Review on Reverse Engineering of Legacy Code and Extraction of Class Diagrams

Upasana Choudhary
Student (M.Tech)
Department of computer science
Sanghvi Institute of Management & Science,
Indore

Maya Yadav
Assistant professor
Department of computer science
Sanghvi Institute of Management & Science,
Indore

ABSTRACT
We are living in the era of software and Information technology. Where Reverse engineering has a big role in the up-gradation and maintenance of old software. Precisely if it comes to the reverse engineering of legacy code; so many tools and software are available in the market but still market requirement for reverse engineering of existing codes is unfulfilled. Present paper focus on the various researches published in consecutive years on the same topic. In this study we have covered legacy code and their reverse engineering feasibility as per the cost and time perspective, generation of class diagrams, various problems faced by the different researchers and possible solutions suggested. Conclusion of the study is that we need to do some more experiments to show the class diagram and their relationship and extracting method level dependency while performing reverse engineering of a legacy code by using different language tools and techniques.

Keywords
Legacy Code, Class Diagram, Dependency

1. INTRODUCTION
Doing maintenance of old software which is developed by old unknown software developers is very typical and difficult job. Because understanding and doing any modification in these codes, which are written in thousands of lines using uncountable variables having multiple relationship among their selves requires a high level of intellect and highly skilled programmers. To make it easy Reverse engineering tools provide useful high level information about the system being maintained or upgraded.

2. BACKGROUND
The legacy code is nothing but the existing codes and class diagram support the program understanding activities, can drive refactoring and restructuring interventions and can be used to assess the traceability of the design into the code. Therefore, it is very important that the diagrams produced from legacy code should be accurate, i.e., exploit all static information present in the code in order to reverse engineer entities and relations. There are so many tools are available in the industry for reverse engineering of legacy codes like Rational Rose, jGRASP, Eclipse and NetBins that will help the developer with this undertaking. To acquire deep knowledge of any subject a deep study of related topic is very important hence present paper is the sequence of wide literature review on reverse engineering published in IJCA as Review on Reverse Engineering Techniques of Software Engineering [8]. We have thoroughly discussed UML diagrams and related study. And we found that there is a need of experiments to be done on various theatrical concept proposed in various research papers.

3. MOTIVATION
The important reason of focusing in reverse engineering of legacy code class diagrams and their relationship is the wide research scope available in this particular field. Still researchers are trying to achieve more and more accurate information. Even though some professionals found inapplicability of reverse engineering of legacy code for very long codes [2], they are accepting the possibility and usefulness of good reverse engineering tools in near future.

4. SURVEY
In the present study some experimental studies are included. First, Reverse engineering of legacy code framed in C++ language to show interrelationship between various classes and second one is the Reverse engineering of PL/SQL legacy code in the steel making domain which made it easier to show a clear picture for stakeholders [1 & 6]. Legacy code and class diagrams are the majorly covered by [1, 2, 3, 4 & 7] where accuracy and interrelationship of various class diagrams produced by various tools are more important than just creating a class diagram with very less information [1, 3 & 7]. Details of every individual research are shown below in a sequential order.

Ati Jain and Swapnil Soner **[1]** did an experiment to perform reverse **engineering on an existing source** code as a part of their project work to implement and upgrade the legacy code framed in C++ language. In this paper the issues about recollecting design information, such as finding variables, functions and classes are discussed in depth. The reverse engineering process used to **extract the Data flow diagrams, Control flow diagrams and class diagrams**. Author has developed a Reengineering method to automate the extraction of DFDs, CFDs and class diagrams from any legacy C++ code. Various problems experienced during the experiment and respective resolution has been reported.

Since it is well understood that all codes are not written by using convention naming, also there is no certainty of a code being well structured. Hence the very first problem discussed is to identify a variable in a code. And the solution suggested for this problem is to provide the user with a list of all probable input and output variables so that user or expert can select one and try to find out if there is any rule implemented with that data or not. Moreover a user can also be provided with a facility to store the description with a variable if needed.

Differentiating between two basic elements which serves to control the software execution i.e. low-level control structure and high level structure is another issue explained. And the Solution proposed was that the former should be included as part of the processing described in the detailed design [], the latter needs to be recorded in a control flow diagram and its control specification. The technical reason behind this is to recover too much of the control structure.

And the very important and key problem discussed is how to show dependency of various classes on each other. This problem was tackled by using various methods i.e. inheritance, aggregation and association to find the relationship between various classes, for detailed explanation refer [1].

Bruce W. Weide and Wayne D. Heym [2] discussed the feasibility of reverse engineering of a long legacy code in terms of cost effectiveness and attainability. Aim of the author was to gain a sufficient understanding of the whats, hows, and whys of the legacy system as a whole that its code can be re-engineered to meet new requirements on behaviour, performance, structure, system dependencies etc.. Observations and implications, the nature of the reverse engineering task and the intractability Results were discussed thoroughly. Author of this paper has reported that the Reverse engineering of large legacy systems is intractable in the following sense: Given real legacy code, the time required to show the validity of a proposed explanation for why it exhibits any significant system-level behaviour is at least exponential in the size of the source code. And the key point can be noted from this research that it does not mean that the task of re-engineering of a legacy code is impossible. It means that it is prohibitively costly for large legacy systems.

Giuseppe Antonio Di Lucca et al. [3] has given brief of a method for recovering an O-O model together with the objects and relationships among them. Author has implemented an approach integrates the results of reverse engineering of both the procedural code and the persistent data stores of the system, and utilized a number of heuristic criteria to display a class diagram. A preliminary experiment carried out to verify the results from proposed method on a COBOL medium-sized system. The effectiveness of the proposed method was evaluated on the basis of recovered adequate (about 90 %) abstractions (both classes and relationship among them). Eventually it is reported that proposed method is giving encouraging results. Measure adequacy is given as,

$$Adequacy\,\frac{|N|}{|M|} * 100\%$$

Where M is the set of abstractions selected by the method, N is the subset of components in M that can be associated with a concept of the application domain, $|N|$ and $|M|$ denote the number of components in the M and N sets. However consideration of aspects like completeness and precision of abstractions along with the adequacy could have been results in accurate measurement of effectiveness while investigation of the proposed method effectiveness is carried out.

Mariano Ceccato et al.[5] have inferred a structured data model in such a language as part of a migration of eight million lines of code to Java and discussed the common idioms of coding that were observed. They explained reverse engineering of a structured data model from the unstructured model provided by BAL, using program transformation. Also described basic cases that may occur in BAL code and how they map them to Java. Conclusion given by author is that the proposed approach is quite general and can be applied to a number of programming languages that support arbitrary data layout in memory.

Martin Habringer et al. [6] developed tool to support the reverse engineering of PL/SQL code into a more abstract and comprehensive representation. Which can be more useful as an input for stakeholders to manually analyze legacy code, so that it will become easy to identify obsolete and missing business cases, and, finally, to support the re-formation of a new system. Through this study they have shown the results of reverse engineering PL/SQL legacy code in the steel making domain. Data flow from the production data to the result data containing error messages is analyzed on the basis of an abstract syntax tree, The results from this analysis is used to produce expected high-level representation as shown in figure.1.

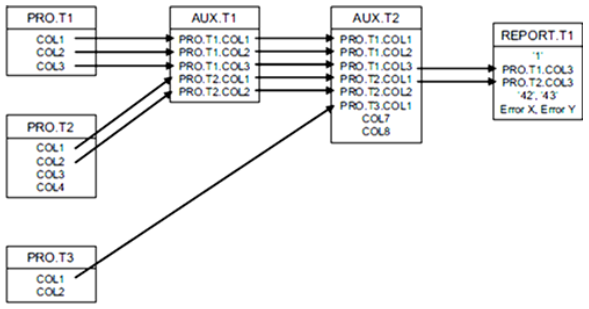

Fig 1. Data Flow Graph generated by our tool

While reading the paper two important points were noted, very First, the high-level representation generated by the proposed tool, which was basically customized for the analyzed legacy code, could be able to give a clear picture for stakeholders on comparing to general-purpose representations which can be created by conventional reverse engineering tools. And second conclusion reported is, generating adequate representations needs a faster feedback from stakeholders and sophisticated analysis techniques such as symbolic execution.

Paolo Tonella and Alessandra Potrich [7] proposed an algorithm in order to achieve more accuracy of the UML class diagram generated from the code. Specifically, author has discussed the unclear and missing inter-class relations in extracted class diagrams. The proposed approach was applied to several software components developed at CERN.

Also improvement of the relations in the UML class diagram recovered from the code was presented. In the proposed tool, Output of the type interface is used as a basis for computation of the relation between classes. Results showed in the study highlights that a substantial improvement is achieved when the container type information is refined with the inferred data. The number of relations otherwise missed is relevant and the connectivity of the associated class diagrams is radically different when containers are considered.

The positive outcome is that the class diagram is more accurate than the one generated by other existing tools, it is reported in research that the Experimental results suggest that there is a relevant difference between the class diagrams which exploit the inferred container type information with respect to those that do not. However it was observed that a large fraction of relations is missed if container types are not determined. Moreover indirect benefits on the program understanding activities are hard to predict. It is reported that the diagrams are much closer to the mental model of the application; therefore it can be used for the higher level of understanding and for its evolution.

5. CONCLUSION

The study shows that reverse engineering of legacy code is a good platform to bring a dynamic change in the field of IT industry as well as other core business industries [6]. But as it is associated with the finding of uncountable variables and identifying relationship among them, hence it is very complicated in nature. One possible way of solving this complicated puzzle is to draw class diagrams that too with detailed information like dependency based on class and method level [1& 3]. However some studies shows that doing reverse engineering of long codes is very costly and time consuming hence seems unreliable. It does not mean that the task of re-engineering of a legacy code is impossible. It means that it is prohibitively costly for large legacy systems [2].

Above conclusion is given by respective authors on the basis of their limited experience and experiments in this particular field. Hence conclusion of the present study is that we need to do some more experiments on reverse engineering of legacy code with the help of some other high level language tools like C#, Java etc. to rectify the problems unclear information associated with class diagrams obtained with various tools.

As a part of our post graduation project work, we will come with one more experiment on showing class diagrams with more and accurate information by using any other language tool discussed above.

6. ACKNOWLEDGMENTS

We are very thankful to all the experts who have contributed towards development of the manuscript.

7. REFERENCES

[1] Ati Jain, Swapnil Soner, 2010. Reverse Engineering”: Extracting Information from C++ code. International Conference on Software Technology and Engineering(ICSTE), IEEE ISBN No:978-1-4244-8666-3, Vol.1, P.154-158

[2] Bruce W, Weide, Wayne D.Heym 1995, Reverse Engineering of Legacy Code Exposed. Proceedings of the 17th International Conference on Software Engineering (ICSE’95) 0270-5257/95, IEEE ISSN No. 0270-5257, P.327

[3] Giuseppe Antonio Di Lucca, Anna Rita Fasolino, Ugo De Carlini 2000, Recovering Class Diagrams from Data-Intensive Legacy Systems. IEEE 1063-6773100

[4] Jianjun Pu, Zhuopeng Zhang, Yang Xu and Hongji Yang 2005, Reusing Legacy COBOL Code with UML Collaboration Diagrams via a Wide Spectrum Language. IEEE 0-7803-9093-8/05

[5] Mariano Ceccato, Thomas Roy Dean, Paolo Tonella, Davide Marchignoli 2008, Data model reverse engineering in migrating a legacy system to Java. 15th Working Conference on Reverse Engineering, IEEE ISSN No.1095-1350

[6] Martin Habringer, Michael Moser, Josef Pichler 2014, Reverse Engineering PL/SQL Legacy Code: An Experience Report. IEEE International Conference on Software Maintenance and Evolution 1063-6773/14

[7] Paolo Tonella, Alessandra Potrich 2001, Reverse Engineering of the UML Class Diagram from C++ Code in Presence of Weakly Typed Containers. IEEE ISSN No. 1063-6773

[8] Upasana Choudhary, Maya Yadav, 2015, Review on reverse engineering techniques of software engineering. IJCA 119(14):7-10