2016 International Conference on Platform Technology and Service (PlatCon)

Proceedings

15-17 February 2016
Jeju, Korea

IEEE Catalog Number: CFP16F03-ART (Xplore)
ISBN: 978-1-4673-8685-2 (Xplore)

IEEE Catalog Number: CFP16F03-CDR (CD)
ISBN: 978-1-4673-8684-5 (CD)

Code Complexity on Before & After applying Design Pattern through SW Visualization

So Young Moon
SELab., Dept. of Computer and Information Communication
Hongik University
Sejong Campus, 30016, Korea
msy@selab.hongik.ac.kr

Bo Kyung Park
SELab., Dept. of Computer and Information Communication
Hongik University
Sejong Campus, 30016, Korea
park@selab.hongik.ac.kr

R. Young Chul Kim
Dept. of Computer and Information Communication
Hongik University
Sejong Campus, 30016, Korea
bob@hongik.ac.kr

***Abstract*— Software is actually depended on his/her coding maturity of each developer. Software is inevitably used in all fields due to ICT convergence, which is increasing on the issue of code quality. His/her developer avoids to show source codes to other persons, and also repeats to modify them, which finally makes spaghetti codes. Therefore, it will be possible to increase bug occurrence, and also fall below legibility and understanding of the code. Its code quality is depended on the maturity of a developer. To protect this problem, we apply SW visualization with GOF design pattern to make good design structure. Design pattern is a verified solution to provide with object oriented design. Code complexity is essentially referred to software quality, which has low complexity to easily understand, and has low possible to occur errors. In this paper, the SW Visualization method is applied to compare the relationship between each code complexity Before & After applying design pattern to enhance the quality of codes.**

Keywords—Design Pattern; Code Complexity; SW Visualization; Static Analysis; SW Quality, Code Quality

I. INTRODUCTION

The demand of software quality is gradually increasing as software used in most industrial fields. One error can cause more errors due to the enlargement and complication of software. Therefore, code complexity must lower to secure legibility and maintenance. High level developers work on 28 times better than low level developers[1]. Also, when a developer moves out, it takes much time for new developer to analyze and understand the source code or project for a system maintenance or improvement. That is one reason why it takes long time due to code complexity, and has low legibility and understanding. To solve these issues, we use design pattern mechanism based on reverse engineering. We reduce code complexity through refactoring operation is performed to lower complexity by measuring the code complexity through static analysis. If a cyclomatic complexity number is between 1~10, the probability of other error occurs 5 percent. A cyclomatic complexity number of over 50 means to occur 40 percent of the probability of another error [2]. Therefore, this paper purposes to make lower code complexity to improve software quality. For a case study, it is performed to analyze complexity on source codes applied with & without design pattern, and to measure cyclomatic complexity and module coupling. SW visualization method based on reverse engineering is used for measuring with visualizing code complexity[3].

This paper is as follows. Chapter 2 mentions related works such as cyclomatic complexity, module coupling, and cohesion are explained. In Chapter 3, comparison between each code complexity before and after application of design pattern are visualized. In Chapter 4, the conclusions and future research are mentioned.

II. RELATED WORK

A. Design Pattern

The purpose of design pattern[3] is to solve problems in particular design and making flexible and reusable object oriented software. We use design patterns as a verified technology for reusing design, and improve system maintenance or documentation. Design pattern is classified into creational pattern, structural pattern, and behavioral pattern. There are five patterns in the creational patterns, seven patterns in the structural patterns, and eleven patterns in behavioral pattern. Table. 1 shows the design pattern field classified by the purpose and range of design pattern. In this paper, as the command of behavioral pattern is applied in source code, we compare each code complexity between before and after applying design pattern on refactoring.

- Creational Patterns: These patterns Deal with the best way to create instances of objects. They create objects at run-time not at compile-time. This makes program more flexibility to be created for a given use case.
- Structural Patterns: These patterns describe how classes and objects can be combined to form larger structures.
- Behavioral Patterns: These design patterns are specifically concerned with communication between objects. They are concerned with Allocates responsibility of a process to an object and defines which algorithm is proper for an object.

Table. 1. Design Pattern Area

Goal / Scope	Creational Patterns	Structural Patterns	Behavioral Patterns

978-1-4673-8685-2/16/$31.00 ©2016 IEEE

Class	1. Factory Method	6. Adapter	13. Interpreter 14. Template Method
Object	2. Abstract Factory 3. Builder 4. Prototype 5. Singleton	7. Bridge 8. Composite 9. Decorator 10. Facade 11. Flyweight 12. Proxy	15. Chain of Responsibility 16. Command 17. Iterator 18. Mediator 19. Memento 20. Observer 21. State 22. Strategy 23. Visitor

B. Code Complexity

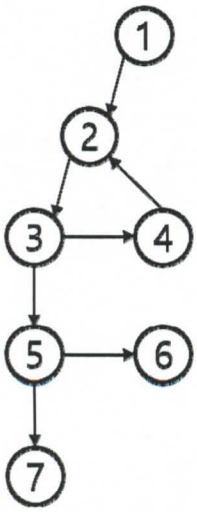
Cyclomatic Complexity is the most commonly used method to measure code complexity, and software metric is the coupling and cohesion.

- Cyclomatic Complexity[4]: measuring code complexity within functions, modules, methods or classes of software which was proposed by McCabe in 1976. The number of control paths within the software are used for calculation.

$$v(G) = e - n + p \quad (1)$$

v(G): cyclomatic number
e = the number of edges of the graph
n = the number of nodes of the graph
p = the number of connected components

Fig. 1 shows the results of calculating cyclomatic complexity of the graph with (1). The number of edges is 7, the number of nodes is 7, the number of connected components is 1 in which $v(G) = 7 - 7 + 1 = 1$.



	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	1	1	0	0
4	0	1	0	0	0	0	0
5	0	0	0	0	0	1	1
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

Fig. 1. Program path

- Coupling and Cohesion[5]: It is a concept that has high legibility and maintenance which was proposed by Larry Constantine, the developer of structured design and is an outstanding design when low coupling and high cohesion is accompanied. As shown in Fig. 2, coupling is composed of data, stamp, control, external, common, and content and coupling closer to data coupling shows higher design quality. Cohesion is composed of coincidental, logical, temporal, procedural, communication, sequential, and functional cohesion and cohesion closer to functional cohesion shows high design quality. Coupling between modules must be weak and cohesion must be strong to create a well-designed independent module.

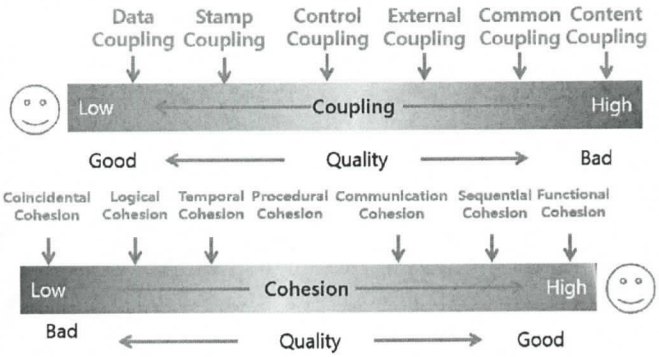


Fig. 2. Quality Indicators of Coupling and Cohesion

III. Relation with Design Pattern and Code Complexity

In this chapter, the code complexities before and after applying design pattern are compared. The example program is a simple document editor implemented by Java. Through reverse engineering based SW visualization method by using the module coupling and code complexity method by MaCABE, 1) analysis on complexity of the source code without applying design pattern and 2) analysis on complexity of the source code applying design pattern are conducted. Fig. 3 shows the results extracted by using the SW visualization method on 1) and 2).

A. Code Complexity of before applying Design Pattern

Fig. 3 shows module coupling extracted by the SW visualization method on the Java based document editor without design pattern application. The module was defined as class and the left result in Fig. 3 shows the number of reference of linked relation between JavaEditor module, FileProcessor module, and Editor UI module. JavaEditor(24) means that 24 numbers of reference were internally used and the (16) in the arrow indication from JavaEditor to EditorUI means that the Editor UI module was referred to the JavaEditor module 16 times. The right result in Fig. 3 means that the stamp coupling from the JavaEditor module to the FileProcess module is 6. Also, the stamp coupling from the FileProcess module to the FileProcess module is 2. 3 in the result 3*2=6 is the number of stamp coupling between the modules and 2 is the weight of stamp coupling.

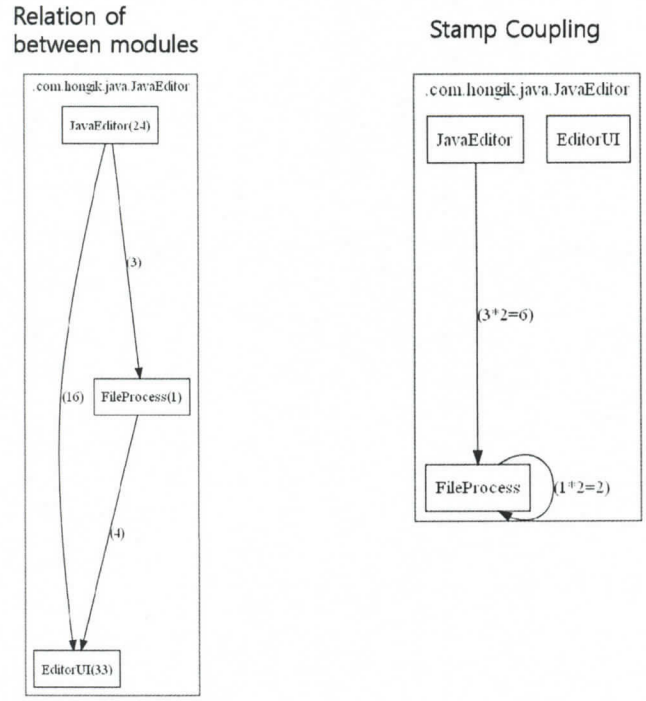


Fig. 3. Coupling with SW Visualization

Table. 2 shows code complexity extracted by the SW visualization method on the Java based document editor without pattern application.

Table. 2. Code Complexity with Cyclomatic Complexity

File Path	Function Name	Complexity
EditorUI.java	EditorUI.EditorUI	1
FileProcess.java	FileProcess.fileOpen	4
FileProcess.java	FileProcess.fileSaveAs	3
FileProcess.java	FileProcess.fileSave	3
JavaEditor.java	JavaEditor.JavaEditor	1
JavaEditor.java	JavaEditor.actionPerformed	6
JavaEditor.java	JavaEditor.main	1

Code complexity is highest in the actionPerformed method of the JavaEditor module with a value of 6. This value is considered as a low complexity when comparing general code complexity values, but the core of this paper is that application of design pattern can reduce code complexity.

B. Code Complexity of after applying Design Pattern

Table. 3 shows the code complexity deducted by the SW visualization method on the Java based document editor applying the pattern.

Table. 3. Code Complexity with Cyclomatic Complexity

File Path	Function Name	Complexity
AbstractCommand.java	execute	1
ExitCommand.java	ExitCommand.ExitCommand	1
ExitCommand.java	ExitCommand.execute	1
ExitCommandActionListener.java	ExitCommandActionListener.actionPerformed	1
NewCommand.java	NewCommand.NewCommand	1
NewCommand.java	NewCommand.execute	1
NewCommandActionListener.java	NewCommandActionListener.actionPerformed	1
OpenFileCommand.java	OpenFileCommand.OpenFileCommand	1
OpenFileCommand.java	OpenFileCommand.execute	1
OpenFileCommandActionListener.java	OpenFileCommandActionListener.actionPerformed	1
SaveAsCommand.java	SaveAsCommand.SaveAsCommand	1
SaveAsCommand.java	SaveAsCommand.execute	1
SaveAsCommandActionListener.java	SaveAsCommandActionListener.actionPerformed	1
SaveCommand.java	SaveCommand.SaveCommand	1
SaveCommand.java	SaveCommand.execute	1
SaveCommandActionListener.java	SaveCommandActionListener.actionPerformed	1
EditorUI.java	EditorUI.EditorUI	1
FileProcess.java	FileProcess.fileOpen	4
FileProcess.java	FileProcess.fileSaveAs	3
FileProcess.java	FileProcess.fileSave	3
JavaEditor2.java	JavaEditor2.JavaEditor2	1
JavaEditor2.java	JavaEditor2.main	1

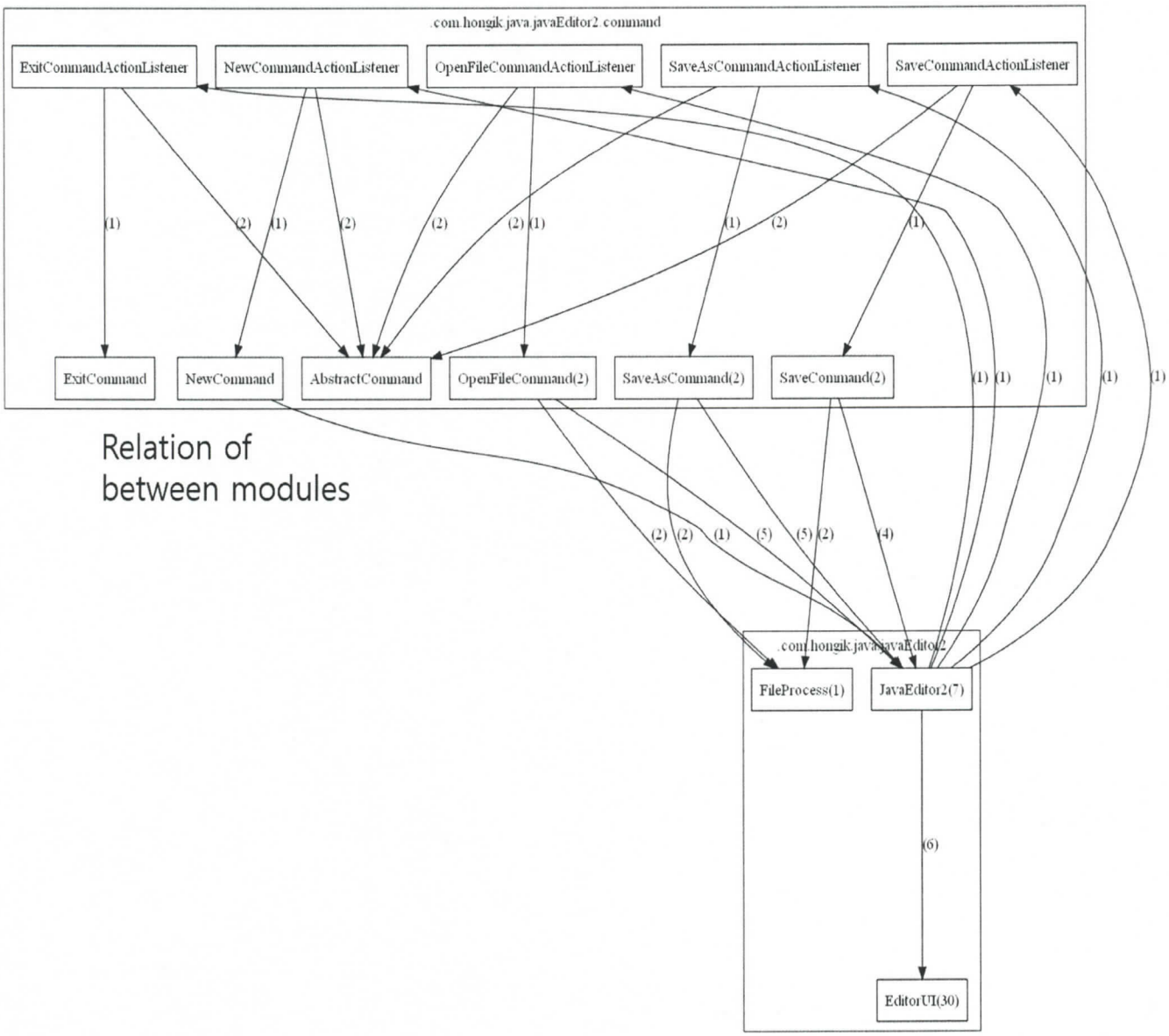


Fig. 4. Relation of between modules

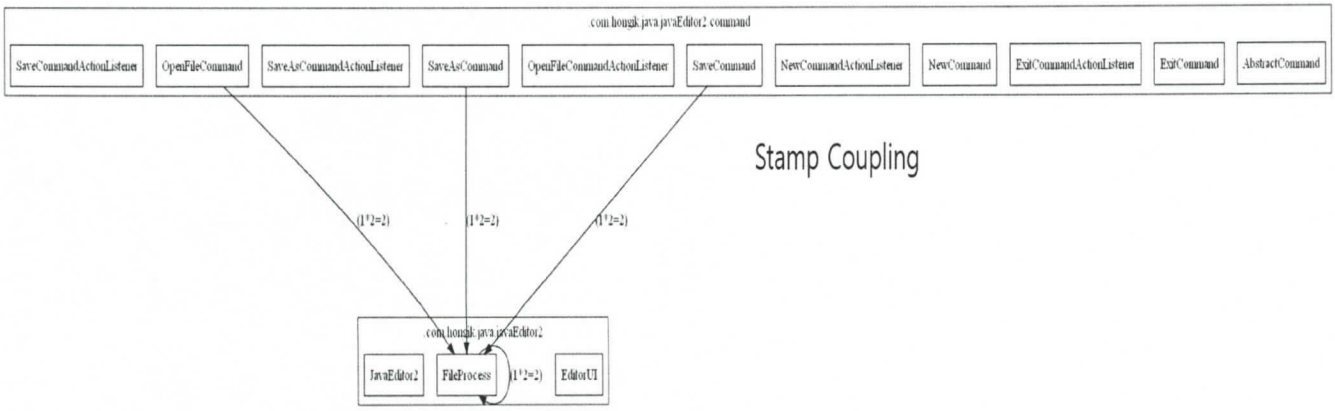


Fig. 5. Coupling of after applying Design Pattern

Unlike the results in Table. 2, there are more modules in the results of Table. 3. As a result, only values of 4 or lower are shown in which this is because command pattern is applied to encapsulate commands demanded by the user on the object for processing. Basically, code complexity influences the level of complexity depending on control. Therefore, it could be found that code complexity decreased by applying the command pattern.

Fig. 4, 5 shows the module coupling extracted by the SW visualization method on the Java based document editor applying design pattern. Analyzing the results in figure, the results in Fig. 3 seem to be more complex than Fig. 4, 5, but the values show that the code complexity of the programs developed by applying design pattern are independent modules with low coupling and low code complexity. In Fig. 4, reference of EditorUI module in the JavaEditor2 is 6 which is lower than the value in Fig. 3 which is 10. It is because the command pattern is applied to segment the module by functions for management. Also, it can be known that the stamp coupling in Fig. 5 is lower than the stamp coupling in Fig. 3.

IV. Conclusion

There exists many differences depending on the effort and competence of developers to improve software quality and code quality. Reusing well-made design saves costs and time to relieve fatigue of developers. Codes are the latest results in software and software quality is proportionate to the quality of codes. To enhance code quality, same codes should not be repeated, one module must perform only one function, improper comments should be avoided, and coding standard should be applied. Also, applying design pattern is a method of reusing well designed modules. Coding standard takes place between developers when design pattern is used, communication is aided, and maintenance is improve to also enhance code quality. Also, developers can reviewer their own code anytime through SW visualization to find problems, modify, and reduce code complexity to lower error occurrence rate.

In this paper, as result of comparing programs applying design pattern and not applying design pattern through SW visualization, it was found that the software developed by applying design pattern showed low code complexity.

Acknowledgment

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

References

[1] Robert L. Glass, Facts and Fallacies of Software Engineering, Addison-Wesley, 2002

[2] http://www.aivosto.com/project/help/pm-complexity.html

[3] So Young Moon, Sang Eun Lee, R. Youngchul Kim, "Inner Visualization for Analyzing Code Complexity", The 5th International Conference on Convergence Technology, vol.5, no.1, pp.346-347, 2015.

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns : Elements of Reusable Object-Oriented Software", 1994.

[5] THOMAS J. McCABE, "A Complexity Measure", IEEE Transactions on Software Engineering, vol. SE-2, no.4, December 1976.

[6] W. P. Stevens, G. J. Myers, L. L. Constantine, "Structured design", IBM Systems Journal, vol 13. No2, pp.115-139, 1974.