

 Printed in Japan

ISSN 1343-4500 (print)  
ISSN 1344-8994 (electronic)

# **i**NFORMATION

*An International Interdisciplinary Journal*



**Volume 20 Number 2(A), February 2017**

Published by International Information Institute  
[www.information-iii.org](http://www.information-iii.org)



Extending Use Case Point (e-UCP) mechanism for Cost Estimation and Priority for the Renewable Energy Monitoring System	<i>Bo Kyung Park, Woo Sung Jang and R. Young Chul Kim</i>	935
Validating Requirement Satisfaction through Software Tracking Matrix Model	<i>Bo Kyung Park and R. Young Chul Kim</i>	945
Evaluation of a Smart Traffic Light System with an IOT-based Connective Mechanism	<i>Hyeon Jun Lee, R. Young Chul Kim and Hyun Seung Son</i>	953
xCodeParser based on Abstract Syntax Tree Metamodel (ASTM) for SW Visualization	<i>Hyun Seung Son and R. Young Chul Kim</i>	963
Characteristics of Dehumidifier using Psychoacoustics Parameters in Sound Signal Processing	<i>Seong-Geon Bae and Myung-Jin Bae</i>	969
A Study on Definition of Unknown Explosion Sound using a Signal Processing	<i>Seong-Geon Bae, Myung-Jin Bae and Geum-Ran Baek</i>	977

### **Medicine and Life Sciences**

Implementation of a Natural Light Chromaticity Coordinates-based Healthy Lighting System	<i>Yang-Soo Kim, Sook-Youn Kwon and Jae-Hyun Lim</i>	985
Vagus Nerve Stimulation System for Treating Tinnitus based the Stimulation Intensity Control According to the Tinnitus Frequency Amplitude	<i>Jaeung Lee and Hojun Yeom</i>	993
Myoelectric Controlled Electrical Stimulation in Sleep Bruxism Treatment with Adaptive Artifact Canceller	<i>Hojun Yeom</i>	999
Secure and Anonymous Health Data Transmission Protocol for Remote Healthcare Monitoring	<i>Youngho Park, Chul Sur and Kyung-Hyune Rhee</i>	1005
A Study on the Real-time Toxic Chemical Management System based IoT	<i>Min Soo Kang, Young Gyu Jung, Ji Young Mun and Chun Hwa Ihm</i>	1015
Visualization Device of Living Organism through Soft x-ray	<i>Ji Young Mun, Kyung Eun Lee, Won Ja Lee, Hwa Shik Youn, Min Soo Kang and Sung Sik Han</i>	1023
Evaluation of Dynamic-motion in Body Index Techniques: Body Mass Index and Physical Sensory Index	<i>Jeong-lae Kim and Kyu-Ok Shin</i>	1031

## Validating Requirement Satisfaction through Software Tracking Matrix Model

Bo Kyung Park\*, R. Young Chul Kim\*\*

*SELab, Dept. of Computer and Information Communication, Hongik University, Sejong, 30016, South Korea*  
*E-mail: {park\*, bob\*\*}@selab.hongik.ac.kr*

### Abstract

In this current time, software is rapidly developed to release at time-to-market, which is so critical to control the quality of the software. In our industrial fields, software developers still work on software maintenance without any design, documentation, and code visualization [1]. Thus, the need of software visualization based on reverse engineering has emerged. In previous papers, we mentioned a procedural method for software visualization and quality improvement [2, 3, and 4]. But this method is difficult to validate requirement satisfaction through tracing the customer requirements. We propose a requirement tracking model for requirement's verification, which can visualize the actual object oriented code based on a SW visualization, and improve software quality through manually refactoring based on inner structure of the visualized code. The improved result can be verified through Software Requirement Tracking Model. We can also check how much to achieve between the quality improvements and the customer requirement satisfaction.

**Key Words:** Software Visualization, Requirement Validation, Tracking Matrix Model, Software Quality

### 1. Introduction

Recently the importance of the quality management of software is increased on software development. It makes software maintenance difficult due to software complexity, software invisibility, and developer's environments. Nowadays, software industry has focused on the development process for software quality and time to market, which still focuses on code-centric development for how rapidly to develop the software. Therefore, we also consider the two more aspects 1) to improve bad habits of programmers, and 2) to work maintenance on code visualization without any design and documentation. In order to improve software quality management and maintenance, software engineers focus on reverse engineering, which analyzes the existing system without any design and documentation, and the original programmers. To easily analyze the inner structure of software, we try to use this approach. In previous studies, we proposed the visualized method of the internal object-oriented code structure, and software quality improvement process based on coupling of the modules [2, 3]. In addition, we measure modules based on coupling, define quality indicators, and improve the quality of software through refactoring. But this is difficult to verify software whether it is



correctly developed system based on requirements during software development or not. 54% of software projects failure factors has occurred from inadequate requirements management. Therefore, a systematic requirement verification methods are needed [5, 6]. In this paper, we propose a verification method through the Software Requirement Tracking Matrix Model. These results define the measure module and quality indicators. And it will improve the quality of software, which can verify the requirement through the requirements traceability model. Through these processes, we can check how much to achieve between the quality improvements and the initial code requirements.

This paper mentions as follows: Chapter 2 introduces to improve quality through the software visualization process. Chapter 3 describes the requirements for verification methods through tracking matrix. Chapter 4 describes the application, and chapter 5 mentions conclusions

## 2. Software Visualization Process

In previous studies, we have proposed a quality improvement through visualization software [2, 3]. The visualized tool-chain has used with Open Source based tools such as Parser (or Analyzer), Database, View Composer. In addition, it consists of source code analysis, DB storage, and structural analysis, which performs each other as independent functions [1, 2, and 3]. With source code analysis, the Tool-Chain derives the data through the Parser, and finds the relationships between the object-oriented components (class, method, variable). With DB storage, it stores the analyzed information to the database (SQLite), and also defines table by class, method, or variable. And it arranges to generalize relations and associations. This classification easily looks up information in the structural analysis [3]. In the structural analysis, the Tool-Chain measures the coupling with arranged information. In the visualization, it can be visible the internal software structure through the View Composer. DOT Program is used as a visualization tools. Quality improvement method measures the software quality based on coupling [1, 2]. This software is based on the design principle of “*strong cohesion and loose coupling*” principle. The previous research suggests the way to measure quantitative measurement method [2, 3] for high software quality. In the module definition, a module unit is defined for the target software code. We also define a quality indicator for the quantitative measurement of all coupling elements such as data, stamp, control, external, common, and internal coupling. In order to develop a high-quality software, the coupling between the modules are needed to be minimized. Therefore, we define the quality indicator’s weighted values as follows: Data Coupling (1), Stamp Coupling (2), Control Coupling (3), External Coupling (4), Common Coupling (5), and Internal Coupling



(6). After them, it automatically calculates the sum results of multiplication between *the coupling index number and weight value of coupling modules* in the whole code. Finally, analyzed the code pattern to detect the defined code patterns.

### 3. Validating Requirement through Requirement Tracking Matrix Model

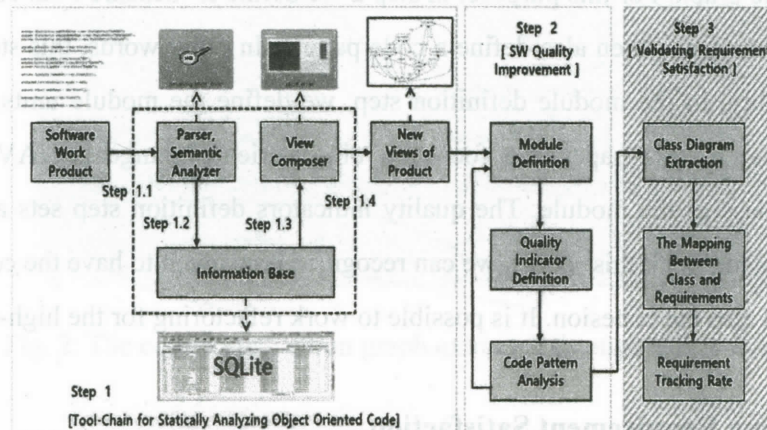


Fig. 1. A tracking process for verifying requirement satisfactions

Figure 1 shows a tracking process for verifying requirement satisfactions. This process consists of three steps as follows; ① Tool-Chain through static analysis of the object-oriented code in Step1, ② SW quality improvement in Step2, and ③ Satisfaction verification of requirements in Step 3. In this paper, we limit to mention the requirements satisfaction verification at step 3 in figure 1.

#### 3.1 Our Tool-Chain for Statically Analyzing Object Oriented Code

In figure 1, step 3 is the process of extracting the UML class diagram from the code [1, 2, 3, 7, 8, 9, and 10]. This step is to extract the class diagram in order to visualize the requirements tracking model. In source code analysis step, the parser analyzes the object-oriented code, which extracts the elements of classes and the relationship information (generalization, association and dependent) among the classes. The database storing step classifies the extracted information, and stores them in each table. Association between the extracted elements is stored by one to one mapping. For example, if Class 'A' has methods 'a', 'b', this mapping information should be stored in two ways such as A->a, A->b. The structural analysis step compares and analyzes the dependencies between the classes and then measures coupling between them. The DB storage step extracts the elements classified from the modules. In this paper, we define a modular unit as a class. Therefore, the extracted information is the information of class and classes, methods, and variables. The visualization



step visualizes relationship between Classes.

### 3.2 Software Quality Improvement

Step 2 measures the quality of the software based on coupling [1, 2, and 3]. The class diagram extracted from step 3 defines the mapping relationship between the requirements and also shows the graph. For this purpose, in step 2 we define to measure both coupling modules and quality index, and then also define a code pattern. In other words, this step improves the software quality. In the module definition step, we define the module units from the target software code. In this paper, we focus on object-oriented language JAVA code, which mentions a class as the module. The quality indicators definition step sets a score per each coupling element. With this scores, we can recognize how much to have the coupling between modules, and also the cohesion. It is possible to work refactoring for the high-quality software.

### 3.3 Validating Requirement Satisfaction

Step 3 validates the initial requirement with the extracted results in step 1 & 2. It compares and analyzes both java source code and list of requirements (Step 1) defined by *Redmine* of the *tool-chain*. In this step, it also assigns the numbering on requirements in the coupling graph. We extract the class diagrams from coupling graph on source code, and also the requirement tracking rate to verify that the correct mapping requirements. The requirement tracking rate is calculated by dividing the number of detected requirement in the total number of the requirements as below:

$$\text{Requirements Tracking Rate} = \frac{\text{The Number of Detected Requirements}}{\text{The Number of Total Requirements}} \quad [5]$$

## 4. Case Study

The case study is applied with the source code of our *Use Case Diagram Drawing Tool*. It can extract a class diagram from the source code (Java) of the use-case diagram tool in step 1. At step 2, it measures software quality metrics based on all coupling elements. To improve the quality of the software, the refactoring repetitively is performed until getting the suitable measured value. Then, we continually extract the bad smell pattern during/after refactoring [11]. In this case, we find bad smell patterns such as the Message Chain and Duplicated Code Pattern. Therefore, as we can remove these pattern, it may possible to improve the strong coupling mechanism to the loose one. Figure 2 shows a part of the code visualization graph extracted through this *tool-chain* after statically analyzing object-oriented code. It also extracts the coupling elements between the measured classes in this visualization graph



through a View Composer based on this source code. The graph shows the rectangle as the identifier of the class, the arrow as indicator between the referencing class and the referenced class.

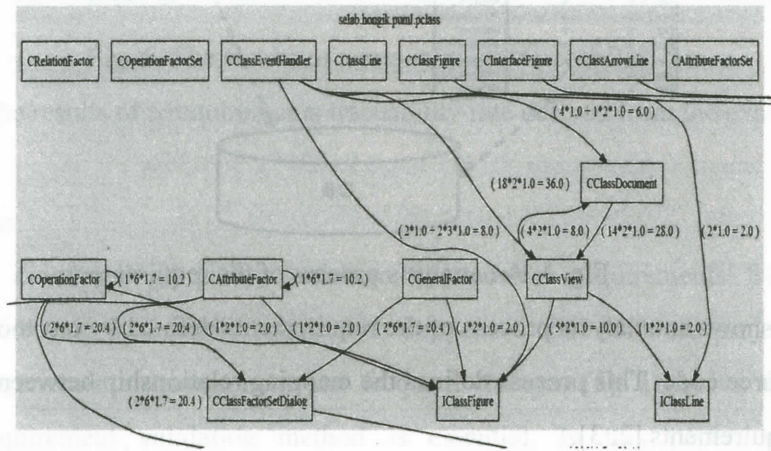


Fig. 2. The code visualization graph extracted through this tool-chain

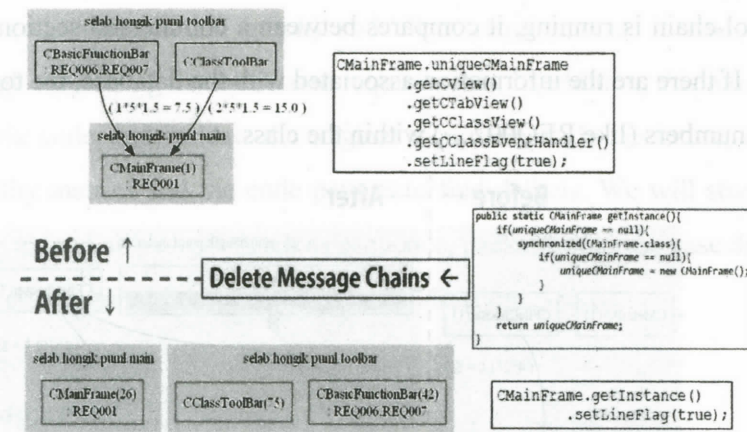


Fig. 3. An example to remove the Message Chain of bad smell pattern

Figure 3 shows an example to remove the Message Chain of bad smell pattern. At step 3, it illustrates the requirements tracking rate the mapping relationship between the class diagram and client requirements.



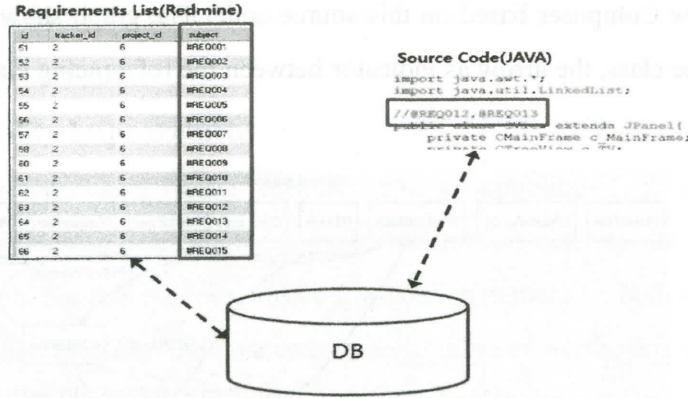


Fig. 4. An analysis process of the requirements

Figure 4 shows an analysis process of the requirements defined by our tool-chain (*Redmine*) and the source code. This process defines the mapping relationship between the class diagram and the requirements [2, 3].

Our mapping relationship method analyzes the java code related with the requirements, and assigns the comment (*//@Requirement Number*) at the start point of the associated code [5]. When the tool-chain is running, it compares between a commented section and the Database information. If there are the information associated with the database, the tool-chain enters the requirement numbers (like *REQ002...*) within the class.

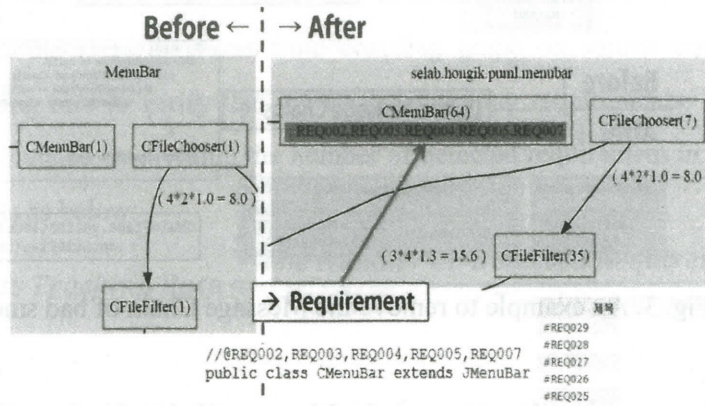


Fig. 5. An extracted mapping relationship

Figure 5 illustrates an extracted mapping relationship. ‘Before’ part is derived from the extracted results of step 1, 2 in figure 1. ‘After’ part is the extracted result from defining a mapping relationship between the classes and requirements. Each requirement number is marked in each associated class (*CMenuBar ↔ REQ002/003/004/005/007*). The extracted information is used to calculate the requirement tracking rate [5]. The total number of requirements of the *Use Case Diagram Drawing Tool* is 29. The number of mapped requirements is 13. Therefore, the requirements traceability rate of the applicable case is 45%. The results shows that only 45%. The results shows that only 45% of the total requirements is reflected in this application tool.




No.	Project	Timestamp	Total Coupling (SN)	Total Violation (PMD)	ReqRate (req/totalReq)	Total LOC	Result Graph	Comment
21	pUML	201501222307	995 8	 Violation View	45.0% (13 0/29 0)	13658	Class Diagram	add reqRate field

Fig. 6. The results of a requirements traceability rate

Figure 6. is the results of a requirements traceability rate derived from the examples.

### 5. Conclusion

In order to develop high-quality software, the correct requirements from the initial development lifecycle is crucially required. It happens to fail software projects due to the insufficient requirements management and frequent changes [6]. To solve this problem, a systematic requirement validation method is essential. In this paper, we propose a requirements traceability model to validate the requirements satisfaction. This model not only maps relationship between the existing results (such as the extracted design from the software code and the code visualization graph) and client requirements. It also shows the class diagram through visualizing the associated requirements. In addition, this model may improve the quality of the code, and validate the original requirements. In the future, we will study the additional quality metrics and the code pattern to find defects. We will study the extraction method how to extract all designs (such as sequence, package, and use case diagram) from the code based reverse engineering through code visualization.

### 6. Acknowledgments

This research is 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) in 2016/2017 and Research and Development Service through the Telecommunications Technology Association (TTA).

### References

[1] Nipa SW Engineering Center, SW Development Quality Management Manual (SW Visualization), 2013.

[2] Bokyung Park, Haeun Kwon, Hyun Seung Son, Young Soo Kim, Sang Eun Lee, R. Young Chul Kim., A Case Study on Improving SW Quality through Software Visualization. Journal of KIISE, 41 (2014), 935-942.

[3] Bokyung Park, Haeun Kwon, Hyeoseok Yang, Soyoun Moon, Young Soo Kim, R. Young Chul Kim., A Study on Tool-Chain for statically analyzing Object Oriented Code.



Korea Computer Congress, 463-465.

- [4] Geon-Hee Kang, R. Young Chul Kim, Geun Snag Yi, Young Soo Kim, Young B. Park, Hyun Seung Son., A Practical Study on Code Static Analysis through Open Source based Tool Chains. *KIISE Transactions on Computing Practices*, 21 (2015), 148-153.
- [5] Bokyung Park, Haeun Kwon, Young Soo Kim, R. Young Chul Kim, Requirement Tracking Visualization for Validating Requirement Satisfaction. *The 5th International Conference on Convergence Technology*, 5 (2015), 368-369.
- [6] Karl Wieggers., "Software Requirements", *Microsoft*, 2013.
- [7] Haeun Kwon, Bokyung Park, R. Young Chul Kim, Sang Eun Lee., Extracting Designs via Code on Reverse Engineering. *The 5th International Conference on Convergence Technology*, 5 (2015), 274-275.
- [8] Haeun Kwon, Bokyung Park, Hyeoseok Yang, Young B. Park, Young Soo Kim, R. Young Chul Kim., Applying Reverse Engineering through extracting Models from Code Visualization. *The 2014 Fall Conference of the KIPS*, 21 (2014), 650-653.
- [9] JunSun Hwang, R. Young Chul Kim, SangEun Lee., A Guideline for Realization on extracting automatic size maturity level of diverse component via Source Codes. *The 5th International Conference on Convergence Technology*, 5 (2015), 268-269.
- [10] So Young Moon, Sang Eun Lee, R. Young Chul Kim., Internal Code Visualization for Analyzing Code Complexity. *The 5th International Conference on Convergence Technology*, 5 (2015), 364-365.
- [11] Martin Fowler, *Refactoring: improving the design of existing code. Addison-Wesley*, 1999.

**\*\*Corresponding author: R. Young Chul Kim, Ph.D.**

Department of Computer & Information Communication (CIC),  
Sejong Campus, Hongik University  
2639 Jochiwon-eup, Sejong-ro, Sejong City, 30016, South Korea  
E-mail: bob@hongik.ac.kr