International Journal of Advanced Smart Convergence Vol.10 No.3 89-96 (2021)
http://dx.doi.org/10.7236/IJASC.2021.10.3.89

IJASC 21-3-12

Automatic UML Design Extraction with Software Visualization based on Reverse Engineering

Se Jun Jung[1], Janghwan Kim[2], Won Young Lee[3],
Bo Kyung Park[4], Hyun Seung Son[5], R. Young Chul Kim[6]

[1,2]M.S, Software Engineering Laboratory, Department of Software and Communication Engineering, Hongik University, Korea
[3]M.S, Defense Korea Agency for Technology and Quality
[4]Assistant Professor, Dept. of Computer Education, ChinJu National University of Education,
[5]Assistant Professor, Dept. of Computer Engineering, Mokpo National University
[6]Professor, S.E Laboratory, Dept. of Software and Communication Engineering, Hongik University,
E-mail : {[1]bvcx79, [2]janghwan, [3]leewy, [6]bob}@selab.ac.kr, [4]parkse@cue.ac.kr, [5]hson@mokpo.ac.kr

Abstract

In various areas of the 4th industry, a big issue is software quality enhancement for stability and reliability of the smart software systems. After revising software promotion law at 2020, we must clearly define requirements and separate design parts and implementation parts of an all public software development contracts. In this study, we need to validate whether the final implementation of software is followed by the original design or not. To do this, we consider the design restoration through software visualization based on reverse engineering. Therefore we propose an UML design extraction and visualization method based on reverse engineering. Based on this, we may validate whether it is implemented according to the original design, and how much visualizes and includes the code the internal complexity for improvement of software quality.

Keywords: *UML, OOP, Reverse engineering, Software visualization*

1. Introduction

As the knowledge-information society continues, the fields in which digital information is used in various ways are increasing. As software that uses digital information appears in various fields, such as robots, artificial intelligence, and the Internet of Things, the scale of software grows and becomes more complex, technology is needed to achieve sophistication of software quality. In addition, since software programs require continuous development and patching, a maintenance system that manages the implemented software for high quality is important for complex software. However, due to invisibility, which is a characteristic of software, it is difficult to measure the complexity of software and to reflect requirements, making it difficult to maintain software [1]. Moreover, in IT ventures or small and medium-sized enterprises, the lack of a maintenance system due to frequent developer turnover, frequent changes in requirements, and lack of design documents is a big problem [2]. Those complex parts of the software that are not discovered due to the invisibility of the software can potentially cause bigger problems and additional costs for the company.

To improve these problems, this paper proposes the ways to improve the software quality by recovering the

Manuscript Received: July. 15, 2021 / Revised: July. 20, 2021 / Accepted: July. 22, 2021
Corresponding Author: bob@selab.hongik.ac.kr
Tel: +82-44-860-2477, Fax: +xx-xx-xxx-xxxx
Professor, Software Engineering Laboratory, Department of Software and Communication Engineering, Hongik University, Korea

design document through the software implemented based on the design to improve and verify the completeness of the design. In the design phase of the software development lifecycle, errors can be identified, corrected, and reduced at a lower cost than the implementation phase. In that case, verification work can be performed in advance through the method suggested by the software engineer, and errors can be identified and corrected in early stage with the verification result. Then, the company can expect to reduce the cost for the maintenance of the software. This paper is organized in the following order. In Chapter 2, reverse engineering and software visualization methods are mentioned as related studies, and in Chapter 3, the visualization process for UML design restoration and verification based on reverse engineering is introduced. Chapter 4 shows the design restoration of the online shopping mall data system as a case study. Section 5 ends this paper with a conclusion.

2. Related Works

2.1 Software Architecture Visualization and Software Process Visualization

Research on visualization of the software development process through reverse engineering is ongoing. J. Chikofsky researched the forward engineering, reverse engineering, and maintenance of the SW process, which is the core of SW reverse engineering and Tool-Chain [3]. Based on these, Software Reverse Engineering is the operation of analyzing developed software to recover its product such as higher-level documents or design drawings from lower-level products. And the J. Park researched of Tool-Chain, a tool for source code analysis, proposed a Matrix to find the Bad Smell of the code and to visualization [4]. And more, the data structure for extracting the design drawing use-case from the source code or tracking object information has been researched [5]. These studies show advantages in terms of maintenance because information about the software development process can be analyzed at the design level and the overall structure can be seen easily.

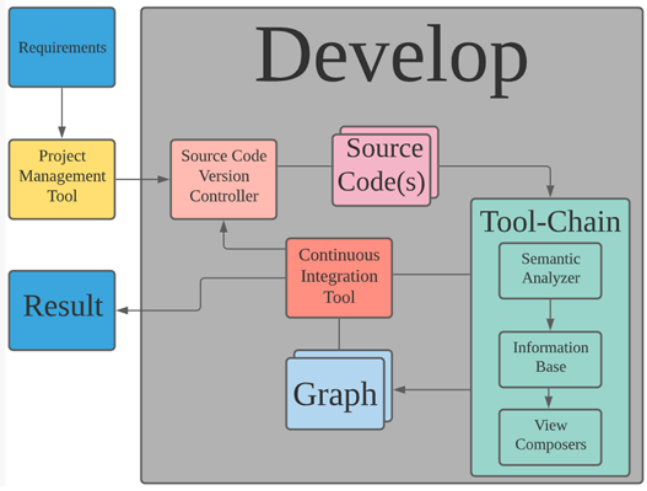


Figure 1. Software Process Visualization

In software development, the invisibility of software is always a problem throughout the development process. This is because these characteristics make it difficult to quickly identify various problems that occur throughout the software development process. So There have been studies to Software Process Visualization. B. Park researched about combines Tool-Chain and process visualization [6]. The software development process can be efficiently managed through software visualization techniques, and the quality of software development can be improved by understanding the entire process. This allows for early detection of software development problems by ensuring transparency in the software development process. In addition, this can reduce the maintenance cost,

thereby reducing the economic burden of the company.

3. UML Designs Extraction

3.1 The Previous Tool-Chain for Software Architecture Visualization

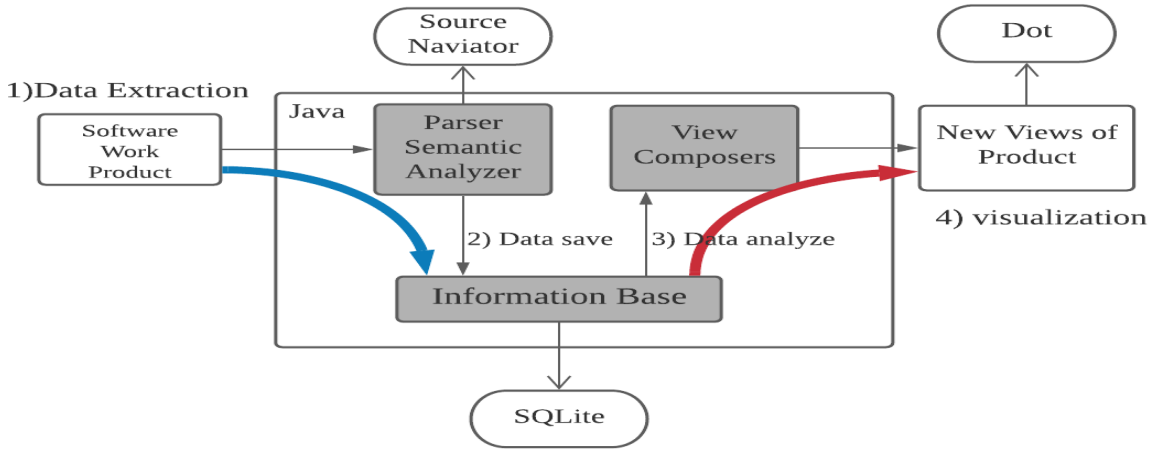


Figure 2. Previous Tool-chain of Software Architecture Visualization[7]

Figure 2 shows Software Architecture Visualization Tool-chain that used in previous researches. Source Navigator is an open-source tool that analyzes both C and Java as a parser in the existing software architecture visualization Tool-Chain. As the first step of this visualization, a file is created as a result of analyzing the target source code using Source Navigator. Although a total of 29 source code analysis files are provided as types of files, there are few types of practically useful analysis result files. When the target source code is written in Java code, there are fewer than 10 useful information can be derived, and there is a limit to obtaining specific information or additional information of the target source code [8]. In addition, in order to apply the analyzed result file to the Tool-Chain, an external program called 'dbdump' must be used, and the analyzed content is stored in the database almost as it is. In other words, since unnecessary work is required to interpret and move data, when the size of the target code is large, more time is required in the program for information analysis, movement, and storage. Moreover, since information is stored in the database, there is a disadvantage that new information must be extracted only from the information stored in the database, and additionally, the process of searching and extracting through a query is additionally required for this extraction operation. Therefore, there is a problem in terms of optimization of the query statement in the process of using the query statement. To improve on these problems, we need a parser that can directly get the information we need and can provide more information.

3.2 Our Proposed Tool-Chain for Software Visualization

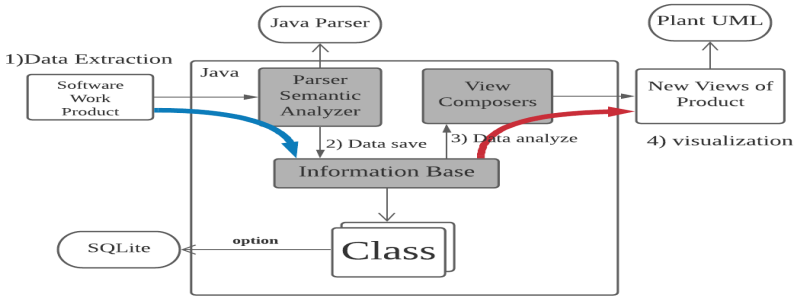


Figure 3. Proposed Tool-Chain for Software Visualization

Figure 3 shows the schematic diagram of the Tool-Chain within the software architecture visualization with the method proposed in this paper. The source navigator, which was used as a static analysis tool of analyzing source code, is replaced with a static analysis for Java called 'Java parser'. Instead of the file extracted by the source navigator, the Java parser creates information in AST (Abstract Sytax Tree) form and stores the extracted results in a Java object. The AST has a top-down structure, allowing top-down access to specific detailed information. You can use the supported libraries to find and work with the specific syntax that you want in the AST. Therefore, the new Tool-Chain uses the JavaParser object to create an AST as an object rather than a file, and extracts the desired data using the created object. In addition, necessary information and information obtained through analysis are processed and stored as objects. This allows for more flexible processing than parsers that used to use and optionally stores the results in a database. By using this method, it is possible to increase modularity by replacing the complexity problem of the existing query statement and to obtain data directly from the object, so the additional tasks that we metioned above can be significantly reduced. Also, Source Navigator imported from legacy tools runs as a process, while Java parser uses Java objects. This tool can also be run together as a process in a Tool-Chain running in Java Eclipse. In other words, since it operates only within the JVM, problems such as specifying the path of the existing source navigator and accessing the OS are eliminated, increasing the modularity of the Tool-Chain.

We choose to create graphs using PlantUML instead of GraphViz as a visualization tool. Both Graphviz and PlantUML generate plots automatically via the Dot language. The tool supports a variety of diagrams, including sequence diagrams, use case diagrams, class diagrams, and more. PlantUML is an open-source extension of Graphviz that provides more convenient support for UML drawing. PlantUML can easily visualize various diagrams by changing the shape of the Dot language for each diagram. It can be applied to Tool-Chain in the same way as GraphViz, and you can use GraphViz's dot language as needed. Therefore, the new Tool-Chain handles all data analysis and storage in the JVM. Therefore, the parsing step is simple compared to the previous Source Navigator and DB installation and connection. DB can be selectively applied, and most processing is possible with one Java program, and data to be analyzed can be directly defined and extended.

3.3 Design restoration using Java parser and PlantUML

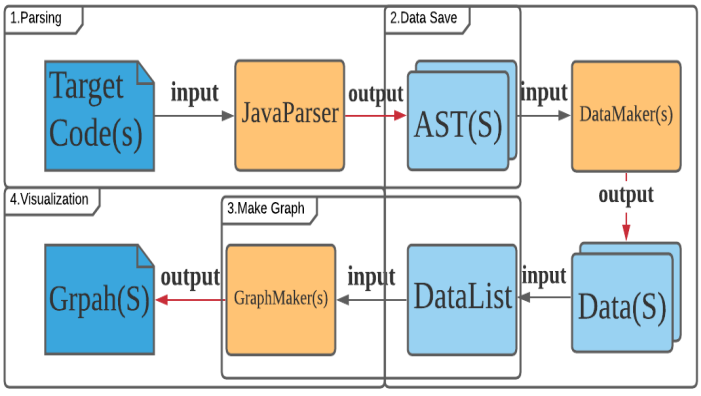


Figure 4. Steps for New Tool Chain

Figure 4 shows the Software Architecture Visualization Tool-Chain process to recover desgin in UML form from source code that is written in Java. The following shows the Tool-Chain process using Java parser and PlantUML step by step.

Step 1. First, when the program is executed, the Tool-Chain reads the configuration file, the library to be used, the path for the target source code. And then, it initializes the Java parser object. This initialized parser parses the target source code and generates AST after a static analysis of the source code. In general, program will generate one AST per one Java file.

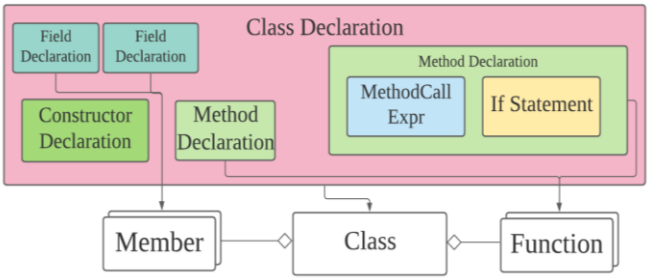


Figure 5. AST Structure diagram

Step 2. In this step, DataMaker iterates through the generated AST, and it extracts the necessary information that set up in DataMaker. And then, it creates and modifies data objects with that information [4]. It extracts the necessary information from each line of codes to create a data object or add information. DataMaker leverages the AST structure to extract the information that are set. Then, it creates a data object or adds information to objects. The extracted and created data objects are collected and managed in the DataList. This operation is repeated until all files are cycled, creating and updating data objects in the DataList. Figure 5 shows some of the AST structures that mentioned above. Nodes such as methods and constructors exist inside the class Node structure, and MethodCallExpr nodes and If Statement nodes exist inside the method nodes.

Step 3. In this step, the program creates a script file for the visualization using the DataList completed through GraphMaker, a class for design restoration. GraphMaker writes scripts to utilize PlantUML. ClassDgMaker needs class information and information such as methods, references, and inheritance relationships to generate class diagrams. SequenceDgMaker needs information such as the object to call, method calls, and order to create a sequence diagram. Following these rules, GraphMaker completes a single script by appending the specified rules of Dot language and the necessary data to the string.

Step 4. In this step, the program runs plantUML on the OS using Java process object via Tool-Chain. The plantUML draws a graph based on the script that are created in the previous step, and recovers the design for the program from the source code. Then, users review the result with the existing design documentation.

4. Applied Practice with Online Shopping Mall

4.1 Target source code

Apply the proposed method to design a simple online shopping mall data system. The requirements are:

- The shopping mall system stores account and product information.
- Each account has a username, a password, and a balance of account.
- Each product has a name, a price, and product ID.
- Users access to their account via their username and password.
- The user adds the balance to the account.
- The user purchases a product with the balance.

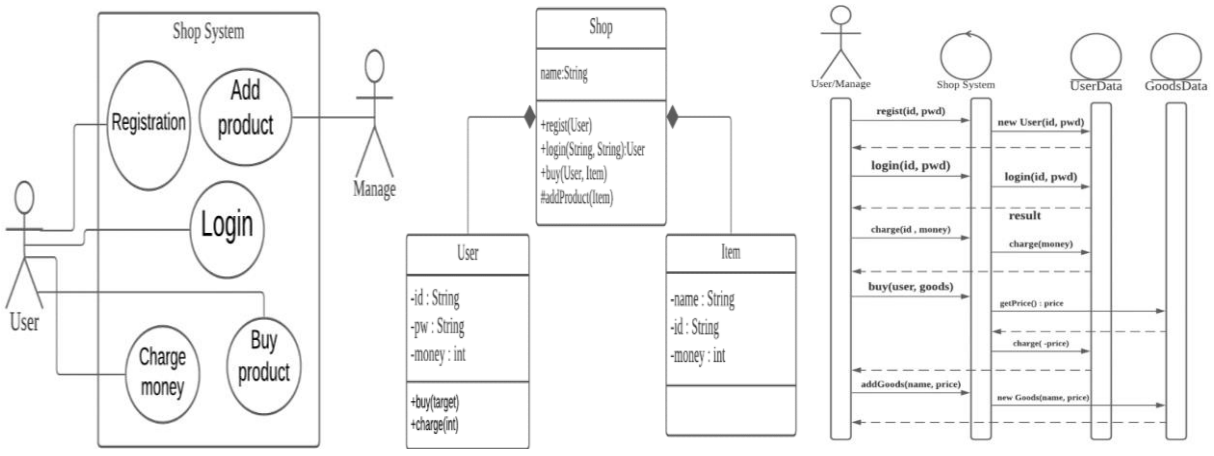


Figure 6. Online Shop System Diagrams

Figure 6 shows the diagram generated according to the requirements. From left, there are Use-Case, Class, and Sequence diagrams.

4.2 Design restoration

Step 1. First, program executes the Tool-Chain by setting the target source code and library information for the Online shopping mall to recover the design for. Figure 7 shows the source code for the 'shop' class, which is the code written according to the design document and requirements of Figure 6, and the 'TestScenario' class for testing this shop class.

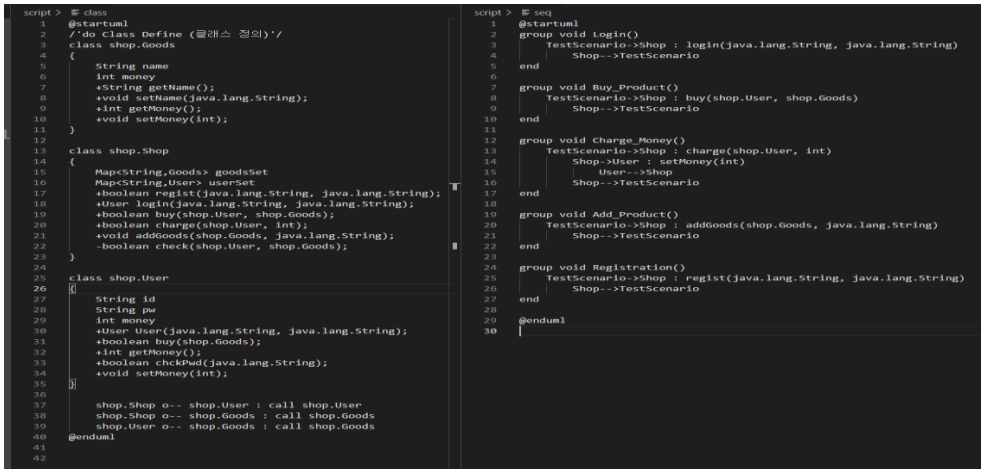
```
public class Shop
{
    Map<String, Goods> goodsSet;
    Map<String, User> userSet;

    public boolean regist(String id, String pwd)
    {
        if( null == userSet.get(id) )
        {
            userSet.put(id, new User(id, pwd));
            return true;
        }else
            return false;
    }
    public User login(String id, String pwd)
    {
        User tryUser = userSet.get(id);
        if( tryUser.chckPwd(pwd) )
            return tryUser;
        else
            return null;
    }
    public boolean buy(User user , Goods goods)
    {
        if( check(user, goods) )
            return user.buy(goods);
        else
            return false;
    }
    public boolean charge(User user, int charge)
    public void addGoods(Goods goods, String id)
    private boolean check(User user, Goods goods)
}
```

```
public class TestScenario
{
    public static Shop shop = new Shop();
    @Test
    public static void Login()
    {
        shop.login("id", "password");
    }
    @Test
    public static void Buy_Product()
    {
        shop.buy(null, null);
    }
    @Test
    public static void Charge_Money()
    {
        shop.charge(null, 0);
    }
    @Test
    public static void Add_Product()
    {
        shop.addGoods(null, null);
    }
    @Test
    public static void Registration()
    {
        shop.regist("id", "password");
    }
}
```

Figure 7. Part of the input source code to be applied to the object-oriented Tool-Chain

Step 2. Executing the program loops through the AST, which is the result of parsing the target source code, making it a data object and completing the DataList.

Step 3. Using the DataList and the Script Maker to create script statements.



```
@startuml
/'do Class Define (클래스 정의)'/
class shop.Goods
{
    String name
    int money
    +String getName();
    +void setName(java.lang.String);
    +int getMoney();
    +void setMoney(int);
}

class shop.Shop
{
    Map<String,Goods> goodsSet
    Map<String,User> userSet
    +boolean regist(java.lang.String, java.lang.String);
    +User login(java.lang.String, java.lang.String);
    +boolean buy(shop.User, shop.Goods);
    +boolean charge(shop.User, int);
    +void addGoods(shop.Goods, java.lang.String);
    -boolean check(shop.User, shop.Goods);
}

class shop.User
{
    String id
    String pw
    int money
    +User User(java.lang.String, java.lang.String);
    +boolean buy(shop.Goods);
    +int getMoney();
    +boolean chckPwd(java.lang.String);
    +void setMoney(int);
}

    shop.Shop o-- shop.User : call shop.User
    shop.Shop o-- shop.Goods : call shop.Goods
    shop.User o-- shop.Goods : call shop.Goods
@enduml
```

```
@startuml
group void Login()
    TestScenario->Shop : login(java.lang.String, java.lang.String)
        Shop-->TestScenario
end

group void Buy_Product()
    TestScenario->Shop : buy(shop.User, shop.Goods)
        Shop-->TestScenario
end

group void Charge_Money()
    TestScenario->Shop : charge(shop.User, int)
        Shop->User : setMoney(int)
            User-->Shop
        Shop-->TestScenario
end

group void Add_Product()
    TestScenario->Shop : addGoods(shop.Goods, java.lang.String)
        Shop-->TestScenario
end

group void Registration()
    TestScenario->Shop : regist(java.lang.String, java.lang.String)
        Shop-->TestScenario
end

@enduml
```

Figure 8. Part of the generated script statement

In Figure 8, the script on the left shows source code to draw a class diagram, and the script on the right is for drawing a sequence diagram.

Step 4. Finally, the program recovers the design by creating a diagram through the script file.

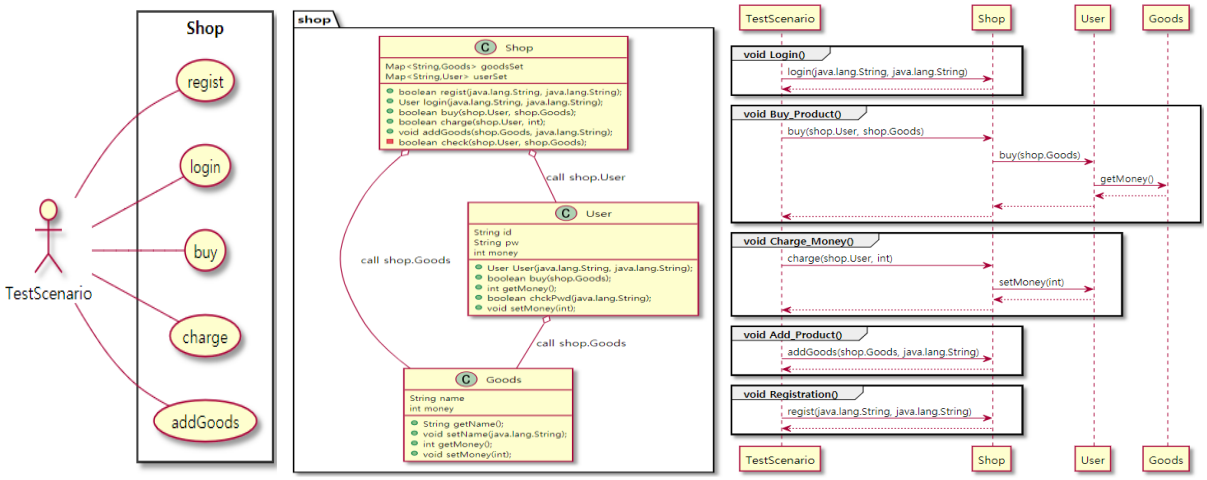


Figure 9. Restored Diagram

Comparing Figure 6 with Figure 9, it can be seen that the diagram has been restored similarly. Looking at the code in Figure 9, the TestScenario class was created by integrating Manage and User. The class diagram in Figure 9 is recoverd based on the fields and methods of the coresponding class. Then, we associate reference relationships between classes based on invocations. Therefore, you can see the difference in Figure 9 in that the has relationship is expressed as a Map.

Similar to the Use-Case diagram, the Sequence Diagram creates labels based on the methods in the Test Scenario and restores the object's method invocation relationship. The purchase scenario in Figure 6 is designed to adjust the amount using a price (-price). However, in the code implemented in practice, the user's ability to purchase controls the cost. Therefore, it satisfies the requirement of "purchase one product with the balance of the logged-in account", but it can be seen that the implementation is different from the design. Also, as in Use-Case, Manage and User are unified into TestScenario. Therefore, there is no Manage Actor of Use-Case, so it can be judged whether the requirements are reflected according to the viewpoint of performing TestScenario. In the implemented code, since there is no case to call or use the Shop class by distinguishing actors, it can be determined that more implementations for users and administrators of the requirements are needed.

5. Conclusion

In various areas of the 4th industry, a big issue is software quality enhancement for stability and reliability of the smart software systems. After revising software promotion law in 2020, we must clearly define requirements and separate design parts and implementation parts of an all-public software development contracts. This means who to be responsible for and how to sperate between requirement & high-level design and low-level design & implementation. We should validate the product whether a software development is followed the requirements of a project or not. Therefore, our research focus on automated UML design extraction with software visualization based on reverse engineering. The proposed method can quickly and easily recover the UML design from the source code through the Tool-Chain system. As a result, we can compare the original and manual design with the design of reverse engineering. Now we need to quantify completely and verify the complexity of the design and source code.

Through the proposed method, a more accurate design could be extracted by improving and extracting more information from the source code. In addition, the modularity of the Tool-Chain is increased by reducing unnecessary processes. We are going to research further on the software design and source code complexity matrix for increasing modularity.

Acknowledgement

This work was supported by the National Research Foundation NRF), Korea, under project BK21 FOUR, and also by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1I1A305040711).

References

[1] S. Moon, and R. Kim, "Code Structure Visualization with A Tool-Chain Method", *International Journal of Applied Engineering Research*, ISSN 0973-4562 Vol.10 No.99, 2015.

[2] C. Kim, J. Park, "A Software Maintenance Capability Maturity Model Based on Service", *Korea Institute of Information Technology*, pp.173-184, 2014.
DOI: http://dx.doi.org/10.14801/kiitr.2014.12.5.173

[3] J. Chikofsky, H. Cross, “Reverse engineering and design recovery: A taxonomy” *IEEE Software*, Vol.7, No.1, pp. 13-17, 1990.
DOI: https://doi.org/10.1109/52.43044

[4] J. Park, et al, “Building a Code Visualization Process to Extract Bad Smell Codes”, *KIPS Transactions on Software and Data Engineering*, Vol.8, No.12, 465~472, 2019.

[5] S. Jung, et al “Code Visualization with Object-Oriented Mapping Structure for Object Traceability”, *The Korea Smart Media Society Spring Conference 2021*, Vol 10 Issue 1, 11-14, 2021.
DOI: https://doi.org/10.3745/KTSDE.2019.8.12.465

[6] B. Park, et al, "Best Practices on Software Development and Management Process for the Republic of Korea Army Information System", *Korean Society of Information Sciences*, Vol.47 No.10, 911-925, 2020
DOI: https://doi.org/10.5626/JOK.2020.47.10.911

[7] W. Lee, et al. "The Constructing & Visualizing Practices in Effective Static Analyzer for analyzing the Quality of Object-Oriented Source Code", *The Korea Information Processing Society (KIPS) Fall Conference 2019*, Vol. 38, No.2, 704-707, 2019.

[8] B. Park, et al. "A Case Study on Improving SW Quality through Software Visualization ", *Journal of the Korean Society of Information Sciences*, Vol.41, No.11, 935-942, 2014.
DOI: https://doi.org/10.5626/JOK.2014.41.11.935