International Journal of Applied Engineering Research, ISSN 0973-4562 Vol. 10 No.90 (2015)

Code Structure Visualization with A Tool-Chain Method

So Young Moon[1], R. Youngchul Kim[2]
[1]Dept. of Computer and Information Communication, Hongik University, Sejong, Korea, 339-701
msy@selab.hongik.ac.kr
[2*]Dept. of [1]Dept. of Computer and Information Communication, Hongik University, Sejong, Korea, 339-701
bob@hongik.ac.kr

Abstract - It is an important issue to deliver high-quality software products with huge-scale codes and shorter time-to-market. While developers still focus on code-based development across the board, the domestic software industry is witnessing a shift to development/testing process and maturity measurement as a means of implementing high-quality software. Yet, the challenging issues of software quality remain rarely addressed, e. g., invisibility, increasing complexity and unfavorable development environment in small businesses, which impedes proper software quality management. More importantly, existing legacy systems fail to preserve their original design, while their code complexity increases due to more patching of the original codes. To address these problems, we adopted a code visualization technique that substantially reduces the code complexity between modules. To do this, we suggest a tool chaining method based on the existing open source software tools, which extend NIPA's Software Visualization techniques. More specifically, the proposed technique serves two purposes: 1) rectifying overall bad development habits, and 2) maintaining software codes with visualization with no need for design-related specification and documentation. This paper presents how to visualize the inner structure of codes and how to improve the quality of software codes using the proposed Tool-Chain. In addition, refactoring is used to fix bad coupling of the quality measurement indicators in code visualization. As a result, it is possible to quantitatively analyze source codes and rectify developers' bad smells. Ultimately, the proposed method is conducive to delivering high-quality software products.

Keywords - SW Visualization; Tool-Chain; Reverse Engineering

I. INTRODUCTION

Software has been widely used across diverse fields, serving as a key to add values to final products and ensure their competitive edge. However, its invisibility and complexity as well as domestic SMEs (Small and Medium Enterprise) software development environment have thwarted software quality management [1]. High-quality software development requires 1) certifying the quality of software products. In Korea, SP (Software Process) and GS(Good Software) certificates are recommended. Furthermore, TMMi and CMMi are used to improve and assure the quality of software. 2) The latest software development methodologies or processes need to be employed to develop high-quality software. 3) Once developed, software undergoes a series of test processes for debugging and ultimately for quality enhancement. These approaches, however, impose cost and other burdens on SMEs. In this context, the present paper intends to contribute to high-quality software development by focusing on visualization of core domains of software, viz. visibility of development process, reduction of complexity, and the absence of documentation about development and design. In addition, source underpinning software need be updated in time to reflect up-to-date information for the operability of software, at the same time the quality must be kept at its highest, as software can be explained by its source codes only [2]. Thus, the present paper applies the Software Visualization Technique developed by the NIPA(National IT Industry Promotion Agency) with a view to: 1) detect, alter and modify the problems of legacy codes; 2) provide guidelines for rectifying software developers' bad smells by applying a reverse engineering technique via code visualization; and 3) cope with the absence of developers or relevant documentation to help maintain legacy systems. To visualize the internal structure of codes, this paper constructs a tool chain by connecting a range of open sources, i.e., Source Navigator, DOT Script and SQLite. Then, using the tool-chain method, JAVA-based software is applied to the visualization of software and thus derives a visualized output that gives some insight into original code structures. Also, the coupling measurement module and quality indicators are defined to perform refactoring (with the output) in order to develop high-quality software.

This paper presents the following chapters. Chapter 2 elucidates software visualization and reverse engineering in light of related studies. Chapter 3 discusses the method of configuring the tool chain. Chapter 4 illustrates cases of application. Chapter 5 mentions the conclusion and future studies.

II. RELATED WORK

2.1 SW Visualization

High-quality software development needs to go through development, test automation and quality certification. Above methods, however, are challenging to venture start-ups, SMEs and even established companies in the IT industry due to personnel cost and other expenses. NIPA's software

visualization may be fit for high-quality software development at IT venture startups, SMEs and even established entities that are typically constrained by a lack of personnel and financial resources [3]. Software visualization is a technique intended for the betterment of software quality control and maintenance by visualizing and documenting source codes and development processes.

First, visualization addresses the most challenging aspect of software development, i.e., invisibility, putting an entire software development process into perspective for the benefit of controlling the quality of both the software and its development process. Second, the documentation serves to manage corporate know-how of software development, to enhance intra-organizational understanding of tasks and to support communication with third parties in certain situations. SW visualization aims to manage sources and development processes, specifically involving visualization and documentation as a means of managing the quality of SW development. An entire process of software development needs be efficiently managed to produce valuable software. It takes clear-cut goal setting, efficient fulfillment, on-going monitoring and control activities to successfully manage software development. SW visualization enables 1) clarification of goals in line with guidelines, 2) system-based efficient development activities, and 3) continuous monitoring and controlling via visualization so as to lay a foundation for successful management of software development. SW developers draw on SW visualization to overcome the invisibility of software and ensure the transparency of an entire process of SW development, this contributes to securing SW quality, detecting development-related issues early on, reducing development cost and ultimately attaining corporate competitiveness [1].
Third, by providing options for documentation of diverse outputs piled up inside the system in the course of development the SW visualization minimizes related workloads while at the same time maximizing the usability.

2.2 Reverse Engineering

Forward engineering starts with outputs from high-level abstraction including requirements specification and then progressively goes through meticulous analysis and design to implement software products. On the contrary, reverse engineering is the process of analyzing software with the objective of recovering its design and specification. Reverse engineering can extract the design information from a source code [4]. The software source code is usually available as the input to the reverse engineering process. The objective of reverse engineering is to derive the design or specification of a system from its source code. Reverse engineering, which is used to develop a better understanding of a system is often part of or the re-engineering process. Reverse engineering is used during the software re-engineering process to recover the program design which engineers use to help them understand a program before reorganizing its structure [5]. Reverse engineering is used for system analysis with the intent to identify software components, their interactions, and to restore different forms or higher levels of abstract representations.

The first and foremost reason that reverse engineering is used for visualization is to understand certain software programs without the help of original developers. The second reason is to inspect legacy systems without any in-depth analysis. Therefore, reverse engineering may be viewed as a system analysis technique for identifying software components and their inter-relations or for restoring different forms or higher levels of abstract representations [1].

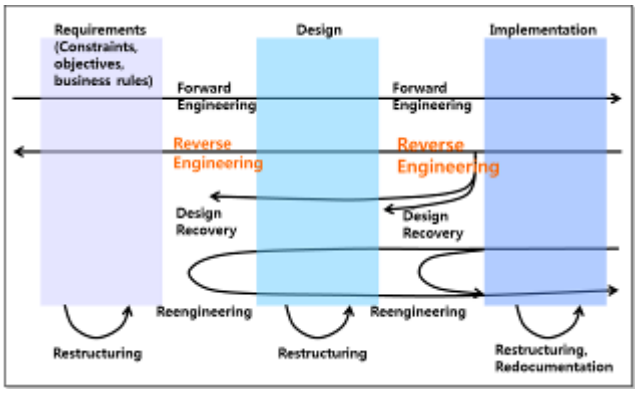

Figure 1. Reverse Engineering Process

Figure 1 schematizes the reverse engineering process, where tracing goes from the implementation phase to the design phase. This makes it possible to restore the design and requirements in the implementation step. Using this technique, the present paper carries out the visualization to analyze and understand software without the help of developers, and thus to inspect legacy systems. Differently put, reverse engineering is used for software visualization as it addresses traceability and complexity issues, restores lost information, detects adverse effects and consequently enables reuse of software. This paper is consist of a tool for the visualization and reverse-engineering by using the open-source tools. We will talk more about the consistent in chapter 3.

III. Tool-Chain Method

Existing tools for reverse engineering/static code analysis have difficulties in visualizing architectures of certain companies. The present paper employs a static code analysis method composed of tool-chain process and software quality improvement. Figure 2 shows the tool-chain process built on existing open sources.

The proposed Tool-Chain process is based on open sources such as Source Navigator[6], SQLite[7], Graphviz[8].

3.1 Description of each step of tool-chain

- Step 1: Source Code Analysis

  This step analyzes source codes from SN (Source Navigator). Once analyzed, the source code is extracted in compliance with the parser's format. Extracted files (SNDB Files) contain overall information about the program code (e.g., class, method, local variables, global variables and parameters).

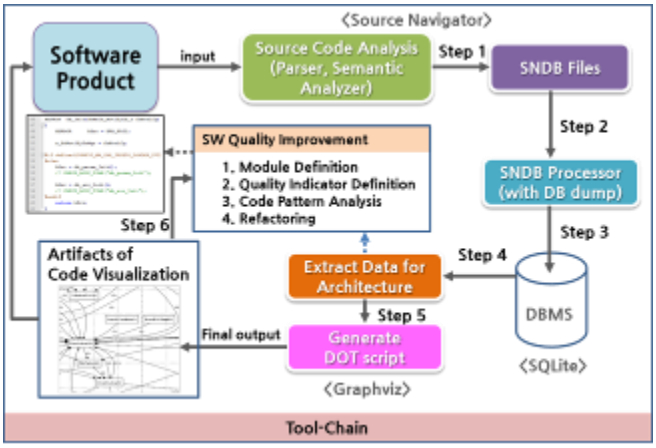

Figure 2. Tool-Chain Method Process

- Step 2: SNDB Files Analysis

  SNDB files contain the data extracted from SN in binary formats. To analyze the binary file content, dbdump.exe provided by SN is run internally and transformed to text formats.

- Step 3: Create DB from SNDB Processor

  The data analyzed in Step 2 is stored in a table generated in the DB. This step saves the information of all files extracted in the DB for the purpose of all analyses.

- Step 4: Extract Data for Architecture

  This step for extracting data for architecture reinterprets the sorted information in the database in compliance with pre-defined modules. It also extracts modules from the sorted components. This paper defines classes as the module unit and writes query statements. This generates the information about the relations between classes, between classes and methods, and between classes and variables. This also quantifies the quality indicators.

- Step 5: Visualization

  This visualization step runs the queries written in Step 4, reinterprets their results, and generates a DOT script and its graph to run Graphviz's DOT.

- Step 6: SW Quality Improvement

  Developing high-quality software requires a weaker inter-module coupling and a stronger inter-module cohesion [4]. The present paper defines the coupling as a quality indicator and performs the visualization accordingly.

3.2 Modulity

- Module Definition

  The module definition step defines a module unit suitable for the target software code to be visualized. The present paper defines classes as modules.

- Quality Indicator Definition

  In designing software, inter-module coupling needs minimizing whereas inter-module cohesion needs increasing in order to develop high-quality software. Thus, quantitative measurement indicators for coupling and cohesion need be set [9]. Figure 3 illustrates the coupling status represented as good to bad conditions. Here, coupling refers to inter-dependence or inter-relation between two modules. High inter-module coupling means strong inter-dependence between modules. This has adverse effects on transformation, maintenance and reuse of modules. Independent modules require low inter-module coupling and dependence. Coupling is sub-divided into data, stamp, control, external, common, and content couplings. Inter-module dependence increases in the direction of the content coupling while decreasing in the direction of the data coupling.

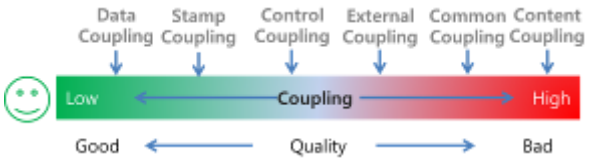

Figure 3. Coupling Spectrum

a) Data Coupling

In data coupling, the interface between modules consists solely of data. As a module calls another, it hands over the data as parameters or arguments. At the same time the called module returns the results of data processed. Here, the data coupling is most desirable in the modules that never affect each other. Therefore there is no need to know about the content.

b) Stamp Coupling

In stamp coupling, data structures such as arrangements or records are delivered as inter-module interfaces. Both modules view a data structure. Any changes in the data structure of a module affect the other module even when it is not referred to.

c) Control Coupling

Control coupling provides control components (e.g., Function Code, Switch, Tag, and Flag) used to control logical flows. Control coupling occurs when a higher module controls a lower module. This includes knowing the procedural details of its processing or when the functions for processing are designed separately between the two modules.

d) External Coupling

In external coupling, data (variables) that a module externally declares are referred to by another module.

e) Common Coupling

In common coupling, multiple modules share a common data domain. Any changes in the content of the common

data domain affect all modules sharing the domain, which weakens the independence.

f) Content Coupling

In content coupling, a module directly refers to or modifies the internal functions or data of another module.

- Code Pattern Analysis

  A code pattern is determined in the code pattern definition. As the present paper defines classes as module units, the coupling is decided in line with classes and inner class relationships.

- Refactoring

  Source codes are directly refactored to lower the high levels of coupling between modules.

IV. Case Study

The case employed here is a system developed in Struts Framework-based JAVA. The proposed tool chain is used to analyze the source code.

- Source Code Analysis: In the source code analysis step, the source codes in SN are located for automatic code analysis. Based on the content analyzed, SNDB files are generated. The SNDB files have such extensions as 1, by, cl, mi, f, mv, lv, and iv.
- SNDB Files Analysis: The SNDB files contain different types of binary data extracted from the SN. To analyze the content of binary files, dbdump.exe from the SN is run internally and transformed to text formats.
- Create DB from SNDB Processor: The data from the analysis of SNDB files are analyzed further using the parser developed here, and stored in the DB Tables in the DB.

SNDB4BY	table	
REFERRED_CLASS	field	TEXT
REFERRED_SYMBOL_NAME	field	TEXT
REFERRED_TYPE	field	TEXT
CLASS	field	TEXT
SYMBOL_NAME	field	TEXT
SYMBOL_TYPE	field	TEXT
ACCESS	field	TEXT
LINE_NO	field	TEXT
FILENAME_WITH_PATH	field	TEXT
CALLER_ARGUMENT_TYPES	field	TEXT
REFERRED_ARGUMENT_TYPES	field	TEXT
PROJECT_NAME	field	TEXT
SNDB4CL	table	
SNDB4IN	table	
SNDB4IU	table	
SNDB4IV	table	
SNDB4LV	table	
SNDB4MD	table	
SNDB4MI	table	
SNDB4TO	table	

Figure 4. DB Tales and Contents of SNDB files

- Extract data for architecture: We extract data from database in Figure 4 by using SQL.

Figure 5 shows the total number of coupling among modules. For drawing this graph, we count callings among modules. So we know intuitively how loosely coupled among modules. “BaseAction” module in Figure 5 is called 28 times from other modules.

Figure 6 shows the coupling weight value between two modules. The coupling weight value of the “ValuationMonitoringAction” module and the “DmethodCall” module in Figure 6. is higher than others. If the weight value is more than 100, it is illustrated by a red line on the graph. So we see intuitively how loosely coupled or tightly coupled two modules are. If tightly coupled, we need to do a refactoring source to reduce coupling weight values.

Figure 7 shows a class diagram. When you want to see a structure of code without any SW documents, this class diagram will help you. A module is a class, so a higher level is a package. We describe a package with a blue outlined (?) box, and it includes classes, also we describe a relation among classes. “com.hmpms.board.dao” package includes “BoardDAOImpl” and “BoardDAO” classes. “BoardDAO” class has methods such as selectList, selectTree.

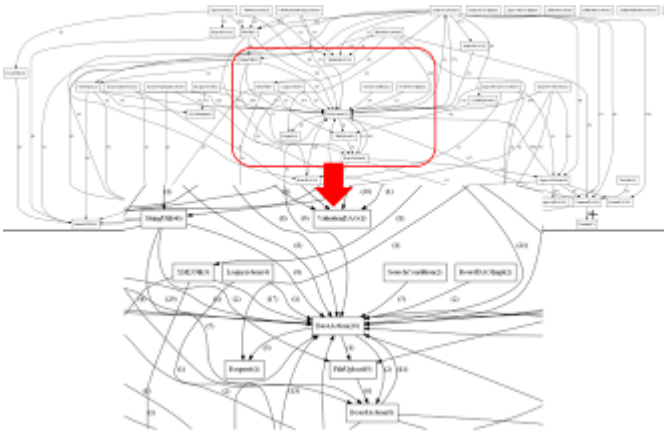

Figure 5. The total number of coupling of among modules

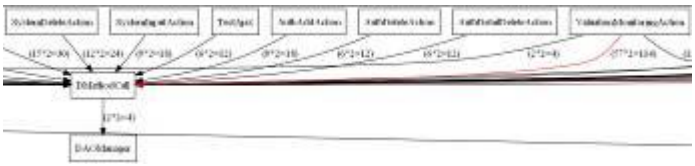

Figure 6. Coupling weight value of between two modules

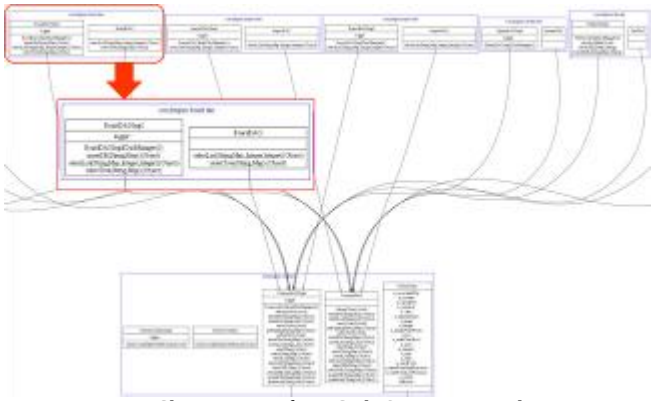

Figure 7. Class Diagram from Code Structure Visualization

V. CONCLUSIONS

The domestic software development industry focuses on development/testing and maturity measurement to deliver high-quality software. However, this is of little service to venture start-ups, SMEs, and established entities engaging in software development. For the purpose of developing high-quality software, the present paper focuses on detecting and altering problems of existing codes and rectifying software developers' bad smells. Developers need to other tasks if they focus on the development process, testing process, and maturity measurement of high quality software. Especially above methods cannot be the alternative to make high quality software about legacy systems. To address this issue, the proposed tool-chain method defines modules, quantifies the complexity of codes based on software structures and the frequency of inter-module relations, and digitizes the quality of codes with quality indicators for inter-module coupling as part of software visualization. The proposed method enables even developers of bad smells to lessen the code complexity with refactoring.

Future research will deal with the visualization of software quality in terms of cohesion, extract design documents using reverse engineering in addition to class diagrams representing the inheritance while continuing to carry out software quality measurement and refactoring to shed light on any patterns that could help improve software quality, and finally to visualize diverse cases of software using the proposed tool-chain method. Last we will develop our parser for making up for SN's faults to show various useful graphs.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of KOREA (NRF) and Center for Women In Science, Engineering and Technology (WISET).

REFERENCES

[1] NIPA SW Engineering Center, 2013, "SW Development Quality Management Manual (SW Visualization)", pp. 3-4.
[2] Steve McConnell, 2001, "Code Complete (2nd ed.)", Microsoft Press.
[3] Geon-Hee Kang, R. Young Chul Kim, Geun Sang Yi, Young Soo Kim, Yong B. Park, Hyun Seung Son, 2015, "A practical Study on Code Static Analysis through Open Source based Tool Chains," KIISE Transactions on Computing Practices, vol. 21, no. 2, pp. 148-153.
[4] Roger S, Pressman, 2010, "Software Engineering: a practitioner's Approach (7th ed.)", McGrawHill.
[5] Ivan Sommerville, 2001, "Software Engineering (6th ed.)", Pearson Education.
[6] http://sourceforge.net/projects/sourcenav/
[7] https://www.sqlite.org/
[8] http://www.graphviz.org/
[9] 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", KCC2014, pp.463-465.