World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:6, 2008

ReSeT: Reverse Engineering System Requirements Tool

Rosziati Ibrahim, and Tiu Kian Yong

***Abstract*—**Reverse Engineering is a very important process in Software Engineering. It can be performed backwards from system development life cycle (SDLC) in order to get back the source data or representations of a system through analysis of its structure, function and operation. We use reverse engineering to introduce an automatic tool to generate system requirements from its program source codes. The tool is able to accept the C++ programming source codes, scan the source codes line by line and parse the codes to parser. Then, the engine of the tool will be able to generate system requirements for that specific program to facilitate reuse and enhancement of the program. The purpose of producing the tool is to help recovering the system requirements of any system when the system requirements document (SRD) does not exist due to undocumented support of the system.

***Keywords*—**System Requirements, Reverse Engineering, Source Codes.

I. Introduction

REVERSE engineering is the process of discovering the technological principles of a device or object or system through analysis of its structure, function and operation [9]. Most of the time, it involves taking something apart, for example the device or the system program, and analyzing its working in detail, and trying to make a new device or program that does the same thing without copying anything from the original [6].

In reverse engineering, the process is often tedious but necessary in order to study the specific technology or device. In system programming, reverse engineering is often done because the documentation of that particular system has never been written or the person who developed the system is no longer working in the company. We use this concept to introduce an automatic tool for retrieval of requirements of a system from the program source codes.

The purpose of producing the tool is to be able to recover the system requirements of any system due to the cause that the system does not have the necessary documents. Documenting the process involved in developing the system is important. In many organizations, 20 percent of system development costs go to documenting the system [4]. In software development life cycle (SDLC), documenting process in requirements analysis ends with a system requirements document (SRD) [9]. SRD is important in order to develop a system. It shows the system's specification before a developer would be able to develop the system. Once the system demonstrates fault after implementation phase, the SRD can be used as a reference for finding errors of the system requirements. However, if documenting is not proper, the source codes of the system will be used to find errors. This is a difficult process considering the lines of the source codes would be thousand. Therefore, by having a tool that would be able to retrieve the system requirements back from the source codes would be an added advantage to the developer of any system application.

This paper discusses on retrieval of system requirements from its source codes. The rest of the paper is organized as follows. Section II presents the related work and Section III discusses the system requirements. We also present our idea on how to read the source codes, parse it to parser and then convert it to system's requirements in Section III. Section IV discusses our tool in details, in particular on how to retrieve data from the source codes using the engine of the tool. Finally, we conclude our paper in Section V and give some suggestions for future work of the tool.

II. Related Work

Reverse engineering has become a viable method to measure an existing system and reconstruct the necessary model from its original. In the older days, disassembler is used to recreate the assembly codes from the binary machine codes, where the assembler is used to convert the codes written in assembly language into binary machine codes. Decomplier, on the other hand, is used to recreate the source codes in some high level language for a program only available in machine codes or bytecodes.

Based on the decompiler and disassembler, there has been a significant amount of study focusing on disassembly of the machine code instructions. Schwarz et al. [8], for example, study the disassembly algorithms and propose an hybrid approach in disassembly of the machine code. Tilley et al. [10], on the other hand, propose the programmable approach using the scripting language to enable the users to write their own routines for common reverse engineering activities such as graph layout, metrics and subsystem decompositions.

There has been a significant amount of study for looking at the best technique in reverse engineering focusing on studying the source code in order to get the design or requirements

Rosziati Ibrahim is with the Research Management and Innovation Centre, Universiti Tun Hussein Onn, Malaysia (UTHM), Parit Raja, Batu Pahat, Johor, Malaysia (phone: 607-4537901; fax: 607-4536021; e-mail: rosziati@ uthm.edu.my).

Tiu Kian Yong is with the Faculty of Information Technology and Multimedia, Universiti Tun Hussein Onn Malaysia (UTHM), Pari Raja, Batu Pahat, Johor, Malaysai (e-mail: jonastiu@gmail.com).

documents. Knodel et al. [5] suggest using the graphical elements in order to understand the software architecture better. Graph-based technique is proposed to be one of the good techniques in reverse engineering in order to get the requirements documents. Cremer et al. [3], for example, use the graph-based technique for COBOL applications and provide code analysis. Based on the graph-based technique as well, UML (Unified Modeling Language) reverse engineering [1] imports Java source codes and generates UML class diagram to facilitate requirements analysis.

For our approach, we use graph-based technique as well to get the necessary information from the C++ source codes, convert the information into necessary tokens and then use these detected tokens to generate the UML class diagram. The class diagram can be used for requirements analysis. Altova tool [1] is quite similar to our tool. However, Altova concentrates on Java program source codes for its input to generate the UML class diagram, our tool, on the other hand, concentrate on C++ program source codes for its input to generate the UML class diagram.

III. The System Requirements

In UML specification, requirements analysis and design are usually done using diagrams [2]. One particular diagram (a use-case diagram) is used to specify requirements of the system. In a use-case diagram, two important factors are used to describe the requirements of a system. They are actors and use cases. Actors are external entities that interact with the system and use cases are the behavior (or the functionalities) of a system [7]. The use cases are used to define the requirements of the system. These use cases represent the functionalities of the system. Most often, each use case is then converted into a function representing the task of the system.

In this paper, we present an example of an application for monitoring system of a postgraduate student submitting his/her progress report to Centre of Graduate Studies. The requirements of the system include the capability to submit progress report using the provided form, view the submitted progress report and evaluate the submitted progress report. These three requirements are then transformed into a use-case diagram as shown in Fig. 1.

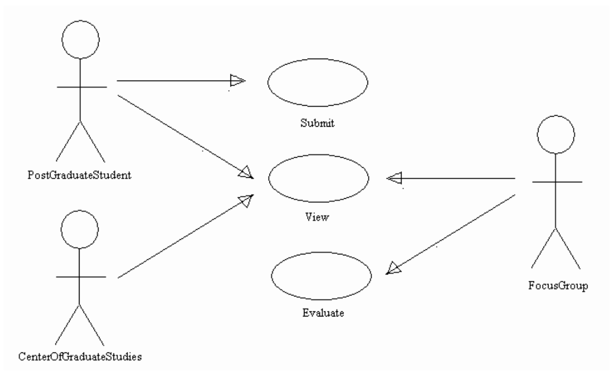

Fig. 1 A use-case Diagram for Monitoring System of Postgraduate Student

Fig. 1 shows a simple use-case diagram for a monitoring system of postgraduate student where a postgraduate student (an actor) can submit his/her progress report to Centre of Graduate Studies. From Fig. 1, a student is able to do two tasks: submit a progress report and view a progress report. A focus group is able to view and evaluate the progress report while the centre is able to view the progress report.

Most often, use cases represent the functional requirements of a system. If the requirements are gathered correctly, then a good use-case diagram can be formed. From this use-case diagram, the use cases are usually used for the functions of the system. Table I shows the mapping of use cases to functions of a system. These functions can then be converted to a class diagram for the system.

TABLE I
Use Cases Mapping to System's Functionalities

Use Case	**Function**
Submit	Submit
View	View
Evaluate	Evaluate

The class diagram is the main static analysis diagram [2]. It shows the static structure of the model for the classes and their relationships. They are connected to each other as a graph. Each class has its own internal structures and its relationships with other classes. Fig. 2 shows an example of a class diagram for Monitoring System of Postgraduate Student. Note that the mapping from use-cases from Fig. 1 into functions in the class diagram in Fig. 2. This mapping is important for the consistency of the UML diagrams.

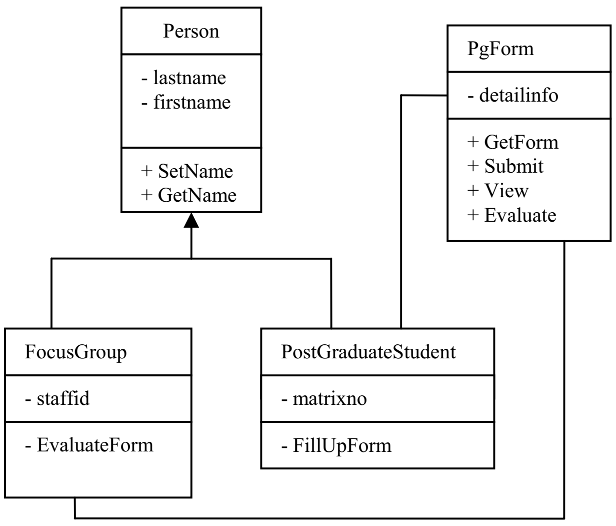

Fig. 2 A Class Diagram for Monitoring System of Postgraduate Student

From Fig. 2, each class consists of a class name, its attributes and methods. For example, a class *Person* has attributes lastname and firstname with no method. Classes *FocusGroup* and *PostgraduateStudent* inherit class *Person*. Class *FocusGroup* declares its own attribute (staffid) and one method (EvaluateForm) and class *PostGraduateStudent* declares its own attribute (matrixno) and one method (FillUpForm). Note that, a subclass inherits all the attributes and methods of its superclass. Class *PgForm*, on the other hand, offers 4 methods namely GetForm, Submit, View and Evaluate. The three methods are translated from the three use cases declared in Fig. 1.

IV. THE RESET

The tool, which we call ReSeT (Reverse Engineering System's Requirements Tool) is implemented using C++ programming language. The tool has two stages of activities. The first stage accepts the source codes of C++ programming language as the input and produces the output as detected tokens in term of the set of class name, its attributes and functions as well as its relationships with other classes. From this output, for the second stage, the tool will suggest the possibility of the class diagram. The targeted user of the tool is software developer who wants to get back the system's requirements specification based on the program source codes.

The main objectives of developing the tool are being able to detect the necessary tokens from the syntaxes of the program source codes and generate the class diagram automatically based on the detected tokens.

To generate the class diagram, a user is required to input a C++ program source codes into the tool. After that, the tool will validate the file format as well as the filename. If an invalid file format has been entered or the file does not exist, the tool will prompt an error message to warn the user. Indeed, the user needs to reinsert the filename. However, if both the filename and file format is valid, the tool will reconfirm whether it is the file that the user needs. All the commands in the tool are case-insensitive where the tool will recognize both lowercase and uppercase command typed by the user. The tool will also provide files searching function in order to list out all the files' name in a folder. The tool will only accept a C++ source codes with the ".cpp" and ".h" extensions. When user tries to insert an invalid file, the tool will display a warning message and ignore the file.

Once the correct source codes file has been verified, the source codes are parsed to the parser. The parser will read the file, line by line, detect the tokens and store the necessary tokens to form the class diagram. Note that, the tool will bypass all the comments found in the file. There are two types of comments which are single line comments (//) and multiple line comments (enclosed between /* and */). At the same time, the tool will also check the syntax error in the program source codes.

Before the class diagram is displayed, the tool will display the scanning results to the user. The result will contain the set of class name, its attributes and methods as well as its relationships with other classes. The tool will then provide two log files to store the error occurred and parsing results. The detected tokens will be stored into another file for generation of class diagram.

For the purpose of ease in understanding in this paper, we present an example of an application for monitoring system of a postgraduate student submitting his/her progress report to Centre of Graduate Studies as our case study using the tool. From Fig. 2, a class *Person* is a superclass of classes *FocusGroup* and *PostGraduateStudent*. Therefore, classes *FocusGroup* and *PostGraduateStudent* inherit all attributes and methods of class *Person*. Fig. 3 shows some of the extracted source codes from the program of this system.

```
class Person {
     private:
        char lastname [30];
        char firstname [30];
     public:
        void SetName();
        char *GetName();
};
…
```

```
class FocusGroup : public Person {
     private:
        char staffid [10];
     public:
        Void EvaluateForm();
};
…
```

```
class PostGraduateStudent : public Person {
     private:
         char matrixno [10];
     public :
         void FillUpForm();
};
…
```

```
class pgForm{
     private :
          struct Data {
                char lastname [30];
                char firstname [30];
                char matrixno [10];
          } detailinfo;
     public:
         void GetForm();
         void Submit();
         void View();
         void Evaluate();
};
…
```

Fig. 3 Extracted Source Codes

From Fig. 3, adopting the hybrid algorithm [8] by using the linear sweep and recursive traversal algorithms, the tool is able to read the source codes line by line and detect the necessary tokens. Then the tokens are stored. Fig. 4 shows the extracted tokens from reading of the program source codes.

Class name: Person Association: Inheritance: Attributes: lastname, firstname Methods: SetName, GetName
Class Name: FocusGroup Association: Inheritance: Person Attributes: staffid Methods: EvaluateForm
Class Name: PostGraduateStudent Association: Inheritance: Person Attributes: matrixno Methods: FillUpForm
Class Name: PgForm Association: Person Inheritance: Attributes: detailinfo Methods: GetForm, Submit, View, Evaluate

Fig. 4 Extracted Tokens from the Source Codes

From Fig. 4, the hybrid algorithm (combination of linear sweep and recursive traversal algorithms) is used in order to identify and extract the necessary tokens. Once the tokens have been identified and extracted from the source codes, the graph-based approach is used in our engine of the tool in order to develop and generate the class diagram from the extracted tokens. Fig. 5 shows the generated class diagram.

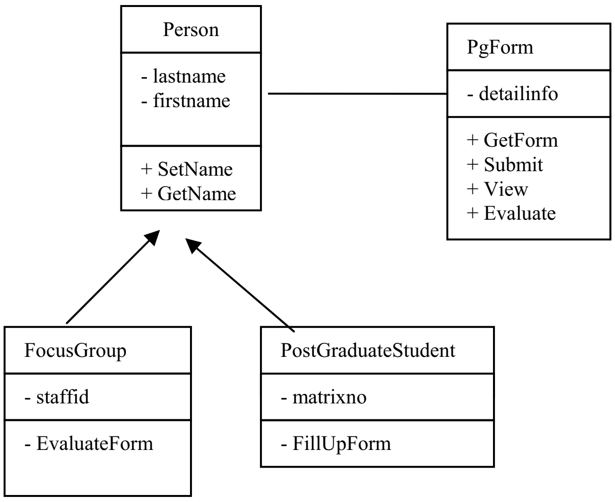

Fig. 5 Generated Class Diagram

Based on Fig. 5, the tool is able to generate the possible class diagram for the system. However, comparing from Fig. 5 and Fig. 2, we still have problems to overcome the associations of the classes and relationships among the classes. The ambiguities of the system requirements are still existed. We are currently looking at the possible solutions to reduce these ambiguities.

The tool offers the system requirements by means of extracted tokens of class name, its attributes and functions as well as its relationships with other classes. Then the tool suggests the possible class diagram based on the extracted tokens.

V. Conclusion and Future Work

The tool provides the ease in coming up with the system requirements when the system does not support the proper documents for requirements analysis. We are currently improving our algorithm of extracting the tokens in order to reduce the ambiguities of the system requirements.

For future work, the tool can also be designed to parse other types of programming languages such as Java and C#.

References

[1] Altova (2008). *UML Reverse Engineering,* http://www.altova.com/features_reverse_engineer.html
[2] Bahrami A. (1999). *Object-Oriented Systems Development*, Mc-Graw Hill, Singapore.
[3] Cremer K., Marburger A. and Westfechtel (2002). *Graph-based Tools for Re-engineering*, Journal of Software Maintenance and Evolution: Research and Practice, Voulme 14, Issue 4, pp 257-292.
[4] Heumann J. (2001). *Generating Test Cases from Use Cases*, Rational Software, IBM.
[5] Knodel J., Muthig D. and Naab M. (2006). *Understanding Software Architectures by Visualization – An Experiment with Graphical Elements*, Proceeding of the 13th Working Conference on Reverse Engineering (WCRE 2006).
[6] Musker D. (1998). *Reverse Engineering*, IBC Conference on Protecting & Exploiting Intellectual Property in Electronics.
[7] Rational. (2003). *Mastering Requirements Management with Use Cases*, Rational Software, IBM.
[8] Schwarz B., Debray S. and Andrews G. (2002). *Disassembly of Executable Code Revisited*, Proceedings IEEE Working Conference on Reverse Engineering, October 2002, pp 45-54.
[9] Sommerville I. (2007). *Software Engineering*, 8th Edition, Addison Wesley, England.
[10] Tilley S., Wong K., Storey M. and Muller H. (1994). *Programmable Reverse Engineering*, Journal of Software Engineering and Knowledge Engineering.