2007 International Conference on Convergence Information Technology

Software Reverse Engineering to Requirements

Syed Ahsan Fahmi and Ho-Jin Choi
Intelligent Software Engineering and Robotics Lab, School of Engineering, Information and Communications University, Korea.
{fahmi, hjchoi}@icu.ac.kr

Abstract

The aim of reverse engineering is to draw out many kinds of information from existing software and using this information for system renovation and program understanding. Based on traditional practice, reverse engineering and requirements engineering are two separate processes in software round trip engineering. In this paper, we argue that it is necessary to recover requirements from the reverse engineered outcome of legacy system and by integrating this outcome in the requirements phase of software life cycle, it is possible to have a better requirements elicitation, and clear understanding of what is redundant, what must be retained and what can be re-used. So we have presented a revised model of traditional re-engineering process and also described the rationality of the proposed model. In the paper we have also discussed briefly about software reverse engineering, requirement engineering and their basic practices and activities.

1. Introduction

Software systems have become very important to many companies. The business process has become so much involved with computer systems that an interruption or down of software system can cause loss of huge amount.

"These systems are usually large and complex, and they have evolved over decades. They are known as legacy systems" [7]. Due to the importance of such legacy systems, the business cannot easily get rid of it. But requirements evolve or change over time due to change in business rule, re-structuring of system, re-organization etc. This is the reason why these systems are sometimes required to be maintained. . "A fundamental problem in maintaining and evolving legacy systems is to understand the subject system" [7]. The reason behind is most of these legacy systems do not have proper documentation. Reverse engineering can assist in some ways to the proper maintenance of these legacy systems.

Recent research and literature mainly focuses on the development of different tools for reverse engineering programs and getting back of physical model and logical structure in the form of different models and structures for the legacy system. Reverse engineering activities are mostly considered as a maintenance phase activities in the life cycle of software development. As stated above, business requirements evolve over time. So while re-engineering the whole system, requirements phase plays an important role, as that is the way of including new requirements. But at the same time if the requirements of the legacy system are known and considered while making the new requirements specification, it can sure surely improve the requirements phase.

So in this paper we have argued and showed rationale of the necessity of eliciting requirements from the reverse engineered outcomes and proposed a modified model of existing simplified re engineering model. In this paper we have also discussed the basic idea and activities of software reverse engineering and requirements engineering.

2. Related Research

Recent research and literature mainly focuses on the development of different tools for reverse engineering programs and getting back of physical model and logical structure in the form of different models for the legacy system. Reverse engineering is considered as a process useful for maintenance phase.

The oldest related research found is named REVERE - REVerse Engineering of REquirements to support business process change. It was started in May 1998 and finished in the year 2000. The proposed work addressed the problem of understanding the requirements for legacy IT evolution in organizations

0-7695-3038-9/07 $25.00 © 2007 IEEE
DOI 10.1109/ICCIT.2007.228

where the pace of business process change outstrips that of its supporting IT. As that was a company funded confidential project, so the outcome was not published in detail. Some papers were published from those researchers but that does not contain enough information and detail about the process they utilized.

In 2005, the working conference on reverse engineering, there was a workshop named RETR Reverse Engineering to Requirements. In that workshop, E.J. Chikofsky mentioned about the possibility of using reverse engineering as requirements. That was in a form of presentation [11].

3. Software Reverse Engineering

"Reverse engineering is the process of analyzing a subject system to identify the system's components and their inter-relationships and to create representations of the system in another form or at a higher level of abstraction" [3]. So in other words, reverse engineering is the way to get back the requirements or the design specifications from a system. The reason behind the activities of reverse engineering is better understandability and aid in maintenance of legacy systems. In other words, it is primarily intended to recover and record high-level information about the system, including [7]:

- Structure- basically the high level architecture of the system.
- Function– what part of the program responsible for what operation.
- Behavior –understanding of runtime activities of system.
- Rationale– design involves decision making between a numbers of alternatives at each design step; and reasoning out the rational behind the design.
- Construction -modules, documentation, test suites, etc.

Several purposes for undertaking reverse engineering are listed by Brown A.J. [4].

"These purposes can be classified into the quality issues, such as the simplification of complex software, improving the quality of software, …..; the management issues, such as enforcing a programming standard, facilitating better software maintenance management techniques, etc.; and the technical issues, such as allowing major changes in software to be implemented, discovering and recording the design of the system, and discovering and representing the underlying business model implicit in the software, etc." [8].

4. Reverse Engineering Process and Research Issues

The reverse engineering process is mainly divided into four steps [7]. Those are:

Context Parsing: this is the first phase and in this phase parsed source code is transformed into in a more re-presentable form like abstract syntax tree and this intermediate form is used in the next phase.

Component Analyzing Phase: in this phase different component artifacts like data flow, structure chart, definition-use graphs etc. ate extracted from the outputs of previous phase.

Design Recovery Phase: this phase is considered as the most critical phase in reverse engineering. Different tools are used in other stages of reverse engineering process but the activities of this phase cannot be done properly through tools as implicit knowledge information is scattered around the code. This phase is time consuming and expensive and these activities are done through domain experts [7].

Design Reconstruction Phase: as the name implies, here the output models and specifications of previous phases are integrated together to re-construct the design model of the whole system.

Some of the main research issues in reverse engineering are stated in Table 1 [7].

Table. 1 Research issues and their involvement in Reverse Engineering [7]

	Program understanding	Specification Recovery	Design recovery	Architecture recovery	Business rules extraction	Cognitive processes in human program understanding	Intermediate representation of source code	Discovery of reusable component	Knowledge-based analysis
Context Parsing Phase	✓	✓	✓	✓			✓		✓
Component Analyzing Phase	✓	✓	✓	✓		✓	✓	✓	✓
Design Recovering Phase	✓	✓	✓	✓	✓	✓		✓	✓
Design Reconstructing Phase	✓	✓		✓	✓	✓		✓	✓

5. Software requirements engineering

Software requirements engineering deals with transforming software requirements into a description of required software, performance parameters, and a configuration through the iterative process definition, tradeoff studies, analysis, and prototyping [8]. Some

other commonly used terminologies for software requirements engineering are requirements analysis, requirements definition, functional and non-functional spec., and requirements. Software requirements represent needs, wishes, expectations, constraints, priorities, traceability, quantification, risk and specification

The common sources of software requirements are customers, buyers, acquisition agencies, users, operators, domain experts, system analysts, system specifications etc.

6. Software Requirements Engineering Activities

The main software requirements engineering activities include requirements elicitation, analysis, specification and validation.

Software requirements elicitation- is the tasks to acquire the requirements of the software. It is considered the stage where the requirements engineers try to find out what the stakeholders really want.

Software requirements analysis- "the tasks to organize, interpret, understand and classify software requirements" [8]. It also refers to the activities to check for the completeness, consistency and feasibility of software requirements. It can help discover the anomalies between, the boundaries of the software, to understanding the interactions with its environment and to elaborate system requirements to derive software requirements. In this stage, actually the requirements engineers focus on the problems and try to understand those without concentrating on the solution. Different type of models like data flow models, state models, and object models etc. are built and at some stage, the architectural of the solution must be derived. That is the point at where requirements and design process overlaps with each other [8].

Software requirements specifications- refer to the activities to describe the software requirements in a formal way usually software requirements specification or SRS. SRS acts as the basis for agreement between customer's side and contractors or suppliers side on "what the software product is to do as well as what is not expected to do" [8].

Software requirements validation is the activity of getting approval of the specification of the software. The requirements documents should go through validation and verification procedures to make sure that all problems are recognized related to requirements. It also helps to make sure that right requirements are there or in the other way, it will help building the right software.

7. Why Reverse Engineering for Requirements

Form the above discussion and based on the review of present literature, it is found that the main concentration of reverse engineering research is on design and architecture recovery, and developing tools for reverse engineering process. Reverse engineering is considered as a process of maintenance of in software life cycle.

Figure 1 shows that reverse engineering is a separate process from requirements engineering in the engineering process. But when we are round-trip engineering a legacy system the integration of reverse engineering with requirements engineering can give us a better understanding of the requirements of for the new system. Recovering the requirements of in-service legacy software to ensure better understanding of what is redundant, what must be retained and what can be reused.

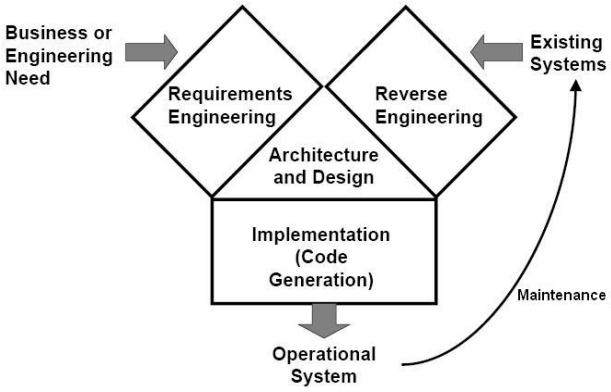

Figure1. Engineering Process[1]

As time changes, the system may require some more functionality. So the activities of requirements engineering is unavoidable to cope with situation. If we look at the Figure 1, it shows that requirements engineering and reverse engineering meet at the phase of architecture and design which goes under forward engineering phase. Actually this is the present practice for round-trip engineering process.

But if the out comes of reverse engineering are used in the earlier life cycle phase of development (i.e. requirements gathering, analysis, specification and validation phase) then it can give better understanding

[1] Ref: index technology, early 1990s. This fig. is taken from the presentation slide of Elliot Chikofsky in RETR Workshop at WCRE,05

of the requirements and improve the quality of the round tripped-engineered outcome.

7.1. The Proposed Idea of Integration

"Reverse engineering of a source code regenerates the model of a system graphically on a higher abstraction level, therefore, the structure is easy to conceive, and the relations between components can be discovered (Figure 2). More generally, reverse engineering is applied to generate a more abstract model from a software artifact" [9].

In the requirements analysis phase it can used to check for completeness, consistency. In this phase the logical models derived from reverse engineering can give a better conceptual idea to the requirements analysts about the previous system through the logical models. In many cases, the requirements engineers act as software architecture. So if the requirements engineer consider and compare both the reverse engineering output and the new requirements in this stage, it can help them draw a better architecture for the new system.

In the software requirements specification, the

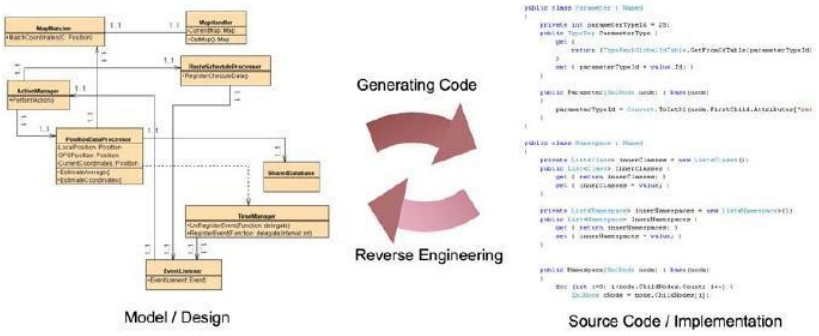

Figure 2. Forward and Reverse Engineering [9]

Now extracting requirements from these graphical models or source code is to be done. Only a few researches have been done in this area.

From the process of [10] it shows that it is possible to get back goal models and non-functional requirements through reverse engineering. In this section I will recite an example, where requirements are elicited from source code. It will show the derivation of source code to goal model by using the methodology proposed by [10] of Columbia mail. (Figure 3 and Figure 4.) These derived goal models and non-functional requirements can be used as a supplement in the requirements elicitation phase. Basically software is the fulfilled outcome of the requirements required by the stakeholders. So, these derived outcomes of the reverse engineered output can be considered as an actor in the requirements process that contains all most all the requirements of the previous legacy system. Thus it can contribute significantly in the requirements elicitation phase. In this paper, only one example is given. Future research can find different techniques to retrieve all most all the requirements of the legacy system from the diagrams or the source code derived from reverse engineering.

```
class Main {
public void run(String args[]) {
1     ColumbaLogger.createDefaultHandler();
2     registerCommandLineArguments();
3     // handle commandline parameters
4     if (handleCoreCommandLineParameters(args)) {
5       System.exit(0);
6     }
7     // prompt user for profile
8     Profile profile = ProfileManager.getInstance().getProfile(path);
9     // initialize configuration with selected profile
10     new Config(profile.getLocation());
11     // if user doesn't overwrite logger settings with commandline arguments
12    // just initialize default logging
13
14     ColumbaLogger.createDefaultHandler();
15     ColumbaLogger.createDefaultFileHandler();
16
17     for ( int i=0; i<args.length; i++) {
18        LOG.info("arg["+i+"]="+args[i]);
19     }
20 ...
```

First 20 lines of code

```
  public void run(String args[]) {
1    ColumbaLogger.createDefaultHandler();
2    registerCommandLineArguments();
3    handler.registerCommandLineArguments();
4    handle_commandline_parameters(args);
5    Profile profile = prompt_user_for_profile();
6    initialize_configuration_with_selected_profile(profile);
7    initialize_default_logging(args);
8    SessionController.passToRunningSessionAndExit(args);
9    enable_debugging_repaint_manager ();
10   StartUpFrame frame = show_splash_screen();
11   register_protocol_handler();
12   load_user_customized_language_pack();
13   initialize_plugins(handler);
14   load_plugins();
15   set_look_and_feel();
16   init_font_configurations();
17   set_application_wide_font();
18   hide_splash_screen(frame);
19   handle_commandline_arguments_in_modules(handler);
20   restore_frames_of_last_session();
21   ensure_native_libraries_initialized();
22   post_startup_of_the_modules(handler);
  }
```

Methods extracted after using refactoring tool

Figure 3. Refactored source code of Columbia mail [10]

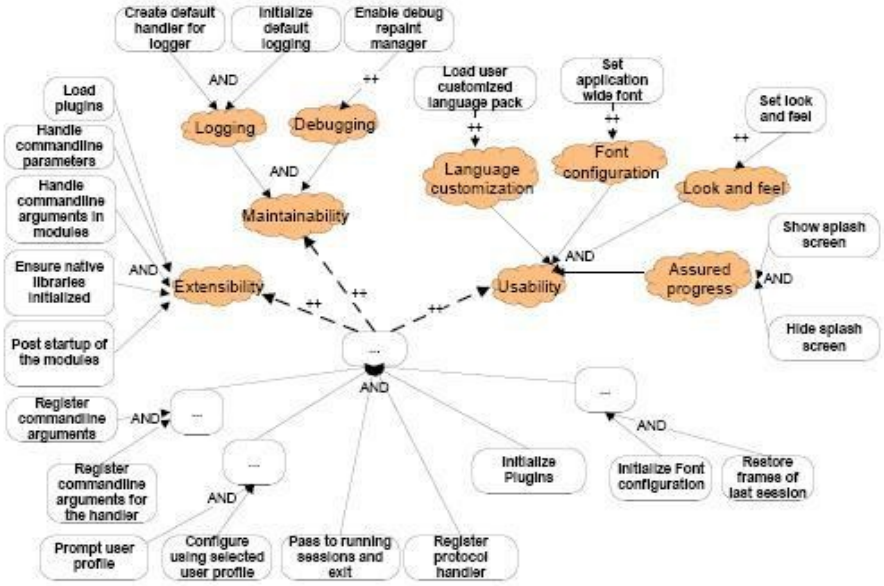

Figure 4. The Goal model and derived NFRs of Columbia mail. [10]

derived models can be used to show clearly the additions, commonalities and rejected specifications to the new software from the legacy one to the customers and developers.

The aim of the software requirements validation is to pick up any problem before resources are committed to addressing the requirements. So through the knowledge of the legacy system's behavior, logical diagrams can help in the validation process of requirements.

So based on the above discussion and arguments, the proposed modified model for round-trip engineering goes like Figure 5.

7.2. Expected Advantages of the Proposed Idea

- It can contribute a large number of requirements of the legacy system to the new system.
- It can help as a supplement in the requirements stage of the roundtrip engineering.
- By the proposed integration it is possible to check if any previous requirements are missing or not.
- It can show if any requirements are lacking or not.
- It can trace if any requirement has been changing or changed or not from the previous one.
- Any change of requirement can easily be identified.

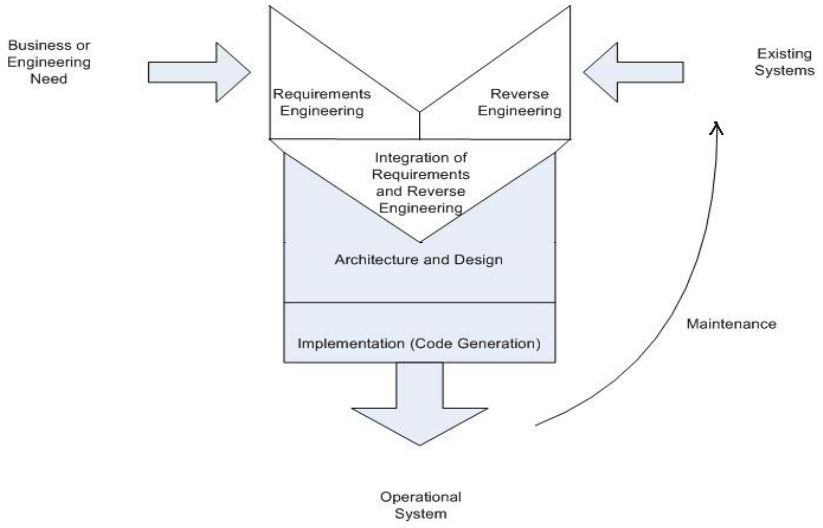

Figure 5. The modified proposed process of Roundtrip Engineering

- Any useless requirement can be traced out.

8. Conclusion and Future Work

The main reason for any change or maintenance of a software system is done usually because it can no longer fulfill the requirements of the customers or users. Legacy systems still exist because of their important role in the industry. But to cope up with the change of requirements and business rules, maintenance of these software become necessary. The main problem that arrives in the maintenance activity of these legacy systems is the problem of understanding the system as in most cases; the systems are not well documented.

To solve these types of problems, researchers are trying to find ways to make the unclear information to a clear picture. But most of the researchers concentrate only on design recovery and use the information in architecture and design phase of the life cycle. In the example of this paper from several researches show that it is possible to get back requirements from the reverse engineering process. In this paper, we have argued and showed advantages of combining the requirements received from reverse engineering process and the normal requirements engineering process. The modified diagram of the whole process is also provided with this paper.

In this paper, we have proposed the possibilities of combining and comparing the requirements engineering outcomes with that of reverse engineering. Because of shortage of time in the term, we could not apply it to any real situation. But based on the advantages we have said and from the theoretical point of view, it will surely improve the quality of the round tripped software.

Though we are talking about getting requirements from reverse engineering outcome, a very few research have been done in this area. So future research can be done on finding more and effective techniques of getting requirements from reverse engineering outcomes. Moreover the proposed idea is not yet applied to any real situation. So to prove its affectivity and efficiency, it should be applied in real situations and based on the result, the proposed model can be revised.

9. Acknowledgement

The research described in this paper was supported by University IT Research Center Project of the Ministry of Information and Communications, Korea.

10. References

[1] Sommerville, I., Software Engineering, Fifth Edition, Addison-Wesley Publishing Co. Inc., Wokingham, England, pp. 700-712, 1995.

[2] Mayrhauser, A.V.; Vans, A.M., “Program Understanding: Models and Experiments”, Advances In Computer, 40, pp. 1-38, August 1995.

[3] Chikofsky, E.J.; Cross, J.H., II , “Reverse Engineering and Design Recovery: A Taxonomy”, IEEE Software, 7(1), pp.13-17, Jan. 1990.

[4] Brown, A.J., “Specifications and Reverse Engineering”, Journal of Software Maintenance: Research And Practice, 5(3), pp. 147-153, 1993.

[5] Richner, T.; Ducasse, S., “Recovering High-level Views of Object-Oriented Applications from Static and Dynamic Information”, Proceedings of International Conference on Software Maintenance, IEEE Computer Society Press, pp. 13-22, 1999.

[6] Dayani-Fard, H.; Jurisica, I., “Reverse Engineering by Mining Dynamic Repositories”,Proceedings of the 5th Working Conference On Reverse Engineering, IEEE Computer Society Press, pp. 174-182, 1998.

[7] 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 Vol. 2

[8] Lee Dan H.: Overview of Requirements Engineering, Software Requirements Engineering, Lecture Notes, ICU, Module 1.2

[9] László Angyal, László Lengyel, Hassan Charaf: An Overview of the State-of-The-Art Reverse Engineering Techniques, proceedings to 7th International Symposium of Hungarian Researchers on Computational Intelligence.

[10] Yijun Yu, Yiqiao Wang, John Mylopoulos, Sotirios Liaskos, Alexei Lapouchnian, Julio Cesar Sampaio do Prado Leite: Reverse Engineering Goal Models from Legacy Code,RE 05: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05)

[11] Chikofsky, E.J.: Learning From Past Trial and Error: Some History of Reverse Engineering to Requirements, presentation slide in RETR Workshop at WCRE,05