Extracting Use Case Design Mechanisms via Programming based on Reverse Engineering Technique

Haeun Kwon[1], R. Young Chul Kim[1*]
[1]SE Lab, Hongik University Sejong Campus, Korea, 339-701
{kwon, bob}@selab.hongik.ac.kr

Abstract— In this time, there are gradually increasing the damages caused by defects in software. This increases the demand for high-quality software [1]. The previous researches [2,3,4] proposed a tool-chain mechanism, which automatically performs code visualization through reverse engineering technique. This paper suggests to extract possible designs such as object diagram, and sequence diagram via programming on code analysis based on an extension of the previous researches. In other words, it presents class, methods, and coupling relationship between classes, and extraction mechanism for sequence diagram. Thus we can trace designs via programming based on requirements. Therefore, it is expected for synchronization of codes, design, and documents in the development lifecycle.

Keywords: Software Visualization, Class Diagram, Sequence Diagram, Use Case Mechanism;

I. INTRODUCTION

Today, it leads to an increase in the ranges and functions of software, and also rises the magnitude of the damage caused by software defects. Therefore, this increases the demand for high-quality software [1]. One way uses the software development process and methodology to produce high-quality software. In the previous researches [2,3,4], we applied to the redevelopment process of legacy software to extract UML class diagram via code with a software visualization technique, and the method for the coupling relationship between classes.

However, the class diagram of UML Diagrams is included with dependency, association and inheritance relationship between classes. But this diagram represents a difficulty in identifying the order in which the object is created, or the sequence of method calls. Therefore, this paper attempts to extract sequence diagram in the object-oriented paradigm to understand the operation of the entire system. Furthermore, it seeks to propose quality management techniques in the use case paradigm by extracting coupling graph, class diagram and sequence diagram on a single integrated development process based on reverse engineering.

This paper is organized as follows. Chapter 2 introduces our software visualization and methods for management of coupling relationship between classes mentioned earlier [2,3,4]. Chapter 3 describes the entire process of extracting the use case paradigm in source code. It also explains the integrated process of extracting coupling graph, class diagram, and sequence diagram. However, it cannot be identified only with the class diagram mechanism such that the order in which the object is created, or the sequence of method calls. And then it is also difficult to know the operation of the entire system. To solve this problem, this paper attempts to extract the sequence diagram and class diagram in the object-oriented paradigm. In addition, the coupling corresponds to the message in the sequence diagram, which allows it to trace the process. Chapter 4 shows the proposed process to apply the actual code. Lastly, Chapter 5 mentions conclusions and future work.

II. RELATED WORK

2.1 Software Visualization

One of the characteristics of software is invisibility since it has no physical properties. Because of this, there exist difficulties in identifying the architecture or fining the potential error prior to the development. Software visualization is a technique to overcome the invisibility of software. In other words, it is a technique designed to improve understanding by visually representing the software. The software visualization is divided into architectural visualization, runtime's behavior visualization, and code visualization. The architectural visualization performs a static analysis of software at a structural view point of software. The runtime's behavior visualization monitors the operating procedures of software. Lastly, the code visualization uses easily understanding the internal code structure in a graph [5]. This paper mentions the architectural visualization, which extracts the use case paradigm through a static analysis of object-oriented code.

2.2 A coupling oriented class diagram

The metrics is one way to create the high-quality software [6]. It is measuring the quantitative elements in various software fields such as cohesion, coupling, complexity, and the number of line (LOC) [7]. Accordingly, we proposed the method for coupling between classes on the basis of software visualization techniques [2]. It measures the coupling between classes, and extracts the class diagram, one of design documents. To be more specific, we need the dependency, association, and inheritance relationship between classes. Figure 1 shows a part of the coupling oriented class diagram. First, it confirms to be a dependency relationship from COperationFactorSet class to COperationFactor class, and shows $(1*6*1.7) = 10$ on the coupling relationship. This value is calculated with coupling indicators in Table 1. Our indicators include six items of data, stamp, control, external, common, and content. In addition, there has assigned lower point & weight for weak coupling between modules, and higher point & weight for strong coupling in accordance with the principles of coupling. The lowest coupling is data coupling with 1 and 0.5 of a point and a weight respectively, followed by stamp, control, external, common and content coupling. Table 1 shows the detailed points and weights of them. In the end, the coupling value is represented by the product of weight, point and the number of the coupling relationship. For example, there is one content coupling between COperationFactorSet class and COperationFactor class in figure 1. Figure 1 shows an example of a coupling oriented class diagram.

TABLE 1. COUPLING POINT AND WEIGHT

	Data	Stamp	Control	External	Common	Content
Point	1	2	3	4	5	6
Weight	0.5	0.5	1	1.3	1.5	1.7

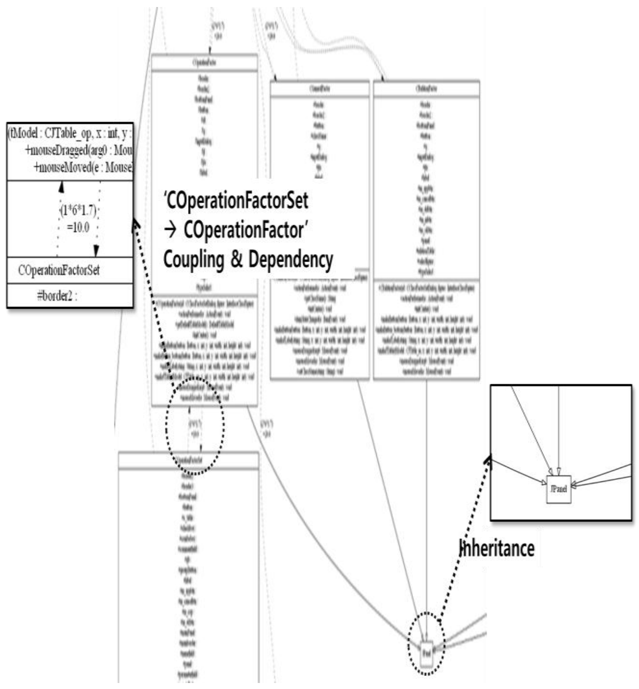


Figure 1. A coupling-oriented class diagram.

III. MATERIALS AND METHODS

Figure 2 shows the proposed use case extraction process. As in the previous studies [2], this process is composed of four stages of code analysis, storing in database, architecture analysis, and visualization. First, a code is analyzed using a parser at the code analysis stage. In this stage, the code is decomposed into various components such as classes, methods, variables, method calls, and inheritance relationship. Second, the decomposed components are classified in the database at the storing database stage. This is to facilitate the architecture analysis through the query in the next stages. Third, it is reinterpreted the classified information extracted at the architecture analysis stage. The coupling data, class data and sequenced data are the intermediate deliverables for extracting coupling graph, class diagram and sequence diagram, respectively. For example, the methods are called between classes should be analyzed to extract the sequence diagram. This can be identified by the class data. As a result, the class diagram should be extracted first prior to the extraction of the sequence diagram. In addition, the information obtained from the coupling data needs to apply to the sequence diagram. Lastly, the interpreted information (intermediate deliverables) is converted to the script at the visualization stage. Finally, the converted script is entered into the view composer to obtain the final deliverables.

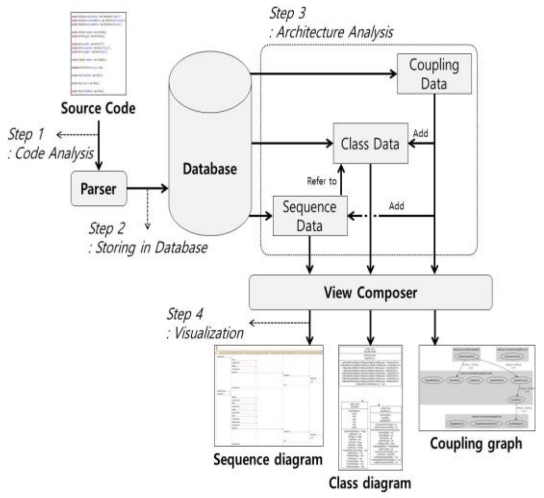
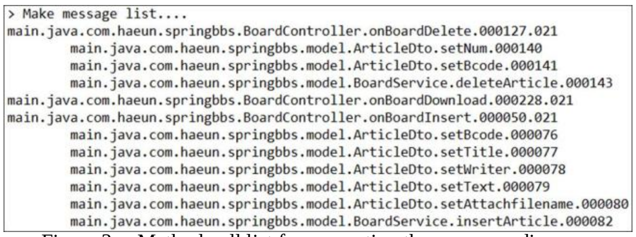


Figure 2. A Generation process of usecase paradiam.

IV. CASE STUDY

It describes the proposed process with the actual code, to and examines the results. The code is an example of a board application written in Java language. In order to extract the class & sequence diagram, the list of methods should be extracted from the code. This information is already calculated in the process of extracting the class diagram of the previous studies. Therefore, the relationship between method calls is created as an adjacency list by referring to the class data in Figure 2. The adjacency list is shown in figure 3. It represents the list of ChildMethods called from inside of the method with the method as a main unit. For example, three methods of setNum, setBcode and deleteArticle are called from the onBoardDelete method. When the methods are called, it should include the information on which lines of the actual code are. This distinguishes with the order in which the methods are called. That is, setNum, setBcode and deleteArticle methods are called from the 140th, 141st and 143rd line of the code. Figure 3 shows method call list for generating the sequence diagram.

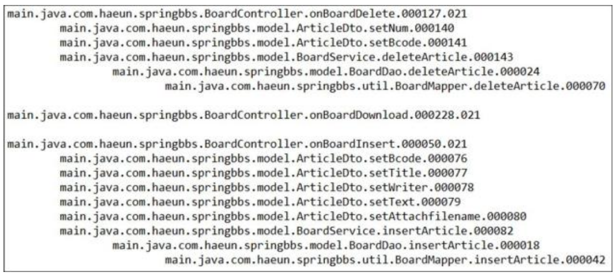
Next, we use the depth-first search to traverse the calculated adjacency list. It notes the line numbers of methods that are called when traversing is in order. For example, setBcode method called from the 141st line should not traverse prior to the setNum method called from the 140th line.



```
> Make message list....
main.java.com.haeun.springbbs.BoardController.onBoardDelete.000127.021
        main.java.com.haeun.springbbs.model.ArticleDto.setNum.000140
        main.java.com.haeun.springbbs.model.ArticleDto.setBcode.000141
        main.java.com.haeun.springbbs.model.BoardService.deleteArticle.000143
main.java.com.haeun.springbbs.BoardController.onBoardDownload.000228.021
main.java.com.haeun.springbbs.BoardController.onBoardInsert.000050.021
        main.java.com.haeun.springbbs.model.ArticleDto.setBcode.000076
        main.java.com.haeun.springbbs.model.ArticleDto.setTitle.000077
        main.java.com.haeun.springbbs.model.ArticleDto.setWriter.000078
        main.java.com.haeun.springbbs.model.ArticleDto.setText.000079
        main.java.com.haeun.springbbs.model.ArticleDto.setAttachfilename.000080
        main.java.com.haeun.springbbs.model.BoardService.insertArticle.000082
```

Figure 3. Method-call list for generating the sequence diagram.

The traversal results are stored in a tree structure in figure 4. But it omits 'Actor node', which is a root node. Figure 4 shows method call tree for generating the sequence diagram.



```
main.java.com.haeun.springbbs.BoardController.onBoardDelete.000127.021
        main.java.com.haeun.springbbs.model.ArticleDto.setNum.000140
        main.java.com.haeun.springbbs.model.ArticleDto.setBcode.000141
        main.java.com.haeun.springbbs.model.BoardService.deleteArticle.000143
                main.java.com.haeun.springbbs.model.BoardDao.deleteArticle.000024
                        main.java.com.haeun.springbbs.util.BoardMapper.deleteArticle.000070

main.java.com.haeun.springbbs.BoardController.onBoardDownload.000228.021

main.java.com.haeun.springbbs.BoardController.onBoardInsert.000050.021
        main.java.com.haeun.springbbs.model.ArticleDto.setBcode.000076
        main.java.com.haeun.springbbs.model.ArticleDto.setTitle.000077
        main.java.com.haeun.springbbs.model.ArticleDto.setWriter.000078
        main.java.com.haeun.springbbs.model.ArticleDto.setText.000079
        main.java.com.haeun.springbbs.model.ArticleDto.setAttachfilename.000080
        main.java.com.haeun.springbbs.model.BoardService.insertArticle.000082
                main.java.com.haeun.springbbs.model.BoardDao.insertArticle.000018
                        main.java.com.haeun.springbbs.util.BoardMapper.insertArticle.000042
```

Figure 4. Method-call tree for generating sequence diagram.

The sequence diagram is made through the coupling data used to extract the existing coupling graphs. The existing coupling graphs show only the information of the coupling relationship between classes. On the other hand, the sequence diagram represents the information decomposed by a unit of the methods. For example, if there is the coupling of 4 between class A and class B, it should be represented as the stamp coupling value 1 due to invocation of method and the control coupling value 3 due to invocation of method b. This is achieved through mapping between the coupling data and the traversal tree in Figure 4.

Finally, the sequence data is translated in script, which is entered in the View Composer to obtain a sequence diagram in figure 5. The sequence diagram is made by extending a series of traversal processes from onBoardDelete method to deleteArticle method of BoardMapper class. Thus, it visualizes the traversal tree of figure 4. In addition, Figure 6 shows an extended representation of the coupling-applied sequence diagram. It is distinguished with the '_' on both the method name called between classes and the corresponding coupling type. For example, setNum method and setBcode method are called from BoardController class to ArticleDto class. Both methods correspond to a data coupling. On the other hand, deleteArticle method called from BoardController class to BoardService class is a stamp coupling.

V. CONCLUSIONS

This paper aims to extend a previous tool-chain mechanism, which extracts automatically use case designs via code visualization based on reverse engineering technique. This is, we extract possible designs such as object diagram, and sequence diagram via programming on code analysis.

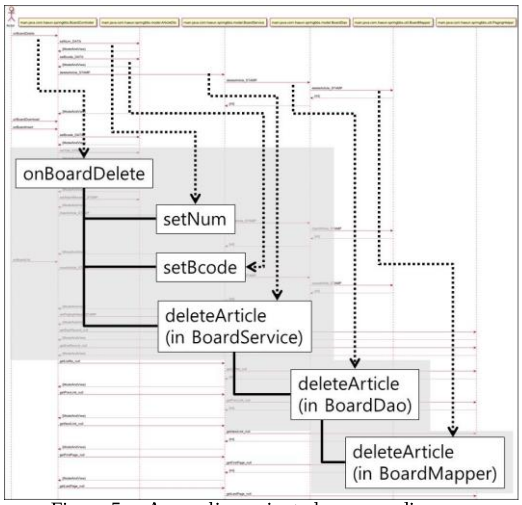


Figure 5. A coupling-oriented squence diagram.

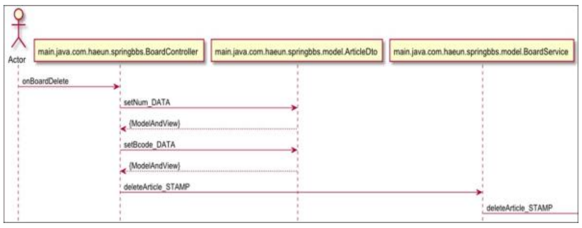

Actor
main.java.com.haeun.springbbs.BoardController
main.java.com.haeun.springbbs.model.ArticleDto
main.java.com.haeun.springbbs.model.BoardService
onBoardDelete
setNum_DATA
(ModelAndView)
setBcode_DATA
(ModelAndView)
deleteArticle_STAMP
deleteArticle_STAMP

Figure 6. An expansion of sequence diagram.

Therefore, it is possible to perform a comparative analysis between design documents and codes in the software development process. In near future, we will trace possible designs via programming based on requirements.

Acknowledgment

This work was supported by the Human Resource Training Program for Regional Innovation and Creativity through the Ministry of Education and National Research Foundation of Korea (NRF-2015H1C1A1035548) and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2013R1A1A2011601).

REFERENCES

[1] NIPA Software Engineering Center, 2013, "2013 Software Engineering White Paper", Korea.
[2] Haeun Kwon, Bokyung Park, Keunsang Yi, Young B. Park, Youngsoo Kim, R. Youngchul Kim, 2014, "Applying Reverse Engineering through extracting Models from Code Visualization," KIPS, vol. 21, no. 2, pp. 650-653.
[3] Haeun Kwon, Hyun Seung Son, Chae Yun Seo, Youngsoo Kim, Byung Ho Park, R. Younchul Kim, 2014, "A study on Comparing Object Oriented Paradigm with the Cohesion and Coupling mechanism between Traditional modules," KCC 2014, pp. 556-558.
[4] Bokyung Park, Haeun Kwon, Hyeoseok Yang, Soyoung Moon, Youngsoo Kim, R. Youngchul Kim, 2014, "A study on Tool-Chain for statically analyzing Object Oriented Code," KCC 2014, pp.463-465.
[5] Thomas Ball, Stephen G. Erik, Bell Laboratories, April 1996, "Software Visualization in the Large," IEEE Computer Society, Vol. 29, Issue 4.
[6] Arun Rai, Haidong Song, Marvin Troutt, Jaunary 1998, "Software Quality Assurance: An Analytical Survey and Research Prioritization," Journal of Systems and Software, Volume 40, Issue 1.
[7] Ince, D., May 1990, "Software Metrics: Introduction", Information and Software Technology, Volume 32, Issue 4.