to measure how tightly coupled classes are. From reverse engineering point of view it will help for concluding component structure.

Handling of Interfaces

An interface is a specified for the externally-visible operations of a class, component, or other classifier (including subsystems) without specification of internal structure. In UML diagrams, interfaces are drawn as classifier rectangles (with a stereotype << Interface >>) or as circles. The interfaces are attached by a dashed generalization arrow to classifiers that support it, known as realization relationship. This indicates that the class provides (implements) all of the operations of the interface. The circle notation is used when the operations of the interface are hidden. A class that uses or requires the operations supplied by the interface may be attached to the circle by a dashed arrow pointing to the circle. From the reverse engineering point of view, generation of such dependencies is important for understanding the usage of

interfaces and for concluding component structures and dependencies.

Role Names

The function of role names at association ends is comparable to that of attribute names in the sense of giving to an association between classes a meaningful descriptor, which depends on the end its attached to. Therefore in reverse engineering, role names can hold relevant additional information about the system infrastructure. We examine, whether role names are used and if, what kind of information they represent.

XI. Expected Outcomes

The goal of reverse engineering activity is to recover the design models (design artifacts) from the source code.

- This Reverse Engineering model imports Java source code and generates UML diagrams to facilitate analysis, enhancement, or reuse.
- "Hand-written or legacy source code can be difficult to decipher, since operability of the software does not depend on complete or accurate documentation, or even on a well structured code body.
- The original developer may no longer be available, or may be development was assigned in pieces and no single individual knows the entire project." Because of that we need a powerful tool by which can recover the design artifacts so that we can better understand the source code along with various logical relationships and internal dependencies in the source code.
- Also by discover all those relationships and dependencies we can more easily analyze the source code and it will also helps us to make further changes and amendments in the existing project code. It will also incorporate reusability of the code.
- This approach will read the java source code and generate UML models for a visual representation that is much clearer and more easily analyzed than a printout of the source as text.
- In this approach after getting the inputs about the relationships and the internal dependencies in the source code, we will generate the activity, class & use case diagrams as UML design artifacts.
- Due to these visual representations we can analyze the source code more easily and also can enhance the existing code as well as reuse it.

XII. Conclusion

The reason for performing reverse engineering is to maintain legacy code. It should not be focused on program understanding but on system maintenance instead. This should be done in a way that frees us from reverse engineering a system again and again because of modifications made to its code over time. Hence, reverse engineering has to be focused on design recovery. There will always be old software that needs to be understood. It is critical for the software industry to deal effectively with the problem of software evolution and the understanding of legacy software system software systems that are developed specially for an organization have a long lifetime. Many software systems that are still in use were developed many years ago using technologies that is now obsolete. These system are still essential for the normal functioning of the business. Since the primary focus of the industry is changing from completely new software construction to software maintenance and evolution. It is usually more expensive to add functionality after a system has been developed rather than design this into the system.

Reverse engineering will most often be done on older software systems whose documentation is out-of–date or non- existent. Such systems will often be candidates for re-engineering: rewriting the system to improve its understandability, easy maintenance, remove dead code, etc. it is easier to change a design than source code. In this way recovered design describes the current system. The recovered high-level design will be updated and a new detailed design generated.

References

[1] Anil Panghal, Pawan Kumar, Sharda Panghal (2009) "Reverse Engineering" Proceeding of 2nd National Conference on "Recent trends and Advancements in Computing", Sirsa, February.

[2] Chikofsky, E. J. and Cross, J. H., "Reverse Engineering and Interaction Diagrams from C++ Code", in Proceedings of Design Recovery. International Conference on Software Maintenance (ICSM'03), Jan.1990, pp. 13-17.

[3] Hausi A. Muller, Kenny Wong, Scott, R .Tilley, "Understanding Software Systems Using Reverse Engineering Technology" Department of Computer Science, University of Victoria P.O. Box 3055 , Victoria, BC ,Canada V8W3P6.

[4] http://www.neiljohan.com/projects/reverse/:-Reverse is non commercial tool to convert java code to class diagram developed by Neil Johan.

[5] IBM Rational Rose Enterprise Edition

[6] Jenkin | Reverse Engineering <http://www.jenkins.eu/articles/reverse-engineering.asp>

[7] K. Wong, Reverse Engineering Notebook. Ph.D. Thesis, Department of Computer Science, University of Victoria, October 1999.

[8] Mappings for Accurately Reverse Engineering UML Class Models from C++ Andrew Sutton, Jonathan I. Maletic, Department of Computer Science Kent State University Kent Ohio 44242 {asutton, jmaletic}@cs.kent.edu

[9] Manoranjan Satpathy , Nils T Siebel, and Daniel Rodriguez, "Maintenance of Object Oriented Systems through Re-engineering": A Case Study, Department of Computer Science, The University of Reading, satpathy@ reading.ac.ukfnts, drgg@ieee.org Proceedings of the International Conference on Software Maintenance (ICSMÌ02)0-7695-1819-2/02 $17.00 © 2002 IEEE

[10] M. L. Domsch and S. R. Schach. A case study in object-oriented maintenance. In Proceedings of the 1999 Inter- national Conference of Software Maintenance (ICSM í99), Pages 346–352, August 1999.

[11] Tonella, P. and Potrich, A., "Reverse Engineering of the Interaction Diagrams from C++ Code", in Proceedings of International Conference on Software Maintenance The Netherlands, Sep 22-26 2003.

[12] Ralf Kollmann , Petri Selonen , Eleni Stroulia , Tarja Syst¨a_, Albert Z¨undorf_A Study on the Current State of the Art in Tool-Supported UML-Based Static Reverse Engineering In Elizabeth Burd and Arie van Deursen, editors, Proc. 9thWorking Conference on Reverse Engineering. IEEE, Los Alamitos, 2002

[13] Rational Corporation website http://www.rational.com
[14] Roger S. Pressman-Software Engineering.
[15] Software Security and Reverse Engineering http://www.infosecwriters.com/text_resources/pdf

AUTHOR'S PROFILE

Namrata Sharma

She received Bachelor's degree in Information Technology from MIT, Ujjain (M.P.) and Pursuing M-Tech in Information Technology from SSSIT, Sehore (M.P.) India. Her research Area is Software Engineering and Computer Graphics. She has three year of teaching experience, presently she is working as a lecturer in Information Technology department in KCBTA, Indore M.P. India. Email Id- namarata_28@yahoo.com

Mr. Gajendra Singh Chandel

Mr. Gajendra Singh Chandel received his Bachelor of Engineering degree in Information Technology from Oriental Institute of Science and Technology, RGPV Bhopal In 2007, M.P., India. He has completed his M.Tech. (Master of Technology) degree in Computer Science & Engineering from Lakshmi Narayan College of Technology, RGPV Bhopal, In 2010 M.P., India. Presently he is HOD Of Computer Science Engineering & Information Technology Department in SSSIST, Sehore M.P. India. He is having 4 Yrs of teaching experience. He has published 18 papers in refereed International/National Journal and Conference including IEEE. He is a Member of Easy Chair Conference System. Email Id-gajendrasingh86@rediffmail.com