Study of Various Methods Used in Reverse Engineering

Preeti Kathiria, Dhaval Jha

Abstract- A process to analyze the system in order to produce its representations at a higher level of abstraction is called Reverse Engineering. In this paper, an outline of the various methods used for reverse engineering of the software is presented along with the elaboration of those methods.

Keywords: *Reverse engineering, Abstraction*.

I. INTRODUCTION

The major factor involved in a program is to understand and document it properly. In order to thoroughly understand a program, we need to identify lop invariants and then analyze the program [2]. The term reverse engineering was coined by Chikofsky and Cross in 1990 in their seminal paper as the process of analyzing the system under study and to identify components of the system and relationships among them and to construct system's representations in some alternative form or at a higher level of abstraction [1]. Reverse engineering means to go backwards through the development cycle in which, the output of the implementation phase, which is in the form of source code is converted back to the analysis phase, which is just the reciprocal of the traditional waterfall model.

II. GOALS AND AREAS

When the source code is only available as a reliable representation, then the reverse engineering serves as the supporting technology which can deal with such a system[3]. Reverse engineering of a software contains multiple goals:

- Generating alternate views of the software (Clones)
- Recovering the lost information
- Reusability of the components already used in that software
- Determining the side effects of a particular software on the system on which it has been implemented
- Synthesizing higher abstractions

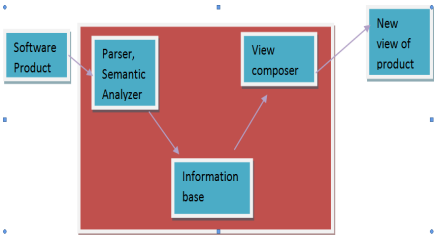
Generally, two categories of software reverse engineering exist. For the first category, availability of the source code for the software is there, and we try to discover the higher-level aspects of the program, which may be poorly documented or documented but no longer valid. For the second category, the source code is unavailable for the software, and, the efforts are made towards discovering one possible source code for the software and these steps are considered as reverse engineering. The second type is the most common type.

The following list includes the examples of problem areas wherein reverse engineering has been successfully applied:-

- Redocumenting program and relational databases
- Identifying reusable assets,
- Recovering architecture
- Recovering design patterns
- Building traceability between code and documentation
- Identifying clones
- Code smells and aspects
- Computing change impacts
- Reverse engineering binary code
- Renewing user interfaces
- Translating a program from one language to another
- Migrating or wrapping legacy code
- To create representations necessary for testing purpose
- To audit security and vulnerability

III. THE PROCESS OF REVERSE ENGINEERING

Reverse engineering involves two steps: information extraction and abstraction. Information extraction analyses artifacts of the system under study in order to gather raw data, while Abstraction involves creation of documents and views which are user-oriented. As an illustration, activities under information extraction include drawing Control Flow Graphs (CFGs), metrics or facts from the source code. The outputs of Abstraction can be design artifacts, traceability links, or business objects. Firstly, the analysis of the software product to be "reversed" is carried out and then the results of this analysis are stored into an information base[5]. This information is in turn used by view composers to produce alternate views of the software product, such as metrics, graphics, reports, etc.



Basic Architecture of reverse engineering process

The aim of most of the reverse engineering tools is to obtain abstractions or different forms of representations, from software system implementations. However, reverse engineering can be performed on any software artifact:

- Requirement
- Design
- Code
- Test case
- Manual pages

Reverse engineering approaches can have two broad objectives:

- *Redocumentation*: The goal of Redocumentation is to create another view of a given artifact, at the same level of abstraction.
- *Design recovery*: The objective of Design recovery is to recreate design abstractions from the source code, existing documentation, and experts' knowledge and from other variety of source.

IV. EXAMPLES OF RCE

Reverse code engineering (RCE) is a process of reverse engineering of the binary software[4].

- For instance, on the Java platform the decompilation of binaries can be done using Jad.
- The Samba software, that permits systems, not running Microsoft Windows, to share files with systems that are running it, can be considered as a standard example of software reverse engineering.
- The Wine project is a free and open source software application that aims to allow applications designed for Microsoft Windows to run on Unix-like operating systems.
- The ReactOS project aims to provide binary (ABI and API) compatibility with the current Windows OSes of the NT branch, which in turn allows the software and drivers written for Windows to run on a clean-room reverse-engineered GPL free software or open-source counterpart.

V. UML TOOLS AND KDM

UML (Unified Modeling Language) tools is a process to import and analyze the source code to generate UML diagrams, which provides a standard way to visualize the system. Some commonly known UML tools are Microsoft Visio, Netbeans, MyEclipse, IBM's Rational Software Architect etc. These diagrams basically represent the structural and behavior aspects of the system.

There is also more advanced approach of providing reverse engineering other than UML, known as Knowledge Discovery Metamodel (KDM). It is publicly available specification from the Object Management Group (OMG).It is a common intermediate representation for existing software systems and their operating environments that defines common metadata required for deep semantic integration of Application Lifecycle Management tools. The extraction and analysis of source, binary and byte code is delivered. In terms of analysis of source code, the standard architecture of KDM allows to extract the flow of the software system, architectures, and business layer knowledge. XML can be used which can correlate with different layers of the system knowledge for either detailed analysis or derived analysis. The task of representation of language constructs is quite cumbersome due to the given number of languages. However, because of the growth and creation of new software languages, the standard permits the use of extensions to support the broad language set as well as evolution. Due to the compatibilty of KDM with UML, BPMN, RDF and other standards, it enables migration into other environments, thereby leveraging system knowledge for efforts such as software system transformation and enterprise business layer analysis.

VI. METHODS USED IN REVERSE ENGINEERING OF THE SOFTWARE:

Various methods can be employed to accomplish the Reverse engineering of software. The major three categories of software reverse engineering are as given below:-

- Analysis through observation of information exchange. It is employed in protocol reverse engineering, which deals with bus analyzers and packet sniffers.
- The bus analyzer is a bus analyzer tool, often combination of hardware and software, which is used in reverse engineering. Its job is to monitor the bus traffic and decode and display the data.
- On embedded systems, reverse engineering is governed by tools introduced by the manufacturer, such as JTAG ports. JTAG(Joint Test Action Group) is widely used for IC debug ports. In embedded systems, essentially all modern processors implement JTAG.
- SoftICE is a kernel mode debugger for Microsoft Windows. It is designed to run underneath Windows in such a way that the operating system is unaware of its presence.
- The packet sniffer is defined as the software or a piece of computer hardware which can intercept and log traffic which passes through the network.
- Wireshark and Microsoft Network Monitor are the examples of the packet sniffer.

VII. DISASSEMBLY USING A DISASSEMBLER

It is program which performs the translation from machine language to assembly language. Disassembly is often

formatted for human-readability rather than suitability for input to an assembler, making it principally a reverse-engineering tool.

- Limitations with disassembler:
- To write a disassembler which produces the code which, when assembled, produces exactly the original binary is possible; but it may not happen every time.
- Inspite of production of a fully correct disassembly, problems remain if the program requires modification. For example, the same machine language jump instruction can be generated by assembly code to jump to a specified location (for example, to execute specific code), or to jump to a specified number of bytes (for example, to skip over an unwanted branch).
- Types of disassembler:

There are two types of the disassembler:

- A stand-alone disassembler
- An interactive disassembler immediately shows the effect of any change made by the user. For instance, the disassembler may be initially unaware of the fact that the section of the program is actually code, and treat it as data; if the user specifies that it is code, the resulting disassembled code is shown immediately, which permit the user to check it and take further actions during the same run.
- Examples: Interactive Disassembler (IDA).
- Decompilation using decompiler:

The decompiler translates the code into a higher level of abstraction. Usually, the decompilers do not exactly reconstruct the original source code, and widely vary in the intelligibility of their outputs.

Mostly, the *decompiler*term is used for a program that converts executable programs into the high-language, which, when compiled, will produce an executable whose behavior is the same as the original executable program.

- Uses of decompilation:
- Recovery of lost source code
- Computer security
- Interoperability (To allow information exchange between systems)
- Error correction
- Phases in design of decompiler:
- Loader
- Disassembly
- Idioms
- Program analysis
- Data flow analysis
- Type analysis
- Structuring
- Code generation
- Examples:
- Java decompiler
- Boomerang decompiler
- JEB decompiler

VIII. CONCLUSION

Different reverse engineering methods are available for the reverse engineering of the given software system. The use of the appropriate method depends upon the source input we are given (i.e. either it is a machine code or high level code) and the kind of output we want through the process of reverse engineering (i.e. assembly code, higher level code or UML diagrams etc.).

References:

[1] Gerardo Canfora and Massimiliano Di Penta "New Frontiers of Reverse Engineering"

[2] H.D. Mills and V.R. Vasuli "Understanding and Documenting Programs" IEEE Transactions on Software Engineering vol. 8 no. 3 pp. 270-283 1982.

[3] Robert H. Lande and Sturgis M. Sobin "Reverse Engineering of Computer Software and U.S. Antitrust Law"

[4] Dennis B. Smith, Hausi A. Muller, Jens H. Jahnke, Kenny Wong, Margaret-Anne Storey and Scott R. Tilley "Reverse Engineering: A Roadmap"

[5] Chikofsky, E. J. and Cross, J. H., "Reverse Engineering and Design Recovery: A Taxonomy"